UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Aplicación móvil de rastreo

DIRECTOR: Miguel Riesco Albizu

AUTOR: Alejandro Flórez Muñiz

Yo, Alejandro Flórez Muñiz, declaro que el presente proyecto y su correspondiente documentación, basada en la plantilla oficial de trabajos de fin de grado de la Escuela de Ingeniería Informática, ha sido realizado de manera íntegra por mí, constituyendo así una obra original. Además, declaro que las fuentes de información utilizadas han sido citadas a lo largo del documento y residen en la bibliografía del mismo.

Agradecimientos

Este proyecto quiero dedicárselo en especial a mis padres Julián y Ariane por todo el apoyo que me han dado no solo durante este trabajo, sino durante estos cuatro años de grado, incluyendo también a mi hermana Ariadne por sus recomendaciones y sugerencias como enfermera.

También me gustaría agradecer todo el esfuerzo y dedicación de muchos de los profesores que durante la carrera no solo han puesto su interés en explicar lo mejor posible las competencias de las asignaturas, sino que también lo han hecho con una gran pasión y dedicación haciendo que nosotros, los alumnos, estemos orgullosos de estudiar este grado. Entre ellos, quería agradecer en especial a Albizu por todo el apoyo y por sus consejos durante el desarrollo de este trabajo, pero también por su papel como profesor en asignaturas como Sistemas Operativos, la cual he de decir que me gustó muchísimo.

Por último, no podía faltar todo el apoyo que me han dado mis amigos y en especial Nacho y Barri, por aguantarme todos estos años, ¡muchas gracias! Gracias también a Col, Sergio y Raquel, por todo el apoyo y sus sugerencias de mejora para la aplicación.

Resumen

El presente proyecto persigue llevar a cabo la construcción de un sistema *software* distribuido para el rastreo de contactos estrechos con ciudadanos positivos en *COVID-19,* todo ello mediante una **aplicación móvil** desarrollada para dispositivos *Android* que pueda ser instalada por el mayor número de ciudadanos de la población. Así, se pretende hacer uso de la **geolocalización** por medio del *GPS* y de las redes *WiFi* para realizar el rastreo de la posición de los ciudadanos, automatizando las tareas de rastreo mediante el registro de sus coordenadas de manera periódica y su conservación en el almacenamiento local del dispositivo. Estas coordenadas se almacenan de manera **anónima** en el dispositivo local, y solamente cuando los usuarios lo decidan, se subirán a la nube representando un **positivo** en el sistema, con la opción de adjuntar sus **datos personales** para facilitar las operaciones de rastreo de las autoridades sanitarias. De este modo, los ciudadanos podrán ejecutar comprobaciones en su dispositivo local para detectar si han estado cerca de un positivo en *COVID-19*, descargando así las tareas de rastreo manuales desempeñadas por el personal sanitario.

El sistema debe tener presente en todo momento la **privacidad** de los usuarios, cuyos itinerarios son privados y residen en sus dispositivos de manera anónima. Cuando algún ciudadano da positivo, tiene la oportunidad de notificarlo a través de su aplicación móvil de modo que sus itinerarios se compartirán públicamente con el resto de las aplicaciones móviles, las cuales tendrán acceso a sus coordenadas para realizar las comparaciones en local para detectar contactos de riesgo. Para ello, se propone también el desarrollo y despliegue de un **servidor *web*** que exponga una ***API REST*** que pueda ser consumida por los clientes *Android* para compartir los itinerarios y descargar las coordenadas de otros positivos. La comprobación de contactos de riesgo se hará en local, en los propios dispositivos de los usuarios de manera que así se preserve al máximo posible su privacidad. Todos los datos del sistema se almacenarán en una base de datos central hospedada en la nube, a la cual se accederá desde el servidor *web*.

De manera adicional, se desarrollará un **cliente *web*** que sirva de panel de control para las autoridades y el personal sanitario, incluyendo los rastreadores. A este panel de control se accederá por medio de un navegador *web* y proporcionará la capacidad de configurar diferentes parámetros del sistema, como el comportamiento de la comprobación de contactos, la visualización de estadísticas de uso de la aplicación y la consulta de los positivos notificados en el sistema. Además, este cliente *web* también consumirá la *API REST* expuesta por el servidor.

Este proyecto constituye un **prototipo** experimental para probar las características de la geolocalización en *Android* como una herramienta para rastrear contactos de riesgo con positivos, por lo que no existe un cliente como tal y el tutor del proyecto será quien actúe como dueño del producto. Así, el desarrollo del proyecto se guiará a través de una **metodología ágil**, concretamente, ***Scrum***, haciendo uso del *framework* de ***Android*** para desarrollar la aplicación móvil con el lenguaje ***Kotlin*****,** el entorno de ejecución ***Node.js*** junto con el *framework* de ***Express.js*** para el desarrollo del servidor *web*, y el *framework* de ***Vue.js*** para la aplicación *web*.

Palabras Clave

COVID-19, *Android, Kotlin,* Geolocalización, Rastreo de contactos, Dispositivo Móvil, *Scrum*, *Node.js*.

Abstract

This project seeks to carry out the construction of a distributed software system to track close contacts between citizens and COVID-19 positives, all through a mobile application developed for Androd devices which can be installed by the majority of the population. Thus, the intention is to use **geolocation** through GPS and WiFi networks to perform citizen's position tracking, automating tracking tasks with the recording of their coordinates periodically and saving them into the local storage of the device. This coordinates are kept **anonimously** in the local device and only when the user decides it, they will be uploaded to the cloud representing a **positive** inside the system, with the option to attach their **personal data** in order to facilitate the tracking operations performed by health authorities. In this way, citizens would be able to execute checks in their local device to detect if they have been close to a COVID-19's positive, unloading manual tracking tasks performed by health personnel.

The system must bear in mind all the time user's **privacy,** whose itineraries are private and reside in their devices anonimously. When a citizen tests positive, they have the opportunity to notify it through their mobile application so all their itineraries will be shared publicly with the rest of applications, which will have access to their coordinates in order to perform local comparations to detect risk contacts. To do this, the development and deploy of a **web server** which exposes a **REST API** to be consumed by Android clients is also proposed in order to share itineraries and download coordinates from other positives. Risk contacts checks will be performed locally within users' devices so their privacy is preserved as much as posible. System data will be stored within a central database hosted in the cloud, which can only be accesed through the web server.

In addition, a **web client** will be developed to serve as a control panel for the heath authorities and personnel, including trackers. This control panel will be accessed through a web browser and it will provide the capacity to configure different system parameters, such as risk contacts check behavior, app use statistics visualization and notified positives consult. This web client will also consume the REST API exposed by the web server.

This project constitues an experimental **prototype** destined to test geolocation features in Android like a tool for positives risk contacts tracking, so there is no final client as such and the project tutor will act as the producto owner. Thus, project development will be guided through an **agile methodology**, specifically, **Scrum**, making use of the Android framework to develop the mobile application with the **Kotlin** language, **Node.js** execution enviroment with the **Express.js** framework to develop the web server, and finally, **Vue.js** framework for the web application.

Keywords

COVID-19, *Android, Kotlin,* Geolocation, Contact Tracking, Mobile Device, *Scrum*, *Node.js*.

Índice General

Índice de Figuras

Índice de Ilustraciones

Índice de Tablas

Capítulo 1. Memoria del Proyecto

Este primer capítulo tiene como objetivo servir de iniciación al resto de capítulos del proyecto desarrollado, de forma que, el lector pueda involucrarse en el contexto del sistema rápidamente, afianzando los conceptos cruciales del mismo. De este modo, se presenta un resumen global de alto nivel del proyecto, sus distintas fases y el entorno que lo rodea, todo ello como una primera aproximación al sistema para que en las secciones futuras se parta ya de una base establecida, aunque no con demasiado contenido técnico.

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

Ante la situación de emergencia sanitaria que se ha desatado en el último año, como consecuencia del virus *COVID-19*, se han desplegado diversos protocolos y sistemas de control de transmisión del virus. Más allá de las medidas típicas de distanciamiento social, uso de mascarilla u otras, han surgido unas figuras sanitarias cruciales para controlar el contagio del virus entre los ciudadanos. Se trata de los **rastreadores**, personas con o sin un título sanitario, designadas por el Estado o los ayuntamientos de las localidades con el objetivo de rastrear los **contactos estrechos** de aquellos ciudadanos que han dado **positivo** en *COVID-19*. De este modo, los rastreadores contactan de manera **manual** con los positivos, por ejemplo, a través de una llamada telefónica, para realizar una serie de preguntas específicas sobre con quién han estado en contacto cercano en los últimos días, durante cuánto tiempo, a qué distancia, si el contacto tuvo lugar en un sitio cerrado, si los individuos involucrados estaban con mascarilla…*etc*.

Esta batería de preguntas sirve a los rastreadores para recabar información sobre el **árbol de contagio**, es decir, las **relaciones** cercanas que ha entablado el positivo en los días anteriores. Una vez obtenida esta información, contacta de manera manual con los posibles contagiados para **notificarles** que son un **contacto estrecho** de un positivo y que deben guardar cuarentena hasta realizarse una prueba de *COVID-19*. Este proceso se repite para todos los positivos que se diagnostiquen, con el objetivo de lograr detener la transmisión del virus lo más pronto posible.

Como se puede ver, se trata de una tarea muy **tediosa** y **lenta**, que puede incluso resultar poco efectiva si el positivo no tiene intenciones de ayudar o ser sincero. Mucha gente puede incluso no coger el teléfono, o mentir respecto a los contactos que ha tenido en los últimos días. Es entonces cuando entra en juego la **tecnología**, concretamente la tecnología **móvil**. Hoy en día, la mayoría de la población dispone de un dispositivo móvil inteligente que tiene capacidades muy numerosas, entre ellas, la capacidad de **geolocalizar** al dispositivo móvil mediante la triangularización por *GPS* o *WiFi*.

Con todo esto, los procesos de rastreo descritos anteriormente podrían **automatizarse**, o más bien, **informatizarse**, de manera que las tareas pasaran a ser gestionadas por un sistema informático, más rápido, preciso, eficiente y cómodo. Esta es precisamente la **motivación** de

este proyecto, trasladar estas tareas de controlar los contactos de riesgo con positivos a los dispositivos móviles de los ciudadanos, de manera que se transforme en una **actividad social**, donde las personas que hayan dado positivo en *COVID-19* tengan la voluntad de notificarlo en su dispositivo móvil para que el resto de ciudadanos que tengan la aplicación instalada puedan **comparar** sus **itinerarios** o **rutas** con las del positivo y así ser notificados si se detecta algún **contacto cercano.** De este modo, se descargaría la responsabilidad de los rastreadores, quienes no tendrían que realizar el rastreo a mano, y se distribuiría entre la población con ayuda de los dispositivos inteligentes, los cuales irían registrando periódicamente las coordenadas de sus usuarios.

De esta forma, los **objetivos** principales de este proyecto se resumen, por un lado, en **facilitar** e **informatizar** las **tareas** de rastreo asignadas a los **rastreadores**, para descargarlos de dicha responsabilidad y así agilizar el proceso y hacerlo más efectivo, y por otro, detener la **propagación** del virus en la medida de lo posible, al notificar con antelación a los posibles contactos cercanos de los usuarios, todo ello de manera automática y sin necesidad de entablar una conversación telefónica con un rastreador. Estos son solo los dos objetivos principales, más adelante se describirán con más detalle los objetivos concretos perseguidos por este proyecto.

El **alcance** de este proyecto está limitado por tratarse de un **prototipo funcional**, es decir, no pretende ser un proyecto real que se vaya a desplegar en un entorno de producción para ser utilizado por la población, sino que, más bien persigue ser una **base sólida** para dar paso a futuros sistemas que sigan este mismo esquema de rastrear contagios mediante geolocalización. Por ello, los **productos** resultantes del desarrollo de este proyecto serán:

- Una **aplicación móvil** de rastreo, que será utilizada por los ciudadanos para ir registrando sus localizaciones cuando lo deseen, notificar un positivo cuando sean diagnosticados y realizar comprobaciones para detectar si han estado cerca de algún positivo.
- Una **aplicación *web*** que servirá a modo de panel de control para los usuarios administradores, en este caso, cualquier individuo del personal sanitario o que sea un rastreador certificado. A través de este panel, los administradores podrán visualizar la evolución del virus viendo estadísticas de positivos notificados, instalaciones…*etc*, así como configurar ciertos aspectos del rastreo para la aplicación móvil utilizada por los ciudadanos.
- Un **servidor *web*** para centralizar todos los datos del sistema en un mismo sitio, y así poder albergar los datos de los itinerarios de los positivos notificados, así como otros aspectos como los parámetros de la configuración.

El sistema será desplegado en un entorno ficticio a modo de prueba, para demostrar su funcionamiento en un contexto controlado. Además, la aplicación móvil irá destinada a los dispositivos móviles ***Android***, es decir, que tengan instalado este sistema operativo, por lo que no será compatible con otros sistemas operativos como *iOS* o *Windows Phone.*

1.2 Resumen de Todos los Aspectos

Una vez comentados los objetivos del sistema y su motivación de manera resumida, en esta sección se describen brevemente los **capítulos** que constituyen este documento, algunos de los cuales constituyen fases del propio desarrollo del proyecto.

- **Capítulo 2. Introducción**. En este capítulo se desarrollan los aspectos comentados en el capítulo 1 de la memoria, incluyendo además los antecedentes y la situación actual del entorno del sistema.
- **Capítulo 3**. **Aspectos teóricos.** Aquí se describen los aspectos teóricos relevantes para comprender mejor el proyecto, incluyendo las herramientas y plataformas utilizadas para el desarrollo del mismo.
- **Capítulo 4. Estudio legal.** Se trata de un estudio para analizar el tratamiento de datos personales de los usuarios que se debe contemplar en el sistema. Se analizan las necesidades de tratamiento de datos junto con la normativa vigente de protección de datos.
- **Capítulo 5. Planificación del Proyecto y Resumen de Presupuestos.** Este capítulo comprende todo el proceso de planificación de las tareas que constituyen el proyecto, agrupadas por fases y junto con sus entregables. Además, también se presenta un resumen del presupuesto del cliente, que muestra el coste total estimado de desarrollar e implantar el sistema.
- **Capítulo 6. Control y seguimiento del Proyecto.** Dada la naturaleza ágil del desarrollo, se presenta en este apartado la monitorización de la evolución del proyecto, incluyendo un breve resumen del desempeño de cada iteración y los resultados obtenidos.
- **Capítulo 7. Análisis.** Este capítulo presenta toda la fase de análisis del sistema, adaptada al enfoque ágil del proyecto. Se muestra lo que debe hacer el sistema y lo que no debe hacer a través de las historias de usuario, diagramas de casos de uso, diagramas de clases preliminares y bocetos de la interfaz de usuario. También se incluye una descripción del Plan de Pruebas para el sistema.
- **Capítulo 8. Diseño.** Contiene el diseño y la arquitectura de los módulos que componen el sistema, incluyendo diagramas de clases, de interacción, de despliegue y otros diagramas que reflejan cómo se implementan los requisitos especificados en el análisis. También se incluye la especificación técnica del Plan de Pruebas para el sistema, donde se presentan los propios casos de prueba diseñados para cada característica bajo pruebas.
- **Capítulo 9. Implementación del Sistema.** Aquí se describen algunos aspectos técnicos relativos a la implementación de los módulos del sistema, incluyendo herramientas y lenguajes utilizados, problemas y obstáculos encontrados, así como explicaciones detalladas sobre el funcionamiento de algunos aspectos significativos del sistema.
- **Capítulo 10. Desarrollo de las pruebas.** Este capítulo expone los resultados obtenidos tras implementar y ejecutar los casos de prueba diseñados en la especificación técnica del Plan de Pruebas. Se incluyen también explicaciones sobre algunos aspectos de

interés relativos al código de la implementación de las pruebas, junto con problemas encontrados y su solución.

- **Capítulo 11. Manuales del sistema.** Los manuales del sistema constituyen una guía paso a paso de cómo utilizar el sistema, tanto para los ciudadanos como para los usuarios administradores, cuyo papel es desempeñado por el personal sanitario o los rastreadores. Incluye capturas de pantalla del sistema para mostrar las instrucciones de uso.
- **Capítulo 12. Conclusiones y Ampliaciones.** Este capítulo expone las conclusiones obtenidas como consecuencia del desarrollo del proyecto, reflexionando sobre el producto obtenido y la experiencia del desarrollo. Además, también se incluyen las ampliaciones propuestas de cara al futuro para mejorar el prototipo.
- **Capítulo 13. Presupuesto.** Contiene el desarrollo completo y detallado de los presupuestos de costes y de cliente, incluyendo el modelo de empresa, los recursos utilizados para el proyecto y las partidas en las que este se desglosa.
- **Capítulo 14. Referencias Bibliográficas.** Contiene las referencias bibliográficas de los artículos y páginas *web* consultadas para recabar la información requerida para el desarrollo del proyecto.
- **Capítulo 15. Apéndices.** Este último capítulo, contiene el glosario del documento y el diccionario de datos junto con el índice alfabético, además de una explicación de la estructura del *ZIP* entregado con el proyecto. También incluye los anexos del documento donde se presentan las figuras, tablas y otros aspectos más detallados y técnicos de los apartados del documento a modo de complemento.

Capítulo 2. Introducción

En el anterior capítulo, se presentó un pequeño resumen del sistema a modo de iniciación al proyecto, incluyendo su motivación, algunos de sus objetivos y el alcance de manera resumida. En este capítulo, se desarrollan con mayor detalle estos aspectos del proyecto para conformar una base más sólida del sistema a construir, a modo de introducción.

2.1 Justificación del Proyecto

En el año 2020 comenzó una de las pandemias más significativas de las últimas décadas causada por el virus *COVID-19* que tuvo su origen en *Wuhan*, *China*. El virus se propagó rápidamente a lo largo del planeta causando un gran número de muertes en poco tiempo. Como consecuencia, una gran crisis sanitaria tuvo lugar en los distintos países, generando un declive económico y un profundo descontento social debido a la necesidad de mantener una **cuarentena** colectiva, donde todo el mundo debía permanecer en sus casas hasta que disminuyera el número de casos. En España se declaró el **estado de alarma** el 14 de marzo de 2020 y se extendió hasta mediados del mes de junio, tras varias prórrogas. Además, se impusieron numerosas medidas y restricciones en todos los ámbitos para evitar al máximo posible el contacto estrecho entre personas y así recortar la propagación del virus.

Una de las figuras clave durante la pandemia fueron los **rastreadores**, personas que no necesariamente tienen una formación sanitaria y que son las encargadas de **rastrear** la propagación del virus entre un caso positivo y todos sus contactos estrechos en los últimos días. Estas personas contactan con los casos positivos en *COVID-19* a través del número de teléfono y realizan numerosas preguntas sobre qué síntomas tienen, cuando se manifestaron por primera vez, con qué personas ha estado en contacto los últimos días…*etc*.

Esta técnica de rastreo se realiza de forma **manual**, es decir, es necesario contactar manualmente con cada una de las personas que han dado positivo y hacerles preguntas de forma oral, lo cual como es de esperar es un proceso tedioso y poco eficiente, haciendo que sea necesario un número considerable de rastreadores en cada comunidad autónoma. Como solución a este problema, se optó por hacer uso de las **nuevas tecnologías**, concretamente la tecnología de los **dispositivos móviles**. Hoy en día, es difícil conocer a alguien que no disponga de un teléfono móvil inteligente con conexión a *internet*, por lo que se comenzaron a desarrollar prototipos de **aplicaciones móviles de rastreo** las cuales estaban destinadas a ser utilizadas por toda la población o al menos en su mayoría. Los diferentes gobiernos comenzaron a adoptar estas aplicaciones como una **herramienta oficial** que recomendaban instalar a sus habitantes en sus dispositivos móviles, siendo incluso obligatoria en ciertos países.

Para llevar a cabo el rastreo de contactos estrechos de *COVID-19* de manera **automática** se propone el desarrollo de este proyecto como una **alternativa** a la aplicación ya existente de *Radar Covid*. El proyecto consistirá en el desarrollo de una **aplicación móvil** de **rastreo** que haga uso de la **geolocalización** incorporada en los dispositivos mediante el *GPS* o *WiFi,* destinada a ser utilizada por toda la población española. La aplicación irá registrando las

posiciones de los usuarios almacenando sus coordenadas en **local**, es decir, en la memoria del dispositivo. De este modo, cuando un usuario de positivo en *COVID-19* tendrá la posibilidad de **notificar un positivo** en la aplicación móvil, enviando sus localizaciones registradas en los últimos días al servidor central hospedado en la nube. En ese momento, todos los usuarios de la aplicación que hayan registrado sus localizaciones en los últimos días podrán realizar una **comprobación** con los positivos almacenados en la nube comparando sus itinerarios con las rutas de los positivos de forma que se detecten **contactos de riesgo** con alguno de estos positivos, es decir, localizaciones en las que se haya estado cerca de un positivo durante un periodo de tiempo determinado.

Así, aquellos usuarios que hubieran estado en contacto con un positivo serían **notificados**, recibiendo además información sobre la **fecha** y **hora** del contacto, el **nivel de riesgo**, **tiempo de exposición** y las **localizaciones exactas** donde tuvo lugar el contacto estrecho pudiendo visualizarlas en un mapa. De forma adicional, se pretende desarrollar una **aplicación *web*** que sirva de panel de control y de visualización de estadísticas para los administradores, el personal sanitario o cualquier persona que actúe de rastreador y que disponga de los permisos necesarios para acceder a la aplicación *web*. Mediante este panel de control, los administradores podrían configurar ciertos parámetros del algoritmo de comprobación, limitar la notificación de positivos, modificar el periodo de infectividad que se tiene en cuenta a la hora de subir las localizaciones a la nube…*etc*, así como visualizar estadísticas de uso de la aplicación y ver los positivos que se van notificando junto con sus itinerarios en un mapa.

Con ello, se pretende **mejorar** y **agilizar** las tareas de rastreo tradicionales sustituyéndolas por un proceso **automático** en el cual intervienen todos los habitantes de la población registrando sus itinerarios y notificando que han dado positivo en la aplicación. De esta forma, se descargaría la responsabilidad de los rastreadores quienes pasarían a trabajar con el panel de control *web*, ya que el rastreo se estaría haciendo automáticamente con los flujos de notificación de positivos y la comprobación de contactos de riesgo, lo cual es responsabilidad de cada usuario.

El proyecto también tiene una motivación **experimental** para comprobar el funcionamiento del rastreo utilizando el *GPS* de los dispositivos móviles como una alternativa a *Radar Covid*, la cual hace uso del *Bluetooth* para llevar a cabo el rastreo y comprobar los contactos con positivos. Además, el proyecto de *Contact Tracker* pretende dar la máxima capacidad de **personalización** a los usuarios, dándoles la oportunidad de configurar el rastreo de coordenadas para que el registro de itinerarios sea más exhaustivo a costa de consumir más batería o viceversa, u otras configuraciones de la comprobación de contactos.

En resumen, el proyecto cubrirá las necesidades de los usuarios para comprobar si han estado en contacto con un positivo, y en ese caso, dar el máximo posible de información acerca de la **gravedad** del contacto para que se dispongan a realizar una *PCR*, sin falta de ser contactados por un rastreador, o peor aún, no ser conscientes de que han sido contagiados de *COVID-19.* También aportará información al personal sanitario y a los rastreadores que podrán ver estadísticas de cuántos positivos son notificados, media de los resultados de las comprobaciones y los positivos que se van notificando a lo largo del tiempo.

2.2 Objetivos del Proyecto

Una vez vista la motivación y justificación del proyecto, en esta sección se describen a grandes rasgos los principales **objetivos** perseguidos por el proyecto. A continuación, se muestra una lista con estos objetivos:

1. Diseñar y desarrollar una aplicación móvil en *Android* para el rastreo de las ubicaciones, rutas o itinerarios de los usuarios, almacenando sus coordenadas en la memoria local del dispositivo.
2. Permitir la notificación de positivos en *COVID-19* de manera automática y remota desde los dispositivos móviles, enviando las ubicaciones y sus coordenadas registradas en los últimos días al servidor central en la nube.
3. Dar a los usuarios la capacidad de realizar comprobaciones de manera manual o automática (periódicamente) comparando sus localizaciones registradas con las de otros positivos almacenados en la nube para ver si han estado en contacto con alguno de ellos.
4. Dar a los usuarios la libertad para configurar el rastreo de coordenadas y la comprobación de los contactos de riesgo.
5. Descentralizar el almacenamiento de las coordenadas de los usuarios persistiéndolas en local en cada dispositivo, subiéndolas a un servidor central solo si el usuario lo desea, asegurando un mínimo de privacidad.
6. Diseñar y desarrollar un panel de control *web* (aplicación *web*) para los administradores, personal sanitario y rastreadores que permita visualizar estadísticas de la aplicación, positivos notificados y sus itinerarios, y configurar los parámetros de la aplicación relacionados con la notificación de positivos y la comprobación de contactos de riesgo.
7. Automatizar, facilitar y agilizar las tareas de rastreo tradicionales que se realizan manualmente a través de los rastreadores.
8. Ahorrar tiempo a los rastreadores eliminando la necesidad de contactar telefónicamente con los positivos y realizarles preguntas sobre las personas con las que se mantuvo contacto en los últimos días.
9. Prevenir los contagios y la propagación del virus *COVID-19* entre la población a través de la instalación y uso de la aplicación móvil por parte de la gran mayoría de la población española.
10. Aportar información detallada a los usuarios sobre los contactos estrechos con positivos, indicando lugar, fecha y hora, tiempo de exposición, proximidad media y gravedad o riesgo del contacto.
11. Convivir con el actual sistema *Radar Covid* adoptado por el Gobierno de España para el rastreo automatizado de *COVID-19*.
12. Garantizar al máximo posible la privacidad de los usuarios en lo que respecta a datos personales sanitarios, dándoles total libertad para subir sus itinerarios y coordenadas a la nube de forma anónima o aportando sus datos personales si lo desean.
13. Extender el uso de un sistema de rastreo y control de pandemias para enfermedades infecciosas entre la totalidad de los habitantes que sirva de base no solo para el *SARS-Cov-2,* sino para cualquier pandemia o enfermedad que surja en un futuro.

14. Experimentar con las tecnologías de geolocalización y *GPS* para evaluar su aplicación en el ámbito sanitario para el rastreo de contactos de riesgo frente al uso del *Bluetooth* utilizado en el sistema actual.

2.3 Antecedentes

Desde el origen del brote en *Wuhan* hasta la propagación mundial del virus *SARS-Cov-2* en torno a comienzos del 2020, se pusieron en marcha numerosos mecanismos y estrategias para tratar de reducir el número de contagios y la transmisión de la enfermedad a través de las personas. Estos mecanismos engloban desde **medidas de carácter individual**, tal como el uso de mascarilla, guantes, gel hidroalcohólico hasta **medidas de carácter comunitario,** como el establecimiento del estado de alarma o el confinamiento social.

Desde un primer momento, se llevó a cabo el **aislamiento** de los infectados por COVID-19, con el objetivo de separar a las personas aparentemente sanas de quienes no lo estaban. De manera complementaria al aislamiento, también se impuso una **cuarentena** a todas aquellas personas que hubieran entrado en contacto con algún positivo en COVID-19, y por tanto existiese alguna probabilidad de haber contraído el virus. Sin embargo, estas medidas no fueron suficientes para frenar la escalada de los contagios, por lo que finalmente se recurrió al **confinamiento domiciliario obligatorio**, que fue aplicado a distintos niveles: local, comunitario, o incluso a nivel de Estado. Este confinamiento supuso una restricción a todos los ciudadanos para que permanecieran en sus hogares y así reducir al mínimo las interacciones sociales que pudieran desembocar en nuevos contagios. Más adelante también se pusieron en práctica los **cierres perimetrales,** que suponían unas restricciones más relajadas respecto a las de un confinamiento, pues se permitía la interacción social, aunque esta se limitaba al interior de una zona o "perímetro".

Además de todas estas medidas, se puso en práctica una técnica que se conoce como el **rastreo de contactos**, cuyo objetivo principal es la detección previa y controlada de posibles casos positivos mediante el seguimiento de los contactos recientes de un positivo.

2.3.1 Rastreo de contactos

En las primeras etapas de la pandemia, se llevó a cabo el rastreo de contactos desde la Atención Primaria de cada comunidad (en el caso de España) como medida de prevención de los contagios, lo cual se realizaba de forma **manual**, sin utilizar ningún intermediario de *software* que facilitase el trabajo. De esta forma, se realizaba un seguimiento de las personas que habían dado positivo en COVID-19 a través de una serie de preguntas básicas acerca de los síntomas, lugares visitados en los últimos días, contacto cercano y prolongado con otras personas…

Mediante la respuesta a estas preguntas, se podía obtener relativamente rápido una **lista de contactos** que pudieran haber sido contagiados como consecuencia de haber estado en contacto directo con el positivo en COVID-19. Una vez obtenida la lista, se podía contactar con esas personas, que posiblemente estuvieran infectadas sin saberlo, para informales sobre la situación y alentarles para que se sometieran a las pruebas.

Este seguimiento también consistía en la **monitorización** de los positivos en COVID-19 para garantizar que se seguían al pie de la letra las indicaciones relativas al aislamiento y distanciamiento social.

Como se comentó, este seguimiento/rastreo se realizaba de manera **manual**, es decir, a través de **vía telefónica**, sin utilizar ningún tipo de herramienta de *software* más que el dispositivo móvil o teléfono fijo para ponerse en contacto con las personas. Por ello, también era necesario disponer de **profesionales de la Atención Primaria y de otros sectores** que se encargaran de llevar a cabo este seguimiento y de realizar las llamadas telefónicas correspondientes, es decir, el rastreo se realizaba "a mano", llamando uno a uno a los distintos contactos recientes del positivo, que pudieran haber contraído el virus.

A estos profesionales se les conoce como **rastreadores**, y no necesariamente son personal sanitario, sino que dicho cargo lo puede desempeñar cualquier persona con una formación sólida en algún campo, sin necesidad de tener conocimientos médicos específicos. Estos rastreadores suelen trabajar desde alguna oficina o Centro de Salud, haciendo llamadas telefónicas tan pronto como se **registra un nuevo positivo**. Es entonces cuando se contacta con el positivo para realizarle preguntas como: **¿con quién has estado estos dos últimos días?**, ¿cuánto tiempo?, ¿al aire libre?, ¿con mascarilla?...

A pesar de tratarse de un trabajo bastante tedioso y complejo, se sigue llevando a cabo hoy en día, aunque existen alternativas *software* informatizadas que serán tratadas más adelante en las siguientes secciones.

2.3.2 Notificación de positivos

Otro de los aspectos que constituyen un antecedente del sistema bajo construcción, es la notificación de positivos, que previamente se realizaba manualmente. El proceso de notificación de un positivo consistía en que los ciudadanos se sometían a una prueba de *COVID-19* y esperaban al resultado, de forma que, si este era positivo, algún rastreador disponible contactaba telefónicamente con ellos. De este modo, los usuarios tenían cierta responsabilidad para someterse a una prueba, aunque no realizaban la notificación explícitamente.

Esta manera de notificar un positivo sigue estando vigente en la situación actual, por lo tanto, también constituye un factor con el que debe convivir el sistema bajo construcción.

2.3.3 Comprobación de contactos

Antes de la introducción de los primeros sistemas de rastreo de contactos, la comprobación de contactos de riesgo, entendiendo como tal al proceso de detectar si un usuario ha entrado en contacto con algún positivo en *COVID-19,* no se realizaba explícitamente, es decir, no existía forma de conocer si un usuario había entrado en contacto con un positivo, más allá de realizarle preguntas sobre qué lugares había visitado o con quién se había relacionado en los últimos días, o bien por la propia intuición del usuario que tuviese sospechas de estar contagiado.

Así, el usuario con sospechas de haber sido contagiado ya sea por la manifestación de síntomas o porque estuviera con alguien que ha dado positivo, se sometía a una prueba para detectar la presencia de *COVID-19* en su organismo.

Como se puede ver, no existe un procedimiento como tal para detectar y analizar los contactos cercanos del usuario ni los lugares que este ha visitado con la intención de averiguar posibles contactos de riesgo con otros positivos.

2.4 Estudio de la Situación Actual

Tras analizar los antecedentes del sistema, en esta sección se describe la situación actual en la que se enmarca el desarrollo del mismo. Ya se ha visto que la práctica del rastreo manual resulta una actividad muy tediosa, que ocupa mucho tiempo y esfuerzo para los encargados de realizar las llamadas telefónicas, aunque lo cierto es, que se sigue utilizando actualmente para rastrear positivos.

Con el objetivo de automatizar este rastreo, diferentes países han recurrido al desarrollo del *software* para tratar de informatizar esta actividad. Concretamente, el desarrollo se ha volcado principalmente en el ámbito de la **movilidad,** dado que los dispositivos móviles son una herramienta muy versátil y que resultan ser los candidatos idóneos para estas actividades de rastreo, pues hoy en día es difícil encontrar a alguien que salga de casa sin su *smartphone.*

Conforme fue avanzando la pandemia, los gobiernos de distintos países fueron recurriendo a las empresas de *software* para desarrollar **aplicaciones de rastreo** para los dispositivos móviles, con la esperanza de que, poco a poco, fuesen desplazando y sustituyendo a los rastreadores humanos para facilitarles el trabajo. La **automatización del rastreo** ha ido evolucionando durante este último año hasta llegar casi al punto de que cada país dispone de su propia aplicación oficial.

El funcionamiento exitoso de estas aplicaciones de rastreo depende principalmente del número de usuarios que la activen en su dispositivo. De no haber suficientes usuarios, la tarea de rastreo apenas tendrá efecto y por tanto no aportará ninguna ventaja frente al rastreo manual. De hecho, es lo que está ocurriendo en la mayoría de los países, donde solo una minoría de los ciudadanos hace uso de las aplicaciones de rastreo.

2.4.1 Evaluación de sistemas similares

En esta sección, se lleva a cabo un análisis de otros sistemas similares ya lanzados al mercado, y que actualmente se encuentran en funcionamiento. Para cada sistema, se hace una descripción y se resume su funcionamiento, incluyendo ventajas y desventajas frente al sistema a construir.

2.4.1.1 *Radar Covid (España)*

Radar Covid es la alternativa oficial del Gobierno de España para el rastreo de contactos a través de una aplicación móvil. El funcionamiento es muy sencillo, se trata de un "**radar**" que hace uso del ***bluetooth*** para llevar un registro de todos los dispositivos con los que se cruza el usuario, a una distancia reducida. Estos dispositivos registrados, constituyen la lista de "sospechosos" que será necesario analizar cuando se notifica un nuevo positivo. Para realizar este registro, es necesario que las dos personas que se cruzan tengan instalada y activada la aplicación, pues de otra forma no tendrá ningún efecto y no se registrará el contacto de riesgo.

2.4.1.1.1 Funcionamiento

El núcleo funcional del sistema es muy sencillo, y realmente no realiza los cálculos por sí mismo, sino que deriva esta complejidad en la **API de *Google* y *Apple*** de ***contact tracing*** **[1]**. Esta *API* ha sido desarrollada como resultado de una alianza entre las dos grandes empresas como medida para facilitar servicios de rastreo de contactos de manera **nativa** en los dispositivos ***Android*** y ***iOS.*** Ha sido diseñada teniendo en cuenta la **privacidad** como principio fundamental, ya que garantiza que no se recopila ningún tipo de información personal ni relativa a la ubicación de los usuarios.

La *API* proporciona lo que se denominan ***Notificaciones de exposición***, las cuales se pueden desactivar desde los ajustes del dispositivo. Estas notificaciones aparecen cuando se ha estado en contacto cercano con otros dispositivos de usuarios que han dado positivo en COVID-19. Para ello, el sistema genera unos ***IDs*** **aleatorios** y **anónimos** que no contienen ningún tipo de información sensible del usuario, y además se renuevan cada 10-20 minutos, para garantizar que no se recopilan datos de ubicación.

A través del *bluetooth*, estos *token*s anónimos se **intercambian** con el resto de los dispositivos cercanos, dando lugar a una especie de *"handshake"* en donde dos dispositivos se envían sus *tokens* el uno al otro. El intercambio se produce con dispositivos cercanos a **menos de 2 metros** de distancia, y con los que se ha estado durante **más de 15 minutos.** Dadas las características de los contagios, solo se tienen en cuenta los *tokens* de cruces con usuarios de los últimos **14 días**. Desde los ajustes de las notificaciones de exposición, el usuario puede eliminar los *tokens* si lo desea, aunque estos se eliminan automáticamente pasados los 14 días. La aplicación funciona en segundo plano buscando *tokens* de otros dispositivos cada **5 minutos**, y emitiendo su *token* actual cada pocos segundos para que otros dispositivos lo reciban.

Llegados a este punto, cada dispositivo tiene almacenado en local un registro con todos los *tokens* anónimos de otros dispositivos con los que se ha cruzado. Cuando un usuario da positivo en COVID-19, se le da la posibilidad de notificarlo a través de la aplicación, lo cual quiere decir que las **autoridades sanitarias** no tienen control sobre los contagios, pues la elección de publicar el positivo reside completamente en el usuario. Tras recibir el positivo, las autoridades sanitarias proporcionan un **código de diagnóstico** que el usuario podrá introducir en la aplicación incluyendo la fecha de **inicio de síntomas** para permitir que se suban a la nube todos sus *tokens* aleatorios que se generaron en los últimos 14 días.

De este modo, cada dispositivo realiza una sincronización periódica cada día con los datos de *tokens* positivos obtenidos desde el servidor, comparándolos con la lista de *tokens* registrados

con los que se ha estado en contacto los últimos 14 días. Si hay algún ***match***, entonces se dispara una **notificación de exposición** que le aparecerá al usuario informándole de que es posible que haya estado en contacto con un positivo.

2.4.1.1.2 Ventajas

Una vez visto el funcionamiento de la aplicación, vamos a describir las principales ventajas que tiene *Radar Covid* frente al sistema a desarrollar.

2.4.1.1.2.1 Sencillez en los cálculos

Al utilizar la *API* de Notificaciones de Exposición, gran parte de los cálculos se simplifican ya que se derivan en dicha *API.* Además, la decisión de diseño de utilizar ***bluetooth*** como modo de rastreo, lo hace aún más sencillo, pues a diferencia de utilizar **geolocalización,** no es necesario ir comprobando la **proximidad espacial** ni la **proximidad temporal** entre coordenadas, sino que es suficiente con comparar dos listas de *tokens*. Por ello, apenas existe un coste computacional significativo y la comprobación se realiza relativamente rápido.

2.4.1.1.2.2 Mayor privacidad

Como el núcleo funcional está diseñado en torno a la **privacidad**, *Radar Covid* garantiza la inexistencia del tratamiento de información de índole personal, pues los *tokens* son **aleatorios**, **anónimos** y no contienen ningún dato personal codificado.

En cambio, el nuevo sistema hará uso de los **datos de ubicación** de los usuarios, que estarán guardados en local hasta que se envíen al servidor tras notificar el positivo. Aunque se les garantiza el mayor **anonimato** y **privacidad** posible (la única información personal que se almacena es la ubicación), el nivel de privacidad ya no es comparable al de *Radar Covid*, donde solamente se almacenan ***IDs*** insignificantes.

2.4.1.1.2.3 Interoperabilidad entre plataformas

Otra de las ventajas frente al sistema a desarrollar, es la **interoperabilidad** con dispositivos ***Android*** y ***iOS***, gracias al esfuerzo unido de *Google* y *Apple* para unificar el rastreo de contactos a través de su *API.* Esto permite que la aplicación sea funcional para la inmensa mayoría de dispositivos que tengan instalado *Android* o *iOS* como sistema operativo.

En contraposición, el futuro sistema solo estará disponible para los dispositivos *Android,* al menos para esta primera versión del **prototipo**, pues en futuro no habría problema en desarrollar también una versión para iOS, bien mediante desarrollo nativo o utilizando *frameworks* multiplataforma como *Flutter.*

2.4.1.1.3 Desventajas

A continuación, se listan los inconvenientes más destacables de *Radar Covid* y en qué sentido se mejoran con la construcción del nuevo sistema.

2.4.1.1.3.1 Consumo de batería

Una de las principales limitaciones de *Radar Covid*, es la necesidad de activar **simultáneamente** tanto el ***bluetooth*** como la **ubicación** del dispositivo para poder escanear a otros dispositivos usando *BLE (Bluetooth Low Energy),* al menos en la versión para *Android.* Esto no significa que la aplicación recopile datos de ubicación, sino que se trata de una restricción que impone *Google* a partir de la versión 6.0, obligando a activar los servicios de

ubicación para poder usar el escáner de *bluetooth*, aunque dichos servicios no se usen explícitamente. Esto da como resultado un **consumo considerable** de la **batería**, pudiendo incluso llegar a drenarla por completo.

Con el nuevo sistema, se pretende limitar la funcionalidad únicamente al uso de la **ubicación** y del ***GPS*** del dispositivo, sin necesidad de activar el *bluetooth*. Esto supone cierta **disminución** del consumo de batería, pues, aunque es cierto que el *GPS* puede consumir bastante, el impacto en la batería no será tan notable como con ambos servicios activados.

2.4.1.1.3.2 Dependencia de la API

Otro inconveniente es la dependencia de la *API* de Notificaciones de Exposición, que se puede configurar desde los ajustes del dispositivo. El grueso funcional se basa, casi en su totalidad, en los servicios que proporciona esta *API*, la cual **desaparecerá** cuando ya no sea necesaria, según indican las dos grandes compañías *Google* y *Apple.* Como consecuencia de esta dependencia, la aplicación quedará **obsoleta** e incluso **dejará de funcionar** cuando estos servicios ya no estén disponibles.

El sistema a construir pretende mejorar esta deficiencia desacoplando la funcionalidad de aquellas *APIs* que puedan dejar de funcionar, haciendo que sea lo más independiente posible.

2.4.1.1.3.3 Escasa configuración

Uno de los aspectos que se echan en falta en *Radar Covid*, es la **capacidad de personalización** del radar, es decir, no existe ningún **parámetro configurable** que permita escoger cada cuánto se realizan las comprobaciones, o con qué frecuencia se recopilan datos de otros dispositivos, por ejemplo.

El futuro sistema pretende ser lo más ***moldeable posible***, es decir, que aporte la mayor flexibilidad posible a la hora de configurar el sistema, proporcionando, por ejemplo, ajustes de intervalos de registro de localización, cómo cada cuántos segundos se recopila la ubicación o la capacidad de establecer una ventana de tiempo con hora de inicio y de fin, durante la cual se registren las coordenadas. De esta forma, el usuario dispone de total libertad para decidir si desea **sacrificar batería** a costa de mayor **precisión**, o viceversa, pudiendo llegar a un compromiso entre ambas variables.

2.4.1.2 *Health Kit (China)*

La aplicación *Health Kit* **[2]** es una de las distintas alternativas que propone el gobierno de **China** para llevar a cabo el rastreo de contagios. Se trata de una aplicación móvil que todo ciudadano está obligado a descargarse para poder realizar tareas cotidianas como entrar en un supermercado o bien desplazarse en metro. Esta aplicación no es la única herramienta que propone el gobierno chino para controlar los contagios, sino que existen otras que utilizan la geolocalización para controlar en todo momento a los ciudadanos.

2.4.1.2.1 Funcionamiento

La aplicación es accesible a través de ***WeChat***, el *WhatsApp* chino muy extendido en el país y que prácticamente usan todos los ciudadanos. De este modo, los usuarios introducen sus datos personales como **DNI**, **nombre** y **foto**, entre otros, y automáticamente se recopilan datos médicos para proporcionar un **código de color**.

Este código de color es el pilar fundamental de la aplicación y sirve para evaluar el estado de salud de los usuarios. Una vez el usuario introduce los datos, se le asigna un **código QR** que puede tomar tres colores diferentes:

- **Verde**: significa que el usuario está sano y puede continuar su rumbo, o entrar en los locales.
- **Amarillo**: implica que el usuario tiene indicios de estar enfermo y por tanto debería de permanecer en aislamiento preventivo durante 7 días.
- **Rojo**: representa el peor de los estados de salud e implica una cuarentena obligatoria de 14 días en un hotel.

Mediante este código de color, se controlan los accesos a locales, supermercados, transportes públicos y otras áreas de la ciudad, de manera que solo se permite continuar con su rumbo a personas sanas, es decir, con un código de color verde. En aquellas áreas en las que se controla el acceso, se dispone un código QR que los usuarios deben escanear con la aplicación, de forma que el sistema comprueba automáticamente el estado de salud de dichos usuarios, permitiéndoles el paso solo si están sanos.

A diferencia de las otras alternativas, la aplicación china sigue un enfoque **centralizado** en el cual las autoridades sanitarias tienen disponible toda la información de los usuarios, pues esta deja de ser privada y se almacena en los servidores centrales.

No existe mucha información acerca de cómo se generan estos códigos *QR* y qué criterios se tienen en cuenta a la hora de determinar el color que representa el estado de salud del usuario. Sin embargo, parece que esta información se recopila a partir de los informes médicos de los ciudadanos por medio de un proceso gestionado mediante *Big Data.*

2.4.1.2.2 Ventajas

La ventaja principal que proporciona *Health Kit* es el **control exhaustivo** y la **prevención** de contagios, ya que se evita el contacto con personas cuyo estado de salud sea deficiente. El hecho de viajar por la ciudad acompañado de un código de color que determine si el usuario está enfermo o no, hace que las probabilidades de que se propague el virus sean ínfimas.

Además, toda la información de los usuarios se recopila en un repositorio central de forma que las autoridades sanitarias tienen total disponibilidad de estos datos para poder **estudiar** cómo avanza la **pandemia** y los **patrones de contagio** del virus, algo que no sería posible en un sistema descentralizado donde se respete la privacidad de los usuarios.

2.4.1.2.3 Desventajas

Llegados a este punto no es difícil imaginar que el principal inconveniente de esta aplicación es su **intrusividad**, que rompe con cualquier principio ético sobre la **privacidad** de los usuarios que utilizan la aplicación.

Los datos personales de los usuarios pasan a ser públicos una vez se registran en la aplicación, ya que no permanecen en el dispositivo, sino que pasan a formar parte de un almacenamiento global. Lo que es aún más grave desde el punto de vista de la privacidad, es el hecho de que se comparta de manera pública algo tan personal como es el **estado de salud**, el cual se refleja en el dispositivo mediante el código de color.

Esta funcionalidad podría incluso considerarse como una **etiqueta discriminatoria** que da lugar a que solo los ciudadanos sanos puedan disfrutar de los privilegios cotidianos como ir a un supermercado o desplazarse por la ciudad, que aparentemente deberían ser accesibles para cualquier ciudadano. De este modo, prácticamente se estaría controlando la vida de los ciudadanos los cuales **dependen** completamente de un ***software*** que determina si puede o no desplazarse o acceder a un local, causando **presión social** o incluso **tensión** y **miedo** a la hora de revisar el color para poder entrar en algún área, por la posibilidad de ser un ser un "infectado".

Además, desde el lanzamiento y despliegue de la aplicación, han ido apareciendo **rumores** acerca de las operaciones que se realizan en **segundo plano** y que generan sospechas que indican que la información personal de los usuarios pueda estar siendo compartida con la policía o con el estado para satisfacer determinados intereses.

2.5 Evaluación de alternativas

En la sección anterior, se ha realizado una descripción de algunos de los sistemas de rastreo de contactos similares al que se pretende construir para contrastar sus ventajas e inconvenientes. En esta sección, se recogen diferentes alternativas que existen a la hora de afrontar ciertas decisiones de diseño del sistema, así como tecnologías a utilizar o diferentes aspectos relativos a la funcionalidad del mismo. Para ello, se evalúan alternativas de **diseño** e **implementación** del sistema, es decir, se exponen otras herramientas, lenguajes y formas distintas de llevar a cabo el desarrollo del futuro sistema. Para no extenderse demasiado, en las siguientes subsecciones solo se presentan las alternativas más significativas e interesantes en las que se puede enfocar el sistema.

2.5.1 Modo de rastreo: *bluetooth* o geolocalización

El rastreo de los casos positivos se puede realizar principalmente a través de dos tecnologías: ***bluetooth*** o **geolocalización**. Ambas herramientas se encuentran integradas en los dispositivos móviles y cada una de ellas tiene sus ventajas e inconvenientes.

A través del ***bluetooth***, se puede hacer uso de la ***API BLE (Bluetooth Low Energy)*** que proporciona la plataforma *Android*, y que permite escanear y obtener información de los dispositivos cercanos, así como transmitir datos entre dispositivos. La idea es que, si un dispositivo se cruza con otro y están lo suficientemente cerca, entonces ambos dispositivos se **intercambian un código único** que almacenan en su base de datos local. De este modo, si el usuario de alguno de los dispositivos da positivo, entonces su **código** se envía al *backend* o a la nube, y de manera periódica se comprueban los códigos de los positivos comparando con la lista de dispositivos cercanos registrados en local.

Otra alternativa es utilizar la **geolocalización** de los dispositivos para ir almacenando los datos de ubicación de cada usuario de forma periódica y en local. Tras registrar el positivo, dichos datos se suben a la nube y cada cliente realiza una comprobación periódica comparando sus datos de ubicación con las ubicaciones de los positivos obtenidas del servidor.

Ambos enfoques son bastante similares, la única diferencia es que en uno se almacenan **códigos de dispositivos**, y en otro lo que se almacenan son **coordenadas de ubicación**. Para

este sistema, se va a seguir el enfoque de utilizar **geolocalización**, dado que es una de las restricciones impuestas en el proyecto. La Tabla 2.1 con las principales ventajas y desventajas de ambos enfoques.

Tabla 2.1. Ventajas y desventajas de los diferentes enfoques para realizar el rastreo.

Enfoque de rastreo	Ventajas	Inconvenientes
Bluetooth	▪ Mayor privacidad: no se guarda ningún tipo de información crítica. ▪ Comprobación rápida y sencilla: solo hay que comprobar la igualdad de códigos	▪ Lentitud en la transmisión de datos. ▪ Puede no ser preciso. ▪ Riesgos de seguridad. ▪ Interferencias: algunos dispositivos inalámbricos utilizan la misma frecuencia de radio que el *Bluetooth.* Si hay muchos dispositivos cercanos, las señales pueden colisionar y dar lugar a interferencias.
Geolocalización	▪ Obtención relativamente rápida de la ubicación. ▪ No le afectan las interferencias. ▪ Más seguro. ▪ Al tratarse de datos reales de ubicación, se pueden utilizar para hacer históricos sobre los itinerarios de los usuarios.	▪ Menor privacidad: los datos que se almacenan son coordenadas del usuario. ▪ Comprobación costosa y compleja: necesario calcular la cercanía de las coordenadas y las horas ▪ Poca precisión en interiores.

2.5.2 Datos: modelo centralizado vs descentralizado

A la hora de almacenar los datos de los usuarios, en este caso sus datos de ubicación, podemos escoger entre dos modelos distintos, un modelo **centralizado** o **descentralizado**. Según el modelo elegido, el procesamiento de la información será diferente.

En un **modelo centralizado** los datos se encuentran en **un servidor central**, es decir, están agrupados en un solo lugar y son accesibles desde el *backend* de la aplicación. Los datos no tienen por qué estar en un único servidor, sino que pueden estar distribuidos entre varios servidores, pero el aspecto fundamental es que dichos datos pueden ser **procesados** desde el **servidor** y por tanto dejan de ser privados.

En contraposición, en un **modelo descentralizado** los datos se encuentran distribuidos entre los distintos **clientes**, que en este caso se corresponden con los dispositivos móviles. Esto implica que los datos no están disponibles en el servidor, a no ser que se envíen explícitamente, y por tanto se **mantiene la privacidad** de los usuarios, que son quienes tienen control total sobre su información personal almacenada en su dispositivo móvil. Esta información no será enviada al servidor hasta que el usuario de su consentimiento.

La **diferencia** fundamental, es que, en el caso del modelo **centralizado**, la comprobación de los datos de ubicación se haría en el **servidor**, liberando dicha carga de trabajo de los dispositivos móviles, mientras que en el modelo **descentralizado** esas comprobaciones se realizan de **manera individual** en cada dispositivo, luego los datos privados siempre permanecen en local hasta que el usuario decida enviarlos al servidor.

El modelo centralizado y el descentralizado se corresponden con los protocolos abiertos ***PEPP-PT*** **[3]** y ***DP-3T*** **[4]** respectivamente, desarrollados recientemente por diferentes instituciones como medida para el rastreo de contactos de positivos.

2.5.3 Base de datos SQL vs NoSQL

A la hora de desarrollar la capa de persistencia del servidor *web* es necesario escoger entre una base de datos relacional, con relaciones y entidades, o bien, una base de datos documental, más sencilla, donde los datos se almacenan en forma de documentos y colecciones.

Para este proyecto, se ha optado por la segunda opción, almacenando los datos en una base de datos **no relacional**, es decir, los datos se almacenan en colecciones de documentos. Esto simplifica considerablemente la implementación de la capa de persistencia, ya que no es necesario diseñar previamente el esquema de la base de datos, incluyendo las restricciones de integridad, sino que, en una base de datos documental, no existe un esquema como tal, sino que los documentos son algo flexible. Esto da la capacidad de moldear las entidades de la base de datos de manera dinámica.

Otra de las ventajas de utilizar un enfoque no relacional, es la **rapidez de las consultas**, pues no se requiere de comprobaciones previas de las restricciones de integridad, ni de ir enlazando tablas para obtener los datos deseados. En el enfoque de documentos y colecciones es tan fácil como acceder directamente a las colecciones deseadas y hacer **consultas simples** formadas simplemente por cláusulas *where*. Esto implica que no se pueden hacer consultas complejas ni *joins* entre tablas, lo cual supone una desventaja respecto del enfoque relacional.

Al no permitir realizar *joins* es importante definir correctamente el esquema en el que se organizarán los documentos y colecciones, para evitar posibles problemas en el futuro y así facilitar la implementación de las consultas.

2.5.4 Desarrollo nativo vs multiplataforma

Otra de las alternativas existentes a la hora de desarrollar la aplicación móvil surge al decidir si realizar la aplicación a través del *framework* nativo de *Android* o por el contrario hacer uso de *frameworks* multiplataforma que permitan desarrollar para distintos sistemas operativos. En el contexto de este sistema, se ha optado por desarrollar la aplicación únicamente para los dispositivos ***Android***. El motivo de esta decisión es principalmente la simplicidad a la hora de utilizar las características de **geolocalización** en los dispositivos.

Al trabajar directamente sobre el sistema operativo nativo, en este caso *Android*, es más fácil acceder a las herramientas de bajo nivel del sistema, como, por ejemplo, las utilidades para

establecer alarmas para ejecutar servicios de *Android*. La razón principal para trabajar directamente con *Android* nativo es el uso de las herramientas del ***GPS*** proporcionadas por *Google Play* para obtener información sobre las coordenadas del dispositivo móvil.

Por otro lado, la principal ventaja de desarrollar la aplicación a través de un *framework* multiplataforma es la capacidad de desplegar la aplicación no solo para los dispositivos *Android*, sino también para otros *SSOO* como *iOS*, aunque todo ello a costa de una mayor dificultad a la hora de utilizar las herramientas proporcionadas por el sistema operativo de los dispositivos móviles. Otra ventaja de los *frameworks* multiplataforma es la posibilidad de desentenderse de las distintas **versiones** dentro de cada sistema operativo. En el caso de *Android* existen multitud de *APIs* y versiones distintas lo cual da lugar a **inconsistencias** y posibles fragmentos de código que pueden dejar de funcionar a partir de ciertas versiones, tal y como se verá más adelante. Esto hace que sea necesario contemplar en todo momento las versiones existentes y las diferencias que hay entre ellas para garantizar el correcto funcionamiento de la aplicación con la mayor consistencia posible entre distintas versiones.

2.5.5 Arquitectura MVVM vs MVC

Como se verá en el capítulo dedicado al diseño del sistema, para construir la aplicación móvil *Android*, se hará uso de la denominada arquitectura *MVVM* **[5]**. Esta arquitectura implica dividir el sistema en una serie de componentes: modelos, vistas y *viewmodels*. Los más interesantes son los *viewmodels*, los cuales contienen unos objetos **observables** que son los que representan los datos de las vistas. De este modo, las vistas **observan** a dichos objetos **observables** recibiendo actualizaciones en la vista cada vez que cambian los datos. Esto facilita enormemente la implementación gracias al patrón *Observer*, además de que es mucho más mantenible y respeta la separación de responsabilidades entre las vistas y los datos.

Por otro lado, está la alternativa tradicional de utilizar el clásico Modelo-Vista-Controlador. En el contexto de la aplicación móvil, los controladores se corresponden con los fragmentos o actividades en *Android*, y las vistas son los propios ficheros *XML*. Este enfoque es menos eficiente que el anterior, ya que en él no está tan clara la separación de dependencias entre los componentes, además de que no se actualizan las vistas automáticamente cuando se modifican los datos en el modelo.

Capítulo 3. Aspectos Teóricos

En este capítulo, se pretende describir y explicar aquellos aspectos teóricos más relevantes y que están involucrados con el proyecto, para introducir al lector en la jerga técnica y para que comprenda mejor el resto de las secciones del documento, las cuales explican más a fondo la estructura interna del sistema. Estos aspectos engloban no solo conceptos teóricos como tal, sino también herramientas, plataformas y cualquier utilidad que se haya utilizado en el desarrollo del proyecto.

Se pondrá mayor énfasis en los conceptos teóricos del *framework* de *Android*, ya que son aquellos que tienen un mayor impacto en el proyecto y los más interesantes a la hora de conocer e investigar las herramientas proporcionadas por este *framework*. También se darán unas pinceladas sobre aspectos teóricos de la pandemia y la transmisión del virus para establecer una base teórica de cara al análisis del sistema. Por último, se enumerarán el resto de las herramientas, plataformas o conceptos genéricos relacionados con el proyecto y su desarrollo, pero en este caso con un menor nivel de detalle dada su menor relevancia.

3.1 Aspectos teóricos acerca del COVID-19

En el apartado de introducción se habló acerca de la situación sanitaria que predomina actualmente, y que rodea al entorno del sistema que se desea construir. Esta sección de aspectos teóricos tiene como objetivo describir los conceptos más relevantes acerca del virus COVID-19, centrándose especialmente en el contagio y la transmisión entre personas, así como periodos de incubación, distancia mínima, duración de la enfermedad y cualquier otra característica que afecte de alguna manera al modo de funcionamiento del sistema.

Es importante aclarar que el objetivo de esta sección no es realizar un análisis exhaustivo del virus, sino mencionar las características más relevantes. Para obtener información detallada y fiable acerca del COVID-19 se puede consultar este documento mantenido y actualizado por Ministerio de Sanidad **[1]**, también consultando la página de CDC **[6]**.

3.1.1 Definición y origen de la enfermedad

El COVID-19 proviene de lo que se denomina **la familia de los coronavirus**, lo cual significa que los coronavirus ya existen mucho antes del comienzo de esta pandemia, y se manifestaban en forma de enfermedades responsables de infecciones respiratorias, como puede ser el resfriado común o el síndrome respiratorio agudo grave (SARS por sus siglas en inglés). Sin embargo, en el año 2019 se descubrió un nuevo tipo de coronavirus, el ***SARS-CoV-2***, al que se le conoce popularmente como COVID-19 o *coronavirus,* aunque este último término recibe un uso erróneo pues, la palabra "coronavirus" hace referencia a la familia de virus que causan infecciones respiratorias y no a uno en concreto, como puede ser el COVID-19.

A día de hoy, aun no se puede garantizar con certeza cuál es el **origen** de esta enfermedad, aunque existen indicios de que el virus proviene de una familia de murciélagos, y que por medio de un intermediario se haya transmitido a los humanos. Se sospecha que dicho

intermediario puede tratarse de algún animal vivo con los que se comercia en el mercado de ***Wuhan, China***, probablemente un pangolín.

3.1.2 Síntomas

Aunque la manifestación de la enfermedad puede presentar síntomas diferentes en función del paciente, existen un conjunto de síntomas comunes que se reflejan en todos los casos de COVID-19. A continuación, se listan los **síntomas más habituales** de acuerdo con el CDC y al Ministerio de Sanidad:

- Fiebre
- Tos seca
- Dolor de garganta
- Cansancio y fatiga
- Dificultad respiratoria
- Escalofríos

Existen otros síntomas **menos frecuentes**, pero que también han sido observados en diversos pacientes. Estos síntomas menos comunes son:

- Dolor de cabeza
- Pérdida del sentido del olfato o del gusto
- Diarrea
- Dolores musculares
- Vómitos/nauseas
- Erupciones cutáneas

Además de todos estos síntomas, existen algunos **signos** que representan un **caso grave** de COVID-19, y en que en caso de padecer alguno de estos signos es necesario acudir a servicios médicos urgentemente. Alguno de estos signos graves, pueden ser:

- Sensación de falta de aire
- Disnea
- Dolor o presión fuerte en el pecho
- Incapacidad para hablar, despertarse o permanecer despierto
- Confusión
- Coloración azulada de los labios

Cabe decir que esta lista de síntomas no es definitiva, ya que pueden ir surgiendo nuevos síntomas al cabo del tiempo, según va evolucionando el virus. Si se desea obtener la última versión de la lista de síntomas actualizada, se puede consultar desde la página oficial del CDC **[6]**.

3.1.3 Vías de contagio y transmisión

Una vez definida la enfermedad, su origen y algunos de los síntomas más comunes detectados hasta la fecha, vamos a pasar a ver cómo se contagia el virus entre personas.

La transmisión del virus se realiza a través del aire cuando una persona con COVID-19 habla, tose, estornuda o respira con mucha intensidad. Esto hace que se expulsen de la boca y nariz unas partículas que se dispersan en el aire, las denominadas **partículas líquidas respiratorias.** Estas pueden ser de diversos tamaños y contener distinta **carga viral**, de forma que, si se depositan en la membrana mucosa, es probable que el virus prolifere y dicha persona termine contagiándose. El riesgo de contagio se reduce si existe una distancia de al menos **2 metros** entre ambos individuos, pues las partículas respiratorias se desintegran según viajan a través del aire, por tanto, **a mayor distancia**, **menor** será la **carga viral**. Sin embargo, también existe el riesgo de que estas partículas permanezcan el aire durante horas y no se lleguen a desvanecer, con lo cual la persona que inhale estas partículas también podrá contraer el virus. Esto es lo que se conoce como **transmisión por aire.**

Otro de los factores que pueden hacer que aumente el riesgo de contagio, es la ubicación de los individuos, pues si se encuentran en un **espacio cerrado** con **ventilación deficiente**, también se incrementa la probabilidad de que se transmita el virus.

También existe la **transmisión por contacto directo,** que como cabe esperar, es la situación más favorable para la propagación del virus, pues en estos casos, la probabilidad de que el virus se transmita es prácticamente del 100 %.

Por último, existe otro medio de transmisión mucho menos común que los anteriores, se trata de la **transmisión por contacto de superficies contaminadas.** En este escenario, las partículas contagiosas se depositan sobre alguna superficie, que luego es tocada por alguna persona. Si esta persona se lleva las manos a la boca, ojos o nariz, existe un pequeño riesgo de contagio, aunque no suele ser frecuente.

3.1.4 Periodo de contagio e infectividad

Dado que el proyecto consiste en una aplicación de **rastreo** de COVID-19, es muy importante conocer con detalle el **ciclo de vida del contagio**, es decir, ver cómo va evolucionando la **carga viral** desde que el virus se introduce en el organismo hasta que empiezan a aparecer los primeros síntomas, incluyendo los días posteriores hasta la recuperación total.

Para poder llevar a cabo un rastreo de los contagios, es necesario comprender lo que se denomina **periodo de infectividad**. El periodo de contagio o infectividad consiste en un plazo de tiempo durante el cual, una persona con COVID-19, es contagiosa y por tanto puede transmitir el virus a otras personas. La probabilidad de que la persona transmita el virus durante este periodo no es constante, sino que, en función de los días transcurridos, el grado de infectividad será más o menos intenso. La infectividad está ligada a la **carga viral** que reside en el organismo, esto es, la cantidad de material vírico que está presente en el organismo del infectado, por lo que, a mayor carga viral, más infecciosa será la persona. No existe una ventana de tiempo fija durante la cual una persona puede contagiar a otras, sino que varía de unas personas a otras en función de diversas variables, como la variante del virus, la intensidad de la enfermedad u otros. Sin embargo, la mayoría de los casos presenta una ventana de

tiempo de infección similar, que va desde los **2-3 días antes** de **presentar los síntomas,** hasta los **7 o incluso 10 días después**.

El **periodo de incubación** de la enfermedad oscila entre los 2 y los 12 días, con un promedio de **5-6 días** desde la entrada del virus en el organismo, lo cual significa que desde el contacto con una persona con COVID-19 hasta que aparecen los primeros síntomas, suelen transcurrir aproximadamente 5 días. Por otro lado, la **duración** de la enfermedad varía en función de si se trata de una infección leve, o un caso grave de COVID-19. En el primer caso, el tiempo medio de recuperación es de **2 semanas** desde el inicio de los síntomas, mientras que, en casos graves, la recuperación se alcanza entre **3** y **6 semanas** después del primer día de aparición de los síntomas.

Sin embargo, no solo hay que tener en cuenta el **estado** en el que se encuentra el individuo infectado, sino también otros factores como el tiempo de exposición y la distancia, lo cual se detalla en la siguiente sección.

3.1.4.1 *Detección de contactos de riesgo*

El grueso de la funcionalidad de la aplicación consiste en la detección de **situaciones de contactos de riesgo,** es decir, detectar aquellos escenarios de la vida cotidiana en los que haya una posibilidad de entrar en contacto con un individuo que es positivo en COVID-19. Para ello, a la hora de comparar la localización de dos individuos (uno de ellos con COVID-19) para ver si ha habido un riesgo de contagio/infección, existen **tres factores** fundamentales a tener en cuenta:

1. Distancia o proximidad.
2. Tiempo de exposición.
3. Grado de infectividad del individuo con COVID-19.

El tratamiento de estos factores debe realizarse de manera **conjunta,** es decir, no podemos basarnos en un solo factor para determinar si se trata de un contacto de riesgo, al menos desde el punto de vista programático, sino que es necesario, contemplar estos tres factores en conjunto. Por ejemplo, si sabemos que el individuo está a 1 metro de distancia del infectado, aún no tenemos suficiente información para afirmar que se trata de un contacto de riesgo, pues también deberíamos comprobar **cuánto tiempo** ha estado con el positivo, y también si el positivo está en una fase de alta infectividad o por el contrario se encuentra en las últimas semanas del ciclo de vida de la enfermedad.

3.1.4.1.1 Distancia o proximidad

La distancia constituye el primer factor a tener en cuenta a la hora de evaluar contactos de riesgo, ya que, si dos individuos no están próximos el uno del otro, no tiene sentido contemplar el resto de los factores. La distancia mínima o **distancia de seguridad**, es una medida estimada de los metros que hay que guardar con el resto de las personas para evitar el contacto con las partículas líquidas respiratorias que se encuentran en el aire.

Según el convenio establecido, lo recomendable es mantener un mínimo de **6 pies** o **1.5-2 metros** entre personas, aunque lo cierto es que esta distancia varía en función de cada situación concreta. Algunos factores que influyen ya se han visto en la sección de vías de

transmisión, por ejemplo, la **ventilación** o si el **lugar es un sitio cerrado** son dos factores importantes que hacen que las partículas permanezcan más tiempo en el aire y sean propulsadas a mayor distancia. Además, no es lo mismo hablar, que estornudar, ya que la fuerza de un estornudo puede llegar a propulsar las partículas hasta **7-8 metros** de distancia, según se indica en los estudios publicados en **[7]**.

Por ello, la distancia a tener en cuenta será variable y dependerá de cada situación, aunque por convenio se tomarán **2 metros** como la medida universal para realizar las comparaciones. Sin embargo, a esta medida estándar, será necesario sumarle una serie de metros de error que variará en función de la precisión de las coordenadas de los individuos.

3.1.4.1.2 Tiempo de exposición

Como ocurre con la distancia, no existe una medida universal infalible acerca del número de minutos que son suficientes para contraer el virus tras estar en contacto con un positivo. De nuevo, entran en juego muchos factores como la carga viral, si la persona habla, tose o estornuda, la ventilación…

De acuerdo con lo que se indica en la última actualización de **julio de 2021** del documento de ***Manejo en Atención Primaria de COVID-19 en Asturias*** **[8]**, se considera como **contacto estrecho** a una persona infectada que, dos días antes de manifestarse los síntomas, haya estado en contacto con otra persona a menos de 2 metros de distancia y durante **más de 15 minutos** consecutivos. Sin embargo, la denominada "regla de los 15 minutos" no es algo inflable a lo que haya que ajustarse de manera exacta, ya que existen casos de infección con menos de 15 minutos de exposición cercana a la persona infectada.

3.1.4.1.3 Grado de infectividad del individuo con COVID-19

Este último factor, ya se comentó en secciones anteriores y hace referencia al **estado** actual en el que se encuentra la persona infectada. El hecho de estar contagiado no implica que necesariamente se transmita el virus a otras personas, ya que depende de los días transcurridos desde el contagio. Si el virus se encuentra en las últimas fases de su vida, es poco probable que se transmita a otros individuos, pues la **carga viral** será menor. En la Figura 3.1, de Guillermo Aldama **[9]**, se muestra una gráfica que refleja la evolución de la **carga viral** en un organismo a lo largo de los **días**.

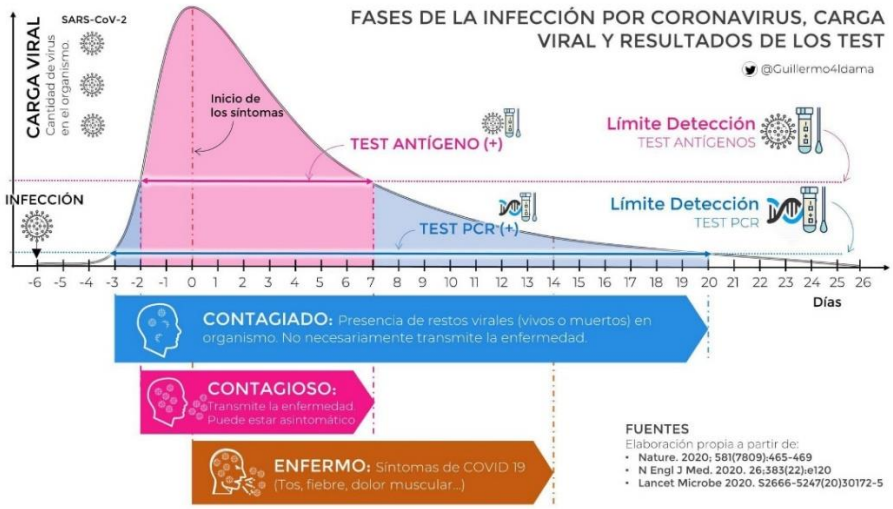


Figura 3.1. Imagen de Guillermo Aldama de las fases de contagio por coronavirus en función de la carga viral.

En la figura superior, el **eje de abscisas** (eje X) representa los **días transcurridos** desde que se contrae la enfermedad hasta que desaparece por completo del organismo, mientras que el **eje de ordenadas** (eje Y) representa la **carga viral** presente en el organismo. Es importante destacar que los días transcurridos no empiezan en 0, sino que se empieza a contar en negativo y se toma como **día 0** el primer día de inicio de los síntomas, tal y como se representa en la gráfica. También podemos observar que desde la infección al inicio de los síntomas transcurren entre 5 y 6 días, lo cual coincide con el periodo de incubación descrito anteriormente.

De acuerdo con la Figura 3.1, existen **tres etapas** o **estados** por los que pasa un individuo infectado. Estos estados se pueden **superponer**, es decir, el infectado puede estar en más de uno de estos estados a la vez. Estos estados son:

- **CONTAGIADO:** engloba a todo el ciclo de vida de la enfermedad, desde que el virus entra en el organismo, hasta su desaparición total. En este estado, no necesariamente se transmite el virus.
- **CONTAGIOSO:** el infectado se encuentra en el estadio con mayor infectividad de la enfermedad. Se transmite el virus a otras personas. La carga viral existente en el organismo llega a sus picos más altos, siendo máxima al inicio de los síntomas.
- **ENFERMO:** representa el periodo de tiempo desde que se manifiestan los primeros síntomas hasta que desaparecen. Al igual que con CONTAGIADO, en este estado el virus se puede transmitir o no.

El área bajo la curva de color **rosado** representa el periodo de infectividad, que va desde los **dos días** antes del **día 0**, es decir, del inicio de los síntomas, y se prolonga hasta **7 días** después del **día 0,** dando como resultado un total de **9-10 días** aproximadamente durante los cuales la persona infectada está en estado CONTAGIOSO, y por tanto puede transmitir el virus con facilidad. Por el contrario, las zonas **azules** representan los periodos temporales durante los cuales no se transmite el virus, o existe una probabilidad muy baja.

De manera adicional, la gráfica también refleja los **límites de detección** del virus para los dos tipos de ***tests*: *ANTÍGENOS*** y **PCR.** El *test* de antígenos tiene **menor sensibilidad** a la hora de

detectar la carga viral en el organismo, por lo que solamente notificaría el positivo si la persona se encuentra en algún momento del periodo de infectividad, o lo que es lo mismo, en el estado **CONTAGIOSO**, pues en otros casos, este *test* no sería capaz de detectarlo. En cambio, el *test* PCR es el más preciso y sensible ya que es capaz de detectar la presencia de carga viral incluso 3 días antes del comienzo de los síntomas, y hasta casi la total desaparición del virus, es decir, en cualquier instante del tiempo mientras la persona esté en estado **CONTAGIADO.**

En el contexto del proyecto, lo que realmente interesa es el estado **CONTAGIOSO** o **periodo de infectividad**, pues será necesario tener en cuenta los días transcurridos dentro de esta ventana temporal para poder realizar las comprobaciones de contactos de riesgo entre diferentes individuos.

3.2 Aspectos teóricos de *Android*

En esta sección, se introducen los aspectos más relevantes relacionados con la tecnología *Android*, que será la plataforma de destino de la aplicación cliente, y que afectan en gran medida tanto a la implementación como a los requisitos funcionales del sistema. Por ello, es importante dedicar unas secciones del documento a describir detalladamente las principales características técnicas de aquellos elementos que afectan al presente proyecto, como por ejemplo la geolocalización o la ejecución de tareas en segundo plano.

En primer lugar, se describe en términos generales el sistema operativo *Android*, para luego pasar a ver cómo funciona la localización y qué alternativas podemos seguir para utilizar dicha funcionalidad de manera programática. De manera adicional, también se describen las principales tecnologías que ofrece *Android* para la ejecución de tareas y trabajos en segundo plano, así como un análisis de las estrategias actuales que utilizan los dispositivos para promover el ahorro de batería. Por último, se habla de las divergencias más destacadas que existen entre las versiones de *Android*, centrándose sobre todo en aquellas que afectan a la localización y a la ejecución en segundo plano.

3.2.1 Sistema operativo Android

Android **[11]** es un sistema operativo diseñado para ejecutarse en dispositivos móviles inteligentes con pantalla táctil, aunque cada vez son más los dispositivos que conforman la familia de este sistema operativo, pasando por *Smart TVs*, relojes o incluso las pantallas de los automóviles. Su núcleo está basado en el ***kernel*** de ***Linux*** y el código fuente que conforma la parte de más bajo nivel recibe el nombre de ***AOSP*** (*Android Open Source Project*) **[10]**, es decir, un proyecto de código abierto que se puede consultar de manera libre para revisarlo. Este sistema operativo fue creado por la empresa *Android Inc* en 2003, y tras pasar solo dos años, fue adquirido por *Google* en 2005, quien lo iría desarrollando y mejorando hasta día de hoy. De este modo, irían surgiendo numerosas versiones del sistema operativo, cada una con nuevas características notables respecto de la anterior o incluso con restricciones de seguridad, lo cual hace que ciertas partes del código de una aplicación *Android* funcionen en unas versiones y no en otras, como se verá más adelante.

3.2.2 Geolocalización en Android

La geolocalización es uno de los componentes vitales que conforman el núcleo del sistema y quizás el más importante, dado que el grueso de la funcionalidad se basa principalmente en esta herramienta. Por ello, es importante describir las alternativas existentes para trabajar con la localización en *Android* **[12]**, haciendo una revisión de las ventajas y desventajas de cada una, y analizando de qué manera afectan las distintas restricciones impuestas por el sistema operativo, así como sus versiones.

Hoy en día, la gran mayoría de dispositivos móviles llevan integrado un *chip* que permite geolocalizarlos y cuya funcionalidad puede ser utilizada mediante diferentes abstracciones que proporciona *Android*. Sin embargo, cada fabricante presenta sus peculiaridades, ya sea añadiendo capas de abstracción al núcleo de *Android* para incluir funcionalidades exclusivas, variaciones en los *chips* y sensores integrados, mejoras para el ahorro de batería, restricciones de ejecución u otras características, dando lugar a una gran **heterogeneidad** a la hora de desarrollar *software* para *Android*. Si además se tiene en cuenta el extenso repertorio de versiones de *Android,* el abanico de posibilidades aumenta y esta divergencia no solo se manifiesta en términos de funcionalidad, sino también en restricciones nuevas o limitaciones a las que se tienen que ajustar los desarrolladores de *software,* y que como se verá, generan un impacto negativo a la hora de desarrollar aplicaciones relacionadas con la localización.

En primer lugar, se detallan las distintas fuentes de procedencia que utiliza *Android* para obtener información acerca de la posición del dispositivo, para luego describir las diferentes capas de abstracción o *APIs* que se encuentran disponibles para trabajar con la localización. Al final de la sección, se añade un resumen de los distintos proveedores, así como una tabla comparativa con las ventajas e inconvenientes que presenta cada uno.

3.2.2.1 *Proveedores de localización en Android*

Cuando se habla de localización, *Android* proporciona una serie de capas de abstracción que permiten trabajar con esta información sin necesidad de involucrarse en aspectos de muy bajo nivel. Los componentes fundamentales de una localización son la **latitud** y la **longitud**, pero también existen otros atributos como por ejemplo la **altitud**, la **precisión** (***Accuracy***), la **velocidad** (***speed***) y el **proveedor** (***provider***). Este último representa la forma o medio a través del cual se obtiene la localización del dispositivo. Independientemente de la capa de abstracción o *API* que se utilice, a bajo nivel se estará utilizando alguno de estos proveedores para obtener la información. De este modo, cada proveedor tiene sus ventajas e inconvenientes, lo cual hace que sea necesario analizar el problema concreto para ver qué proveedor se ajusta mejor a la solución. A continuación, se describen los tres proveedores principales que hoy en día utilizan los dispositivos *Android* para obtener los datos de geolocalización.

3.2.2.1.1 Proveedor GPS

El proveedor *GPS (Global Positioning System*) hace uso del *chip* integrado en los dispositivos para obtener la localización a través de **satélites**. De esta forma, se utiliza un sistema de satélites que envían señales al *chip* del dispositivo, que se conoce como **receptor**. El receptor

responde enviando de nuevo las señales a dichos satélites para calcular el tiempo que tardan en llegar y así calcular la distancia aproximada. Con 3 satélites es posible determinar una localización en dos dimensiones, aunque lo normal es que se utilice un número más elevado de satélites. Además de las coordenadas, este sistema permite calcular también la dirección y velocidad de un dispositivo.

El correcto funcionamiento de este sistema depende exclusivamente de que los receptores de los dispositivos estén a la vista de los satélites para poder enviar y recibir las señales, por lo que **en zonas interiores** o con demasiada obstrucción aérea, este método **no funcionará correctamente**. Sin embargo, se trata de la **forma más precisa** de obtener la posición de un objeto, cometiendo errores de apenas unos metros.

Además de la necesidad de permanecer en lugares abiertos, este sistema también tiene dos limitaciones importantes. Por un lado, se trata de un **sistema lento** y que puede tardar bastante **tiempo** en encontrar una **localización fija** para el dispositivo, por lo que las primeras actualizaciones de localización pueden tardar más de lo previsto. También es **costoso** en cuanto a los **cálculos** que se deben de realizar, por lo que consume más batería que otros proveedores, lo cual puede derivar en **drenajes y descargas de la batería** importantes si se utiliza de forma intensiva.

Para poder utilizar este proveedor, es necesario definir el permiso de *Android* ***ACCESS_FINE_LOCATION***, que permite acceder a los datos de localización más precisos utilizando el *GPS.*

3.2.2.1.2 Proveedor de red

Este proveedor se conoce como *Network Provider* ya que utiliza las **redes de *internet*** disponibles para construir un **mapa de red** y así obtener una localización, a diferencia del *GPS* que hace uso de los satélites. Existen diferentes alternativas a su vez para obtener actualizaciones de la localización a través de la red. Una se basa en la utilización de ***Wifi*** para detectar puntos de acceso o *routers*, analizando la intensidad de las señales y la dirección *MAC* para determinar una localización aproximada del dispositivo. Otra forma es hacer uso de receptores de ***bluetooth*** para triangular la posición del dispositivo o bien utilizar las **torres de telefonía** a las que se conecta el dispositivo móvil, así como los identificadores de estas torres para ir registrando la posición. Estas técnicas no son excluyentes, sino que se pueden usar en conjunto para obtener localizaciones con la máxima disponibilidad posible.

Como ocurría con los satélites, si el dispositivo no está conectado a una torre de telefonía o a un punto de acceso, o bien la intensidad de la señal *Wifi* no es lo suficientemente buena, este sistema no funcionará. Además, ciertos puntos de acceso pueden no estar disponibles o algunas torres pueden no proporcionar la suficiente información y por tanto dar lugar a malas aproximaciones a la hora de realizar los cálculos. Todo ello, hace que las **estimaciones** de la posición de un dispositivo sean de **peor calidad** que las que proporciona el *GPS*, por lo que las localizaciones tendrán una **menor precisión,** cometiendo unos porcentajes de error considerables.

Las ventajas que presenta la localización por red frente al *GPS* son principalmente la **rapidez,** la **disponibilidad** y el **ahorro de energía**. La numerosa cantidad de fuentes de red y de telefonía, así como la proximidad de estas al dispositivo móvil, hace que los cálculos se realicen

rápidamente, obteniendo actualizaciones en cuestión de segundos. Además, no es necesario que el espacio aéreo esté despejado, sino que se pueden seguir obteniendo actualizaciones aun estando en interiores o en zonas cubiertas y obstruidas, dando lugar a una mayor disponibilidad. Por último, obtener las coordenadas a través de la red no resulta tan costoso como utilizar satélites, por lo que el consumo de batería es mínimo.

Para hacer uso de este proveedor, se requiere al menos el permiso ***ACCESS_COARSE_LOCATION***, que permite acceder a ubicaciones proporcionadas por los datos de red, que no son tan precisas como las que aporta el *GPS*.

3.2.2.1.3 Proveedor pasivo

Como su propio nombre indica, se trata de un sistema *lazy*, es decir, no solicita directamente actualizaciones de localización, sino que aprovecha las solicitudes realizadas por otras aplicaciones para obtener datos de localización. Por tanto, no se puede garantizar la fuente de las coordenadas, es decir si proceden del *GPS* o de *internet,* y no se asegura que se obtengan datos si ninguna aplicación obtiene localizaciones.

En cuanto a los permisos necesarios, como mínimo se necesita ***ACCESS_FINE_LOCATION*** ya que no se conoce de antemano el proveedor a utilizar.

3.2.2.1.4 Resumen y comparación de proveedores

Una vez vistas las tres alternativas, y teniendo en cuenta la importancia de la precisión de las coordenadas en este sistema, el proveedor más adecuado y que mejor se ajusta a las necesidades del *software*, es aquel que proporciona una mayor exactitud en las coordenadas, es decir el ***GPS***. Sin embargo, no siempre será posible obtener localizaciones a través esta fuente en todas las situaciones, en cuyo caso también será necesario utilizar el *NetworkProvider*, sacrificando **exactitud** y **precisión** a cambio de la **disponibilidad** de las coordenadas. La **¡Error! No se encuentra el origen de la referencia.** resume las fortalezas y limitaciones de los distintos proveedores de localización.

Tabla 3.1. Comparación entre los diferentes proveedores de localización.

Proveedor	Ventajas	Limitaciones
GPS_PROVIDER	▪ Alta precisión.	▪ No disponible en interiores o zonas con el espacio aéreo obstruido. ▪ Mayor lentitud para obtener coordenadas. ▪ Alto consumo de batería.
NETWORK_PROVIDER	▪ Mayor rapidez para fijar una localización. ▪ Alta disponibilidad (funciona en interiores) ▪ Bajo consumo de batería.	▪ Baja precisión. ▪ Necesidad de estar conectado a una red.
PASSIVE_PROVIDER	▪ Apenas consume batería.	▪ No se garantiza que haya localizaciones disponibles. ▪ Precisión variable (depende de otras aplicaciones) ▪ No proporciona control sobre la solicitud de localizaciones.

Es importante tener en cuenta que el funcionamiento de estos proveedores solo tendrá efecto si el usuario da su consentimiento y acepta las condiciones y **permisos** necesarios relativos a la obtención de la localización de su dispositivo. Estos permisos son requeridos en tiempo de ejecución y es necesario implementar la solicitud de los mismos haciendo uso de la *API* que proporciona *Android* para trabajar con permisos. Además, la **ubicación** del dispositivo debe estar activada en todo momento para poder recibir actualizaciones de localización. En función del modelo y del fabricante, el usuario tendrá libertad para activar o desactivar las fuentes de procedencia de dicha ubicación, ya sea *GPS* o *NETWORK*, e incluso la capacidad de configurar la precisión a su gusto, pudiendo sacrificar batería a cambio de una mayor precisión. En la sección del estudio legal sobre la privacidad se detallan más a fondo los aspectos relativos a los permisos y la localización del usuario.

3.2.2.1.5 Precisión de las coordenadas

Independientemente de los proveedores de localización que se utilicen, todos los datos de ubicación proporcionados por las abstracciones tienen asociado una **precisión** o ***Accuracy***, que mide la calidad de las coordenadas en términos del grado de similitud con la posición real.

Según la documentación de *Android,* la **precisión horizontal** de una posición ***p(lat, lng)*** se define como el **radio** de **68 % de confianza**, es decir, que existe un 68 % de probabilidades de que la localización real del usuario se encuentre dentro de la circunferencia de radio igual al *accuracy* en metros y con centro en `p(lat, lng)`. Por ejemplo, si la coordenada ***p1*** tiene un *accuracy de* ***6***, significa que hay un 68 % de probabilidades de que la ubicación real esté dentro de la circunferencia de 6 metros de radio y con centro *p1.*

Por tanto, podemos deducir que a menor valor de *accuracy*, más precisa será dicha coordenada y por tanto tendrá mayor calidad.

3.2.2.2 *Abstracciones para acceder a la localización*

Los proveedores de localización vistos anteriormente solo son distintas **opciones** de configurar la fuente de obtención de las coordenadas, es decir, por si solos no sirven para trabajar con la localización de los dispositivos. Para ello, el *framework* de *Android* proporciona una serie de **niveles y capas de abstracción,** las cuales exponen servicios de localización a través de una *API* amigable para los desarrolladores.

Actualmente, existen dos abstracciones o *APIs* distintas para trabajar con la geolocalización en *Android*, las cuales representan distinto nivel de detalle. A más bajo nivel, está la ***API*** de **servicios** de **localización nativa de *Android* [13]**, que encapsula los distintos componentes para obtener actualizaciones periódicas de la localización. Por encima de esta abstracción y desde un punto de vista de más alto nivel, se encuentra la **alternativa de *Google*** conocida como ***Fused Location Provider API* [14]**, que añade una serie de mejoras sobre la capa tradicional de localización más cercana al núcleo de *Android.*

En las siguientes secciones se describen las dos alternativas, destacando las principales ventajas e inconvenientes que presenta cada una, así como las principales diferencias que existen entre ambas abstracciones.

3.2.2.2.1 Android Location Services

Esta *API* representa la capa de más bajo nivel para acceder a los servicios de localización en *Android*. Está disponible desde las primeras versiones del sistema operativo, lo cual la convierte en la alternativa **más tradicional** y cercana al **núcleo de *Android***. Esta cercanía, permite una mayor **flexibilidad** desde el punto de vista del programador, ya que es posible configurar la **fuente de obtención** de coordenadas, es decir, se puede escoger cuál de los tres ***providers*** utilizar, ya sea el *GPS,* el *NETWORK* o el *PASSIVE.*

Sin embargo, esto implica una mayor dificultad a la hora de utilizar la *API* ya que es necesario tener en cuenta estos detalles de bajo nivel para configurar los servicios de localización. Además, obliga al programador a **asumir** la **responsabilidad** de seleccionar el *provider* que mejor se ajuste a cada **situación,** lo cual se traduce en implementar más código extra para gestionar la lógica de selección de la fuente de coordenadas.

Si se selecciona como fuente el ***GPS***, las actualizaciones de localización solo se reciben al **aire libre**, dada la tecnología por satélite que se utiliza para triangular el dispositivo, la cual se comentó en secciones anteriores. De este modo, es necesario comprobar la calidad de las coordenadas manualmente con cada actualización, y en base a dicha medición ir alternando entre *GPS* y *NETWORK* si queremos obtener la máxima disponibilidad.

3.2.2.2.2 Fused Location Provider API

Se trata de una capa de abstracción de más alto nivel construida sobre los servicios de localización nativos de *Android*, de forma que los "**decora**" añadiendo nuevas características que hacen que esta *API* sea más sencilla de utilizar. La *API Fused Location Provider* es la alternativa desarrollada por ***Google*** que viene incluida dentro de ***Google Play Services***, por lo que es necesario tener instalados los servicios de *Google Play* en el dispositivo para poder utilizarla.

La palabra ***Fused*** hace referencia a que mediante esta *API* ya no es necesario contemplar los distintos *providers* de localización como ocurría con la *API* nativa, sino que en este caso se **fusionan** todos los ***providers*** desplazando la responsabilidad del programador para seleccionar un proveedor de manera manual. Esto hace que la abstracción proporcione un **enfoque declarativo**, en el sentido de que ya no es necesario gestionar la lógica de selección *provider* manualmente, sino que es suficiente con declarar las necesidades y condiciones deseadas en las que se desea obtener actualizaciones de localización, y el *fused provider* ya se encarga automáticamente de decidir qué fuente utilizar para obtener las coordenadas. Además, se pueden configurar **peticiones de localización**, que representan las condiciones y características que se desea que tengan las actualizaciones de localización, esto es, seleccionar por ejemplo entre alta precisión (*HIGH ACCURACY*) o ahorro de energía (*LOW POWER*), o bien definir el mínimo intervalo de tiempo que debe pasar entre una actualización de localización y otra. También se puede indicar cuántos metros de diferencia debe haber como mínimo entre dos localizaciones para que se reciba una actualización.

Todas estas características aportan una máxima **disponibilidad**, independientemente de si se está al aire libre o en interiores, pues la *API* ya gestiona por debajo la lógica necesaria para alternar entre distintas fuentes de localizaciones, seleccionando la más adecuada en cada contexto y ajustándose a la configuración indicada a través de las peticiones de localización.

Como la *API* ya realiza la gestión por su cuenta, se **simplifican** las operaciones y el manejo necesario para trabajar con los servicios de localización, pues permite **abstraerse** de implementar **compromisos** entre **ahorro** de energía, **precisión** y **disponibilidad**, para centrarse únicamente en la funcionalidad principal.

Además, la *API Fused* de *Google* es ***battery-friendly,*** es decir, gestiona los recursos para ahorrar batería siempre que sea posible, alternando entre *GPS* y *red* para balancear el consumo energético, pero sin llegar a afectar demasiado a la precisión y a la disponibilidad.

En general, la *API Fused* obtiene localizaciones fijas con mayor **rapidez**, en comparación con la abstracción nativa de *Android* donde pueden pasar varios minutos hasta que se proporcione una buena localización fija del dispositivo. Sin embargo, debido a las capas de abstracción **extra** que se añaden para gestionar todos los aspectos mencionados, es posible que haya situaciones en las que **disminuya** la **precisión** de las coordenadas, obteniendo localizaciones más precisas con el *framework* nativo de *Android,* aunque no suele ser lo común.

Por último, cabe destacar que esta *API* no está integrada directamente en el *framework* nativo de *Android*, sino que forma parte de los servicios de *Google Play*, lo cual significa que se puede actualizar independientemente del sistema operativo, obteniendo una mejora continua con cada nueva actualización de los servicios de *Google Play* y sin tener que esperar a una nueva actualización de *Android*. Esta característica también tiene sus inconvenientes, pues es necesario que los dispositivos dispongan de *Google Play Services*, de lo contrario esta *API* no funcionará.

3.2.2.2.3 Resumen de ambas alternativas

A modo de resumen, en la Tabla 3.2 se contrastan las principales ventajas e inconvenientes de cada *API* de servicios de localización disponible en *Android.*

Tabla 3.2. Ventajas e inconvenientes de las dos APIs de localización disponibles para Android.

API de Servicios de Localización	***Ventajas***	***Inconvenientes***
Android Location (nativo)	▪ Se puede utilizar en cualquier dispositivo *Android,* aunque no tenga instalado *Google Play Services.* ▪ Flexibilidad para escoger el *provider (GPS, NETWORK, PASSIVE).*	▪ Más complejo de utilizar. ▪ No se actualiza frecuentemente (solo cuando hay una actualización del *SO*). ▪ Mayor consumo de batería.
Fused Location (Google Play)	▪ Enfoque declarativo (simplicidad). ▪ Rapidez. ▪ Gestiona automáticamente la selección del *provider*. ▪ Máxima disponibilidad. ▪ Se actualiza con frecuencia independientemente del *SO.* ▪ Centrado en disminuir el consumo de batería.	▪ Servicios de *Google Play* necesarios para funcionar. ▪ Es posible que en algún caso se disminuya la precisión.

3.2.3 Ejecución de tareas en segundo plano

Además de la localización, la ejecución en segundo plano **[15]** es otro de los aspectos relevantes que tienen un gran impacto en el sistema a desarrollar. Esto es así debido a que gran parte de la funcionalidad a implementar se basa en la ejecución de tareas en segundo plano, es decir, mientras la aplicación no es visible al usuario.

Cuando hablamos de segundo plano o *background* nos referimos a aquel estado de la aplicación en el que el usuario no está interactuando con la aplicación, ya sea porque la aplicación está cerrada o simplemente ha sido minimizada. En la tecnología *Android* podemos distinguir entre **dos tipos** de tareas en segundo plano:

1. **Tareas exactas:** son aquellas tareas que requieren de una ejecución exacta en el tiempo.
2. **Tareas diferidas:** tareas que no necesitan de una ejecución inmediata y por tanto se pueden diferir, es decir, trasladar en el tiempo para ser ejecutadas cuando el sistema operativo lo considere oportuno.

En función del tipo de trabajo que se desee realizar en segundo plano, se utilizarán unas tecnologías u otras dentro del amplio repertorio de posibilidades que ofrece *Android* para ejecutar tareas en *background*. Dentro de este repertorio podemos distinguir multitud de *APIs* y utilidades del *framework* como por ejemplo los **servicios,** el ***JobScheduler***, el ***WorkManager***, los **hilos** convencionales, las **corrutinas** en ***kotlin*…** y otras muchas herramientas.

En las siguientes secciones se hablará únicamente de los servicios, los *broadcast receivers* y las alarmas, ya que son las tres principales utilidades de ejecución en *background* que mayor influencia tienen sobre la aplicación *Android* que se pretende desarrollar.

3.2.3.1 *Servicios*

Los servicios **[16]** son **componentes** de *Android*, del mismo modo que las *Activities*, pero difieren de estas en que no tienen una interfaz de usuario, sino que permanecen ocultos a simple vista. Por lo general, los servicios tienen un **ciclo de vida independiente** y separado de las actividades de la aplicación, por lo que pueden seguir ejecutándose aun cuando se cierra por completo la aplicación.

Estos servicios están pensados para ejecutar trabajos de larga duración en segundo plano, aunque es importante resaltar que se ejecutan en el **mismo hilo principal** o ***main thread*** que la aplicación, desde donde se realizan todas las operaciones de refresco de los componentes de la interfaz, por lo que aquellos trabajos **bloqueantes** no se deberían de ejecutar sobre este hilo.

Existen principalmente dos tipos de servicios según el grado de conocimiento que tiene el usuario acerca de la existencia de estos:

1. **Servicios *background***
2. **Servicios *foreground***

Los servicios en ***background*** son básicamente servicios que realizan operaciones de fondo sin que el usuario tenga conocimiento de ello, es decir, no recibe ningún tipo de ***feedback*** por

parte de la aplicación de que se están realizando tareas en segundo plano. Como cabe esperar, esta **opacidad** da lugar a que el sistema limite la ejecución de estos servicios, suspendiendo las tareas en segundo plano cuando considere necesario. Este comportamiento forma parte de las restricciones impuestas a partir de cierta versión de *Android*, lo cual se explicará en las siguientes secciones.

Por otro lado, los servicios ***foreground*** sí proporcionan información al usuario acerca de su existencia. A diferencia de los anteriores, la aplicación muestra obligatoriamente una **notificación** permanente asociada al ciclo de vida del servicio, de forma que el usuario es **consciente** de que hay algún servicio o trabajo pendiente ejecutándose en segundo plano. A modo de ejemplo, un servicio *foreground* típico sería el propio reproductor de música, que tras cerrar la aplicación permanece activo en forma de notificación y que contiene los controles de pausa, avance y retroceso del audio. Esta **transparencia** le proporciona a este tipo de servicios ciertas ventajas en cuanto al tiempo de vida que el sistema les otorga, pues como el usuario está siendo notificado en todo momento acerca de este trabajo de fondo, podría considerarse como una operación en **primer plano** y por tanto será menos probable que el sistema cancele o suspenda este tipo de trabajos por falta de memoria o bien por ahorrar consumo de batería.

3.2.3.2 *Receivers*

Otra de las *API's* que proporciona el sistema operativo de *Android* para la ejecución en segundo plano son lo que se denominan **transmisiones** o ***Broadcast Receivers* [17]**, basados en el patrón ***publish-subscribe*** y que sirven para notificar a las aplicaciones acerca de determinados eventos.

Estos eventos pueden ser intrínsecos del sistema operativo, como, por ejemplo, el evento de conectar el dispositivo a la corriente, o bien, un evento que se dispara una vez completado el encendido del dispositivo. Los eventos también pueden ser personalizados, es decir, se pueden especificar eventos propios por medio de acciones, para notificar a las aplicaciones sobre eventos propios.

Las aplicaciones registran los *broadcast*s correspondientes de tal forma que estos estén escuchando activamente a la espera de ser notificados cuando se dispare un evento determinado. Para escuchar un evento, es necesario **suscribirse** a él, del mismo modo que se haría con el patrón *publish-subscribe*. De este modo, el sistema se encarga de enrutar los eventos a los diferentes *broadcasts* registrados que estén suscritos. Se trata de una manera de intercambiar mensajes ya sea dentro de una sola aplicación, o bien entre **varias aplicaciones** independientes.

3.2.3.3 *Alarmas*

El último componente para la ejecución de tareas en segundo plano que será de utilidad en la aplicación móvil de rastreo es el ***Alarm Manager* [18]** que proporciona *Android*. Mediante esta herramienta es posible programar alarmas en el tiempo, es decir, se trata de una forma de ejecutar tareas basadas en la temporalidad, concretamente en un *timestamp* determinado.

Para entender su funcionamiento, se puede pensar en el propio sistema de alarmas que proporciona *Android*, ya que sigue los mismos principios. Permite programar una alarma a una hora determinada, de tal forma que se dispare cuando el reloj del sistema marque dicha hora.

Sin embargo, la alarma no consiste en un elemento sonoro, sino que se trata de código ejecutable, concretamente de alguno de los componentes ya vistos anteriormente, como, por ejemplo, un *broadcast receiver* o bien un servicio en primer plano. De este modo, se pueden ejecutar tareas en segundo plano y además programadas en el tiempo.

El *Alarm Manager* de *Android* permite utilizar diversos métodos para establecer las alarmas, en función de la **exactitud** y la **precisión** que se desea que tengan. Hay métodos para establecer alarmas inexactas, es decir, que no se garantiza que se ejecuten exactamente a la hora prevista, sino que *Android* se encarga de priorizar otros procesos que estén en ejecución y ordenar su ejecución en el tiempo junto con las alarmas, con el objetivo de reducir el consumo de batería. Por otro lado, se puede sacrificar batería para obtener una mayor precisión, asegurándose de que las alarmas programadas se ejecutan en las horas planificadas.

De manera adicional, también es posible controlar la **periodicidad**, es decir, cada cuánto tiempo se repiten las alarmas, como, por ejemplo, una vez al día.

El sistema de alarmas es algo externo a las aplicaciones, por lo que funciona incluso cuando la aplicación está cerrada, aunque en función del método que se utilice, si el dispositivo se encuentra en *Doze Mode*, lo más probable es que las alarmas no se disparen. Para que las alarmas sigan funcionando incluso cuando el dispositivo está "dormido" es necesario utilizar otros métodos del *Alarm Manager* más exhaustivos y que consumen una mayor cantidad de batería.

3.2.4 Versiones: restricciones y limitaciones

Cuando se desarrollan aplicaciones para *Android*, uno de los aspectos clave a tener en cuenta es la **diversidad** de **versiones** de este sistema operativo. Con cada nueva actualización que lanza *Google,* se incluyen mejoras, nuevas características y también **restricciones** en el funcionamiento de ciertos componentes de *Android*, lo cual puede limitar la funcionalidad del *software* desarrollado o incluso hacer que deje de funcionar.

La parte cliente de *Android* que se va a desarrollar, depende de tecnologías como la geolocalización o la ejecución en segundo plano, que son dos de las características que han ido sufriendo cambios a lo largo de las distintas versiones de *Android*. Como consecuencia de estos cambios, se han ido aplicando restricciones de diseño, rendimiento y ejecución, entre otras, que es necesario tener en cuenta cuando se está desarrollando aplicaciones para *Android* que trabajan con localización y ejecución en *background.*

En esta sección, se presenta una descripción de las restricciones, cambios y características incluidas en aquellas versiones de *Android* que más impacto tienen en el desarrollo y que se consideran relevantes de cara al diseño e implementación del sistema.

3.2.4.1 *Estrategias de ahorro de energía*

A partir de la versión ***Android* 6.0 (*Marshmallow*)**, que se corresponde con la ***API 23***, se introducen dos nuevas estrategias para tratar de reducir el consumo de batería: el **modo *Doze*** y el ***App StandBy* [19]**. Ambas estrategias, afectarán en mayor o menor medida a las características funcionales de la aplicación cliente de *Android*, concretamente a los servicios de

localización y a la ejecución en segundo plano. A continuación, se explican estas dos estrategias en detalle.

3.2.4.1.1 Doze Mode

Esta estrategia consiste en una característica de *Android* que busca reducir al máximo posible el consumo de batería, **cancelando** y **pausando** la ejecución de las aplicaciones, cuando transcurre cierto periodo de tiempo durante el cual el dispositivo **no** está **enchufado a la corriente** y además se encuentra **estacionario**.

Concretamente, restringe la ejecución de aquellos servicios que hagan **uso intensivo de la *CPU*** o bien accedan a **servicios de red**, como, por ejemplo, descargar archivos de *internet*. Además, en este modo, el sistema operativo también pausa cualquier trabajo, tarea, alarma o sincronización que se esté llevando en curso en las aplicaciones, y vuelve a reanudar estas operaciones cuando se da alguna de las siguientes situaciones:

- El usuario activa la pantalla del dispositivo.
- El dispositivo se enchufa a la corriente.
- El usuario mueve el dispositivo (deja de estar estacionario).
- El sistema entra en una **ventana de mantenimiento**.

Las tres primeras situaciones se desencadenan de manera natural cuando el usuario realiza alguna acción con el dispositivo. Sin embargo, la cuarta opción, no es visible al usuario ya que es una cuestión interna relativa al sistema operativo, se trata de la **ventana de mantenimiento**.

La ventana de mantenimiento consiste en un **corto periodo de tiempo** durante el cual se **"despierta"** al sistema del modo *Doze* para terminar de completar las tareas o trabajos pendientes de las aplicaciones. Durante este periodo, se permite a las aplicaciones acceder a la red y al uso de *CPU*. El tiempo que dura esta ventana no es algo fijo y además depende de cada modelo de dispositivo. En la Figura 3.2 se puede ver una representación visual del *Doze Mode* sacada directamente de la documentación de *Android* **[19]**.

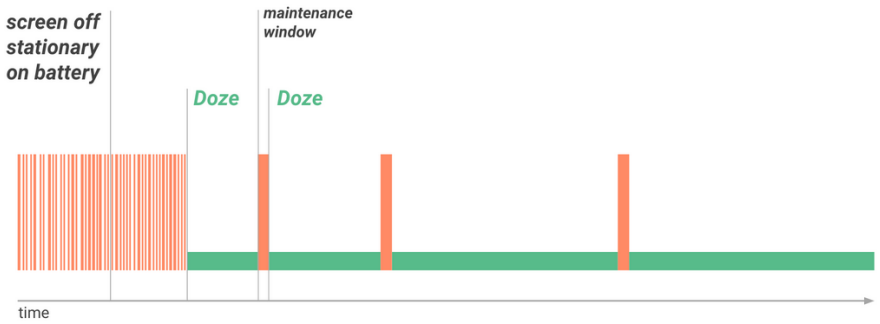


Figura 3.2. Representación del Doze Mode en Android

Como se puede observar, el sistema espera un periodo de tiempo antes de entrar en *Doze Mode*. Una vez dentro, se van sucediendo varias ventanas de mantenimiento durante las cuales se reanudan las operaciones. Cabe destacar, que el **intervalo de tiempo** entre una ventana y otra se va haciendo cada vez más grande con el tiempo. Esto tiene sentido ya que a

mayor tiempo de inactividad menor dedicación se debería de dar a las tareas y trabajos de las aplicaciones.

Esta es una enumeración de las principales restricciones que impone el sistema operativo mientras está en *Doze Mode*:

1. Se suspenden las operaciones de acceso a la red.
2. El sistema ignora los *wake locks*, es decir, solicitudes por parte de las aplicaciones para mantener al dispositivo despierto.
3. No se escanean puntos de acceso *Wi-Fi.*
4. Se suspenden las alarmas y los trabajos en segundo plano.

3.2.4.1.2 App Stand By

Por otro lado, está el modo *App Stand By*, menos restrictivo que el *Doze Mode*, ya que en este caso se aplica a una aplicación concreta y no a todas las aplicaciones.

El sistema determina qué aplicaciones están inactivas y suspende temporalmente sus operaciones tanto de *CPU* como de acceso a la red, de forma que sólo se les permite completar sus trabajos pendientes una vez al día aproximadamente.

Una vez que el dispositivo se enchufa a la corriente, se liberan todas aquellas aplicaciones que estuvieran en *Stand By*, permitiéndoles de nuevo ejecutar sus operaciones.

El criterio que adopta el sistema operativo para determinar qué aplicaciones están inactivas se basa en el tiempo de inactividad pero también en si dicha aplicación tiene algún componente en **primer plano**, recibe **notificaciones en la pantalla de bloqueo** o bien se trata de una **aplicación** de **administración**.

3.2.4.2 *Restricciones de ejecución en segundo plano*

Dado que gran parte de la funcionalidad principal de la aplicación móvil de *Android* se basa en operaciones en **segundo plano**, es decir, mientras la aplicación no está visible al usuario o bien está destruida, es importante tener en cuenta los **cambios** que se introdujeron con la versión ***Android 8.0 (Oreo),*** correspondiente a la ***API 26***.

Hasta la versión *Oreo*, las aplicaciones tenían un alto grado de independencia en relación con lo que podían hacer mientras se encontraban en segundo plano, de forma que apenas había restricciones que limitasen su ejecución con la intención de reducir el consumo. Sin embargo, esto cambió con el lanzamiento de *Android Oreo*, que impuso nuevas **restricciones [20]** a las aplicaciones que tenían servicios ejecutándose en segundo plano, para tratar de reducir el consumo y también la **carga** del sistema, que en algunos casos podía ser muy elevada, causando el cierre repentino de algunas aplicaciones para liberar recursos.

De la versión *Oreo* en adelante, todas las aplicaciones que tengan servicios en segundo plano que no sean de tipo ***foreground*** o bien no estén visibles para el usuario, son consideradas por el sistema operativo como aplicaciones ***background.*** Estas aplicaciones reciben un periodo de tiempo, que varía en función de la carga del sistema en ese momento dado y de cada modelo, y durante el cual se les permite continuar normalmente con sus operaciones de fondo. Tras transcurrir este periodo, el sistema pasa estas aplicaciones a estado **inactivo** y por tanto se **suspenden** las operaciones en segundo plano.

Esto supone una **limitación** respecto a **qué alternativas** de ejecución en segundo plano **utilizar**, dado que para la aplicación cliente de *Android* es necesario realizar diversas tareas y operaciones de fondo sin que estas sean **interrumpidas** o **canceladas**.

De manera adicional, a partir de ***Android 9 (Pie)***, es necesario solicitar un nuevo tipo de permiso para poder ejecutar servicios en primer plano. Este permiso es concedido automáticamente por el sistema, luego no es necesario que el usuario proporcione explícitamente este permiso.

3.2.4.3 *Localización en segundo plano*

La última de las restricciones que se va a tratar, tiene que ver con los propios **servicios** de **geolocalización** de *Android* y con la **tasa de actualización** de localizaciones del dispositivo **[21]**. Esta restricción también es derivada de los cambios introducidos en la ***API 26*** de ***Android Oreo*** relativos a la ejecución en segundo plano.

En *Oreo* y versiones superiores, el sistema operativo **limita** la **frecuencia de actualización** de la **localización** del usuario, es decir, restringe el número de veces que se recibe información sobre la posición del usuario en aquellas aplicaciones que estén funcionando en segundo plano. Estas restricciones se aplican a ambas **abstracciones de localización** mencionadas en las secciones anteriores, es decir, tanto la *API* de ***FusedLocationProvider*** como el ***framework* de localización de *Android*** se ven afectados por estas limitaciones.

De este modo, solo se recibirán actualizaciones de posición **frecuentes** y **continuas** si las aplicaciones están en primer plano, de lo contrario, solo se obtendrán unas **pocas actualizaciones** por **hora**. En nuestro caso, es conveniente que el registro de ubicación sea consistente y constante, de forma que no sufra interrupciones inesperadas, ya que la **calidad** de los **rastreos** está fuertemente ligada a la precisión de los **itinerarios** que se registren en el dispositivo de los usuarios.

Para evitar estas restricciones, *Google* recomienda utilizar **servicios en primer plano** los cuales implican mostrar una notificación que permanece visible durante toda la vida del servicio. Solo de esta forma se garantizan actualizaciones de ubicación frecuentes y constantes.

Por último, es necesario tener en cuenta otra restricción relacionada con los **permisos** de las aplicaciones para acceder a la **ubicación** del usuario en segundo plano, ya no impuesta por *Android Oreo*, sino por ***Android 10***, correspondiente a la ***API 29***. A partir de esta *API*, cualquier aplicación que vaya a acceder a servicios de ubicación mientras está en segundo plano, debe solicitar los permisos correspondientes al usuario para que este conceda dicho privilegio a la aplicación. De no ser así, la ubicación no funcionará en segundo plano. En contraposición, en versiones anteriores a la de *Android 10*, cualquier aplicación puede utilizar servicios de localización ya sea en primer o segundo plano.

3.3 Herramientas y plataformas utilizadas

En esta sección, se enumeran y describen brevemente las distintas herramientas y plataformas utilizadas en la implementación del sistema, así como sus pruebas y despliegue. Como el objetivo no es realizar una guía sobre cómo usarlas, se muestra para cada una de ellas un enlace al sitio oficial para obtener más información.

3.3.1 Firebase/Firestore

Firebase **[22]** es una plataforma en la nube ideada por *Google* para proporcionar diversos servicios relacionados con la tecnología *web* y *móvil*. Integrar *Firebase* con un proyecto de desarrollo *web* o *móvil* aporta un gran abanico de servicios y funcionalidades, como envío de mensajes, analíticas, servicios de *testing*, almacenamiento en la nube, autenticación…*etc*.

Para este proyecto se han utilizado únicamente dos servicios de esta plataforma. Por un lado, la base de datos en la nube ***Firestore*** para el almacenamiento de datos en un *backend* independiente gestionado por *Firebase*, y por otro, el servicio de envío de mensajes a los clientes móviles, el cual se explicará en otro apartado.

La base de datos *Firestore* permite almacenar datos desde una aplicación *web* o aplicación móvil siempre y cuando se vincule con *Firebase*. Se trata además de una base de ***datos* documental**, es decir, sigue el enfoque ***NoSQL***, lo cual hace que sea muy fácil de utilizar y muy eficiente en las operaciones.

3.3.2 Node.js

Node.js **[23]** es un entorno de ejecución para aplicaciones escritas en *Javascript*, tal y como se indica en su página oficial, construido con el motor *V8* de *Google*. Su principal utilidad es escribir código *javascript* en el servidor, de manera que se pueda disponer de un servidor *web* en el *backend* ejecutándose con *javascript*. Una de sus particularidades, es el modelo de concurrencia a través de un **único hilo**, es decir, modela la ejecución asíncrona de tareas simultáneas mediante un solo hilo. Esto lo consigue mediante el ***Event Loop***, un bucle que va procesando y despachando los eventos que recibe desde la cola de eventos.

Esta tecnología se ha utilizado para desarrollar el servidor *web* de *Contact Tracker*, junto con el *framework Express.js* que se comentará a continuación.

3.3.3 Express.js

Express.js **[24]** es un *framework* para el desarrollo *web* pensado para ser utilizado junto con *Node.js* y así construir un *backend* soportado por *Node.js* y con el enrutamiento definido mediante *Express.js*. Permite definir verbos *HTTP* con diferentes rutas para gestionar las peticiones entrantes enviadas por los clientes, incluyendo la gestión de parámetros y datos enviados en el cuerpo de la petición.

3.3.4 Vue.js

Para el desarrollo del *frontend web* se hace uso del *framework Vue* **[25]**, el cual contiene una serie de herramientas y una arquitectura ya definida para facilitar y agilizar el proceso de construcción de una página *web*. Se trata de un marco de trabajo para *javascript* el cual se basa en la idea de **componentes**. Un componente es una pieza o fragmento de una página *web* que encapsula la **infraestructura *html***, la funcionalidad escrita en un ***script*** y los estilos ***css*** de los elementos *html*, todo ello en el mismo sitio. Esto proporciona una mayor **reutilización** y **encapsulación** del código, lo cual hace que sea muy sencillo reutilizar componentes en otras partes de la aplicación.

3.3.5 Android Studio

La aplicación móvil se desarrolla empleando el propio *framework* de *Android*, para lo cual se utiliza la herramienta *Android Studio* **[26]** proporcionada por *Google*. Esta herramienta gratuita constituye el entorno de desarrollo ideal para aplicaciones *Android*, ya que contiene todo lo necesario para implementar una aplicación con *Java* o *Kotlin* que funcione en un dispositivo *Android*.

A través de este *IDE*, se puede acceder a las múltiples herramientas y *APIs* del *framework* de *Android*, incluyendo además un panel para el diseño de interfaces *Android* siguiendo el estándar de *Material Design*, el cual permite definir *layouts* a través del lenguaje de marcado *XML*.

3.3.6 ROOM

ROOM **[27]** es una librería de persistencia mantenida por *Google* que proporciona una capa de abstracción sobre la clásica base de datos *SQLite* integrada en los dispositivos *Android*. Se trata de un ***ORM***, es decir, un *Object Relational Mapper* que mapea los objetos del dominio representados como clases planas en el código y los transforma a **tablas** correspondientes a las entidades junto con sus relaciones en la base de datos *SQLite*.

Esta herramienta simplifica considerablemente el desarrollo ya que permite ignorar todo el código *boilerplate* necesario para establecer la conexión con la base de datos local y construir las consultas. Mediante *ROOM* se define de manera declarativa el **esquema** de la base de datos mediante **anotaciones** en las clases planas del dominio.

3.3.7 Retrofit y Axios

Retrofit **[28]** es una librería **cliente** de ***HTTP*** diseñada para consumir servicios *web* mediante su *URL* y distintos verbos *http* para *Android*. Superpone una capa de abstracción por encima de las peticiones de red para convertir el código *boilerplate* en simples anotaciones que permiten implementar un cliente *http* de manera declarativa. Mediante *Retrofit* se define una **interfaz** con las diferentes peticiones de red indicado el verbo, la *URL* de localización del recurso, así como los parámetros y el cuerpo, en caso de que existan. Además, por debajo transforma las

respuestas *JSON* obtenidas de las peticiones en los objetos del dominio definidos como clases planas. Para controlar la asincronía de las llamadas en el contexto del proyecto, *Retrofit* se integra junto con las **corrrutinas** de *kotlin*, explicadas más adelante.

Por otro lado, *Axios* **[29]** es una librería de *Javascript* para *node.js* que también cumple la función de cliente *HTTP* para poder realizar peticiones a una *API* desplegada en un servicio *web* desde un cliente *javascript*. *Axios* permite gestionar las peticiones de manera **asíncrona** a través de ***callbacks*** de éxito y de error, que se ejecutan una vez se recibe la respuesta del servicio *web*.

3.3.8 FCM

FCM o por sus siglas, *Firebase Cloud Messaging* **[30]**, es un servicio proporcionado por la plataforma de *Firebase,* comentada anteriormente, para el envío de mensajes *downstream* entre distintos clientes, ya sean aplicaciones *web* o móviles integradas con esta plataforma. En el contexto del proyecto, se hace uso de esta tecnología para enviar las notificaciones a los dispositivos que constituyen los clientes *Android*. De este modo, se pueden configurar y personalizar los mensajes, cambiando el título, cuerpo, *payload* o carga útil, color, sonido… entre otras características, además de especificar los dispositivos concretos a los que enviar dichos mensajes o bien enviarlos a grupos de dispositivos a través de **tópicos**. Los dispositivos que estén suscritos a un tópico concreto recibirán notificaciones relacionadas con ese tópico, siguiendo el típico patrón *publish-subscribe*.

3.3.9 Google Maps

Para poder dibujar mapas físicos dentro de las aplicaciones *web* y móvil, se hace uso de la *API* de *Google Maps* **[31]**. Esta *API* constituye uno de los muchos servicios dentro de la plataforma de ***Google Cloud***, la cual gestiona toda la parte de facturación y utilización de los servicios por las aplicaciones clientes que estén integradas. La aplicación móvil y la aplicación *web* utilizan esta *API* a través de la **clave** proporcionada, una vez configurada, junto con el *SDK* correspondiente de *Google Maps*.

El *SDK* contiene funcionalidades para dibujar líneas, marcadores u otras figuras encima de los mapas, además de la posibilidad de cambiar las capas de los mismos.

3.3.10 Mocha

Mocha **[32]** es un *framework* de *Javascript* para el desarrollo de pruebas en aplicaciones *web* basadas en *Node.js*. Permite el desarrollo de pruebas de manera asíncrona, así como la obtención de informes detallados sobre los resultados de las pruebas. Se utiliza conjuntamente con la librería de aserciones *Chai*, comentada en la siguiente sección. Las pruebas escritas a través de *Mocha* se ejecutan sobre el propio navegador.

3.3.11 Chai y Chai-http

Las pruebas de la *API REST* se implementan a través del *framework Mocha*, descrito anteriormente, junto con las librerías de aserciones *Chai* **[33]** y *Chai-http* **[34]**. Estas librerías permiten escribir pruebas de una manera más declarativa, a través de las cláusulas *should*, *expect* o *assert* para definir aserciones sobre los resultados esperados y los obtenidos. De manera adicional, el *plugin Chai-http* para *Chai* permite definir estos asertos para los resultados de las peticiones *http*.

De este modo, mediante *chai-http* se puede simular la interacción entre los clientes con la *API REST,* lo cual constituye un *mock http* que simula las peticiones con los distintos verbos *http*. Así, las respuestas generadas por la *API REST* se pueden verificar mediante las aserciones proporcionadas por esta librería.

3.3.12 Espresso

Las pruebas de interfaz de usuario para la aplicación móvil están escritas mediante el *framework Espresso* **[35]**, desarrollado por *Google*, el cual permite realizar aserciones sobre componentes de la interfaz, identificándolos en la jerarquía a través de diversos métodos. Para ello, *Espresso* proporciona ***ViewMatchers***, los cuales permiten localizar elementos de la interfaz en función de diversas condiciones (*id*, clase del componente, texto contenido…), ***ViewActions***, que permiten ejecutar acciones sobre los elementos localizados, y por último, ***ViewAssertions***, para realizar aserciones sobre el estado o el contenido de dichos elementos.

La *API* proporcionada por *Espresso* es además personalizable y extensible, de tal manera que se pueden definir *matchers* y acciones personalizadas para cada caso concreto. Las pruebas de *Espresso* son instrumentadas, es decir, se ejecutan directamente sobre un dispositivo móvil o un emulador dentro de *Android Studio.*

3.3.13 JUnit

JUnit **[36]** es el *framework* por excelencia para implementar y ejecutar pruebas unitarias de componentes escritos en *Java*. Las pruebas unitarias para los componentes de la aplicación móvil se implementan utilizando *JUnit*, que también da soporte al lenguaje *Kotlin*. *JUnit* permite definir clases de prueba utilizando las cláusulas *Before* y *After* para ejecutar código antes y después de cada método de prueba etiquetado con *Test,* además de poder agrupar clases de prueba para crear *suites* de prueba que se ejecuten al mismo tiempo. Las pruebas unitarias implementadas con *JUnit* se ejecutan sobre la máquina virtual de *Java*.

3.3.14 Mockito

La librería *Mockito* **[37]** define un marco de pruebas para *Java* que proporciona utilidades para ***mockear*** componentes en las pruebas. Como funciona sobre *Java*, también es compatible con el *framework* de *Android* y, por consiguiente, con *Kotlin,* además de que se integra a la perfección con el *framewok* de *JUnit*. *Mockito* permite definir **dobles** de prueba para simular el

comportamiento de clases y métodos durante la ejecución de las pruebas, concretamente se pueden definir lo que se conoce como ***stubs***, es decir, incrustar datos falsos a la hora de invocar ciertos métodos. También permite controlar y verificar las **llamadas** a los métodos, pudiendo realizar aserciones sobre los parámetros con los que se invocan dichos métodos.

3.3.15 Kotlin

Kotlin **[38]** es un lenguaje de programación de tipado estático desarrollado por *JetBrains* que se ejecuta sobre una máquina virtual de *Java.* Esto lo transforma en una alternativa al lenguaje *Java* a la hora de desarrollar aplicaciones para *Android*. *Kotlin* es mucho más flexible y eficiente que *Java,* además de proporcionar utilidades como las corrutinas, y *azúcares sintácticos* equivalentes a los que se pueden encontrar en lenguajes de la *web* como *Javascript*. Tiene soporte para el paradigma de programación orientada objetos, además del clásico paradigma estructurado por procedimientos. También incluye aspectos propios de la **programación funcional** pudiendo utilizar funciones *lambda*.

La principal característica de este lenguaje es la comprobación estática de los tipos nulables, es decir, ninguna variable puede contener una referencia vacía (*null*) a menos que se especifica explícitamente. Esto se comprueba en tiempo de compilación, de tal forma que si una variable no nulable, no ha sido inicializada con un valor esto se traducirá en un error de compilación. Este enfoque hace que se reduzcan considerablemente los clásicos errores de *NullPointerException* muy vistos en *Java.*

3.3.16 Corrutinas

Las corrutinas de *Kotlin* **[39]** son un equivalente a los hilos en *Java*, con la diferencia de que estas son una evolución respecto de los clásicos hilos de ejecución. Las corrutinas permiten ejecutar código **asíncrono** en diferentes *scopes* que se ejecutan por debajo sobre distintos hilos. Esto facilita la implementación de la concurrencia siguiendo un enfoque más declarativo. Las corrutinas se utilizan junto con las funciones ***suspend*** de *kotlin*, las cuales representan funciones asíncronas que tras terminar su trabajo devuelve un valor, pero sin detener la ejecución del resto de líneas de código. Estas funciones especiales dan la sensación de estar implementando código síncrono, lo cual simplifica bastante el desarrollo, aunque por debajo, *Kotlin* gestiona los hilos necesarios para ejecutar el código y devolver el resultado en el hilo correspondiente.

3.3.17 DataBinding y ViewBinding

La vinculación de datos o *data binding* **[40]** es una utilidad opcional que proporciona el *framework* de *Android* para simplificar la población de las vistas de la aplicación *Android*, definidas en los *XML* correspondientes. Permite acceder a los datos desde los propios *layouts XML*, de forma que toda la lógica para enviar los datos a la vista desaparece, pues esta reside en los propios elementos *XML* que representan los componentes de la interfaz.

Por otro lado, también se encuentra la vinculación de vistas o *view binding* **[41]**, que sigue un enfoque similar con las propias vistas en lugar de con los datos, de forma que los componentes

de la interfaz son accesibles directamente desde el código de la aplicación. Esto simplifica la lógica para localizar los componentes de la interfaz y acceder a sus datos, pues estos están disponibles directamente desde el código.

Activando ambas características en el *framework* de *Android*, se estaría usando lo que se conoce como ***two-way binding***, donde se vinculan tanto los accesos a los componentes de la interfaz como los datos desde dichos componentes.

3.3.18 Jetpack Navigation

Jetpack Navigation **[42]** es una librería para el *framework* de *Android,* desarrollada por *Google* con el objetivo de facilitar la lógica para implementar la navegación entre pantallas en una aplicación *Android.* Esta librería proporciona una herramienta para definir **grafos de navegación** entre los fragmentos de la aplicación de manera visual en lugar de implementarlo por medio de código. Esto hace que sea muy sencillo y visual definir las acciones para moverse entre las pantallas de la aplicación móvil, ahorrando numerosas líneas de código. También proporciona el **paso de argumentos** de un fragmento a otro con comprobación segura de tipos, utilizando la librería *Safe Args*.

3.3.19 Azure DevOps

Azure DevOps **[43]** una plataforma de *Microsoft* para dar soporte a las operaciones de *DevOps* de un proyecto. Contiene gran cantidad de funcionalidades relacionadas con las metodologías ágiles, como un tablero *Kanban*, pilas de producto, división del proyecto en iteraciones o *sprints*, *story mapping*… Además, también contiene funcionalidades típicas de despliegue e integración continua que serán descritas en la siguiente sección.

En el contexto del proyecto, esta plataforma se utiliza para gestionar la planificación del mismo y las historias de usuario que definen lo que debe hacer el sistema, organizándolas a través de las pilas de producto y el *story mapping*. También presenta un *dashboard* para mostrar gráficas visuales sobre la evolución de las iteraciones, mediante *burndown charts* o gráficos de quemado.

3.3.20 CI/CD

La plataforma de *Azure DevOps* también aporta funcionalidad para el despliegue y la integración continua, lo que se denomina CI/CD **[44]**, integrando estas operaciones con los repositorios *git* utilizados para albergar el código. De este modo, cada vez que se realiza un *commit* en la rama *master* se dispara el sistema de ***pipelines*** de *Azure* para empaquetar una nueva versión del sistema y desplegarla automáticamente en los servicios de *Azure*. Para el caso de la aplicación *web*, se utilizan los **contenedores estáticos** que permiten albergar los recursos de una aplicación *web* en la nube, mientras que el servidor *web* se despliega utilizando el ***Azure App Service***, que permite hospedar aplicaciones *Node.js*.

De este modo, *Azure* proporciona *pipelines* para empaquetar el código y generar lo que se denomina un ***artifact*** que estará disponible para ser desplegado a través de otro tipo de

pipelines, las cuales se encargan de realizar el despliegue como tal de los artefactos en un determinado entorno.

3.4 Otros conceptos

Tras describir las distintas herramientas y tecnologías involucradas en el desarrollo del proyecto, en esta sección se describen otros conceptos teóricos relacionados con el proyecto y su entorno.

3.4.1 MVVM

Las siglas *MVVM* **[45]** se corresponden con la arquitectura *Model-View-ViewModel*, muy utilizada en los desarrollos de aplicaciones móviles. Se trata de la arquitectura recomendada por *Google* para estructurar las aplicaciones *Android* de forma mantenible y reutilizable, respetando los principios de separación de responsabilidades. Esta arquitectura será descrita en detalle en el capítulo dedicado al diseño del sistema, a alto nivel, consiste en dividir el sistema en componentes que conforman el modelo, las vistas y unos elementos especiales denominados ***view models***, los cuales mantienen los datos que son mostrados en las vistas. De este modo, las vistas son notificadas con cada actualización o cambio de estado que sufren los datos mantenidos en el *viewmodel*. Como se puede apreciar, el núcleo de esta arquitectura se basa en el patrón *Observer,* que es implementado a través de los componentes que se denominan ***LiveData.*** Estos son los elementos observables que cada vez que cambia su estado se notifica a los clientes que los observan.

3.4.2 API REST

Una *API REST* **[46]** consiste en un tipo de arquitectura *web* basada en la transferencia del **estado representacional** de los recursos, es decir, cuando se solicita un recurso por medio de una petición *http* se devuelve una representación del estado de este recurso solicitado en formato *JSON* u otros derivados. El servidor *web* del sistema sigue este tipo de arquitectura para exponer una serie de rutas, denominadas *endpoints,* sobre las cuales pueden hacer peticiones los clientes a través de los distintos verbos *http* (*GET, POST…*). El servidor responde con objetos *JSON* que representan el estado del recurso solicitado.

3.4.3 Metodología ágil: Scrum

Las metodologías ágiles son las sucesoras de las metodologías tradicionales, pues cada vez son más utilizadas en los desarrollos donde se requiere *software* funcionando rápidamente. El enfoque ágil se describe a través de lo que se denomina el **manifiesto ágil [47]**, el cual expone una serie de aspectos que representan los principios fundamentales del desarrollo ágil:

- Individuos e interacciones sobre procesos y herramientas.
- *Software* que funciona sobre documentación extensiva.
- Colaborar con el cliente antes que una negociación contractual.

- Responder ante el cambio en lugar del seguimiento de un plan.

Scrum **[48]** es un marco de trabajo para las metodologías ágiles que proporciona una manera de estructurar el proceso de desarrollo en *sprints* con una serie de reuniones en las que participa el equipo de desarrollo, el dueño del producto y en ocasiones el *scrum master*, persona encargada de garantizar que se cumple con los principios de *Scrum*. Como se puede ver, en *Scrum* se distinguen distintos roles, el dueño del producto, el *product owner* y el equipo de desarrollo. Todos ellos trabajan en conjunto para garantizar la máxima calidad del producto.

Una de las peculiaridades de *Scrum* es trabajar con la cercanía del cliente del producto, lo cual facilita las tareas de análisis y recopilación de requisitos, ya que se dispone del conocimiento del negocio cuando sea necesario, a diferencia de las metodologías tradicionales donde apenas se colabora con el cliente. Además, al dividir el desarrollo en iteraciones que se van planificando sobre la marcha, se reduce el riesgo de obtener un producto con el que el cliente no esté satisfecho, pues este revisa al final de cada *sprint* lo que se denomina el **incremento del producto**.

En el contexto de este proyecto, se utiliza *Scrum* como marco de trabajo para orientar el desarrollo del sistema a lo largo de diversos *sprints* y colaborar con el dueño del producto, que en este caso se corresponde con el propio tutor del proyecto.

Capítulo 4. Estudio legal

En este capítulo, se pretende llevar a cabo un análisis de todos los **aspectos legales** que tienen influencia sobre el sistema a desarrollar, situando en un contexto legal al proceso de desarrollo. Aunque existe mucha normativa legal relacionada con los sistemas informáticos, en este caso concreto solo tiene sentido centrar el análisis en la **protección de datos personales**, ya que para alcanzar los objetivos definidos es necesario tratar una serie de datos personales que se describirán más adelante.

4.1 Contexto de la protección de datos

La protección de datos incluye todos los aspectos legales relacionados con el tratamiento y gestión de datos personales por parte de un encargado o responsable de los datos en un sistema u organización. Los **datos personales** son cualquier tipo de datos o información que pueda ser asociada a una persona física y que por tanto permita identificarla o atribuirle ciertas características.

La trayectoria de la protección de datos en España, en lo que respecta a los sistemas informáticos, ha ido evolucionando a lo largo del tiempo, desde la primera mención de la informática en el **artículo 18.4** de la **Constitución española de 1978** hasta la última **Ley Orgánica 3/2018 de Protección de Datos Personales y Garantía de los Derechos Digitales,** también conocida como ***LOPD/GDD*** **[3]**. Durante esos años se fueron sucediendo leyes y reglamentos que implementaban directrices y normativas para la protección de datos en sistemas automatizados y no automatizados.

Atendiendo a esta trayectoria, podemos distinguir entre **dos leyes orgánicas** fundamentales relativas a la protección de los datos en España:

1. **Ley Orgánica de Protección de Datos de 1999.** Se trata de la antigua normativa aplicada a la protección de datos en España, la cual entró en vigor en 1999 como resultado de la **transposición** de la **Directiva Europea de Protección de Datos de 1995**.
2. **Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales de 2018.** Es la sucesora de la antigua *LOPD* de 1999, la cual hoy en día está obsoleta. Esta nueva ley se trata de una **especialización** del **Reglamento General de Protección de Datos (RGPD) [4],** puesto en vigor en 2016, para España e incluye ciertas mejoras y ampliaciones respecto a la ley orgánica de 1999. Este reglamento es de ámbito europeo, y a diferencia de una directiva, no se puede transponer, sino que se trata de una normativa **común** de **aplicación directa** a todos los **estados miembros** de la **Unión Europea.** En cambio, propone una serie de aspectos *personalizables* por cada país, como, por ejemplo, definir la edad mínima a partir de la cual un menor puede dar su consentimiento para el tratamiento de los datos, que en el caso de España son 14 años.

La LOPD/GDD de 2018 permanece vigente hoy en día, por lo que será la que se tome como referencia para diseñar la privacidad de los datos del sistema en construcción. Como ya se mencionó, la nueva LOPD se trata de la **versión adaptada** a **España** del **reglamento europeo** sobre la protección de datos. Existen múltiples diferencias entre la antigua LOPD y la nueva, aunque no es el objetivo de este trabajo describirlas, sí que es interesante comentar las más significativas y enumerar las novedades que propone la nueva *LOPD/GDD*. Antes de pasar a ver estas novedades, se explican brevemente los conceptos más importantes relativos a la protección de datos.

4.1.1 Conceptos sobre la protección de datos

Para poder entender los aspectos tratados en este capítulo es necesario comprender previamente una serie de **conceptos vitales** sobre la protección y el tratamiento de los datos personales, ya que estos conceptos y terminología irán apareciendo a lo largo de todo el capítulo.

Datos personales / Ficheros de datos

Los datos personales son cualquier información que permita identificar de forma directa o indirecta a un individuo, es decir, son datos que se pueden asociar a las personas físicas.

Para que se aplique todo lo relacionado con la protección de datos, es necesario que estos datos personales estén **organizados** en **ficheros de datos** y tengan una **finalidad específica**. El concepto de **fichero** es un **concepto lógico** y se debe asociar necesariamente a un fichero de texto físico (el concepto de fichero de datos se corresponde a la antigua LOPD de 1999 y ya no se utiliza en la nueva ley orgánica de 2018). Además, estos datos personales reciben un **tratamiento** en base a la finalidad para la que se destinan.

Tratamiento de los datos

El tratamiento de los datos incluye todas las actividades y operaciones que se realizan sobre los datos personales organizados en los ficheros de datos. El simple hecho de almacenar los datos personales en el sistema ya se considera como tratamiento de datos.

Interesado

Cuando se habla de ***interesado*** se está haciendo referencia a la persona que es **titular de los datos,** es decir, se trata de la persona específica a la cual corresponden dichos datos.

Responsable del tratamiento

El responsable del tratamiento tiene **facultad** para **decidir** sobre el tratamiento de los datos personales y puede ser una entidad, persona física o jurídica. Es quien proporciona las **instrucciones** para llevar a cabo el tratamiento de los datos y por tanto será quien reciba las multas en aquellos casos en los que se infrinja alguno de los artículos de la ley.

Encargado del tratamiento

A diferencia del responsable, es quien realmente lleva a cabo y **ejecuta** el tratamiento de los datos y por tanto responde ante las instrucciones del responsable. Puede ser una entidad o persona interna o externa al responsable del tratamiento. En algunos casos, el responsable del tratamiento y el encargado son la misma persona.

Agencia de Protección de Datos

También conocida como *APD*, se trata del organismo u entidad responsable de velar por la protección de datos en un sector determinado. Proporciona las instrucciones y directrices sobre el tratamiento y la protección de datos, además de imponer las sanciones y realizar informes sectoriales sobre el estado de la protección de datos en un sector y posibles acciones de mejora.

4.1.2 Novedades de la nueva LOPD

La puesta en vigor de la nueva *LOPD* en 2018 resultado de la especialización del RGPD europeo introdujo ciertas novedades, algunas de las cuales se describen a continuación.

Responsabilidad proactiva

Con la nueva LOPD se propuso llevar a cabo una buena protección de datos ya desde los inicios de la construcción de un sistema, lo cual se conoce como **privacidad desde el diseño**, es decir, contemplar la seguridad de los datos y los riesgos potenciales desde las primeras etapas de la construcción. Además, también se contempla la **privacidad por defecto**, lo cual significa establecer la configuración de protección de datos con la máxima seguridad por defecto, y que sea el propio usuario quien tenga la potestad para cambiar dicha configuración.

De manera adicional, se establece una **limitación de la finalidad**, es decir, se trata de ajustar el tratamiento de los datos al mínimo posible para poder realizar las operaciones necesarias, de tal forma que así se **minimicen los daños** a los interesados.

Análisis de riesgos del tratamiento

En la antigua LOPD, se utilizaba un **Reglamento de Medidas de Seguridad (2008)** de obligado cumplimiento, el cual recogía las directrices y medidas de seguridad que debía seguir toda empresa para garantizar la conformidad con lo establecido en la ley orgánica de 1999. Sin embargo, con la nueva LOPD, ya no existe ningún reglamento que especifique las medidas de seguridad que se tengan que implementar, sino que es necesario llevar a cabo un **análisis de riesgos** del tratamiento que se le quiere dar a los datos y a partir de dicho análisis decidir cuáles son las medidas de seguridad oportunas.

Estas medidas de seguridad serán diferentes en función del ámbito de cada sistema y del tratamiento que se le dé a los datos, teniendo en cuenta el nivel de riesgo de dicho tratamiento.

Multas elevadas

En la LOPD de 1999 las multas por violar o incumplir alguno de los aspectos relativos a la protección de datos eran exactamente iguales para cualquier organización u empresa. Sin embargo, con la nueva LOPD la cuantía económica de estas multas viene determinada por el **volumen de negocio** de la organización y por tanto pueden alcanzar cifras muy elevadas de millones de euros.

Consentimiento positivo

Para llevar a cabo el tratamiento de datos de forma lícita, antiguamente el **consentimiento del interesado** se conseguía mediante el **consentimiento táctico**, es decir, si el interesado no

manifestaba su negación explícitamente ante el tratamiento de los datos, entonces se consideraba que el interesado daba su consentimiento.

Con la nueva LOPD llegó lo que se denomina **consentimiento positivo u *opt-in***, es decir, solo se permite llevar a cabo el tratamiento si el propio interesado da su **consentimiento explícitamente** con una afirmación.

Registro interno del tratamiento de datos

Se trata de un registro interno que tiene que proporcionar cualquier empresa u organización que lleve a cabo el tratamiento de datos personales y en el cual se debe recoger una **descripción** de los **datos personales** que están siendo tratados, así como una explicación del **tratamiento** que se le da a dichos datos. Es el equivalente al antiguo **registro de ficheros de datos** en la **AGPD (Agencia General de Protección de Datos)** que se exigía realizar con la LOPD de 1999.

4.1.3 Ámbito de aplicación

Como ya se dijo anteriormente, la nueva LOPD de 2018 proviene de una adaptación especializada en España del ***RGPD*** europeo, lo cual significa que es una normativa de aplicación directa y que es común a todos los estados miembros de la UE. Por ello, la LOPD/GDD de 2018 es aplicable a España, en aquellos aspectos adaptables a cada país como es por ejemplo la edad mínima de un menor para dar consentimiento. Sin embargo, al ser una correspondencia directa con el RGPD, se aplica también en todos los **estados miembros** de la **UE** a todas aquellas **personas físicas** que estén **vivas**.

Además, la RGPD también es aplicable en un **ámbito extra-europeo**, es decir, si los destinatarios de los servicios son de origen europeo, entonces también se aplican los aspectos contemplados en este reglamento, aunque la empresa u organización responsable de los datos no pertenezca a la Unión Europea.

4.2 Principios de la protección de datos

Una vez se ha visto el contexto general de la protección de datos en Europa, y concretamente en España, con la nueva LOPD/GDD de 2018, en esta sección se describen los **principios** fundamentales relativos a la protección de datos.

4.2.1 Licitud del tratamiento

La licitud del tratamiento hace referencia a la **validez legal** del tratamiento realizado sobre los datos, es decir, se refiere a la legitimidad del tratamiento. Dicha licitud del tratamiento puede conseguirse a través de **dos vías**:

- **Consentimiento del interesado:** se trata de la vía más segura para obtener la licitud del tratamiento. Consiste en obtener el **consentimiento** del interesado de manera explícita, voluntaria y demostrable para poder llevar a cabo el tratamiento. El consentimiento ha de ser **libre** y **demostrable,** es decir, el responsable debe ser capaz

de demostrar la veracidad del consentimiento, bien de forma manual mediante una firma, o de manera automática con un certificado digital u otras alternativas. En el caso de España, solo los mayores de 14 años podrán dar su consentimiento para el tratamiento de sus datos personales. Además, previamente a solicitar el consentimiento del interesado, es necesario indicar los **fines** del **tratamiento**, así como el **responsable** del mismo, incluyendo otros datos que se describen más adelante y que constituyen la **Política de Privacidad** del sistema.

- **Necesidad del tratamiento:** existen casos en los que la licitud del tratamiento está **implícita** y por tanto no es necesario obtener el consentimiento del interesado de forma explícita, como, por ejemplo, para poder ejecutar un contrato, cerrar una factura… Otras veces existe una **obligación legal** por parte del responsable que le permite llevar a cabo directamente dicho tratamiento. Otro caso es cuando el tratamiento de los datos se realiza en ámbito de **interés público,** es decir, de la **sociedad**, y vela por la protección de los **intereses vitales** de las personas.

4.2.2 Calidad y seguridad de los datos

Este principio hace referencia a que los datos deben tener un **mínimo de calidad**, es decir, los datos que están siendo tratados tiene que ser **correctos, exactos, actualizados** y **adecuados** con los objetivos del tratamiento. Además de la calidad de los datos, también se debe garantizar la **seguridad** de los datos, lo cual se conoce como **deber de secreto**, es decir, los datos personales deben permanecer en secreto y no pueden ser divulgados.

La calidad y seguridad de los datos se encuentra ligada a los principios ***CID*** que se describen a continuación:

- **Confidencialidad (C):** se corresponde con el deber de secreto, es decir, los datos deben ser confidenciales, solo aquellas personas autorizadas tienen derecho a acceder a los datos.
- **Integridad (I):** los datos deben ser correctos y conformes con la realidad, es decir, no pueden existir inconsistencias entre los datos y estos deben representar correctamente la realidad del interesado.
- **Disponibilidad (D):** los datos tienen que estar disponibles siempre que sean necesarios.

4.2.3 Deber de información

Según este principio, es necesario mantener una **transparencia total** a la hora de llevar a cabo el tratamiento de los datos, informando con el máximo nivel de detalle al interesado sobre todos los aspectos relacionados con la protección de datos dentro del sistema.

Esta información se proporciona mediante **cláusulas informativas** de protección de datos, las cuales constituyen la **política de privacidad** que debe ser accesible por el interesado. Previamente a ejecutar el tratamiento de los datos, se debe informar explícitamente al interesado sobre la intención de tratar sus datos personales y otros detalles como el autor responsable del tratamiento…etc.

4.2.4 Derechos de los interesados

Los titulares de los datos personales, es decir, los interesados, están amparados por una serie de **derechos** que les atribuyen ciertos privilegios en cuanto a la protección de sus datos. Originalmente, estos eran los denominados **derechos ARCO (*Acceso, Rectificación, Cancelación* y *Oposición*)**, aunque con la introducción del nuevo reglamento europeo y por tanto de la ley orgánica de protección de datos del 2018, estos derechos se ampliaron incluyendo algunos **nuevos derechos** como son el derecho al **olvido** y el derecho de **portabilidad** de los datos. A continuación, se describe brevemente cada uno de estos derechos ARCO, incluyendo el derecho al olvido y a la portabilidad incluidos en la nueva ley orgánica.

4.2.4.1 *Derecho de Acceso*

El derecho de acceso hace referencia al derecho que tienen los interesados para conocer **qué datos** tiene almacenados la empresa u organización o bien están siendo tratados en la aplicación. El interesado tiene el derecho de **preguntar** al **responsable** del tratamiento acerca de si sus datos personales están siendo tratados, y en dicho caso **solicitar** información sobre estos.

De acuerdo con la AEPD (*Agencia Española de Protección de Datos*), el interesado puede solicitar diversos datos al responsable, algunos de los cuales se muestran a continuación:

- Descripción de los fines del tratamiento.
- Copia de los datos personales que están siendo tratados.
- Plazo previsto de conservación de los datos, durante el cual se mantendrán dichos datos en el sistema.
- Origen de los datos personales en aquellos casos en los que no hayan sido obtenidos directamente del interesado.

Esto es solo un fragmento de la información que puede ser solicitada por el interesado, pero existen otros tipos de información que puede ser solicitada y que puede consultarse en la página oficial de la AEPD **[49]**.

4.2.4.2 *Derecho de Rectificación*

En aquellos casos en los que los datos personales del interesado sean **inexactos** y disconformes con la realidad, es decir, no representan correctamente el estado real del interesado, el derecho de rectificación permite a los interesados **solicitar** al responsable del tratamiento la rectificación de aquellos datos incorrectos o incompletos.

Esta rectificación consiste en **corregir** los datos desactualizados, inexactos o incompletos a través de una solicitud enviada directamente por el interesado afectado.

4.2.4.3 *Derecho de Cancelación*

El derecho de cancelación dota a los interesados de la capacidad de solicitar la **eliminación** de sus datos personales del sistema en el que se encuentran almacenados. Ante dicha petición, el

responsable del tratamiento deberá eliminar cualquier dato personal que esté albergado en el sistema.

4.2.4.4 *Derecho de Oposición*

El derecho de oposición consiste en que el interesado se **opone** al tratamiento de sus datos personales, por lo que la funcionalidad dependiente de dichos datos personales se queda **bloqueada.** Esto implica que los datos personales **no** serán **eliminados** del sistema, pero tampoco serán procesados para los fines previstos.

4.2.4.5 *Derecho al Olvido*

El derecho al olvido es uno de los nuevos derechos introducidos con la nueva LOPD de 2018 y también es conocido como el derecho de **supresión**. Se trata de una variante del derecho de cancelación en la que además de eliminar los datos personales del sistema, también deben ser eliminados todos los **enlaces** y **copias de respaldo** asociadas a dichos datos personales, así como cualquier copia almacenada en la *caché* o los motores de búsqueda.

De este modo, se borra el ***pasado*** del usuario de los confines de *internet* sin quedar rastro de su información.

4.2.4.6 *Derecho a la portabilidad*

Este derecho también es una de las novedades introducidas con el RGPD europeo y se trata de una **ampliación** del derecho de **acceso**, el cual dotaba a los interesados de la capacidad de solicitar información sobre sus datos personales sometidos a tratamiento.

En este caso, el interesado no solo tiene derecho a solicitar qué datos están siendo procesados, sino que además tiene el derecho de solicitar sus datos personales en un **formato estructurado** y **portable** que pueda ser procesado por otros sistemas, como por ejemplo *json* o *xml*.

4.2.5 Prohibición de cesión de los datos

La cesión de los datos personales de los interesados a **terceros** es considerado uno de los peores delitos en el ámbito de la protección de datos. De acuerdo con lo establecido en la LOPD/GDD de 2018, no está permitido trasladar los datos personales a una entidad o persona de terceros para que los **procese** sin el consentimiento del interesado y además para **otros fines**.

4.3 Política de privacidad

En las anteriores secciones, se dio una visión general del contexto legislativo sobre el cual está cimentado el desarrollo del sistema. Esta sección tiene como objetivo detallar las características básicas sobre **política de privacidad** que toda aplicación ***Android*** que trate con datos personales debería disponer.

De acuerdo con la política de *Google Play*, cualquier aplicación *Android* que procese datos sensibles de carácter personal, ya sea de forma directa porque sean proporcionados por el propio usuario o bien de manera indirecta a través de las herramientas del dispositivo (micrófono, *gps*…), debe ser **transparente** en cuanto a cómo trata dichos datos, es decir, tiene la obligación de **divulgar** a los usuarios los procedimientos y características del tratamiento empleado mediante **cláusulas informativas**, las cuales deben ser accesibles desde la propia aplicación.

4.3.1 Contenido de la cláusula informativa

En el contexto de las aplicaciones *Android*, para que el sistema que procesa los datos sea conforme con la nueva LOPD de 2018 y por tanto con el RGPD europeo, es necesario que aporte una o varias cláusulas informativas que incluyan como mínimo los siguientes datos:

- Información sobre el **responsable** del tratamiento.
- Datos de **contacto** del responsable.
- Información sobre el **dueño** de la aplicación móvil.
- Información sobre **qué** datos están siendo tratados.
- Información relativa al **origen** de los datos.
- Descripción de la **finalidad** del tratamiento.
- Información sobre los **destinatarios** de terceros que reciben los datos.
- Información sobre **transferencias internacionales** de los datos, en caso de que hubiera.
- Derechos **ARCO** de los interesados.
- **Plazo** de **conservación** durante el cual permanecerán los datos personales en el sistema.

Existen otros aspectos importantes que se pueden incluir en la política de privacidad, pero los enumerados anteriormente son los más destacables.

4.4 Descripción de los datos a tratar

En esta última sección dedicada al estudio legal se pretende realizar una breve **descripción** de los datos de los usuarios que van a ser procesados en el propio sistema en desarrollo.

Se trata de una aplicación basada mayoritariamente en la **geolocalización** de los usuarios, por lo tanto, la normativa y legislación en ámbito de protección de datos es de aplicación en este contexto.

Mediante un primer análisis, podemos clasificar los datos que van a ser tratados en **dos grupos:**

1. **Datos de geolocalización**
2. **Datos personales**

4.4.1 Datos de geolocalización

Los datos de geolocalización están constituidos por las **ubicaciones** de los usuarios y son proporcionadas por el sensor *GPS* del dispositivo móvil. Estas ubicaciones son datos sensibles de los usuarios y por ello pueden ser considerados datos personales que se pueden asociar con personas físicas. Sin embargo, existen algunos casos en los que dichas ubicaciones se almacenan de manera **anónima**, es decir, sin asociarse con ningún dato personal de una persona física. En esos casos, podría contemplarse que los datos sobre las localizaciones de los usuarios no fueran realmente datos personales, sino simples coordenadas en un mapa que no estuvieran asociadas a un usuario.

Aunque las ubicaciones se almacenen de manera anónima, es importante notificar igualmente a los usuarios de que se está **registrando** su posición actual, así como solicitar los **permisos** necesarios de localización al usuario para poder rastrear su ubicación, tal y como se explicó en la sección 3.2.2 GEOLOCALIZACIÓN EN ANDROID.

Para las ubicaciones de los usuarios se almacenan principalmente los siguientes datos:

- Coordenadas: Latitud y Longitud.
- Fecha y hora en la que se registraron las coordenadas.
- Proveedor utilizado (Ver sección 3.2.2.1 PROVEEDORES DE LOCALIZACIÓN EN ANDROID).
- Precisión (Ver sección 3.2.2.1.5 PRECISIÓN DE LAS COORDENADAS).

4.4.2 Datos personales

Además de las ubicaciones, la aplicación *Android* permitirá a los usuarios la opción de proporcionar sus **datos personales** de manera **voluntaria**, de forma que se contribuya a facilitar las tareas de rastreo de contagios de COVID-19. En este caso, si el usuario proporciona sus datos personales, estos quedarán **asociados** a las ubicaciones que se han ido registrando en el dispositivo, de forma que las localizaciones que antes eran anónimas ahora estarán ligadas a una persona física y por tanto se aplicarán los términos legales establecidos en la LOPD de 2018.

Es importante solicitar el **consentimiento** del interesado previamente a la petición de datos personales para asociarlos con unas coordenadas, de otra forma, se estaría incumpliendo uno de los principios fundamentales de la protección de datos: la **licitud del tratamiento**.

A continuación, se enumeran los principales datos personales que se recogen de los usuarios en la aplicación móvil:

- Nombre y apellidos.
- DNI.
- Número de teléfono.
- Localidad.
- Código postal.

Una vez introducidos estos datos, quedarán asociados a los **itinerarios** registrados por el dispositivo del usuario y serán subidos a la *nube*, de forma que estos datos serán visibles por las autoridades sanitarias.

Capítulo 5. Planificación del Proyecto y Resumen de Presupuestos

El objetivo de este capítulo es presentar una idea general de la planificación realizada para ejecutar el proyecto *Contact Tracker*, incluyendo una estructura de desglose de tareas agrupadas en etapas y su correspondiente diagrama de *Gantt*. Para complementar este desglose de tareas, también se proporciona un diagrama WBS/PBS que muestra las tareas de alto nivel junto con sus productos derivados.

De manera adicional, también se presenta un resumen del presupuesto calculado para el proyecto.

5.1 Planificación

Como se comentó en el capítulo 3 sobre aspectos teóricos, el desarrollo del proyecto está guiado por una **metodología ágil**, concretamente ***Scrum,*** con lo cual no es posible realizar una planificación exhaustiva como la que se lleva a cabo en los proyectos que siguen una metodología tradicional (por ejemplo, *Métrica V3*). Las tareas a realizar se planifican al inicio de cada **iteración,** dando lugar al ***Sprint Backlog*** que contiene las funcionalidades que se pretenden desarrollar dentro del *sprint* para entregar un incremento funcional del producto al cliente.

Sin embargo, es posible realizar una pequeña planificación global del proyecto que involucre las principales etapas de forma general, pero sin entrar en detalles específicos sobre las tareas que se realizarán en cada iteración. En esta sección, se presentan las diferentes etapas en las que se ha decidido estructurar el proyecto mediante una **EDT (Estructura de Desglose de Tareas)** junto con su **diagrama de Gantt** correspondiente, además de una descripción de los recursos y el calendario de trabajo del proyecto. Además, también se incluye un diagrama ***WBS/PBS*** global del proyecto y se explica el papel de la planificación dentro de la metodología ágil.

5.1.1 Etapas del proyecto

El proyecto *Contact Tracker* se estructura en una serie de **etapas** compuestas por diferentes **tareas**, lo cual da lugar a la denominada **EDT** o **Estructura de Desglose de Tareas**. Estas etapas son:

1. *Etapa de Investigación*
2. *Etapa de Preparación*
3. *Etapa de Análisis y Diseño preliminar*
4. *Etapa de implementación*

5. *Etapa de implantación del sistema*

Antes de pasar a describir las tareas a realizar en cada una de estas etapas, se muestra la EDT del proyecto junto con su diagrama de *Gantt* asociado.

5.1.1.1 *EDT y diagrama de Gantt*

La EDT global del proyecto contiene todas las tareas definidas del proyecto organizadas en las distintas etapas vistas anteriormente. Para definir la EDT global del proyecto se ha utilizado el programa ***MS Project***, el cual se encarga de generar el diagrama de *Gantt* correspondiente a la EDT definida. Se ha marcado como **fecha de comienzo** del proyecto el **25 de enero de 2021,** dejando que el *MS Project* calcule automáticamente la fecha de fin en base a las estimaciones de las tareas.

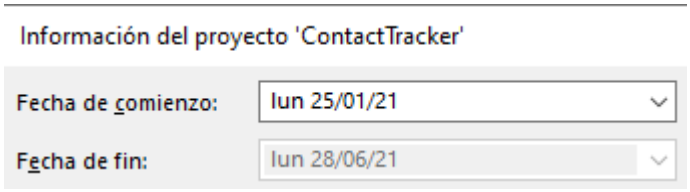


Ilustración 5.1 Fecha de comienzo y de fin del proyecto.

En base a la duración de las tareas, el calendario de trabajo y las relaciones de precedencia entre tareas, *MS Project* estima que el proyecto **concluye** el **lunes 28 de junio de 2021.**

A continuación, se muestra la EDT global del proyecto con todas las tareas definidas.

DIAGRAMA DE GANTT

	Modo de tarea	Número de esquema	Predecesoras	Nombre de tarea	Duración	Comienzo	Fin
1		**1**		**Etapa de Investigación**	**61 horas**	**lun 25/01/21**	**lun 15/02/21**
2		1.1		Estudio de Geolocalización en Android.	15 horas	lun 25/01/21	sáb 30/01/21
3		1.2	2	Estudio de ejecución en 2o plano y consumo de batería en Android.	15 horas	sáb 30/01/21	vie 05/02/21
4		1.3	3	Estudio de las versiones Android: limitaciones y restricciones de geolocalización y ejecución en 2o plano.	10 horas	vie 05/02/21	dom 07/02/21
5		1.4	4	Estudio de tecnologías y herramientas a utilizar (ROOM, MVVM, Firebase, Azure Dev Ops, Kotlin, AlarmManager, Corrutinas…).	15 horas	lun 08/02/21	sáb 13/02/21
6		1.5	5	Estudio de antecedentes y sistema anterior.	2 horas	sáb 13/02/21	dom 14/02/21
7		1.6	5	Estudio de la situación actual y sistemas existentes (RadarCovid, HealthKit).	4 horas	sáb 13/02/21	dom 14/02/21
8		1.7	5	Estudio de la normativa, aspectos legales y protección de datos.	6 horas	sáb 13/02/21	lun 15/02/21
9		1.8	5	Estudio del contexto sanitario: aspectos teóricos sobre la pandemia de COVID19.	2 horas	sáb 13/02/21	dom 14/02/21
10		**2**	**1**	**Etapa de Preparación del Entorno**	**24 horas**	**lun 15/02/21**	**jue 25/02/21**
11		2.1		Creación de los repositorios GIT.	2 horas	lun 15/02/21	lun 15/02/21
12		2.2	11	Creación y configuración del proyecto para la aplicación cliente Android.	4 horas	lun 15/02/21	jue 18/02/21
13		2.3	12	Creación y configuración del proyecto para la aplicación backend del servidor web (API REST).	3 horas	jue 18/02/21	vie 19/02/21
14		2.4	13	Creación y configuración del proyecto para la aplicación frontend (App Web).	3 horas	vie 19/02/21	sáb 20/02/21
15		2.5	14	Creación y configuración del proyecto en Firebase (base de datos documental en la nube).	3 horas	sáb 20/02/21	sáb 20/02/21
16		2.6	15	Creación y configuración del proyecto en Google Cloud Console para usar la API de Google Maps.	1 hora	dom 21/02/21	dom 21/02/21
17		2.7	16	Creación del proyecto en Azure Dev Ops (tablero kanban, storymap).	3 horas	dom 21/02/21	dom 21/02/21
18		2.8	17	Configuración de la integración continua (CI) y despliegue (CD) con las pipelines de Azure DevOps.	5 horas	lun 22/02/21	jue 25/02/21
19		**3**	**10**	**Etapa de Análisis y Diseño inicial**	**15 horas**	**jue 25/02/21**	**dom 28/02/21**
20		3.1		Análisis a alto nivel del alcance y objetivos del sistema.	3 horas	jue 25/02/21	jue 25/02/21
21		3.2	20	Descripción a alto nivel de las principales funcionalidades del sistema.	4 horas	vie 26/02/21	vie 26/02/21
22		3.3	21	Realización de Prototipos, Sketchs y bocetos iniciales de la arquitectura del sistema.	5 horas	sáb 27/02/21	dom 28/02/21
23		3.4	22	Diseño preliminar del modelo de datos.	3 horas	dom 28/02/21	dom 28/02/21
24		**4**	**19**	**Etapa de Implementación**	**320 horas**	**lun 01/03/21**	**dom 20/06/21**
25		4.1		Sprint 1	80 horas	lun 01/03/21	dom 28/03/21
26		4.2	25	Sprint 2	80 horas	lun 29/03/21	dom 25/04/21
27		4.3	26	Sprint 3	80 horas	lun 26/04/21	dom 23/05/21
28		4.4	27	Sprint 4	80 horas	lun 24/05/21	dom 20/06/21
29		5		Redacción de la documentación	444 horas	lun 25/01/21	lun 28/06/21
30		**6**	**24**	**Etapa de Implantación del Sistema**	**15 horas**	**lun 21/06/21**	**sáb 26/06/21**
31		6.1		Desplegar API REST en el servidor web de Azure.	1 hora	lun 21/06/21	lun 21/06/21
32		6.2	31	Desplegar Aplicación Web en el contenedor de Azure.	1 hora	lun 21/06/21	lun 21/06/21
33		6.3	32	Publicar APK de la aplicación Android en Google Play Store.	3 horas	lun 21/06/21	jue 24/06/21
34		6.4	33	Formar al personal sanitario para el uso del panel de control web.	10 horas	jue 24/06/21	sáb 26/06/21
35		**7**		**Hitos**	**175 horas**	**dom 25/04/21**	**sáb 26/06/21**
36		7.1	26	Versión estable de la aplicación móvil Android.	0 horas	dom 25/04/21	dom 25/04/21
37		7.2	28	Versión estable de la API REST.	0 horas	dom 20/06/21	dom 20/06/21
38		7.3	28	Versión estable de la aplicación web para el personal sanitario.	0 horas	dom 20/06/21	dom 20/06/21
39		7.4	30	Sistema funcionando en Producción.	0 horas	sáb 26/06/21	sáb 26/06/21

Ilustración 5.2 Estructura de Desglose de Tareas global para el proyecto Contact Tracker.

En la ilustración superior, se muestran las diferentes etapas en las que se divide el proyecto, junto con las tareas que las componen, incluyendo las relaciones de precedencia, la duración y la fecha de inicio y fin de cada tarea. La duración de cada tarea ha sido estimada en horas teniendo en cuenta el esfuerzo necesario para realizarla.

Debajo de la etapa de implementación, se puede observar que hay una tarea denominada ***Redacción de la documentación*** con una duración de 444 horas, debido a que esta tarea engloba todo el ciclo de vida del proyecto, desde su comienzo hasta su fin. Además, en las últimas filas, se pueden apreciar los **hitos** del proyecto que tienen una duración de 0 horas, ya que representan **momentos en el tiempo** destacados en los que se producen las **entregas parciales** al cliente. Estos hitos se describen a continuación:

1. Versión estable de la aplicación móvil *Android:* este hito será alcanzado una vez finalizado el *Sprint 2* ya que su predecesora es la tarea 26 (véase la relación de precedencia entre las tareas).
2. Versión estable de la API REST: se alcanza al finalizar el *Sprint 4,* pues su tarea predecesora es la 28.
3. Versión estable de la aplicación *web* para el personal sanitario: se alcanza tras finalizar el *Sprint 4*, su tarea predecesora es la 28.

4. Sistema funcionando en Producción: la aplicación móvil se ha publicado en la Play Store de Google y la *API REST* se encuentra desplegada junto con la aplicación *web* para el personal sanitario. Además, el personal sanitario ha recibido la formación necesaria para poder utilizar el panel de control *web*.

Una posible representación visual de esta EDT es el **diagrama de *Gantt*** que se genera en *MS Project*, el cual puede verse a continuación.

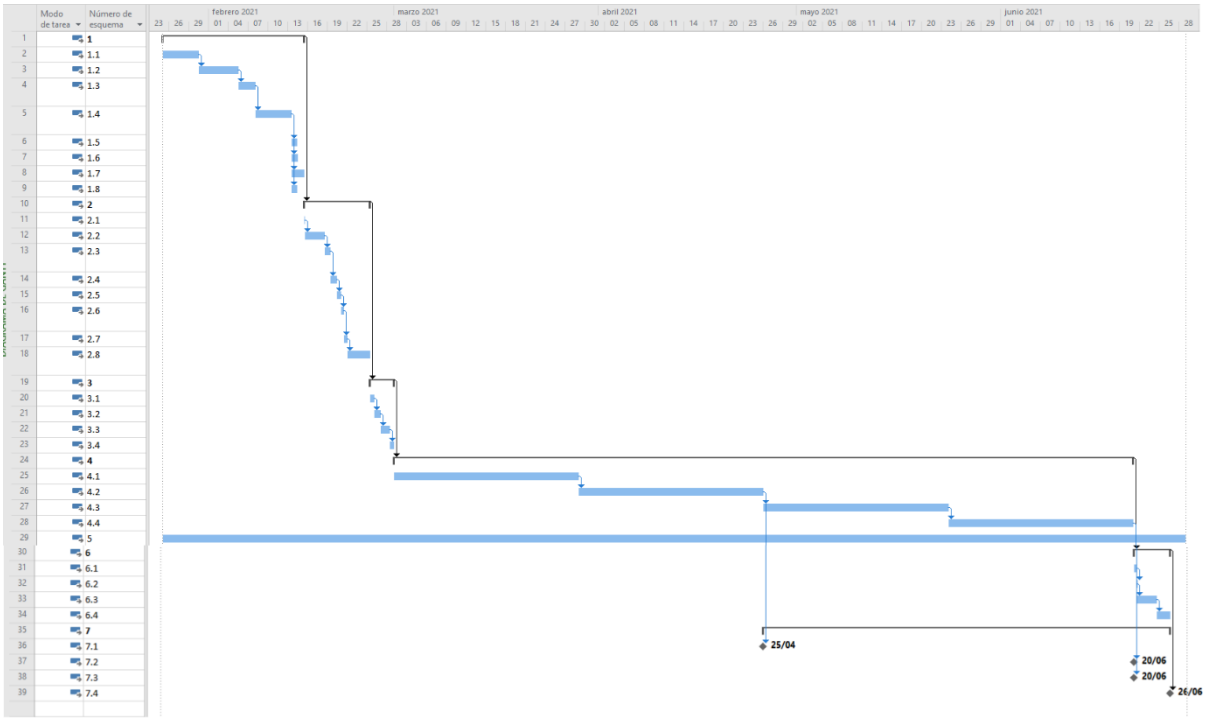


Ilustración 5.3 Diagrama de Gantt de la EDT global del proyecto.

Como se puede observar, las iteraciones del desarrollo se llevan a cabo de forma secuencial. Además, la tarea de redacción de la documentación (número de esquema 5) engloba todo el ciclo de vida del proyecto, pues la documentación se comienza a redactar desde el primer día.

Una vez se ha terminado la fase de implementación, se inicia la fase de implantación del sistema, donde se pretende formar al personal sanitario en el uso del sistema y desplegar los módulos implementados para que el sistema comience a funcionar en producción.

Por último, cabe destacar que los **rombos** que aparecen en el diagrama representan los **hitos** mencionados anteriormente, los cuales son alcanzados al final del *Sprint 2* (*25/04*) y del *Sprint 4* (*20/06*) respectivamente.

5.1.1.2 *Etapa de investigación*

N.º de esquema en *Project:* **1**
Duración: **61 horas**
Comienzo: 25/01/2021 Fin: 15/02/2021

En esta primera etapa se han planificado todas las tareas de investigación para recabar toda la información posible que será necesaria a la hora de implementar el sistema, como, por ejemplo, estudiar cómo funciona la geolocalización en *Android*, la ejecución en 2º plano…

incluyendo otras tareas de investigación como el estudio de antecedentes y de la situación actual, aspectos legales…

5.1.1.3 *Etapa de Preparación*

N.º de esquema en *Project:* **2**
Duración: **24 horas**
Comienzo: 15/02/2021 Fin: 25/02/2021

Consiste en todas las tareas de preparación de los entornos de desarrollo, repositorios, bases de datos en la nube, plataformas como *Firebase*, *Google Cloud Console (GCC)* y *Azure DevOps,* incluyendo la infraestructura para el despliegue del sistema mediante los *pipelines* de *Azure DevOps*.

5.1.1.4 *Etapa de Análisis y diseño preliminar*

N.º de esquema en *Project:* **3**
Duración: **15 horas**
Comienzo: 25/02/2021 Fin: 28/02/2021

En esta etapa se pretende realizar un análisis de alto nivel del sistema que se pretende construir, identificando el alcance del sistema y los principales objetivos perseguidos, además de determinar las funcionalidades principales que debe proporcionar el sistema. También se persigue definir una primera versión del modelo de datos y de la arquitectura general del sistema.

5.1.1.5 *Etapa de implementación*

N.º de esquema en *Project:* **4**
Duración: **320 horas**
Comienzo: 01/03/2021 Fin: 20/06/2021

Finalmente, se encuentra la etapa de implementación que engloba las cuatro iteraciones/*sprints* que se estima serán necesarios para finalizar la implementación del sistema. De acuerdo con lo establecido en el marco de trabajo *Scrum*, las iteraciones o *sprints* tienen una duración de aproximadamente **4 semanas**, por lo que teniendo en cuenta el calendario de trabajo que define unas 20 horas de trabajo semanales, la duración en horas de estos *sprints* sería de **80 horas**, aunque esto se trata de una **aproximación teórica** que no se asemeja con la realidad, pues en la metodología ágil no tiene sentido estimar previamente la duración de los *sprints*, sino que se estima las horas de trabajo de cada *item* del *Product Backlog* que es seleccionado para ser implementado en una iteración concreta.

Se ha optado por incluir los *sprints* dentro de la EDT para poder obtener una representación visual de lo que sería el ciclo de vida del proyecto en su totalidad, aunque realmente **no** es del todo **correcto** dentro del marco de las metodologías ágiles.

La siguiente ilustración refleja la etapa de implementación con sus iteraciones, cada una con su fecha de inicio y de fin estimadas, incluyendo además los hitos definidos en el apartado 5.1.1.1 EDT Y DIAGRAMA DE GANTT.

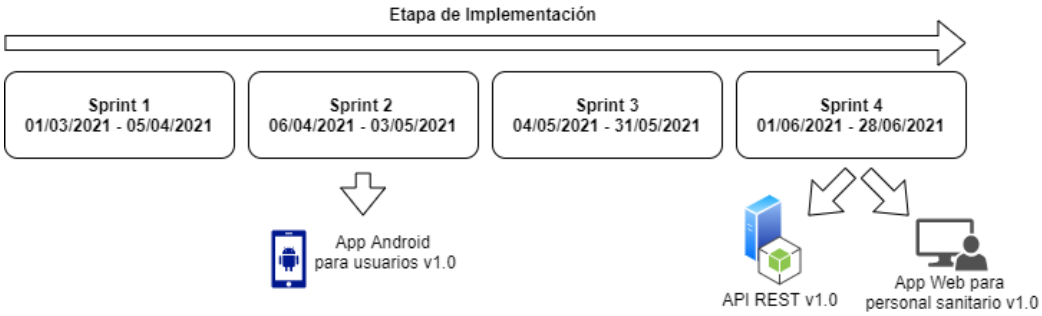


Ilustración 5.4 Estructura de la etapa de implementación junto con los hitos a alcanzar.

5.1.1.6 *Etapa de Implantación del sistema*

N.º de esquema en *Project:* **6**
Duración: **15 horas**
Comienzo: 21/06/2021 Fin: 26/06/2021

Esta etapa se inicia una vez finalizada la implementación del sistema en su totalidad y tiene como objetivo poner el sistema en **producción**, es decir, poner en marcha el funcionamiento del sistema. Las tareas que componen esta etapa engloban el despliegue e implantación de los distintos módulos resultantes de la anterior etapa, lo cual consiste en desplegar la *API REST* en el servicio de *Azure*, desplegar la aplicación *web* sobre el contenedor de *Azure*, publicar el *APK* de la aplicación móvil de *Android* en *Google Play Store* e impartir la formación requerida al personal necesario acerca de cómo utilizar el panel de control *web*.

Una vez finalizada esta etapa, se completa el hito correspondiente a conseguir implantar el sistema funcionando en producción.

5.1.2 Recursos y calendario de trabajo

Dado que se trata de un proyecto realizado por una sola persona, no existe un equipo de trabajo formado por diferentes roles como puede ser el de jefe de proyecto, analista, arquitecto de *software*…etc, por lo tanto, solo existirá un **recurso de trabajo** que representa todos los roles que puedan identificarse dentro de un equipo de desarrollo.

Dentro de los distintos tipos de recursos de un proyecto podemos distinguir principalmente los siguientes que serán utilizados en el desarrollo del sistema:

- Recursos de trabajo
  - Un solo trabajador (alumno) que toma los roles de ingeniero de *software*, analista, arquitecto, jefe de proyecto, ingeniero de pruebas, *scrum master*…
- Recursos de coste (aquellos que se pueden amortizar a lo largo de varios proyectos)
  - Ordenador de sobremesa con dos monitores, teclado y ratón.
  - Licencia de *Windows 10 Pro*.
  - Dispositivo móvil con sistema operativo *Android.*

- Recursos materiales (aquellos que se consumen durante la vida del proyecto)
    - API *Key* para utilizar *Google Maps* en el proyecto.
    - Licencia gratuita de *Firebase* para hospedar la base de datos documental en la nube.
    - Licencia gratuita de *Azure DevOps* para la gestión del proyecto (tablero *Kanban*, *storymap*…) y la configuración de la integración y despliegue continuo.

En cuanto al **calendario de trabajo** de los recursos humanos, se ha definido una jornada laboral para el único recurso de trabajo existente:

- De jueves a lunes, de **09:00 h** a **13:00 h** (5 días/semana, 20 horas/semana).
- Martes y miércoles no laborables.

A continuación, se muestra una captura del programa *MS Project* donde se puede ver la configuración del calendario del proyecto.

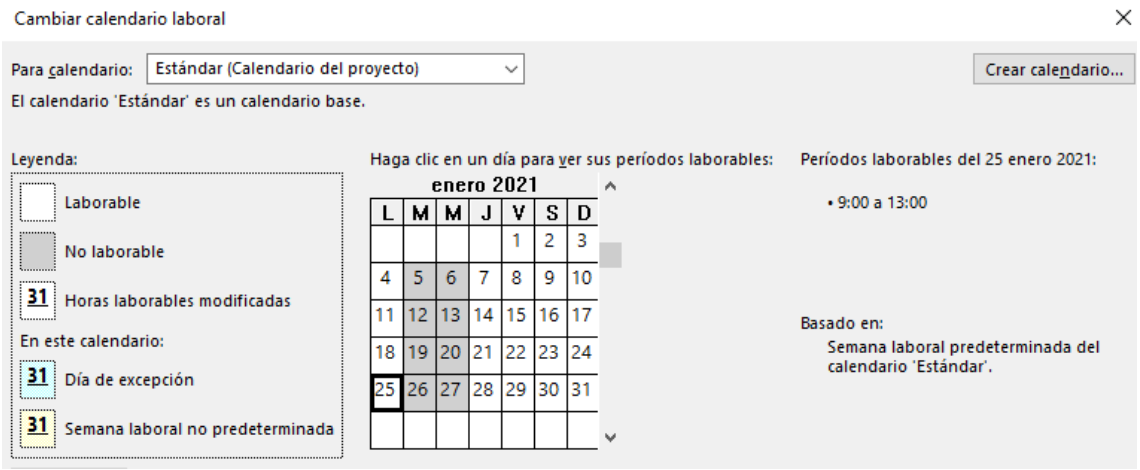


Ilustración 5.5 Configuración del calendario del proyecto en MS Project.

Se ha definido de esta manera debido a que los martes y miércoles son los dos días de mayor carga académica para el trabajador que también está cursando el grado en ingeniería informática. Este calendario es aproximado ya que habrá días en los que no se trabaje las 4 horas correspondientes o incluso se dedique tiempo de trabajo en alguno de los días no laborables.

Teniendo en cuenta este calendario de trabajo, se obtienen las fechas estimadas de las tareas y el correspondiente diagrama de *Gantt* del proyecto que se muestra en la sección anterior 5.1.1 ETAPAS DEL PROYECTO.

5.1.3 WBS/PBS

Una vez visto el desglose de tareas y el diagrama de *Gantt*, en esta sección se presenta el diagrama ***WBS/PBS*** (*Work Breakdown Structure / Product Breakdown Structure*), el cual refleja de manera visual el alcance del sistema y sus entregables. Este diagrama representa las etapas del proyecto mediante rectángulos y los productos derivados de cada una de ellas mediante

óvalos. Proporciona una idea general y simplificada de cuáles son las actividades involucradas en el proyecto y los principales productos resultantes.

Este diagrama únicamente contiene el nivel de detalle correspondiente a un desglose de alto nivel del proyecto que sirva de base para desarrollar la EDT del proyecto, aunque se podría profundizar más en cada uno de los productos y tareas para generar a su vez nuevos diagramas *WBS/PBS*. Sin embargo, no se va a profundizar en demasiados detalles ya que se estarían abarcando las actividades propias del análisis del sistema.

La siguiente figura muestra el aspecto de este diagrama *WBS/PBS* global del proyecto de *Contact Tracker*.

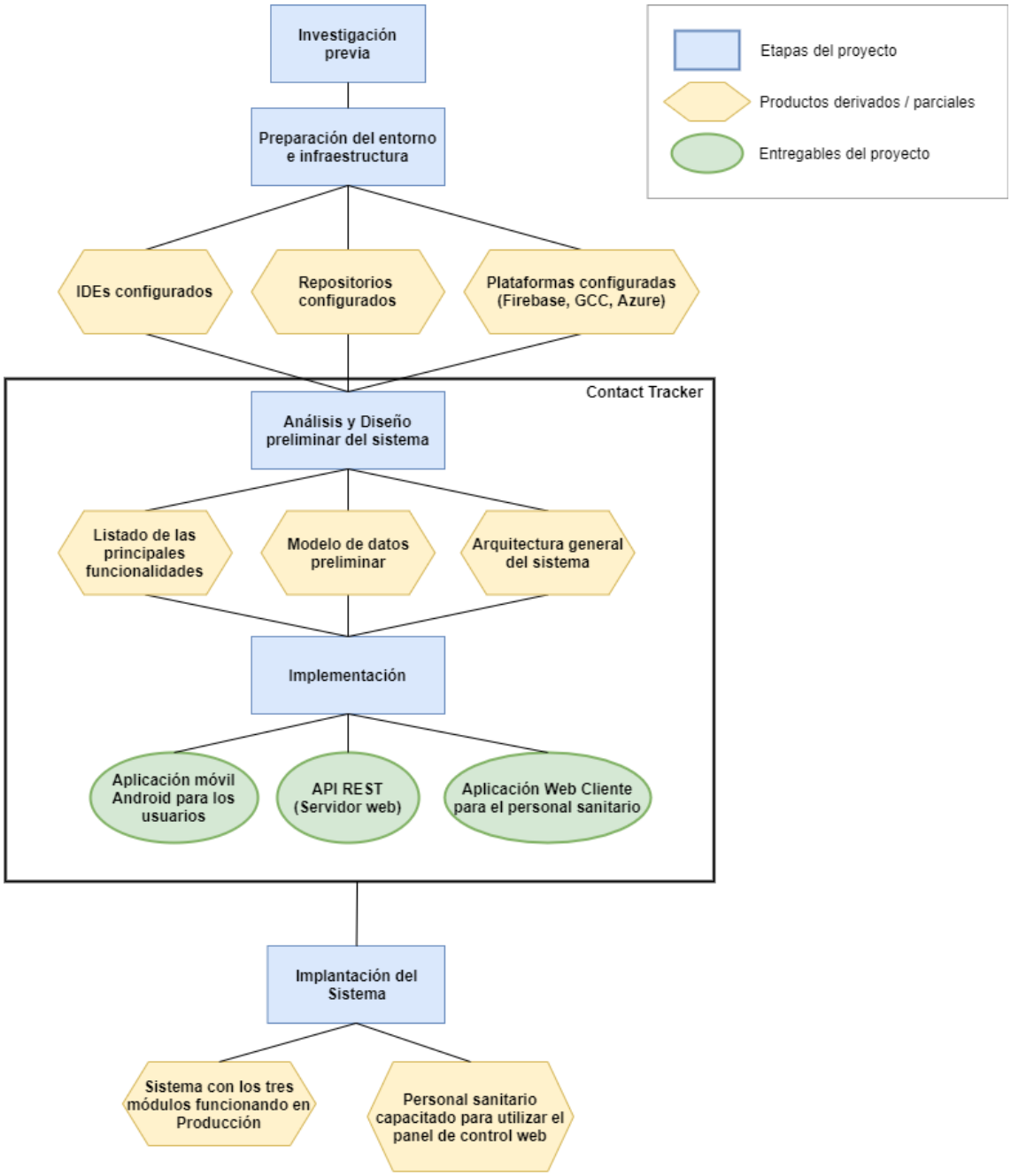


Ilustración 5.6 Diagrama WBS/PBS global del proyecto.

Como se puede apreciar, los óvalos verdes representan los **entregables del proyecto**, es decir, los productos resultantes del desarrollo y que serán entregados al cliente. Estos entregables son la **aplicación *Android,*** la **aplicación *web* para el personal sanitario** y la **API REST.** Por otro lado, los hexágonos amarillos representan los productos parciales resultantes de otras tareas o etapas del proyecto.

Por último, en la parte inferior del diagrama, se encuentra la etapa de implantación del sistema que da como resultado dos productos parciales, el **sistema puesto en producción** y el **personal sanitario formado en el panel de control *web*.**

5.1.4 Planificación dentro de la metodología ágil

Ya se ha visto que la naturaleza ágil de la metodología de desarrollo empleada hace que sea complicado realizar una planificación completa antes de comenzar a implementar. En las secciones anteriores se ha tratado de dar una visión general sobre las etapas que se van a llevar a cabo para completar el proyecto mediante una EDT, pero todo ello sin entrar en detalles de tareas concretas en las distintas iteraciones.

Uno de los principios fundamentales de la metodología ágil es **satisfacer al cliente con entregas tempranas de *software* que funcione**, es decir, dar valor al cliente lo antes posible, trabajando conjuntamente con él para decidir cuáles son las funcionalidades más prioritarias en un momento dado, además de revisar y aclarar cualquier aspecto relacionado con el proyecto. Por ello, se prioriza la implementación de código por encima de la planificación previa propia de los desarrollos tradicionales.

Scrum es la metodología que se pretende utilizar para la construcción del sistema. Establece las bases para un desarrollo flexible e iterativo, dividiendo el proyecto en ***sprints*** en los cuales se lleva a cabo el desarrollo de ciertas funcionalidades que se seleccionan al inicio de cada iteración y que se especifican mediante **historias de usuario**. Al final del *sprint* se espera tener un producto parcialmente entregable, es decir, que funcione y que aporte cierto valor al cliente.

La mayor carga de planificación dentro de *Scrum* se da al **inicio** de cada ***sprint***, en una reunión denominada ***Sprint Planning***. A esta reunión acude el equipo de desarrollo y el ***product owner*** (cliente) y se divide en dos partes. En la primera, se decide **qué** se va a hacer, es decir, el *product owner* da prioridad a las historias de usuario que se apilan en el ***Product Backlog***, de forma que se pone de acuerdo con el equipo de desarrollo para seleccionar las historias de usuario a implementar durante el *sprint.* En la segunda parte, se reúne el equipo de desarrollo sin el *producto owner* para decidir **cómo** se van a implementar las historias además de **estimar el esfuerzo** necesario para llevarlas a cabo. Una vez seleccionadas, estas historias pasan a formar parte del ***Sprint Backlog*** y los miembros del equipo de desarrollo dialogan entre sí para coordinarse y analizarlas brevemente.

A continuación, se describen las características más destacadas de *Scrum* que serán aplicadas durante el desarrollo del sistema:

- **Reuniones**: a lo largo de todo el ciclo de vida del proyecto, se planifican reuniones para diversos cometidos. Al inicio de cada *sprint* se planifica una reunión con el tutor del proyecto para seleccionar las historias de usuario que se van a implementar

durante el *sprint* (***Sprint Planning***) y para estimar su duración. Del mismo modo, al final de cada *sprint* se planifica otra reunión con el tutor del proyecto, que juega el papel de *Product Owner*, para revisar el producto resultante de esa iteración, de forma que proporciona *feedback* al equipo (formado en este caso por una sola persona) sobre aspectos a mejorar o modificar (***Sprint Review***). En esa misma reunión, también se realiza la **retrospectiva** del *sprint* donde el tutor pasa a formar parte del equipo de desarrollo para reflexionar sobre lo que se ha hecho bien, lo que se puede mejorar, que obstáculos o preocupaciones tiene el equipo…*etc*. Durante todo el ciclo de vida, el equipo puede consultar al tutor (dueño del producto) cualquier duda o incidencia que surja durante el desarrollo. También se pueden planificar reuniones para refinar la pila del producto, detallando historias de usuario, fragmentándolas, cambiando su prioridad, estimándolas…*etc* (***Backlog Grooming***).

- **Iteraciones**: son periodos de tiempo de 4 semanas (en este caso, pero pueden variar entre 2 semanas y 2 meses) durante los cuales se construye el sistema. Según el calendario de trabajo visto antes, durante estas 4 semanas el equipo trabajará 5 días a la semana a 4 horas por día, aunque esto puede variar en función de la carga de trabajo del equipo.
- ***Backlog**:* representa la pila en la que se almacenan las historias de usuario. Hay una para el proyecto en su totalidad y otra para cada *sprint*. En este caso se utilizará un **tablero *Kanban*** para organizar las historias en función de su estado que será explicado más adelante en el CAPÍTULO 6 sobre control y seguimiento del proyecto.

Dada la flexibilidad de *Scrum*, es más interesante llevar a cabo un buen **control** y **seguimiento** del proyecto en lugar de una planificación exhaustiva, lo cual se describirá en el siguiente capítulo dedicado a la monitorización del proyecto.

5.2 Resumen del Presupuesto

Como ya se ha comentado, el desarrollo del proyecto se enmarca dentro de la metodología ágil de *Scrum*, lo cual impide realizar una planificación exhaustiva desde el principio de los *ítems* y tareas que constituyen las fases del proyecto junto con su duración estimada. Esto hace que tampoco sea posible diseñar un presupuesto exacto de lo que van a costar las distintas fases del proyecto y el desarrollo de los entregables. En su lugar, se ha tratado de dividir el proyecto en unas fases globales dentro de las cuales se han distinguido tareas de alto nivel, mientras que las cuatro iteraciones planificadas no se han desglosado en tareas, sino que estas se van definiendo sobre la marcha en la planificación inicial de cada *sprint*.

Teniendo en cuenta el coste por hora calculado a partir del salario medio de un ingeniero informático con unos conocimientos determinados tal y como se detalla en el 13.1 MODELO DE EMPRESA, y la duración estimada de los *ítems* de la EDT presentada en la sección 5.1.1.1 EDT Y DIAGRAMA DE GANTT, se obtiene el **coste total** de las **partidas** del proyecto.

De cara al **cliente**, el presupuesto se simplifica ponderando ciertos costes que no se presentan de forma que se resume en tres partidas:

1. **Etapa de implementación**. Constituye la construcción del sistema durante las cuatro iteraciones planificadas.

2. **Etapa de implantación del sistema y formación**. Representa el despliegue e implantación del sistema una vez construido, incluyendo la formación para los usuarios administradores del panel de control *web*.
3. **Etapa de mantenimiento del sistema**. Proceso de mejora, adaptación y mantenimiento del sistema durante un periodo de 12 meses.

A continuación, se muestra el presupuesto de cliente para el sistema de *Contact Tracker* incluyendo las partidas anteriores.

Tabla 5.1. Presupuesto final de cliente para Contact Tracker.

Presupuesto de cliente			
Código	**Partida**	**Subtotal**	**Total**
1	**Etapa de Implementación**		15.868,15 €
2	**Etapa de Implantación del Sistema y Formación**		371,91 €
	Coste total sin el año de mantenimiento del sistema		**16.240,06 €**
3	**Etapa de Mantenimiento del Sistema**		47.604,46 €
		COSTE TOTAL	**63.844,53 €**

Capítulo 6. Control y seguimiento

Partiendo de la planificación inicial mostrada en el capítulo anterior, en este capítulo se describen los procesos y herramientas usadas para controlar y monitorizar el avance del proyecto, así como una visión de la evolución de las iteraciones del mismo.

6.1 Historias de usuario

A diferencia de las metodologías tradicionales donde predominan los requisitos funcionales y no funcionales, en el marco de desarrollo *ágil*, las **historias de usuario** son el núcleo de la especificación en torno a las cuales se organizan las funcionalidades y servicios que debe prestar el sistema. Las historias de usuario constituyen los elementos atómicos de funcionalidad, por lo tanto, serán utilizadas para **especificar** las funcionalidades del sistema a construir. A continuación, se muestra una imagen de una historia de usuario de este proyecto en concreto.

Ilustración 6.1 Ejemplo de historia de usuario para el proyecto Contact Tracker.

Atendiendo a la figura superior, podemos distinguir los siguientes **componentes** dentro de una historia de usuario de *Contact Tracker*:

- **Identificador**: número único que identifica a cada historia de usuario (Ej.: *HU 60*)
- **Título**: se trata del texto que resume la funcionalidad de la historia de usuario. Sigue el esquema clásico de *Como [tipo de usuario] deseo [realizar operación] para [obtener un valor].*
- **Asignación**: indica qué miembro del equipo de desarrollo es el responsable de implementar la historia de usuario.
- **Estado**: representa en qué estado se encuentra la historia de usuario.
    - ***New***: historia de usuario creada y almacenada en el *product backlog* (color lila).
    - ***Active:*** historia de usuario que ha comenzado a implementarse, se ha pasado al *sprint backlog* (color amarillo).
    - ***Resolved:*** historia de usuario terminada y con los *tests* unitarios pasando (color azul).

  - ***Closed:*** pasa las pruebas de aceptación, es decir, cumple con los criterios de aceptación establecidos en la historia. (color verde).
  - ***Removed:*** la historia de usuario ha sido eliminada (color rojo).
- **Iteración:** representa la iteración a la que está asociada la historia de usuario.
- **Etiqueta:** etiqueta que representa alguna característica de la historia, como, por ejemplo, que involucra al *backend,* al *frontent* o a la *app* móvil, o que está sin refinar, entre otras. En la siguiente figura se pueden ver las diferentes etiquetas que se pueden asignar a las historias de usuario.

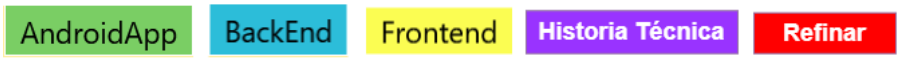


Ilustración 6.2 Etiquetas de las Historias de Usuario.

- **Descripción**: se trata de la descripción detallada de lo que debe hacer el sistema, que datos debe solicitar al usuario, que datos debe mostrar, las comprobaciones que deben hacerse…*etc.*
- **Criterios de aceptación:** son los criterios que debe cumplir el sistema para considerar que la historia de usuario está terminada y es aceptada por el cliente.
- ***Story Points*:** son los puntos de historia que se utilizan para cuantificar las historias de usuario en lo que respecta al esfuerzo necesario para completarlas. En este proyecto concreto, los puntos de historia expresan las **horas de trabajo** que se estima serán necesarias para completar la historia.
- ***Priority***: es un número comprendido entre 1 y 4 que expresa cómo de importante es la historia de usuario para el cliente.
- ***Risk:*** representa el riesgo que supone la historia de usuario para el proyecto. Puede ser bajo, medio o alto.
- **Clasificación**: es la categoría en la que se enmarca la historia de usuario.
  - **Business (negocio):** se trata de una historia de usuario con valor de negocio para el cliente.
  - **Arquitectural (técnica):** se trata de una historia de usuario con valor técnico para el equipo de desarrollo. Estas historias suelen tratar sobre temas de diseño y arquitectura del código.

En aquellos casos en los que surjan ***bugs*** significativos en alguna funcionalidad de una historia de usuario, se representan como hijos de esta, de tal forma que se mantenga la **trazabilidad** entre historias de usuarios y *bugs* asociados a las historias, tal y como se aprecia en la siguiente figura.

Ilustración 6.3 Ejemplo de Historia de Usuario con un bug.

Como se puede observar, se especifican los **pasos** para **reproducir** el *bug*, incluyendo opcionalmente información sobre el sistema en el que se está probando, la versión de la aplicación en la que se produce el fallo…*etc.* También se puede indicar la **gravedad** del fallo, qué **prioridad** tiene o el **esfuerzo** que conllevará arreglarlo. Una vez resuelto el problema, se puede añadir un comentario asociado al *bug* indicando cuál era el **origen** del fallo y cómo se **arregló**, tal y como se ve en la figura superior.

6.2 Pila del producto y de la iteración

Las historias de usuario descritas anteriormente se organizan en **pilas** ordenadas por **prioridad** en función del valor que suponen para el cliente. Por lo general, las historias de usuario más prioritarias están más detalladas y refinadas que aquellas historias que se encuentran en el fondo de la pila. En el proyecto se utilizan dos pilas para gestionar las historias de usuario:

- ***Product Backlog:*** se trata de la pila del producto que representa la funcionalidad de todo el sistema en su totalidad.
- ***Sprint Backlog***: es la pila de una iteración concreta, que contiene las historias de usuario que se seleccionaron durante la planificación del *sprint* y que el equipo se compromete a completar.

Estas pilas permiten al equipo de desarrollo y al dueño del producto visualizar el **alcance** que tiene el sistema, además de poder **monitorizar** el estado del proyecto, pues se puede ver cuántas historias de usuario faltan por completar en cada iteración, cuantas han sido ya completadas…*etc.*

La siguiente figura muestra un fragmento de la pila del producto en un momento del tiempo.

Work Item ...	Title	State	Story Points	Priority	Value Area	Iteration Path	Tags
Feature	Rastreo de la localización (location tracker).	New		2	Business	TFG - Contact Tracker\Sprint 1	
User Story	Como usuario Android deseo que la aplicación registre mis coordenadas de manera periódic...	Closed	12	4	Business	TFG - Contact Tracker\Sprint 1	AndroidApp
User Story	Como usuario Android deseo poder configurar múltiples alarmas de localización para registr...	Closed	12	2	Business	TFG - Contact Tracker\Sprint 2	AndroidApp
User Story	Como usuario Android deseo poder INICIAR/PAUSAR el registro de coordenadas desde la ap...	Closed	6	3	Business	TFG - Contact Tracker\Sprint 1	AndroidApp
User Story	Como usuario Android deseo que se guarden mis coordenadas en el almacenamiento local d...	Closed	4	3	Business	TFG - Contact Tracker\Sprint 1	AndroidApp
User Story	Como usuario Android deseo poder ver en tiempo real el rastreo de localización para ver los ...	Closed	4	2	Business	TFG - Contact Tracker\Sprint 1	AndroidApp
User Story	Como usuario Android deseo poder visualizar las localizaciones que se van registrando en un...	Closed	4	1	Business	TFG - Contact Tracker\Sprint 5	AndroidApp
Feature	Interfaz de usuario.	New		2	Architectu...	TFG - Contact Tracker\Sprint 1	
User Story	Refactorizar aplicación móvil para utilizar data binding.	Closed		2	Business	TFG - Contact Tracker\Sprint 5	AndroidApp Historia Técnica
User Story	Refactorizar aplicación móvil para utilizar la API Navigation de Jetpack.	Closed		2	Business	TFG - Contact Tracker\Sprint 5	AndroidApp Historia Técnica
User Story	Diseñar infraestructura y menú de la aplicación web.	Closed	5	2	Architectu...	TFG - Contact Tracker\Sprint 3	Frontend Historia Técnica
User Story	Como usuario Android deseo poder navegar a través de un menú para poder acceder a las d...	Closed	4	3	Business	TFG - Contact Tracker\Sprint 1	AndroidApp Historia Técnica
User Story	Mejorar la visualización de los resultados de las comprobaciones de contactos de riesgo.	Closed	4	2	Business	TFG - Contact Tracker\Sprint 6	AndroidApp Historia Técnica
Feature	Base de Datos y Persistencia.	New		3	Architectu...	TFG - Contact Tracker\Sprint 1	
User Story	Modificar la base de datos para que se almacenen milisegundos en lugar de strings.	Closed		2	Business	TFG - Contact Tracker\Sprint 5	AndroidApp BackEnd ...
User Story	Definir la capa de persistencia para el servidor del BackEnd.	Closed	8	2	Architectu...	TFG - Contact Tracker\Sprint 2	BackEnd Historia Técnica
User Story	Definir la capa de persistencia de los datos para la aplicación Android.	Closed	6	3	Business	TFG - Contact Tracker\Sprint 1	AndroidApp Historia Técnica
Feature	Configuración del sistema/aplicación y Ajustes Generales.	New		3	Business	TFG - Contact Tracker\Sprint 1	
User Story	Refactorizar configuración local de la aplicación Android.	Closed		2	Business	TFG - Contact Tracker\Sprint 5	AndroidApp Historia Técnica
User Story	Como administrador deseo poder configurar los rangos de valor de los parámetros de la co...	Closed	8	2	Business	TFG - Contact Tracker\Sprint 4	AndroidApp BackEnd ...
User Story	Como usuario Android deseo poder establecer una alarma periódica para que la aplicación re...	Closed	7	2	Business	TFG - Contact Tracker\Sprint 1	AndroidApp
User Story	Como administrador deseo poder configurar los porcentajes de peso de los parámetros de c...	Closed	6	2	Business	TFG - Contact Tracker\Sprint 4	AndroidApp BackEnd ...

Ilustración 6.4 Fragmento de la pila del producto en un instante del tiempo.

Como se puede observar, las historias de usuario se agrupan a su vez en ***Features***. Estas representan un *caso de uso* o característica concreta del sistema que agrupa varias historias de usuario.

Por otro lado, se encuentra la pila del *sprint* que se genera tras la planificación inicial de cada *sprint*. La siguiente figura muestra la pila del *sprint* 1.

Order	Title	State	Assigned To	Priority	Story Points	Remaining Work
1	Como usuario Android deseo poder eliminar las localiza...	Closed	Alejandro Flór...	2	5	
2	Como usuario Android deseo poder navegar a través de un ...	Closed	Alejandro Flór...	3	4	
3	Como usuario Android deseo poder INICIAR/PAUSAR el regi...	Closed	Alejandro Flór...	3	6	
4	Como usuario Android deseo que la aplicación registre mis ...	Closed	Alejandro Flór...	4	12	
5	Definir la capa de persistencia de los datos para la aplicació...	Closed	Alejandro Flór...	3	6	
6	Como usuario Android deseo que se guarden mis coordena...	Closed	Alejandro Flór...	3	4	
7	Como usuario Android deseo poder ver en tiempo real el ra...	Closed	Alejandro Flór...	2	4	
8	Como usuario Android deseo poder ver un histórico con las ...	Closed	Alejandro Flór...	2	5	
9	Como usuario Android deseo poder establecer una alarma p...	Closed	Alejandro Flór...	2	7	

Ilustración 6.5 Pila del Sprint de la primera iteración.

De manera adicional, las historias de usuario se pueden dividir a su vez en **pequeñas tareas** de carácter técnico que es necesario completar para poder cerrar la historia. Estas tareas no están dirigidas al cliente, sino que son interpretadas por el equipo de desarrollo. Sin embargo, en el contexto de este proyecto, solo las historias de usuario más complejas se dividen en tareas que especifican detalles técnicos de cara a ayudar al equipo de desarrollo a definir cómo se deben implementar estas historias más densas.

6.3 Story mapping

Además de la pila del producto y del *sprint* descritas anteriormente, durante todo el desarrollo del proyecto se ha utilizado una herramienta típica del marco de trabajo *Scrum* denominada ***story mapping***, la cual permite organizar y ordenar las historias de usuario a través de **dos**

dimensiones independientes, en este proyecto en concreto por **característica** y por **iteración**. Una característica agrupa varias historias de usuario relacionadas entre sí y que dan lugar a una funcionalidad de la aplicación.

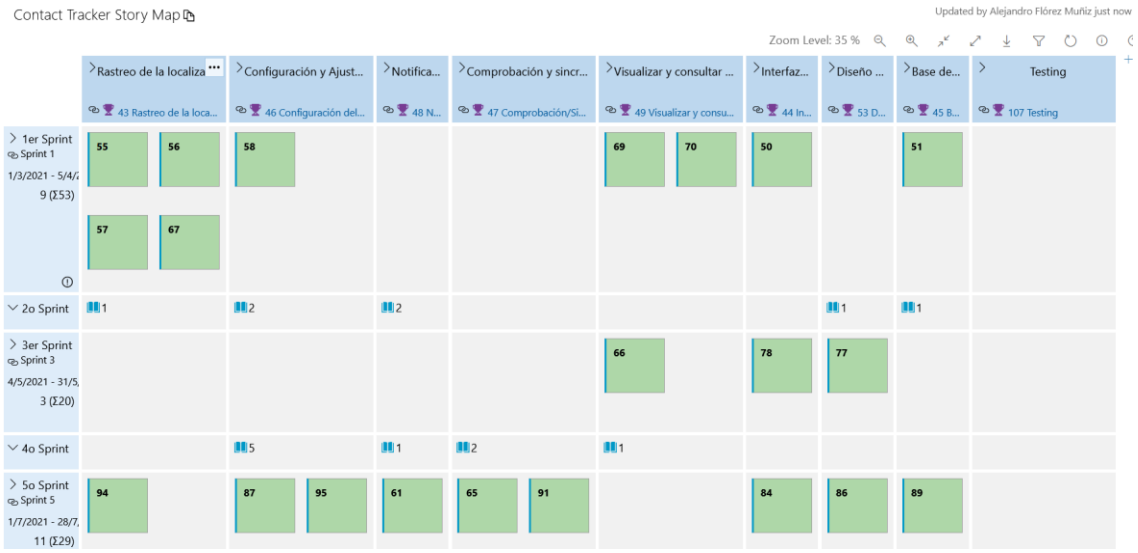


Ilustración 6.6 Fragmento del Story Mapping para el proyecto.

La imagen superior muestra solo un fragmento resumido del *story mapping*. Las **columnas** representan las **características** principales del sistema en su totalidad, mientras que las **filas** representan las distintas **iteraciones** o ***sprints***. Como se puede ver, es muy cómodo y efectivo para llevar a cabo la gestión de las historias de usuario durante las iteraciones.

Al inicio de cada fila, el *story mapping* muestra el **número de iteración,** la **fecha de inicio y de fin** y un recuento con el **número total de historias** en esa fila (iteración) y la **suma total de *story points***.

A continuación, se muestra una captura del *story mapping* con un mayor nivel de detalle, donde se puede ver cada historia de usuario con su título, sus etiquetas, sus puntos de historia y su estado actual. En el anexo 15.4.6, se puede consultar este *story mapping* en su totalidad.

Ilustración 6.7 Fragmento del Story Mapping con más detallado.

Mediante esta herramienta, al inicio de cada *sprint* se realizaba la reunión del *sprint planning* para acordar y estimar las nuevas historias de usuario que pasarían a formar parte del *sprint*, colocándolas en el *story mapping*, bajo su característica o *feature* correspondiente y en la fila adecuada a la nueva iteración.

El *story mapping* facilita el proceso para **planificar** cada iteración, así como las sesiones de **refinamiento de la pila del producto**, ya que permite acceder a las historias de usuario correspondientes a una iteración y a una característica concreta de manera rápida, sencilla y visual.

6.4 Evolución y seguimiento del proyecto

Dentro del marco de trabajo de *Scrum*, lo realmente interesante es llevar a cabo un seguimiento del proyecto monitorizando el estado de cada *sprint* para comprobar cuál es el ritmo de trabajo por encima de planificar exhaustivamente todas las tareas del ciclo de vida del proyecto. Para monitorizar el estado del proyecto se ha hecho uso de una herramienta muy típica dentro de las metodologías ágiles: los **diagramas/gráficos de quemado (*burndown charts*).**

A partir de las historias de usuario y de sus *story points* se construye un diagrama de quemado por cada **iteración**. Estos diagramas presentan de manera muy visual cómo va avanzando el desarrollo del *sprint* dibujando mediante una gráfica las historias o *story points* que faltan por completar en cada momento del tiempo dentro del *sprint*, es decir, expresan el **trabajo *quemado*** en un momento dado. Además, muestran una **línea de tendencia** que representa el **flujo ideal** de quemado de las historias, es decir, cómo se debería de ir completando las historias de usuario para finalizar el *sprint* con éxito y de forma constante, sin subidas o bajadas repentinas en la gráfica, y todo ello respetando el **calendario de trabajo** definido.

Los principales **componentes** de los diagramas de quemado utilizados en cada *sprint* son:

- **Puntos de historia pendientes (eje Y).** El eje vertical del diagrama representa los valores acumulados de los *story points* de las historias de usuario correspondientes al *sprint* del gráfico, de forma que en un día concreto se muestra el número de puntos de historia que faltan por completar.
- **Día del *sprint* (eje X).** El eje horizontal representa los días de toda la iteración.
- **Línea de Alcance.** Representa el alcance del *sprint*, es decir, la cantidad de trabajo planificado para realizar durante el *sprint*. En este caso concreto, el alcance se expresa mediante el número de puntos de historia acumulados en total de todas las historias de usuario que forman parte de la iteración.
- **Línea de Tendencia Ideal.** Es el flujo de trabajo ideal teniendo en cuenta el calendario de trabajo, es decir, los días no laborables. Expresa el quemado de los puntos de historia de manera constante, aunque en la mayoría de los casos no se asemeja a la realidad.

Mediante estos diagramas de quemado se toma el ***software*** que funciona como **medida del progreso**, tal y como establece uno de los principios del **manifiesto ágil**. Además, cabe destacar que otro de los principios fundamentales del desarrollo ágil es el hecho de que el **alcance** de un *sprint* es **cerrado**, es decir, no se pueden incluir nuevas funcionalidades, sino que estas se posponen para las siguientes iteraciones. Sin embargo, como se verá a continuación en los gráficos de quemado resultantes de cada *sprint*, esto parece no cumplirse ya que la **línea de alcance** va variando a lo largo del *sprint*. Esto es debido a las sesiones de ***backlog grooming***, es decir, de refinamiento de la pila, pues por ciertos contratiempos se refinaban algunas historias de usuario durante el desarrollo del *sprint,* dividiendo o **fragmentando** las historias en otras más pequeñas o bien añadiendo nuevas historias a partir de otras existentes, lo cual hacía que aumentase el **alcance** del *sprint*.

A continuación, se describe el diagrama de quemado de cada una de las iteraciones incluyendo las funcionalidades planificadas, el trabajo realizado y los obstáculos o contratiempos que se fueron encontrando.

6.4.1 Sprint 1

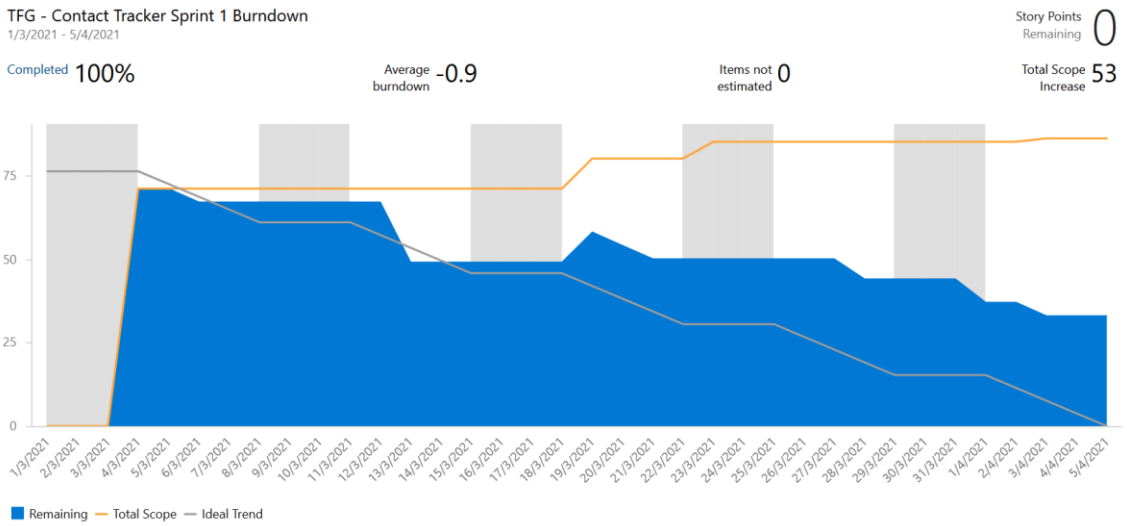


Ilustración 6.8 Diagrama de quemado del Sprint 1.

El proyecto despega con esta primera iteración el día 3 de marzo, tras refinar y estimar las historias de usuario seleccionadas para el *sprint.* Durante este, se comenzó la implementación de la aplicación móvil, creando la estructura de navegación para el menú principal, definiendo la capa de datos y la persistencia en la base de datos interna del dispositivo, todo ello envuelto en la arquitectura *MVVM* ya mencionada. Además, se desarrolló el **registro de localizaciones** en la aplicación móvil, tanto en primer plano como con la aplicación apagada, y su almacenamiento en la base de datos local, incluyendo la visualización de un histórico con las coordenadas registradas. También se logró implementar la programación de alarmas periódicas para iniciar el registro de localizaciones de manera automática a una hora determinada.

Durante esta primera iteración no hubo ningún obstáculo significativo, aunque como se puede observar en la gráfica, no se logró completar a tiempo todas las historias de usuario planificadas, posponiéndolas para el siguiente *sprint.*

6.4.2 Sprint 2

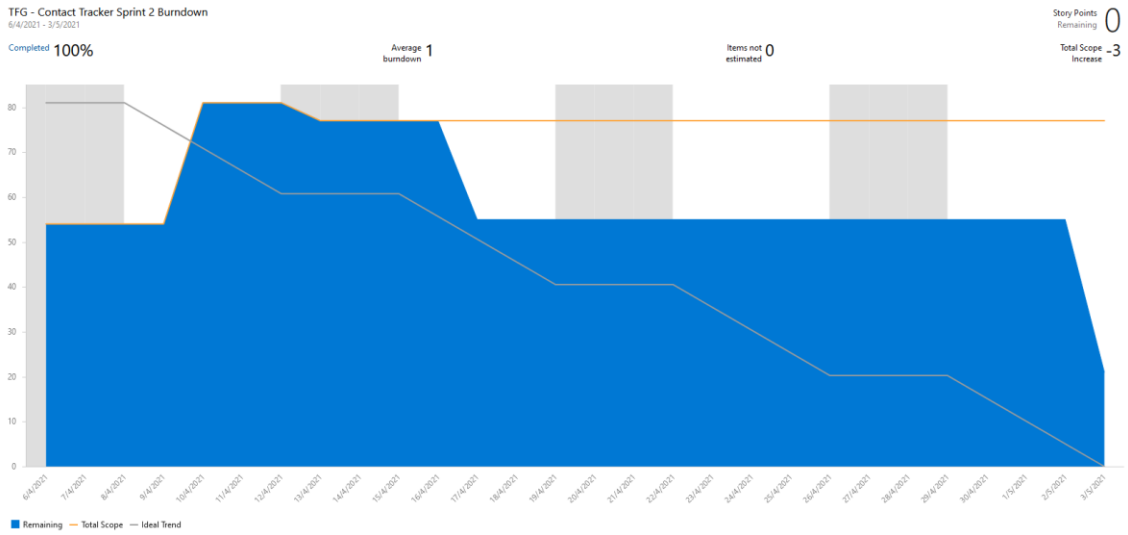


Ilustración 6.9 Diagrama de quemado para el Sprint 2.

En esta iteración, se continúa con la implementación de las funcionalidades de la aplicación móvil. En torno al día 9 de abril tiene lugar una sesión de refinamiento de historias de usuario detallándolas y fragmentándolas en otras historias por lo que aumenta el alcance. Se puede observar que desde el día 17 hasta el final del *sprint* la gráfica permanece constante sin quemar nada de trabajo, esto es debido a contratiempos causados por la carga de trabajo de otras tareas externas del equipo.

Se implementa la programación de **múltiples alarmas** de localización además de comenzar con la **configuración del rastreo**, es decir, se añade un apartado de configuración donde se puede parametrizar el **intervalo de tiempo** del registro de coordenadas y el **desplazamiento mínimo** necesario para que se reciban localizaciones. También se inicia el desarrollo de otra de las funcionalidades principales del sistema, la **notificación de positivos**, permitiendo al usuario notificar un positivo subiendo a la nube todas sus localizaciones registradas en los últimos días, adjuntando opcionalmente sus datos personales.

Por último, en este *sprint* comienza la implementación de la ***API REST***, configurando el enrutamiento y los *endpoints* del *backend* para responder a las peticiones de la aplicación móvil. También se define la **comunicación** entre el servidor *web* y la plataforma de ***Firebase*** para establecer la capa de persistencia con la base de datos documental ***Firestore***.

De nuevo se observa en la gráfica que algunas historias de usuario quedaron pendientes y fueron pospuestas para el siguiente *sprint*.

6.4.3 Sprint 3

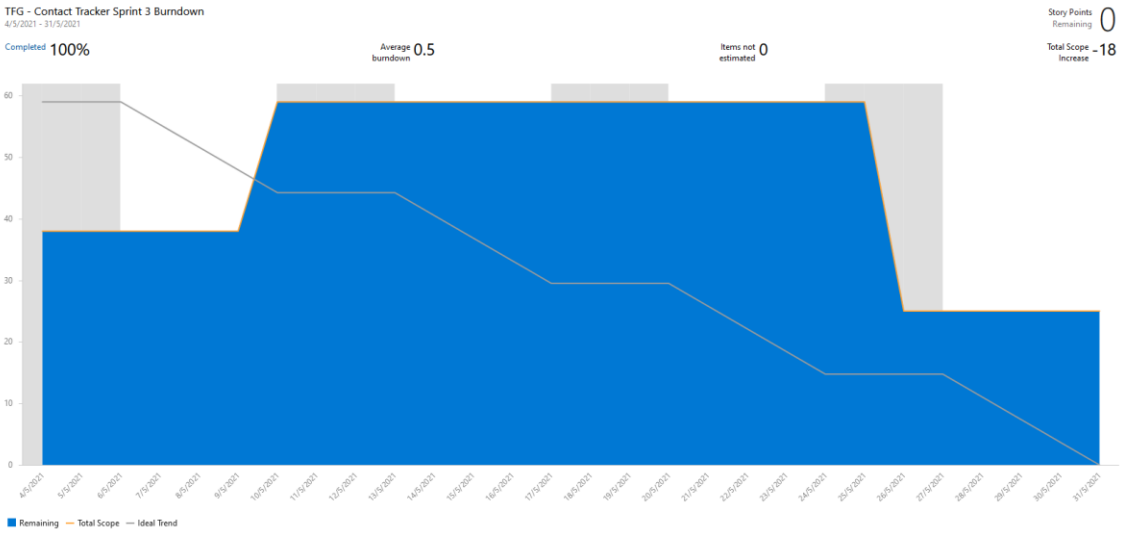


Ilustración 6.10 Diagrama de quemado del tercer Sprint.

El tercer *sprint* se desarrolla durante la **época de exámenes** y **entrega de trabajos**, por lo que se trata de la peor iteración en cuanto a trabajo realizado. Como se puede observar en la gráfica, apenas existen zonas de pendiente negativa, lo cual indica que casi no se ha quemado trabajo. En este caso la entrega de múltiples trabajos y realización de exámenes obstaculizaron el desarrollo del proyecto durante la mayoría del *sprint*.

Las funcionalidades implementadas durante este *sprint* fueron la **visualización** de localizaciones registradas en un **mapa** en el histórico de la aplicación móvil y la **arquitectura** y **configuración inicial** de la aplicación **cliente *web*** para el personal sanitario. Se definió el menú principal de la aplicación *web* y la estructura interna de paquetes para organizar las vistas y sus componentes, aunque aún no se estableció la conexión con la *API REST* del *backend*.

6.4.4 Sprint 4

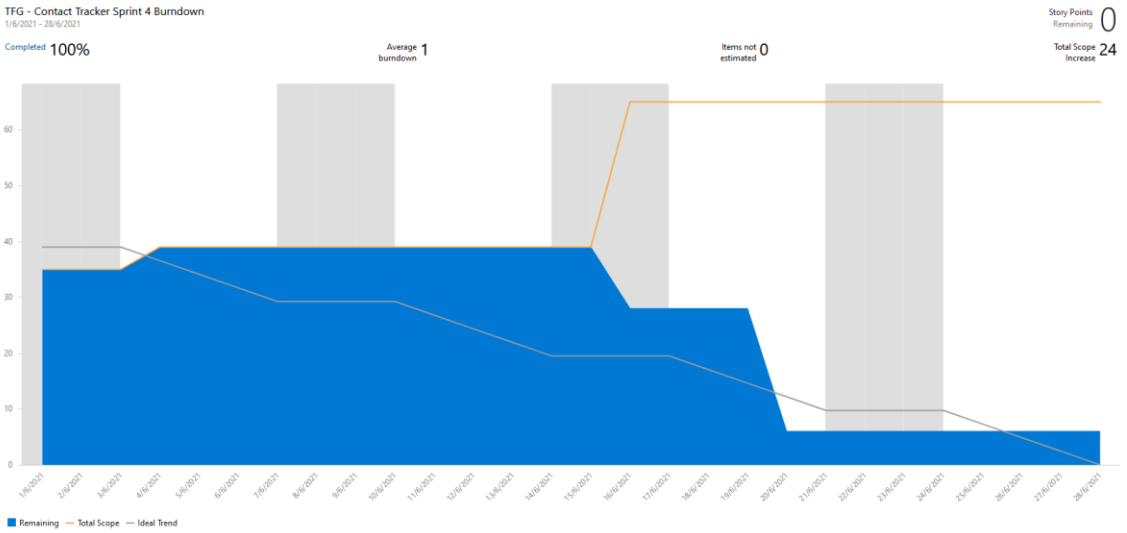


Ilustración 6.11 Diagrama de quemado para el Sprint 4.

Durante el cuarto *sprint*, también hubo diversos contratiempos debido a la entrega de trabajos externos al proyecto, lo cual se ve reflejado en el diagrama de quemado. En torno a la mitad del *sprint,* se puede apreciar de nuevo otra sesión de refinamiento de historias, haciendo que aumente el alcance.

En cuanto a las funcionalidades, se llevó a cabo la implementación de la **configuración** de los **parámetros** del **algoritmo de comprobación** por parte de los usuarios administradores (personal sanitario) a través de la aplicación *web*. También es añadida una **cláusula de consentimiento** para el tratamiento de los datos personales del interesado, una vez se notifica un positivo añadiendo los datos personales.

Lo más destacado del *sprint* fue el desarrollo del **algoritmo de comprobación** para comparar las coordenadas de un usuario con las coordenadas de los positivos de los días anteriores y así detectar posibles contactos de riesgo. De forma adicional, se implementa la **visualización** de los **detalles** de un **resultado de la comprobación**, mostrando los contactos de riesgo detectados en un mapa y los valores de tiempo de exposición, proximidad media, y porcentaje de riesgo.

Por último, se realizaron diversas **pruebas unitarias** sobre el algoritmo de comprobación y alguna de sus clases auxiliares para comprobar su correcto funcionamiento en diferentes situaciones de prueba.

6.4.5 Sprint 5

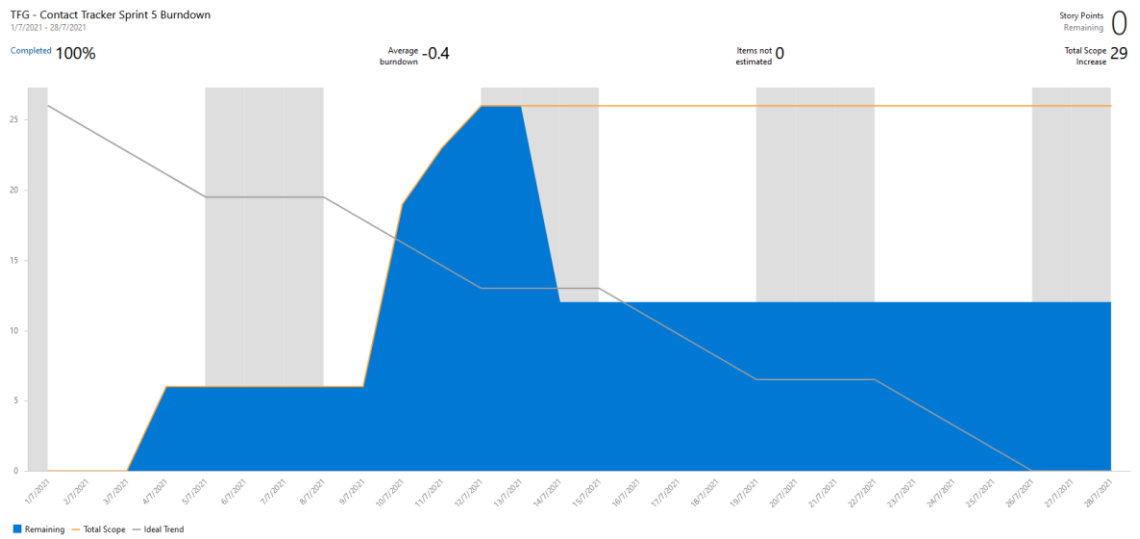


Ilustración 6.12 Diagrama de quemado para el Sprint 5.

Según lo planificado en el apartado 5.1.1.5 ETAPA DE IMPLEMENTACIÓN, el proyecto constaría de un total de cuatro *sprints* para ser finalizado, sin embargo, debido a los diversos contratiempos y retardos reflejados en los diagramas de quemado vistos anteriormente, fue necesario **posponer la entrega** del proyecto incluyendo **nuevos *sprints*.**

El *sprint 5* es el primero de estas iteraciones extra donde se llevaron a cabo múltiples **refactorizaciones internas** de la aplicación móvil, pasando a utilizar las ***APIs*** más modernas de *Android* en lo que respecta a la navegación, *data binding*… entre otros. También se **simplificó el código** de algunas clases, mejorando el diseño y reestructurando los componentes. A la hora de notificar positivos, se comprueba los **códigos identificadores** de los positivos para que no se comparen las localizaciones de un usuario con su propio positivo notificado (en el caso de que hubiera notificado uno).

Se añadieron nuevas funcionalidades como la configuración de un **límite de notificación de positivos** diario por parte del personal sanitario, la configuración por parte del usuario del **alcance de las comprobaciones,** la programación de **alarmas de comprobación** automáticas y la recepción de una notificación con los resultados de una comprobación.

6.4.6 Sprint 6

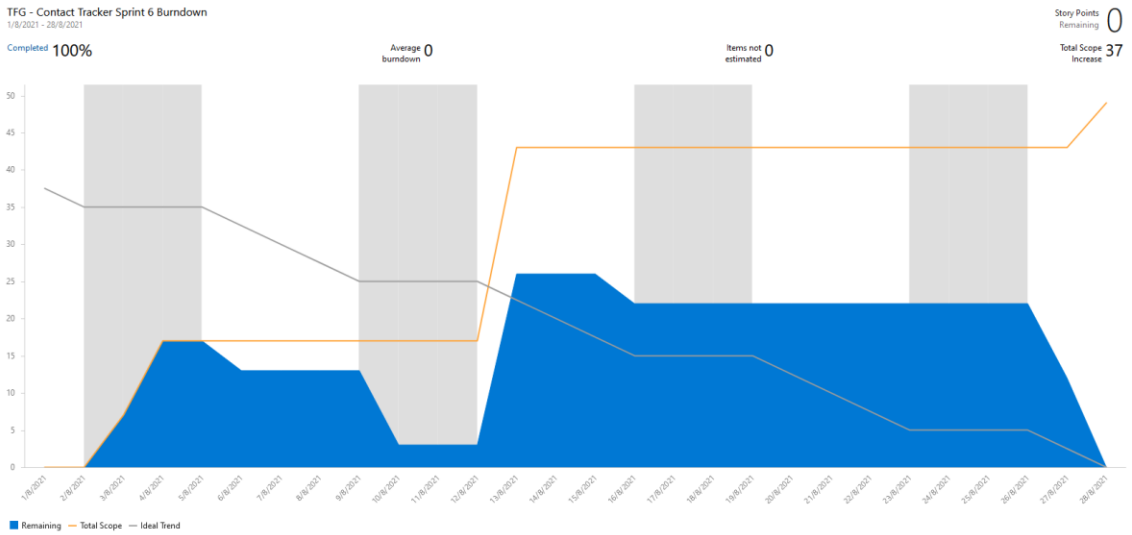


Ilustración 6.13 Diagrama de quemado del Sprint 6.

Este es el último *sprint* en lo que respecta al desarrollo de la **funcionalidad principal** del sistema. Tiene lugar durante el mes de agosto y en él se pulieron ciertos detalles de la aplicación *web,* del *backend* y se refactorizó la visualización de los resultados de las comprobaciones en la aplicación móvil para que fuese más amigable visualmente, así como el listado de comprobaciones que ahora se organizan por fecha de comprobación.

La funcionalidad añadida más destacable fue la **visualización de estadísticas** y de **positivos notificados** en la aplicación *web*. Ahora cuando se notifica un positivo, se solicita al usuario que indique si es **asintomático** y si está **vacunado,** todo ello con objetivos estadísticos.

Se añadió un *dashboard* que muestra ciertos datos estadísticos de uso de la aplicación, positivos notificados, y medias aritméticas de los resultados de comprobaciones de contactos de riesgo, entre otros. También se incluyó un nuevo apartado para visualizar un listado con los positivos notificados, donde el administrador puede seleccionar uno para ver sus datos personales (si existen) o para visualizar su itinerario en un mapa, es decir, las rutas que ha seguido ese positivo.

Con estos desarrollos se cierra la implementación del sistema, dando paso al siguiente *sprint* dedicado fundamentalmente a las pruebas del sistema.

6.4.7 Sprint 7

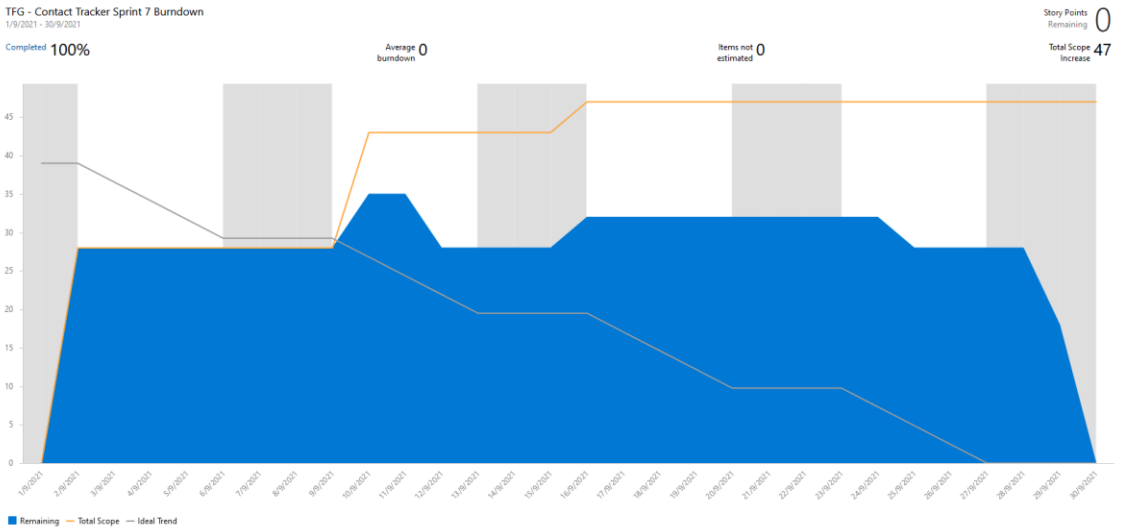


Ilustración 6.14. Diagrama de quemado del Sprint 7.

Este último *sprint* no es realmente un *sprint* de desarrollo, sino que se trata un *sprint* auxiliar para representar las últimas etapas del desarrollo del sistema, involucrando pequeños retoques y refactorizaciones, además del grueso del desarrollo de las **pruebas** de *Contact Tracker*.

Lo más significativo de esta iteración es el desarrollo de todas las pruebas unitarias, de integración y sistema especificadas en el Plan de Pruebas. Además, también se modifica la **restricción** del **límite** de notificación de positivos para que pase a ser por días, es decir, es necesario que transcurra un número dado de **días** para poder notificar otro positivo. Este número de días es configurable por el administrador.

Otro de los pequeños desarrollos realizados es el **envío** de **notificaciones *push*** a los clientes *Android* indicando el **número** de **positivos** que se han notificado en ese día. Dichas notificaciones se envían diariamente a una hora determinada que puede ser configurada por el administrador desde el panel *web*. Además, los propios usuarios también pueden habilitar o deshabilitar el envío de estas notificaciones a través de los ajustes generales de la aplicación móvil.

6.5 Resumen global del proyecto

Una vez vista la evolución de cada iteración individual, se presenta una visión **global** de la evolución de todo el proyecto, involucrando a todas las iteraciones. Al igual que con los anteriores gráficos de quemado, la siguiente gráfica muestra el **trabajo quemado** frente al trabajo pendiente durante el transcurso de las siete iteraciones, con un intervalo de tiempo representado en **días**. Dada la limitación impuesta por la herramienta de *Azure*, estos gráficos solo pueden representar hasta 180 puntos, por lo que el siguiente gráfico solo representa el periodo de tiempo desde la iteración 1 a la iteración 6.

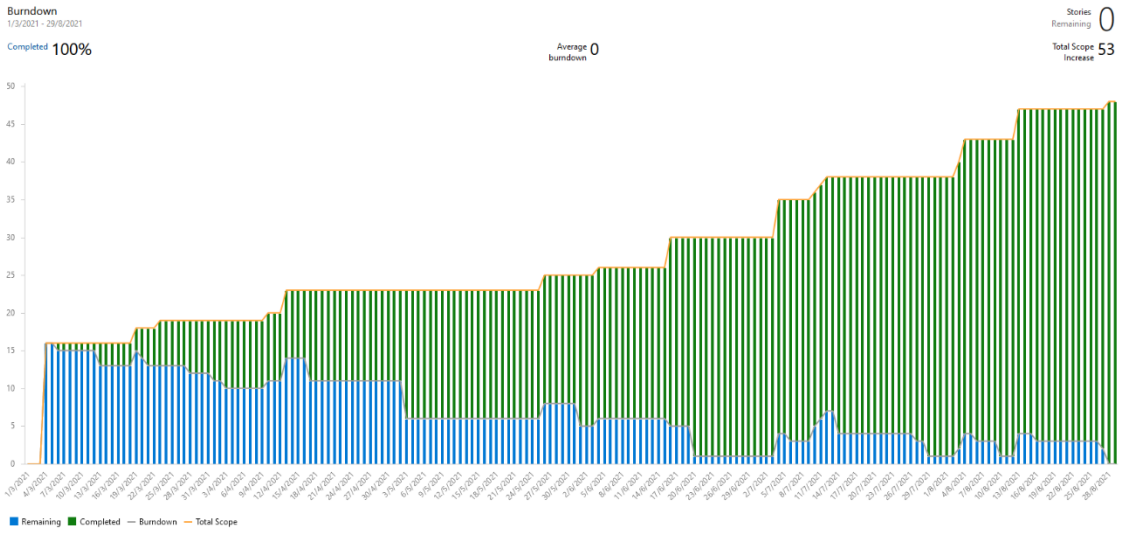


Ilustración 6.15. Diagrama de quemado global de todas las iteraciones del proyecto con intervalo de días.

El eje horizontal representa el periodo de tiempo en días desde el inicio del proyecto hasta el final de la iteración 6, mientras que el eje vertical representa el número de historias de usuario acumuladas. De este modo, las barras azules representan el **trabajo pendiente** en un momento dado y las barras verdes muestran el **trabajo completado** acumulado en un momento del tiempo. La **línea gris intermedia** representa la evolución real de quemado de las historias de usuario.

La siguiente gráfica refleja también el trabajo quemado, pero con un intervalo medido en **semanas**, de forma que en este caso se muestra el periodo de tiempo comprendido por todas las iteraciones.

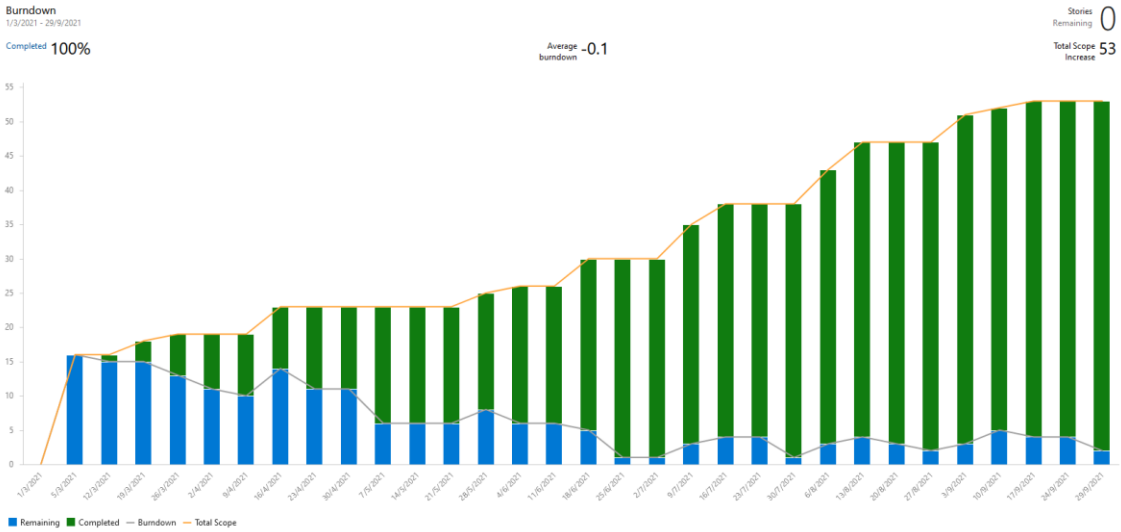


Ilustración 6.16. Diagrama de quemado global para todo el proyecto con un intervalo de semanas.

Capítulo 7. Análisis

En este capítulo se detalla la **especificación** del sistema de *Contact Tracker*, describiendo qué se debe construir mediante requisitos funcionales y no funcionales. Esta especificación constituye la base para el posterior diseño del sistema y sirve para delimitar qué funcionalidades y servicios debe soportar el sistema. A diferencia de los desarrollos tradicionales, se utilizarán historias de usuario para especificar las funcionalidades del sistema.

7.1 Definición del Sistema

Esta primera sección tiene como objetivo definir a grandes rasgos el sistema a construir, tomando como base los primeros capítulos donde se especificaban los antecedentes y la situación actual, así como los objetivos perseguidos por el sistema.

7.1.1 Determinación del Alcance del Sistema

Como ya se comentó en el capítulo 2, la pandemia desatada en el año 2020 ha obligado a desarrollar diversos protocolos de rastreo para tratar de frenar la subida de contagios, lo cual implica controlar a los positivos en *COVID-19* y sus contactos cercanos. Por ello, el objetivo de este proyecto es desarrollar un sistema de rastreo que permita a los usuarios conocer si han entrado en contacto con otros positivos, utilizando para ello sus propias ubicaciones que se registran mediante el *GPS* del dispositivo móvil.

El registro de ubicaciones se lleva a cabo de forma periódica de modo que las coordenadas se van almacenando en el dispositivo local del usuario, manteniendo así la anonimidad. Esto permite tener un histórico local de los itinerarios que sigue el usuario agrupando sus localizaciones por días. En el momento en que el usuario sea diagnosticado como positivo en *COVID-19*, puede indicarlo en la aplicación, de forma que todas sus ubicaciones registradas en los últimos días sean subidas al servidor central. El resto de usuarios puede entonces realizar comprobaciones para comparar sus itinerarios con el de los positivos registrados en la nube y así detectar si ha tenido contactos estrechos con alguno de ellos.

Uno de los pilares fundamentales de este sistema, es el mantenimiento de la **privacidad** de los usuarios, mediante un tratamiento de los datos acorde con la ley de protección de datos vigente. Así, las coordenadas de los usuarios residen en el dispositivo local de forma anónima, hasta que deciden notificar que han dado positivo, tras lo cual se suben a la nube guardándose en la base de datos central, dándoles el control total para decidir si compartir o no sus ubicaciones. Las funciones del sistema deben ser afines con la **normativa** de **protección de datos** establecida actualmente en Europa, concretamente debe respetar las cláusulas definidas en la ***RGPD/LOPD***, tal y como se expone en el CAPÍTULO 4 ESTUDIO LEGAL dedicado expresamente a ello.

El desarrollo de este sistema está lejos de ser un proyecto real, pues se trata más bien de un **prototipo** funcional, de manera que sirva de base como una iniciativa para desarrollar un sistema real en un futuro. Por ello, los **límites** de este sistema residen principalmente en la

implantación del mismo en un entorno real, donde se integre con otros sistemas actuales de rastreo, como los descritos en la sección 2.4 ESTUDIO DE LA SITUACIÓN ACTUAL. Este sistema está pensado para ser un experimento donde se evalúe el uso de las tecnologías de geolocalización *Android* para detectar contactos cercanos entre usuarios y positivos, por tanto, **no** será llevado a un entorno de producción real donde tenga que convivir con el personal sanitario y los rastreadores que llevan las tareas de control de contagios.

Si se considera un entorno real, el **cliente** de este sistema sería la **administración pública** de España, pues se presentaría como una alternativa al sistema actual de rastreo para automatizar y agilizar los procesos del personal sanitario ante la pandemia. La naturaleza de este cliente implica que el desarrollo debería realizarse mediante una **metodología tradicional**, como puede ser *Métrica V3*, sin embargo, como se trata de un prototipo experimental se sigue un enfoque ágil más moderno.

Con todo lo anterior, el **alcance** del sistema que se pretende desarrollar está limitado a los siguientes subsistemas:

- Una aplicación móvil desarrollada en *Android* para llevar a cabo el registro de coordenadas en local y realizar las comprobaciones de contactos de riesgo.
- Una aplicación *web* que sirva de panel de control para las autoridades sanitarias de manera que se pueda controlar algunos aspectos de la configuración del sistema y visualizar estadísticas.
- Un servidor *web* que constituya el *backend* del sistema, exponiendo una *API REST* para ser consumida tanto por la aplicación móvil como por el panel de control *web*.
- Una base de datos central hospedada en la nube, donde se almacenen todos los itinerarios de los positivos notificados, así como la configuración de algunos aspectos del sistema.

De este modo, el alcance se resume en el desarrollo e integración de estos subsistemas, incluyendo las pruebas correspondientes para verificar su correcto funcionamiento y su despliegue en un entorno de producción ficticio, de manera que se pueda evaluar su comportamiento en un contexto controlado.

7.2 Requisitos del Sistema

Una vez se ha definido el sistema a alto nivel junto con el alcance del proyecto, en esta sección se describen textualmente los requisitos del sistema, organizándolos en **historias de usuario** y requisitos no funcionales. Además, se indican aspectos de la obtención de requisitos y se enumeran los **actores** y ***stakeholders*** del sistema, incluyendo diagramas de casos de uso.

7.2.1 Obtención de los Requisitos del Sistema

En otras secciones anteriores ya se expuso el dilema de la metodología ágil frente a los desarrollos tradicionales, como en la sección destinada a la planificación. En la ingeniería de requisitos ocurre exactamente lo mismo, no existe un **proceso definido** para la **obtención**, **especificación**, **refinamiento** y **gestión** de los requisitos dentro del marco de trabajo de *Scrum*, sino que estas etapas, que en un desarrollo tradicional están bien separadas, se entremezclan durante el avance de las distintas iteraciones. De este modo, el análisis de requisitos se realiza mayoritariamente en los ***sprint planning***, es decir, en la planificación previa a cada iteración, donde el dueño del producto expone sus necesidades y el equipo de desarrollo las captura para especificar las funcionalidades que se comprometen a desarrollar en ese *sprint*. También se aprecia el refinamiento de requisitos durante las reuniones de ***backlog grooming***, en donde, se refina la pila del producto, detallando las historias de usuario y reorganizándolas.

Por otro lado, la **gestión** de requisitos dentro del enfoque ágil es algo más borrosa pues no existe un proceso de **gestión de cambios** en los requisitos, sino que cualquier petición nueva por parte del cliente que afecte a la especificación del sistema se pospone para futuras iteraciones, en lugar de pasar por un proceso de evaluación de cambios.

Para obtener los requisitos, se recurre a diversas **fuentes de información** como actores, *stakeholders*, sistemas existentes…*etc.* La captura de requisitos se realiza principalmente mediante **entrevistas no estructuradas** con los diferentes actores y *stakeholders* del sistema, aunque mayoritariamente se realizan con el **dueño del producto**, que en este caso coincide con el tutor del proyecto. Previo a estas entrevistas, se recogen requisitos a partir de **documentación externa** y de los **sistemas similares** para analizar cómo estos realizan las operaciones que debe soportar el sistema a construir. Otra fuente de información utilizada para descubrir más requisitos y detallar las historias de usuario ya especificadas son los **diagramas de contexto** y de **casos de uso**, los cuales se verán más adelante.

Una vez capturados, estos requisitos sirven de base para especificar las **historias de usuario** propias de una metodología ágil, en lugar de la clásica lista numerada de requisitos con sus identificadores únicos. Es importante destacar que este proceso de obtención de requisitos no es **estático**, sino que se reitera sobre él múltiples veces y en distintos momentos de las iteraciones para tratar de obtener el máximo de información, lo más detallada posible y de la mejor calidad para redactar las historias de usuario, dada la naturaleza ágil del desarrollo.

Las siguientes subsecciones muestran las historias de usuario especificadas a lo largo de todas las iteraciones del proyecto, incluyendo una lista de requisitos no funcionales complementaria.

7.2.1.1 *Historias de usuario*

El formato de las historias de usuario ya se presentó en el capítulo de la planificación del proyecto, en la sección 6.1 HISTORIAS DE USUARIO. A modo de recordatorio, una historia de usuario es un elemento que representa una **unidad de funcionalidad** del sistema, es decir, representa una necesidad de un tipo de usuario que interacciona con el sistema para obtener un beneficio. Así, el formato de las historias de usuario tiene el siguiente aspecto:

Como [tipo de usuario] deseo [realizar una acción/funcionalidad] para [obtener un beneficio].

Esta frase constituye el **título** de la historia de usuario, pero también se incluye una **descripción** más detallada de la funcionalidad y de otros aspectos clave, además de unos **criterios de aceptación** que sirven para determinar cuándo la historia de usuario ha sido completada. Estos criterios de aceptación exponen los eventos e interacciones que realiza el usuario junto con la respuesta esperada del sistema ante esos eventos, todo ello mediante el lenguaje natural y de manera clara y concisa.

Es importante destacar que, en el contexto de este proyecto, las historias de usuario se dividen en historias de usuario **funcionales** e historias **técnicas**. Las primeras, constituyen las funcionalidades atómicas del **negocio** de cara a los usuarios, mientras que las historias técnicas están orientadas al equipo de desarrollo y hacen referencia a aspectos **arquitectónicos** y estructurales del sistema.

En los apéndices de este documento, se puede consultar en el anexo 15.4.6 una imagen del ***story mapping*** con todas las historias de usuario agrupadas por iteraciones y por características globales / *features* del sistema, aunque es difícil distinguir los textos y puede ser necesario ampliar el documento para poder ver las historias con claridad.

A continuación, se muestran las historias de usuario redactadas durante todo el desarrollo del proyecto, agrupadas por iteraciones. Para cada historia se indica su código de historia único, su título, un descripción detallada, los criterios de aceptación en forma de lista enumerada, los puntos de historia (*Story Points – SP*), que representan las horas de trabajo estimadas para completarla, el tipo de historia (técnica o funcional) y el subsistema o subsistemas que involucra.

Nota importante: Los códigos de las historias de usuario no empiezan en 1 y pueden no ser secuenciales, debido al orden de creación de las historias mediante la plataforma de *Azure*. Para mantener la trazabilidad, se ha decidido dejar los mismos códigos de historia que tienen las historias de usuario en el tablero del proyecto en *Azure.*

7.2.1.1.1 1er Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
55	12	Funcional	Aplicación móvil *Android*

Como usuario Android deseo que la aplicación registre mis coordenadas de manera periódica para detectar contactos de riesgo.

La aplicación rastreará la ubicación del usuario obteniendo actualizaciones de localización en un intervalo dado, es decir, cada cierto número de segundos se obtendrá una nueva localización. El rastreo de ubicación debe funcionar tanto en primer como en segundo plano, y también, aunque la aplicación esté cerrada o el móvil bloqueado. La aplicación deberá solicitar los permisos necesarios para acceder a la ubicación del usuario tanto con la app abierta como en segundo plano, además de comprobar que el Bluetooth del dispositivo está activado. Las coordenadas recibidas en cada actualización se deben almacenar en la base de datos local del dispositivo.

- Se obtienen actualizaciones de localización cada cierto intervalo de segundos.
- Los datos de localización deben incluir principalmente:
  - Latitud y longitud
  - Accuracy
  - Fecha y hora (timestamp)
- El registro de ubicación debe funcionar tanto en 1er plano como en 2o plano o bien con la app cerrada.
- Las localizaciones se almacenan en la base de datos local.

Código	SP (horas)	Tipo	Subsistemas involucrados
56	6	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder INICIAR/PAUSAR el registro de coordenadas desde la aplicación para tener el control sobre el rastreo de mi posición.

El usuario tendrá la capacidad de **iniciar** el servicio de localización desde la propia aplicación para que se comience a registrar sus coordenadas. Del mismo modo, también podrá **pausar** dicho servicio cuando lo solicite, también desde la propia aplicación. Al pausar el servicio, la aplicación dejará de registrar las coordenadas del usuario en la base de datos local.

- El usuario puede iniciar el rastreo de ubicación desde una opción dentro de la aplicación.
- El usuario puede pausar el rastreo de ubicación desde una opción de la aplicación.

Código	SP (horas)	Tipo	Subsistemas involucrados
57	4	Funcional	Aplicación móvil *Android*

Como usuario Android deseo que se guarden mis coordenadas en el almacenamiento local del dispositivo para mantener mi privacidad.

El sistema deberá almacenar todas las actualizaciones de las coordenadas del usuario estándar, obtenidas durante el rastreo de ubicación, en el almacenamiento local del dispositivo del usuario. Los principales datos a guardar son:

- Latitud y longitud
- Accuracy
- Fecha y hora (timestamp)

- Los datos de la ubicación del usuario se guardan en la base de datos local de la aplicación.
- Dichos datos almacenados son accesibles desde la app móvil, pero son privados y permanecen en local.

Código	SP (horas)	Tipo	Subsistemas involucrados
58	7	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder establecer una alarma periódica para que la aplicación registre mis coordenadas durante un intervalo de horas dadas.

El usuario podrá escoger una **hora de inicio** y una **hora de fin**, para que se registren sus coordenadas en ese intervalo de tiempo automáticamente. El sistema debe comprobar que la hora de inicio es anterior a la hora de fin, de no ser así, se mostrará un error al usuario indicando lo sucedido. El usuario podrá programar varias alarmas de localización a diferentes horas. Cada alarma podrá ser activada o desactivada independientemente del resto, además de poder ser eliminada. El sistema comprobará que no existen colisiones entre las alarmas de localización.

El usuario podrá deshabilitar o habilitar el rastreo automático y además podrá visualizar en todo momento las alarmas de localización que ha programado.

- La aplicación permite seleccionar una hora de inicio y otra de fin.
- Se comprueba que las fechas sean correctas (hora de inicio < hora de fin).
- Se comprueba que no hay colisiones entre alarmas de localización.
- Se inicia el servicio de rastreo de ubicación en la hora de inicio, y finaliza en la hora de fin.
- Se permite habilitar/deshabilitar cada una de las alarmas de localización programadas.
- Se permite eliminar una alarma de localización determinada.

Código	SP (horas)	Tipo	Subsistemas involucrados
67	4	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder ver en tiempo real el rastreo de localización para ver los datos de mi posición que se están registrando.

El usuario podrá visualizar los datos en tiempo real relativos al rastreo automático de su posición, concretamente se mostrarán las coordenadas y la fecha y hora de cada localización registrada. Estos datos se irán actualizando según se van obteniendo nuevas coordenadas. Los datos que se deben mostrar son los siguientes:

- Latitud
- Longitud
- Precisión (Accuracy)
- Timestamp

- Se muestran los datos de localización correspondientes a cada localización registrada.
- Los datos se actualizan en tiempo real conforme se van recibiendo nuevas actualizaciones de localización.

Código	SP (horas)	Tipo	Subsistemas involucrados
69	5	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder ver un histórico de las ubicaciones para ver las localizaciones que ha ido registrando la aplicación.			
El usuario podrá ver un histórico con las localizaciones registradas por la aplicación en función de la fecha que se seleccione. Se podrá filtrar por fechas para ver solo un subconjunto de las localizaciones registradas en el día especificado. Para cada localización filtrada se debe mostrar: • Latitud y Longitud. • Accuracy • Timestamp			
▪ La aplicación permite al usuario estándar seleccionar una fecha para filtrar las localizaciones. ▪ La aplicación muestra las coordenadas registradas en la base de datos en el día especificado en el filtro.			

Código	SP (horas)	Tipo	Subsistemas involucrados
70	5	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder eliminar las localizaciones asociadas a una fecha que hayan sido registradas para vaciar la base de datos local de mi dispositivo.			
El sistema debe permitir al usuario estándar eliminar todas las coordenadas que hayan sido registradas en una fecha determinada. El usuario podrá seleccionar la fecha concreta de las coordenadas que desea eliminar.			
▪ El sistema permite al usuario estándar seleccionar una fecha concreta para eliminar las localizaciones de esa fecha. ▪ El sistema elimina completamente las coordenadas registradas en la fecha seleccionada de la base de datos local.			

Código	SP (horas)	Tipo	Subsistemas involucrados
50	4	Técnica	Aplicación móvil *Android*
Como usuario Android deseo poder navegar a través de un menú para poder acceder a las distintas funciones de la aplicación.			
El usuario de la aplicación *Android* podrá navegar por la aplicación móvil a través de un menú de navegación de tipo ***Bottom Navigation,*** que mostrará las principales opciones y funciones de la aplicación.			
▪ Se muestra un menú en la barra inferior con las distintas opciones. ▪ Al pulsar en una opción se reemplaza el contenido del contenedor principal por el correspondiente a dicha opción.			

Código	SP (horas)	Tipo	Subsistemas involucrados
51	6	Técnica	Aplicación móvil *Android*
Definir la capa de persistencia de los datos para la aplicación Android.			
Diseñar las clases del modelo de datos para la aplicación cliente de *Android*, así como la capa con las clases de persistencia de los datos en una base de datos local del dispositivo, incluyendo la conexión a la misma.			
▪ Se ha definido todo el modelo de datos de la aplicación. ▪ Se han implementado las clases para la persistencia y conexión con la base de datos del dispositivo.			

7.2.1.1.2 2º Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
59	5	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder configurar un intervalo de tiempo para indicar cada cuánto tiempo se registra mi posición.			
La aplicación debe permitir al usuario estándar indicar el intervalo de tiempo durante el cual se registran las coordenadas. El servicio de rastreo de ubicación registrará las localizaciones aproximadamente tras cada intervalo de tiempo indicado. Este intervalo se indica en minutos y segundos y tendrá un máximo de 5 minutos y 59 segundos.			
▪ El sistema permite indicar un número de minutos y segundos para definir el intervalo de tiempo. ▪ El servicio de rastreo de ubicación registra coordenadas aproximadamente tras cada intervalo indicado. ▪ El sistema no permite indicar un intervalo más grande que 5 minutos y 59 segundos.			

Código	SP (horas)	Tipo	Subsistemas involucrados
73	12	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder configurar múltiples alarmas de localización para registrar mi ubicación de forma periódica.			
La aplicación debe permitir al usuario estándar configurar varias alarmas mediante una hora de inicio y una hora de fin. Al llegar la hora de inicio, se iniciará el servicio de localización para registrar la ubicación hasta que llegue la hora de fin. La hora de inicio debe ser anterior a la hora de fin, y además no puede haber colisiones entre alarmas, en lo que respecta a las horas de inicio y de fin. Además, el usuario podrá activar o desactivar cada una de estas alarmas de manera independiente, así como eliminarlas por completo.			
▪ Se puede añadir una alarma de localización indicando la hora de inicio y de fin del servicio de rastreo. ▪ La alarma se dispara a las horas previstas. ▪ La aplicación no permite que se superpongan alarmas ni que la hora de inicio sea posterior a la de fin. ▪ Se puede visualizar una lista con las diferentes alarmas programadas, pudiendo cancelar o eliminar cada una de ellas.			

Código	SP (horas)	Tipo	Subsistemas involucrados
74	5	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder configurar el número de metros de distancia mínimos que tiene que desplazarse el dispositivo para que se registre mi ubicación.			
La aplicación permitirá al usuario estándar seleccionar el número de metros de diferencia que tiene que haber entre una actualización de una localización y otra (desplazamiento mínimo) para que se registre en el sistema y se almacene en la base de datos local. Solo se registrarán aquellas localizaciones que estén a la distancia mínima establecida de la última localización registrada. El sistema solo permitirá un valor máximo de 10 metros para el desplazamiento mínimo.			
▪ La aplicación permite seleccionar un número de metros de distancia entre 0 y 10 metros. ▪ Solo se registrarán las ubicaciones que estén a una distancia mínima especificada en el desplazamiento mínimo.			

Código	SP (horas)	Tipo	Subsistemas involucrados
60	8	Funcional	Aplicación móvil *Android, API REST (backend)*
Como usuario Android deseo poder registrar y notificar un positivo en COVID-19 para indicar que soy un contacto de riesgo en el sistema.			
La aplicación permitirá al usuario estándar registrar un positivo en COVID-19, de tal forma que el usuario podrá notificar dicho positivo. Al notificar que ha dado positivo, se debe solicitar al usuario su consentimiento para que se suban a la nube sus localizaciones pasadas. La aplicación enviará los datos de localización del usuario que fueron registrados en los últimos días, según el periodo de infectividad (3 días por defecto) hacia el servidor del Backend para ser almacenados. Una vez recibidos en el servidor, se almacenarán en la base de datos en la nube para persistirlos en el sistema.			
▪ La aplicación permite al usuario notificar que ha dado positivo en COVID19. ▪ La aplicación recupera todas las localizaciones registradas en los últimos días (según el periodo de infectividad) y las almacena en la base de datos en la nube.			

Código	SP (horas)	Tipo	Subsistemas involucrados
76	6	Funcional	Aplicación móvil *Android, API REST (backend)*
Como usuario Android deseo poder adjuntar mis datos personales a la hora de notificar un positivo para facilitar el rastreo a las autoridades sanitarias.			
La aplicación móvil debe permitir al usuario estándar adjuntar sus datos personales a la hora de notificar un positivo, de tal forma que todas las localizaciones que se suban a la nube estarán asociadas a dicho usuario. De este modo se facilitarán los procesos de rastreo a las autoridades sanitarias. Esta funcionalidad es opcional y por tanto el usuario tiene todo el derecho de mantener sus localizaciones de manera anónima en la nube. Los datos solicitados son: • Nombre y apellidos. • DNI • N.º de teléfono • Localidad • Código postal Una vez introducidos, el sistema debe notificar al usuario sobre la cláusula de consentimiento del tratamiento, solicitando su acuerdo con la política de privacidad. Si el usuario da su consentimiento, se procede a notificar el positivo, de lo contrario se anula la operación.			

- Se le da la opción al usuario de adjuntar sus datos personales a la hora de notificar un positivo.
- Se solicitan nombre, apellidos, DNI, n.º de teléfono, localidad y código postal al usuario.
- Se notifica al usuario sobre la cláusula de tratamiento de datos y se solicita su consentimiento.
- Las localizaciones del usuario que se suben a la nube quedan registradas y asociadas a estos datos personales del usuario.

Código	SP (horas)	Tipo	Subsistemas involucrados
54	7	Técnica	*API REST (backend)*
Diseñar API REST y enrutamiento para el servidor de BackEnd.			
Definir los **endpoints** de la **API REST** para el servidor de BackEnd, el diseño de los routers y la lógica de enrutamiento. Definir estructura de paquetes y arquitectura del Backend, incluyendo la conexión entre enrutadores, controladores y repositorios.			
▪ Se ha definido la arquitectura y la estructura de paquetes para la API REST. ▪ Se han definido los principales endpoints del enrutamiento. ▪ Se ha establecido la conexión entre enrutadores, controladores y repositorios.			

Código	SP (horas)	Tipo	Subsistemas involucrados
52	8	Técnica	*API REST (backend)*
Definir la capa de persistencia para el servidor del BackEnd.			
Definir las capas de abstracción de los repositorios para la persistencia de los datos y el modelo de datos dentro del servidor BackEnd. Implementar la conexión con la base de datos no relacional en la nube. Configurar el proyecto Firebase para desplegar la base de datos Firestore en la nube.			
▪ Se ha creado un proyecto Firebase y se ha configurado la base de datos Firestore. ▪ Se ha establecido la conexión entre los repositorios y la base de datos Firestore. ▪ Se han definido las capas de persistencia del backend.			

7.2.1.1.3 3er Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
66	10	Funcional	Aplicación móvil *Android*
Como usuario Android quiero poder consultar mis coordenadas en un mapa para ver un histórico de mis itinerarios de forma más visual.			
La aplicación Android debe permitir consultar los itinerarios diarios del usuario, es decir, además del listado de coordenadas filtrado por fecha, se debe permitir visualizar las coordenadas de una fecha dada en un mapa. Las nuevas coordenadas se irán mostrando dinámicamente en el mapa según se vayan obteniendo a través del servicio de rastreo. Las coordenadas se deben representar con marcadores y además estas tienen que ser unidas mediante una línea, representando una ruta.			
▪ La aplicación Android permite visualizar las coordenadas de una fecha dada en un mapa. ▪ Las coordenadas se muestran mediante marcadores unidos con una línea. ▪ Las coordenadas se van mostrando en el mapa dinámicamente, según se van registrando.			

Código	SP (horas)	Tipo	Subsistemas involucrados
78	5	Técnica	Aplicación *web (frontend)*
Diseñar infraestructura y menú de la aplicación web.			
Definir y diseñar la infraestructura de la interfaz de la aplicación web para el panel de control, incluyendo la navegación y las opciones de menú.			
▪ Se ha definido la infraestructura del menú de la aplicación web. ▪ Se ha definido a alto nivel la navegación de la aplicación web.			

Código	SP (horas)	Tipo	Subsistemas involucrados
77	5	Técnica	Aplicación *web (frontend)*
Diseñar arquitectura de la aplicación web.			
Definir e implementar la estructura de paquetes y la arquitectura del FrontEnd para los componentes y las llamadas a la API REST.			
▪ Se ha definido la estructura de paquetes y la arquitectura general de la aplicación web.			

7.2.1.1.4 4º Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
75	5	Funcional	*API REST (backend),* Aplicación *web (frontend)*
Como administrador deseo poder configurar el periodo de infectividad para decidir cuántas localizaciones se suben a la nube.			
La aplicación web permitirá al administrador seleccionar el número de días atrás que se tendrá en cuenta a la hora de subir las localizaciones a la nube cuando se notifica un positivo en la aplicación móvil. Los clientes Android tendrán en cuenta este periodo de infectividad para enviar al Backend todas aquellas localizaciones registradas desde ese mismo día hasta el número de días atrás indicado por este parámetro. Por defecto, este número de días será 3, que se corresponde con el periodo de infectividad medio.			
▪ La aplicación web permite seleccionar el n.º de días del periodo de infectividad. ▪ Por defecto el número de días es 3. ▪ Cuando se notifica un positivo, se suben a la nube todas aquellas ubicaciones registradas desde ese día hasta el número de días atrás indicado por el parámetro del periodo de infectividad.			

Código	SP (horas)	Tipo	Subsistemas involucrados
62	4	Funcional	*API REST (backend),* Aplicación *web (frontend)*
Como administrador deseo poder configurar los metros de distancia de seguridad para controlar la precisión de las comprobaciones de contactos de riesgo.			
La aplicación web debe permitir configurar la cantidad de metros de distancia de seguridad que se tienen en cuenta para considerar que un usuario a estado "cerca" de un positivo. Este parámetro será utilizado en los clientes Android, en el algoritmo de comprobación, para realizar las comprobaciones con los contactos de riesgo, de forma que si la distancia entre alguna de las localizaciones es menor o igual a dicha distancia de seguridad entonces se tendrá en cuenta como posible contacto de riesgo.			
▪ El sistema permite modificar el valor numérico de los metros de distancia de seguridad. ▪ Esta distancia de seguridad se tendrá en cuenta a la hora de realizar las comprobaciones de contactos de riesgo.			

- Se utiliza esta distancia de seguridad para considerar si dos localizaciones están próximas en el espacio o no.

Código	SP (horas)	Tipo	Subsistemas involucrados
63	4	Funcional	*API REST (backend),* Aplicación *web (frontend)*

Como administrador deseo poder configurar el margen de diferencia temporal con un contacto de riesgo para controlar la precisión de las comprobaciones.

La aplicación web debe permitir a los usuarios administradores configurar el margen de diferencia temporal, es decir, la cantidad de tiempo de diferencia existente entre dos localizaciones (de un positivo y de un usuario) a partir de la cual el algoritmo de comprobación considera que ambas localizaciones coinciden en el tiempo. Si la diferencia de tiempo entre dos localizaciones es menor o igual a dicho margen de diferencia, entonces se considera un posible contacto de riesgo. El margen de diferencia temporal se especificará en segundos.

- El sistema debe permitir modificar el valor numérico del margen de diferencia temporal en segundos.
- Estos segundos de diferencia se tendrán en cuenta a la hora de hacer las comprobaciones de contactos de riesgo en el cliente Android.
- Se utiliza este margen de diferencia temporal para determinar si dos localizaciones coinciden o no en el tiempo.

Código	SP (horas)	Tipo	Subsistemas involucrados
80	8	Funcional	Aplicación móvil *Android, API REST (backend),* Aplicación *web (frontend)*

Como administrador deseo poder configurar los rangos de valor de los parámetros de la comprobación de contactos de riesgo para poder ajustar la precisión del cálculo del nivel de riesgo.

La aplicación web debe permitir modificar los rangos, es decir, el valor mínimo y máximo de los **parámetros de comprobación** que se utilizan en el algoritmo para detectar contactos de riesgo. Estos parámetros de comprobación permiten determinar el nivel de riesgo que tiene un contacto de riesgo, ponderando con diferentes porcentajes cada uno de estos parámetros. Los parámetros de comprobación cuyos rangos se pueden modificar son:

- **Tiempo de exposición**
- **Proximidad media**
- **Intervalo de tiempo medio (entre localizaciones)**

Los valores mínimos y máximos que se configuren para cada parámetro se tendrán en cuenta para **normalizar** el valor de estos parámetros y así calcular un nivel de riesgo.

- Se permite modificar el valor mínimo y máximo de cada uno de los parámetros de la comprobación.
- Se comprueba que el valor máximo sea mayor que el valor mínimo.
- Estos rangos se utilizan para normalizar el valor de los parámetros a la hora de calcular el nivel de riesgo en el algoritmo de comprobación.

Código	SP (horas)	Tipo	Subsistemas involucrados
83	6	Funcional	Aplicación móvil *Android, API REST (backend),* Aplicación *web (frontend)*

Como administrador deseo poder configurar los porcentajes de peso de los parámetros de comprobación para poder ajustar la ponderación en el nivel de riesgo resultante de un contacto.

El sistema debe permitir a los usuarios administradores configurar los **porcentajes de peso** que se utilizan para ponderar los valores de los parámetros de comprobación para determinar el **nivel de riesgo** de un contacto en el algoritmo de comprobación. La suma de todos los porcentajes de peso debe ser del 100 %, de lo contrario el sistema no dejará modificar los pesos. Cada uno de los parámetros de comprobación pondera dicho porcentaje de peso en el nivel de riesgo resultante.

- Se permite modificar el porcentaje de peso de cada uno de los parámetros de comprobación.
- El sistema valida que la suma total de los porcentajes sea del 100 %.
- Se utilizan los porcentajes de peso establecidos para ponderar los valores de cada uno de los parámetros de comprobación y así calcular el nivel de riesgo.

Código	SP (horas)	Tipo	Subsistemas involucrados
79	4	Funcional	Aplicación móvil *Android*

Como usuario Android deseo ser notificado sobre los términos, condiciones y políticas de privacidad a la hora de notificar un positivo añadiendo mis datos personales para poder aceptarlos o rechazarlos.

La aplicación Android debe mostrar los términos y condiciones relativos a la protección de datos personales del usuario en lo que respecta a lo establecido en la LOPD/GDD de 2018, cuando este notifique un positivo y adjunte sus datos personales. Se debe mostrar una **cláusula de consentimiento** del tratamiento de los datos personales, de forma que se permita al usuario Android aceptar o rechazar la política de privacidad. Si la rechaza no se podrán subir las localizaciones junto con sus datos personales. Si la acepta se subirán a la nube las localizaciones junto con sus datos personales. Se debe permitir al usuario consultar la **política de privacidad** relativa a la protección y al tratamiento de sus datos personales.

- El sistema muestra la cláusula de consentimiento de protección de datos cuando el usuario notifica un positivo adjuntando sus datos personales.
- Se permite aceptar o rechazar la política de privacidad.
- Si se acepta, se completa la notificación del positivo con éxito.
- Si no se acepta, no se suben las ubicaciones a la nube.
- El sistema permite consultar la política de privacidad de la aplicación.

Código	SP (horas)	Tipo	Subsistemas involucrados
81	8	Funcional	Aplicación móvil *Android*

Como usuario Android deseo visualizar el nivel de riesgo de un contacto para conocer la gravedad del posible contacto con un positivo.

La aplicación Android debe calcular el **nivel/puntuación de riesgo** de un contacto detectado durante la comprobación de contactos de riesgo, de forma que los contactos sean clasificados según su gravedad. El nivel de riesgo se debe calcular en base a **tres parámetros de comprobación:**

- **Tiempo de exposición:** es el nº de minutos y segundos aproximado durante los cuales se ha mantenido contacto cercano con un positivo en Covid. A mayor tiempo de exposición mayor riesgo.
- **Proximidad media (metros):** es la distancia media en metros de cercanía entre el usuario y un

positivo a lo largo de un tramo de contacto. A menor proximidad media mayor riesgo.

- **Intervalo de tiempo medio**: es el intervalo de tiempo existente entre una localización y otra, de forma que cuanto menor sea dicho intervalo más fiable será el resutlado obtenido y por tanto el nivel de riesgo será mayor.

La aplicación Android debe calcular el valor de estos tres parámetros para cada contacto de riesgo detectado. Estos valores calculados deben ser **normalizados** para transformarlos a una escala de valores entre 0 y 1. Una vez normalizados, cada parámetro se debe **ponderar** de acuerdo con un **porcentaje de peso**, de forma que cada parámetro tendrá diferente impacto en el nivel de riesgo resultante.

- El sistema calcula el valor de estos tres parámetros de comprobación.
- Los valores de cada parámetro son normalizados en una escala de entre 0 y 1.
- Cada valor se pondera con un porcentaje de peso que tendrá un impacto en el nivel de riesgo.
- Los valores de los parámetros ya normalizados y ponderados se utilizan para calcular el nivel de riesgo del contacto.

Código	SP (horas)	Tipo	Subsistemas involucrados
64	16	Funcional	Aplicación móvil *Android, API REST (backend)*

Como usuario Android quiero poder hacer una comprobación manual de los contactos de riesgo para saber si he estado en contacto con un positivo en Covid-19.

La aplicación Android debe comparar las localizaciones del propio usuario (almacenadas en local) con las localizaciones de positivos en Covid-19 registrados en el sistema, para poder detectar los posibles contactos de riesgo. La comprobación tendrá un **alcance** en días que será configurable por el usuario Android, y que define cuántos días atrás se tienen en cuenta para realizar la comprobación de contactos.

Se debe comparar las localizaciones del usuario de los últimos días (definido por el alcance) con las localizaciones de los últimos días de todos aquellos positivos que estén registrados en el sistema. El sistema debe comparar las localizaciones comprobando la existencia tanto de **cercanía en el espacio** como de **cercanía en el tiempo.** Si se cumplen ambas cercanías, se considera un nuevo **tramo de contacto de riesgo** que estará formado por uno o más pares de localizaciones usuario-positivo. Este tramo de contacto es lo que se denomina **contacto de riesgo** el cual debe ser ponderado según su **nivel de riesgo** que se calcula en base a los parámetros de la comprobación.

El sistema debe comprobar que no se comparan las localizaciones del usuario con positivos que hayan sido notificados por sí mismo. Para ello se deben comparar los **identificadores únicos** de los positivos almacenados en **local** en el dispositivo con los identificadores de los positivos almacenados en la nube, e ignorar todos aquellos positivos cuyo identificador coincida con alguno de los almacenados en local.

- El sistema recupera las localizaciones de los últimos días (determinado por el alcance) del usuario y de todos aquellos positivos registrados en el sistema.
- Se comparan las localizaciones del usuario con las de los positivos comprobando la cercanía en el tiempo y en el espacio.
- El sistema obtiene todos los contactos de riesgo, formados por los pares de localizaciones usuario-positivo en los que existe proximidad en el espacio y en el tiempo.
- Para cada contacto de riesgo, el sistema calcula el nivel de riesgo de dicho contacto en base a los parámetros de la comprobación.
- El sistema comprueba que los positivos de la nube que se tienen en cuenta para comparar no coincidan con los positivos notificados por el propio usuario que realiza la comprobación, comparando los identificadores únicos almacenados en local con los identificadores de los positivos en la nube.

Código	SP (horas)	Tipo	Subsistemas involucrados
81	8	Funcional	Aplicación móvil *Android*

Como usuario Android deseo visualizar el nivel de riesgo de un contacto para conocer la gravedad del posible contacto con un positivo.

La aplicación Android debe calcular el **nivel/puntuación de riesgo** de un contacto detectado durante la comprobación de contactos de riesgo, de forma que los contactos sean clasificados según su gravedad. El nivel de riesgo se debe calcular en base a **tres parámetros de comprobación:**

- **Tiempo de exposición:** es el nº de minutos y segundos aproximado durante los cuales se ha mantenido contacto cercano con un positivo en Covid. A mayor tiempo de exposición mayor riesgo.
- **Proximidad media (metros):** es la distancia media en metros de cercanía entre el usuario y un positivo a lo largo de un tramo de contacto. A menor proximidad media mayor riesgo.
- **Intervalo de tiempo medio**: es el intervalo de tiempo existente entre una localización y otra, de forma que cuanto menor sea dicho intervalo más fiable será el resutlado obtenido y por tanto el nivel de riesgo será mayor.

La aplicación Android debe calcular el valor de estos tres parámetros para cada contacto de riesgo detectado. Estos valores calculados deben ser **normalizados** para transformarlos a una escala de valores entre 0 y 1. Una vez normalizados, cada parámetro se debe **ponderar** de acuerdo con un **porcentaje de peso**, de forma que cada parámetro tendrá diferente impacto en el nivel de riesgo resultante.

- El sistema calcula el valor de estos tres parámetros de comprobación.
- Los valores de cada parámetro son normalizados en una escala de entre 0 y 1.
- Cada valor se pondera con un porcentaje de peso que tendrá un impacto en el nivel de riesgo.
- Los valores de los parámetros ya normalizados y ponderados se utilizan para calcular el nivel de riesgo del contacto.

Código	SP (horas)	Tipo	Subsistemas involucrados
82	4	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder visualizar en un mapa los contactos de riesgo detectados para ver las zonas donde estuve en contacto con otros positivos.

La aplicación Android debe permitir al usuario estándar visualizar en un mapa los **tramos de contacto** con otros positivos tras ejecutar la comprobación. Se debe poder visualizar cada uno de los contactos de riesgo detectados mediante la comprobación en un mapa, quedando reflejado el **itinerario del usuario** y también el del **positivo** con el que se tuvo contacto cercano.

- Se permite seleccionar un contacto de riesgo para mostrarlo en un mapa.
- Se muestran en el mapa los tramos del itinerario del usuario y del positivo en los que se dio el contacto de riesgo.

7.2.1.1.5 5º Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
94	4	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder visualizar las localizaciones que se van registrando en un mapa para poder ver el rastreo de ubicación en tiempo real.

La aplicación móvil permitirá visualizar en un mapa los puntos/localizaciones que se van registrando periódicamente en tiempo real. El foco del mapa debe desplazarse automáticamente hasta el último punto registrado. Para cada punto se debe mostrar un marcador, y los puntos se deben ir uniendo

mediante una línea.
▪ Al comenzar el rastreo de ubicación, se muestran en el mapa las localizaciones en tiempo real. ▪ Para cada localización se muestra un marcador y se unen todos los puntos mediante una línea.ç ▪ Al detener el rastreo de ubicación, se debe limpiar el mapa, eliminando los marcadores y la línea que los une.

Código	SP (horas)	Tipo	Subsistemas involucrados
87	4	Técnica	Aplicación móvil *Android*
Refactorizar configuración local de la aplicación Android.			
Trasladar el fragmento de configuración del rastreo a los Ajustes Generales de la aplicación junto con la configuración de la comprobación de contactos. Diseñar Custom Shared Preferences para que el usuario indique los valores necesarios y estos se almacenen en el fichero de preferencias local del dispositivo.			
▪ Configuración del rastreo y de la comprobación trasladada a los ajustes generales de la aplicación móvil.			

Código	SP (horas)	Tipo	Subsistemas involucrados
95	3	Funcional	*API REST (backend),* aplicación *web (frontend)*
Como administrador deseo poder configurar el límite de notificación de positivos para limitar cuántos positivos puede notificar un mismo usuario al día.			
La aplicación web debe permitir seleccionar el **número** de positivos **máximo** que pueden ser notificados por un mismo usuario (dispositivo móvil) en un día. Este límite se tendrá en cuenta en los dispositivos móviles a la hora de notificar un positivo, comprobando previamente que no se ha superado el límite. El sistema debe persistir el valor del límite seleccionado por el administrador.			
▪ El sistema permite seleccionar un número de positivos máximo que servirá de límite. ▪ El sistema persiste este nuevo parámetro en la base de datos. ▪ El sistema comprueba que el valor introducido no es negativo o cero. ▪ La aplicación móvil tiene en cuenta este valor a la hora de notificar un positivo.			

Código	SP (horas)	Tipo	Subsistemas involucrados
99	3	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder configurar el alcance de las comprobaciones de contactos de riesgo para controlar la exhaustividad de la comprobación.			
La aplicación móvil debe permitir al usuario configurar el **número de días** que representan el **alcance** de la comprobación, es decir, es el número de días que se tienen en cuenta a la hora de comparar las localizaciones registradas en los últimos días. El algoritmo de comprobación tomará el alcance como parámetro para recuperar todas las localizaciones del usuario registradas desde el día de la comprobación hasta el número de días atrás definido por el alcance, además de las localizaciones de aquellos positivos que fueran registradas en esos días. El número **mínimo** de días del alcance debe ser de **1 día**, y el número **máximo** de días del alcance debe ser de **10 días**.			
▪ Se permite al usuario decidir el alcance de la comprobación en días en los ajustes generales de la aplicación.			

- El valor del alcance se toma como parámetro de entrada para el algoritmo de comprobación de contactos de riesgo.
- Se comprueban los límites inferior y superior de días que determinan el alcance, de forma que el usuario no pueda seleccionar un número de días fuera del rango permitido.

Código	SP (horas)	Tipo	Subsistemas involucrados
61	7	Funcional	Aplicación móvil *Android, API REST (backend)*

Como usuario Android deseo que mis coordenadas se suban a la nube una vez he notificado un positivo para que se pueda realizar el rastreo de contactos.

El sistema debe almacenar las localizaciones registradas por el dispositivo del usuario que notifica un positivo. Se almacenarán todas aquellas localizaciones registradas en los últimos días, definido por el parámetro del **Periodo de Infectividad**.

El positivo se almacenará en la base de datos en la nube con un **identificador único**, el cual servirá para comprobar que un usuario no haga comprobaciones consigo mismo, es decir, con sus mismas localizaciones. Este identificador único se almacenará en la base de datos **local** del dispositivo del usuario que notifica el positivo, de modo que cuando se ejecute una comprobación, se ignoren todos los positivos notificados por el propio usuario.

El sistema debe comprobar cuántos positivos han sido notificados en ese día mediante ese dispositivo, de forma que se **limite** el número de positivos que pueden ser notificados al día por un mismo dispositivo, en función del parámetro de **Límite de Positivos**.

- Se almacenan todas las localizaciones registradas en los últimos días definidos por el periodo de infectividad.
- Se genera un identificador único en la base de datos en la nube para cada positivo.
- Se envía el identificador único al cliente Android para ser almacenado en la base de datos local del dispositivo.
- Se limita el número de positivos que pueden ser notificados al día por un mismo dispositivo, en función del parámetro de Límite de Positivos.

Código	SP (horas)	Tipo	Subsistemas involucrados
65	6	Funcional	Aplicación móvil *Android, API REST (backend)*

Como usuario Android quiero poder configurar la periodicidad de las comprobaciones de contactos de riesgo para que se realicen de manera automática a una hora determinada.

La aplicación Android debe permitir al usuario configurar varias **alarmas de comprobación** que se dispararán a una hora determinada, de forma que a dicha hora se ejecute la comprobación de contactos de riesgo automáticamente. Estas alarmas de comprobación se repetirán de forma periódica **diariamente** a la misma hora. Además, la aplicación permitirá al usuario deshabilitar la comprobación periódica para volver a la comprobación manual, desactivándose todas las alarmas de comprobación que hubiera establecida.

La aplicación debe permitir al usuario eliminar una alarma de comprobación concreta, de forma que no se ejecute. Además, el sistema debe comprobar el **límite de alarmas** de comprobación, de forma que no se supere dicho límite a la hora de añadir una nueva alarma. Si se intenta añadir una alarma y se supera el límite, se debe notificar al usuario.

- Se permite al usuario seleccionar una hora determinada.
- Se permite al usuario añadir una alarma de comprobación.
- Se programa una alarma de comprobación a dicha hora para que se ejecute la comprobación de contactos de riesgo.
- Se repite la alarma de comprobación una vez al día, a la misma hora.

- La aplicación permite deshabilitar todas las alarmas pasando al modo de comprobación manual.
- La aplicación permite eliminar una alarma de comprobación concreta.
- La aplicación comprueba que no se supere el límite de alarmas de comprobación.

Código	SP (horas)	Tipo	Subsistemas involucrados
91	6	Funcional	Aplicación móvil *Android*
Como usuario Android deseo recibir una notificación al terminar la comprobación para poder visualizar los resultados de los contactos de riesgo.			
Una vez haya terminado la comprobación de contactos de riesgo, la aplicación debe mostrar al usuario una notificación indicando con **cuántos positivos** ha estado en contacto y cuál ha sido el **porcentaje de riesgo más alto**. El **color** de la notificación cambiará en función del nivel de riesgo del contacto de mayor riesgo. Si hubiera algún error durante la comprobación, la aplicación debe mostrar una notificación indicando que ha surgido un error en la comprobación de contactos. Al pulsar sobre la notificación, se debe mostrar la pantalla con los detalles del resultado de la comprobación realizada.			
▪ Se muestra una notificación al terminar la comprobación de contactos de riesgo. ▪ Si hay algún error durante la comprobación, se muestra una notificación indicándolo. ▪ La notificación contiene el número de positivos con los que se ha estado en contacto. ▪ La notificación contiene el porcentaje de riesgo más alto. ▪ El color de la notificación cambia en función del nivel de riesgo del contacto más peligroso. ▪ Al pulsar la notificación, se muestran los detalles del resultado de la comprobación.			

Código	SP (horas)	Tipo	Subsistemas involucrados
84	2	Técnica	Aplicación móvil *Android*
Refactorizar aplicación móvil para utilizar data binding.			
Adaptar los layouts xml de los fragmentos para que hagan uso de los BindingAdapters correspondientes y así vincular los datos directamente desde el viewmodel con los componentes de la interfaz en el layout.			
▪ Se utiliza Data Binding para vincular los datos de los viewmodels con los layouts xml.			

Código	SP (horas)	Tipo	Subsistemas involucrados
85	3	Técnica	Aplicación móvil *Android*
Refactorizar aplicación móvil para utilizar la API Navigation de Jetpack.			
Adaptar los layouts xml de los fragmentos y el código para hacer uso de la librería de Jetpack Navigation. Modelar la navegación entre fragmentos mediante el grafo de navegación, así como el paso de parámetros con SafeArgs.			
▪ La navegación de la aplicación móvil se ha modelado mediante la API de Jetpack Navigation.			

Código	SP (horas)	Tipo	Subsistemas involucrados
86	2	Técnica	Aplicación móvil *Android*
Refactorizar modelo de dominio de la aplicación Android.			
Mejorar y refactorizar el modelo de dominio de la aplicación móvil de rastreo, eliminando clases redundantes y añadiendo nuevas. Implementar mappers para transformar los objetos auxiliares de las relaciones 1 -> n de la base de datos (ROOM) a objetos del dominio.			
▪ Implementados mappers para transformar clases auxiliares de las relaciones de ROOM a clases del dominio. ▪ Refactorizado el modelo de dominio.			

Código	SP (horas)	Tipo	Subsistemas involucrados
89	2	Técnica	Aplicación móvil *Android, API REST (backend)*
Modificar la base de datos para que se almacenen milisegundos en lugar de strings.			
Cambiar la lógica de la persistencia para que se almacenen milisegundos en la base de datos en lugar de los timestamps con formato string. De este modo se evitarán muchos errores de formato de fechas. Contemplar la Zona Horaria del dispositivo en el que se ejecuta la aplicación para parsear los milisegundos a un objeto fecha.			
▪ Se almacenan los timestamps de las fechas como milisegundos en la base de datos en la nube.			

7.2.1.1.6 6º Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
104	4	Funcional	Aplicación móvil *Android, API REST (backend)*
Como usuario Android deseo poder indicar si soy asintomático y si estoy vacunado a la hora de notificar un positivo para aportar datos con objetivos estadísticos.			
La aplicación móvil debe solicitar al usuario Android los datos siguientes a la hora de notificar un positivo: • ¿usuario asintomático? con respuesta "Sí" o "No". • ¿usuario vacunado? con respuesta "Sí" o "No". El sistema debe almacenar estos datos en la base de datos junto con el positivo notificado, pues serán utilizados para cometidos estadísticos de utilidad para los usuarios administradores y sanitarios.			
▪ Se solicita al usuario Android indicar si es asintomático y si está vacunado. ▪ Se almacenan ambos datos en la base de datos junto con el Positivo notificado.			

Código	SP (horas)	Tipo	Subsistemas involucrados
100	4	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder visualizar un listado con los resultados de las comprobaciones para ver un resumen de las comprobaciones realizadas.			
La aplicación móvil debe mostrar una lista con los **resultados** de las comprobaciones ordenados por fecha. Los resultados deben aparecer **agrupados por fecha**, de forma que, para cada día, se muestren			

agrupados los resultados de las comprobaciones realizadas en dicho día.

Para cada resultado se debe de mostrar un **resumen** con los siguientes datos principales obtenidos a partir de la comprobación:

- Fecha y hora en la que se realizó la comprobación.
- Número de positivos con los que se tuvo uno o más contactos de riesgo.
- Porcentaje del nivel de riesgo más alto calculado por el algoritmo de comprobación.

- Se muestran los resultados en una lista ordenada por fecha y agrupados por día en el que se realizó la comprobación.
- Para cada resultado se muestra un resumen con los datos principales obtenidos con el algoritmo de comprobación.

Código	SP (horas)	Tipo	Subsistemas involucrados
101	6	Funcional	Aplicación móvil *Android*

Como usuario Android deseo poder ver los detalles de un resultado para obtener más información sobre los contactos de riesgo que he tenido.

La aplicación debe permitir al usuario consultar los **detalles** de un resultado obtenido a partir de la comprobación de contactos de riesgo. En los detalles de un resultado deben aparecer los siguientes datos:

- Datos generales
  - Número de positivos con los que se entró en contacto.
  - Número total de contactos de riesgo.
  - Riesgo medio en total.
  - Tiempo de exposición medio en total.
  - Proximidad media en total.
- Contactos de riesgo (**ordenados** de mayor a menor riesgo)
  - Fecha y hora de inicio y fin del contacto de riesgo.
  - Identificador del positivo al que corresponde el contacto.
  - Tiempo de exposición.
  - Proximidad media.
  - Porcentaje de riesgo calculado por el algoritmo de comprobación.

Para cada contacto de riesgo, se debe permitir al usuario visualizarlo en un **mapa,** en el cual se debe mostrar una **línea trazada** con las localizaciones del positivo durante las cuales se mantuvo el contacto, además de una **línea** que representa el camino que tomó el propio usuario durante el contacto de riesgo con el positivo.

- Se permite pulsar sobre un resultado para visualizar sus detalles.
- Se muestran todos los detalles especificados para un resultado de la comprobación.
- Se muestra un listado con todos los contactos de riesgo ordenados de mayor a menor riesgo.
- Se permite visualizar en un mapa los contactos de riesgo, dibujándolos mediante líneas que representan las localizaciones durante las cuales se mantuvo el contacto de riesgo con el positivo.

Código	SP (horas)	Tipo	Subsistemas involucrados
106	6	Funcional	Aplicación *web (frontend)*

Como administrador deseo poder visualizar un listado de los positivos que se han notificado para consultar sus datos personales y sus itinerarios.

La aplicación web debe permitir al administrador visualizar una lista con los positivos notificados. Para cada positivo se debe mostrar:

1. Código identificador del positivo (el generado por la base de datos).
2. Fecha y hora en la que se notificó el positivo.
3. Número total de localizaciones registradas.
4. Etiquetas indicadoras de positivo Asintomático y positivo Vacunado.

El administrador debe poder realizar las siguientes dos acciones:

- Pulsar sobre un positivo para ver sus datos personales (en caso de que existan).
- Pulsar sobre un positivo para ver su itinerario.

El itinerario de un positivo debe mostrarse en un mapa, dibujando una línea que una las localizaciones registradas en un día, de forma que se mostrará una línea por cada día en el que se registraran localizaciones.

- La aplicación web muestra un listado de los positivos notificados, y para cada uno de ellos muestra el código identificador y la fecha y hora de notificación.
- El administrador puede pulsar sobre un positivo para ver sus datos personales (si estos existen).
- El administrador puede pulsar sobre un positivo para ver su itinerario en un mapa.
- En el mapa se muestra una línea de color por cada día en el que se registraron localizaciones.

Código	SP (horas)	Tipo	Subsistemas involucrados
105	10	Funcional	Aplicación móvil Android, *API REST (backend),* Aplicación *web (frontend)*

Como administrador deseo poder visualizar estadísticas de la aplicación móvil para obtener una idea del uso y de algunas estadísticas de la aplicación.

La aplicación web debe permitir al usuario administrador consultar las siguientes estadísticas sobre la aplicación móvil:

- **Positivos notificados en los últimos días.**
  - **Positivos asintomáticos** y **no asintomáticos.**
  - **Positivos vacunados** y **no vacunados.**
- **Número de descargas de la aplicación.**
  - Estas descargas son virtuales, es decir, no simbolizan una descarga real, sino que es un indicador de que la aplicación se ha abierto por 1a vez en un dispositivo móvil.
- **Número de comprobaciones realizadas en los últimos días.**
  - **Porcentaje de riesgo medio calculado entre dichas comprobaciones (2 decimales).**
  - **Tiempo de exposición medio calculado entre dichas comprobaciones (minutos y segundos).**
  - **Proximidad media calculada entre dichas comprobaciones (3 decimales en metros).**

La aplicación web debe permitir al administrador, **filtrar** los resultados de las estadísticas por diferentes **rangos de tiempo**:

- **Hoy**: se muestran las estadísticas del día de Hoy.
- **Ayer**: se muestran las estadísticas desde hoy hasta el día de Ayer.

- **Hace dos días:** se muestran las estadísticas desde hoy hasta hace dos días.
- **Hace tres días.**
- **Hace una semana.**
- **Desde el origen de los tiempos**: se muestran las estadísticas calculadas en base a todos los datos existentes.

Las estadísticas del número de positivos asintomáticos, vacunados...etc se obtienen a partir de la base de datos, ya que los usuarios indican si son asintomáticos y si están vacunados a la hora de notificar un positivo mediante la aplicación móvil.

El número de descargas debe ser calculado por la aplicación móvil incrementando en uno la variable cuando se abra la aplicación móvil por primera vez.

El número de comprobaciones en los últimos días se obtiene de la base de datos, junto con los valores medios de riesgo, tiempo y proximidad calculados en el backend. Para ello, la aplicación móvil debe realizar una petición a la API REST de estadísticas cada vez que se termina de realizar una comprobación, para registrar los resultados de esta. De cada comprobación, se registra:

- Timestamp de realización de la comprobación.
- Porcentaje de riesgo medio entre todos los contactos de riesgo detectados.
- Tiempo de exposición medio entre todos los contactos de riesgo detectados.
- Proximidad media entre todos los contactos de riesgo detectados.

- La aplicación móvil incrementa las variables globales en la base de datos del número de descargas y número de comprobaciones cuando se abre la aplicación por primera vez y cuando se realiza una comprobación exitosa respectivamente.
- La aplicación móvil realiza una petición a la API de estadísticas para almacenar los resultados de una comprobación cuando esta finaliza (se registra timestamp, riesgo medio, tiempo de exposición y proximidad medios).
- El administrador puede visualizar las siguientes estadísticas de la aplicación en función del rango de tiempo seleccionado:
  - N.º de positivos (asintomáticos y no asintomáticos, vacunados y no vacunados) notificados en los últimos días indicados.
  - N.º de descargas de la app registradas en los últimos días indicados.
  - N.º de comprobaciones realizadas junto con los valores medios de riesgo, tiempo de exposición y proximidad registrados en los últimos días indicados.

Código	SP (horas)	Tipo	Subsistemas involucrados
98	4	Técnica	Aplicación móvil Android
Mejorar la visualización de los resultados de las comprobaciones de contactos de riesgo.			
Revisar y mejorar la presentación de los datos de los resultados de las comprobaciones de contactos de riesgo, refactorizando el layout que muestra el listado de resultados, así como los layouts de los ítems que representan cada resultado. Mejorar la visualización de los **detalles** de un resultado de la comprobación. Mejorar el listado de contactos de riesgo de un resultado, así como las estadísticas generales del resultado, incluyendo los detalles concretos de cada contacto de riesgo. Todo ello implementando Data Binding para vincular los datos con las vistas (layouts).			
▪ Presentación mejorada de los datos de los resultados de comprobaciones y de sus detalles.			

Código	SP (horas)	Tipo	Subsistemas involucrados
97	3	Técnica	Aplicación móvil Android
Refactorizar algoritmo de comprobación de contactos de riesgo.			
Revisar y mejorar el algoritmo de comprobación de contactos de riesgo, implementado en el RiskContactDetector, además de refactorizar el RiskContactManager encargado de recuperar los datos de partida para ejecutar el algoritmo. Intentar simplificar al máximo posible la implementación del algoritmo para evitar código duplicado y para tratar de mejorar el rendimiento del mismo.			
▪ Refactorizado el algoritmo de comprobación implementado. ▪ Refactorizado el manager de contactos de riesgo.			

7.2.1.1.7 7º Sprint

Código	SP (horas)	Tipo	Subsistemas involucrados
103	4	Funcional	*API REST (backend)*, aplicación *web (frontend)*
Como administrador deseo poder configurar la hora determinada a la que se envían las notificaciones de positivos notificados para que sean recibidas por los clientes Android.			
La aplicación web debe permitir al usuario administrador configurar la hora concreta (horas y minutos) a la que se deberá enviar la notificación con el número de positivos notificados en ese día. El sistema debe programar una alarma diaria la cual envíe una notificación con dicho contenido a la hora establecida.			
▪ El usuario administrador puede indicar una hora concreta del día (horas y minutos) para que se envíe la notificación. ▪ El sistema programa una alarma diaria con la hora establecida por el administrador para enviar la notificación. ▪ La notificación con el número de positivos notificados es enviada a todos los dispositivos clientes a la hora establecida y de forma diaria.			

Código	SP (horas)	Tipo	Subsistemas involucrados
112	3	Funcional	Aplicación móvil *Android*
Como usuario Android deseo poder configurar el envío de notificaciones diarias para decidir si quiero recibir o no notificaciones del número de positivos registrados.			
La aplicación móvil debe contener una opción de configuración en el menú de Ajustes Generales que permita al usuario Android alternar entre recibir o no **notificaciones push** informando sobre el número de positivos notificados en ese día. Si el usuario habilita esta opción, recibirá una notificación diaria a la			

hora seleccionada por el administrador desde el panel de control web. Si la deshabilita, no recibirá más notificaciones hasta que se vuelva a habilitar la opción.

- La aplicación muestra una opción de configuración para habilitar/deshabilitar el recibo diario de notificaciones.
- Si el usuario habilita la opción, se reciben diariamente las notificaciones.
- Si el usuario deshabilita la opción, no se envían las notificaciones al dispositivo de dicho usuario.

Código	SP (horas)	Tipo	Subsistemas involucrados
113	4	Funcional	Aplicación móvil *Android, API REST (backend)*, aplicación *web (frontend)*

Como administrador deseo poder configurar un número de días que deben transcurrir antes de poder notificar otro positivo para limitar la notificación de positivos.

El sistema debe comprobar que han transcurrido un número de días determinado (configurables por el administrador) desde la última notificación que realizó el usuario Android para permitirle notificar un nuevo positivo. El número de días que tienen que transcurrir para poder notificar de nuevo debe ser configurable por el administrador desde la configuración en el panel de control web. Esta historia consiste en una **modificación** de la historia de usuario del límite de positivos, surgida como consecuencia de una petición por parte del **dueño del producto** respecto a la manera de limitar la notificación de positivos.

- El sistema comprueba los días transcurridos desde la última notificación cuando el usuario Android notifica un nuevo positivo.
- Si el número de días transcurridos es superior a los definidos en la configuración del panel de control web, no se permitirá al usuario notificar un positivo.
- El administrador puede configurar cuántos días deben de transcurrir desde la configuración en el panel del control web.

Código	SP (horas)	Tipo	Subsistemas involucrados
102	8	Funcional	Aplicación móvil *Android, API REST (backend)*

Como usuario Android deseo recibir notificaciones diarias para ser informado sobre el número de positivos que han sido notificados en ese día.

El sistema debe enviar notificaciones diarias a una **hora** determinada (configurable por el administrador) que serán recibidas por todos los dispositivos clientes que tengan habilitado el recibo de notificaciones. Estas notificaciones deben mostrar el **número de positivos** que han sido notificados en el día actual y se deben repetir diariamente a la misma hora especificada.

- Se envía una notificación a todos los dispositivos clientes a una hora determinada de forma diaria.
- La hora a la que se envía la notificación viene determinada por la configuración almacenada en la base de datos.
- La notificación contiene el número de positivos que se han notificado actualmente en ese día.

Código	SP (horas)	Tipo	Subsistemas involucrados
108	10	Técnica	Aplicación móvil *Android*
Diseñar e implementar pruebas unitarias y de integración de componentes para el cliente Android.			
Diseñar las pruebas unitarias y de integración entre componentes para la aplicación móvil Android.			
▪ Diseño de las pruebas de la aplicación móvil realizado. ▪ Pruebas implementadas y ejecutadas. ▪ Resultados obtenidos documentados.			

Código	SP (horas)	Tipo	Subsistemas involucrados
109	10	Técnica	*API REST (backend)*
Diseñar Pruebas de Integración para la API REST.			
Diseñar e implementar las pruebas de integración para los endpoints de la API REST.			
▪ Diseño realizado para las pruebas de integración de la API REST. ▪ Pruebas implementadas y ejecutadas. ▪ Resultados de las pruebas documentados.			

Código	SP (horas)	Tipo	Subsistemas involucrados
110	8	Técnica	Aplicación móvil *Android*
Diseñar Pruebas de Integración con la UI del cliente Android.			
Diseñar e implementar las pruebas de integración de la aplicación móvil con la interfaz de usuario, también denominadas pruebas del sistema, pues se debe probar el sistema de la aplicación móvil en su totalidad.			
▪ Diseño realizado de las pruebas de sistema para la UI de la aplicación móvil. ▪ Pruebas implementadas y ejecutadas. ▪ Resultados obtenidos documentados.			

7.2.1.2 *Requisitos no funcionales*

Una vez mostradas las historias de usuario que representan los requisitos funcionales del sistema, en este apartado se describen los **requisitos no funcionales** que se han podido identificar a partir de las fuentes de información consultadas. Los requisitos no funcionales son requisitos **transversales** al sistema, es decir, no están relacionados directamente con la funcionalidad y las necesidades que solicita el cliente, sino más bien con los **atributos de calidad** del sistema, restricciones de diseño, aspectos legales y de seguridad, entre otros.

Aunque es cierto que en un desarrollo ágil no se realiza una búsqueda exhaustiva de requisitos no funcionales, en el contexto de este proyecto, se ha optado por buscar algunos de los más significativos y redactarlos de manera formal mediante **formato** de **requisito**, de manera que así se complemente a las historias de usuario y se disponga de más información para garantizar la calidad del sistema.

Para identificar este tipo de requisitos, se ha tomado como referencia el estándar ***IEEE Std 830*** **[51]**, que indica unas pautas para redactar el documento de requisitos de un sistema. En él, se clasifican los requisitos específicos de un sistema en varias categorías, como requisitos de interfaz externa, requisitos de rendimiento, restricciones de diseño y otras categorías que especifica el estándar. Para este proyecto, no se realiza un documento de requisitos siguiendo este estándar, sino que solo se contempla esta clasificación de los requisitos para agrupar los requisitos no funcionales que identificados para el sistema de *Contact Tracker*. No se cubren todas las categorías de requisitos especificadas en el estándar, sino más bien algunas de ellas que encajan dentro de este proyecto.

A continuación, se listan los requisitos no funcionales identificados agrupados por alguna de las categorías especificadas en el estándar *IEEE Std 830*, mediante formato de requisito, es decir, enumerados en forma de lista y etiquetados mediante un **código** único y significativo.

7.2.1.2.1 Requisitos de interfaces externas

La siguiente lista contiene los requisitos relacionados con la comunicación del sistema con otros sistemas e interfaces.

IE1. El sistema debe comunicarse con la plataforma de *Firebase* para realizar las siguientes operaciones.

IE1.1 Leer y escribir en la base de datos en la nube de *Firestore*.

IE1.2 Enviar notificaciones *push* a los clientes *Android*.

IE2. El sistema debe comunicarse con la plataforma de *Azure* para ejecutar el despliegue continuo de la *API REST*.

IE2.1 Se empaquetará una nueva versión con cada *push* realizado en la rama *master* del servidor *web*.

IE2.2 La nueva versión generada se debe desplegar en el *AppService* de *Azure*.

IE3. El sistema debe comunicarse con los servicios de geolocalización de *Google Play Services* para obtener información sobre la posición aproximada del dispositivo, registrando los siguientes datos:

IE3.1 Latitud y longitud de la posición.

IE3.2 Fecha y hora en la que se registra la localización.

IE4. El sistema debe comunicarse con los servicios de *Google Maps* para mostrar mapas certificados del mundo.

IE5. La aplicación móvil y la aplicación *web* deben comunicarse con el servidor *web* a través de la *API REST* expuesta.

IE5.1 Las peticiones se realizarán siguiendo el protocolo de red *http*.

IE6. El sistema debe ser capaz de procesar ficheros de texto con coordenadas que representan un itinerario.

IE6.1 Los ficheros contendrán líneas con la latitud, longitud, fecha y hora de la localización.

7.2.1.2.2 Requisitos lógicos de bases de datos

La siguiente lista contiene los requisitos relacionados con el diseño de las bases de datos del sistema.

BD1. La base de datos central en la nube debe contener dos versiones del modelo de datos.

BD1.1 Una versión para el entorno real de producción donde residen los datos reales.

BD1.2 Una versión de *test* para el entorno de desarrollo y de pruebas del sistema donde se almacenan datos ficticios.

BD2. La configuración general del sistema debe estar almacenada en la base de datos central en la nube.

7.2.1.2.3 Restricciones de diseño

La siguiente lista contiene los requisitos no funcionales relacionados con las restricciones de diseño impuestas para el sistema, como limitaciones *hardware*, estándares y normas…*etc.*

RD1. La aplicación móvil de rastreo debe ser implementada para el sistema operativo *Android.*

RD1.1 La aplicación móvil debe funcionar en todos los dispositivos que tengan instalada alguna versión de *Android* comprendida entre las siguientes versiones.

RD1.1.1 *API* 24 Android 7.0 (*Nougat*).

RD1.1.2 *API* 30 Android 11.

RD1.2 La aplicación móvil debe funcionar solo para dispositivos móviles.

RD1.2.1 No estará disponible para dispositivos *smartwatch*, *Smart TV* o *Android Auto*.

RD1.3 La aplicación móvil solo debe funcionar para la orientación vertical del dispositivo.

RD2. Las ubicaciones de los usuarios que se registren mediante la geolocalización deben almacenarse en la base de datos local del dispositivo móvil.

RD2.1 Solo cuando el usuario estándar lo decida, se enviarán estas ubicaciones a la base de datos central en la nube.

RD3. La aplicación móvil debe ser afín a la normativa de protección de datos vigente en Europa.

RD3.1 El tratamiento de datos personales debe ser lícito de acuerdo con lo establecido en la LOPD/GDD española.

RD3.2 La aplicación móvil debe permitir al usuario estándar consultar la política de privacidad.

RD3.2.1 La política de privacidad debe contener las cláusulas indicadas en el nuevo *RGPD.*

RD3.3 La aplicación móvil debe solicitar el consentimiento del usuario estándar para cualquier operación que realice con sus datos personales.

7.2.1.2.4 Requisitos de seguridad

La siguiente lista contiene los requisitos no funcionales relativos a los aspectos de seguridad de sistema.

S1. El sistema debe utilizar el protocolo *https* para encriptar las peticiones de red cuando esté en el entorno de producción.

S2. Los *endpoints* (rutas) de la *API REST* deben estar protegidos mediante un modelo de autenticación basado en *tokens*.

S2.1 Solo se admitirán las peticiones de red de los clientes que contengan un *token* válido de autenticación.

S2.2 El servidor *web* comprobará si el *token* de la petición concuerda con un *token* válido enviado a un cliente.

S3. La base de datos en la nube solo puede ser accedida desde el servidor *web*.

S4. El código fuente del *APK* de la aplicación móvil debe ser reducido y ofuscado mediante el compilador *R8* para la versión *release*.

7.2.2 Identificación de Actores del Sistema

Los actores del sistema son cualquier persona, entidad, plataforma o sistema que reside fuera del sistema bajo construcción e interactúa con este. Los actores están involucrados en el entorno del proyecto ya sea porque se relacionan directamente con él (**actores primarios**) o bien porque reciben la influencia del mismo de manera indirecta (**actores secundarios**). Estos actores también se denominan ***stakeholders*,** y son una **fuente** primordial de **información** para capturar requisitos tanto funcionales como no funcionales.

Algunos de los actores identificados que interaccionan directamente con el sistema también constituyen **roles** dentro del mismo, es decir, pueden considerarse tipos de usuario que disponen de un conjunto de **operaciones** definido para cada rol, lo cual se verá en la siguiente sección.

A continuación, se describen brevemente los principales actores o *stakeholders* identificados para el sistema de *Contact Tracker*, indicando para cada uno de ellos, un código único, el nombre del actor y una breve descripción de cómo influye o se relaciona con el sistema.

Tabla 7.1. Actores y stakeholders identificados para el sistema de Contact Tracker.

Código	Actor	Descripción
A1	Usuario estándar	Representa un ciudadano que utiliza la aplicación móvil de rastreo.
A2	Administrador	Representa un usuario especial con privilegios para operar con el panel de control *web*.
A3	Personal sanitario	Cualquier profesional de la sanidad con conocimientos relativos a la pandemia.
A4	Rastreadores	Cualquier profesional certificado por el Estado o los ayuntamientos para ejercer de rastreador de contactos estrechos con positivos.
A5	Tutor	Se trata del propio tutor que supervisa este proyecto de fin de grado.
A6	Administración Pública	En un caso real, constituiría el cliente principal de este sistema.
A7	*Firebase*	Plataforma de *Google* con la que interacciona el *backend* para leer y escribir en la base de datos en la nube, así como para gestionar el envío de notificaciones *push* a los clientes *Android*.
A8	*Firestore*	Actor que hereda de *Firebase* y representa la plataforma para la base de datos documental hospedada en la nube.
A9	*FCM*	*Firebase Cloud Messaging* hereda de *Firebase* y es la tecnología para enviar mensajes *push* a los clientes *Android*.
A10	*Google Maps*	Servicios de *Google* para la visualización de mapas utilizados para representar los itinerarios de los usuarios en un mapa.
A11	*Azure DevOps*	Plataforma de *Microsoft* para las operaciones *DevOps* de CI/CD, concretamente para el despliegue continuo de la *API REST* y de la aplicación *web*.
A12	*Framework Android*	*Framework* para el desarrollo de aplicaciones nativas de *Android*. Constituye todas las tecnologías con las que interacciona la aplicación móvil.
A13	AEPD	Agencia Española de Protección de Datos que supervisa la normativa relativa al tratamiento de datos personales en medios digitales.
A14	*Radar Covid*	Sistema actualmente en producción para el rastreo de *COVID-19* mediante la tecnología *Bluetooth.*
A15	SNS	Sistema Nacional de Salud, en un futuro desarrollo de este prototipo podría ser necesario integrarse con este sistema para mantener la interoperabilidad con el sistema informático de salud actualmente en uso.
A16	*Google Play Services*	Servicios de *Google Play* para operar con los servicios de geolocalización en *Android* y obtener actualizaciones de las coordenadas de un dispositivo.

Estos actores y *stakeholders* constituyen las fuentes de información para obtener los requisitos y las consecuentes historias de usuario especificadas en la sección 7.2.1 Obtención de los Requisitos del Sistema descrita anteriormente.

7.2.3 Tipos de usuario

Algunos de los actores primarios identificados en el apartado anterior pueden derivar en **tipos de usuario** que interaccionan con la interfaz de usuario del sistema. Estos actores son personas que constituyen el usuario final al que está destinado el sistema, es decir, se relacionan directamente con el producto final.

Cada tipo de usuario dispone de un **conjunto de operaciones** que se le permite realizar a través del sistema. Además, el **nivel de destreza** requerido será distinto en función del tipo de usuario, así como los conocimientos previos necesarios para poder desenvolverse con las operaciones ofrecidas por el sistema.

Las siguientes subsecciones tratan de describir los distintos tipos de usuario identificados para el sistema de *Contact Tracker* a partir de los actores enumerados en la sección 7.2.2 IDENTIFICACIÓN DE ACTORES DEL SISTEMA.

7.2.3.1 *Usuario estándar*

El usuario estándar es aquel que interacciona con la **aplicación móvil** para el rastreo de contactos de riesgo. No requiere de autenticación para poder operar con la aplicación, por lo que también podría considerarse un usuario **anónimo**. Cualquier ciudadano de la población española que opere con la aplicación móvil es considerado un usuario estándar.

El **nivel de destreza** requerido para operar como usuario estándar es de un nivel **intermedio-bajo**, es decir, no se requiere de excesivos conocimientos técnicos ni de dispositivos móviles para poder usar la aplicación móvil. Tampoco es necesario disponer de conocimientos previos más allá de los conceptos básicos sobre la pandemia, como, por ejemplo, la distancia de seguridad, tiempo de exposición, saber lo que significa un contacto estrecho de riesgo…*etc*.

En cuanto a las **operaciones** que puede realizar este tipo de usuarios se encuentran:

- Registrar sus itinerarios de forma periódica y automática sin tener la aplicación encendida.
- Configurar algunos parámetros del rastreo como el intervalo de tiempo entre coordenadas, y de la comprobación, como el alcance.
- Programar alarmas con hora de inicio y fin para que se registre su ubicación de manera automática.
- Notificar que ha sido diagnosticado como positivo en *COVID-19*.
- Realizar una comprobación manual para comparar sus itinerarios con los de otros positivos y así detectar contactos de riesgo.
- Establecer alarmas para ejecutar la comprobación de contactos a una hora determinada.
- Consultar un histórico de sus itinerarios pudiendo visualizarlos en un mapa.

7.2.3.2 *Personal sanitario o rastreador*

Cualquier individuo que forme parte del personal sanitario de España o sea un rastreador certificado por el ayuntamiento de una comunidad autónoma, sin necesidad de disponer un título sanitario, constituye el rol de **administrador** del sistema. En un caso real, estos administradores tendrían unas credenciales de acceso especiales para poder autenticarse y acceder al panel de control *web*.

El **nivel de destreza** requerido para operar como un administrador es de un nivel **intermedio**, pues en este caso es importante que los administradores tengan un mínimo de destreza informática para operar con el panel de control *web* del sistema. En cuanto a **conocimientos previos** es importante que tengan un conocimiento **profundo** del comportamiento del virus para poder tomar decisiones en cuanto a la configuración general del sistema, como, por ejemplo, para configurar el periodo de infectividad a la hora de notificar un positivo o bien para establecer los valores de los parámetros para el algoritmo de comprobación de contactos,

como el margen de cercanía temporal y espacial. Las **operaciones** que pueden realizar este tipo de usuarios se describen a continuación:

- Configurar la notificación de positivos en los dispositivos móviles.
    - Configurar el comportamiento del algoritmo de comprobación de contactos, variando los valores de márgenes de cercanía, peso de los parámetros y rango de los valores.
    - Visualizar estadísticas de uso de la aplicación, como las descargas, comprobaciones realizadas y positivos notificados en los últimos días.
    - Visualizar los datos de los positivos que se han notificado a través de las aplicaciones móviles, incluyendo datos personales (si los hay) y los itinerarios registrados en un mapa.

7.2.4 Diagramas de contexto y de casos de uso

En el contexto de este proyecto, no se especifican casos de uso detallados ni sus escenarios derivados dada la naturaleza ágil y cambiante de la metodología seguida. Como ya se comentó antes, las fases del análisis se reparten entre las distintas iteraciones en las que se desarrolla el proyecto, de forma que no hay una primera etapa previa de análisis antes de comenzar con la implementación. Por ello, los diagramas de casos de uso se realizan con el objetivo de obtener un **mayor nivel** de **detalle** sobre lo que debe hacer el sistema y así refinar los requisitos obtenidos durante las reuniones de planificación de los *sprints*.

Estos diagramas solo son un esbozo general de las operaciones que los actores que interaccionan con el sistema pueden realizar, incluyendo también los actores que influyen o son afectados por estas operaciones. De este modo, no se describirán todos los casos de uso de manera exhaustiva ni sus escenarios ya que no es necesario en este entorno ágil, además de que sería demasiado extenso. El objetivo principal de estos diagramas es representar el **papel** que juegan los diferentes **actores** en el sistema, identificando también qué operaciones debería realizar el propio sistema en el contexto de un caso de uso.

Como se verá en las figuras de los diagramas, existen unas relaciones especiales entre los casos de uso, ***extends*** e ***includes***. La primera se refiere a la relación de extensión, es decir, que un caso de uso necesita opcionalmente a otro, habiendo distintos comportamientos. La segunda representa la inclusión, es decir, un caso de uso necesita obligatoriamente a otro caso de uso para funcionar.

En las siguientes subsecciones, se muestran los diagramas de casos de uso más significativos identificados en el sistema, pero antes, se presenta un diagrama general denominado **diagrama de contexto**. Este diagrama agrupa todos los casos de uso identificados en el sistema y los asocia con sus respectivos actores, permitiendo así establecer los **límites** del sistema bajo construcción. De este modo, las fronteras entre este sistema y el exterior quedan bien determinadas.

7.2.4.1 *Diagrama de contexto*

Este apartado presenta el diagrama de contexto para el sistema de *Contact Tracker*, incluyendo los actores y *stakeholders* identificados en el apartado DE 7.2.2 IDENTIFICACIÓN DE ACTORES DEL SISTEMA. Cabe destacar que no todos los actores identificados anteriormente aparecen en el diagrama de contexto, debido a que no interactúan directamente con las operaciones proporcionadas por el sistema.

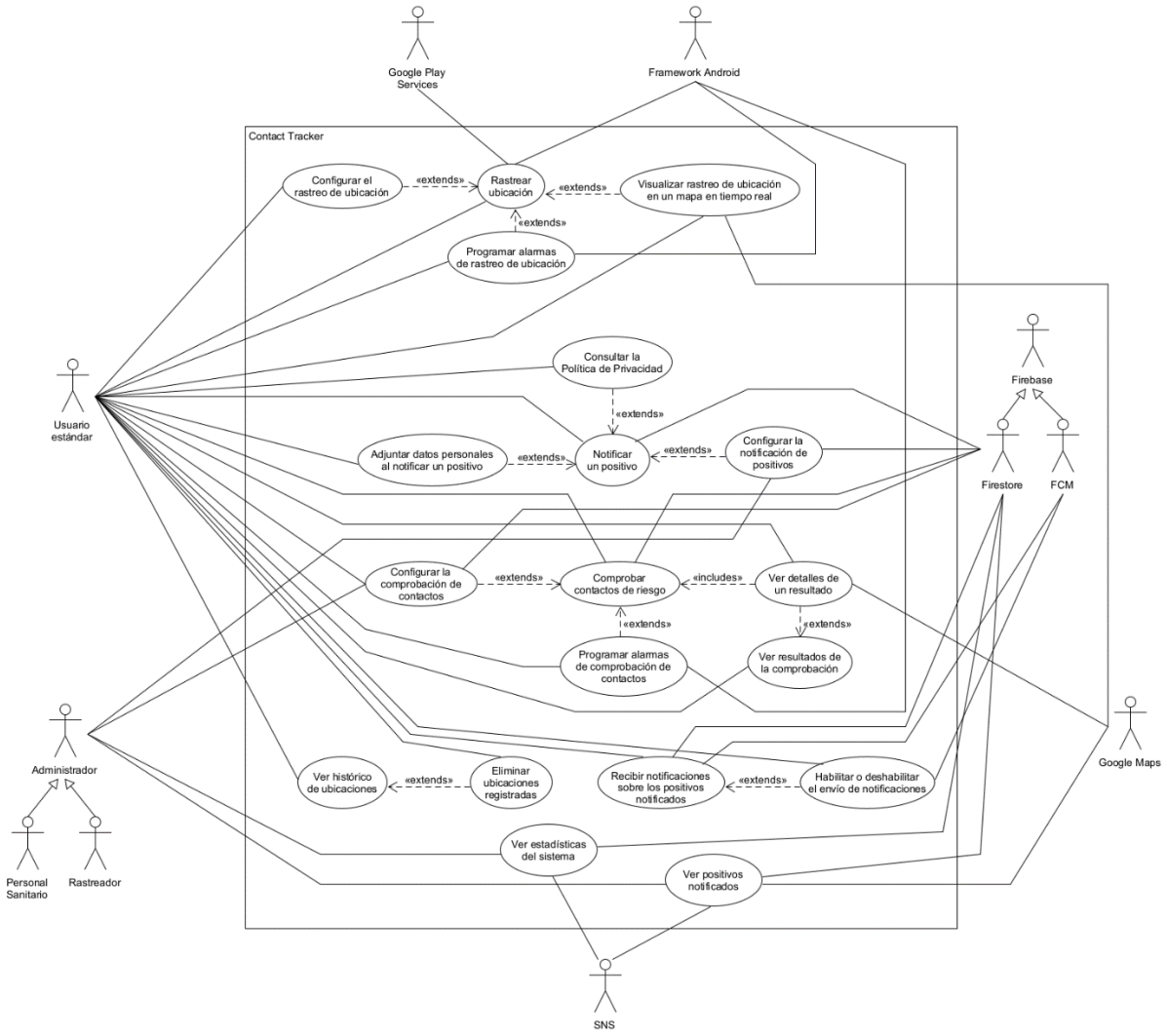


Figura 7.1. Diagrama global de contexto para el sistema de Contact Tracker.

Los **bordes** delimitan las fronteras del sistema con el exterior, donde residen los actores externos que se relacionan con los casos de uso principales identificados en el interior del sistema.

7.2.4.2 *Diagramas de casos de uso*

Para no extenderse más de lo necesario, en este apartado se desarrollan los casos de uso más significativos de los presentados en el diagrama de contexto anterior. En un proyecto real, sería necesario cubrir con un buen nivel de detalle todos los casos de uso especificados incluyendo escenarios, sobre todo si se trata de un desarrollo tradicional. En este proyecto, la

metodología ágil hace que los casos de uso sean **desplazados** por las **historias de usuario**, que ya especifican de otra manera los actores involucrados y en cierto modo los escenarios posibles, aunque como ya se comentó, también se desarrollan diagramas de casos de uso para tratar de refinar las historias de usuario garantizando una mayor calidad de los requisitos.

A continuación, se muestran los diagramas de casos de uso realizados para las principales funcionalidades detectadas en la captura de requisitos.

7.2.4.2.1 Caso de uso: Rastrear ubicación

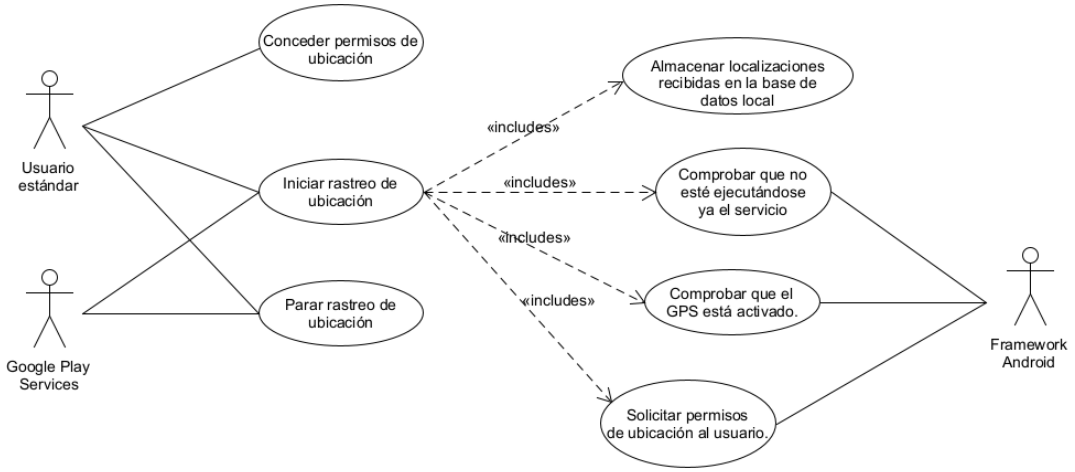


Figura 7.2. Diagrama de caso de uso para el rastreo de la ubicación.

7.2.4.2.2 Caso de uso: Notificar positivo

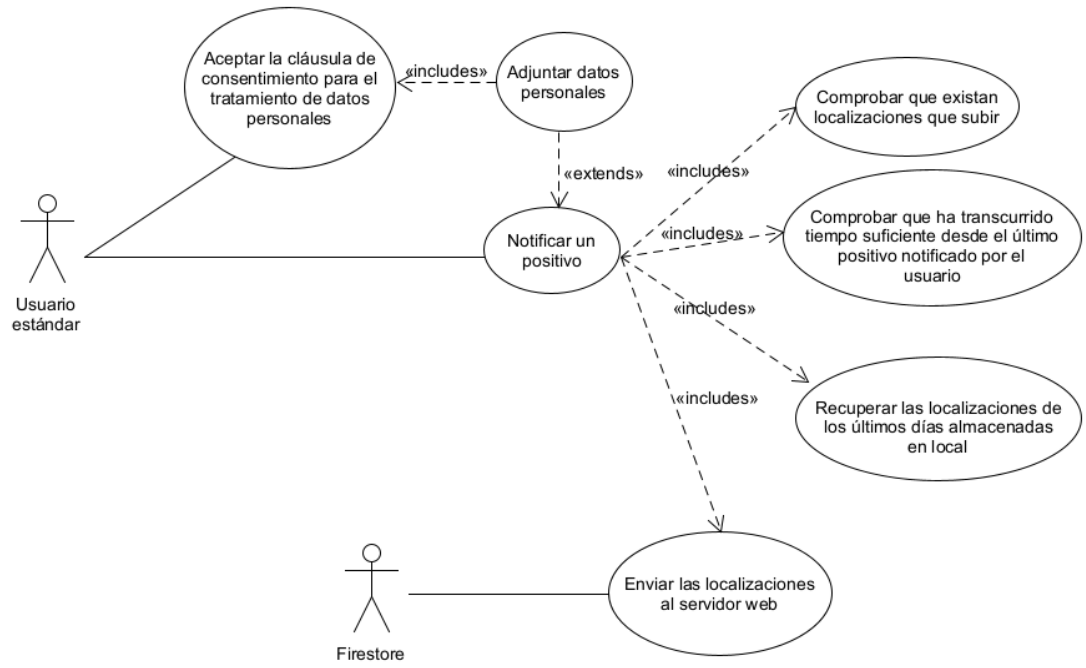


Figura 7.3. Diagrama de caso de uso para notificar un positivo.

7.2.4.2.3 Caso de uso: realizar una comprobación de contactos

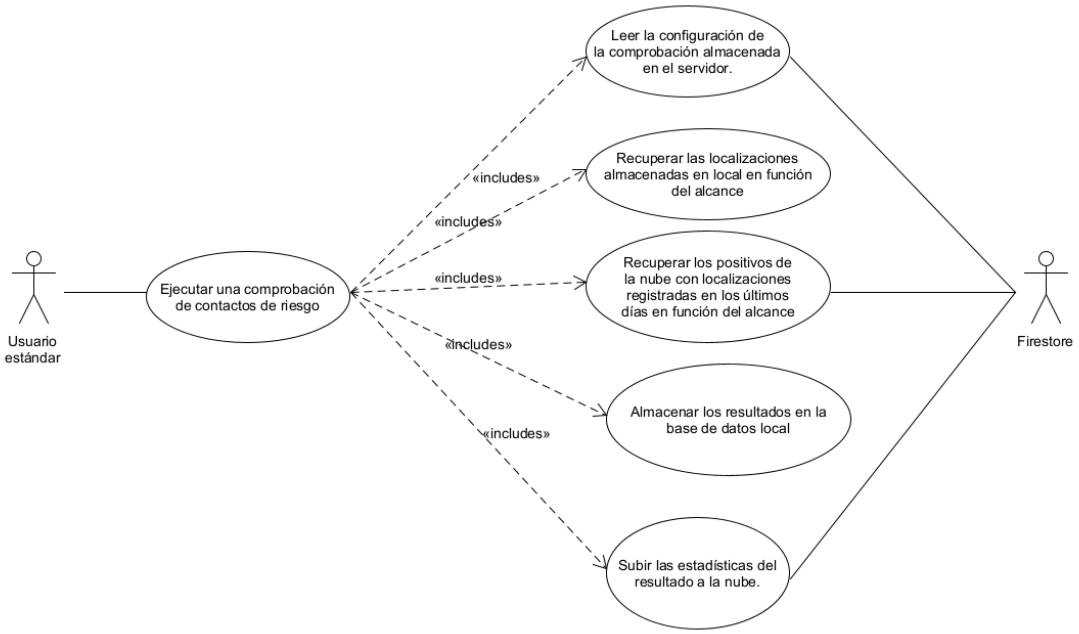


Figura 7.4. Diagrama de caso de uso para realizar una comprobación de contactos.

7.2.4.2.4 Caso de uso: Ver detalles de un resultado

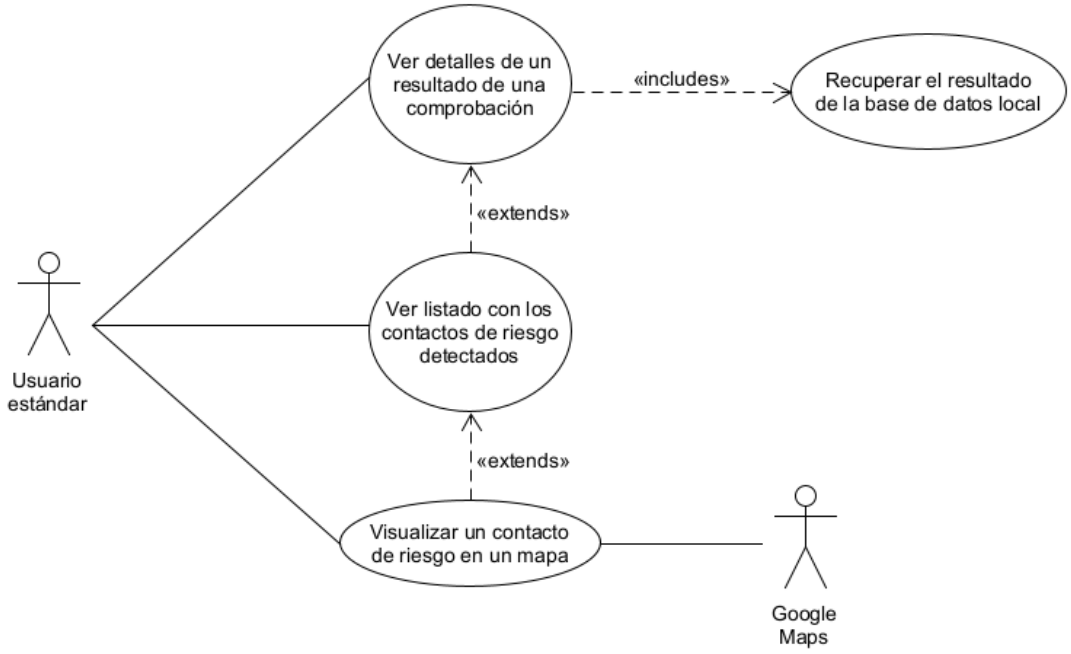


Figura 7.5. Diagrama de caso de uso para ver los detalles de un resultado de una comprobación.

7.2.4.2.5 Caso de uso: Ver positivos notificados

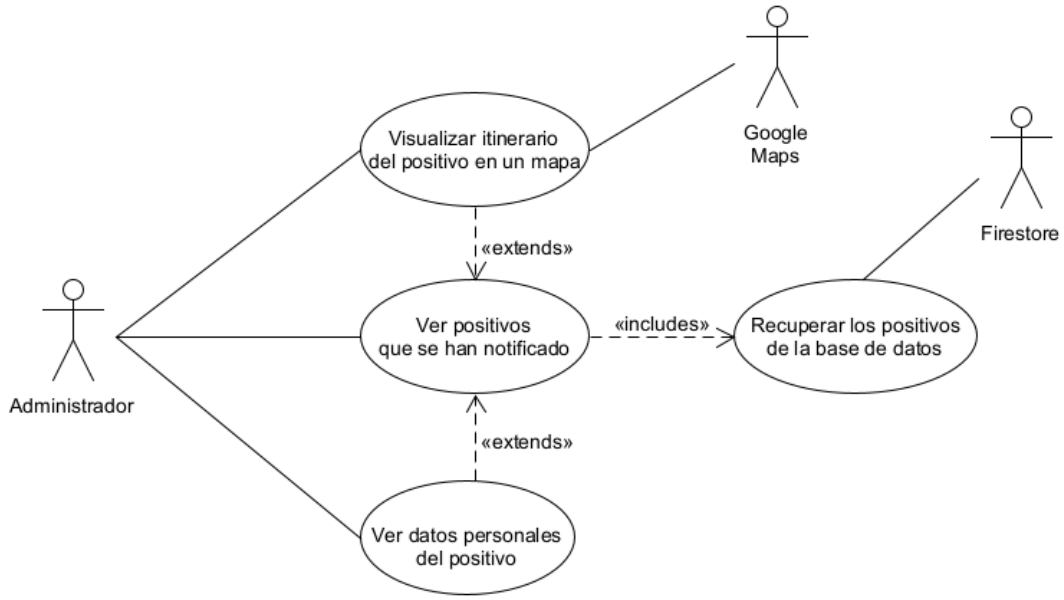


Figura 7.6. Diagrama de caso de uso para ver los positivos notificados desde el panel de control web.

7.2.4.2.6 Caso de uso: Ver estadísticas

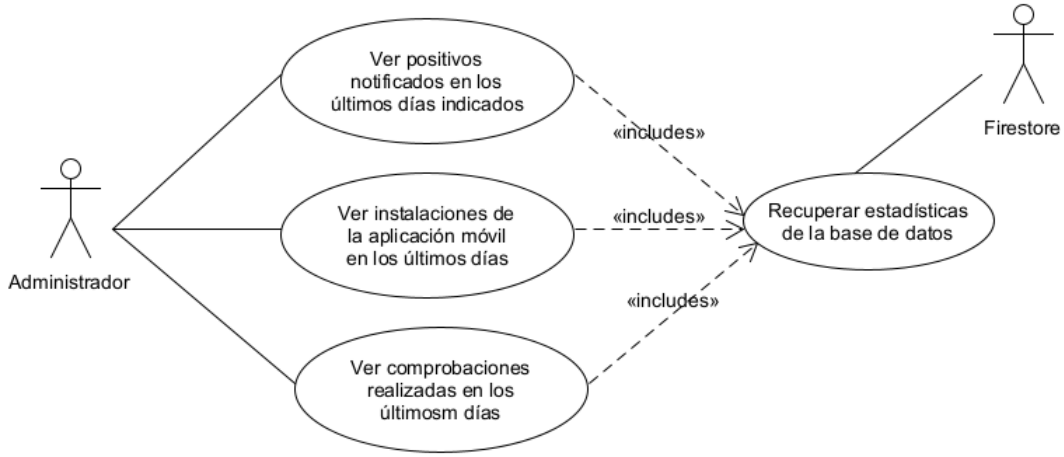


Figura 7.7. Diagrama de caso de uso para ver las estadísticas de la aplicación móvil desde el panel de control web.

7.3 Identificación de los Subsistemas en la Fase de Análisis

Tras esta primera aproximación del análisis del sistema se pueden ir apreciando los subsistemas en los que este se descompone. En esta sección se describen brevemente esos subsistemas, así como las interfaces entre ellos y la manera en la que se deben comunicar.

7.3.1 Descripción de los Subsistemas

El sistema de *Contact Tracker* puede descomponerse en **cuatro** subsistemas principales de alto nivel que interaccionan entre sí. Aunque se puede profundizar en cada uno de ellos para descomponerlos a su vez en otros subsistemas, en este caso solo se describen los subsistemas de alto nivel que constituyen el sistema en su totalidad.

7.3.1.1 *Subsistema: Aplicación móvil*

Este es quizás uno de los módulos más importantes del sistema y que constituye el grueso funcional del sistema. La aplicación móvil será el **punto de entrada** para que los usuarios registren sus itinerarios y estos se almacenen en la base de datos local, para después ser enviados al servidor cuando se notifique un positivo. Además, será la herramienta utilizada para ejecutar la comprobación de contactos de riesgo en local, comparando las coordenadas del usuario con las de otros positivos para detectar contactos estrechos. Para ello, este subsistema interacciona con el servidor *web* intercambiando diferentes datos y con su **base de datos local**, donde se almacenan entre otras cosas las coordenadas del usuario.

Esta aplicación se desarrollará en *Android* e irá destinada a los dispositivos móviles que tengan instalada alguna de las versiones de *Android* indicadas en la sección 7.2.1.2.3 RESTRICCIONES DE DISEÑO.

7.3.1.2 *Subsistema: panel de control web*

Del mismo modo que los ciudadanos disponen de la aplicación móvil para las operaciones de rastreo y comprobación de contactos, los **administradores**, normalmente personal sanitario y rastreadores, también disponen una aplicación *web* que sirve de panel de control para configurar algunos aspectos del sistema, visualizar estadísticas de uso de la aplicación móvil y consultar los positivos que se han notificado a través de la misma junto con sus itinerarios.

Este subsistema se trata de una aplicación *web* que podrá ser accedida desde los navegadores modernos utilizados típicamente, como *Mozilla* o *Chrome*, y al igual que la aplicación móvil, se comunicará con el servidor *web* para el intercambio de datos.

7.3.1.3 *Subsistema: servidor web*

Este subsistema juega el papel de **intermediario** entre los clientes *Android* y *web* y la base de datos central que está hospedad en la nube. Gestiona todas las operaciones de lectura y

escritura en esta base de datos central, respondiendo a las **peticiones *http*** enviadas desde la aplicación móvil y la aplicación *web*.

Se trata de un servidor desplegado en una máquina que escucha peticiones enviadas a un puerto determinado, y responde con una respuesta dada, tras realizar una operación interna en la base de datos. Esto asegura que la base de datos central solamente se acceda desde este servidor *web* y no directamente desde los clientes, aumentando considerablemente la **seguridad**.

El servidor *web* se comunica con la base de datos central por medio de la plataforma de *Firebase*, mientras que con la aplicación móvil y el panel de control *web* se comunica a través de una ***API REST***.

7.3.1.4 *Subsistema: base de datos central*

El último de los cuatro subsistemas identificados es la base de datos central que se encuentra hospedada en la nube, a través del servicio proporcionado por la plataforma de *Firebase*, que ya tiene integrada esta base de datos. Consistirá en una base de datos documental, al estilo de colecciones de documentos y solo podrán realizarse operaciones de lectura y escritura desde el servidor *web* descrito anteriormente.

La comunicación entre el servidor y esta base de datos central se realiza a través de la ***API*** que proporciona *Firebase* para comunicarse con su base de datos ***Firestore***. A través de esta librería de funciones, el servidor *web* puede enviar y recibir datos por parte de la base de datos.

De manera adicional, las **notificaciones *push*** también se envían desde la plataforma de *Firebase*, aunque no provienen directamente desde la base de datos, sino que se utiliza una *API* dedicada explícitamente al envío de **mensajes** entre sistemas, denominada ***Firebase Cloud Messaging***. En este caso, el envío de los mensajes se dispara por orden del servidor *web* y se envían directamente a los dispositivos *Android* que tengan instalada la aplicación cliente de *Contact Tracker*.

7.3.2 Descripción de los Interfaces entre Subsistemas

Una vez vistos los subsistemas, se procede a describir brevemente cómo son las interconexiones entre ellos, es decir, sus interfaces.

7.3.2.1 *Interfaz: aplicación móvil y base de datos local*

Aunque la base de datos local de la aplicación móvil no se considera un subsistema como tal, es interesante destacar cómo es la comunicación entre ambos. La aplicación móvil realizará operaciones de persistencia para almacenar datos en el dispositivo local, a través de una interfaz expuesta por una base de datos local *SQLite*, que es el tipo de bases de datos más

utilizado en dispositivos *Android*. En este caso, la comunicación se realiza dentro del mismo dispositivo móvil.

7.3.2.2 *Interfaz: aplicación móvil y servidor web*

El servidor *web* expone una ***API REST*** a los clientes para proporcionarles los servicios necesarios. Así, la aplicación móvil dispondrá de un módulo cliente para realizar **peticiones de red** a dicha *API REST* mediante el protocolo *http*. En este caso, la comunicación entre ambos subsistemas se realiza en remoto a través de la red, pues se encuentran en máquinas diferentes. Esta *API* contiene diferentes ***endpoints*** que representan las **rutas** accesibles desde los clientes.

7.3.2.3 *Interfaz: aplicación web y servidor web*

La interfaz entre la aplicación *web* y el servidor es exactamente igual que la de la aplicación móvil. De nuevo, la conexión se realiza en remoto a través de llamadas de red a la *API REST* expuesta por el servidor mediante el protocolo *http*. La manera en la que se realizan las peticiones y se gestionan las respuestas desde la aplicación *web* será distinta de la forma en la que lo haga la aplicación móvil, pero **ambos** clientes consumen la misma ***API REST*** expuesta por el servidor.

7.3.2.4 *Interfaz: servidor web y base de datos central*

Por último, la interfaz entre el servidor *web* y la **base de datos central** hospedada en la plataforma de *Firebase*, se basa en el *admin SDK* proporcionado por *Google* para conectarse desde un servidor *node* a los servicios de *Firebase*. De este modo, las lecturas y escrituras se realizan a través de llamadas a las funciones de la librería interna del *SDK* para ser enviadas a la base de datos documental de *Firestore*.

Del mismo modo, las notificaciones *push* se envían a los clientes *Android* que estén **suscritos** al tópico especificado para enviar las notificaciones y que tengan instalado el cliente de *Contact Tracker*, todo ello a través de la herramienta ***FCM*** comentada anteriormente y que ya se encarga por debajo de realizar las conexiones oportunas.

En este caso, es de esperar que la comunicación entre ambos subsistemas se realiza por medio de *internet* al ser máquinas remotas diferentes.

7.4 Diagrama de Clases Preliminar del Análisis

A partir de los requisitos descritos anteriormente y los diagramas de casos de uso, es posible realizar un boceto de los diagramas de clases de los subsistemas. En esta sección, se presenta un esbozo de lo que sería el diagrama del modelo de dominio para la aplicación móvil y una aproximación del diagrama de clases para el servidor *web*. Para la aplicación *web* no se realizará ningún diagrama ya que apenas carece de lógica de negocio, pues todo se resume a llamadas a la *API REST* cuyos resultados son procesados mostrando el resultado en pantalla.

7.4.1 Modelo de Dominio de la aplicación móvil

El modelo de dominio de la aplicación móvil no es excesivamente complejo, pues viendo los requisitos capturados, no existe una gran cantidad de entidades y relaciones entre dichas entidades. A continuación, se muestra un **boceto** del modelo de dominio de la aplicación móvil, el cual servirá de base para desarrollar el modelo de dominio más detallado en el capítulo de diseño del sistema.

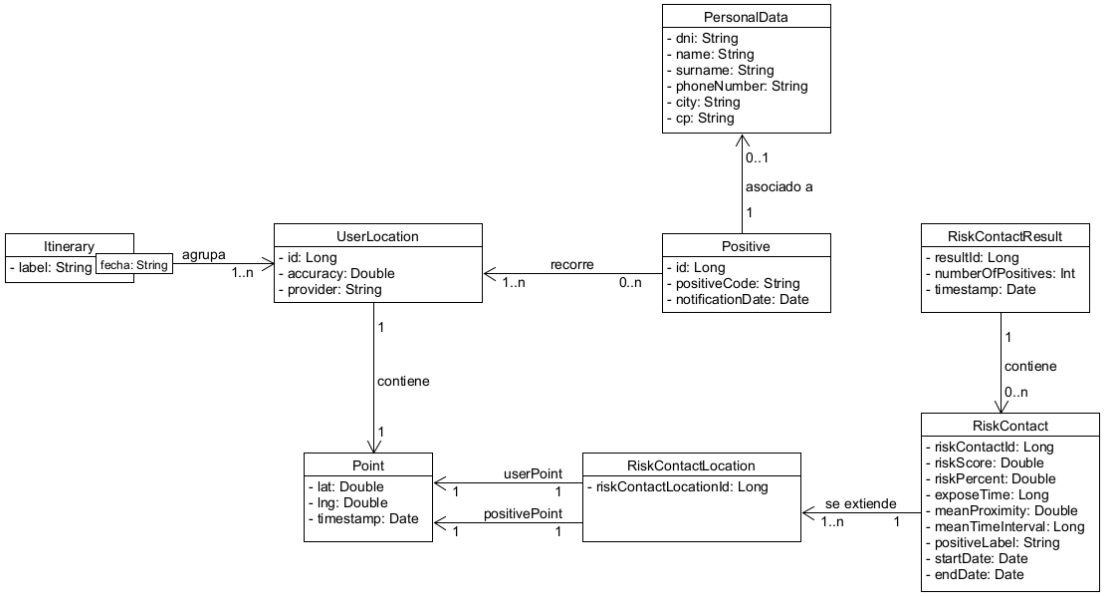


Figura 7.8. Modelo de dominio preliminar para la aplicación móvil.

Una vez visto este diagrama, se describen brevemente las clases que lo componen, indicando para cada una de ellas una descripción, sus responsabilidades y las relaciones con otras clases.

Nombre	
UserLocation	
Descripción	**Relaciones con otras clases**
Representa una localización del usuario en un mapa.	Contiene un punto que representa las coordenadas geográficas junto con la fecha y hora.
Responsabilidades	
Contiene los datos relativos a la posición de un usuario en un punto.	

Nombre
Point

Descripción	**Relaciones con otras clases**
Representa un punto geográfico en el mapa.	
Responsabilidades	
Contiene la latitud y longitud del punto, así como la fecha y hora en la que se registró.	

Nombre
Itinerary

Descripción	**Relaciones con otras clases**
Representa un itinerario seguido por un usuario a lo largo de varios días.	Contiene un mapa cuya clave es el día en el que se registraron las coordenadas y como valor una lista de localizaciones registradas en ese día.
Responsabilidades	
Agrupa las localizaciones de un usuario por días, creando un mapa de fechas asociadas a una o varias localizaciones.	

Nombre
Positive

Descripción	**Relaciones con otras clases**
Representa un positivo en *COVID-19* en el sistema.	Puede contener opcionalmente los datos personales del usuario que dio positivo. Contiene una lista de las localizaciones del usuario.
Responsabilidades	
Contiene la lista de localizaciones recorridas por el positivo registrada durante varios días, incluyendo la fecha y hora de la notificación.	

Nombre
PersonalData

Descripción	**Relaciones con otras clases**
Representa los datos personales del usuario.	
Responsabilidades	
Almacena los datos básico de DNI, nombre, apellidos…*etc* aportados por el usuario.	

Nombre
RiskContactResult

Descripción	**Relaciones con otras clases**
Representa un resultado de una comprobación de contactos.	Mantiene una lista con los contactos estrechos con otros positivos que se han detectado.
Responsabilidades	
Almacena el número de positivos con los que se entró en contacto incluyendo una lista con los contactos estrechos detectados.	

Nombre
RiskContact

Descripción	**Relaciones con otras clases**
Representa un tramo de contacto de riesgo con otro positivo, durante el cual han estado cerca en el espacio y también han coincidido en el tiempo.	Contiene una lista de pares de localizaciones que representan los puntos de contacto entre el usuario y el positivo que conforman el tramo de contacto de riesgo.
Responsabilidades	
Calcula los parámetros de tiempo de exposición, proximidad media y riesgo del contacto en base a los pares de localizaciones que representan el tramo de contacto.	

Nombre	
RiskContactLocation	
Descripción	**Relaciones con otras clases**
Representa un par de localizaciones en el mapa que se encuentran cercanas en el tiempo y espacio.	Contiene una referencia a un punto que representa la localización del usuario y otra a un punto que representa la localización del positivo.
Responsabilidades	
Almacena una localización de usuario y otra de positivo que coinciden en el tiempo y están cerca en el espacio.	

Como se puede observar, las clases del dominio del sistema no contienen excesiva **lógica de negocio**, sino que más bien juegan el papel de **contenedores de datos** para hacer fluir la información a través de las capas de la aplicación. La lógica de negocio residirá en otros componentes responsables de diferentes operaciones, como se verá en el capítulo del diseño.

7.4.2 Diagrama de Clases del servidor *web*

Una vez visto el diagrama de clases del dominio para la aplicación móvil, se presenta un diagrama general de la **estructura** del **servidor *web***. En este caso, no existe un modelo de dominio como tal, sino que más bien se trata de un flujo de información que discurre entre las clases del servidor *web*. Este diagrama de clases servirá de punto de referencia para diseñar la arquitectura del servidor *web*.

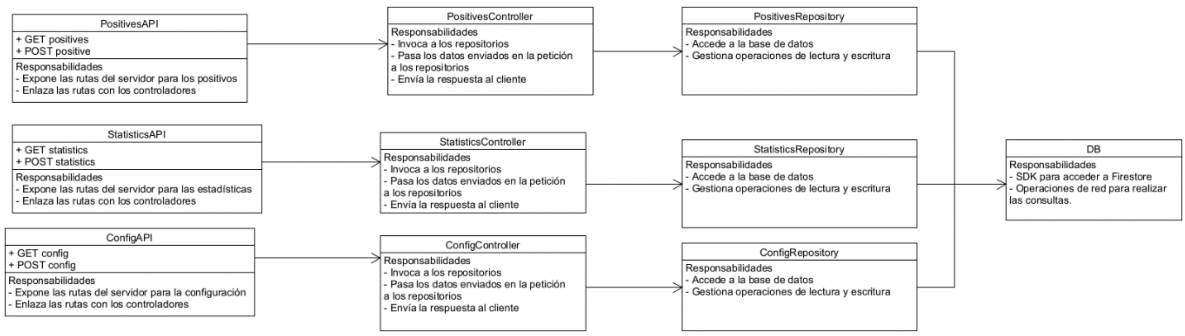


Figura 7.9. Diagrama preliminar de clases para el servidor web.

Como se puede observar, no hay nombres de clases concretos ya que aún no se ha pasado al diseño concreto del servidor *web*. Así, el objetivo de este diagrama es representar de manera global cuál será la estructura de clases que tendrá el módulo final del servidor *web* que expone la ***API REST***.

La estructura de clases es muy simple: los *endpoints* de la *API REST* se exponen a través de los ***routers***, representados por las entidades cuyo nombre termina en *API*. Estos componentes redirigen las peticiones a los **controladores**, que pueden realizar alguna transformación de los datos recibidos en la petición, y estos a su vez invocan a los repositorios con dichos datos de entrada. Los repositorios realizan las operaciones de persistencia conectándose con la base de datos en la nube, a través del *SDK* de *Firebase,* y una vez terminan, envían la respuesta a los controladores, quienes responden a los clientes con los resultados.

7.5 Análisis de Interfaces de Usuario

Tomando como referencia la información sobre **qué** hay que construir, obtenida mediante la captura de requisitos, las historias de usuario y los diagramas de casos de uso, se desarrollan los **bocetos** de la interfaz de usuario para el sistema, también conocidos como ***sketches***, en la jerga de la ingeniería de requisitos. Estos bocetos sirven de base para el posterior diseño de la interfaz de usuario que se verá en el siguiente capítulo, por lo que en este caso no contendrán mucho nivel de detalle, sino que serán más bien unos dibujos de alto nivel cuyo objetivo es proporcionar una idea general del aspecto final de la interfaz de usuario.

A continuación, se presentan los bocetos de las pantallas para la interfaz de usuario tanto del subsistema de la aplicación móvil, como de la aplicación *web.*

7.5.1 Interfaz de usuario: aplicación móvil

Las siguientes subsecciones muestran las distintas pantallas de la aplicación móvil junto con una breve descripción incluyendo su comportamiento.

7.5.1.1 *Rastrear ubicación*

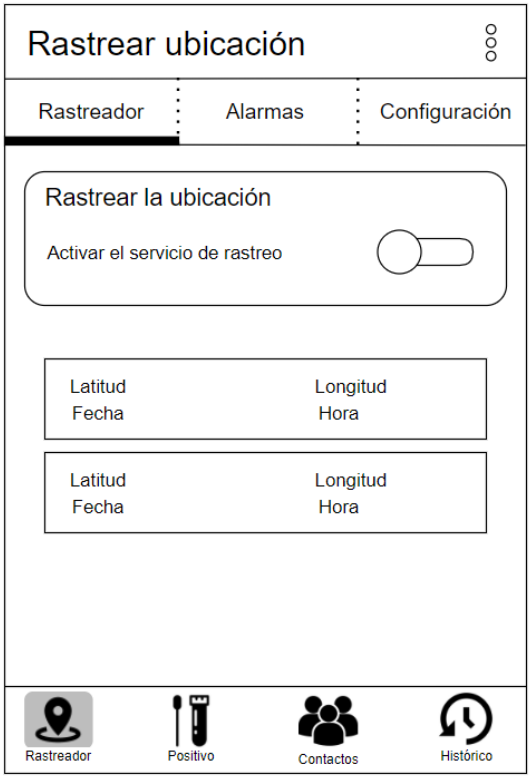


Figura 7.10. Boceto de la pantalla del rastreo de ubicación.

Esta es la pantalla principal de la aplicación móvil y la primera que se muestra al abrir la aplicación. Al activar el *switch* del servicio de rastreo de ubicación, se inicia el servicio de rastreo mostrando una notificación *Android*, tras lo cual irán apareciendo en la lista las localizaciones que están siendo registradas en tiempo real.

Cabe destacar que el *tab de* la opción de ***Configuración*** no corresponde con la versión final de la interfaz, ya que la configuración del rastreo reside en los ajustes generales como se verá más adelante.

7.5.1.2 *Alarmas de localización*

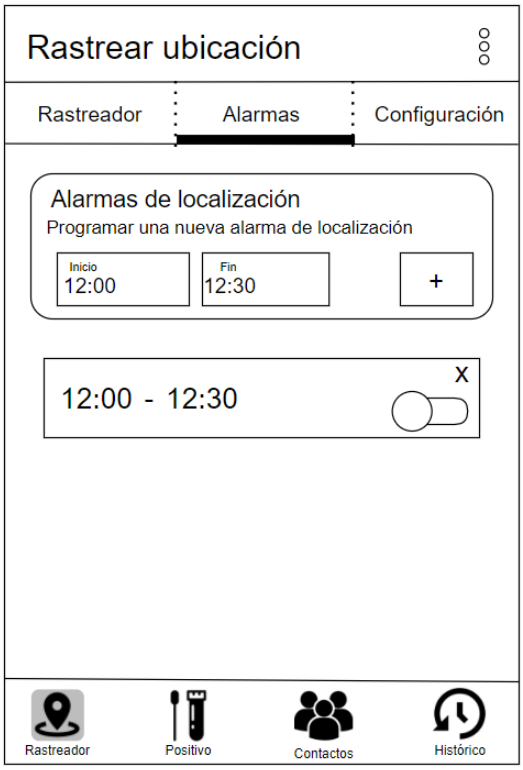


Figura 7.11. Boceto de la pantalla de alarmas de localización.

Desde esta pantalla se pueden programar alarmas de localización a unas horas determinadas. Al pulsar sobre los campos de texto, se mostrará un selector de horas para escoger la hora del día. El sistema comprobará que no existan **colisiones** entre las alarmas. Las alarmas irán apareciendo en la lista inferior, cada una con un *switch* que permitirá **desactivarla** y un icono de **cruz** para **eliminarla**.

7.5.1.3 *Notificar un positivo*

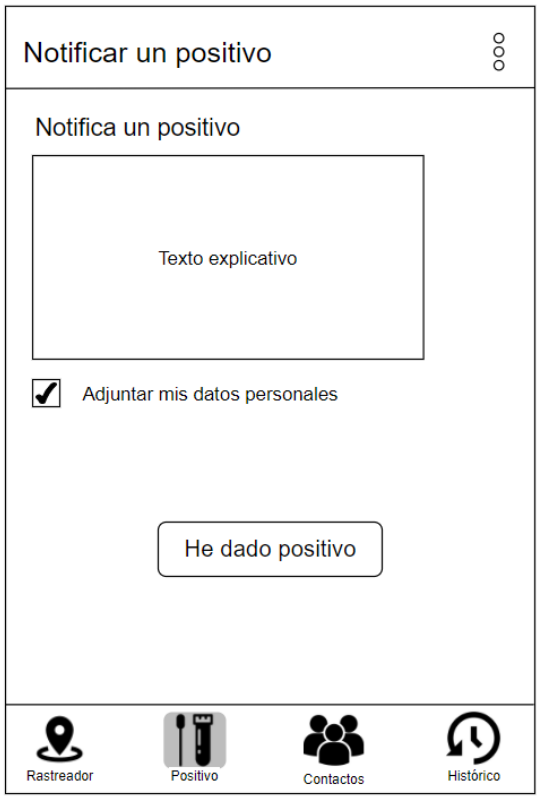


Figura 7.12. Boceto de la pantalla para notificar un positivo.

Desde esta pantalla se puede notificar un positivo en el sistema, con la posibilidad de adjuntar los datos personales, que quedarán asociados a las coordenadas que se suban a la nube. Al pulsar sobre el botón de notificar, si la casilla de datos personales está seleccionada, se mostrará un **formulario** de datos personales, como el que se muestra en el siguiente apartado.

7.5.1.4 *Adjuntar los datos personales*

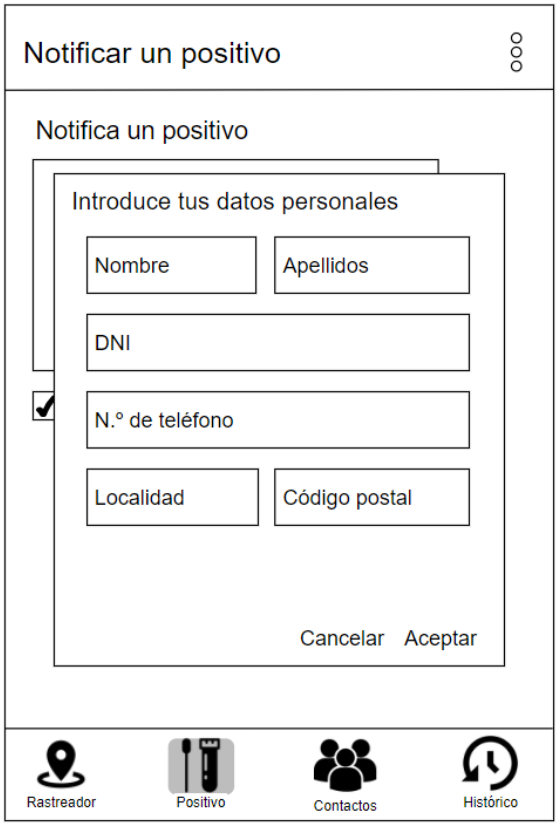


Figura 7.13. Boceto del formulario para introducir los datos personales.

Mediante este formulario, el usuario puede introducir los datos personales que se asociarán a sus coordenadas. El sistema verificará la validez de estos campos, mostrando un error explicativo en aquellos que no sean correctos.

7.5.1.5 *Comprobación de contactos*

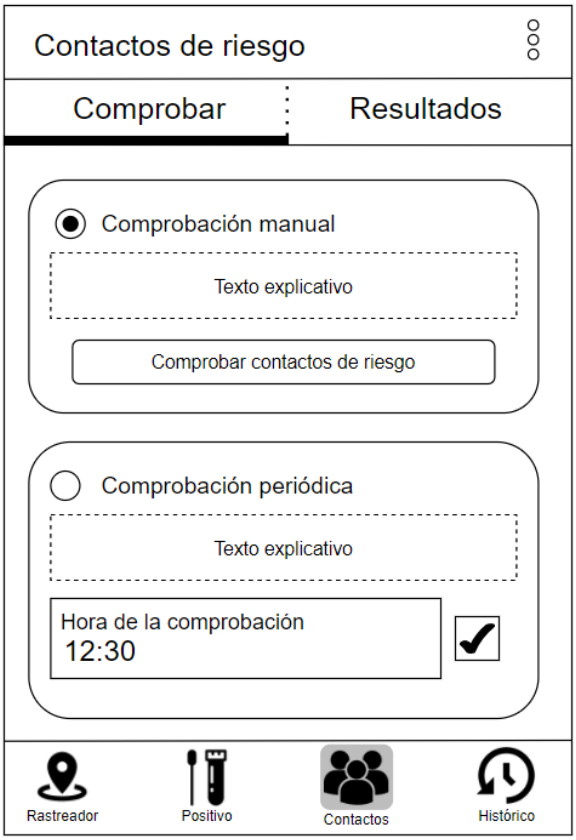


Figura 7.14. Boceto de la pantalla de comprobación de contactos.

Esta es otra de las pantallas principales de la aplicación móvil, desde la cual se pueden realizar comprobaciones para comparar las coordenadas del usuario con las de otros positivos y así detectar contactos estrechos. También se pueden programar alarmas para ejecutar la comprobación a ciertas horas del día. El sistema comprobará que no se supera el límite de alarmas de comprobación establecido en los requisitos. Además, solo se podrá tener seleccionado un modo de comprobación al mismo tiempo, quedando el otro desactivado.

7.5.1.6 *Resultados de la comprobación*

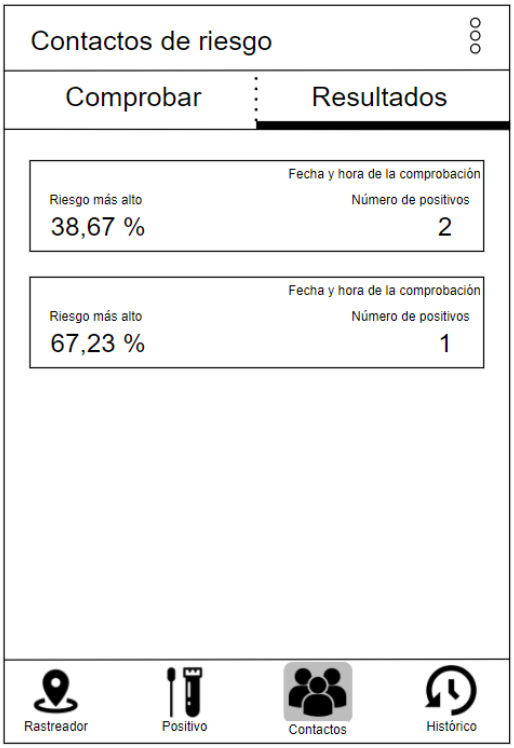


Figura 7.15. Boceto de la pantalla del listado de resultados de las comprobaciones.

Esta pantalla mostrará un listado con los resultados de las comprobaciones realizadas por el usuario, ordenadas de más reciente a más antigua. A modo de resumen, los resultados mostrarán solo el **nivel de riesgo medio** entre todos los contactos detectados y el **número** de **positivos** con los que se ha entrado en contacto. Al pulsar sobre un resultado, se muestran los detalles del mismo, como se detalla en el siguiente apartado.

7.5.1.7 *Detalles de un resultado*

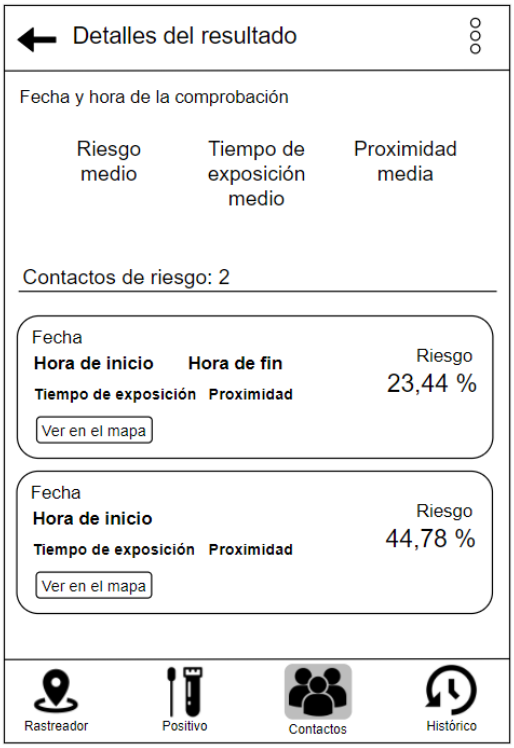


Figura 7.16. Boceto de la pantalla para ver los detalles de un resultado.

Al pulsar sobre un resultado, se muestran sus detalles, como se ve en el boceto superior. Se muestran los **valores medios** de los parámetros calculados de riesgo, tiempo de exposición y proximidad entre todos los contactos detectados. Además, se muestra un listado con los **contactos de riesgo** que se han detectado, ordenados de mayor a menor porcentaje de riesgo. Para cada uno de ellos, se muestra la fecha, la hora de inicio y fin del contacto, el tiempo de exposición, la proximidad media y el porcentaje de riesgo calculado por el algoritmo en base a dichos valores. Al pulsar sobre el botón de *Ver en el mapa*, se mostrará el contacto de riesgo en un mapa, como se verá a continuación.

7.5.1.8 *Contacto de riesgo en un mapa*

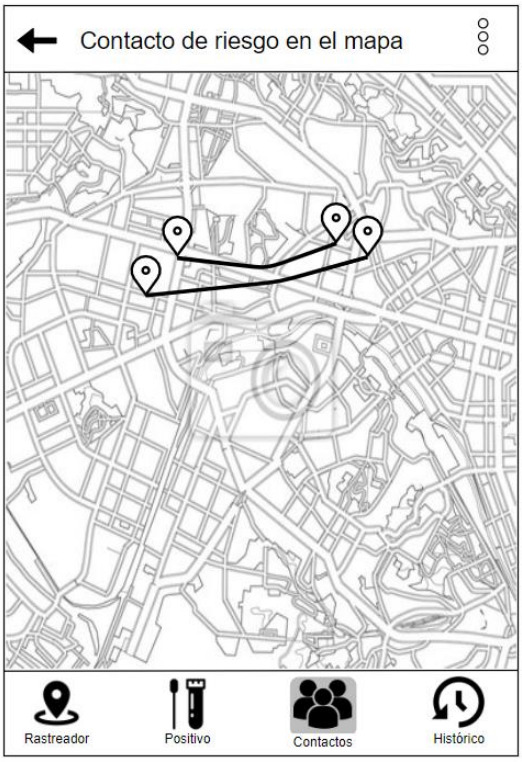


Figura 7.17. Boceto de la pantalla del mapa para ver un contacto de riesgo.

Este mapa muestra los tramos de contacto entre un usuario y un positivo cercano detectados en un contacto de riesgo.

7.5.1.9 *Histórico*

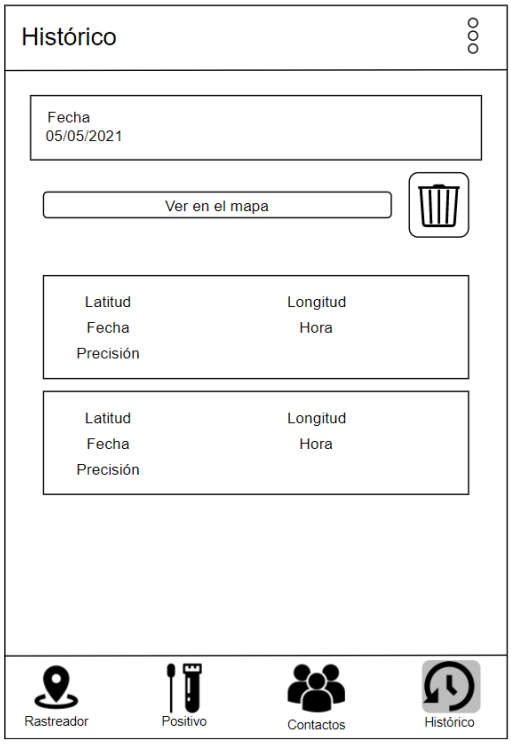


Figura 7.18. Boceto de la pantalla del histórico de localizaciones.

Esta pantalla permite visualizar un histórico de los itinerarios recorridos por el usuario a lo largo de los días. El campo de fecha permite seleccionar una fecha a través de un **calendario**, para mostrar las localizaciones registradas en ese día. Además, también se podrán eliminar dichas localizaciones a través del botón de la papelera y mostrarlas en un mapa, de manera similar al mapa de contactos de riesgo mostrado en el apartado 7.5.1.8 CONTACTO DE RIESGO EN UN MAPA. De forma adicional, al pulsar sobre una localización, se debe redirigir al usuario a la aplicación de *Google Maps* para mostrar las coordenadas en un mapa.

7.5.1.10 *Ajustes generales*

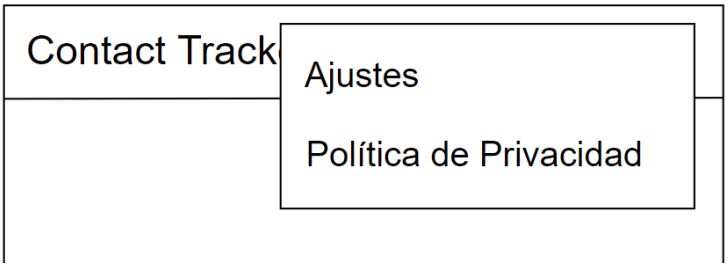


Figura 7.19. Boceto del menú superior de la aplicación móvil.

El boceto superior muestra el menú típico de las aplicaciones móviles, representado por los tres puntos en la barra superior.

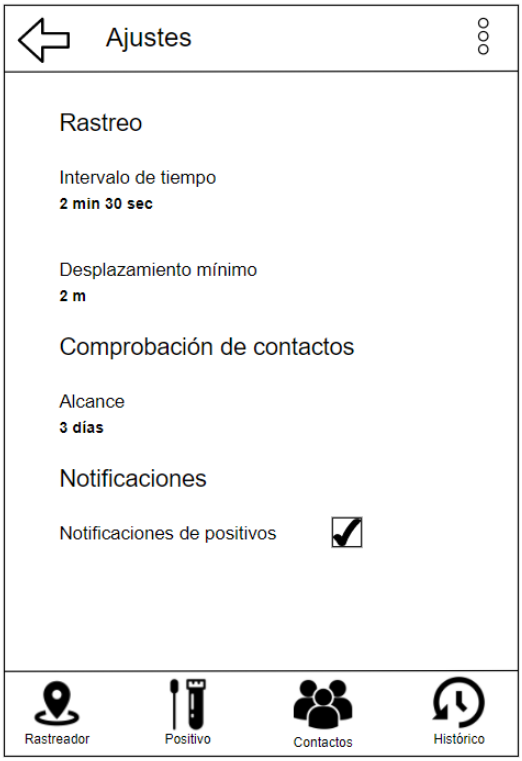


Figura 7.20. Boceto de la pantalla de ajustes generales.

Esta pantalla permitirá configurar diferentes aspectos de la aplicación móvil, como el rastreo de ubicación, el alcance de la comprobación y activar o desactivar el envío de notificaciones. Al pulsar sobre un parámetro de configuración de selección de valores numéricos, se mostrará un selector numérico como el que se muestra a continuación.

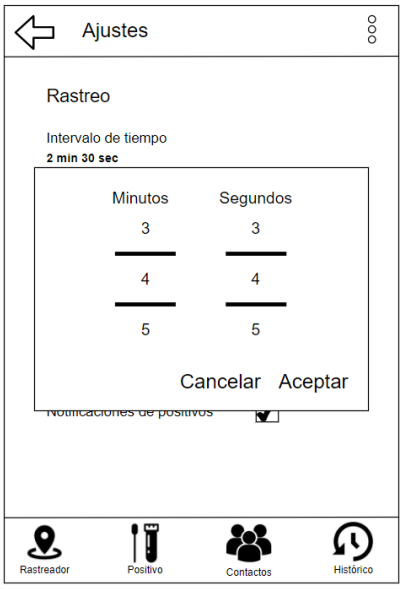


Figura 7.21. Boceto del selector numérico para los ajustes generales.

Aunque no se muestra ningún boceto dada su simpleza, desde este menú también se puede acceder a la **política de privacidad**, donde se exponen las cláusulas explicativas de protección de datos.

7.5.2 Interfaz de usuario: aplicación *web*

Las siguientes subsecciones muestran los bocetos de las pantallas que componen la aplicación *web* del panel de control para los administradores.

7.5.2.1 *Configurar la notificación de positivos*

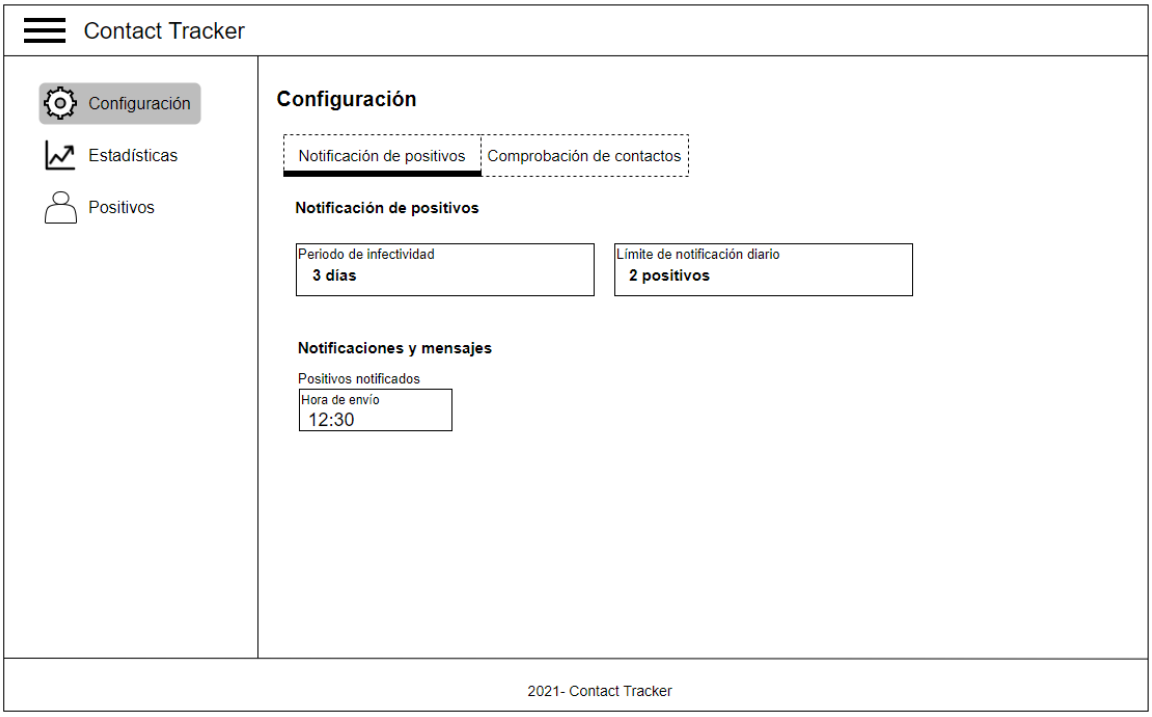


Figura 7.22. Boceto de la pantalla de configuración de la notificación de positivos.

Desde esta pantalla de la aplicación *web*, el administrador podrá configurar la notificación de positivos, incluyendo la hora de envío de notificaciones.

7.5.2.2 *Configurar la comprobación de contactos*

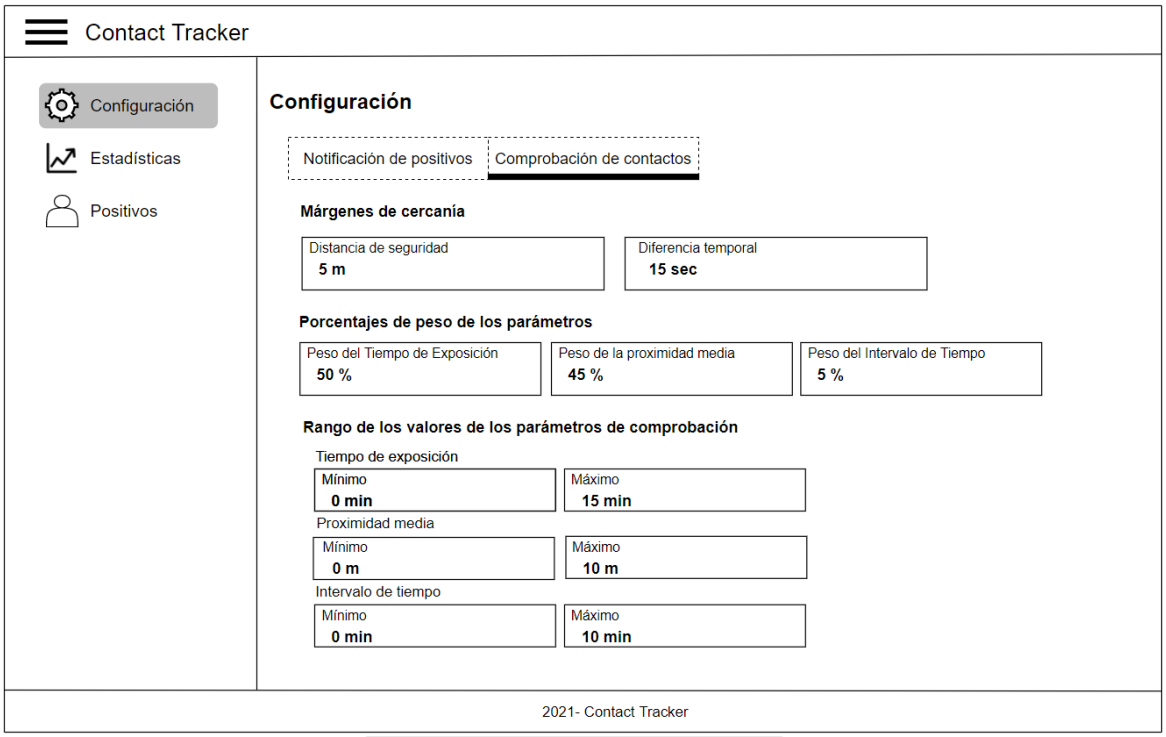


Figura 7.23. Boceto de la pantalla de configuración de la comprobación de contactos.

Esta pantalla permitirá configurar diferentes aspectos de la comprobación de contactos, como los márgenes de diferencia de tiempo y distancia o los pesos de los parámetros que ponderan en el riesgo. En este último caso, el sistema comprobará que la suma de los pesos es del 100 %.

Además, cuando se modifique algún parámetro, tanto en esta configuración como en la configuración de la notificación, expuesta anteriormente, se mostrarán dos opciones para **cancelar** o **aplicar** los **cambios**.

7.5.2.3 *Ver estadísticas*

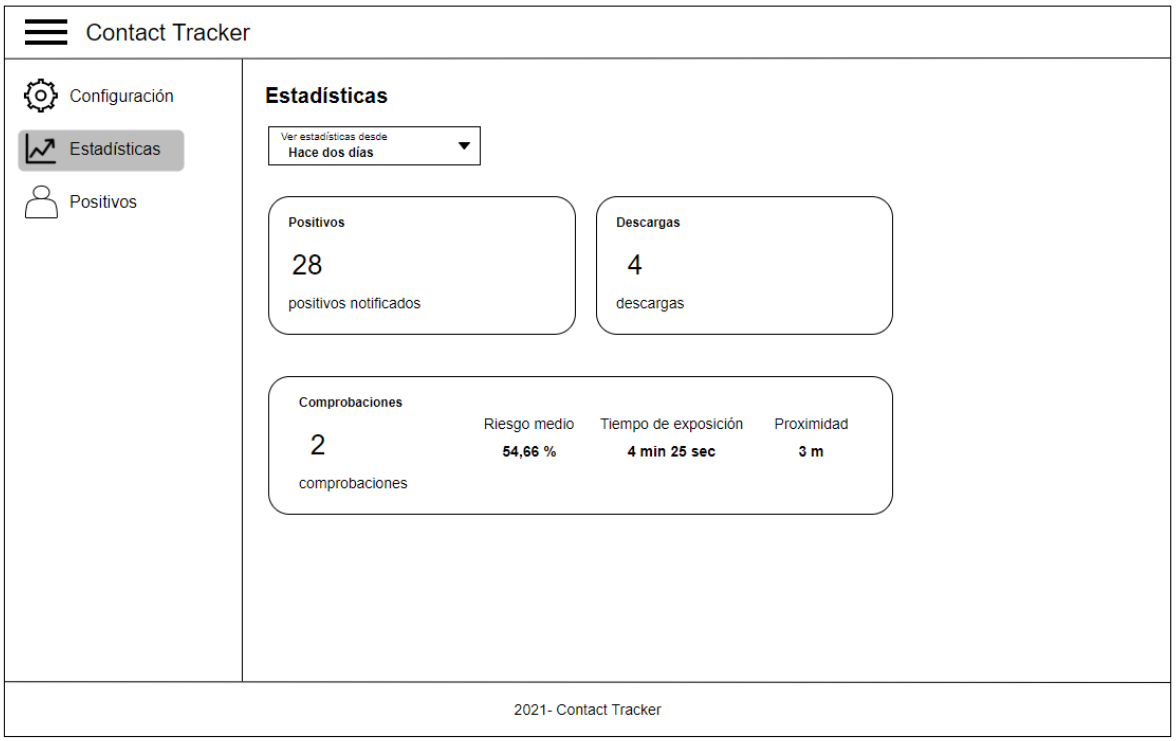


Figura 7.24. Boceto para la pantalla de visualizar estadísticas del sistema.

En esta vista, se deben mostrar características de uso de la aplicación móvil, como, por ejemplo, el número de positivos notificados, número de descargas y el número de comprobaciones realizadas. El **campo de filtro** permitirá filtrar las estadísticas por el número de días atrás desde el día actual que se tienen en cuenta para recuperar las estadísticas.

7.5.2.4 *Ver Positivos*

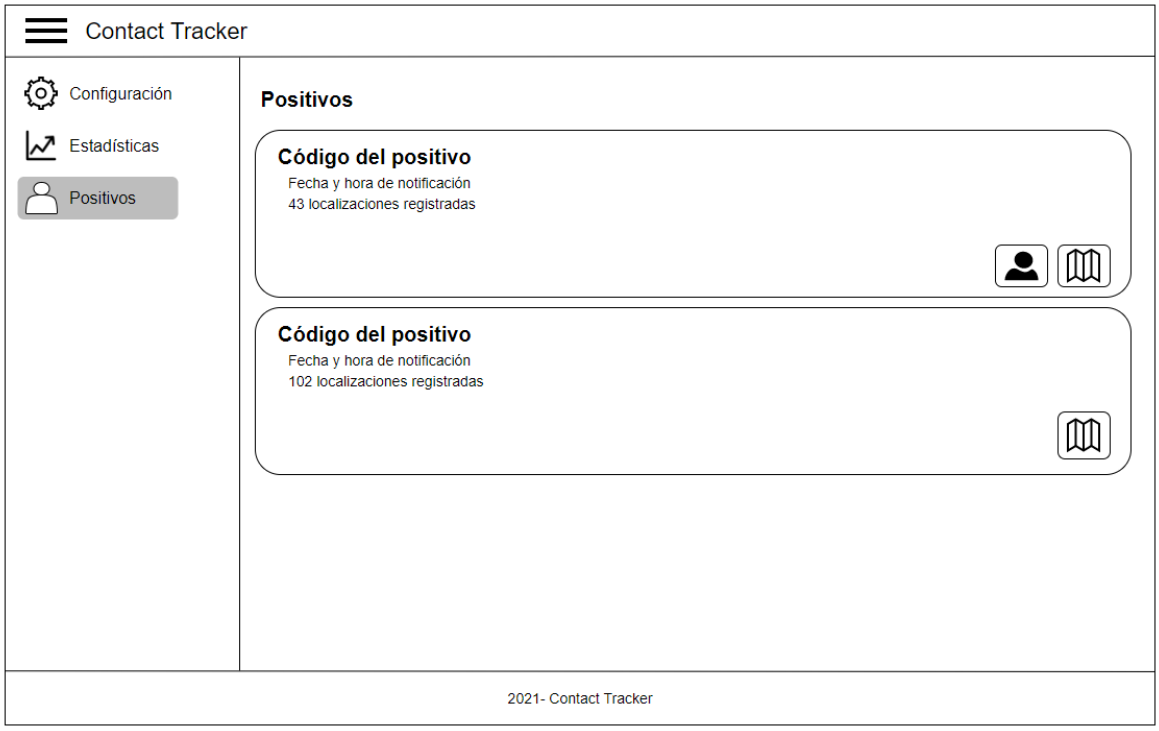


Figura 7.25. Boceto de la pantalla para ver los positivos notificados.

Desde esta opción de menú, el administrador podrá visualizar un listado con los positivos notificados en el sistema desde los clientes móviles. Para cada positivo, se mostrará su código único, la fecha y hora de notificación y el número de localizaciones que componen su itinerario. Además, para aquellos positivos que hayan adjuntado sus **datos personales,** el sistema permitirá visualizarlos en pantalla. Por otro lado, al pulsar sobre el botón del mapa, se mostrará el itinerario seguido por el positivo sobre un mapa, como se mostrará en la siguiente sección.

7.5.2.5 *Ver itinerario de un positivo*

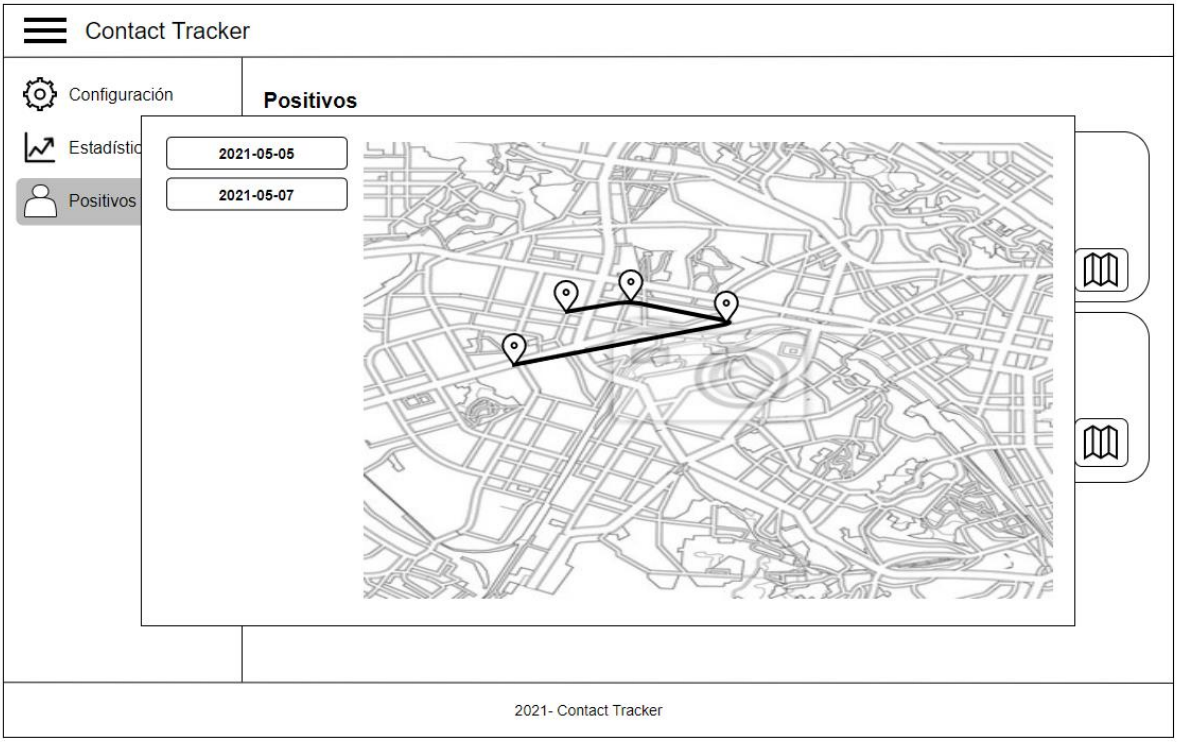


Figura 7.26. Boceto de la pantalla del mapa donde se muestra el itinerario del positivo.

Esta pantalla muestra un diálogo con un mapa donde se muestran las localizaciones registradas del positivo seleccionado. Estas localizaciones representan el itinerario del usuario distribuido entre varios días. El panel de la izquierda contiene las fechas del itinerario que se pueden seleccionar para que en el mapa se muestren las localizaciones de dicho día.

7.5.3 Diagramas de navegabilidad

Una vez vistos los prototipos de pantalla para la aplicación móvil y el panel de control *web*, aquí se presentan los **diagramas** de **navegación** para ambos subsistemas. Estos diagramas representan los **itinerarios** que puede seguir el usuario que utiliza la aplicación, es decir, la navegación a través de las diferentes pantallas expuestas anteriormente.

En primer lugar, se muestra el diagrama de navegación para la aplicación móvil de rastreo.

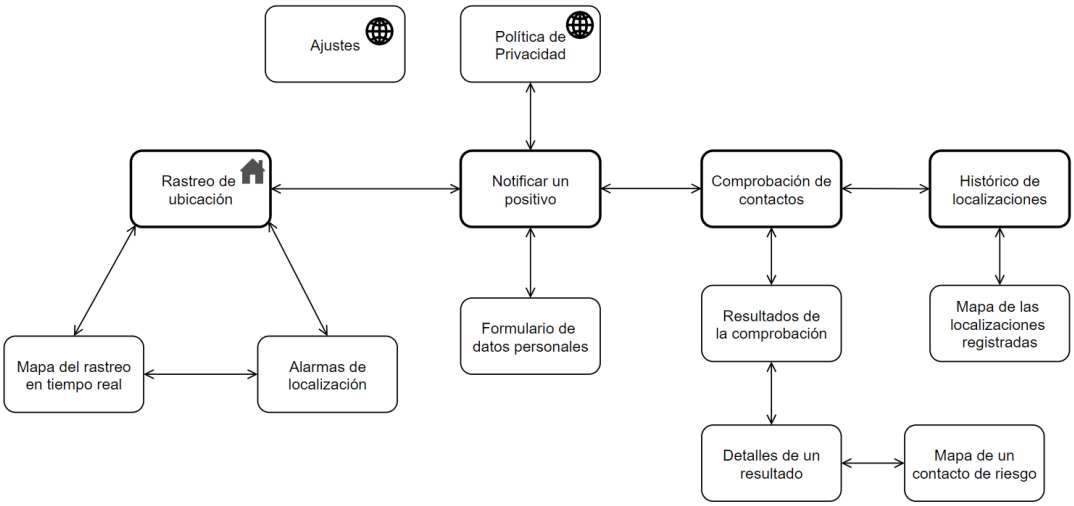


Figura 7.27. Diagrama de navegación de la aplicación móvil.

En el diagrama superior, los nodos de **borde grueso** representan las **opciones principales** del menú de la aplicación móvil, de tal forma que están conectadas todas con todas, aunque en el diagrama no esté representado así para evitar cargarlo de demasiadas relaciones y que termine siendo incomprensible. Así, desde cada opción de menú se puede navegar a cualquier otra opción. Además, el nodo etiquetado con un icono de una **casa** representa el ***home*** de la aplicación, es decir, la pantalla que se muestra por primera vez al abrir la aplicación.

Por otro lado, están los nodos del **menú complementario**, a los cuales se puede acceder desde cualquier sitio de la aplicación, por eso están etiquetados con un icono **global**. A la política de privacidad, se puede acceder desde la pantalla de notificación de positivos o bien de manera global.

Por último, se muestra el diagrama de navegación para el panel de control *web*.

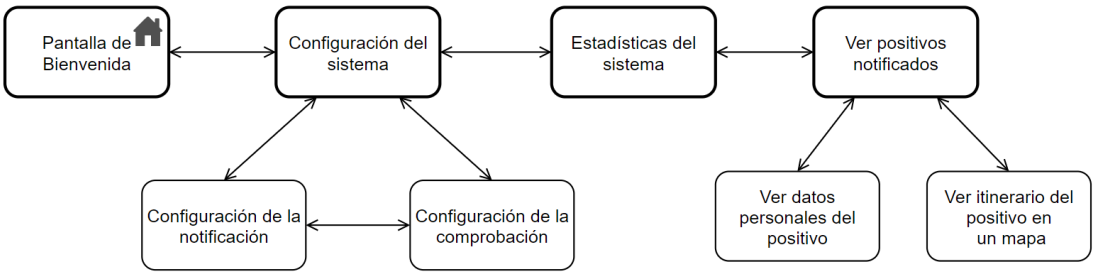


Figura 7.28. Diagrama de navegación para el panel de control web.

Como se puede apreciar, los nodos de borde grueso constituyen las opciones de menú principales de la aplicación *web*, al igual que ocurría con la aplicación móvil, de forma que están conectadas todas con todas. La pantalla de bienvenida representa la pantalla *home* de la aplicación *web*.

7.6 Especificación del Plan de Pruebas

En esta sección se presenta una primera aproximación del **Plan de Pruebas** a seguir para comprobar el correcto funcionamiento del sistema, describiendo los distintos tipos de pruebas que se contemplan, sus respectivas finalidades y sobre qué modulo o módulos del sistema se someten. En la sección 8.6 correspondiente a la ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS se especificarán los detalles **técnicos** de los distintos tipos de pruebas que se pretenden realizar junto con las herramientas utilizadas, así como los componentes concretos que serán sometidos a las pruebas.

El plan de pruebas contempla tanto pruebas internas destinadas a verificar el correcto funcionamiento de los componentes que conforman el sistema, como pruebas de cara a los usuarios finales de la aplicación.

A continuación, se describen los **cuatro tipos de prueba** a los que se someterá el sistema:

- **Pruebas unitarias.** Son pruebas destinadas a verificar el correcto funcionamiento de un módulo o componente atómico, es decir, probar que funciona como debería independientemente del resto de módulos. Esto implica aislar el componente bajo pruebas para centrar el desarrollo de las pruebas únicamente sobre dicho componente. De este modo, la base de todas las pruebas debería estar formada mayoritariamente por pruebas unitarias, según la **pirámide de *Cohn* [52]**, verificando que cada uno de los componentes funciona bien por separado. Se realizarán pruebas unitarias de los componentes fundamentales de la aplicación móvil.
- **Pruebas de integración.** Las pruebas de integración tienen como objetivo comprobar el funcionamiento entre varios componentes o módulos ensamblados, es decir, se basan en la verificación de las interacciones entre varios componentes que deben trabajar juntos. De esta forma, se comprueba que los componentes del sistema trabajan correctamente de manera conjunta. El peso de este tipo de pruebas será inferior al de las pruebas unitarias, y se centrarán principalmente sobre algunos componentes importantes de la aplicación móvil que deben trabajar conjuntamente. También se realizarán pruebas de integración de la *API REST*, para comprobar que los componentes del servidor *web* funcionan bien conjuntamente con una base de datos de prueba.
- **Pruebas de sistema.** Este tipo de pruebas se refiere a la integración entre la interfaz de usuario y el resto de componentes de la estructura interna del sistema. Se trata de pruebas de integración destinadas a comprobar el correcto funcionamiento a nivel de interfaz de usuario, es decir, comprobar que la interfaz de usuario integra correctamente con el resto de componentes. En el contexto del proyecto, solo se realizarán pruebas automatizadas de sistema para la aplicación móvil, de modo que se verifique el funcionamiento de los principales aspectos de la interfaz de usuario junto con los componentes internos de la aplicación móvil.
- **Pruebas de Usabilidad.** Las pruebas de usabilidad tienen como objetivo validar el funcionamiento del sistema desde la perspectiva de los usuarios finales. Se trata de garantizar la satisfacción del cliente o de los usuarios finales con la aplicación resultante. Se diseñarán pruebas de usabilidad para la aplicación móvil, a través de unas pautas e itinerarios guiados para los usuarios finales, de forma que se pueda

evaluar el grado de satisfacción de dichos usuarios en cuanto a usabilidad de la aplicación móvil.

Capítulo 8. Diseño del Sistema

En el capítulo anterior, se expuso la especificación del sistema indicando qué se tenía que construir. En este capítulo, se expone **cómo** se va a construir el sistema especificado basándose en los requisitos funcionales y no funcionales descritos anteriormente, junto con el resto de información recabada en las reiterativas fases de **análisis** llevadas a cabo durante los *sprints.*

8.1 Arquitectura del Sistema

Esta primera sección, muestra la **arquitectura** del sistema global y de los diferentes subsistemas identificados en la fase de análisis para proporcionar una visión de alto nivel de los componentes que lo conforman y sus relaciones, todo ello mediante **diagramas**, algunos de los cuales no siguen ningún **estándar *UML***, sino que son meros esquemas que muestran la estructura interna del sistema y sus componentes.

8.1.1 Arquitectura global

El sistema de *Contact Tracker* estará compuesto por varios subsistemas, los cuales interaccionan entre sí intercambiando mensajes y peticiones, siguiendo una arquitectura **cliente-servidor**. De este modo, el servidor expone una serie de puntos o *endpoints* a los que se conectan los clientes realizando peticiones ***HTTP*** para obtener una respuesta en formato ***JSON***. El centro del sistema es el **servidor *web***, que se hospeda en la nube y expone una ***API REST*** con los distintos servicios que ofrece a los clientes. Esto se ve reflejado en el siguiente diagrama de la arquitectura global del sistema.

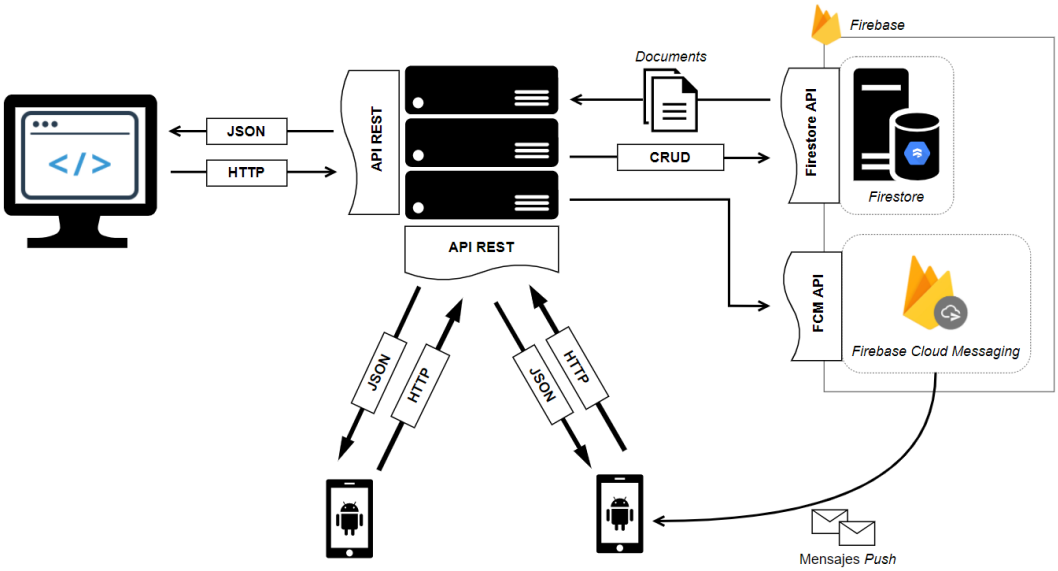


Figura 8.1. Diagrama de la arquitectura global del sistema de Contact Tracker.

El servidor *web* sigue la **arquitectura *REST*** en donde los recursos se representan a través de su estado y son accesibles mediante ***URIs,*** de forma que proporciona a los clientes la capacidad de modificar y obtener el **estado representacional** de los recursos que alberga el servidor.

Como se puede observar, los clientes que consumirán esta *API* son el cliente ***Android*** instalado en los dispositivos móviles y el cliente ***web*** que representa el panel de control. Estos clientes utilizan el protocolo ***http*** para realizar peticiones al servidor a través de su ***URL*** y **puerto** de conexión, el cual variará en función de si el servidor se despliega en local o en una plataforma en la nube. Las peticiones incluyen un **verbo** (*GET, POST…*) que representa la acción solicitada junto con una **ruta** (*/positives*) que representa el recurso requerido, ante la cual, el servidor responde con el resultado en formato ***JSON***. Los clientes también pueden enviar datos al servidor en el **cuerpo** de las peticiones.

Por otro lado, el servidor *web* se conecta a la plataforma de ***Firebase***, la cual juega el papel de *backend* en la nube para hospedar ciertos servicios del sistema. Uno de estos servicios, es el **almacenamiento** en la nube a través de una base de datos **documental** (*NoSQL*), denominada *Firestore*. El servidor *web* utiliza el *SDK* de *Firebase* para realizar peticiones ***CRUD*** a la *API* de *Firebase* y así modificar o leer los documentos de la base de datos. Otro de los servicios, es el envío de **mensajes *push*** a los clientes *Android* mediante *Firebase Cloud Messaging*, el cual se configura desde el servidor *web*. Los mensajes contienen un ***topic*** o **tema** para indicar a qué dispositivos se tienen que enviar las notificaciones, de manera que aquellos clientes *Android* suscritos a un tema reciben las notificaciones de ese tema. Esto se corresponde con el clásico **patrón arquitectónico *Publish-Subscribe*** ya que los clientes *Android* se subscriben o desvinculan de los canales de notificaciones.

En las siguientes secciones se muestra la arquitectura de cada subsistema concreto.

8.1.2 Arquitectura de la aplicación móvil

La aplicación móvil de *Contact Tracker* hace uso de la arquitectura ***MVVM***, es decir, **modelo-vista-modelo de vista**, en donde se divide el sistema en una serie de componentes que se corresponde con sus siglas. De este modo, se divide la aplicación móvil en modelos, vistas y lo que se denominan *viewmodels*, desacoplando al máximo las vistas de la lógica de negocio y de las entidades del modelo. Esta arquitectura tiene cierta similitud con la **arquitectura hexagonal [53]**, pues trata de aislar el modelo de dominio, poco volátil y que apenas sufre cambios, del resto de componentes externos y más variables, para reducir así sus dependencias y desacoplarlo de los cambios frecuentes. La característica más notable de la arquitectura *MVVM* es el uso del patrón de diseño ***Observer***, a través de los objetos observables que reciben el nombre de *LiveData*. A continuación, se muestra el diagrama de la arquitectura de la aplicación móvil, que refleja perfectamente esta arquitectura *MVVM*.

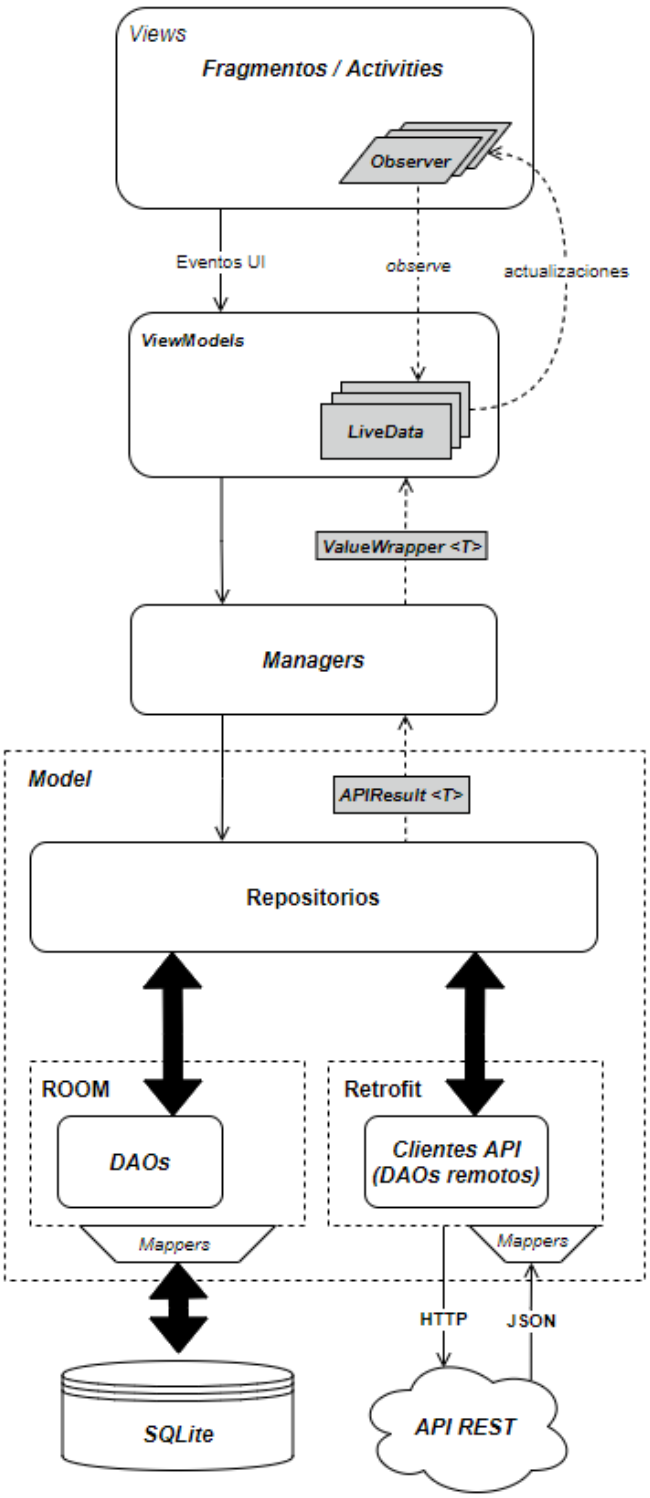


Figura 8.2. Diagrama de la arquitectura de la aplicación móvil, reflejando la arquitectura MVVM.

Como se puede apreciar, en la primera capa se encuentran las vistas de la aplicación, representadas por los fragmentos y actividades de *Android* junto con sus *layouts*. En las vistas, residen los objetos ***observer*** que observan los **cambios** en los elementos *LiveData* que representan los **observables**. Los observables *LiveData* residen en la siguiente capa de los ***viewmodels***, donde se almacenan los datos de la vista para ser observados y así enviar

actualizaciones automáticamente cuando se produce algún cambio en los datos. En el diagrama superior, se puede ver esta relación entre los *observers* y los *LiveData*, además de los eventos de la interfaz que se envían a los *viewmodels* y que pueden afectar a los componentes *LiveData*. A continuación de los *viewmodels*, se encuentran lo que se denominan ***managers***, componentes que albergan la gran mayoría de la **lógica de negocio** de la aplicación. Estos *managers* se encargan de diferentes aspectos de los servicios que ofrece la aplicación, como la notificación de positivos o la comprobación de contactos, entre otros, y no forman parte de la arquitectura *MVVM*, sino que son un **añadido** concreto para este proyecto. Como se ve en el diagrama, los *managers* devuelven los resultados a través de objetos **genéricos** que se denominan ***ValueWrappers***. Estos envuelven los datos en un objeto de **éxito** o de **fallo** según el resultado, lo cual facilita su manejo en las capas superiores.

Siguiendo el diagrama, el **modelo** de la aplicación está esparcido entre los **repositorios**, y los **objetos de acceso** local y remoto. Los repositorios contienen todas las operaciones de **persistencia** de la aplicación móvil, ya sea en la base de datos local del dispositivo o en la nube consumiendo la *API REST* expuesta por el servidor. En este último caso, los repositorios devuelven las respuestas de la nube en objetos envoltorio denominados ***APIResult*** que funcionan de manera similar a los *value wrappers*, distinguiendo entre una respuesta **exitosa**, un **error *http*** y otro tipo de **error de red**. Para acceder a los datos locales, se utilizan los ***DAOs*** o *Data Access Objects*, que son interfaces que representan las consultas ***SQL*** a realizar sobre la base de datos ***SQLite*** integrada en el dispositivo. Estas interfaces son implementadas por la librería *ROOM* que simplifica la implementación y juega el papel de un ***ORM*** (*Object Relational Mapper*) mapeando las entidades del dominio y sus relaciones a tablas de la base de datos.

Por otro lado, para acceder a los datos **remotos**, se utilizan los clientes de la *API* a través de interfaces que contienen las peticiones a realizar, incluyendo el verbo, la ruta del recurso y los parámetros o datos en el cuerpo de la petición, en caso de que existan. Estas interfaces son implementadas por *Retrofit* a bajo nivel mediante peticiones *http*. Las respuestas recibidas en formato *JSON* se procesan y son **mapeadas** a entidades del dominio también a través de *Retrofit*.

A modo de resumen, esta manera de organizar los componentes siguiendo la arquitectura *MVVM*, permite mantener la **fluidez** de la interfaz de usuario de la aplicación móvil a la vez que se realizan procesamientos internos y llamadas en red, gracias al patrón *Observer* que actualiza los componentes de la interfaz cada vez que se actualizan los datos del modelo, todo ello sin necesidad de **monitorizar** exhaustivamente dichos cambios en los datos.

8.1.3 Arquitectura del servidor *web*

Como ya se comentó en la arquitectura global, el servidor *web* hace uso de una arquitectura ***REST*** para exponer los recursos y servicios por medio de *URIs* accesibles para los clientes. Además, para organizar los componentes se utiliza una versión simplificada de la arquitectura ***MVC*** de modelo, vista y controlador. El modelo está representado por los **repositorios**, ya que no existen entidades como tal, sino que son objetos en formato *JSON* dada la sencillez del esquema de datos. Las vistas en este caso se corresponden con los clientes que realizan las peticiones y reciben las respuestas y los controladores son los intermediarios entre dichas peticiones y el modelo.

A continuación, se muestra el diagrama de la arquitectura para el servidor *web* donde se puede ver lo comentado anteriormente.

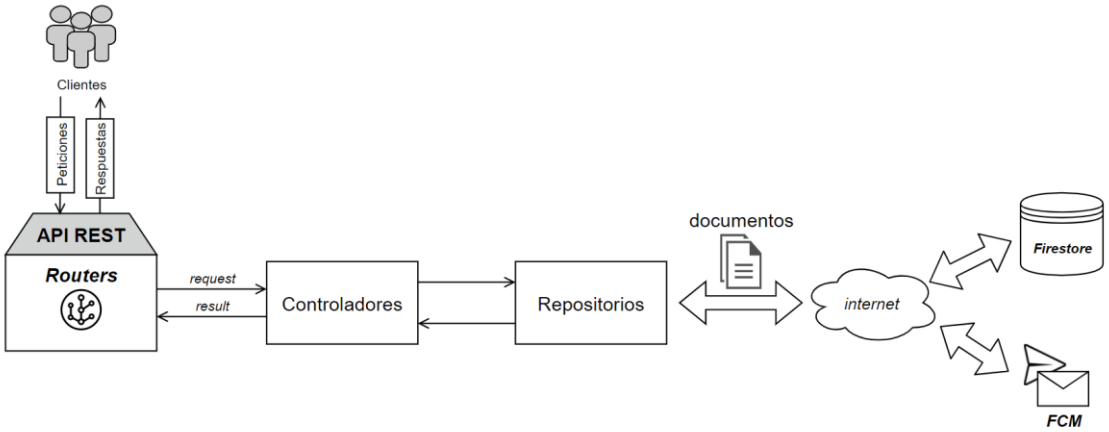


Figura 8.3. Diagrama de la arquitectura del servidor web que expone la API REST.

En el diagrama, los ***routers*** son los elementos encargados de **redirigir** las peticiones que los clientes lanzan a la *API REST* y enviarlas a los controladores correspondientes. Estos a su vez invocan a los repositorios para realizar las operaciones de persistencia necesarias y devuelven el resultado en la **respuesta** que es redirigida por los *routers* a los clientes. Así, los *routers* juegan el papel de intermediario entre la *API REST* y los controladores, pues son los responsables de emparejar las rutas del servidor con sus respectivos controladores.

Dada la naturaleza **ágil** del desarrollo, la arquitectura de este subsistema se organiza en torno a **módulos**, cada uno con su responsabilidad, de forma que se puedan hacer cambios rápidamente de cara a posibles fluctuaciones en los requisitos durante las iteraciones. Además, como se puede apreciar, desde los controladores se accede directamente a los repositorios, dado que la lógica de negocio es tan escasa en el servidor *web* que no merece la pena extraerla a módulos de **servicios** que sirvan de intermediarios entre los controladores y los repositorios.

Por último, los repositorios acceden a través de la red a los servicios de *Firebase* mediante el *SDK*, realizando lecturas y escrituras en la base de datos documental, o bien, realizando llamadas a la *API FCM* para enviar las notificaciones *push* a los clientes *Android*.

8.1.4 Arquitectura de la aplicación *web*

Para terminar con la arquitectura, en esta sección se presenta la arquitectura interna de la aplicación *web* cliente que sirve de panel de control para los administradores. En este caso, no se sigue ningún patrón arquitectónico específico más allá de los **enrutadores** para renderizar las vistas de la aplicación en función de la ruta especificada en la *URL*. El resto de este subsistema gira en torno al concepto de **componente**. Un componente es una pieza de código que contiene tres elementos: el renderizado de dicho componente en la interfaz, a través de su **marcado *html***, la **lógica de negocio**, es decir, el comportamiento del componente y los **estilos *css*** del componente. De este modo, la aplicación *web* se construye en base a una **jerarquía** de componentes que van conformando la estructura y el contenido de las vistas,

tomando como raíz el componente *App* del cual heredan el resto de las componentes. El siguiente diagrama muestra esta organización interna de la aplicación *web*.

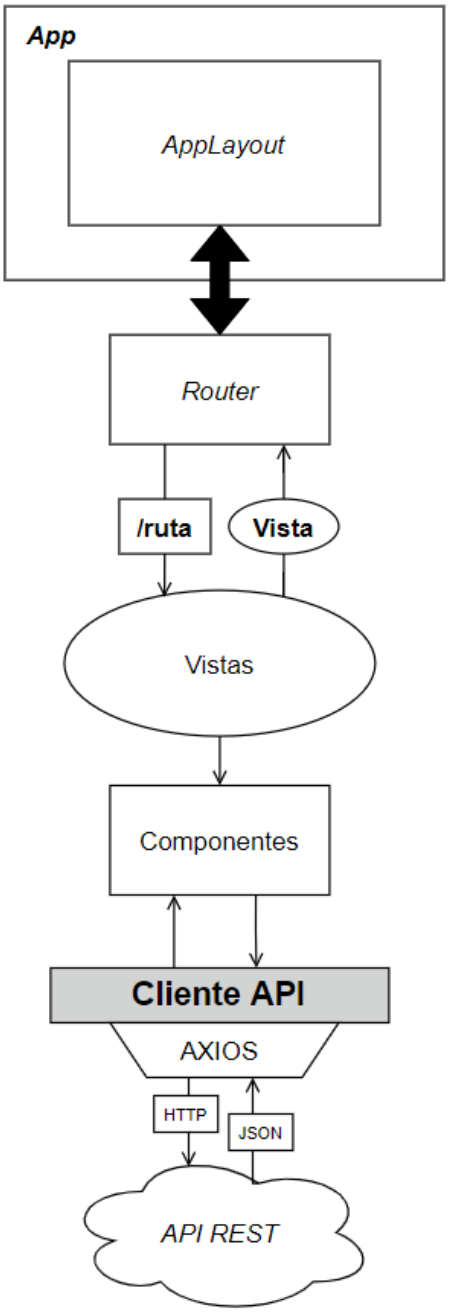


Figura 8.4. Diagrama de arquitectura de la aplicación web.

En el diagrama se aprecian los enrutadores que a partir de una ruta específica renderizan una vista u otra dentro del ***layout*** de la aplicación, representado por el componente *App* que incluye un componente ***AppLayout***, el cual contiene toda la infraestructura de la aplicación *web* incluyendo la lógica de navegación. Las vistas están formadas por componentes, los cuales manejan los datos del servidor a través de **clientes *http*** de la *API REST*. De manera similar a como se hace en la aplicación móvil con *Retrofit*, las peticiones de red y las respuestas se gestionan utilizando ***Axios***, el cual permite manejar las peticiones y respuestas *JSON* de manera **asíncrona**. De este modo, los componentes acceden a los datos del servidor *web* consumiendo su *API* REST y manteniendo la fluidez de la interfaz.

8.1.5 Diagramas de Paquetes

Una vez vista la organización de los componentes y la estructura de alto nivel de los distintos subsistemas que componen el sistema, en esta sección se presentan diagramas de paquetes siguiendo el estándar ***UML***. Mediante estos diagramas se muestra la estructura interna de cada subsistema con mayor nivel de detalle, con los principales paquetes en los que se agrupan las clases de código. En las próximas secciones se presenta para cada subsistema un diagrama de la **estructura de descomposición** en paquetes, un diagrama de **paquetes** junto con sus **interacciones** y finalmente una tabla que muestra una breve **descripción** del contenido y las responsabilidades de cada uno de esos paquetes.

Cabe destacar que estos diagramas de paquetes son producto de varias **iteraciones** de análisis y diseño durante los *sprints* del desarrollo del proyecto, de tal forma que su contenido sea lo más **afín** posible con el futuro sistema construido, pero sin excederse en demasiados detalles de bajo nivel.

8.1.5.1 *Diagrama de paquetes de la aplicación móvil de rastreo*

La aplicación móvil de rastreo constituye el subsistema más complejo de los identificados en el apartado 7.3 IDENTIFICACIÓN DE LOS SUBSISTEMAS EN LA FASE DE ANÁLISIS y por tanto su diagrama paquetes es el más extenso de los presentados. Como ya se mencionó antes, al igual que los otros diagramas, este no constituye el diagrama preliminar de paquetes realizado previo al desarrollo, sino que, como consecuencia del enfoque **ágil,** es el resultado de múltiples iteraciones tras las cuales se han ido añadiendo detalles al diagrama hasta el punto de representar lo mejor posible el sistema construido. Sin embargo, no se representan todos los detalles de bajo nivel para evitar que el diagrama se ofusque y se dificulte su lectura. La siguiente figura muestra la estructura de descomposición en paquetes de la aplicación móvil de rastreo.

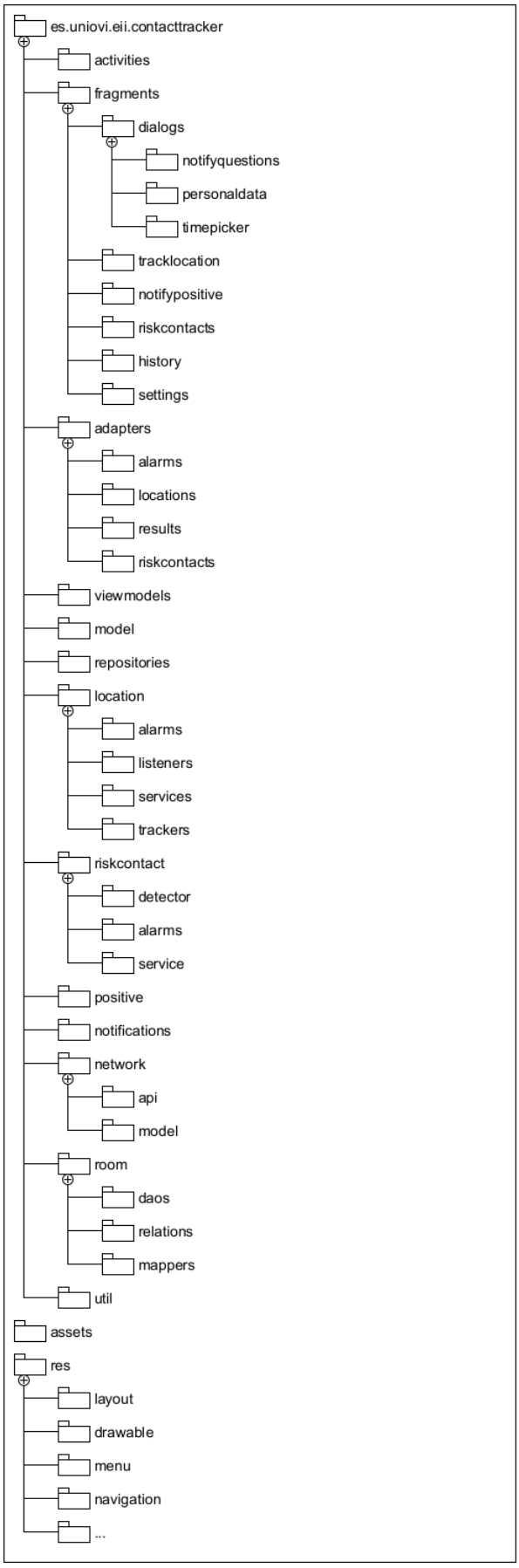


Figura 8.5. Estructura de descomposición de paquetes de la aplicación móvil de rastreo.

Como se puede ver, se trata de una estructura de paquetes bastante extensa que, como es de esperar, da lugar a un diagrama de relaciones entre paquetes bastante extenso. La siguiente figura muestra el diagrama de paquetes junto con sus interacciones y relaciones para la aplicación móvil.

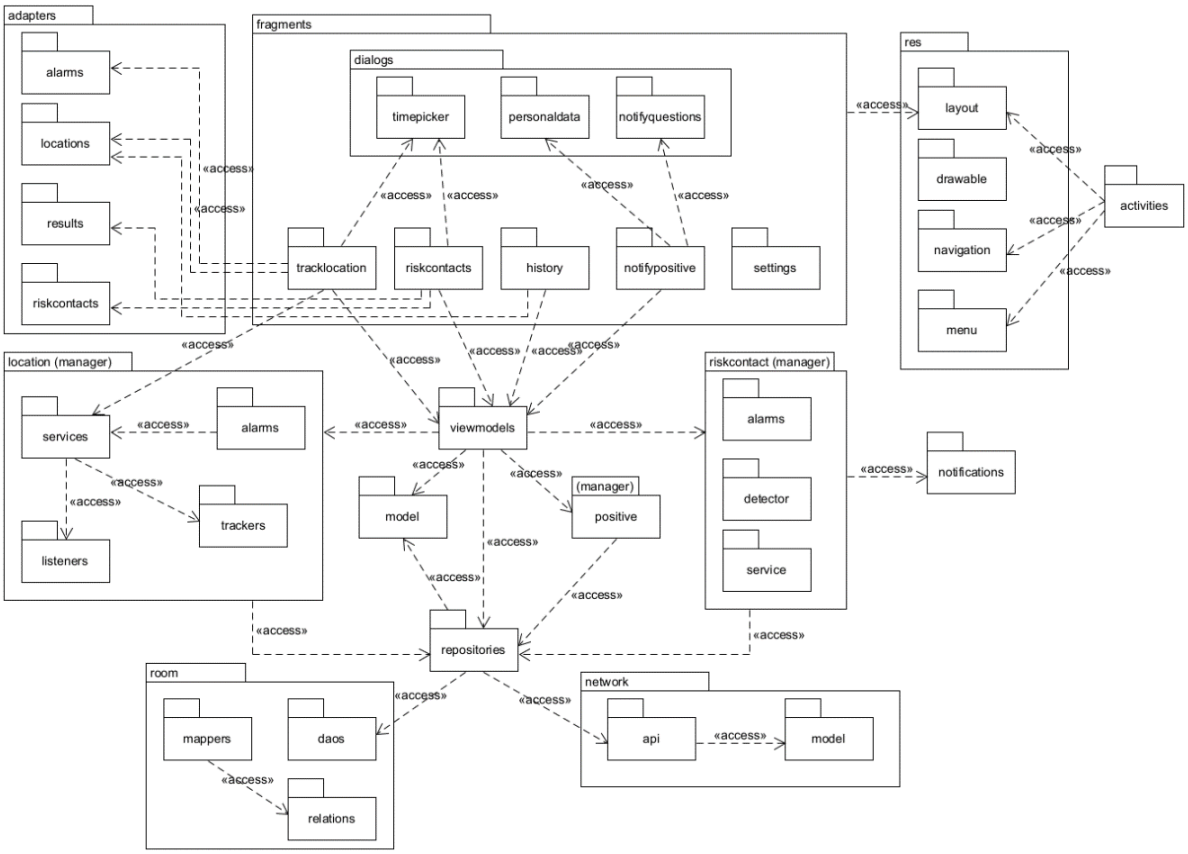


Figura 8.6. Diagrama de paquetes de la aplicación móvil de rastreo incluyendo sus relaciones.

Aunque el diagrama superior parece tener una complejidad considerable, realmente se puede apreciar que sigue la estructura simple de la arquitectura ***MVVM*** explicada en la sección 8.1.2 ARQUITECTURA DE LA APLICACIÓN MÓVIL. Se puede ver que los paquetes de las vistas (fragmentos y actividades) se relacionan con el paquete de *viewmodels* y este a su vez se relaciona tanto con los paquetes de los *managers* como con los de los repositorios. El paquete de repositorios interacciona con los *DAOs* y los clientes de la *API*.

Para facilitar la lectura de los paquetes, a continuación, se muestran varias tablas con la descripción de cada paquete, cada una de las cuales agrupa los paquetes relacionados entre sí o que están bajo un mismo paquete padre.

Tabla 8.1. Descripción de los paquetes de adapters de la aplicación móvil.

adapters	
Nombre del paquete	**Descripción**
alarms	Contiene los *adapters* para las alarmas de localización utilizados en las listas de los *recycle views.*
locations	Contiene los *adapters* para los objetos de localización utilizados en las listas de los *recycle views.*

results	Almacena los *adapters* para los objetos del dominio que representan los resultados de una comprobación.
riskcontacts	Almacena los *adapters* para los objetos del dominio que representan los contactos de riesgo detectados en una comprobación.

Un dato interesante sobre estos paquetes es que hacen uso del **patrón de diseño *Adapter***, para adaptar las entidades del dominio a las vistas personalizadas de *Android* que se muestran en las listas recicladas.

fragments	
Nombre del paquete	**Descripción**
tracklocation	Contiene los fragmentos relacionados con el rastreo de ubicación.
riskcontacts	Contiene los fragmento relacionados con la comprobación de contactos de riesgo y visualización de resultados.
history	Contiene los fragmentos relacionados con el histórico de localizaciones.
notifypositive	Contiene los fragmentos relacionados con la notificación de positivos.
settings	Contiene los fragmentos relacionados con los ajustes generales de la aplicación.
dialogs	Paquete padre que almacena otros paquetes relativos a los diálogos modales de la aplicación.
timepicker	Alberga los diálogos para seleccionar una hora del día, es decir, horas y minutos.
personaldata	Alberga los diálogos para introducir los datos personales, así como los diálogos con las cláusulas de consentimiento.
notifyquestions	Alberga los diálogos con las preguntas realizadas al usuario a la hora de notificar un positivo.

Tabla 8.2. Descripción de los paquetes relacionados con el rastreo de ubicación en la aplicación móvil.

location (manager)	
Nombre del paquete	**Descripción**
services	Representa los servicios *Android* para ejecutar el rastreo de ubicación en primer o segundo plano, o con la aplicación apagada.
alarms	Contiene la lógica de negocio para programar alarmas de rastreo de ubicación a ciertas horas.
listeners	Alberga los objetos *listener* de escucha de nuevas actualizaciones de la posición del dispositivo. Contiene los *callbacks* que son llamados con las nuevas localizaciones obtenidas.
trackers	Es el responsable de la lógica de inicio y detención del rastreo de ubicación, a través de las abstracciones que proporciona el *framework* de *Android* para utilizar el *GPS* del dispositivo.

Tabla 8.3. Descripción de los paquetes relacionados con la comprobación de contactos de riesgo en la aplicación móvil.

riskcontact (manager)	
Nombre del paquete	**Descripción**
alarms	Contiene la lógica para programar alarmas de comprobación de contactos de riesgo que se ejecuten a ciertas horas.
detector	Representa la lógica del propio algoritmo de comparación de itinerarios para detectar contactos de riesgo.
service	Contiene los componentes necesarios para ejecutar un servicio de *Android* en primer o segundo plano para realizar la comprobación de contactos.

Tabla 8.4. Descripción de los paquetes relacionados con la persistencia en local en la aplicación móvil de rastreo.

room	
Nombre del paquete	**Descripción**
mappers	Mapeadores de *ROOM* para realizar transformaciones de los datos que vienen y van a la base de datos *SQLite* y convertirlos a objetos del dominio.
daos	Contiene todas las interfaces que representan los *DAOs* con las operaciones y consultas que se pueden realizar sobre la base de datos local.
relations	Almacena algunas clases auxiliares utilizadas para representar relaciones complejas entre entidades, como relaciones muchos a muchos (*n-m*).

Tabla 8.5. Descripción de los paquetes relacionados con las operaciones de red de la aplicación móvil de rastreo.

network	
Nombre del paquete	**Descripción**
api	Contiene los clientes de la *API REST* con las interfaces que representan las peticiones *http* a las rutas concretas con los parámetros indicados. Especifican los verbos *http* utilizados en la petición, así como el cuerpo de esta en caso de que se envíen datos.
model	Alberga algunas clases auxiliares para representar el resultado obtenido en la respuesta del servidor.

Tabla 8.6. Descripción de otros paquetes importantes de la aplicación móvil de rastreo.

Otros paquetes	
Nombre del paquete	**Descripción**
viewmodels	Contiene todos los componentes *viewmodel* que forman parte de la arquitectura *MVVM*.
model	Alberga las entidades del modelo de dominio que se emplean en las operaciones de la lógica de negocio.
positive (manager)	Representa el manager de positivos. Contiene los elementos involucrados en la notificación de positivos a través de la aplicación móvil.
repositories	Contiene todos los repositorios que albergan las operaciones de

	persistencia y que se relacionan con los *DAOs* y los clientes de la *API.*
activities	Contiene los componentes *Activity* de *Android* que conforman la aplicación móvil.
notifications	Responsable de la lógica de recepción de notificaciones *push* provenientes de la nube. También se encarga de construir y mostrar las notificaciones *in app*, es decir, las notificaciones internas de la aplicación móvil.

Tabla 8.7. Descripción de los paquetes relacionados con los recursos de la aplicación móvil.

res	
Nombre del paquete	**Descripción**
layout	Contiene los elementos *XML* que representan los *layouts* de la aplicación móvil.
drawable	Alberga los objetos *drawable* como imágenes, iconos o logotipos que se pueden dibujar dentro de la aplicación móvil.
navigation	Contiene el grafo de navegación de la aplicación móvil que define las relaciones de navegación entre los fragmentos incluyendo la definición de las acciones para desplazarse de una pantalla a otra.
menu	Contiene los *XML* que representan los distintos menús de la aplicación junto con sus opciones.
…	Existen multitud de paquetes dentro de *res* que no se muestran aquí dado que no son objeto de este apartado y para evitar excederse demasiado en detalles que no aportan información útil.

8.1.5.2 *Diagrama de paquetes de la API REST*

La arquitectura *REST* del servidor *web* vista anteriormente da lugar a la siguiente estructura de descomposición en paquetes.

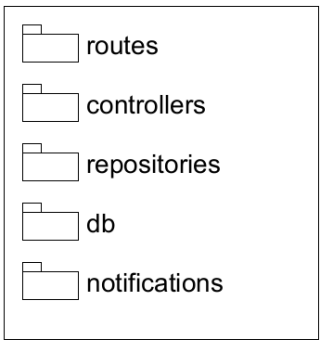


Figura 8.7. Diagrama de descomposición de paquetes del servidor web.

Como se puede ver, es una estructura muy simple compuesta solo por paquetes raíz que no contienen en su interior otros paquetes hijo. Esto se debe a la sencillez del servidor y la escasa lógica de negocio. La naturaleza ágil del proyecto hace que surja la posibilidad de que en un futuro esta estructura de paquetes crezca como consecuencia del aumento de la lógica de negocio y del dominio, de forma que se organicen los módulos en distintos subpaquetes.

A continuación, se muestra el diagrama que representa las relaciones de uso entre los paquetes vistos en la estructura de descomposición.

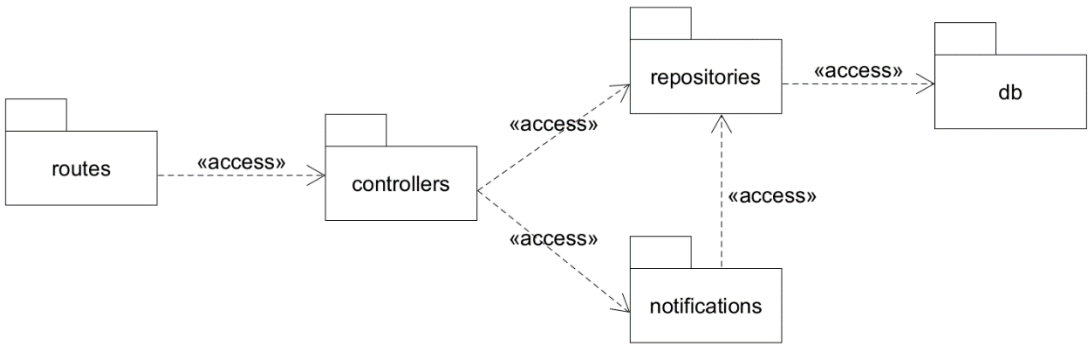


Figura 8.8. Diagrama de paquetes del servidor web junto con sus relaciones.

Los paquetes del servidor *web* se relacionan mediante flechas *Access* ya que no se importa el código, sino que solo se accede a sus servicios expuestos. En el diagrama superior, el paquete *routes* representa la ***API REST*** que expone las rutas a los clientes para acceder a los recursos, además de gestionar la lógica para redirigir las peticiones a los controladores correspondientes. La siguiente tabla muestra una breve descripción para cada uno de los paquetes que conforman la aplicación *web*.

Tabla 8.8. Descripción de los paquetes del servidor web.

Nombre del paquete	Descripción
routes	Representa el módulo de enrutado del servidor, redirigiendo las peticiones a los controladores que las manejan. Alberga las rutas *http* para acceder a los recursos del servidor *web.*
controllers	Contiene los controladores encargados de gestionar las peticiones y las respuestas a los clientes. Procesa las peticiones enviadas desde los enrutadores y realiza las llamadas a los repositorios correspondientes para completar las operaciones de persistencia.
repositories	Alberga los repositorios que representan las operaciones de persistencia para leer y escribir datos en la base de datos documental en la nube.
notifications	Responsable de la lógica de envío de notificaciones *push* periódicas a los clientes *Android*, utilizando la *API FCM* de *Firebase*.
db	Contiene las dependencias de configuración de la base de datos documental *Firestore*, así como la lógica de conexión con el proyecto en *Firebase.*

8.1.5.3 *Diagrama de paquetes de la aplicación web*

Como se vio anteriormente, la arquitectura de la aplicación *web* es muy simple y no sigue ninguna arquitectura o patrón especifico, con lo cual su diagrama de paquetes resulta bastante simple. La siguiente figura muestra su estructura de descomposición en paquetes.

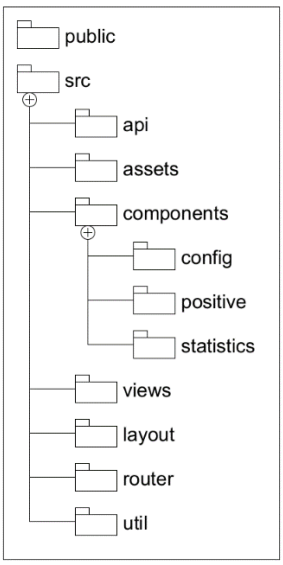


Figura 8.9. Estructura de descomposición de paquetes de la aplicación web.

A partir de esta estructura de paquetes, se puede inferir el siguiente diagrama de paquetes que muestra las relaciones e interacciones entre los mismos.

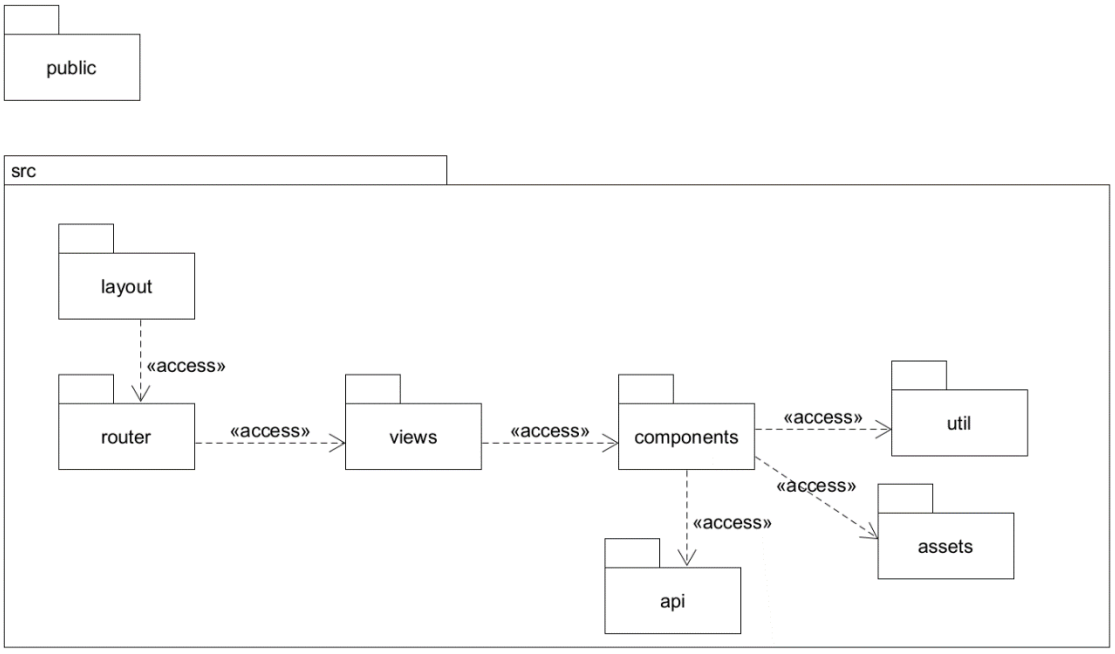


Figura 8.10. Diagrama de paquetes de la aplicación web.

Como se puede apreciar, los paquetes se relacionan entre sí a través de relaciones *Access* ya que no se importa el código del paquete, sino que simplemente se accede a las operaciones que ofrece el mismo. El paquete *public* está aislado del resto ya que representa la infraestructura de la aplicación *web*, con el fichero *index.html* típico sobre el cual se van cargando los componentes *web*. El paquete *src* contiene el resto de paquetes fuente de la aplicación *web*.

A continuación, se muestra una tabla con la descripción de cada uno de estos paquetes de la aplicación *web*.

Tabla 8.9. Descripción de los paquetes de la aplicación web.

Nombre del paquete	Descripción
public	Contiene los recursos públicos de la aplicación *web*, incluyendo el fichero *index.html* que representa la raíz del proyecto sobre la que se cargan los componentes *web*.
src	Contiene el código fuente de toda la aplicación *web*, incluyendo las vistas y sus componentes, además de otros módulos importantes.
layout	Alberga el *layout* general de la aplicación, es decir, la infraestructura de la *web* y su menú de navegación.
router	Contiene la lógica de enrutado para renderizar distintas vistas en función de la ruta especificada en la *URL*.
views	Contiene los componentes *web* que representan una vista dentro de la aplicación y que agrupan a varios componentes que conforman sus elementos visuales.
components	Contiene los propios componentes que representan elementos de la interfaz *web* y que conforman las vistas. Incluyen el marcado *html*, el *script* de comportamiento y sus estilos *css.*
api	Alberga los clientes de la *API REST* encargados de realizar las peticiones *http* y recibir las respuestas *JSON* desde el servidor *web*. Son utilizados por los componentes para recuperar y modificar datos del servidor.
util	Paquete con distintas utilidades utilizadas por los componentes como formateo de fechas.
assets	Contiene los recursos de la aplicación utilizados por los componentes, incluyendo ficheros, imágenes u otros.

8.1.6 Diagramas de Despliegue

El sistema de *Contact Tracker* se desplegará en un entorno controlado para probar el funcionamiento del prototipo de la aplicación de rastreo. Concretamente, el servidor *web* será desplegado en un contenedor de ***Azure*** en la nube mediante una instancia de ***Azure App Service***, accesible en remoto desde un cliente *web* o móvil, además de desplegar la aplicación *web* en una **cuenta de almacenamiento** para albergar recursos estáticos. La aplicación móvil no será desplegada en ninguna plataforma para ser descargada por los usuarios, sino que se proporcionará el ***APK*** de la aplicación manualmente.

En un futuro sistema basado en este prototipo, con cada nueva versión de la aplicación móvil se generaría una nueva ***release*** la cual sería necesario firmar mediante un certificado para subirla a la ***Google Play Store*** mediante una licencia. De este modo, el *APK* oficial de la aplicación estaría disponible en los servidores de *Play Store* para ser descargado por cualquier usuario que disponga de un dispositivo *Android* con una versión compatible.

A continuación, se muestra el diagrama de despliegue correspondiente al sistema de *Contact Tracker*.

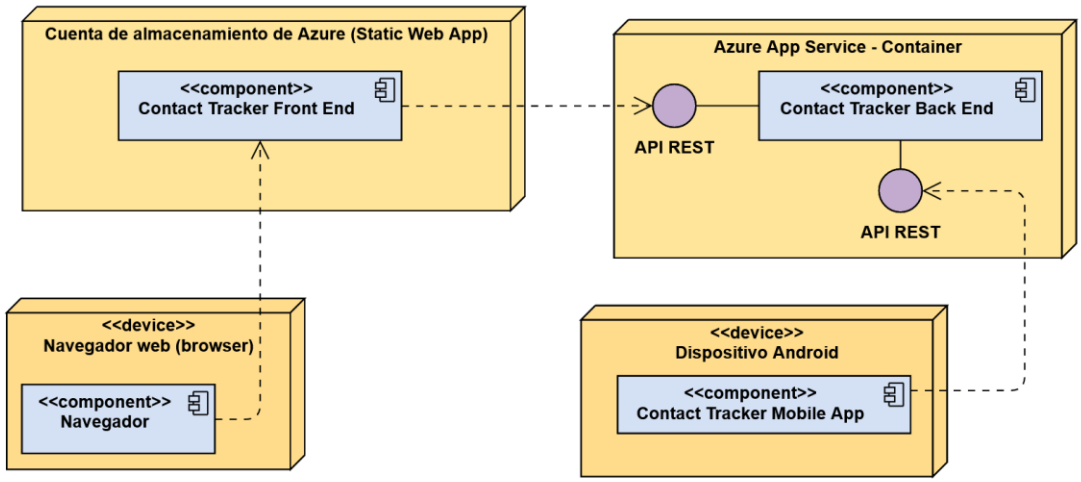


Figura 8.11. Diagrama de despliegue del sistema de Contact Tracker.

Como se puede apreciar, el *APK* de la aplicación móvil se ejecuta sobre un dispositivo con *Android* instalado y consume los servicios de la *API REST* expuesta por el servidor *web* de *Contact Tracker*. Este servidor *web* se hospeda en un ***Azure App Service*** en la plataforma en la nube de *Azure*. Además, la *API REST* también se expone para la aplicación *web* cliente, la cual se despliega sobre una **cuenta de almacenamiento** de *Azure* para albergar recursos estáticos. Por último, para acceder en remoto a la aplicación *web* desplegada se hace uso de un navegador *web* que cargue los recursos desde la cuenta de *Azure* donde se alberga la aplicación *web*.

8.2 Diseño de Clases

Basándose en los diagramas de arquitectura expuestos anteriormente, en esta sección se presenta el diseño de clases para cada uno de los subsistemas. Como se viene viendo a lo largo de todo el documento, la naturaleza **ágil** del desarrollo implica que las fases de análisis y diseño se lleven a cabo con mayor espontaneidad y de manera iterativa a lo largo de los diferentes *sprints*. Con ello, los diagramas de clases aquí presentados son producto de diversas iteraciones sobre las cuales se fueron añadiendo detalles de forma que representen el estado final del sistema construido.

Estos diagramas de clases se han realizado siguiendo el **estándar *UML*** y no contienen todos los detalles de bajo nivel para no excederse demasiado y así evitar que los diagramas no sean legibles. Además, para facilitar la lectura de los diagramas, estos se presentan por separado en hojas con orientación en horizontal, de modo que el nivel de *zoom* sea suficiente para leerlo con facilidad.

8.2.1 Diagrama de Clases de la aplicación móvil

Ya se vio en la arquitectura el nivel de complejidad de la aplicación móvil de rastreo, que constituye el núcleo del sistema de *Contact Tracker,* con lo que los diagramas de clases en este caso también serán extensos y por ello se dividen en diversos apartados donde se trata cada

uno de ellos por separado. De nuevo, se recalca que el número de clases de la aplicación móvil es bastante elevado y por ello no se contemplan todas ellas en los siguientes diagramas de clases, solo se tienen en cuenta las clases más importantes de la aplicación.

8.2.1.1 *Modelo de dominio*

En el apartado 7.4.1 relativo al modelo de dominio de la aplicación móvil visto en el capítulo del análisis del sistema, ya se realizó una versión preliminar del modelo de dominio de la aplicación móvil basada en los requisitos recopilados en las historias de usuario junto con los requisitos no funcionales. Ahora, partiendo de ese boceto inicial del dominio, se construye un diagrama de clases que representa las entidades principales del dominio de la aplicación móvil junto con sus relaciones, con un mayor nivel de detalle.

En este diagrama, se incluyen no solo las entidades del dominio, sino también otros objetos auxiliares como los empleados para envolver los resultados de la *API REST* o de los *managers*, así como los objetos de configuración y aquellos que modelan las respuestas enviadas por el servidor. En la siguiente página se muestra este diagrama de modelo de dominio de la aplicación móvil.

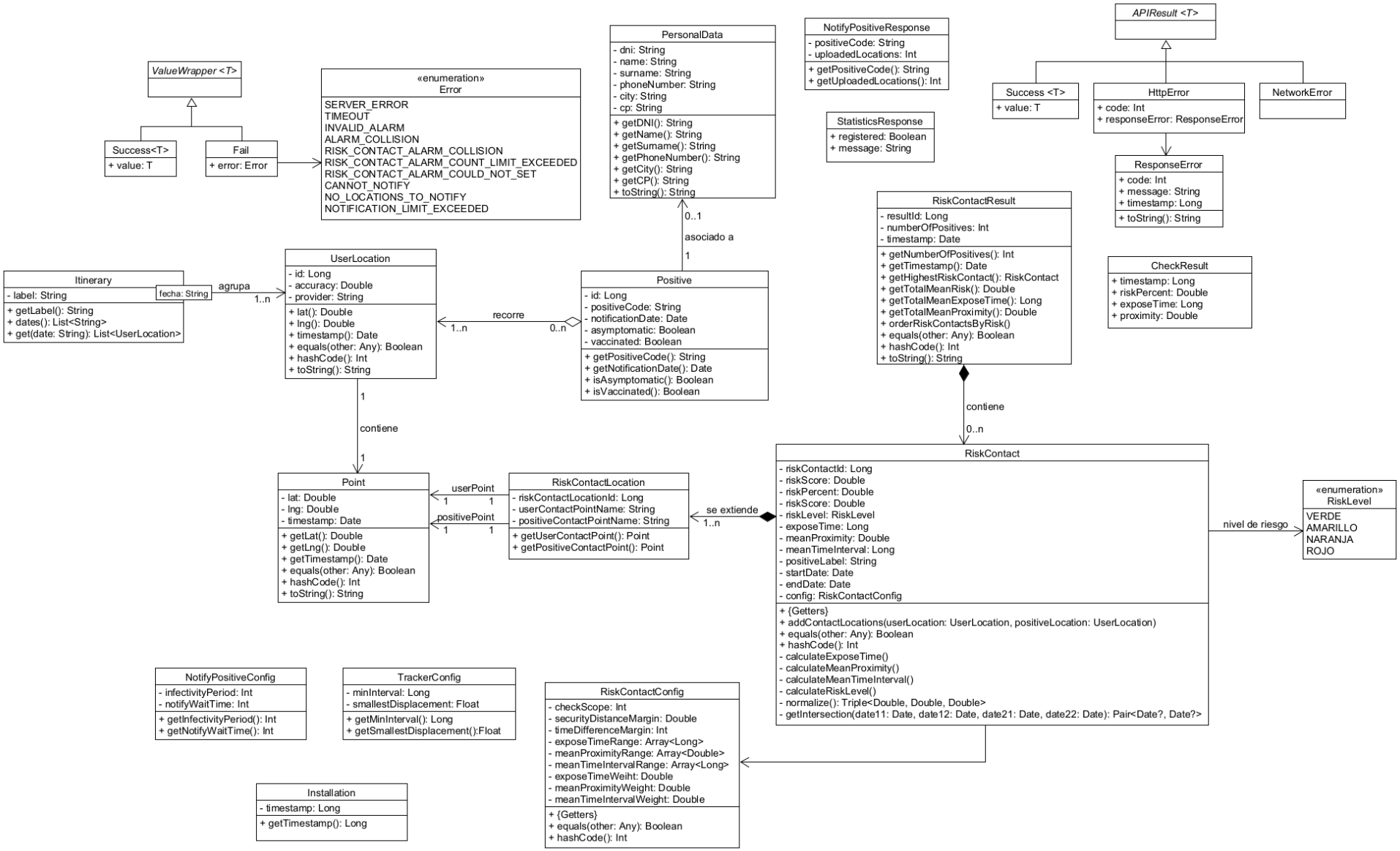


Figura 8.12. Diagrama de clases del modelo de dominio de la aplicación móvil.

Como se puede apreciar, en el diagrama se representan los objetos envoltorio utilizados para envolver los resultados. Por un lado, la clase genérica ***ValueWrapper*** representa un envoltorio para los resultados obtenidos de la lógica de negocio ejecutada en los *managers*. De esta clase heredan una clase ***Success*** que representa un resultado exitoso y una clase ***Fail*** que representa algún error de la lógica de negocio, el cual es representado por una clase ***Error*** que se almacena en esta clase de fallo. Por otro lado, está la clase ***APIResult*** que modela un resultado de una llamada a la ***API REST*** del servidor. En este caso, son tres clases las que heredan de esta clase envoltorio. La primera, ***Success*** representa un resultado *200 OK* de éxito, la segunda, ***HttpError***, representa un error *http* en la gama de los ***4xx-5xx***, albergando un objeto con el código de error, un mensaje y la hora, y la última, ***NetworkError***, modela simplemente un error de red en la petición.

Abajo a la izquierda, se pueden apreciar las clases que representan los **parámetros** de **configuración** del sistema que deben ser leídos por la aplicación móvil para ajustar su comportamiento. Además, en el diagrama también se muestran las clases que modelan las **respuestas** recibidas tras una petición a la *APIREST*, como la clase *NotifyPositiveResponse* que modela la respuesta del servidor una vez se notifica un positivo, o la clase *StatisticsResponse* que modela la respuesta tras registrar un valor estadístico en el servidor.

En cuanto a las relaciones entre las clases, se puede observar que son relaciones simples que modelan en la mayoría de los casos que una clase contiene a otras, como en el caso del resultado de una comprobación que tiene una relación de **composición** con los contactos de riesgo, ya que esta almacena en su interior una lista con los contactos de riesgo detectados, los cuales existen gracias a la existencia del resultado. Otro ejemplo, es la clase *Positive* que mantiene una relación de **agregación** con la clase *UserLocation*, ya que las localizaciones pueden existir sin necesidad de estar asociadas a un positivo.

8.2.1.2 *Rastreo de ubicación*

Dentro de la aplicación móvil, el rastreo de ubicación está modelado por un conjunto de clases que trabajan conjuntamente para obtener actualizaciones de la posición del dispositivo y almacenar las coordenadas en la base de datos local, todo ello funcionando tanto con la aplicación encendida como apagada. Además, como ya se comentó en la SECCIÓN 3.2.2.2 ABSTRACCIONES PARA ACCEDER A LA LOCALIZACIÓN en los Aspectos Teóricos del proyecto, existen dos abstracciones para acceder a los servicios de geolocalización de *Android*, por lo que se debe diseñar el rastreo de forma que se pueda alternar entre una manera u otra de obtener información sobre la posición del dispositivo. Antes de pasar a explicar el diseño del rastreo de ubicación, se muestra a continuación el correspondiente diagrama de clases.

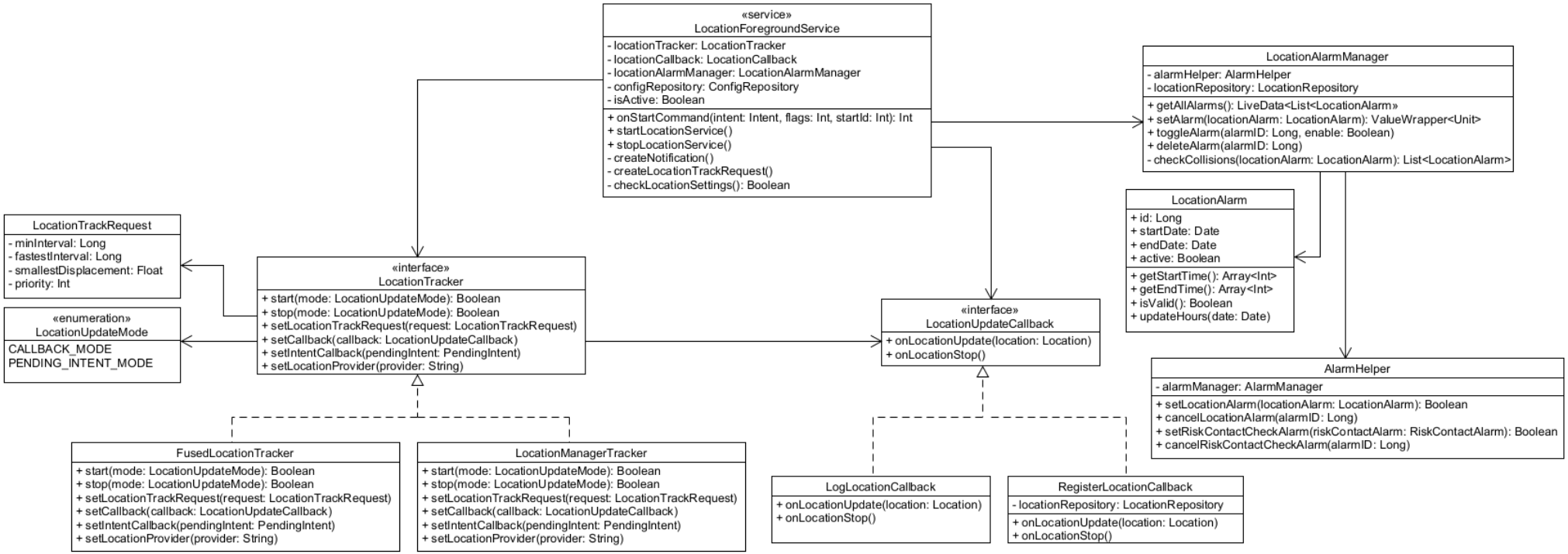


Figura 8.13. Diagrama de clases del rastreo de ubicación dentro de la aplicación móvil.

En el diagrama superior, se aprecia la **interfaz *LocationTracker*** la cual representa la abstracción del rastreador de ubicación cuya responsabilidad principal será la de iniciar y pausar la obtención de actualizaciones de posición del dispositivo empleando las dependencias de *Android* para acceder al servicio ***GPS***. De este modo, existen dos **implementaciones** de dicha interfaz, una para utilizar el ***Fused Location Provider*****,** que proporciona coordenadas a través de los servicios de *Google Play*, y otra para usar el ***Location Manager*** de *Android* integrado en el propio *framework*. Para ello, se ha hecho uso del patrón de diseño ***Strategy***, el cual permite modelar diferentes comportamientos de la misma abstracción de forma que se pueda cambiar el comportamiento dinámicamente y sin afectar al resto de clases. De manera adicional, en la interfaz del rastreo de ubicación, para iniciar y pausar el rastreo se puede seleccionar un ***Location Update Mode***, es decir, un modo de actualización de localizaciones. Esto quiere decir que para acceder a las coordenadas proporcionadas por las abstracciones de la geolocalización se puede utilizar un *callback* o *listener* normal, o bien utilizar un *Pending Intent* de *Android*. En este caso, se podría aplicar el patrón de diseño ***State*** para modelar dos estados distintos y ahorrar líneas de código sin necesidad de comprobar el estado en el que se realiza el rastreo mediante una rama *if-else*, sin embargo, no se ha empleado este patrón de diseño dado que se llegó a un sobrediseño donde solo se ahorraban unas pocas líneas de código. En un futuro donde hubiera más modos de recuperar las localizaciones sería interesante realizar esta refactorización para emplear el patrón *State*.

Por otro lado, en el diagrama se aprecia un **servicio** de *Android* que es quien hace uso del rastreador de ubicación. El rastreo de ubicación es iniciado o pausado por este servicio en función del **comando** que reciba de las clases cliente. Mediante este servicio, se garantiza que el rastreo de ubicación es **persistente** en el tiempo y que no se detiene hasta que el usuario lo desee, es decir, se mantiene encendido aunque la aplicación se cierre o incluso el dispositivo móvil se bloqueé, gracias a las características especiales de los servicios de *Android*.

En el diagrama también se muestra la interfaz ***LocationUpdateCallback*** que representa un *listener* para recuperar las localizaciones obtenidas, de tal forma que con cada actuaalización de localización se **notifique** a los objetos de escucha que estén pendientes de estas actualizaciones. Es similar a los patrones *Observer* o *Listener* utilizados en los eventos de interfaz de usuario.

En cuanto a las alarmas de localización, se emplea un ***manager*** específico que se encarga de gestionar la programación de dichas alarmas a distintas horas, haciendo uso del *AlarmManager* integrado en el *framework* de *Android*. La clase ***LocationAlarmManager*** juega este papel utilizando una clase auxiliar denominada ***AlarmHelper***, la cual encapsula el código que interactúa con el *AlarmManager* de *Android* para programar una alarma a una hora determinada. De este modo, se desacopla el código de las dependencias concretas del *framework* de *Android*. El *AlarmHelper* programa una alarma que iniciará y dentendrá el **servicio de rastreo**, descrito anteriormente, a las horas indicadas. Cabe destacar, que desde este servicio de rastreo se accede al *manager* de alarmas de localización porque es necesario actualizar el estado de estas alarmas en la base de datos local del dispositivo una vez se inicia o detiene el rastreo.

De manera adicional, en la jerarquía de la interfaz del rastreador y sus implementaciones, se hace uso de la **orientación a objetos** para **reutilizar código** mediante una clase **abstracta**, aunque en el diagrama superior esto no se muestra ya que es simplemente una herramienta

de cara a la implementación para ahorrar líneas de código. A continuación, se muestra el fragmento del diagrama con la refactorización de la clase abstracta aplicada para reutilizar código.

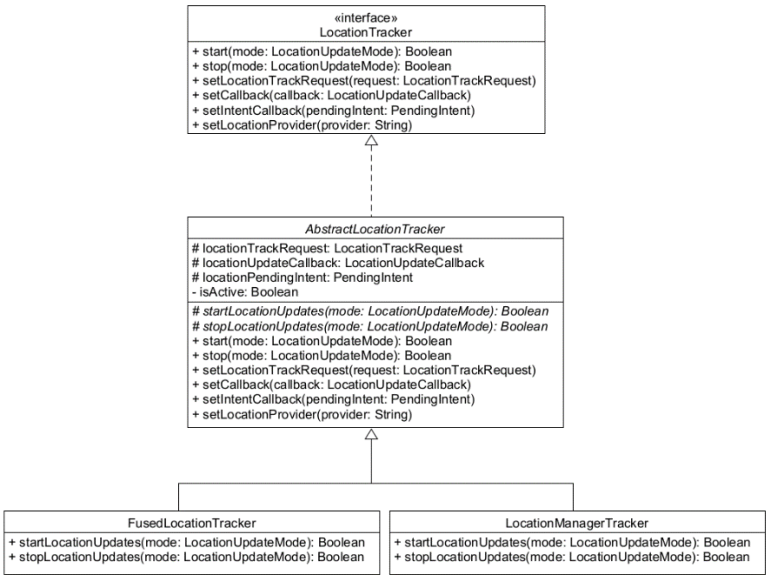


Figura 8.14. Clase Abstracta para reutilizar código en las clases de implementación del rastreo de ubicación.

En ambas implementaciones de la interfaz, existe código duplicado que se puede extraer a una clase abstracta a modo de **plantilla**, con lo cual podría considerarse una especie de ***Template Method*** donde la clase abstracta contiene la plantilla a seguir y las clases hijas redefinen los métodos abstractos necesarios para incluir su comportamiento característico. Como se puede observar en el diagrama, hay dos métodos abstractos, uno para iniciar el rastreo y otro para detenerlo, de forma que cada clase hija implemente estos métodos a su manera, una de ellas utilizando el *Fused Provider* y la otra mediante el *Location Manager*. Ambas clases heredan el resto de comportamiento de la clase abstracta, quien, a su vez, implementa la interfaz del rastreador de ubicación.

En este caso, esta refactorización es mera implementación y realmente no forma parte del diseño del sistema, aunque se menciona en esta sección dado que está relacionado con el diagrama de clases del rastreo.

8.2.1.3 *Comprobación de contactos*

De manera similar al rastreo, la comprobación de contactos de riesgo también se modela mediante un conjunto de clases relacionadas entre sí. En este caso no se utiliza ningún patrón de diseño determinado, sino que se hace uso de la propia orientación a objetos para guiar el diseño de la comprobación de contactos. A continuación, se muestra el diagrama de clases correspondiente a la comprobación de contactos de riesgo en la aplicación móvil.

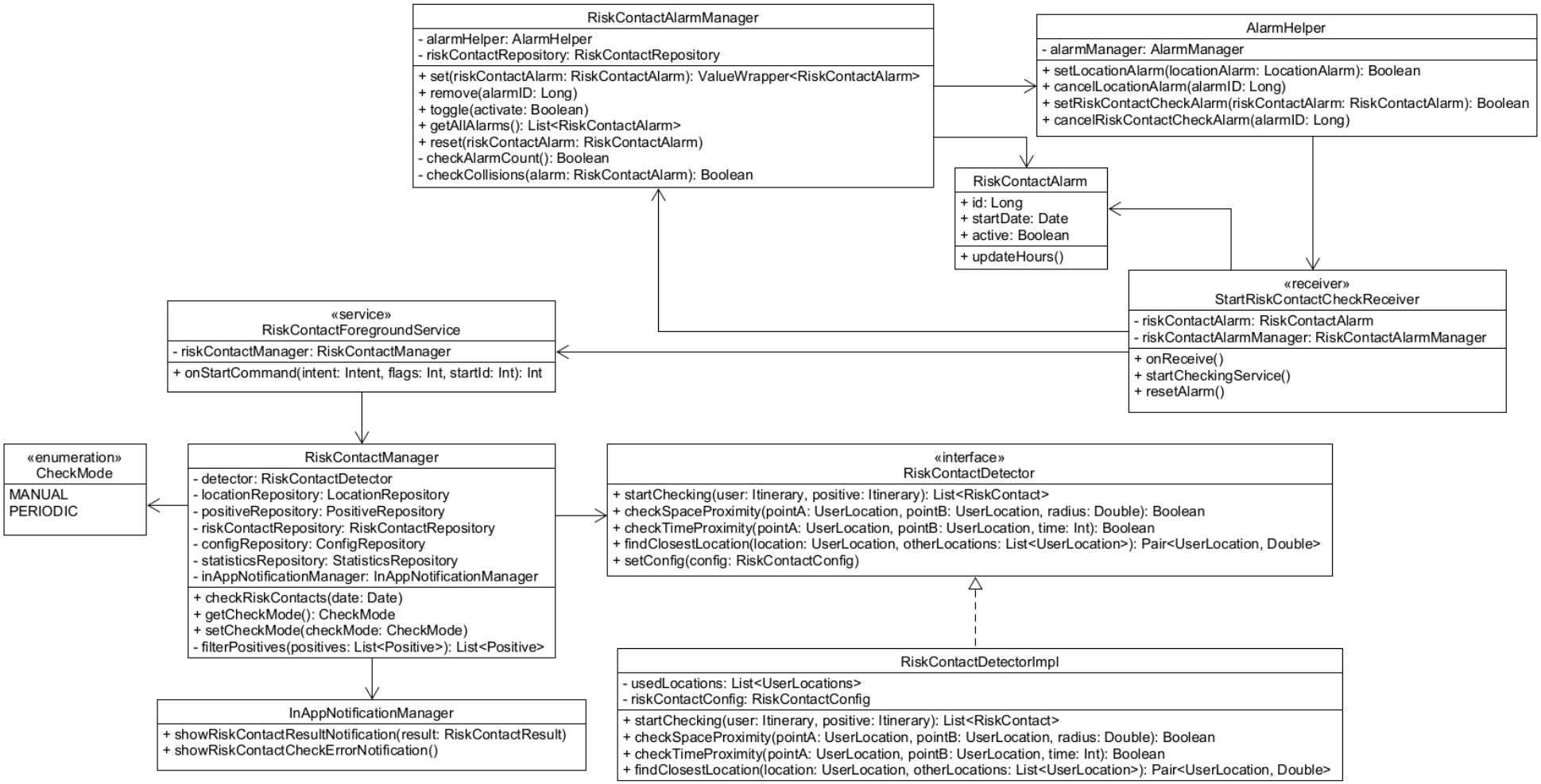


Figura 8.15. Diagrama de clases de la comprobación de contactos de riesgo en la aplicación móvil de rastreo.

En primer lugar, se encuentra el **detector** de contactos de riesgo modelado mediante una **interfaz *RiskContactDetector***, la cual establece las cabeceras de los métodos empleados en el **algoritmo de comprobación**. La responsabilidad de esta clase es comparar dos itinerarios conformados por localizaciones, y detectar los distintos pares de coordenadas en los que se aproximan lo suficiente en el tiempo y en el espacio como para ser considerado un contacto estrecho. Esta interfaz es implementada por una clase que alberga el código concreto del algoritmo, el cual puede ser implementado de distintas formas. Este podría considerarse de nuevo un patrón ***Strategy*** si en el futuro se decidieran añadir nuevas implementaciones del algoritmo de comprobación. Este detector de contactos se usa desde el ***manager*** de contactos de riesgo, que se encarga de recuperar los positivos de la nube y las localizaciones almacenadas en local que se registraron en los últimos días para construir los itinerarios e invocar al detector. De nuevo, aparecen dos modos, en este caso un modo de comprobación manual y otro de comprobación periódica a una hora determinada, donde podría emplearse un patrón *State*, aunque no aportaría demasiado.

En el modo de comprobación **periódica**, el *manager* de **alarmas de comprobación** hace uso del *AlarmHelper* para programar alarmas en el *framework* de *Android* que se disparen a ciertas horas para ejecutar una comprobación de contactos, de manera similar a como se hacía en las alarmas de localización. En este caso, cuando se dispara la alarma de *Android*, esta ejecuta un ***BroadcastReceiver*** que se encarga de reprogramar una nueva alarma a la misma hora para el día siguiente (de este modo la alarma se ejecuta diariamente) y después inicia un **servicio** en **primer plano** sobre el que se ejecuta el detector de contactos, a través del *manager* de contactos de riesgo. Cuando termina la comprobación se almacenan los resultados en la base de datos local. Todo esto se ve reflejado con claridad en el diagrama superior. Como ya se comentó, el servicio en primer plano sobrevive a los cambios de estado de la aplicación, incluso con el móvil bloqueado. De este modo, la comprobación de contactos se ejecutará a las horas especificadas sin importar que la aplicación esté encendida o apagada.

8.2.1.4 *Diagrama de clases de la arquitectura MVVM*

Para terminar con los diagramas de clases de la aplicación móvil, en esta última sección se presentan las clases que componen el **núcleo** estructural de la aplicación móvil de rastreo y que conforman la arquitectura ***MVVM*** descrita anteriormente. Estos diagramas de clases son un reflejo detallado de la estructura y los diagramas de paquetes vistos en la arquitectura de la aplicación móvil, de forma que siguen el mismo esquema basado en los modelos, vistas y vista-modelos, donde las actualizaciones de los componentes observables son notificadas a los observadores de las vistas. En ellos, se reflejan las vistas modeladas por los fragmentos y actividades de la aplicación, junto con los repositorios y otras abstracciones utilizadas para el acceso a la persistencia local y remota.

Como este diagrama de clases es muy extenso, se ha optado por dividirlo en dos partes, cada una de las cuales se muestra en páginas separadas para facilitar su lectura. A continuación, se presentan estas dos partes del diagrama de clases de la arquitectura *MVVM* para la aplicación móvil.

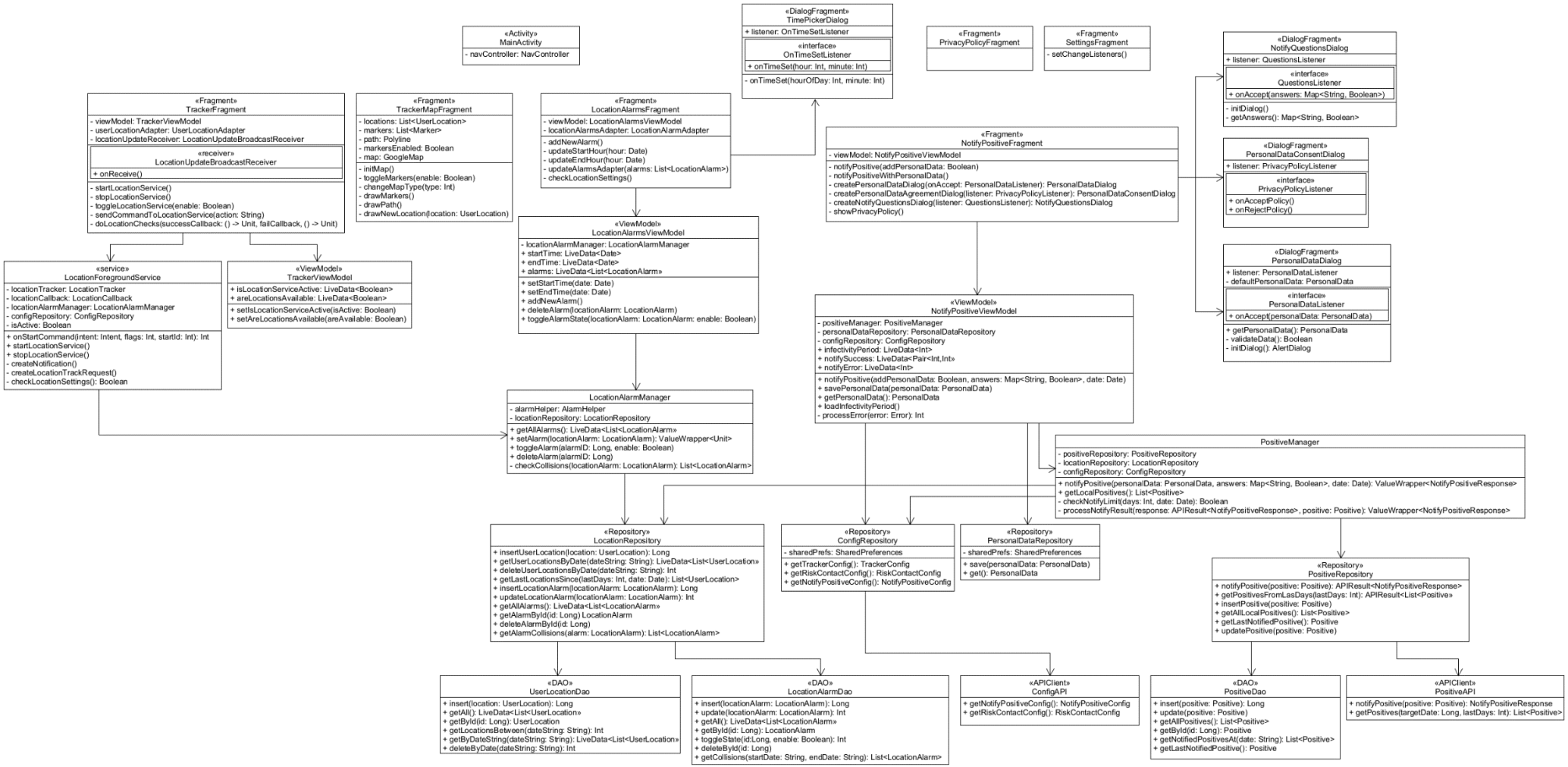


Figura 8.16. Diagrama de clases de la arquitectura MVVM de la aplicación móvil de rastreo (parte I).

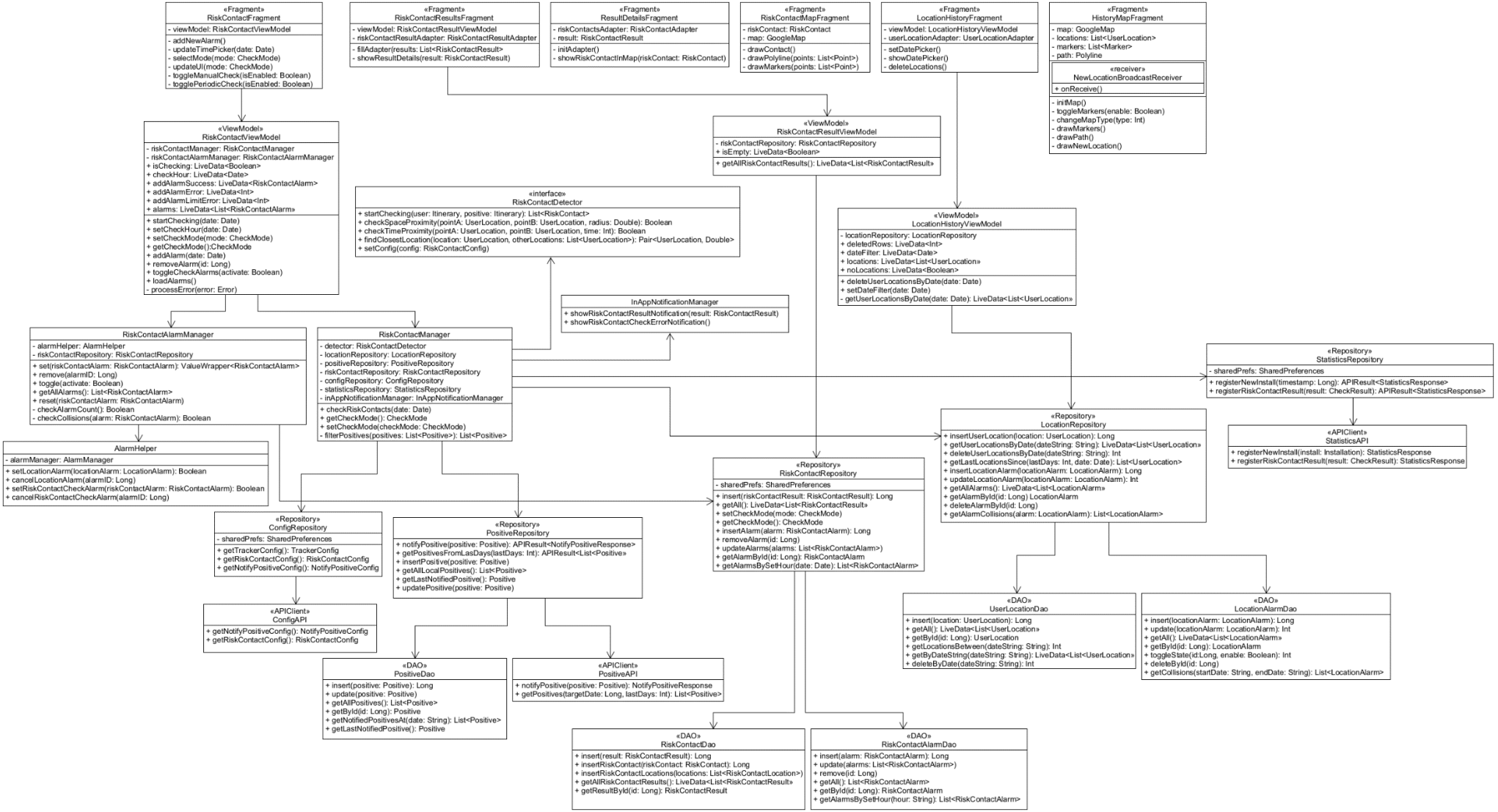


Figura 8.17. Diagrama de clases de la arquitectura MVVM de la aplicación móvil de rastreo (parte II).

Como se puede apreciar, la organización de los componentes sigue la estructura marcada por la arquitectura *MVVM* vista anteriormente. Cabe destacar que, en algunos casos, los *viewmodels* acceden directamente a los repositorios cuando no existe mucha lógica de negocio de por medio. En otros casos, los *viewmodels* acceden a un ***manager*** que hace de intermediario entre estos y los repositorios, y que alberga la lógica de negocio de algún aspecto de la aplicación. En cualquier caso, los *viewmodels* están desacoplados de las vistas gracias al patrón de diseño ***Observer***, que permite a las vistas actualizar sus componentes de la interfaz cada vez que se modifican los datos de los *viewmodels*, todo ello manteniendo un **bajo acoplamiento** del código y una **alta cohesión**, pues las responsabilidades están claramente separadas entre los distintos componentes.

8.2.2 Diagrama de Clases y Módulos del servidor *web*

En este caso, el subsistema del servidor *web* que alberga la *API REST* se desarrolla sobre un *framework* de *ECMASCRIPT6*, lo cual significa que se hará uso tanto de clases como de **módulos** para encapsular el código. Con ello, los diagramas de clases para el servidor *web* estarán formados por clases de *ECMASCRIPT* y también por módulos cuya organización interna es distinta. En los módulos, es necesario **exportar** los miembros que se desea exponer a los clientes que hagan uso del módulo, de tal forma que al importarlo ya se tiene acceso a dichos miembros sin necesidad de instanciarlo como las clases. A continuación, en la siguiente página se muestra el diagrama de clases *UML* para el servidor *web*.

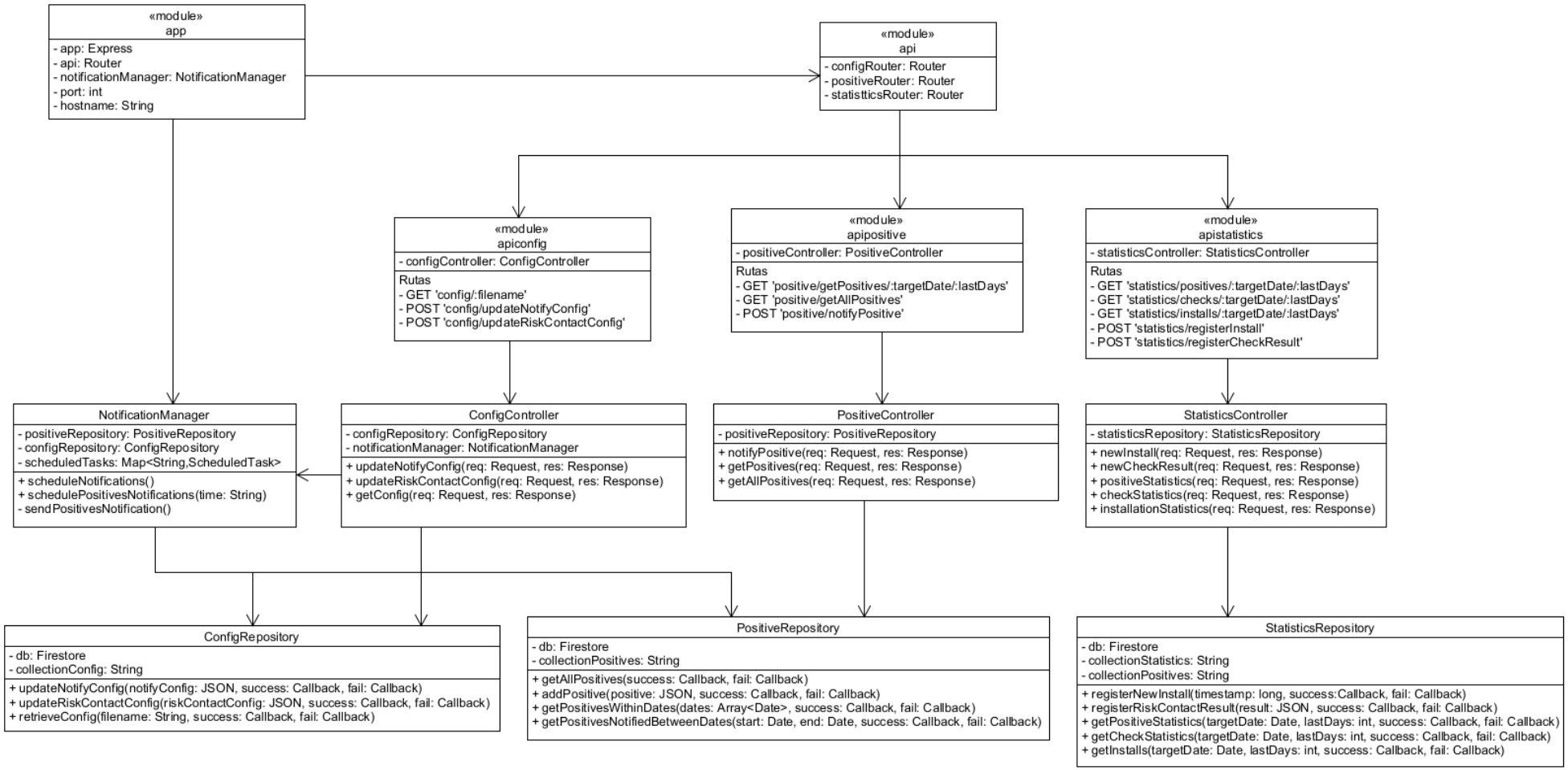


Figura 8.18. Diagrama de clases del servidor web.

Como se puede observar en el diagrama, el módulo ***app*** constituye la raíz del proyecto que se encarga de inicializar las dependencias y configurar el servidor para después comenzar con la escucha de peticiones. Además, configura el ***router*** del servidor que se encarga de emparejar las rutas con los controladores correspondientes. En el diagrama, la *API REST* está representada por los módulos de ***api*** que son los encargados de gestionar las rutas de los recursos y sus parámetros para enviarlos a los controladores, así como enviar las respuestas obtenidas a los clientes. Para cada una de estas *APIs* se muestran las rutas soportadas junto con sus verbos.

La estructura de las clases sigue la organización especificada en el apartado de la arquitectura del servidor *web*, donde las peticiones se gestionaban a través de los *routers* quienes las enviaban a los controladores y estos a su vez invocaban las operaciones de persistencia de los repositorios correspondientes.

Cabe destacar que la clase ***NotificationManager*** es la responsable de gestionar el envío de **mensajes *push*** a los clientes *Android*, utilizando el *SDK* de *Firebase* y programando tareas cronometradas en el tiempo, de forma que se puedan enviar las notificaciones automáticamente a una hora determinada.

Por último, en el nivel más bajo se encuentran los repositorios que acceden a la base de datos documental en la nube a través de la dependencia ***db*** que permite realizar las operaciones *CRUD* típicas.

8.2.3 Diagrama de Clases y Módulos de la aplicación *web*

De manera similar al servidor *web*, en los diagramas de clases de la aplicación *web* también aparece el concepto de ***módulo*** de *ECMASCRIPT* para albergar piezas de código con una responsabilidad común. Además, aquí también surge el concepto de **componente**, ya descrito anteriormente, y que aparecerá en el diagrama para representar los componentes de la interfaz *web*. En el diagrama, se verá que estos componentes **heredan** de un componente raíz denominado ***App,*** el cual contiene el componente *AppLayout* que define la infraestructura de la aplicación y su navegación a través del ***router***. Este *router* funciona como un mapa donde las entradas son las rutas y los valores son las vistas de la aplicación, de forma que cuando en la *URL* se especifica un ruta determinada, se sustituye el contenido principal por la vista asociada a esa ruta***.***

En la siguiente página, se muestra el diagrama de clases para la aplicación *web* de *Contact Tracker*.

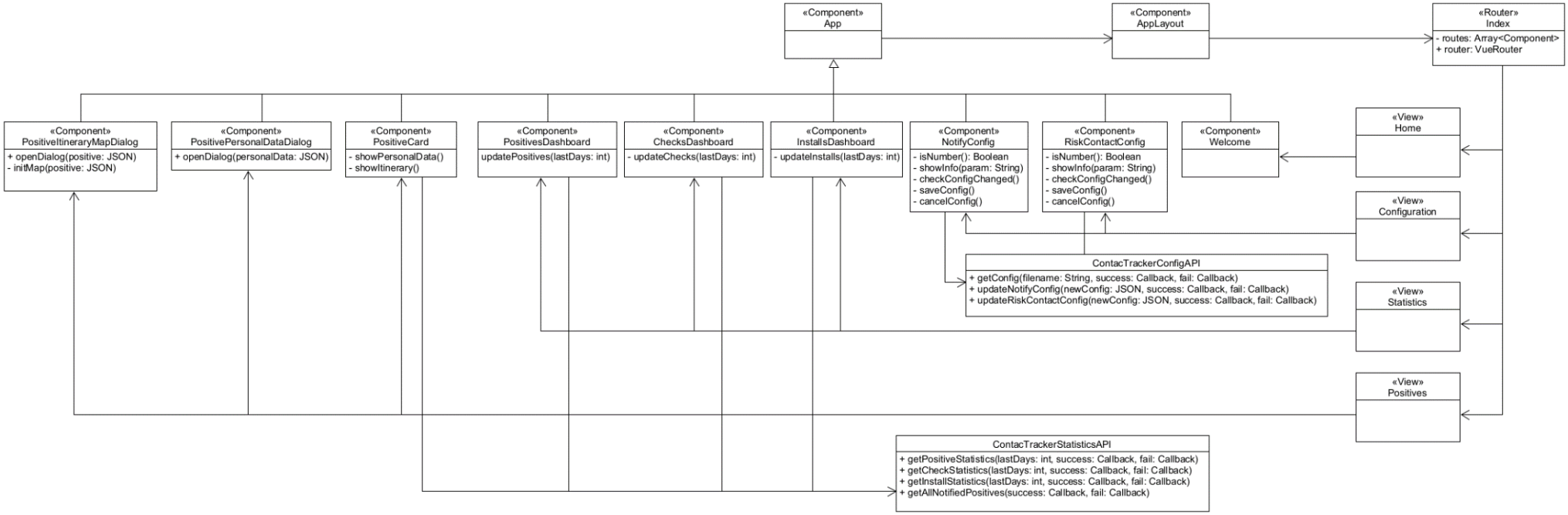


Figura 8.19. Diagrama de clases de la aplicación de panel de control web.

Como se puede apreciar, las vistas están compuestas por los distintos componentes *web*, quienes acceden a los clientes de la *API* para obtener y modificar los datos del servidor. Estos clientes de la *API* contienen los métodos con las peticiones *http* para interactuar con el servidor *web* a través de su *API REST*. Además, también se puede observar la relación de herencia entre el componente raíz *App* y el resto de los componentes que conforman las vistas.

Además, la organización de las clases y los módulos sigue el esquema marcado por la arquitectura de la aplicación *web* mostrada en las secciones anteriores, aunque no se sigue ningún patrón arquitectónico o patrón de diseño digno de mencionar.

Por último, los distintos componentes albergan en su interior las variables con los datos requeridos para su procesamiento y para ser mostrados en los elementos *html* empleando lo que se denomina ***two-way binding***, un enfoque similar a los *viewmodels* de *MVVM* pero en ambos sentidos, es decir, los elementos de la interfaz se actualizan automáticamente cada vez que se modifican los valores de los datos y viceversa, de modo que cada vez que se modifican los elementos de la interfaz como consecuencia de la interacción del usuario se actualizan automáticamente las variables con los datos asociados.

8.3 Diagramas de Secuencia

Con el objetivo de complementar la arquitectura y diseño del sistema vistos anteriormente, en esta sección se presentan los diagramas de **secuencia** correspondientes a las funcionalidades principales que ofrece la aplicación móvil. En los apartados anteriores, se hizo uso de diagramas *UML* **estructurales** para modelar la estructura **estática** del sistema. Ahora se utilizan diagramas *UML* de **comportamiento** para modelar los aspectos dinámicos del sistema, es decir, las relaciones e interacciones entre los componentes y clases que conforman los subsistemas.

Dado que la aplicación móvil es el módulo más complejo de los identificados en el análisis, solamente se presentan diagramas de secuencia para los casos de uso más interesantes de este subsistema. La aplicación *web* y el servidor *web* no tienen suficiente lógica de negocio como para modelar su comportamiento, por lo que carece de sentido utilizar diagramas de comportamiento en este caso.

Estos diagramas de comportamiento representan la transferencia de **mensajes** síncronos y asíncronos entre las instancias de las clases de un módulo, de modo que son un reflejo del sistema en **tiempo de ejecución**. Estos diagramas son lo más cercano que hay a la implementación final del código, y por tanto suelen tener bastante nivel de detalle hasta el punto de mostrar las cabeceras de posibles métodos a implementar. Así, estos diagramas modelan el paso de mensajes entre las instancias como consecuencia de una acción o petición realizada por un actor, en este caso el usuario estándar, a lo largo del tiempo.

En las siguientes subsecciones se muestran los diagramas de secuencia diseñados para modelar el comportamiento de los principales casos de uso del sistema.

8.3.1 Diagrama de secuencia del rastreo de ubicación

El rastreo de ubicación a través de la aplicación móvil ya fue modelado a través de un diagrama *UML* de clases en la sección 8.2.1.2.RASTREO DE UBICACIÓN. A modo de complemento, se desarrolla un diagrama de secuencia para modelar el paso de mensajes entre las clases vistas en dicho diagrama y así ver cómo son las invocaciones en el proceso para **activar/desactivar** el servicio de rastreo de ubicación. En la siguiente página se muestra este diagrama de secuencia.

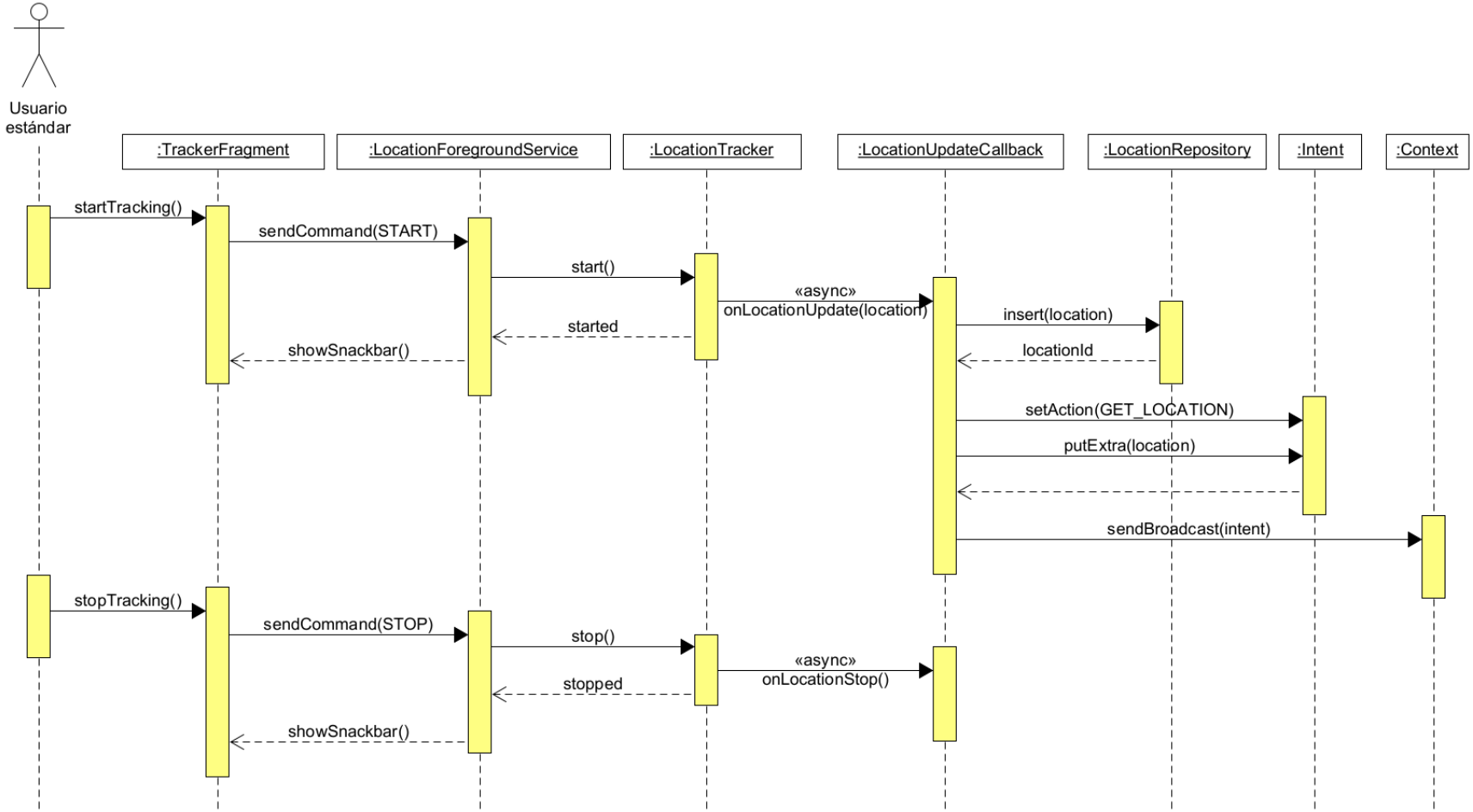


Figura 8.20. Diagrama de secuencia del rastreo de ubicación en la aplicación móvil de rastreo.

Como se puede apreciar en la figura superior, el usuario estándar inicia o detiene el rastreo de ubicación a través de la interacción con la interfaz de usuario, tras lo cual se envía un mensaje al fragmento asociado a dicha interfaz. Desde el fragmento de rastreo, se envía un **comando** de inicio (*START*) o de fin (*END*) al **servicio** *Android* en primer plano. Este a su vez, se comunica con la interfaz del ***tracker*** de ubicación, sin necesidad de acceder a sus implementaciones concretas, y le indica la orden de inicio o detención del rastreo.

El *tracker* ejecutará las operaciones de negocio necesarias para consumir los servicios de **geolocalización** del dispositivo y registrar un ***callback*** que reciba las actualizaciones de localización de dicho dispositivo. Este *callback* será la manera a través de la cual se recojan las coordenadas que se van actualizando periódicamente gracias al *framework* de *Android*. Como se puede observar, el *callback* de actualización es invocado de forma arbitraria por lo que se denota en el diagrama como ***async*** indicando que es un mensaje asíncrono. Con cada actualización de la posición del dispositivo, desde el *callback*, primero se **almacena** dicha localización en la base de datos interna a través del repositorio de localizaciones y luego se envía un ***broadcast*** con la nueva localización de forma que pueda ser leída por los clientes que registren un receptor para ese *broadcast,* para lo cual se requiere acceder al objeto ***Context*** de *Android*. Esto último es necesario para poder mostrar en la interfaz de usuario las localizaciones que se van registrando en tiempo real.

El proceso para detener el rastreo es análogo, pero excluyendo toda la parte de insertar la localización y enviar el *broadcast,* tal y como se aprecia en la FIGURA 8.20.

8.3.2 Diagrama de secuencia para la notificación de positivos

En este caso, la notificación de positivos solo fue modelada a través del diagrama de clases mostrado en el apartado 8.2.1.4.DIAGRAMA DE CLASES DE LA ARQUITECTURA MVVM, junto con el resto de los componentes de la arquitectura *MVVM*. A diferencia del rastreo, la notificación de positivos no se diseñó de forma individual dado que el *manager* de positivos solo se relacionaba directamente con los repositorios. En esta sección, se presenta un diagrama de secuencia que modela el **comportamiento** del *manager* de positivos a la hora de notificar un positivo en el sistema a través de la aplicación móvil. En la siguiente página se muestra dicho diagrama que incluye el paso de mensajes entre las instancias que interactúan entre sí para recuperar las coordenadas del usuario registradas en los últimos días y subirlas al servidor *web* en la nube.

Cabe destacar que en este diagrama se ha contemplado la notificación de un positivo adjuntando además sus **datos personales,** por lo que la lógica de negocio para solicitar dichos datos también aparecerá en el diagrama, aunque la notificación de positivos sin añadir los datos personales es muy similar.

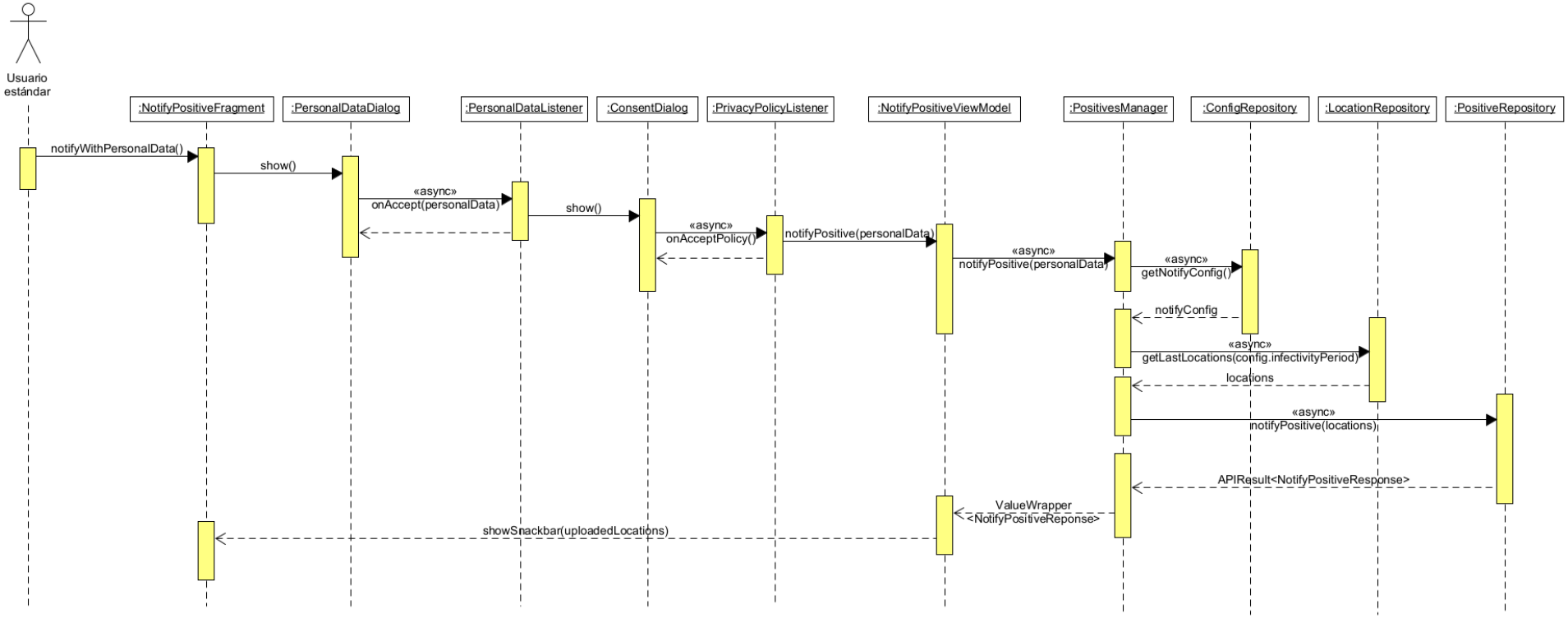


Figura 8.21. Diagrama de secuencia del procedimiento para notificar un positivo a través de la aplicación móvil.

En la figura superior, el usuario interacciona con la interfaz de usuario para solicitar la notificación de un positivo. El procedimiento se pone en marcha cuando el fragmento de notificación de positivos invoca al método ***show*** del **diálogo** de **datos personales**, en el cual el usuario introduce sus datos. Cuando el usuario pulsa siguiente, se invoca al ***listener*** de datos personales a través del método ***onAccept*** que recibe como parámetro un objeto con dichos datos. A continuación, se muestra el **diálogo** de **consentimiento** del tratamiento de datos personales, que también contiene un *listener* para la aceptación o rechazo de la política de privacidad. Si el usuario acepta, se invoca el método ***onAcceptPolicy*** del *listener* y ya se puede proceder con la propia notificación del positivo.

Una vez almacenados los datos personales, se invoca al ***viewmodel*** de notificación de positivos pasándole dichos datos. Este inicia una llamada asíncrona al *manager* de positivos que realiza una serie de operaciones con los repositorios:

1. Recupera la **configuración** de la notificación mediante una llamada de red a la ***API REST***, por medio del repositorio de configuración.
2. Recupera las **localizaciones** almacenadas en **local** que fueron registradas en los últimos días en función del periodo de infectividad leído de la configuración.
3. Construye el objeto positivo y lo envía al servidor *web* mediante una llamada de red a la *API REST* mediante el repositorio de positivos. De este modo, quedará almacenado en la nube.
4. El *manager* recibe el resultado de la *API REST*, representado por un objeto ***APIResult***, y lo envuelve en un ***ValueWrapper*** que es devuelto al *viewmodel*.

Tras completarse esta serie de pasos, el resultado de la *API* es procesado por el *viewmodel* quien invoca al método del fragmento encargado de mostrar un mensaje al usuario sobre el estado de la operación. En este caso, la operación puede ser exitosa o haber fallado por alguna razón como un **error** de **red**, error del **servidor**, error de **superación** del **límite** de notificación de positivos o bien un error por la inexistencia de **localizaciones** en el almacenamiento local del dispositivo.

8.3.3 Diagrama de secuencia para la comprobación de contactos de riesgo

Al igual que con el rastreo, la comprobación de contactos fue modelada mediante un diagrama de clases *UML* en la sección 8.2.1.3.COMPROBACIÓN DE CONTACTOS. En esta sección se propone una evolución de dicho diseño a través de un diagrama de secuencia que modela la transmisión de mensajes entre las instancias involucradas en la comprobación de contactos. Concretamente, se modela la recuperación de coordenadas tanto del usuario como de los positivos almacenados en la nube, para después ser comparadas por el algoritmo de comprobación y así detectar contactos de riesgo con otros positivos.

En la siguiente página se muestra este diagrama de secuencia para la comprobación de contactos de riesgo en la aplicación móvil de rastreo.

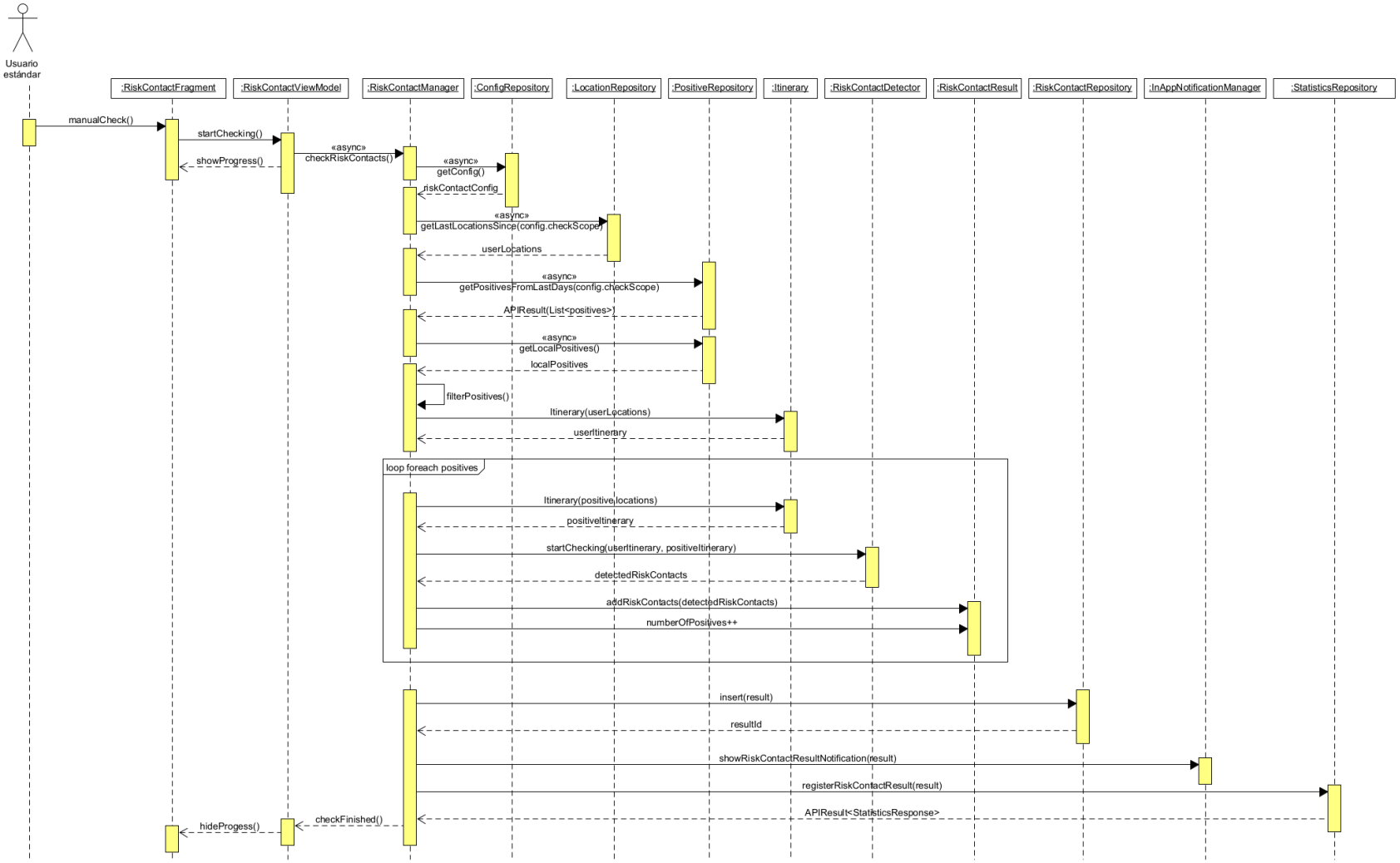


Figura 8.22. Diagrama de secuencia de la comprobación de contactos de riesgo en la aplicación móvil.

Como ya se comentó anteriormente, la comprobación puede realizarse en modo **manual** o periódico. En el diagrama superior, se modela la comprobación de contactos de forma manual, es decir, **bajo demanda** por el usuario estándar que interactúa con la interfaz para ejecutar una comprobación. De este modo, el fragmento de contactos de riesgo invoca al ***viewmodel*** a través del método ***startChecking()*** que inicia la comprobación de manera asíncrona. Durante la comprobación se muestra un indicador de progreso en la interfaz de usuario.

El *viewmodel* redirige la orden al ***manager*** de contactos de riesgo que es quien accede a los distintos repositorios para recopilar los datos necesarios para la comprobación y finalmente invocar al detector de contactos. A continuación, se enumeran y describen las operaciones que realiza este *manager:*

1. Recupera la **configuración** de la comprobación a través de una petición de red a la *API REST* mediante el repositorio de configuración.
2. Recupera las **localizaciones** en **local** registradas en los últimos días en función del alcance de la comprobación leído desde la configuración.
3. Recupera los **positivos** almacenados en la base de datos en la nube que tengan localizaciones registradas en los últimos días según el alcance.
4. Crea el **itinerario** del usuario a través de las localizaciones del dispositivo local.
5. Para cada positivo (bucle *foreach*), se crea un itinerario y se envía junto con el itinerario del usuario al **detector** de contactos, el cual ejecuta el algoritmo de comprobación y devuelve una lista con los **contactos de riesgo** detectados.
6. Esta lista se añade al objeto **resultado** y si no está vacía se incrementa el número de positivos con los que se ha tenido contacto.
7. Se **inserta** el resultado en la base de datos local junto con sus contactos de riesgo a través del repositorio de contactos.
8. Se muestra una **notificación** interna indicando el resultado obtenido tras la comprobación.
9. Por último, se hace una llamada a la *API REST* de **estadísticas** para registrar el resultado de la comprobación en la nube, de modo que el personal sanitario pueda consultar datos estadísticos sobre las comprobaciones.

Estas operaciones se ven reflejadas en la Figura 8.22. Una vez se ha terminado el proceso de comprobación, se notifica al *viewmodel* para que este a su vez indique al fragmento que debe esconder el icono de progreso de la comprobación. Ni el *viewmodel* ni el *manager* de contactos de riesgo devuelven un valor como resultado, sino que el resultado obtenido de la comprobación se almacena en la base de datos local para después ser leído por otro *viewmodel* que se encarga de mostrar un listado con los resultados de las comprobaciones realizadas.

En cuanto al **algoritmo de comprobación** implementado en la instancia del detector, este se describirá con más detalle en la futura sección dedicada a la implementación del sistema.

8.4 Diseño de la Base de Datos

A la hora de diseñar la persistencia en el sistema de *Contact Tracker* es necesario tener en cuenta **dos** sistemas gestores de bases de datos, uno para el almacenamiento interno de la

aplicación móvil y otro para la base de datos documental hospedada en la nube. Ambos se describen en esta sección, incluyendo los respectivos diagramas que muestran la organización interna de las tablas de la base de datos.

8.4.1 Descripción del SGBD Usado

En este apartado se describen los SGBD que se emplean para gestionar la persistencia de los datos tanto en la aplicación móvil como en el servidor *web* a través de la base de datos en la nube.

8.4.1.1 *SGBD para la aplicación móvil*

Para almacenar los datos en el almacenamiento interno de la aplicación móvil se utiliza una base de datos ***SQLite*** integrada en los dispositivos *Android*. Sin embargo, no se trabaja directamente con *SQLite*, sino que se utiliza una **capa de abstracción** intermedia que simplifica el código *boilerplate* para definir el esquema, las consultas y otros fragmentos de código necesarios para conectarse a la base de datos. Esta capa de abstracción recibe el nombre de ***ROOM***, una herramienta desarrollada por *Google* para trabajar con las bases de datos integradas en los dispositivos *Android*.

ROOM juega el papel de un **mapeador objeto-relacional** o ***ORM***, el cual se encarga de transformar las entidades del dominio, representadas por clases planas, en filas dentro de las tablas de la base de datos. Para ello, permite el uso de **anotaciones** en el código que simplifican la implementación para definir el **esquema** de la base de datos. Además, permite la verificación en tiempo de **compilación** de las consultas ***SQL***, las cuales se definen mediante anotaciones de manera declarativa, sin necesidad de implementarlas mediante código.

En la aplicación móvil, se hace uso de la **versión 2.3.0** de *ROOM*, además de incluir unas extensiones para trabajar con corrutinas.

8.4.1.2 *SGBD para la base de datos en la nube*

En cuanto al servidor *web*, este realiza sus tareas de persistencia a través de la interacción con la plataforma de ***Firebase***, también desarrollada por *Google*. Esta plataforma proporciona distintos servicios en la nube, entre los cuales se encuentra la base de datos ***Firestore*** que sigue un enfoque ***NoSQL***. Esto quiere decir que los datos se almacenan en **documentos** que se agrupan en **colecciones**, no existe un esquema como tal y tampoco hay **restricciones** de **integridad**, es decir, los documentos son flexibles y se pueden modificar tanto en contenido como en forma (organización de sus campos). Esto simplifica bastante las consultas, las cuales se implementan por código utilizando los métodos de la ***API*** de *Firestore.*

Las consultas de *Firestore* tienen una naturaleza asíncrona y están basadas en promesas, de forma que, para obtener los resultados sin bloquear la ejecución del programa, se utilizan *callbacks* de éxito y de fallo en cada operación que se lanza a la base de datos en la nube.

Para poder acceder a *Firestore*, es necesario vincular el proyecto ***node.js*** con la plataforma de *Firebase* a través de un proyecto, lo cual será explicado en la siguiente sección de integración del sistema con los SGBDs.

8.4.2 Integración del SGBD con el Sistema

Una vez vista una breve descripción de cada SGBD utilizado, en este apartado se explica cómo se integran estos con los subsistemas correspondientes.

8.4.2.1 *Integración de ROOM con la aplicación móvil*

En el caso de *ROOM*, para poder integrarse con la aplicación *Android* es necesario definir una clase abstracta que extienda de ***RoomDatabase*** y que contenga un método abstracto para cada *DAO* existente. Los *DAOs* son interfaces que definen las operaciones de persistencia, consultas, inserciones…*etc*. que se pueden realizar, agrupándolas por entidad del dominio.

De este modo, para construir una instancia de la base de datos se utiliza el *builder* integrado de *ROOM* al cual se le pasa la clase abstracta definida anteriormente de forma que este implemente los métodos abstractos para obtener los *DAOs*.

Con esto, ya se pueden obtener las referencias a los *DAOs* a través de los métodos abstractos de la instancia de la base de datos. Como ya se mencionó antes, el esquema de la base de datos se define dinámicamente mediante anotaciones que se sitúan sobre las clases planas del dominio y sus miembros. En aquellos casos que se quiera representar relaciones complejas de uno a muchos o muchos a muchos, es necesario, emplear clases auxiliares que modelen estas relaciones a través de las claves primarias de las entidades.

8.4.2.2 *Integración de Firestore con el servidor web*

Para el servidor *web*, es necesario crear un proyecto en *Firebase* y vincularlo con el proyecto *node.js*. Existen diferentes formas de hacerlo, en el contexto de este proyecto se hace uso de un fichero ***json*** generado por la plataforma *Firebase* que sirve a modo de certificado para autenticar al proyecto en la plataforma y que se almacena dentro del proyecto *node.js*. De este modo, se hace uso de un módulo *javascript* para realizar la **configuración inicial** para conectarse a *Firebase*, leyendo los datos del fichero *json*. Estos datos se utilizan para autenticarse mediante el *SDK Admin* de *Firebase*.

Hecho esto, ya se puede acceder a la instancia de la base de datos *Firestore* para lanzar peticiones *CRUD* mediante su *API*.

8.4.3 Diagrama E-R

En este último apartado se presentan los diagramas entidad-relación que representan los esquemas de las bases de datos utilizadas. Para la aplicación móvil, se hará uso del estándar de diagramas E-R para definir el esquema relacional de las entidades. Sin embargo, para la base

de datos documental que sigue un enfoque *NoSQL*, el estándar E-R se sustituye por un boceto simple que muestra la organización de los documentos.

8.4.3.1 *E-R de la aplicación móvil*

La siguiente figura muestra el diagrama entidad-relación que representa el esquema seguido en la base de datos local integrada en cada dispositivo *Android* que tenga instalada la aplicación.

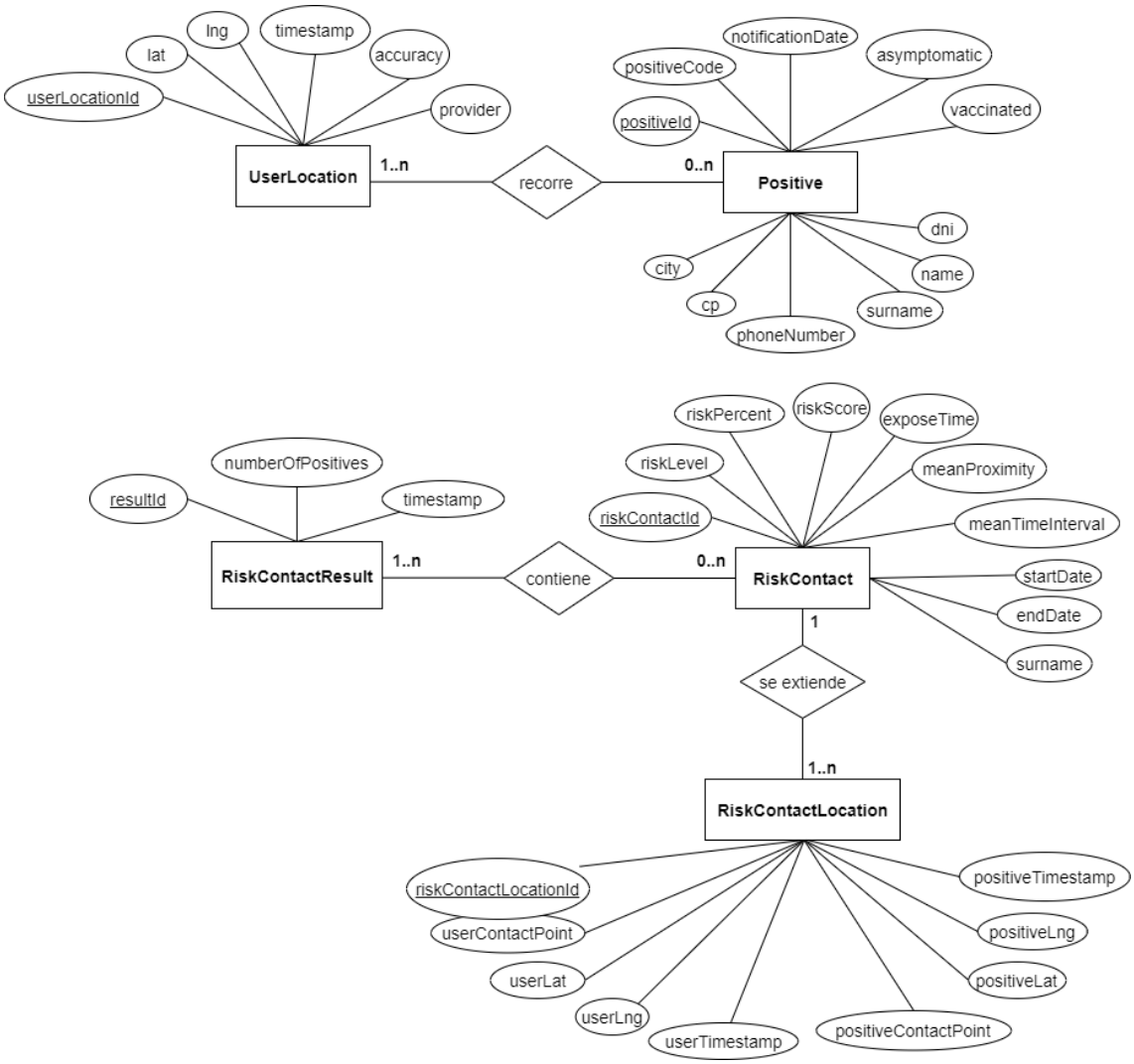


Figura 8.23. Diagrama Entidad-Relación para la aplicación móvil de rastreo.

A partir de este diagrama E-R, se comenzaría a desarrollar el correspondiente **diagrama de tablas** de más bajo nivel y que descompusiera las relaciones 1..n y n..m para facilitar su implementación a través del SGBD. Sin embargo, para evitar extenderse demasiado y dado que no aporta demasiada información de cara a la documentación, se ha decidido no incluir aquí este diagrama de tablas.

8.4.3.2 *Esquema de documentos de la base de datos en la nube*

En este caso no existe ningún estándar para definir diagramas de bases de datos *NoSQL*, por lo que se hace uso de un simple boceto que muestra de forma visual el esquema de **colecciones** y **documentos** que se emplea en el sistema. A continuación, se muestra la organización de los documentos en colecciones para la base de datos en la nube.

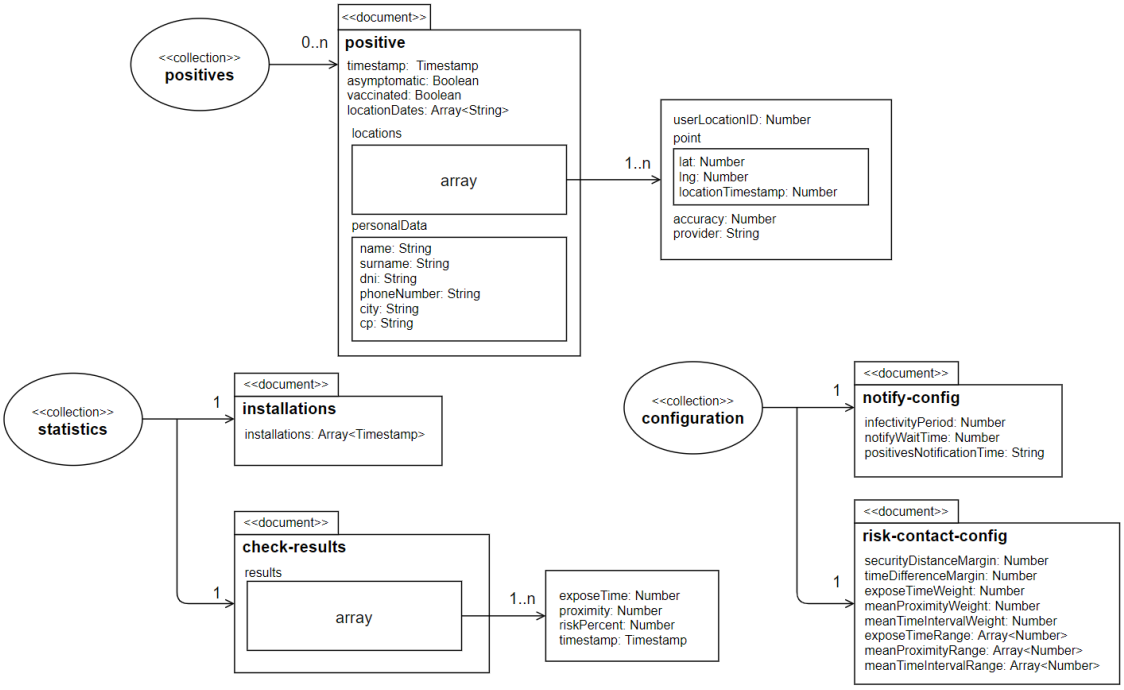


Figura 8.24. Diagrama del esquema de documentos y colecciones de la base de datos central en la nube.

En la figura superior, se puede apreciar cómo se organizan los documentos en las distintas colecciones, concretamente en las tres colecciones existentes. En el diagrama, la naturaleza de los componentes está definida mediante un **estereotipo** entre << >> que indica si se trata de un documento o una colección. Además, es importante destacar, que en el documento que modela un positivo, el objeto de datos personales (***personalData***) es **opcional**, es decir, un positivo puede contener o no los datos personales del usuario que lo ha notificado.

8.5 Diseño de la Interfaz

Las anteriores secciones mostraban el diseño de la estructura interna del sistema, mientras que en esta sección se presenta el diseño final de las interfaces de usuario de los dos subsistemas que ofrecen servicios a los usuarios: la aplicación móvil y el panel de control *web*. Estos diseños de la interfaz están basados en los bocetos presentados en el apartado 7.5 ANÁLISIS DE INTERFACES DE USUARIO en el capítulo dedicado al análisis, y por tanto constituyen una evolución de los mismos incluyendo un mayor nivel de detalle y ciertos cambios con respecto a los bocetos.

No se hará una descripción detallada de cada pantalla ya que no es el objeto de esta sección y además ya se han descrito anteriormente los bocetos en el análisis. Únicamente en aquellos casos que hayan surgido cambios o sea excesivamente necesario se describirán las pantallas.

8.5.1 Interfaz de la aplicación móvil

En esta primera sección, se presentan los diseños finales de las pantallas de la aplicación móvil de rastreo agrupadas por caso de uso.

8.5.1.1 *Rastreo de ubicación*

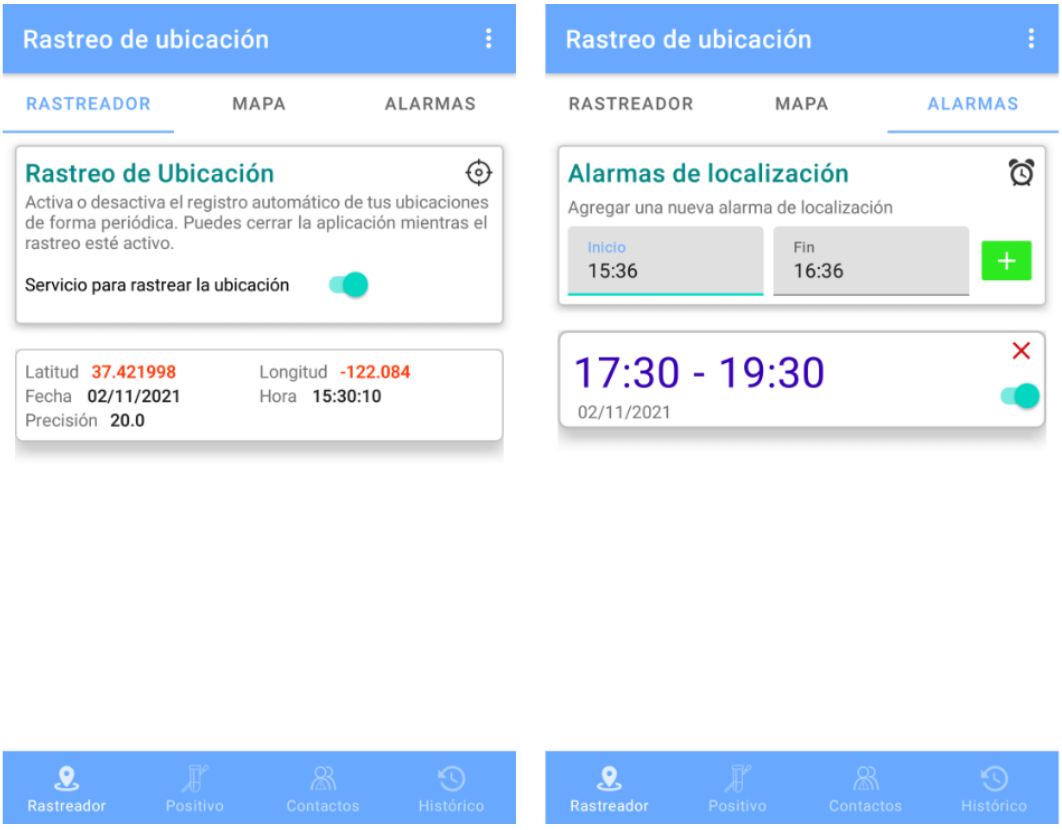


Ilustración 8.1. Diseño final de las vistas del rastreador de ubicación (izquierda) y de la programación de alarmas de localización (derecha).

En la figura superior izquierda, se puede activar o desactivar el servicio de rastreo de ubicación mediante el *switch*. Al hacerlo, se mostrará un mensaje indicando esta acción y se mostrará una **notificación** cuando se active el servicio en primer plano. En esta notificación, se indica que el servicio de ubicación está activo y que registra coordenadas cada cierto número de minutos y segundos.

En la figura superior derecha, se muestra la vista con las alarmas de localización programas, donde a través del *switch* se pueden activar/desactivar las alarmas y mediante la cruz se pueden eliminar.

El cambio más notable en la interfaz del rastreo de ubicación es la inclusión de un **mapa** que va mostrando el itinerario seguido por el usuario en tiempo real, de forma que las localizaciones que se van registrando se dibujan en el mapa mediante **marcadores** y una línea que une las coordenadas registradas. Para acceder a este mapa basta con pulsar sobre la **pestaña** *Mapa*. El diseño de esta nueva vista se muestra a continuación.

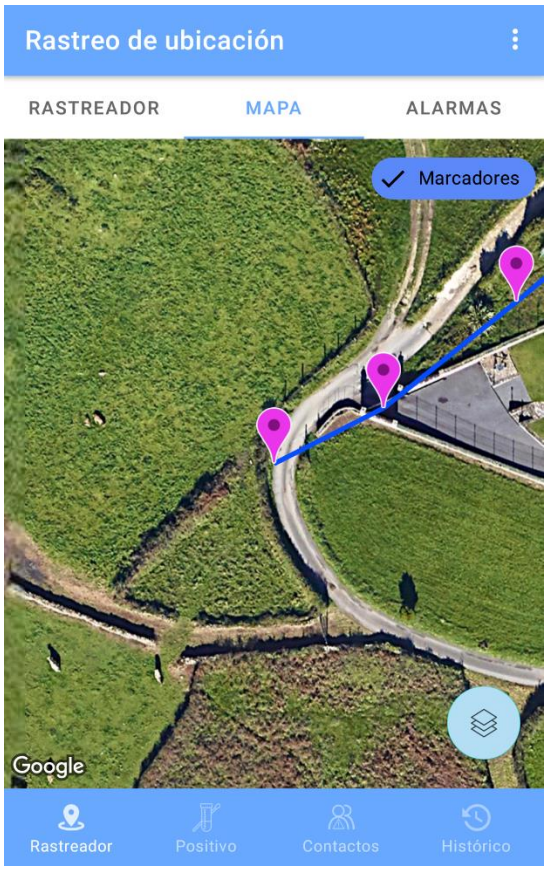


Ilustración 8.2. Diseño final de la vista del mapa del rastreo de ubicación en tiempo real.

El botón de ***Marcadores*** permite ocultar o mostrar los marcadores que representan las localizaciones que han sido registradas.

8.5.1.2 *Notificación de positivos*

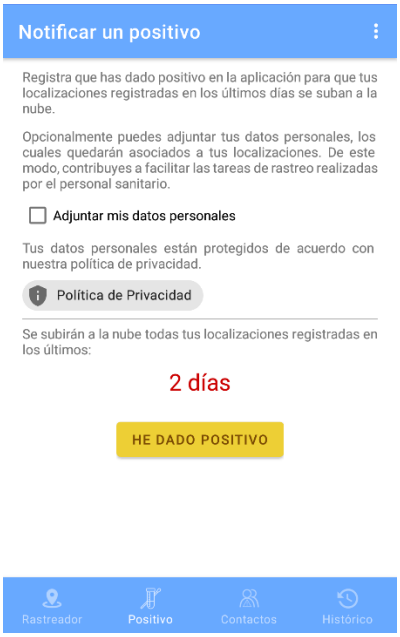


Ilustración 8.3. Diseño final de la pantalla de notificación de positivos.

En esta ilustración, se aprecia la pantalla para notificar un positivo, la cual ha sufrido ciertos cambios desde su boceto inicial en el análisis. Por un lado, se muestra un botón que lleva directamente a la **política de privacidad** de la aplicación, y por otro, se indica al usuario el **periodo de infectividad**, es decir, el número de días que se tienen en cuenta desde el día actual hacia atrás para subir las localizaciones registradas a la nube. Al pulsar sobre el botón para notificar un positivo, se comienza el proceso para subir las localizaciones a la nube, pudiendo indicar los datos personales tal y como se muestra en las siguientes ilustraciones.

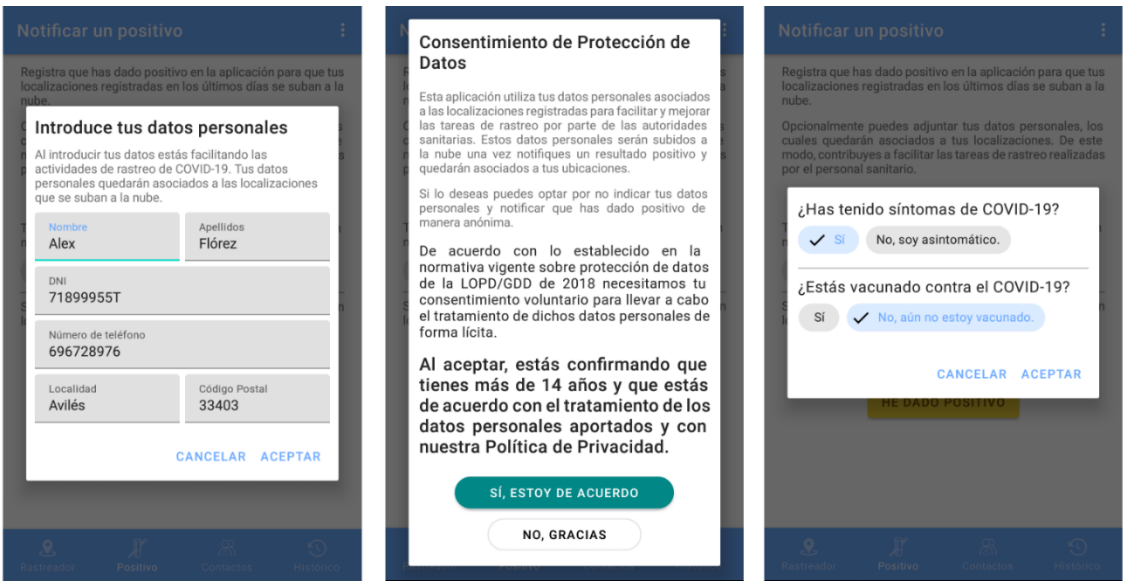


Ilustración 8.4. Diseño final de las pantallas del procedimiento para notificar un positivo.

Como se puede apreciar, el primer paso es indicar los datos personales a través del formulario, en caso de que se hubiera marcado la casilla de datos personales. Si existe algún campo vacío,

no se permite avanzar en el proceso y se muestra un error en rojo debajo del campo que falta. A continuación, se muestra la **cláusula** de consentimiento del tratamiento de datos que el usuario puede aceptar o rechazar. En caso de aceptar, se muestra un último diálogo con unas **preguntas** generales sobre si se es asintomático y si se está vacunado contra *COVID-19* con objetivos estadísticos.

8.5.1.3 *Comprobación de contactos de riesgo*

Ilustración 8.5. Diseño final de la pantalla de comprobación de contactos de riesgo.

Este es el diseño final de la pantalla para ejecutar una comprobación de contactos de riesgo. Existen dos modos de comprobación: la comprobación manual permite ejecutar directamente los contactos de riesgo cuando el usuario lo desee, mientras que la comprobación periódica permite especificar una hora concreta del día en la que se debe ejecutar la comprobación de forma diaria. Al activar un modo, el otro queda desactivado y viceversa, tal y como se aprecia en la ilustración superior donde se ven ambos modos de comprobación. En el modo periódico (derecha), se van mostrando las alarmas establecidas, hasta un límite máximo de **tres alarmas** de comprobación.

Los resultados de las comprobaciones ejecutadas se muestran en forma de listado tal y como se indica en las siguientes ilustraciones.

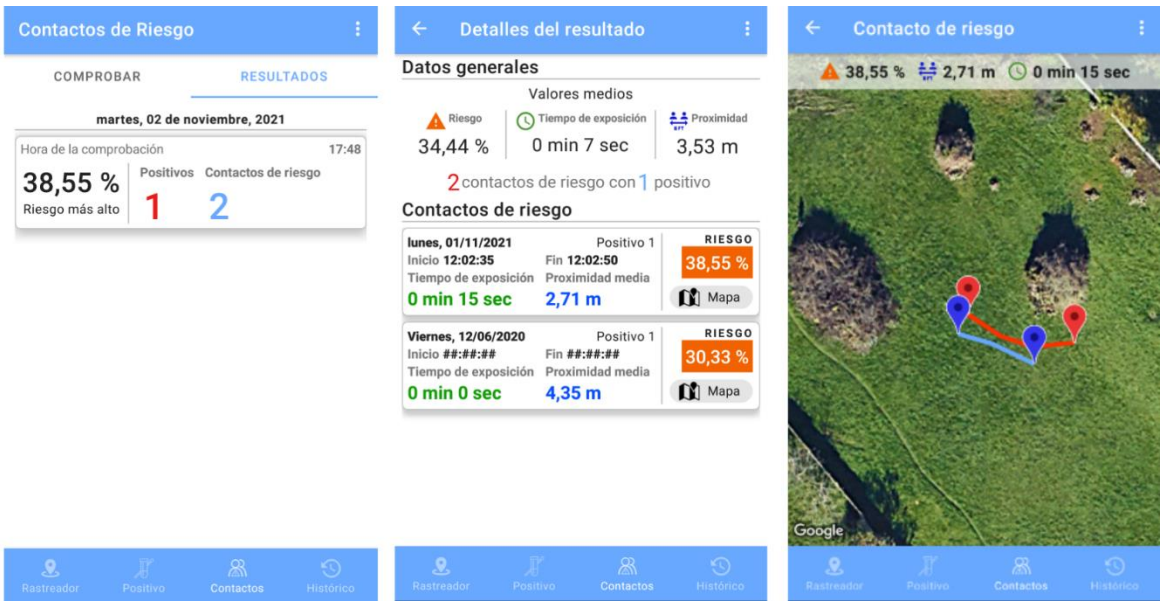


Ilustración 8.6. Diseño final de las pantallas de los resultados de la comprobación de contactos.

La primera pantalla muestra el listado de los resultados de las comprobaciones ejecutadas. En esta versión final, los resultados se agrupan por **días**, en función del día en el que se ejecutaron las comprobaciones. Al pulsar sobre un resultado, se abren sus **detalles**, lo cual se muestra en la ilustración central. En estos detalles, se indican los valores **medios** de tiempo de exposición, proximidad y porcentaje de riesgo entre todos los contactos de riesgo detectados. Además, se indica con cuántos positivos se ha entrado en contacto. Para cada contacto de riesgo, se indican los valores de tiempo de exposición, proximidad y porcentaje de riesgo, además de indicar la hora de inicio y de fin aproximada del contacto.

Por último, al pulsar sobre un contacto de riesgo, se abre la pantalla del **mapa** donde se muestra el tramo de **ruta** del usuario y del positivo en donde tuvo lugar el contacto, tal y como se muestra en la ilustración derecha.

8.5.1.4 *Histórico*

A continuación, se muestra el diseño final de las pantallas del histórico en la aplicación móvil de rastreo.

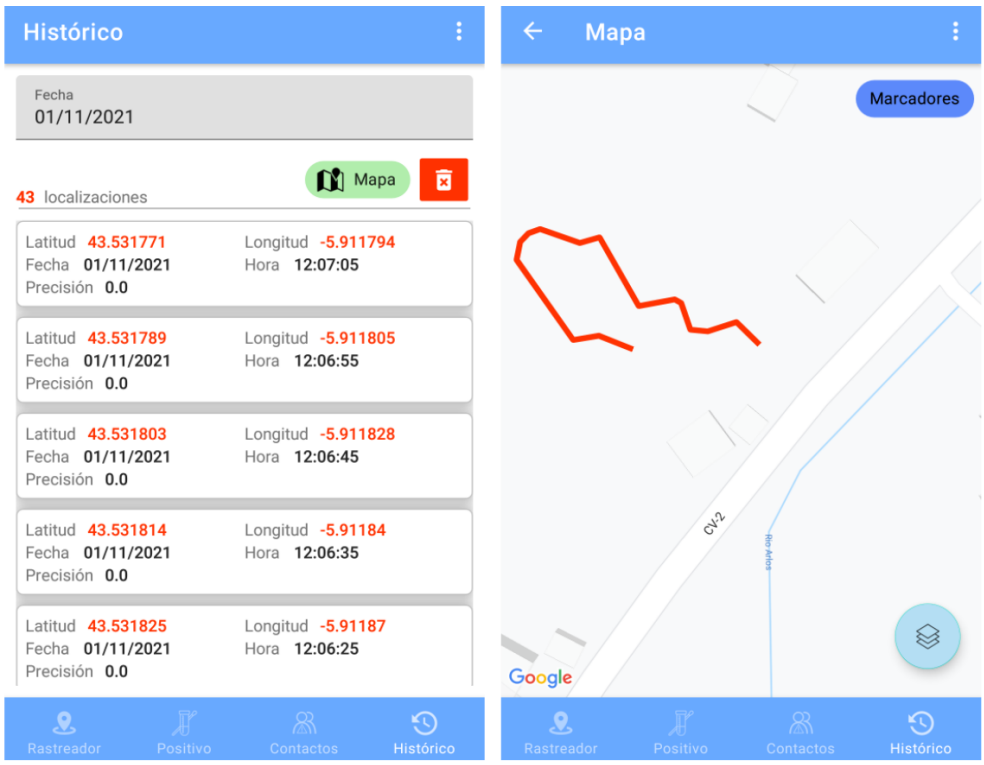


Ilustración 8.7. Diseño final de las pantallas del histórico de localizaciones.

Como se puede apreciar, los cambios más significativos en estos diseños finales son, por un lado, que se muestra el **número** total de localizaciones que hay registradas en el día seleccionado en el campo de texto, y por otro, la organización de los botones para mostrar el itinerario en un mapa y para eliminar las localizaciones de ese día. En la ilustración derecha, se muestra el mapa en el que se visualizan las localizaciones registradas en el día seleccionado.

Aunque aquí no se muestra, para seleccionar la fecha se muestra al usuario un diálogo con un **calendario** que permite escoger día, mes y año para filtrar la búsqueda de las localizaciones registradas en la fecha indicada.

8.5.1.5 *Ajustes Generales*

Para terminar con el diseño de interfaces de la aplicación móvil, se muestra a continuación el diseño final de los ajustes generales de la aplicación junto con la política de privacidad.

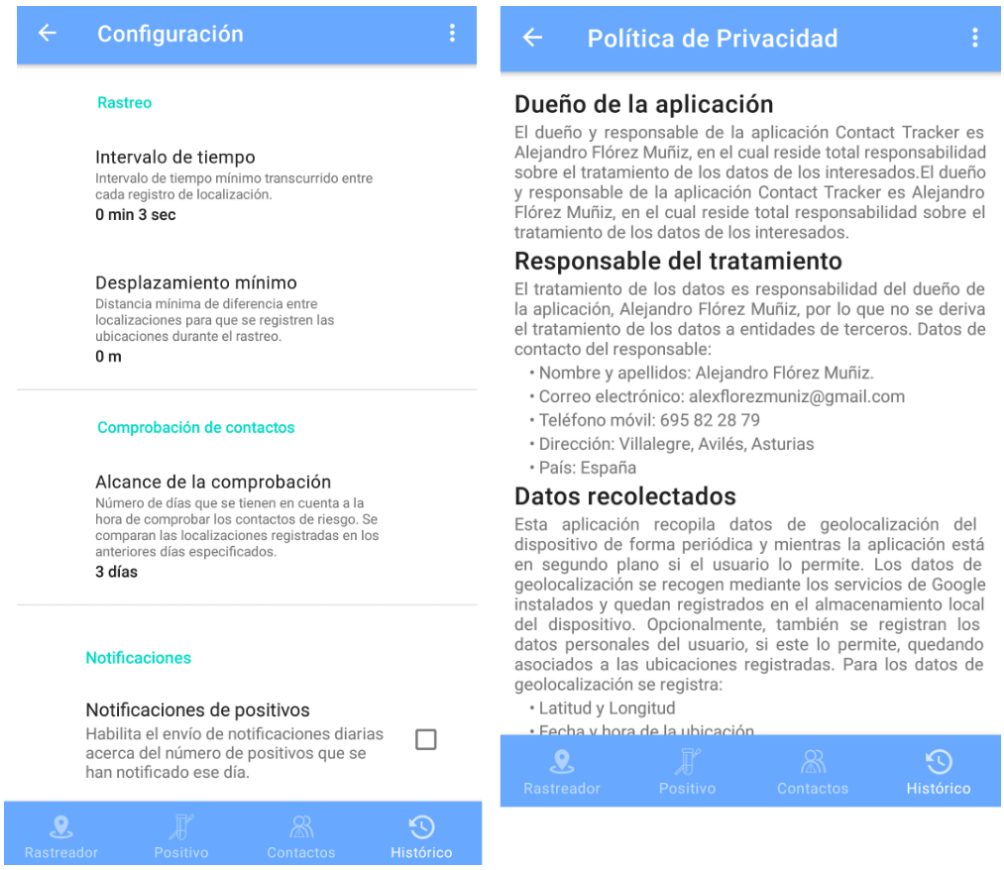


Ilustración 8.8. Diseño final de las pantallas de ajustes generales y de política de privacidad.

Para acceder a estas dos pantallas, se hace uso del menú de opciones de los tres puntos, situado en la esquina superior derecha de la aplicación. A la izquierda, se muestra la pantalla de ajustes generales de la aplicación, desde donde se pueden modificar los parámetros del rastreo, el alcance de la comprobación o activar/desactivar el envío de notificaciones sobre el número de positivos notificados. A la derecha, se muestra la pantalla que contiene las cláusulas principales de la política de privacidad de la aplicación y que tratan sobre el tratamiento de datos del usuario dentro de la aplicación.

8.5.2 Interfaz de la aplicación web

En esta segunda sección del diseño de la interfaz de usuario del sistema, se presentan los diseños finales de las vistas de interfaz de usuario para el panel de control *web* del personal sanitario. Antes de pasar a ver cada pantalla en concreto, a continuación, se muestra el diseño final de la pantalla de bienvenida de la aplicación *web* que no había sido contemplado en los bocetos preliminares del análisis.

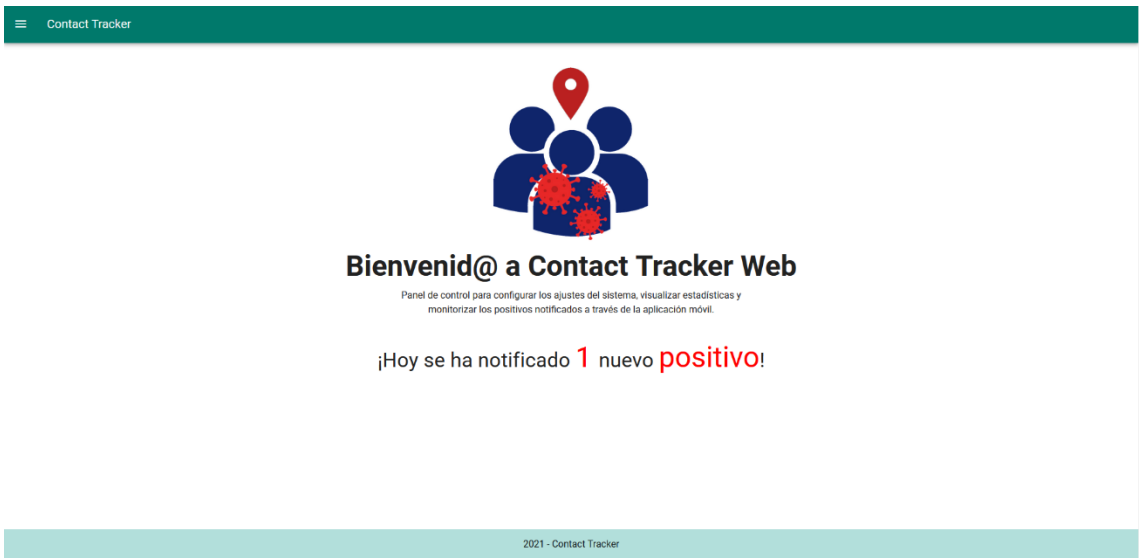


Ilustración 8.9. Diseño final de la pantalla de bienvenida del panel de control web.

Como se puede apreciar, el menú lateral que aparecía en el boceto inicial ahora aparece colapsado, y se accede a él mediante el botón situado en la esquina superior izquierda. Además, en esta pantalla de bienvenida se muestra un mensaje con el número de positivos que se notificaron en el día de hoy, si hay alguno.

8.5.2.1 *Configuración de la notificación de positivos*

La siguiente ilustración muestra el diseño final para la pantalla de configuración de la notificación de positivos en el panel de control *web*.

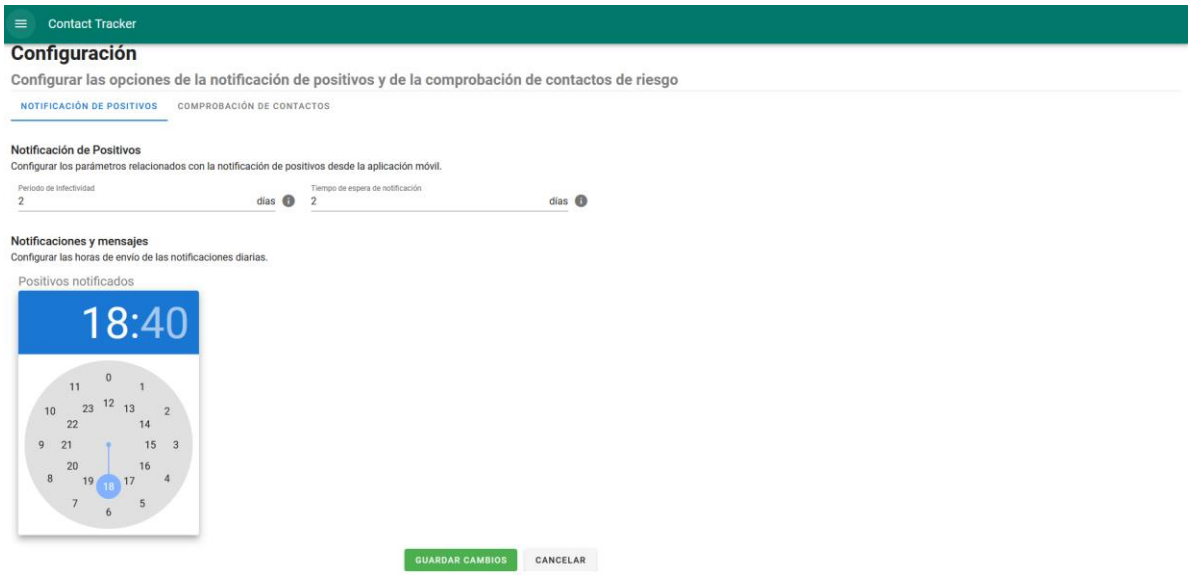


Ilustración 8.10. Diseño final de la pantalla para modificar la configuración de la notificación de positivos.

El diseño de esta pantalla se asemeja bastante al boceto preliminar realizado en el análisis. En este caso, cabe destacar los botones para **guardar** o **cancelar** los cambios que aparecen en la

interfaz cuando el valor de alguno de los parámetros es modificado por el usuario administrador.

8.5.2.2 *Configuración de la comprobación de contactos*

La siguiente ilustración muestra el diseño final de la pantalla para configurar los parámetros de la comprobación de contactos de riesgo en el panel de control *web.*

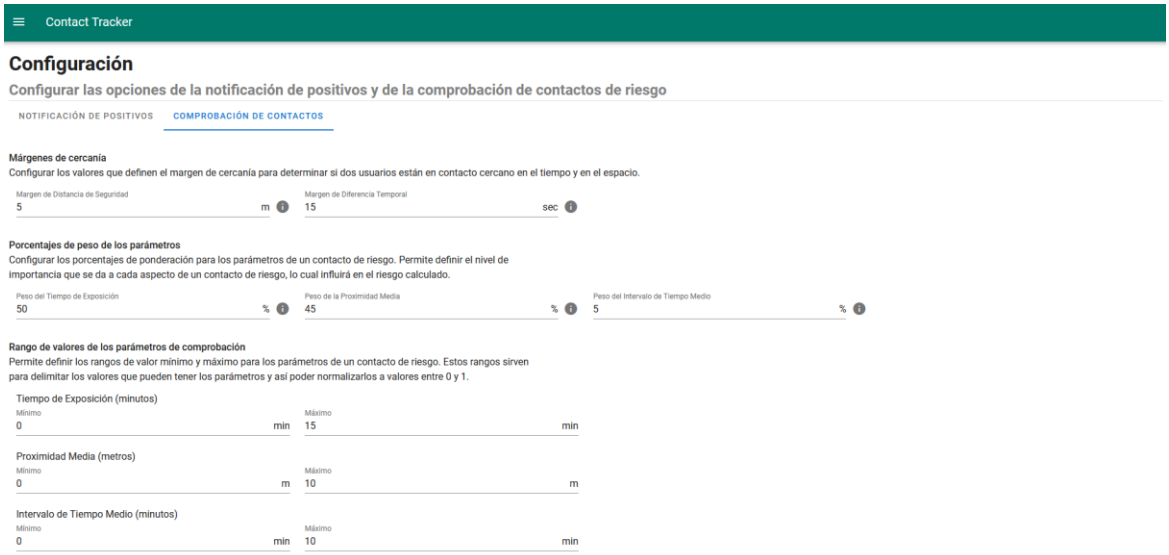


Ilustración 8.11. Diseño final de la pantalla de configuración de la comprobación de contactos.

En este caso, el diseño final también es bastante afín con el modelo preliminar del análisis, la única diferencia notable es de nuevo la pareja de botones para guardar o cancelar los cambios y la **descripción** explicativa de cada uno de los parámetros de configuración, que en el boceto inicial no aparecía.

8.5.2.3 *Visualización de estadísticas*

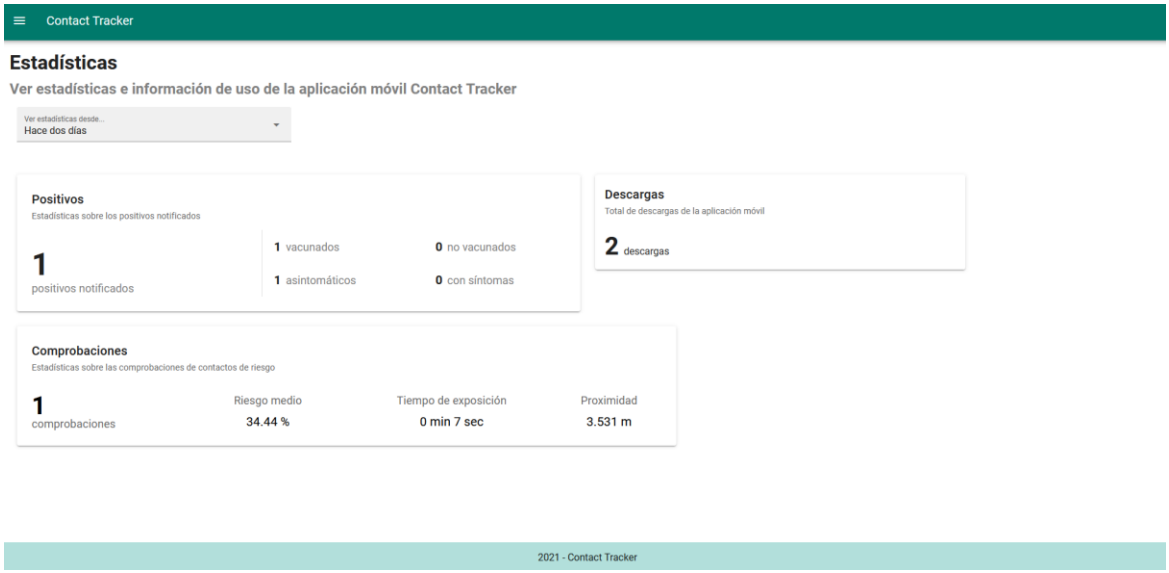


Ilustración 8.12. Diseño final de la pantalla de estadísticas en el panel de control web.

La ilustración superior muestra el diseño final de la interfaz de usuario para visualizar las estadísticas del sistema, de nuevo muy similar al boceto inicial realizado en el análisis. La única diferencia notable es la **descomposición** de los positivos notificados en el número de **vacunados** y de **asintomáticos**. El resto de los paneles es muy similar al boceto preliminar, como el panel donde se muestra el número de instalaciones de la aplicación móvil, o el panel del resumen de comprobaciones, donde se muestra la media de los resultados obtenidos entre las comprobaciones realizadas por los usuarios.

8.5.2.4 *Visualización de positivos*

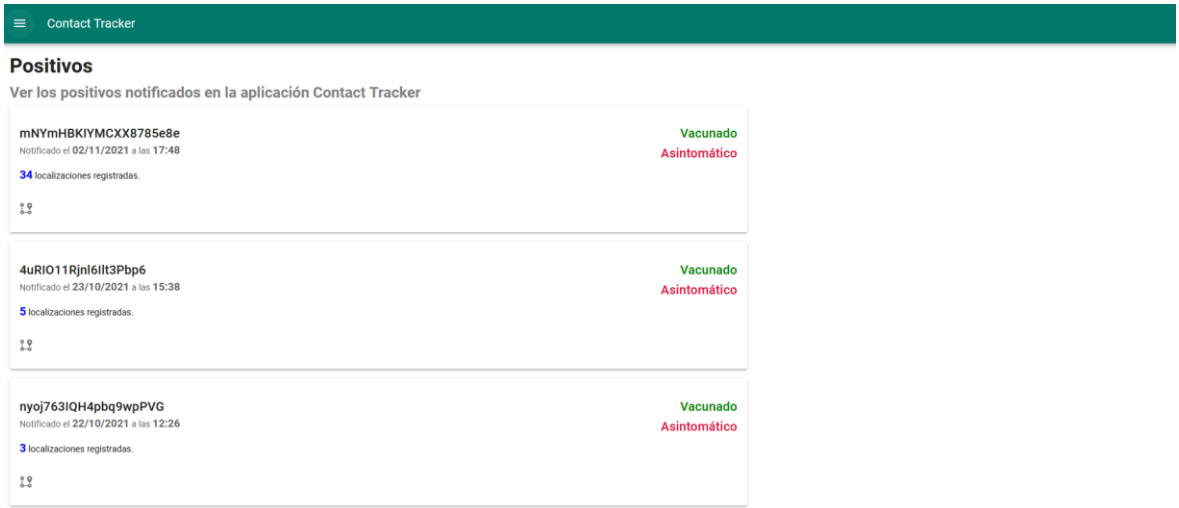


Ilustración 8.13. Diseño final de la pantalla de visualización de positivos.

Esta ilustración muestra el diseño final para el listado de positivos notificados en el sistema. Para cada positivo se indica si está vacunado y si es asintomático, la única diferencia notable

con el boceto preliminar. Al pulsar sobre el icono para ver los datos personales de un positivo se muestra el siguiente diálogo.

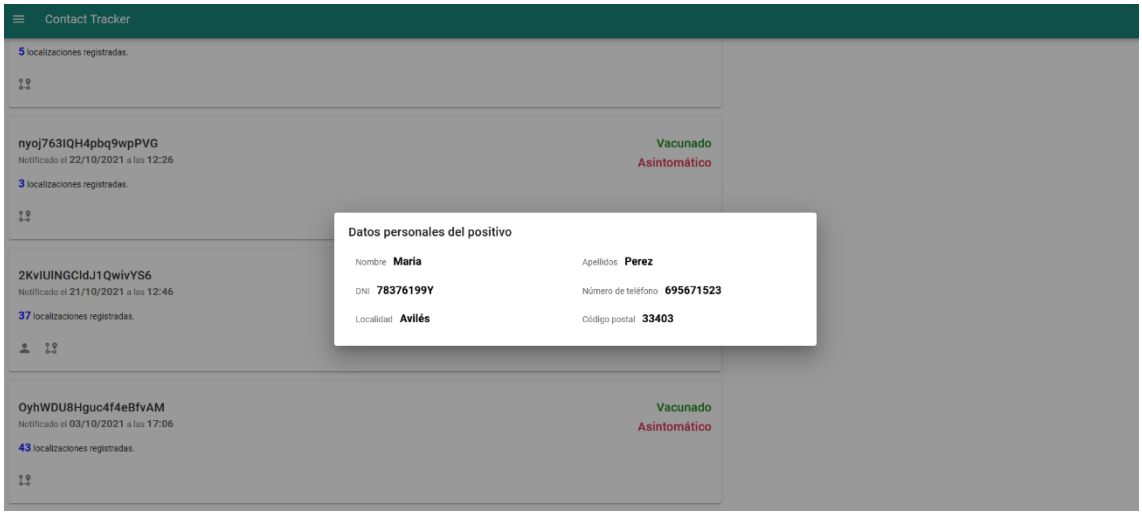


Ilustración 8.14. Diseño final del diálogo para visualizar los datos personales de un positivo.

Además, guiándose por el boceto preliminar del **mapa** para visualizar los **itinerarios** de los positivos, se obtiene el siguiente diseño definitivo.

Ilustración 8.15. Diseño final del diálogo para mostrar el itinerario de un positivo en un mapa.

Como se puede observar, este diseño final es completamente análogo al boceto preliminar realizado en el análisis, y en él se aprecia el mapa que contiene la **ruta** seguida por el positivo en la fecha especificada, la cual se puede seleccionar desde el panel lateral izquierdo.

8.6 Especificación Técnica del Plan de Pruebas

Una vez visto el diseño y arquitectura global del sistema de *Contact Tracker,* en esta sección se describe técnicamente el *Plan de Pruebas* planificado en la sección 7.6 ESPECIFICACIÓN DEL PLAN DE PRUEBAS dentro del capítulo de *Análisis.* Concretamente, se especifican los **componentes** que serán objeto de las distintas pruebas a realizar, qué componentes será necesario *mockear*, es decir, sustituir por un falso objeto que simule su comportamiento o bien solo algunas llamadas a métodos, así como las características técnicas de los dispositivos sobre los cuales se ejecutan las pruebas.

Para cada componente bajo pruebas, se han utilizado **técnicas** como la **partición en clases de equivalencia** o **situaciones** y **condiciones de prueba**, tratando de seguir un **procedimiento** conformado por los siguientes pasos:

1. Identificar las **condiciones de prueba** (condiciones de entrada/salida).
2. Especificar las **clases de equivalencia** de cada condición de prueba.
3. Combinar las clases de equivalencia de las condiciones de prueba para obtener las **situaciones de prueba.**
4. Cubrir las situaciones de prueba mediante **casos de prueba** concretos especificando en la medida de lo posible los datos de prueba utilizados. Para cada caso de prueba se indica las situaciones de prueba que cubre (cuando las haya), el procedimiento a realizar en el caso de prueba, las entradas y la salida esperada.
5. Implementar mediante código cada uno de los casos de prueba obtenidos.

Las condiciones y las situaciones de prueba se etiquetan con **códigos** que los identifican unívocamente, de forma que así se facilita la **trazabilidad** entre los casos de prueba y las situaciones que cubren.

Este procedimiento se ha llevado a cabo para las pruebas unitarias, de integración y del sistema de cada componente o módulo bajo pruebas. Sin embargo, como se verá más adelante, en algunos componentes muy sencillos o que son responsables de tareas muy concretas y simples no se han realizado los pasos 1, 2 y 3 expuestos anteriormente, sino que se han especificado directamente los casos de prueba concretos y se han implementado.

Cabe destacar, que en el paso 4, dedicado a combinar las situaciones de prueba, no se ha utilizado una **estrategia de combinación** específica, sino que en función del componente bajo pruebas se ha centrado la combinación en aquellas condiciones de prueba más interesantes, de forma que las pruebas sean más **exhaustivas** en aquellas situaciones más interesantes o con más flujos de ejecución posibles. En otros componentes más sencillos, se ha combinado utilizando la estrategia de combinación **múltiple** (***multiple choice***) cubriendo todas las casuísticas posibles. Idealmente, los casos de prueba cubren más de una situación de prueba de forma que se simplifique la implementación de los mismos, aunque esto se ha llevado a cabo solo en algunos componentes.

El diseño de las pruebas para cada componente se ha especificado utilizando **hojas de *Excel*** antes de su implementación, y serán expuestos en los siguientes apartados.

8.6.1 Dispositivos de pruebas

Las pruebas se ejecutan sobre un equipo o dispositivo móvil en concreto, en función del tipo de pruebas y de los componentes que se vayan a probar. En este proyecto, se utilizan dos equipos sobre los que se ejecutan las pruebas.

- Dispositivo móvil ***Nexus 5X*** con ***Android Oreo 8.1 API 27.***
- Equipo de sobremesa
    - Procesador *AMD Ryzen 5 3600 6-core 3.95 GHz*
    - 16 *GB RAM*

8.6.2 Pruebas unitarias

Las pruebas unitarias están destinadas a probar el correcto funcionamiento de los componentes de forma atómica e independiente, es decir, probar cada componente por separado. En este caso concreto, las pruebas unitarias se han enfocado principalmente en probar los componentes básicos de la **aplicación móvil** de *Contact Tracker*. Estas pruebas se ejecutarán sobre el equipo de sobremesa especificado en el apartado anterior, ya que los componentes atómicos de la aplicación móvil no requieren de un entorno *Android* para ejecutarse.

Según la **pirámide de *Cohn***, las pruebas unitarias deben conformar la base de todas las pruebas, de forma que la mayoría de las pruebas sean unitarias. En este proyecto, se ha tratado de seguir este enfoque, realizando gran cantidad de pruebas unitarias frente al resto de tipos de pruebas. Las pruebas unitarias se ejecutan cada vez que se realizan cambios en el código y constituyen una pequeña parte para verificar el correcto funcionamiento de las historias de usuario.

8.6.2.1 *Clases de utilidad*

Los componentes más básicos que se prueban de forma unitaria son las **clases de utilidad** que proporcionan servicios básicos como formateo de fechas, cálculos matemáticos u otras operaciones de bajo nivel que no están relacionadas directamente con la **lógica de negocio** especificada en las historias de usuario. Por ello, para este tipo de componentes no se ha seguido el procedimiento especificado anteriormente, sino que se han descrito directamente los casos de prueba con sus entradas y la salida esperada, etiquetando cada caso de prueba con un código único. Se han realizado pruebas unitarias para las siguientes clases de utilidad:

- ***DateUtils:*** contiene utilidades de formateo y manipulación de fechas.
- ***LocationUtils:*** utilidades para el formateo de localizaciones.
- ***NumberUtils****:* utilidades de manipulación de números.

A continuación, se muestran las distintas tablas con los casos de prueba diseñados para cada clase de utilidad.

Tabla 8.10. Casos de prueba para la clase de utilidad DateUtils.

DateUtils			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
DU1	Dar formato a una fecha para convertirla en String.	Fecha y Formato: "dd-MM-yyyy HH-mm"	Fecha formateada a String con el formato indicado.
DU2	Obtener un objeto fecha a partir de las horas y de los minutos.	Horas: 12 Minutos: 29	Objeto fecha "12:29"
DU3	Obtener un objeto fecha a partir de los milisegundos.	Milisegundos	Fecha correspondiente a esos milisegundos.
DU4	Obtener hora y minutos a partir de una fecha con hora de mañana y otra de tarde.	Fechas con hora de mañana y de tarde	Horas y minutos correspondientes a ambas fechas.
DU5	Sumar un día a una fecha y luego restárselo.	Fecha y 1 día	Primero la fecha con un día más, y luego con un día menos.
DU6	Sumar un día a una fecha a finales de mes y luego restárselo.	Fecha y 1 día	Se incrementa un día en la fecha y además cambia al siguiente mes. Lo mismo, pero al contrario cuando se resta 1 día.
DU7	Sumar minutos a una fecha.	Fecha y 45 minutos	Fecha con 45 minutos más.
DU8	Obtener minutos y segundos a partir de milisegundos, pero sin completar un minuto.	54560 ms	54 segundos
DU9	Obtener minutos y segundos a partir de milisegundos completando un minuto.	83489 ms	1 minuto 23 segundos
DU10	Obtener minutos y segundos a partir de milisegundos completando varios minutos.	124402 ms	2 minutos 4 segundos
DU11	Obtener milisegundos a partir de minutos y segundos sin completar un minuto.	0 minutos 34 segundos	34000 ms
DU12	Obtener milisegundos a partir de minutos y segundos completando varios minutos.	2 minutos 80 segundos	200000 ms
DU13	Obtener los segundos de diferencia entre dos fechas iguales.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 11:20"	0 segundos
DU14	Obtener los segundos de diferencia entre dos fechas con la misma hora y minutos.	Fecha 1: "19/09/2021 11:20:00" Fecha 2: "19/09/2021 11:20:34"	34 segundos
DU15	Obtener los segundos de diferencia entre dos fechas distintas.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 11:35"	900 segundos

DU16	Modificar año, mes y día de una fecha.	Fecha: "19/09/2021 11:20" Año:2023 Mes:2 Día:15	Nueva fecha: "15/03/2023 11:20"
DU17	Obtener los días de diferencia entre dos fechas con el mismo día.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 12:23"	0 días
DU18	Obtener los días de diferencia entre dos fechas con distintos días, pero misma hora.	Fecha 1: "19/09/2021 11:20" Fecha 2: "15/09/2021 11:20"	4 días
DU19	Obtener los días de diferencia entre dos fechas con distintos días y distintas horas.	Fecha 1: "19/09/2021 11:20" Fecha 2: "15/09/2021 12:00"	3 días
DU20	Obtener los días de diferencia entre dos fechas de distintos meses.	Fecha 1: "19/09/2021 11:20" Fecha 2: "02/10/2021 12:00"	13 días

Tabla 8.11. Casos de prueba para la clase de utilidad LocationUtils.

LocationUtils			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
LU1	Dar formato a una localización de usuario para transformarla en String.	Localización de usuario	Localización formateada a String con todos sus datos.
LU2	Convertir una localización de usuario a un objeto LatLng.	Localización de usuario	Objeto LatLng con la latitud y longitud de la localización.
LU3	Obtener la distancia entre dos puntos cercanos representados mediante coordenadas (lat y lng) utilizando la distancia de Haversine.	Punto 1, Punto 2	Distancia en metros entre ambos puntos indicados.
LU4	Obtener la distancia entre dos puntos lejanos representados mediante coordenadas (lat y lng) utilizando la distancia de Haversine.	Punto 1, Punto 2	Distancia en metros entre ambos puntos indicados.

Tabla 8.12. Casos de prueba para la clase de utilidad NumberUtils.

NumberUtils			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
NU1	Redondear un número a un decimal.	Número: 2,8776512 Decimales: 1	2,9
NU2	Redondear un número a varios decimales.	Número: 2,8776512 Decimales: 3	2,878
NU3	Redodear un número a ningún decimal.	Número: 2,8776512 Decimales: 0	3

8.6.2.2 *Data Access Objects (DAOs)*

Otros componentes de mayor importancia sobre los que se centran las pruebas unitarias son los objetos ***DAO*****,** encargados de gestionar todas las operaciones que entran y salen de la **base de datos** local del dispositivo móvil. Estos componentes actúan de intermediarios entre los repositorios y la base de datos *SQLite* de la aplicación móvil, por tanto, constituyen una parte importante en el correcto funcionamiento del código de las historias de usuario. Las pruebas unitarias de los *DAOs* se centran en probar las operaciones de **escritura, actualización, borrado** y principalmente las **consultas** que se lanzan a la base de datos local.

En este caso, si es necesario un dispositivo móvil para ejecutar las pruebas, dado que se va a utilizar una base de datos **en memoria**, es decir, no se utilizará una base de datos falsa ni tampoco se hará uso de la base de datos real del dispositivo, sino que para cada caso de prueba se configura una base de datos **nueva** que reside en la memoria del dispositivo bajo pruebas. Esto permite ejecutar los casos de prueba de forma atómica y consistente, con una base de datos siempre en el mismo **estado** para cada caso de prueba.

Al igual que con las clases de utilidad, tampoco se ha realizado un diseño previo de las pruebas de los *DAOs*, sino que se han especificado directamente los casos de prueba que cubren las situaciones más interesantes para detectar posibles errores. Los *DAOs* que se van a probar son los siguientes:

- ***UserLocationDao***: operaciones con las entidades de localizaciones de usuario.
- ***PositiveDao***: operaciones con las entidades de positivos.
- ***LocationAlarmDao***: operaciones con las entidades de alarmas de localización.
- ***RiskContactAlarmDao***: operaciones con las entidades de alarmas de comprobación.
- ***RiskContactDao***: operaciones con las entidades de resultados de comprobaciones y contactos de riesgo.

A continuación, se muestra una tabla para cada *DAO* con sus casos de prueba especificados.

Tabla 8.13. Casos de prueba para el UserLocationDao.

UserLocationDao			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
ULDAO1	Insertar una localización de usuario.	Localización de usuario	Se inserta la localización de usuario correctamente.
ULDAO2	Obtener todas las localizaciones de usuario.		Lista con todas las localizaciones de usuario almacenadas y ordenadas descendentemente por fecha.
ULDAO3	Buscar una localización de usuario por ID.	ID de la localización	Localización de usuario de ID indicado.
ULDAO4	Obtener las localizaciones de usuario registradas en una fecha dada en la que no existen localizaciones.	Fecha: "2021-12-10"	Lista vacía sin localizaciones porque no existen en esa fecha.
ULDAO5	Obtener las localizaciones de usuario registradas en una fecha dada en la que existen varias localizaciones.	Fecha: "2021-09-19"	Lista con las localizaciones filtradas por la fecha indicada y ordenada descendentemente por fecha.
ULDAO6	Eliminar todas las localizaciones de usuario.		Se eliminan todas las localizaciones de la base de datos.
ULDAO7	Eliminar localizaciones por fecha en la que no existen localizaciones.	Fecha: "2021-12-10"	No se elimina ninguna localización porque en esa fecha no existen.
ULDAO8	Eliminar localizaciones por fecha en la que si existen varias localizaciones.	Fecha: "2021-09-19"	Se eliminan 3 localizaciones de usuario que fueron registradas en esa fecha.
ULDAO9	Obtener localizaciones entre dos fechas indicadas donde no existen localizaciones por la derecha.	Inicio: "2021-09-23" Fin: "2021-09-25"	Lista vacía porque no existen localizaciones registradas en ese rango de fechas.
ULDAO10	Obtener localizaciones entre dos fechas indicadas donde no existen localizaciones por la izquierda.	Inicio: "2021-09-11" Fin: "2021-09-14"	Lista vacía porque no existen localizaciones registradas en ese rango de fechas.
ULDAO11	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones por la izquierda del rango.	Inicio: "2021-09-19" Fin: "2021-09-20"	Lista con 3 localizaciones que han sido registradas dentro de ese rango de fechas.
ULDAO12	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en el interior del rango.	Inicio: "2021-09-16" Fin: "2021-09-20"	Lista con 4 localizaciones que han sido registradas dentro de ese rango de fechas.
ULDAO13	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en por la derecha del rango.	Inicio: "2021-09-20" Fin: "2021-09-22"	Lista con 1 localización que ha sido registrada dentro de ese rango de fechas.
ULDAO14	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en varias zonas del rango.	Inicio: "2021-09-14" Fin: "2021-09-17"	Lista con 2 localizaciones que han sido registradas dentro de ese rango de fechas.

Tabla 8.14. Casos de prueba para el PositiveDao.

PositiveDao			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
PDAO1	Insertar un positivo con localizaciones.	Positivo	Se inserta el positivo y se asocia correctamente con sus localizaciones. Se devuelve su ID.
PDAO2	Obtener todos los positivos con sus localizaciones.		Se obtiene una lista con todos los positivos almacenados y sus localizaciones asociadas ordenados por fecha de notificación descendente.
PDAO3	Obtener todos los códigos identificadores de los positivos.		Se obtiene una lista con todos los códigos identificadores de los positivos almacenados.
PDAO4	Obtener el número de positivos notificados en una fecha en la que no haya positivos.	"2021-09-19"	Se obtiene 0 porque no hay positivos notificados en esa fecha.
PDAO5	Obtener el número de positivos notificados en una fecha en la que se hayan notificado varios positivos.	"2021-09-20"	Se obtiene 2 porque hay dos positivos notificados en esa fecha.
PDAO6	Obtener el último positivo que fue notificado de entre varios positivos.		Se obtiene el último positivo de todos los que se notificaron.
PDAO7	Obtener el último positivo que fue notificado cuando no existen positivos.		Se obtiene NULL porque no existen positivos.

Tabla 8.15. Casos de prueba para el LocationAlarmDao.

LocationAlarmDao			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
LADAO1	Insertar una alarma de localización y luego recuperarla.	Alarma de localización	Se inserta la alarma de localización correctamente.
LADAO2	Actualizar los datos de una alarma de localización existente y luego recuperarla.	Alarma de comprobación actualizada	Se actualiza la alarma de localización con los nuevos datos.
LADAO3	Recuperar todas las alarmas de localización.		Lista con las alarmas de localización ordenadas descendentemente por fecha de creación.
LADAO4	Recuperar alarma de localización existente a partir de su ID.	ID de la alarma de localización	Se obtiene la alarma de comprobación correspondiente con sus datos.
LADAO5	Habilitar una alarma de comprobación y luego deshabilitarla.	Nuevo estado e ID de la alarma	Se actualiza el estado de la alarma de localización de ID indicado.
LADAO6	Eliminar todas las alarmas de localización.		Se eliminan todas las alarmas de localización y se devuelve el número de eliminaciones.
LADAO7	Eliminar alarma por ID.	ID de la alarma de localización	Se elimina únicamente la alarma de localización de ID indicado.
LADAO8	Obtener colisiones con alarmas que coinciden exactamente.	Inicio: "12:00" Fin: "13:00"	Lista con las alarmas que coinciden exactamente con esas horas de inicio y fin.

LADAO9	Obtener colisiones con alarmas que coinciden por el interior.	Inicio: “12:20” Fin: “12:30”	Lista con las alarmas que coinciden por el interior con esas horas de inicio y fin.
LADAO10	Obtener colisiones con alarmas que coinciden por el exterior.	Inicio: “11:30” Fin: “13:30”	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.
LADAO11	Obtener colisiones con alarmas que coinciden por la izquierda.	Inicio: “11:30” Fin: “12:30”	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.
LADAO12	Obtener colisiones con alarmas que coinciden por la derecha.	Inicio: “12:30” Fin: “13:30”	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.
LADAO13	Obtener colisiones con múltiples alarmas, añadiendo una alarma auxiliar que genera otra colisión.	Inicio: “12:30” Fin: “13:30”	Lista con las dos alarmas que colisionan.
LADAO14	Obtener colisiones con alarmas que no coinciden con las horas indicadas.	Inicio: “11:00” Fin: “11:30”	Lista vacía porque no hay colisiones.

Tabla 8.16. Casos de prueba para el RiskContactAlarmDao.

RiskContactAlarmDao			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
RCADAO1	Insertar una alarma de comprobación y luego recuperarla.	Alarma de comprobación	Se inserta la alarma de comprobación correctamente.
RCADAO2	Actualizar varias alarmas de comprobación con nuevos valores.	Alarmas de comprobación actualizadas	Se actualizan correctamente las alarmas de comprobación indicadas con los nuevos valores.
RCADAO3	Eliminar una alarma de comprobación a partir de su ID.	ID de la alarma de comprobación	Se elimina únicamente la alarma de comprobación de ID indicado.
RCADAO4	Obtener todas las alarmas de comprobación existentes.		Listado con todas las alarmas de comprobación almacenadas en el dispositivo.
RCADAO5	Obtener todas aquellas alarmas establecidas para una hora indicada en la que no hay ninguna alarma.	Hora: “12:00”	Lista vacía porque no hay ninguna alarma establecida para esa hora.
RCADAO6	Obtener todas aquellas alarmas establecidas para una hora indicada en la que existen varias alarmas.	Hora: “13:00”	Lista con las alarmas que han sido establecidas en la hora indicada.

Tabla 8.17. Casos de prueba para el RiskContactDao.

RiskContactDao			
Código	**Procedimiento**	**Entradas**	**Resultado esperado**
RCDAO1	Insertar un resultado de una comprobación con sus contactos de riesgo.	Resultado de comprobación	Se inserta el resultado correctamente junto con sus contactos de riesgo.
RCDAO2	Buscar un resultado de una comprobación a partir de su ID.	ID del resultado de la comprobación	Resultado de comprobación de ID pasado como parámetro.
RCDAO3	Obtener todos los resultados de comprobaciones con sus contactos de riesgo.		Listado con todos los resultados de comprobación y sus respectivos contactos de riesgo.

8.6.2.3 *Algoritmo de comprobación de contactos de riesgo*

Siguiendo con las pruebas unitarias, una de las funcionalidades más importantes que conforman el núcleo del sistema de *Contact Tracker* es la **comprobación de contactos de riesgo**, la cual hace uso del **algoritmo de comprobación** diseñado para comparar itinerarios de usuarios y positivos y detectar posibles puntos de contacto. Por ello, para este componente se han diseñado gran cantidad de pruebas unitarias exhaustivas que prueben la mayoría de las casuísticas posibles.

En este caso, las pruebas unitarias se centran en probar la lógica de negocio no solo del algoritmo de comprobación, sino también del modelo de dominio que envuelve a dicho algoritmo, es decir, las entidades que modelan los itinerarios, contactos de riesgo u otros elementos del mundo real. De este modo, no es necesario un dispositivo móvil para ejecutar las pruebas, es suficiente con ejecutarlas sobre el equipo descrito anteriormente.

Dada la **importancia** y **complejidad** del módulo de comprobación de contactos de riesgo es interesante seguir el procedimiento descrito anteriormente identificando condiciones de prueba del algoritmo de comprobación y descomponiéndolas en clases de equivalencia para luego combinarlas y así obtener las situaciones de prueba. Además, cabe destacar que, como consecuencia de esta complejidad, la gran mayoría de casos de prueba son **lógicos**, es decir, no se especifican **valores concretos** en las entradas y salidas esperadas, sino que estas se describen en lenguaje natural.

En las siguientes secciones se describe cada uno de los componentes involucrados en la comprobación de contactos de riesgo incluyendo el **diseño previo** de las pruebas unitarias, con las condiciones de prueba y clases de equivalencia identificadas, incluyendo las situaciones de prueba obtenidas tras la combinación, y las tablas con los correspondientes casos de prueba que cubren las situaciones de prueba.

8.6.2.3.1 Clase *Itinerary*

Esta clase representa un itinerario de un usuario, formado por ubicaciones agrupadas en los días en los que fueron registradas. Los objetos de esta clase son las entradas del detector de contactos de riesgo. A continuación, se muestra el diseño de pruebas para esta clase.

Condiciones de Prueba
Número de días en los que se divide el itinerario.
1 día
2 días
Más de 2 días

Tabla 8.18. Situaciones de prueba para la clase Itinerary.

Situaciones de Prueba	Código
Itinerario dividido en un día.	I1
Itinerario dividido en dos días.	I2
Itinerario dividido en más de dos días.	I3

La siguiente tabla muestra los casos de prueba diseñados para cubrir las situaciones de prueba de la tabla superior para esta clase, donde ***SP*** representa las situaciones de prueba cubiertas por el caso de prueba.

Tabla 8.19. Casos de prueba para la clase Itinerary.

Clase *Itinery*				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
I1	I1	Crear un itinerario con localizaciones divididas en un solo día.	Lista de localizaciones de un solo día.	Localizaciones agrupadas en un solo día.
I2	I2	Crear un itinerario con localizaciones divididas en dos días.	Lista de localizaciones de dos días.	Localizaciones agrupadas en dos días.
I3	I3	Crear un itinerario con localizaciones divididas en más de dos días.	Lista de localizaciones de más de dos días.	Localizaciones agrupadas en más de dos días.

8.6.2.3.2 Clase *RiskContact*

Representa un contacto de riesgo dentro del algoritmo de comprobación, formado por varios pares de localizaciones que están próximas en el espacio y en el tiempo. Las pruebas unitarias de esta clase se centran en verificar el funcionamiento de los **cálculos de los indicadores** de riesgo, tiempo de exposición, proximidad media, fechas de inicio y fin… entre otros valores. Las siguientes tablas reflejan el diseño de las pruebas unitarias para esta clase.

Condiciones de Prueba
Puntos de contacto
1 punto de contacto
2 puntos de contacto
Más de 2 puntos de contacto
Intersección entre fechas
Intersección exacta
Intersección interna
Intersección externa

	Interseccion derecha
	Interseccion izquierda
	No hay intersección
Nivel de riesgo	
	Amarillo
	Naranja
	Rojo

Tabla 8.20. Situaciones de prueba para la clase RiskContact.

Situaciones de Prueba	Código
1 punto de contacto	
Nivel de riesgo amarillo	RC1
Nivel de riesgo naranja	RC2
Nivel de riesgo rojo	RC3
2 puntos de contacto	
Intersección exacta	RC4
Intersección interna	RC5
Intersección externa	RC6
Intersección derecha	RC7
Intersección izquierda	RC8
No hay intersección	RC9
Más de 2 puntos de contacto	
Intersección interna	RC10
Intersección externa	RC11
Intersección derecha	RC12
No hay intersección	RC13

A continuación, se muestra la tabla con los **casos de prueba** diseñados para cubrir las anteriores situaciones de prueba obtenidas para la clase *RiskContact*.

Tabla 8.21. Casos de prueba para la clase RiskContact.

Clase *RiskContact*				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
RC1	RC1	Añadir un punto de contacto con dos localizaciones que generen un riesgo amarillo.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo amarillo.
RC2	RC2	Añadir un punto de contacto con dos localizaciones que generen un riesgo naranja.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo naranja.
RC3	RC3	Añadir un punto de contacto con dos localizaciones que generen un riesgo rojo.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo rojo.

RC4	RC4	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección exacta entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC5	RC5	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección interna entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC6	RC6	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección externa entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC7	RC7	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección derecha entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC8	RC8	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección izquierda entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC9	RC9	Añadir dos puntos de contacto con 4 localizaciones cuyas fechas no generen ninguna intersección.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.
RC10	RC10	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección interna.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.
RC11	RC11	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección externa.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.
RC12	RC12	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección derecha.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.
RC13	RC13	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas no generen intersección.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.

8.6.2.3.3 Clase *RiskContactResult*

Representa un resultado de una comprobación de contactos, incluyendo todos los contactos de riesgo detectados y estadísticas generales del resultado, como el número de positivos y las medias de riesgo, tiempo de exposición y proximidad. Las pruebas unitarias se centran en verificar el funcionamiento del cálculo de las **estadísticas generales**, es decir, las medias de tiempo de exposición, proximidad y riesgo. Las siguientes tablas reflejan el diseño de las pruebas unitarias para esta clase.

Condiciones de Prueba
Número de contactos de riesgo
1 contacto de riesgo
Varios contactos de riesgo

Tabla 8.22. Situaciones de prueba para la clase RiskContactResult.

Situaciones de Prueba	Código
Resultado con un solo contacto de riesgo.	RCR1
Resultado con varios contactos de riesgo.	RCR2

A continuación, se muestran los **casos de prueba** diseñados para cubrir las situaciones de prueba de la clase *RiskContactResult*.

Tabla 8.23. Casos de prueba para la clase RiskContactResult.

Clase *RiskContactResult*				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
RCR1	RCR1	Calcular las medias de tiempo de exposición, proximidad y riesgo para un resultado con un solo contacto de riesgo. Luego obtener el contacto de mayor riesgo y ordenarlos por riesgo.	Resultado con un contacto de riesgo.	Medias de tiempo de exposición, proximidad y riesgo calculadas correctamente. Contacto de mayor riesgo y listado ordenado por riesgo.
RCR2	RCR2	Calcular las medias de tiempo de exposición, proximidad y riesgo para un resultado con 3 contactos de riesgo. Luego obtener el contacto de mayor riesgo y ordenarlos por riesgo.	Resultado con 3 contactos de riesgo.	Medias de tiempo de exposición, proximidad y riesgo calculadas correctamente. Contacto de mayor riesgo y listado ordenado por riesgo.

8.6.2.3.4 Detector de contactos de riesgo

La propia implementación del algoritmo de comprobación se encuentra bajo la clase del detector de contactos que contiene los métodos requeridos para ejecutar el algoritmo y así detectar los contactos de riesgo a partir de **dos itinerarios** de entrada.

Para facilitar la implementación de las pruebas, se han diseñado previamente diversos **itinerarios** especificando sus coordenadas concretas en un mapa y almacenándolos en **ficheros de texto,** incluyendo para cada ubicación su *timestamp* con fecha y hora en la que se registró la localización. De este modo, se procesan los ficheros de texto por código y se cargan los itinerarios en memoria, agilizando así el proceso de pruebas, pues no es necesario crear los itinerarios por código, además de que los ficheros con los itinerarios pueden ser **reutilizados** en otras pruebas.

Como se verá a continuación, existe un número considerable de condiciones de prueba y clases de equivalencia, por lo que será necesario centrar la **combinación** en aquellas clases de equivalencia más interesantes de probar, mientras que el resto se combinen solo parcialmente. De lo contrario, si se pretende utilizar ***multiple choice***, se terminaría con una

jerarquía de situaciones de prueba inmanejable que dificultaría la implementación de las pruebas sin aportar grandes beneficios, ya que no es necesario probar todas las combinaciones posibles para verificar el correcto funcionamiento del algoritmo, además de que se obtendrían situaciones de prueba redundantes.

Las siguientes tablas reflejan el diseño de pruebas unitarias para el detector de contactos de riesgo, incluyendo las condiciones de prueba, sus clases de equivalencia y las situaciones de prueba obtenidas.

Condiciones de Prueba
Número de días del itinerario del usuario
1 día
Varios días
Número de días de itinerario del positivo
1 día
Varios días
Coincidencia entre itinerarios
Itinerarios coinciden en un día
Itinerarios coinciden en varios días
Itinerarios no coinciden en ningún día
Coincidencia entre localizaciones
Coincidencia en el tiempo
Coincidencia en el espacio
Coincidencia en el espacio y el tiempo (contacto de riesgo)
Cierre de un tramo de contacto
No hay más localizaciones del positivo que comprobar
No hay más localizaciones del usuario que comprobar
No hay coincidencia en el tiempo y en el espacio en las siguientes localizaciones.
Número de contactos de riesgo
1 contacto de riesgo
Varios contactos de riesgo
Número de puntos en un contacto de riesgo
1 solo punto de contacto
Varios puntos de contacto

Tabla 8.24. Situaciones de prueba para el detector de contactos de riesgo.

Situaciones de Prueba	**Código**
Coincidencia en el tiempo.	
Localizaciones coinciden en el tiempo.	D1
Localizaciones no coinciden en el tiempo.	D2
Coincidencia en el espacio.	
Localizaciones coinciden en el espacio.	D3
Localizaciones no coinciden en el espacio.	D4
Buscar el punto más cercano a una localización dada entre una lista de varios puntos.	D5
Itinerario del usuario con un solo día.	
Itinerario del positivo con un solo día que no coincide con el del usuario.	
Coincidencia en el tiempo y en el espacio generando un contacto de riesgo.	D6

No existe coincidencia en el tiempo y en el espacio en ningún punto.	D7
Itinerario del positivo con un solo día que coincide con el del usuario.	
Coincidencia solo en el tiempo entre dos localizaciones.	D8
Coincidencia solo en el espacio entre dos localizaciones.	D9
Coincidencia en el tiempo y en el espacio.	
Un solo contacto de riesgo.	
Un solo punto de contacto.	D10
Varios puntos de contacto	
Cierre del tramo de contacto porque no hay más localizaciones del positivo que comprobar.	D11
Cierre del tramo de contacto porque no hay más localizaciones del usuario que comprobar.	D12
Cierre del tramo de contacto porque no hay coincidencia en el tiempo y en el espacio en las siguientes localizaciones.	D13
Varios contactos de riesgo.	
Uno con un punto de contacto y otro con varios puntos de contacto que se cierra porque no hay coincidencia en el espacio y tiempo.	D14
Uno con varios puntos que se cierra porque no hay coincidencia y otro con varios puntos de contacto que se cierra porque no hay más localizaciones de usuario.	D15
Dos con varios puntos de contacto. Uno de ellos se cierra porque no hay más localizaciones de positivo y el otro porque no hay coincidencia en tiempo y espacio.	D16
Coincidencia en el tiempo y en el espacio entre dos localizaciones además de coincidencia solo en el tiempo y coincidencia solo en el espacio entre otras localizaciones.	D17
Itinerario del positivo con varios días donde solo coincide uno con el del usuario y existe un contacto de riesgo.	D18
Itinerario del usuario con varios días.	
Itinerario del positivo con un solo día que coincide con uno de los días del itinerario del usuario y existe un contacto de riesgo.	D19
Itinerario del positivo con varios días que coinciden con varios días del itinerario del usuario y existe un contacto de riesgo por día.	D20
Itinerario del positivo con varios días donde ninguno coincide con los días del usuario.	D21

A continuación, se exponen los **casos de prueba** diseñados para el detector de contactos y que cubren las situaciones de prueba presentadas en la tabla anterior.

Tabla 8.25. Casos de prueba para el detector de contactos de riesgo.

Detector de Contactos de Riesgo				
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**
D1, D2	D12	Introducir dos localizaciones que no coinciden en el tiempo y otras dos que sí coinciden en el tiempo según el margen de diferencia temporal introducido. Introducir otras dos localizaciones que coinciden en el tiempo en el valor límite del margen de diferencia temporal. Comprobar otro par de localizaciones que coinciden en la hora, pero no en el día.	* Margen temporal: 5 segundos. * Pares de localizaciones.	Se devuelve true cuando ambas coinciden en el tiempo y false cuando no coinciden. En el valor límite se devuelve true.
D3, D4	D34	Introducir dos localizaciones que no coinciden en el espacio y otras dos que sí coinciden en el espacio según el margen de distancia de seguridad introducido. Introducir otras dos localizaciones que coinciden en el espacio en el valor límite del margen de distancia de seguridad.	* Margen de distancia de seguridad: 5 metros. * Pares de localizaciones.	Se devuelve true cuando ambas coinciden en el espacio y false cuando no coinciden. En el valor límite se devuelve true.
D5	D5	Dada una lista de varias localizaciones, buscar la más cercana en el espacio a una localización dada. Buscar la localización más cercana a la localización objetivo con una lista vacía de localizaciones.	* Lista de localizaciones. * Localización con la que comparar la distancia.	Se devuelve la localización con menor distancia hasta la localización objetivo.

D6	D6	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con un solo día que no coincide con el día del usuario, pero existe un posible contacto de riesgo porque hay coincidencias en el tiempo y en el espacio.	* Itinerario 1 * Itinerario 2	Lista vacía de contactos de riesgo porque son itinerarios de diferentes días.
D7	D7	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con un solo día que no coincide con el día del usuario, y tampoco existen coincidencias en el tiempo y en el espacio en ninguno de los puntos.	* Itinerario 1 * Itinerario 3	Lista vacía de contactos de riesgo porque son itinerarios de diferentes días.
D8	D8	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, y donde existen varios puntos que coinciden solo en el tiempo.	* Itinerario 1 * Itinerario 4	Lista vacía de contactos de riesgo porque solo hay coincidencia en el tiempo.
D9	D9	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, y donde existen varios puntos que coinciden solo en el espacio.	* Itinerario 1 * Itinerario 5	Lista vacía de contactos de riesgo porque solo hay coincidencia en el espacio.
D10	D10	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con un solo punto de contacto entre dos localizaciones que coinciden en el tiempo y en el espacio.	* Itinerario 1 * Itinerario 6	Lista con un contacto de riesgo formado por un solo punto de contacto.
D11	D11	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no existen más localizaciones del positivo.	* Itinerario 7 * Itinerario 8	Lista con un contacto de riesgo formado por varios puntos de contacto.
D12	D12	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no existen más localizaciones del propio usuario.	* Itinerario 7 * Itinerario 9	Lista con un contacto de riesgo formado por varios puntos de contacto.
D13	D13	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no hay coincidencia en el tiempo y espacio en el siguiente par de localizaciones.	* Itinerario 7 * Itinerario 10	Lista con un contacto de riesgo formado por varios puntos de contacto.
D14	D14	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por un solo punto de contacto y el otro formado por varios puntos de contacto que se cierra porque ya no hay coincidencia en el tiempo y en el espacio.	* Itinerario 7 * Itinerario 11	Lista con dos contactos de riesgo formados por un punto y por varios respectivamente.

D15	D15	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por varios puntos de contacto que se cierra porque ya no hay coincidencia en el espacio y tiempo, y el otro formado por varios puntos de contacto que se cierra porque ya no hay más localizaciones de usuario.	* Itinerario 7 * Itinerario 12	Lista con dos contactos de riesgo formados por varios puntos de contacto.
D16	D16	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por varios puntos de contacto que se cierra porque ya no hay coincidencia en el espacio y tiempo, y el otro formado por varios puntos de contacto que se cierra porque ya no hay más localizaciones de positivo.	* Itinerario 7 * Itinerario 13	Lista con dos contactos de riesgo formados por varios puntos de contacto.
D17	D17	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con un contacto de riesgo, un par de localizaciones que coinciden en el tiempo y otro par que coincide en el espacio, pero no generan contactos de riesgo.	* Itinerario 4 * Itinerario 14	Lista con un solo contacto de riesgo porque el resto no coinciden en el tiempo y espacio.
D18	D18	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con varios días donde uno coincide con el del usuario. En ese día existe un contacto de riesgo con varios puntos que se cierra debido a que no existen más coincidencias en el tiempo y espacio.	* Itinerario 5 * Itinerario 15	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.
D19	D19	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con varios días y otro itinerario del positivo con un solo día que coincide con uno de los del usuario. En ese día existe un contacto de riesgo con varios puntos que se cierra debido a que no existen más coincidencias en el tiempo y espacio.	* Itinerario 15 * Itinerario 5	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.
D20	D20	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario y otro itinerario del positivo ambos con varios días que coinciden. En algunos de los días que coinciden existen contactos de riesgo y en el resto de los días que coinciden no hay contactos de riesgo.	* Itinerario 16 * Itinerario 17	Lista con dos contactos de riesgo, de días distintos.
D21	D21	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario y otro itinerario del positivo ambos con varios días donde ninguno de ellos coincide y existen varios puntos de contacto que darían lugar a contactos de riesgo si coincidieran los días.	* Itinerario 16 * Itinerario 18	Lista vacía sin ningún contacto de riesgo.

8.6.2.3.5 Clase *LocationAlarm*

Esta clase representa una alarma para llevar a cabo el rastreo de ubicación de forma automática, estableciendo una hora de inicio y de fin para ejecutar el servicio de rastreo. Las pruebas unitarias de esta clase están enfocadas a probar la lógica de **validez** de alarmas y de su **estado**, es decir, verificar que, si una alarma está desfasada, esta se actualiza correctamente.

A continuación, se muestra el diseño de pruebas para esta clase, incluyendo las condiciones de prueba y situaciones obtenidas.

Condiciones de Prueba
Validez de la alarma
Fecha de inicio anterior a la de fin (válida).
Fecha de inicio igual que la de fin (inválida).
Fecha de inicio posterior a la de fin (inválida).
Estado de la alarma
Alarma actualizada (posterior a la fecha de prueba).
Alarma desfasada (igual a la fecha de prueba).
Alarma desfasada (anterior a la fecha de prueba).

Tabla 8.26. Situaciones de prueba para la clase LocationAlarm.

Situaciones de Prueba	Código
Fecha de inicio igual que la de fin (alarma inválida).	LA1
Fecha de inicio posterior a la de fin (alarma inválida).	LA2
Fecha de inicio anterior a la de fin (alarma válida)	
Alarma actualizada (posterior a la fecha de prueba).	LA3
Alarma desfasada (igual a la fecha de prueba).	LA4
Alarma desfasada (anterior a la fecha de prueba).	LA5

Como se puede observar, se utilizan **valores límite** en las fechas para comprobar el funcionamiento en aquellas situaciones que se dan en los extremos, como, por ejemplo, el estado de una alarma desfasada porque su fecha de inicio es igual a la fecha actual de prueba.

A continuación, se muestra una tabla con los **casos de prueba** que cubren las anteriores situaciones de prueba.

Tabla 8.27. Casos de prueba para la clase LocationAlarm.

Clase *LocationAlarm*				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
LA1, LA2	LA12	Crear una alarma inválida con fecha de inicio igual a la de fin, y luego crear otra alarma inválida con fecha de inicio posterior a la de fin.	* Fecha de prueba * Alarmas de localización	Se detecta que las dos alarmas son inválidas.
LA3	LA3	Crear una alarma válida que esté actualizada, con la fecha posterior a la fecha de prueba.	* Fecha de prueba * Alarma actualizada	No se modifica la alarma porque ya está actualizada.
LA4, LA5	LA45	Crear una alarma válida que esté desfasada, con la fecha igual a la fecha de prueba. Crear otra alarma desfasada con la fecha anterior a la fecha de prueba.	* Fecha de prueba * Alarmas desfasadas	Se modifican las alarmas posponiéndolas un día más que la fecha de prueba.

8.6.2.4 *Managers*

Los últimos componentes que son sometidos a las pruebas unitarias son los ***managers*** de la aplicación móvil. Son los componentes de más alto nivel dentro de las pruebas unitarias, ya que hacen uso de los repositorios. Los repositorios no se prueban ya que su lógica de negocio

se resume en llamadas a los *DAOs*, por lo tanto, no tiene sentido hacer pruebas unitarias de los repositorios *mockeando* la base de datos. Sin embargo, los *managers* disponen de lógica de negocio que es importante probar de forma unitaria.

Para ello, se ***mockean*** los **repositorios** utilizados desde los *managers*, es decir, las llamadas a los métodos de los repositorios se sustituyen por ***stubs***, de forma que cuando se invoque a un método concreto de un repositorio con ciertos parámetros se devolverán datos falsos incrustados directamente en las pruebas, a conveniencia de las situaciones de prueba. De este modo, se prueba solo la lógica de negocio de los *managers* sin depender de los repositorios.

Las pruebas unitarias se diseñan para el *manager* de alarmas de localización, el *manager* de alarmas de comprobación y el *manager* de positivos. El *manager* encargado de gestionar las **comprobaciones de contactos de riesgo** no se probará de manera unitaria debido a que la lógica de negocio está ligada a los repositorios y a las llamadas de red a la *API REST*, por lo que tiene más sentido realizar pruebas de **integración** del *manager* de contactos de riesgo junto con el *viewmodel* y los repositorios correspondientes, como se verá en el apartado 8.6.3 PRUEBAS DE INTEGRACIÓN.

Como no es necesario ninguna dependencia del entorno de *Android* las pruebas de los *managers* se ejecutan directamente sobre el equipo de sobremesa sin necesidad de utilizar un emulador móvil.

8.6.2.4.1 *Manager* de alarmas de localización

Se encarga de gestionar las operaciones con alarmas de rastreo de la localización del usuario. Las pruebas unitarias de este *manager* se centran en probar la lógica de la validez de alarmas y la detección de colisiones entre alarmas. Para ello, se *mockean* las siguientes **dependencias:**

- *Manager* de alarmas de *Android*
- Repositorio de localizaciones de usuario.

A continuación, se muestra el **diseño** de pruebas unitarias para este *manager*, incluyendo las condiciones de prueba y sus situaciones derivadas.

Condiciones de Prueba
Validez de la alarma de localización
Alarma válida (inicio < fin)
Alarma inválida (inicio == fin)
Alarma inválida (inicio > fin)
Colisiones
No hay colisiones
Colisión interna
Colisión externa
Colisión izquierda
Colisión derecha

Tabla 8.28. Situaciones de prueba para el manager de alarmas de localización.

Situaciones de Prueba	Código
Alarma inválida (inicio == fin)	LAM1
Alarma inválida (inicio > fin)	LAM2
Alarma válida (inicio < fin)	
Alarma válida sin colisiones	LAM3
Alarma válida con colisión interna	LAM4
Alarma válida con colisión externa	LAM5
Alarma válida con colisión izquierda	LAM6
Alarma válida con colisión derecha	LAM7

A continuación, se muestran los **casos de prueba** que cubren las situaciones de prueba expuestas en la tabla superior para el *manager* de alarmas de localización.

Tabla 8.29. Casos de prueba para el manager de alarmas de localización.

***Manager* de alarmas de localización**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
LAM1, LAM2	LAM12	Establecer una alarma inválida con inicio igual a fin y luego otra alarma inválida con inicio posterior a fin.	* Alarma1 * Alarma2	Se devuelve un código de Error de alarma inválida en ambos casos.
LAM3	LAM3	Establecer una alarma válida y sin colisiones.	* Alarma3	Se devuelve un código de Éxito.
LAM4	LAM4	Establecer una alarma válida que genera una colisión interna.	* Alarma3 * Alarma4(colisión)	Se devuelve un código de Éxito y se actualiza la alarma.
LAM5	LAM5	Establecer una alarma válida que genera una colisión externa.	* Alarma4 * Alarma3 (colisión)	Se devuelve un código de Error de colisión de alarmas.
LAM6	LAM6	Establecer una alarma válida que genera una colisión izquierda.	* Alarma3 * Alarma5 (colisión)	Se devuelve un código de Error de colisión de alarmas.
LAM7	LAM7	Establecer una alarma válida que genera una colisión derecha.	* Alarma3 * Alarma6 (colisión)	Se devuelve un código de Error de colisión de alarmas.

Como se puede observar en la columna de datos de **entrada,** se especifican distintas alarmas de prueba numeradas. Estas alarmas concretas que se utilizan en las pruebas serán expuestas en el CAPÍTULO 10 DESARROLLO DE LAS PRUEBAS.

8.6.2.4.2 *Manager* de alarmas de comprobación

Este *manager* tiene responsabilidades similares al anterior, pues se encarga de gestionar las operaciones con **alarmas** de **comprobación de contactos**, de forma que se establecen a una hora determinada del día para ejecutar automáticamente esta comprobación. Las pruebas unitarias de este componente se centran en la lógica de detección de colisiones y del límite de alarmas permitido. En este caso, se *mockean* las siguientes **dependencias**:

- *Manager* de alarmas de *Android*.
- Repositorio de contactos de riesgo.

A continuación, se muestra el **diseño** de pruebas con las condiciones de prueba y sus situaciones de prueba derivadas.

Condiciones de Prueba
Colisiones
No hay colisiones
Existen colisiones
Límite de alarmas
No se supera el límite de alarmas
No se supera el límite de alarmas igualándolo
Se supera el límite de alarmas sobrepasándolo

Tabla 8.30. Situaciones de prueba para el manager de alarmas de comprobación.

Situaciones de Prueba	Código
No hay colisiones	
No se supera el límite de alarmas	RCAM1
No se supera el límite de alarmas igualándolo	RCAM2
Se supera el límite de alarmas sobrepasándolo	RCAM3
Existen colisiones	RCAM4

A continuación, se muestra una tabla con los **casos de prueba** que cubren las situaciones de prueba descritas en la tabla superior.

Tabla 8.31. Casos de prueba para el manager de alarmas de comprobación.

***Manager* de alarmas de comprobación**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
RCAM1	RCAM1	Establecer una alarma sin que se supere el límite y sin que haya colisiones.	* Límite de alarmas * Nº de alarmas existentes: 1 * Alarma1	Se devuelve un código de Éxito.
RCAM2	RCAM2	Establecer una alarma sin que se genere colisión y sin superar el límite igualándolo.	* Límite de alarmas * Nº de alarmas existentes: 2 * Alarma1	Se devuelve un código de Éxito.
RCAM3	RCAM3	Establecer una alarma sin que se genere colisión y superando el límite sobrepasándolo.	* Límite de alarmas * Nº de alarmas existentes: 3 * Alarma1	Se devuelve un código de Error de límite de alarmas sobrepasado.
RCAM4	RCAM4	Establecer una alarma generando colisiones.	* Límite de alarmas * Alarma que genera colisión. * Alarma11	Se devuelve un código de Error de colisión de alarmas.

8.6.2.4.3 Manager de positivos

Este *manager* es el responsable de gestionar la **notificación de positivos** en la aplicación *móvil*, de forma que se realicen las comprobaciones previas a la subida de las localizaciones a la nube, así como la recuperación de las localizaciones almacenadas en local en función del periodo de infectividad. Las pruebas unitarias de este componente se centran en probar la lógica de las comprobaciones previas a la notificación y la recuperación de las localizaciones a subir. Para este *manager* es necesario *mockear* las siguientes **dependencias:**

- Repositorio de positivos.
- Repositorio de localizaciones de usuario.
- Repositorio de configuración.

A continuación, se muestra el **diseño** de las pruebas unitarias para este *manager*, incluyendo las condiciones de prueba y sus situaciones derivadas.

Condiciones de Prueba
Límite de notificación
No se supera el límite de notificación porque no hay positivos notificados.
No se supera el límite de notificación porque ya han transcurrido suficientes días.
Se supera el límite de notificación
Existencia de localizaciones
Existen localizaciones para notificar
No existen localizaciones para notificar
Resultado de la notificación
Éxito
Error de red
Error HTTP

Tabla 8.32. Situaciones de prueba para el manager de positivos.

Situaciones de Prueba	Código
Se supera el límite de notificación	PM1
No se supera el límite de notificación porque no hay positivos notificados.	PM2
No se supera el límite de notificación porque ya han transcurrido suficientes días.	
No existen localizaciones para notificar	PM3
Existen localizaciones para notificar	
Resultado de éxito	PM4
Resultado con error de red	PM5
Resultado con error HTTP	PM6

Tras el diseño de las pruebas, se muestran en la siguiente tabla los **casos de prueba** que cubren las situaciones de prueba anteriores.

Tabla 8.33. Casos de prueba para el manager de positivos.

***Manager* de positivos**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
PM1	PM1	Notificar un positivo superando el límite de notificación, es decir, aún no han transcurrido los suficientes días para permitir notificar de nuevo un positivo. Probar el valor límite de días transcurridos con la fecha del último positivo notificado.	* Configuración de la notificación. * Último positivo notificado en la fecha actual.	Se obtiene un código de Error indicando que se ha superado el límite de notificación de positivos.
PM2	PM2	Notificar un positivo sin superar el límite de notificación porque no existen otros positivos notificados.	* Configuración de la notificación. * Localizaciones a subir.	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones subidas.
PM3	PM3	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación, pero no existen localizaciones que notificar.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores.	Se obtiene un código de Error indicando la inexistencia de localizaciones que notificar.
PM4	PM4	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado exitoso.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones subidas.
PM5	PM5	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado de error de red.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Error de tiempo de espera agotado.
PM6	PM6	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado de error HTTP.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Error genérico de que no es posible notificar el positivo.

8.6.3 Pruebas de Integración

Las pruebas de integración se sitúan en un nivel superior por encima de las pruebas unitarias, dentro de la pirámide de *Cohn*. Por ello, la cantidad de pruebas de integración es más reducida que la de pruebas unitarias. Una vez se han diseñado las pruebas de los componentes **aislados,** el objetivo principal de las pruebas de integración es verificar el correcto funcionamiento de **grupos de componentes** que interaccionan entre sí, es decir, se trata de probar que varios componentes funcionan bien cuando trabajan juntos.

En este apartado se presenta la **especificación técnica** para las pruebas de **integración** del sistema de *Contact Tracker*, que en este caso están enfocadas a probar dos aspectos:

1. Funcionamiento de los *ViewModels* dentro de la aplicación móvil.
2. *Endpoints* de la *API REST* dentro del servidor *web*.

Tanto para los *viewmodels* como para la *API REST* se expone en primer lugar el **diseño de pruebas** que incluye las **condiciones de prueba**, **clases de equivalencia** y **situaciones de prueba** derivadas de combinar las clases, y, por otro lado, se presentan los **casos de prueba** diseñados para cubrir las situaciones de prueba.

Dado que se trata de pruebas de integración, los ***mocks*** solo se utilizarán en aquellos casos que sea necesario, garantizando que no afecten a la lógica de los componentes integrados sometidos a las pruebas.

8.6.3.1 *ViewModels*

Las pruebas de integración para los *viewmodels* de la aplicación móvil están enfocadas en verificar el funcionamiento adecuado entre los ***viewmodels***, los ***managers***, los **repositorios** y los ***DAOs***. Estas pruebas se diseñan para comprobar que las interacciones entre estos componentes es la esperada.

En este caso, sería necesario un dispositivo móvil con *Android* para poder ejecutar las pruebas, dado que los *viewmodels* son abstracciones que forman parte del *framework* de *Android*. Sin embargo, esto no es necesario gracias a una herramienta que permite *mockear* el *framework* de *Android* de tal manera que las pruebas se puedan ejecutar directamente en una máquina virtual de *Java*, haciendo que la ejecución de las pruebas sea mucho más rápida. Esta herramienta se explicará en el CAPÍTULO 10 DESARROLLO DE LAS PRUEBAS.

Las pruebas de integración se realizan únicamente sobre los *viewmodels* más importantes y que disponen de lógica de negocio significativa, este es el caso del *viewmodel* de comprobación de contactos de riesgo, el de resultados de las comprobaciones y el *viewmodel* de notificación de positivos. Al igual que con las pruebas de los *DAOs*, para estas pruebas se hace uso de una **base de datos en memoria** cuyo estado es el mismo para cada caso de prueba.

8.6.3.1.1 *ViewModel* de notificación de positivos

Este *viewmodel* se encarga de las operaciones relacionadas con la notificación de positivos, por lo que estas pruebas irán destinadas a probar la lógica de comprobación de errores de notificación, así como los propios resultados de notificar un positivo. Mediante estas pruebas se verifica el comportamiento **conjunto** de los siguientes componentes:

- *ViewModel* de notificación de positivos.
- *Manager de notificación de positivos.*
- Repositorios de localizaciones de usuario y de positivos.

En este caso concreto, se ***mockean*** las siguientes **dependencias**:

- Repositorio de configuración: para establecer una configuración determinada sin hacer llamadas de red.
- Repositorio de datos personales: para establecer unos datos personales de prueba sin acceder a las *Shared Preferences* de la aplicación *Android.*
- Cliente de la *API* de positivos: para *mockear* las llamadas de red al notificar un positivo, devolviendo una respuesta falsa de prueba.

A continuación, se muestra el **diseño** de pruebas con las condiciones y situaciones de prueba.

Condiciones de Prueba
Límite de notificación
No han transcurrido suficientes días para volver a notificar
Han transcurrido suficientes días para volver a notificar
Localizaciones a subir
Existen localizaciones
No existen localizaciones
Resultado de la notificación
Éxito
Tiempo de espera agotado
Error HTTP
Datos personales
Incluir datos personales
No incluir datos personales
Periodo de infectividad
Cargar periodo de infectividad
Localizaciones filtradas por el periodo de infectividad

Tabla 8.34. Situaciones de prueba para el viewmodel de notificación de positivos.

Situaciones de Prueba	Código
No han transcurrido suficientes días para volver a notificar	NPVM1
Han transcurrido suficientes días para volver a notificar	
No existen localizaciones	NPVM2
Existen localizaciones y son filtradas por el periodo de infectividad	
Éxito	
Incluir datos personales	NPVM3
No incluir datos personales	NPVM4
Tiempo de espera agotado	NPVM5
Error HTTP	NPVM6
Cargar periodo de infectividad	NPVM7

La siguiente tabla muestra los **casos de prueba** que cubren las situaciones de prueba (SP) descritas en la anterior tabla.

Tabla 8.35. Casos de prueba para el viewmodel de notificación de positivos.

***ViewModel* de notificación de positivos**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
NPVM1, NPVM3, NPVM4	NPVM134	Notificar un positivo con éxito sin incluir datos personales y comprobar que se almacena correctamente. Intentar notificar otro positivo, no se podrá porque no han transcurrido suficientes días. Actualizar la fecha del positivo para que transcurran suficientes días y volver a notificar otro positivo incluyendo datos personales.	* Configuración de la notificación. * Localizaciones a subir. * Respuestas a las preguntas. * Datos personales.	Se notifica correctamente el primer positivo con código de éxito, el segundo no se notifica con código de error y el tercero se notifica con éxito incluyendo datos personales. Se filtran las localizaciones en función del Periodo de Infectividad.
NPVM2	NPVM2	Notificar un positivo sin superar el límite y sin datos personales, pero sin existir localizaciones que subir.	* Configuración de la notificación. * Respuestas a las preguntas. * Lista vacía de localizaciones.	Se obtiene un código de Error indicando que no existen localizaciones para subir a la nube.
NPVM5	NPVM5	Notificar un positivo generando un error de red.	* Configuración de la notificación. * Respuestas a las preguntas. * Localizaciones a subir.	Se obtiene un código de Error indicando Tiempo de Espera Agotado.
NPVM6	NPVM6	Notificar un positivo generando un error HTTP.	* Configuración de la notificación. * Respuestas a las preguntas. * Localizaciones a subir.	Se obtiene un código de Error indicando que no se ha podido notificar un positivo.
NPVM7	NPVM7	Cargar el periodo de infectividad de la configuración en el LiveData.	* Configuración de la notificación.	Se carga el LiveData con el periodo de infectividad establecido en la configuración.

8.6.3.1.2 *ViewModel* de comprobación de contactos

Gestiona todas las operaciones involucradas en la **comprobación de contactos de riesgo**. En estas pruebas se incluyen **dos** *viewmodels*: el de la pantalla de comprobación de contactos y el de la pantalla de resultados. Las pruebas de integración para estos componentes se centran en comprobar la lógica para recuperar las localizaciones del usuario y compararlas con las de un positivo ficticio mediante el ***manager* de contactos de riesgo** que no fue probado en las pruebas unitarias descritas en la sección 8.6.2.4 MANAGERS. Mediante estas pruebas se verifica que los siguientes componentes funcionan bien en **conjunto**:

- *ViewModel* de comprobación de contactos y de resultados de comprobación.
- *Manager* de contactos de riesgo y de alarmas de comprobación.
- Repositorios de contactos de riesgo, positivos y localizaciones de usuario.

Por otro lado, se ***mockean*** las siguientes **dependencias:**

- Repositorio de configuración: para establecer una configuración fija.
- *Shared Preferences* de *Android.*
- Repositorio de estadísticas: para simular las llamadas de red a la *API* de estadísticas.
- Cliente de la *API* de positivos: para establecer unos positivos ficticios que se obtienen de la nube.
- *Manager* de notificaciones internas de la aplicación: para simular que se muestran las notificaciones.
- *Helper* de alarmas de *Android.*

Además, para las **entradas** de los casos de prueba, se reutilizan los **ficheros de coordenadas** que se utilizaban en las pruebas unitarias del algoritmo de comprobación.

A continuación, se muestra el **diseño** de pruebas para estos componentes, incluyendo las condiciones y situaciones de prueba obtenidas.

Condiciones de Prueba
Añadir alarma de comprobación
Éxito
Colisión entre alarmas
Límite de alarmas superado
Error al insertar una alarma
Actualizar alarmas de comprobación
Activar alarma
Desactivar alarma
Eliminar alarma
Itinerario del usuario
No hay localizaciones
Hay localizaciones
Positivos
No hay positivos notificados dentro del alcance de la comprobación
Hay 1 positivo dentro del alcance de la comprobación
Hay varios positivos dentro del alcance de la comprobación
Positivo notificado por el usuario
Existe un positivo notificado por el usuario recientemente (dentro del alcance de la comprobación)
No existe un positivo notificado por el usuario
Resultado de la comprobación
No hay contactos de riesgo
Hay contactos de riesgo
Días del itinerario
Un día
Varios días
Resultado de la obtención de positivos
Éxito
Tiempo de espera agotado
Error HTTP

Tabla 8.36. Situaciones de prueba para los viewmodels de comprobación de contactos y resultados.

Situaciones de Prueba	Código
Añadir una alarma de comprobación con éxito.	RCVM1
Añadir una alarma de comprobación superando el límite.	RCVM2
Añadir una alarma de comprobación generando una colisión.	RCVM3
Añadir una alarma de comprobación generando un error de inserción de alarma.	RCVM4
Activar alarma.	RCVM5
Desactivar alarma.	RCVM6
Eliminar alarma.	RCVM7
No hay localizaciones en el itinerario del usuario.	RCVM8
Hay localizaciones en el itinerario del usuario divididas en varios días.	
No hay positivos notificados dentro del alcance de la comprobación.	RCVM9
Hay un positivo notificado dentro del alcance de la comprobación con un itinerario de varios días.	
No hay contactos de riesgo entre el positivo y el usuario.	RCVM10
Hay contactos de riesgo entre el positivo y el usuario.	RCVM11
Hay varios positivos dentro del alcance de la comprobación.	
Existe un positivo notificado por el usuario dentro del alcance de la comprobación.	
Hay contactos de riesgo entre el otro positivo y el usuario.	RCVM12
No existe un positivo notificado por el usuario dentro del alcance.	
Hay contactos de riesgo entre un positivo y el usuario, el resto de positivos no generan contactos.	RCVM13
Hay contactos de riesgo entre varios positivos y el usuario repartidos en varios días.	RCVM14
No hay positivos porque surge un error de tiempo de espera agotado.	RCVM15
No hay positivos porque surge un error HTTP.	RCVM16

La siguiente tabla muestra los **casos de prueba** diseñados para cubrir las situaciones de prueba expuestas en la tabla superior.

Tabla 8.37. Casos de prueba para los viewmodels de comprobación de contactos y resultados de la comprobación.

***ViewModels* para la comprobación de contactos y los resultados de la comprobación**				
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**
RCVM1, RCVM2, RCVM3, RCVM4	RCVM1234	Añadir una nueva alarma de comprobación con éxito. Añadir otra alarma de comprobación distinta que genere un error de inserción. Insertar una segunda alarma con éxito. Intentar añadir una tercera alarma de comprobación que genere colisión con la segunda alarma. Añadir una tercera alarma de comprobación distinta que no genere colisión. Por último, intentar añadir una cuarta alarma de comprobación que no genere colisión, no se debería insertar porque ya se ha superado el límite.	* Alarmas de comprobación. * Límite de alarmas de comprobación (3).	Se almacenan las tres alarmas de comprobación en la base de datos devolviéndose un código de Éxito. Para la alarma que genera error de inserción se devuelve el código de error de inserción. Para la alarma que genera colisión se devuelve un código de Error de colisión de alarmas y para la que supera el límite se devuelve un código de error de Límite superado.

RCVM5, RCVM6	RCVM56	Añadir una nueva alarma de comprobación con éxito. Desactivarla y luego volver a activarla, comprobando que cambia de estado correctamente.	* Alarma de comprobación.	Se añade la alarma con éxito, luego se desactiva y activa correctamente.
RCVM7	RCVM7	Añadir una nueva alarma de comprobación con éxito y luego eliminarla.	* Alarma de comprobación.	Se añade la alarma con éxito y luego se elimina por completo.
RCVM8	RCVM8	Ejecutar la comprobación de contactos de riesgo cuando no existen localizaciones del usuario registradas dentro del alcance de la comprobación.	* Configuración de la comprobación. * Fecha actual: "28/06/2021 17:49:00" * Itinerario 15 (usuario)	Se obtiene un resultado vacío sin contactos de riesgo.
RCVM9	RCVM9	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local, pero no hay positivos notificados con localizaciones dentro del alcance de comprobación.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 2 (positivo)	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.
RCVM10	RCVM10	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y existe un positivo notificado con un itinerario de varios días dentro del alcance de la comprobación, pero no genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 16 (positivo)	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.
RCVM11	RCVM11	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y existe un positivo notificado con un itinerario de varios días dentro del alcance de la comprobación que genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo)	Se obtiene un resultado de comprobación con dos contactos de riesgo con el positivo.
RCVM12	RCVM12	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Uno de esos positivos ha sido notificado por el propio usuario y genera contactos, pero debería ser ignorado y el otro genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo) * Itinerario 5 (positivo del usuario)	Se obtiene un resultado con dos contactos de riesgo porque el positivo notificado por el propio usuario genera contactos, pero no se tiene en cuenta en la comprobación.

RCVM13	RCVM13	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Solo uno de los positivos genera contactos de riesgo, el resto no.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo que genera contactos) * Itinerario 14 (positivo) * Itinerario 1 (positivo)	Se obtiene un resultado con dos contactos de riesgo generados por un positivo porque el resto no genera contactos.
RCVM14	RCVM14	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Varios de estos positivos generan contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 20 (positivo que genera contactos) * Itinerario 5 (positivo que genera contactos) * Itinerario 19 (positivo que genera contactos)	Se obtiene un resultado con cuatro contactos de riesgo generados por varios positivos.
RCVM15	RCVM15	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días, pero no hay positivos porque se genera un error de Tiempo de Espera agotado.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario)	No se obtiene ningún resultado porque no se ha ejecutado la comprobación debido a un error de Tiempo de espera agotado.
RCVM16	RCVM16	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días, pero no hay positivos porque se genera un error HTTP.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario)	No se obtiene ningún resultado porque no se ha ejecutado la comprobación debido a un error HTTP.

8.6.3.2 *API REST*

Además de los *viewmodels*, las pruebas de integración también se realizan sobre la ***API REST*** que proporciona el *backend*. En este caso, las pruebas se ejecutan sobre el propio equipo de sobremesa ya que el servidor *web* es desplegado en **local**.

Para llevar a cabo estas pruebas, se ***mockea*** el **cliente *web*** que consume la *API*, de forma que se simulan las **peticiones *GET*** y ***POST*** lanzadas sobre los *endpoints* expuestos por el servidor. El cliente falso envía las peticiones a la *API* con los parámetros y los datos en el cuerpo de la petición que sean necesarios, se espera a recibir la respuesta desde el servidor y una vez recibida se realizan las comparaciones y asertos necesarios para verificar el funcionamiento.

De este modo, se verifica el funcionamiento **integrado** de los siguientes componentes de la arquitectura del *backend*:

- *Routers* con las rutas de los *endpoints*.
- Controladores para gestionar las peticiones.

- Repositorios para leer y escribir datos de la base de datos en la nube.

Cabe destacar que no se utiliza una base de datos en memoria como en los anteriores casos, sino que para las pruebas de la *API REST* se hace uso de la **base de datos real** que se encuentra desplegada en la nube, dentro de la plataforma de *Firebase*. Una alternativa sería emplear un **emulador** de la base de datos *Firestore* que se ejecutara en la máquina local, pero para simplificar la implementación se ha decidido emplear la base de datos real.

Por ello, con el objetivo de evitar en la medida de lo posible alterar el **estado** de la base de datos que se utiliza en **producción,** se ha reestructurado el directorio de la base de datos para servir una versión de **producción** y otra versión de ***test***. De este modo, en función del entorno de ejecución en el que se despliegue el servidor, las lecturas y escrituras se realizarán en una versión u otra de la base de datos. Ambas versiones son idénticas, incluyendo los nombres de las colecciones y documentos, pero son utilizadas en entornos distintos para evitar mezclar los **datos reales** con los **datos de prueba.**

En los siguientes apartados, se describe el diseño de pruebas, identificando las condiciones de prueba, descomponiéndolas en clases de equivalencia y combinando estas para obtener las situaciones de prueba para las distintas *APIs* proporcionadas por el servidor *web*, incluyendo los casos de prueba que cubren dichas situaciones.

8.6.3.2.1 *API* de configuración

Las pruebas de integración de esta *API* verifican el funcionamiento de las operaciones de actualización y lectura de los ficheros de configuración de *Contact Tracker*. Las siguientes tablas muestran las **condiciones de prueba** identificadas con sus clases de equivalencia y la jerarquía de **situaciones de prueba** obtenida tras la combinación.

Condiciones de Prueba
Fichero de configuración
Configuración de la notificación de positivos
Configuración de la comprobación de contactos
Fichero de configuración vacío
Operación
Recuperar la configuración
Actualizar la configuración
Existencia del fichero
Fichero de configuración existe
Fichero de configuración no existe

Tabla 8.38 Situaciones de prueba para la API de configuración.

Situación de Prueba	Código
Recuperar configuración.	
Fichero de configuración no existe.	APIC1
Fichero de configuración existe.	
Configuración de la notificación.	APIC2
Configuración de la comprobación.	APIC3
Actualizar la configuración.	

Fichero de configuración pasado como parámetro vacío.	APIC4
Configuración de la notificación.	APIC5
Configuración de la comprobación.	APIC6

A partir de estas situaciones de prueba, se obtienen los **casos de prueba** juntos con las situaciones que cubren (columna **SP**) que se muestran en la siguiente tabla.

Tabla 8.39. Casos de prueba para la API de configuración.

***API* de configuración**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
APIC1	APIC1	Recuperar la configuración de un fichero que no existe.	* Nombre de fichero que no existe: “no-existe”	* Código: 404 Not found
APIC2	APIC2	Recuperar la configuración de la notificación de positivos.	* Nombre de fichero: “notify-config”	* Código: 200 OK * Objeto con la configuración de la notificación.
APIC3	APIC3	Recuperar la configuración de la comprobación de contactos de riesgo.	* Nombre de fichero: “risk-contact-config”	* Código: 200 OK * Objeto con la configuración de la comprobación.
APIC4	APIC4	Intentar actualizar cualquier configuración con un fichero de configuración vacío y luego directamente sin enviar ningún parámetro en la petición.	* Objeto de configuración vacío.	* Código: 400 Bad Request
APIC5	APIC5	Actualizar todos los campos de la configuración de la notificación con valores distintos.	* Objeto con la nueva configuración de la notificación de positivos.	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: ‘Configuración actualizada correctamente.’}
APIC6	APIC6	Actualizar todos los campos de la configuración de la comprobación con valores distintos.	* Objeto con la nueva configuración de la comprobación de contactos.	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: ‘Configuración actualizada correctamente.’}

8.6.3.2.2 *API* de positivos

Para esta *API*, las pruebas de integración verifican el correcto funcionamiento de las operaciones de notificación de un positivo, así como de las consultas para obtener los positivos con localizaciones registradas un número de días atrás determinado. A continuación, se muestra el **diseño** de las pruebas para la *API* de positivos.

Condiciones de Prueba
Obtener todos los positivos
No hay positivos notificados
Hay varios positivos notificados

Notificar un positivo
Positivo enviado en la petición es un objeto vacío
Positivo enviado en la petición tiene localizaciones registradas en un solo día
Positivo enviado en la petición tiene localizaciones registradas en varios días
Obtener positivos con localizaciones registradas días atrás
No hay positivos con localizaciones registradas en alguno de los últimos días indicados
Solo un positivo con localizaciones registradas en varios de los últimos días indicados
Varios positivos con localizaciones registradas en varios de los últimos días indicados
Filtro de fechas
Localizaciones registradas en los últimos días desde la fecha de referencia.
Fecha de referencia distinto mes al de las localizaciones registradas en los últimos días.

Tabla 8.40. Situaciones de prueba para la API de positivos.

Situaciones de Prueba	Código
Obtener todos los positivos cuando hay varios positivos notificados.	APIP1
Obtener todos los positivos cuando hay no hay ningún positivo notificado.	APIP2
Notificar un positivo enviando un objeto vacío en la petición.	APIP3
Notificar un positivo enviando un positivo con localizaciones registradas en un solo día.	APIP4
Notificar un positivo enviando un positivo con localizaciones registradas en varios días.	APIP5
Obtener positivos con localizaciones registradas días atrás cuando ningún positivo tiene localizaciones registradas en algunos de esos días.	APIP6
Obtener un solo positivo que tiene localizaciones registradas en varios de los días atrás.	
Localizaciones registradas en los últimos días desde la fecha de referencia.	APIP7
Fecha de referencia es de distinto mes al de las localizaciones registradas en los últimos días.	APIP8
Obtener varios positivos que tienen localizaciones registradas en varios de los días atrás.	APIP9

La siguiente tabla refleja los **casos de prueba** diseñados para cubrir las situaciones de prueba expuestas en la tabla anterior.

Tabla 8.41. Casos de prueba para la API de positivos.

***API* de Positivos**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
APIP1	APIP1	Recuperar todos los positivos notificados cuando en la base de datos hay varios positivos almacenados.	* Positivos almacenados en la BD.	* Código 200 OK * Lista con todos los positivos almacenados en la base de datos y sus datos correspondientes.
APIP2	APIP2	Recuperar todos los positivos notificados cuando en la base de datos no hay ningún positivo almacenado.	* Base de datos sin positivos.	* Código 200 OK * Lista de positivos vacía.
APIP3	APIP3	Intentar notificar un positivo enviando un objeto vacío en la petición.	* Objeto vacío en la petición.	* Código 400 Bad request

APIP4	APIP4	Notificar un positivo enviando un objeto con localizaciones registradas en un solo día.	* Positivo con localizaciones registradas en un solo día.	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: “”, uploadedLocations: 5}
APIP5	APIP5	Notificar un positivo enviando un objeto con localizaciones registradas en vario días.	* Positivo con localizaciones registradas en varios días.	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: “”, uploadedLocations: 5}
APIP6	APIP6	Obtener los positivos notificados con localizaciones registradas en los últimos días sin que coincida ningún positivo.	* Fecha de referencia: “05/10/2021 15:24:06” * N.º de días: 2	* Lista vacía porque no hay positivos con localizaciones registradas en esos dos últimos días.
APIP7	APIP7	Obtener solo un positivo que tiene localizaciones registradas en los últimos días indicados.	* Fecha de referencia: “05/10/2021 15:24:06” * N.º de días: 3	* Lista con el positivo 2 porque tiene localizaciones registradas en el día 02 de octubre.
APIP8	APIP8	Obtener positivos que tienen localizaciones registradas en los últimos días indicados tomando como referencia una fecha de un mes distinto.	* Fecha de referencia: “02/10/2021 15:24:06” * N.º de días: 3	* Lista con el positivo 1 y el positivo 2 porque tienen localizaciones registradas en los días indicados.
APIP9	APIP9	Obtener varios positivos que tienen localizaciones registradas en alguno de los días indicados teniendo en cuenta la fecha de referencia.	* Fecha de referencia: “28/09/2021 15:24:06” * N.º de días: 5	* Lista con el positivo 1, el positivo 2 y el positivo 3, porque tienen localizaciones registradas en los días indicados.

8.6.3.2.3 *API* de estadísticas

La última *API* a probar es la *API* dedicada a las estadísticas de *Contact Tracker*. Mediante las pruebas de integración se verifica la lógica de negocio relativa a las operaciones de registro y recuperación de estadísticas, probando con diferentes parámetros en las peticiones. Las siguientes tablas reflejan el **diseño** de pruebas para esta *API.*

Condiciones de Prueba
Registrar estadística
Registrar instalación
Registrar resultado de una comprobación
Obtener estadísticas
Positivos
Resultados de comprobaciones
Instalaciones
Parámetro de número de días atrás
Negativo (todos los días)
0
Mayor que 0
Resultado de obtener estadísticas

No hay estadísticas (lista vacía)
Hay estadísticas
Parámetro de registrar estadística
Objeto vacío
Objeto correcto con la estadística

Tabla 8.42. Situaciones de prueba para la API de estadísticas.

Situaciones de Prueba	Código
Registrar estadística	
Registrar una instalación	
Objeto enviado en la petición está vacío.	APIS1
Objeto enviado en la petición es una instalación correcta.	APIS2
Registrar un resultado de una comprobación	
Objeto enviado en la petición está vacío.	APIS3
Objeto enviado en la petición es un resultado de una comprobación correcto.	APIS4
Obtener estadísticas	
Obtener estadísticas cuando no hay datos (lista vacía)	APIS5
Obtener estadísticas cuando si hay datos	
Parámetro del número de días 0.	
Obtener positivos notificados.	APIS6
Obtener resultados de comprobaciones.	APIS7
Obtener instalaciones registradas.	APIS8
Parámetro del número de días mayor que 0.	
Obtener positivos notificados.	APIS9
Obtener resultados de comprobaciones.	APIS10
Obtener instalaciones registradas.	APIS11
Parámetro del número de días negativo (todos los días).	
Obtener positivos notificados.	APIS12
Obtener resultados de comprobaciones.	APIS13
Obtener instalaciones registradas.	APIS14

A continuación, se muestra una tabla con los **casos de prueba** que cubren las situaciones de prueba obtenidas en la tabla superior.

Tabla 8.43. Casos de prueba para la API de estadísticas.

***API* de estadísticas**				
SP	**Código**	**Procedimiento**	**Entradas**	**Salida esperada**
APIS1	APIS1	Registrar una instalación enviando un objeto vacío en la petición POST.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Objeto vacío en el cuerpo de la petición.	* Código 400 Bad Request. * No se registra la instalación.

APIS2	APIS2	Registrar una instalación enviando un objeto correcto en el cuerpo de la petición POST.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Nueva instalación a registrar.	* Código 200 OK. * Instalación registrada correctamente. * Objeto de respuesta: {registered: true, msg: "Instalación registrada."}
APIS3	APIS3	Registrar un resultado de comprobación enviando en el cuerpo de la petición POST un objeto vacío.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Objeto vacío en el cuerpo de la petición.	* Código 400 Bad Request. * No se registra el resultado.
APIS4	APIS4	Registrar un resultado de comprobación enviando en el cuerpo de la petición POST un objeto que representa una comprobación correcta.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Nueva resultado a registrar.	* Código 200 OK. * Resultado registrado correctamente. * Objeto de la respuesta: {registered: true, msg: "Resultado de la comprobación registrado."}
APIS5	APIS5	Obtener estadísticas de positivos, instalaciones y resultados de comprobación cuando no existen datos filtrados en los últimos días indicados desde la fecha de referencia.	* Fecha de referencia: "05/10/2021 12:55:04" *Número de días: 2	* Código 200 OK * Resultados vacíos
APIS6	APIS6	Recuperar las estadísticas de los positivos notificados en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: "26/09/2021 12:00:00" *Número de días: 0	* Código 200 OK * Objeto con las estadísticas del número de positivos notificado hoy.
APIS7	APIS7	Recuperar las estadísticas de los resultados de las comprobaciones registrados en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: "26/09/2021 12:00:00" *Número de días: 0	* Código 200 OK * Un resultado de una comprobación.
APIS8	APIS8	Recuperar las estadísticas de las instalaciones registradas en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: "26/09/2021 12:00:00" *Número de días: 0	* Código 200 OK * Dos instalaciones registradas.
APIS9	APIS9	Recuperar las estadísticas de positivos notificados en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: "01/10/2021 12:00:00" *Número de días: 2	* Código 200 OK * Dos positivos notificados.
APIS10	APIS10	Recuperar las estadísticas de resultados de comprobaciones registrados en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: "01/10/2021 12:00:00" *Número de días: 3	* Código 200 OK * Dos resultados de comprobación.
APIS11	APIS11	Recuperar las estadísticas de instalaciones registradas en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: "28/09/2021 12:00:00" *Número de días: 2	* Código 200 OK * Tres instalaciones registradas.

APIS12	APIS12	Recuperar las estadísticas de positivos notificados en los últimos días pasando como parámetro del número de días un valor negativo igual a -1, que representa todos los días.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todos los positivos notificados.
APIS13	APIS13	Recuperar las estadísticas de resultados de comprobaciones registrados en los últimos días pasando como parámetro del número de días un valor negativo igual a -1, que representa todos los días.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todos los resultados registrados.
APIS14	APIS14	Recuperar las estadísticas de instalaciones registradas en los últimos días pasando como parámetro del número de días un valor negativo igual a -1 que representa todos los días.	* Fecha de referencia: “28/09/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todas las instalaciones registradas.

8.6.4 Pruebas del Sistema

Las pruebas del sistema son el último tipo de pruebas implementadas mediante código. Se sitúan por encima de las pruebas de integración, ya que están enfocadas en probar el funcionamiento global del sistema, con todos sus componentes integrados. Esto incluye la **interfaz de usuario**, que interacciona con el resto de los componentes de la aplicación que fueron probados anteriormente. Por ello, aunque se prueben todos los componentes en conjunto, el **objetivo** principal de estas pruebas es verificar que el comportamiento de la interfaz de usuario ante las acciones que este realiza es el esperado.

Este tipo de pruebas se realizan sobre la **aplicación móvil**, y dado que se requiere de la interfaz de usuario para ejecutarlas, será necesario un **emulador** virtual con *Android* instalado.

Las pruebas del sistema para la aplicación móvil se dividen en **casos de uso** a grandes rasgos, es decir, existen diferentes pruebas del sistema por cada operación de alto nivel que se pueda realizar, de modo que se distingue entre:

- Rastreo de ubicación.
- Notificación de positivos.
- Comprobación de contactos.
- Histórico de localizaciones.
- Política de privacidad.
- Ajustes.

Se podrían hacer pruebas de sistema más complejas y cubriendo más detalles de la interfaz de las que se han diseñado. Sin embargo, para este proyecto se ha optado por cubrir a grandes rasgos los principales aspectos de la interfaz de usuario para evitar contratiempos y simplificar la implementación.

Antes de pasar a ver el diseño de estas pruebas, es importante destacar que no se ha realizado un diseño previo identificando condiciones de prueba, clases de equivalencia y situaciones de prueba como en los anteriores apartados. En su lugar, se han diseñado los casos de prueba directamente para simplificar, dada la complejidad de probar todo el sistema con sus

componentes integrados. Además, antes de ejecutar cualquier caso de prueba, se define un **procedimiento previo** compuesto de unos pasos a seguir para preparar el entorno, los datos de prueba y la base de datos, de forma que todas las pruebas se ejecuten en el mismo estado y su resultado sea consistente.

8.6.4.1 *Rastreo de ubicación*

Estas pruebas se centran en las pantallas de rastreo de ubicación y creación de alarmas de localización. El procedimiento previo a cada caso de prueba está formado por los siguientes pasos:

1. Encender el servidor *web* en local con la base de datos de *test* (*npm run startTest*).
2. Dar permisos de rastreo de ubicación a la aplicación móvil.
3. Activar el *GPS* del dispositivo de pruebas.

A continuación, se muestra la tabla con los casos de prueba diseñados para el rastreo de ubicación.

Tabla 8.44. Casos de prueba para las pantallas del rastreo de ubicación.

Rastreo de ubicación			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
R1	1. Ir al rastreador. 2. Activar el servicio de rastreo. 3. Esperar 3 segundos. 4. Desactivar el servicio de rastreo.		Comprobar los textos de la vista. Comprobar que desaparece la etiqueta de lista vacía cuando se activa el servicio, así como que aparece la notificación correspondiente. Comprobar que aparecen los *Snackbars* de encendido y apagado del servicio.
R2	1. Ir a la vista del mapa. 2. Comprobar que los marcadores están activados. 3. Desactivar los marcadores. 4. Volver a activarlos. 4. Pulsar sobre el FAB del menú de capas. 5. Comprobar que aparecen las tres capas posibles.		Se muestra la vista del mapa con las distintas opciones de capas. Comprobar el estado del Chip de marcadores.
R3	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 11:10 5. Comprobar que se muestra el Snackbar de error de alarma inválida. 6. Comprobar que no se inserta ninguna alarma.	Hora de inicio: 12:15 Hora de fin: 11:10	No se inserta ninguna alarma de rastreo y se muestra un Snackbar con el error de alarma inválida.
R4	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 12:45 5. Comprobar que se inserta la alarma correctamente. 6. Insertar como hora de inicio: 12:20 y como hora de fin 12:35 7. Comprobar que se muestra el Snackbar de colisión entre alarmas 8. Comprobar que no se ha insertado la alarma.	Hora de inicio: 12:15 Hora de fin: 12:45	Se inserta la primera alarma de rastreo, pero no la segunda.

R5	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 12:45 5. Comprobar que se inserta la alarma correctamente. 6. Pulsar sobre la cruz roja de la alarma. 7. Comprobar que se elimina la alarma y se muestra la etiqueta de que no hay alarmas.	Hora de inicio: 12:15 Hora de fin: 12:45	Se inserta la alarma de rastreo y luego se elimina.

8.6.4.2 *Notificación de positivos*

Involucra las pruebas relacionadas con la pantalla de notificación de positivos mediante la aplicación móvil. El procedimiento previo a los casos de prueba es el siguiente:

1. Encender el servidor *web* en local con la base de datos de test (*npm run startTest*).
2. Borrar los datos de la aplicación en el dispositivo.
3. Cargar en la aplicación el fichero *positive.txt* que contiene las localizaciones del positivo.
4. Establecer como fecha de esas localizaciones la correspondiente a la actual menos dos días.
5. Establecer como Periodo de Infectividad 3 días.

La siguiente tabla muestra los casos de prueba diseñados para la notificación de positivos.

Tabla 8.45. Casos de prueba para la pantalla de notificación de positivos.

Notificación de positivos			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
NP1	1. Ir a la pantalla de Notificación de Positivos. 2. Comprobar el contenido de los textos explicativos. 3. Comprobar que el Checkbox de adjuntar datos personales está desactivado. 4. Comprobar que se muestran los días del periodo de infectividad. 5. Pulsar sobre el CheckBox y comprobar que cambia su estado. 6. Pulsar sobre el botón de Política de Privacidad. 7. Comprobar que se muestra la pantalla con la política de privacidad. 8. Volver atrás.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos.	Se muestran los textos explicativos correspondientes, se muestra la política de privacidad y el número de días del periodo de infectividad establecido en la configuración del Backend.
NP2	1. Ir a la pantalla de Notificación de Positivos. 2. Pulsar sobre el botón para notificar un positivo. 3. Comprobar que aparece el diálogo con las preguntas de asintomático y vacunado directamente y luego cancelar la notificación. 4. Pulsar sobre el CheckBox para adjuntar datos personales. 5. Volver a pulsar sobre el botón de Notificar un Positivo. 6. Comprobar que ahora aparece el formulario para rellenar datos personales. 7. Rellenarlo con los datos personales y pulsar Aceptar. 8. Comprobar que aparece la Cláusula de Consentimiento. 9. Cancelar la cláusula.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos. * Datos personales.	Se muestran los diálogos correspondientes con los formularios para adjuntar los datos personales solo cuando el Checkbox está activado, sino se muestra directamente las preguntas de asintomático y vacunado.

NP3	1. Ir a la pantalla de Notificación de Positivos. 2. Pulsar sobre el CheckBox para adjuntar los datos personales. 3. Pulsar sobre el botón para notificar un positivo. 4. Pulsar Aceptar en el formulario de datos personales sin rellenar ningún campo. 5. Comprobar que no se avanza de diálogo y que se muestran los errores correspondientes debajo de cada campo de texto.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos.	Comprobar que se muestran los errores de campos requeridos y que no se avanza de diálogo cuando se intenta notificar un positivo adjuntando datos personales vacíos.
NP4	1. Hacer que las localizaciones leídas desde el fichero tengan fechas de hace 5 días atrás. 2. Ir a la pantalla de Notificación de Positivos. 3. Pulsar sobre el botón de Notificar un Positivo. 4. Pulsar el botón de Aceptar en el diálogo de las preguntas. 5. Comprobar que aparece un Snackbar indicando un error de que no existen localizaciones para subir a la nube.	* Configuración de la notificación de positivos.	Comprobar que se muestra el Snackbar con el error de inexistencia de localizaciones cuando se intenta notificar un positivo.
NP5	1. Ir a la pantalla de notificación de positivos. 2. Pulsar sobre el botón para notificar un positivo. 3. Aceptar el diálogo modal con las preguntas. 4. Comprobar que aparece el Snackbar indicando el número de localizaciones que se han subido a la nube. 5. Pulsar de nuevo en el botón de notificar un positivo. 6. Comprobar que aparece el Snackbar indicando el error de que se ha superado el límite de notificación de positivos.	* Localizaciones del fichero de positivo. * Configuración de la notificación de positivos.	El primer positivo se notifica correctamente, mostrando un Snackbar con el número de localizaciones subidas a la nube. El segundo positivo no se notifica y se muestra un Snackbar indicando que se ha superado el límite de notificación.

8.6.4.3 *Comprobación de contactos*

Las pruebas de la comprobación de contactos se centran en la interfaz de las pantallas para ejecutar una comprobación y programar alarmas de comprobación, y en la pantalla de resultados de comprobaciones. Los pasos a seguir antes de ejecutar los casos de prueba son:

1. Encender el servidor *web* en local con la base de datos de test (*npm run startTest*).
2. Eliminar de la base de datos de *test* todos los positivos que haya notificados.
3. En otro dispositivo distinto al de pruebas, cargar las localizaciones desde el fichero *positive.txt.*
4. Establecer como fecha de las localizaciones la correspondiente a la actual menos dos días atrás.
5. Notificar un positivo con ese dispositivo subiendo a la nube las localizaciones del fichero *positive.txt.*
6. En el dispositivo bajo pruebas, cargar las localizaciones desde el fichero *user.txt.*
7. Establecer como fecha de las localizaciones del usuario la correspondiente a la actual menos dos días atrás.
8. Borrar los datos de la aplicación en el dispositivo bajo pruebas.

A continuación, se muestra una tabla con los casos de prueba diseñados para la comprobación de contactos.

Tabla 8.46. Casos de prueba para las pantallas de comprobación de contactos.

Comprobación de contactos			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
CC1	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Comprobar la presencia de los títulos y textos correspondientes. 3. Comprobar que el modo de comprobación manual está seleccionado. 4. Seleccionar el modo de comprobación periódica y comprobar que el otro modo se deshabilita. 5. Moverse a la pantalla de Notificación de positivos y volver a la de comprobación. 6. Comprobar que el modo de comprobación periódica sigue habilitado.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación.	Se muestran los títulos y textos explicativos de la comprobación de contactos. Se cambia de modo de comprobación correctamente, persistiendo entre cambios de pantalla.
CC2	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Seleccionar el modo de comprobación periódica. 3. Insertar como hora de comprobación 10:30. 4. Comprobar que aparece el Snackbar de alarma programa correctamente. 5. Comprobar que aparece el Chip correspondiente a la alarma recién insertada. 6. Intentar insertar otra alarma con la misma hora 10:30. 7. Comprobar que aparece el Snackbar de error de colisión de alarmas. 8. Seleccionar como hora de comprobación las 11:00 e insertar una segunda alarma. 9. Comprobar que aparece el Snackbar y el Chip correspondiente a la alarma insertada. 10. Insertar una tercera alarma de comprobación con la hora 16:00. 11. Seleccionar como hora de comprobación las 18:00. 12. Comprobar que aparece el Snackbar de error de que se ha superado el límite de alarmas de comprobación permitido. 13. Eliminar la segunda alarma de comprobación. 14. Insertar la cuarta alarma con hora 18:00 y comprobar que se inserta correctamente.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación. * Horas de comprobación: [10:30, 11:00, 16:00, 18:00]	Las tres primeras alarmas se insertan correctamente. La alarma que genera colisión no se inserta y la cuarta alarma tampoco porque se supera el límite. Se borra la segunda alarma y se permite insertar la cuarta alarma.
CC3	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Pulsar sobre el botón para realizar una comprobación de contactos de riesgo. 3. Ir al Tab con los resultados de la comprobación. 4. Comprobar que se muestran los valores correspondientes en el item del resultado. 5. Pulsar en el item del resultado y comprobar que se navega hacia los detalles del resultado. 6. Comprobar que todos los valores que se muestran se corresponden con los esperados. 7. Ir atrás en la aplicación. 8. Comprobar que sigue existiendo el item con el resultado.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación.	Se realiza la comprobación de contactos de riesgo correctamente, mostrándose el resultado con los valores calculados correctamente. Se muestran los detalles esperados del resultado.

8.6.4.4 *Histórico de localizaciones*

Estas pruebas engloban el comportamiento de la interfaz en la pantalla del histórico de localizaciones registradas en el dispositivo. En el procedimiento previo se incluyen los siguientes pasos:

1. Encender el servidor *web* en local con la base de datos de test (*npm run startTest*).
2. Cargar en la aplicación móvil las localizaciones del fichero *positive.txt.*

3. Establecer como fecha de registro de las localizaciones el día *04/MM/2021*, sustituyendo el mes por el mes en el que se realice la prueba.

La siguiente tabla muestra los casos de prueba diseñados para el histórico de localizaciones.

Tabla 8.47. Casos de prueba para la pantalla del histórico de localizciones.

Histórico de localizaciones			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
H1	1. Ir a la pantalla del histórico de localizaciones. 2. Comprobar la presencia de la etiqueta de lista vacía. 3. Seleccionar en el campo de fechas la fecha 05/MM/2021 4. Comprobar que no hay localizaciones y que por tanto se muestra la etiqueta de lista vacía.	* Localizaciones del fichero del positivo.	No hay localizaciones en los días especificados, ni en el día actual ni en el día 05/MM/2021.
H2	1. Ir a la pantalla del histórico de localizaciones. 2. Comprobar la presencia de la etiqueta de lista vacía. 3. Seleccionar en el campo de fechas la fecha 04/MM/2021 4. Comprobar la existencia de los botones de Mapa y Eliminar localizaciones. 5. Comprobar el contenido de la primera localización. 6. Seleccionar la fecha 05/MM/2021 y comprobar que se muestra de nuevo la etiqueta de lista vacía.	* Localizaciones del fichero del positivo.	Se muestran las localizaciones registradas en el día 04/10/2021 y al volver a seleccionar el día 05/10/2021 se vacía la lista mostrándose la etiqueta de lista vacía.
H3	1. Ir a la pantalla del histórico de localizaciones. 2. Seleccionar en el campo de fechas la fecha 04/MM/2021 3. Comprobar la existencia de los botones de Mapa y Eliminar localizaciones. 4. Pulsar sobre el botón para Eliminar todas las localizaciones de ese día. 5. Comprobar que se vacía la lista y se muestra la etiqueta de lista vacía. 6. Cambiar de día y volver al día 04/MM/2021 para comprobar que se han eliminado correctamente.	* Localizaciones del fichero del positivo.	Se muestran las localizaciones registradas en el día 04/10/2021 y al pulsar sobre el botón de Eliminar se vacía la lista mostrándose la etiqueta de lista vacía.

8.6.4.5 *Política de privacidad*

En este caso, las pruebas solo se limitan a probar la presencia de los textos y títulos correspondientes en la pantalla de la política de privacidad, por lo que no existen pasos previos necesario para ejecutar las pruebas.

La siguiente tabla muestra el único caso de prueba diseñado para probar la pantalla de la política de privacidad.

Tabla 8.48. Casos de prueba para la pantalla de la política de privacidad.

Política de privacidad			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
PP1	1. Pulsar sobre el menú de la barra superior (tres puntos). 2. Comprobar que aparece la opción de Política de Privacidad. 3. Pulsar sobre la opción de Política de Privacidad. 4. Comprobar que aparecen los textos esperados, haciendo scroll hacia abajo cuando sea necesario.		Se muestra la Política de Privacidad de la aplicación, con las diferentes cláusulas.

8.6.4.6 *Ajustes*

El último caso de uso que se somete bajo pruebas del sistema es la configuración y los ajustes generales de la aplicación. El procedimiento previo está compuesto por los siguientes pasos:

1. Entrar en los detalles de la aplicación desde el dispositivo.
2. Borrar Datos de la Aplicación para que se restauren los valores de las *SharedPreferences*.

A continuación, se muestra una tabla con los casos de prueba diseñados para los ajustes de la aplicación.

Tabla 8.49. Casos de prueba para la pantalla de ajustes de la aplicación.

Ajustes generales de la aplicación			
Cod.	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**
A1	1. Abrir el menú de la barra superior (tres puntos). 2. Pulsar sobre la opción de Ajustes. 3. Comprobar el texto y los valores de la configuración según los datos de entrada.	* Configuración establecida: - Intervalo de tiempo: 3 sec - Despl. Mínimo: 0 m - Alcance de la comprobación: 3 días - Notificaciones de positivos: false	Se muestra la pantalla de ajustes de la aplicación. Los textos y títulos de los parámetros de configuración son correctos y sus valores se corresponden con la configuración establecida.
A2	1. Ir a los ajustes de la aplicación. 2. Cambiar el intervalo de tiempo y comprobar que se ha cambiado correctamente. 3. Cambiar el desplazamiento mínimo y comprobar que se ha cambiado correctamente. 4. Cambiar el alcance de la comprobación y comprobar que se ha cambiado correctamente. 5. Modificar la preferencia de envío de notificaciones y comprobar que se ha cambiado correctamente.	* Configuración establecida: - Intervalo de tiempo: 3 sec - Despl. Mínimo: 0 m - Alcance de la comprobación: 3 días - Notificaciones de positivos: false * Nueva configuración: - Intervalo de tiempo: 2 min 5 sec - Despl Mínimo: 3 m - Alcance: 1 día - Notificaciones de positivos: true	Se muestra la pantalla de ajustes de la aplicación. Al cambiar las preferencias, se actualizan correctamente los valores de las preferencias en las vistas.

8.6.5 Pruebas de Usabilidad

Las pruebas de usabilidad tienen como objetivo verificar el comportamiento del sistema en un entorno real con los usuarios finales, de forma que estos interactúen con el sistema y lo evalúen desde su punto de vista. Esto permite obtener retroalimentación de los usuarios finales acerca de los aspectos positivos y negativos del sistema, y así tratar de obtener una **afinidad** entre todas las opiniones de los usuarios y el sistema final.

Para llevar a cabo estas pruebas, se selecciona un grupo de usuarios de distintas edades para responder a unas cuestiones y realizar unas operaciones determinadas con el sistema. Este grupo de usuarios representa la **población** a la que está orientada la aplicación móvil y el panel de control *web*.

En primer lugar, se presentan una serie de **preguntas de carácter general** que sirven para determinar el nivel de destreza informático de los usuarios. En segundo lugar, se diseñan unas **actividades guiadas** compuestas por operaciones que los usuarios deben realizar con la aplicación móvil y el panel *web*. Por último, se definen unas **preguntas cortas** relativas a la usabilidad del sistema y otros aspectos relacionados, incluyendo un apartado de **observaciones**.

8.6.5.1 *Preguntas de carácter general*

Las preguntas de carácter general están orientadas a determinar cuál es el nivel de conocimiento que tienen los usuarios sobre la tecnología y la informática, además de comprobar la frecuencia de uso de estas tecnologías. A continuación, se muestra el cuestionario de preguntas de carácter general diseñado para la aplicación *Contact Tracker*.

Tabla 8.50. Preguntas de carácter general para las pruebas de usabilidad de Contact Tracker.

¿Usa un dispositivo móvil frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Qué tipo de actividades realiza con el dispositivo móvil?
1. Es parte de mi trabajo o profesión 2. Lo uso principalmente para ocio (juegos, música…) 3. Lo uso principalmente para las redes sociales y mensajería instantánea 4. Únicamente leo el correo y navego ocasionalmente por *internet*
¿Ha usado alguna vez software para el rastreo de COVID-19 (Ej. *Radar Covid*)?
1. Sí, he utilizado *Radar Covid* 2. Sí, he utilizado un software similar pero no *Radar Covid* 3. No, pero me lo he planteado 4. No, nunca
¿Qué busca Vd. Principalmente en una aplicación móvil para el rastreo de COVID-19?
1. Que sea intuitiva y fácil de usar 2. Que sea muy rápida, precisa y exhaustiva detectando contactos de riesgo a costa de un mayor consumo de batería 3. Que consuma menos batería a costa de ser menos precisa y exhaustiva detectando contactos de riesgo 4. Que respete lo máximo posible mi privacidad

8.6.5.2 *Actividades guiadas*

Las actividades guiadas son una serie de operaciones propuestas para realizar con el sistema. Estas operaciones permiten evaluar las interacciones de los usuarios con la aplicación móvil y el panel de control *web*, de forma que se pueda medir la usabilidad de estas operaciones. Estas actividades son distintas en función del tipo de usuario, distinguiendo entre **usuario estándar** y **administrador**.

8.6.5.2.1 Usuario estándar

Rastrear la ubicación de forma periódica:

1. Iniciar el servicio de rastreo automático.
2. Visualizar las ubicaciones que están siendo registradas en tiempo real en la lista de ubicaciones y en la pestaña de Mapa.
3. Cambiar la capa del mapa para ver el mapa con satélite y desactivar los marcadores.
4. Cerrar la aplicación por completo y bloquear el móvil comprobando que el servicio de rastreo está activo (aparece una notificación).
5. Volver a encender la aplicación y comprobar que se siguen registrando las ubicaciones.
6. Ver el histórico de ubicaciones.
7. Ver las ubicaciones registradas en el mapa.
8. Eliminar las ubicaciones registradas en el día de hoy.
9. Cambiar el intervalo de tiempo a 10 segundos entre el registro de una ubicación y otra en los ajustes.
10. Volver a activar el servicio de ubicación comprobando que las ubicaciones se registran aproximadamente cada 10 segundos.

Programar una alarma de rastreo de ubicación:

1. Ir a la pestaña de alarmas dentro del rastreo.
2. Programar una alarma para rastrear la ubicación a una hora determinada y seleccionar la hora de fin para terminar de registrar las ubicaciones.
3. Comprobar que a la hora de inicio establecida se comienzan a registrar las ubicaciones.
4. Programar otra alarma de rastreo y desactivarla, comprobando que no se activa el rastreo a la hora de inicio establecida.

Notificar un positivo:

1. Registrar algunas ubicaciones con el servicio de rastreo.
2. Visualizar la política de privacidad.
3. Notificar que el usuario ha dado positivo adjuntando sus datos personales.
4. Rellenar el formulario con los datos personales y aceptar la política de privacidad.

Realizar una comprobación de contactos de riesgo:

1. Cambiar el alcance de la comprobación en los ajustes.
2. Realizar una comprobación manual para detectar contactos de riesgo.
3. Visualizar el resultado de la comprobación.
4. Ir a los detalles de un resultado.
5. Ver los valores medios del resultado.
6. Ver los valores calculados de los contactos de riesgo que han sido detectados.
7. Visualizar en el mapa alguno de los contactos de riesgo detectados.

Programar una alarma de comprobación de contactos:

1. Programar una comprobación de contactos todos los días a una hora determinada.
2. Comprobar que a dicha hora se ejecuta la comprobación de contactos.
3. Eliminar la alarma de comprobación.

8.6.5.2.2 Administrador (personal sanitario)

Modificar la configuración del sistema:

1. Cambiar el periodo de infectividad para que pase a ser de 5 días.
2. Cambiar el tiempo de espera a 7 días para poder notificar otro positivo.
3. Cambiar la hora a la que se envían las notificaciones de positivos para que sea a las 15:00.
4. Modificar los parámetros de la comprobación de contactos.

Ver las estadísticas del sistema:

1. Visualizar las estadísticas desde los últimos 3 días hasta hoy.
2. Visualizar las estadísticas desde la última semana hasta hoy.

Ver los positivos notificados:

1. Visualizar los positivos que han sido notificados.
2. Ver los datos personales de un positivo que tenga asociados datos personales.
3. Ver el itinerario seguido por un positivo en un mapa.

8.6.5.3 *Preguntas cortas sobre la aplicación y Observaciones*

Una vez la población de usuarios ha experimentado con el sistema mediante las interacciones guiadas definidas en el apartado anterior, se proponen una serie de preguntas cortas sobre algunos aspectos del sistema para que los usuarios evalúen su experiencia con el mismo. Además, también se proporciona una sección de observaciones para que aporten cualquier sugerencia de mejora. A continuación, se muestra la plantilla con el cuestionario de preguntas cortas para el sistema de *Contact Tracker*, una para la aplicación móvil y otra para el panel de control *web*.

Tabla 8.51. Cuestionario de preguntas cortas y observaciones para la aplicación móvil.

Facilidad de Uso de la Aplicación Móvil	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad de la Aplicación Móvil	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				

Calidad de la Interfaz de la aplicación móvil				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				
Los iconos e imágenes usados son				
Los colores empleados son				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que la aplicación está bien estructurada?				
Observaciones				
Cualquier comentario del usuario				

Tabla 8.52. Cuestionario de preguntas cortas y observaciones para el panel de control web.

Facilidad de Uso del panel de control *Web*	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad del panel de control *Web*	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
Calidad de la Interfaz del panel de control *web*				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				
Los iconos e imágenes usados son				
Los colores empleados son				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que la aplicación está bien estructurada?				
Observaciones				
Cualquier comentario del usuario				

Capítulo 9. Implementación del Sistema

En los dos anteriores capítulos, se desarrolló el análisis y diseño del sistema para partir de una base que definiera **qué** hay que construir y **cómo** se debe construir. En este capítulo, partiendo de esa base se pretende resumir los principales aspectos de la propia implementación de los subsistemas identificados, los cuales conforman el sistema en su totalidad.

9.1 Estándares y Normas Seguidos

En esta primera sección, se enumeran y describen brevemente aquellos estándares, normas o leyes que se hayan utilizado para guiar el desarrollo del sistema, así como para llevar a cabo las tareas transversales de las distintas fases del proyecto como tal. Para presentarlos de la manera más visual posible, a continuación, se muestra una tabla que recoge estos estándares, leyes y normas contemplados en la ejecución del proyecto, indicando para cada uno de ellos, su nombre, una breve descripción junto con su aplicación dentro de este proyecto, y si se trata de un estándar, una norma o una ley.

Tabla 9.1. Estándares, normas, leyes y reglamentos utilizados como guía en la construcción del sistema.

Nombre	Tipo	Descripción
LOPD/GDD (Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y Garantía de los Derechos Digitales)	Ley (Normativo)	Ley orgánica puesta en vigor en España en diciembre de 2018 resultado de la especialización del reglamento europeo de protección de datos. Contiene la normativa y las cláusulas relativas a la protección de datos personales en los medios digitales incluyendo una garantía de los derechos digitales. Esta ley se tiene en cuenta a la hora de llevar a cabo el **tratamiento** de los datos personales dentro de la aplicación móvil y solicitar el **consentimiento** del interesado para llevar a cabo dicho tratamiento.
RGPD (UE) 2016/679 del Parlamento Europeo y del Consejo	Reglamento (Normativo)	Reglamento europeo relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales. Este reglamento junto con la LOPD se toma como referencia para desarrollar las cláusulas de la **Política de Privacidad** del sistema, accesible desde la aplicación móvil.
IEEE Std. 830-1998	Estándar	Estándar para la especificación de requisitos *software* funcionales y no funcionales de un sistema informático. En el contexto de este sistema, no se ha aplicado dicho estándar de manera estricta, sino que se ha utilizado como una **guía** de referencia para especificar algunos requisitos en el capítulo dedicado al **análisis** del sistema.
Scrum	Marco de trabajo (metodología)	La metodología empleada para guiar los procesos de desarrollo del proyecto es ***Scrum***, que, pese a ser clasificado como un marco de trabajo, puede incluirse en este apartado ya que establece una serie de pautas y

		estándares a seguir para llevar a cabo un desarrollo ágil del producto.
Material Design	Estándar	Estándar para el **diseño** de **interfaces** de usuario que proporciona una serie de guías y pautas para estructurar y representar los componentes de las interfaces de usuario, así como la forma en la que los usuarios interaccionan con la interfaz. Para este proyecto, se ha seguido el estándar *Material Design* tanto para la **aplicación móvil** *Android* como para la **aplicación *web*** desarrollada con *Vue.js.*

9.2 Lenguajes de Programación

Aquí se presentan los diferentes lenguajes de programación, junto con su versión, utilizados para llevar a cabo la codificación de los distintos módulos que componen el sistema, incluyendo además los *frameworks* y tecnologías complementarias usadas para guiar el desarrollo. Algunas de estas tecnologías y herramientas ya fueron contempladas en el apartado 3.3. HERRAMIENTAS Y PLATAFORMAS UTILIZADAS de modo que en esta sección solo se describirán los lenguajes de programación como tal y algunos de sus complementos y *frameworks*.

9.2.1 *Kotlin (v1.5.31)*

Kotlin es un lenguaje de tipado **estático** con soporte para el paradigma de **orientación** a **objetos** y también **funcional** desarrollado por *JetBrains* y administrado por ***Kotlin Foundation***, una entidad formada por *JetBrains* y *Google*. Está orientado a ejecutarse sobre una máquina virtual de *Java*, ya que su código puede compilarse a *Java* o incluso a *JavaScript* para ser ejecutado. En este proyecto, *Kotlin* se utiliza en el entorno de desarrollo de ***Android*** como sustituto de *Java,* aunque realmente por debajo se transforma a instrucciones *Java*. De este modo, permite la **interoperabilidad** con *Java* y viceversa.

Para trabajar en *Android* con *Kotlin* se hace uso de la ***API Android KTX,*** que contiene extensiones y *plugins* de *kotlin* para trabajar con las distintas *APIs* del *framework* de *Android*.

La principal razón por la que se ha escogido *Kotlin* en lugar de *Java* es la migración progresiva de las aplicaciones *Android* hacia este nuevo lenguaje, más moderno y flexible, que aglutina características muy interesantes de otros lenguajes como *C#*. Así, el lenguaje *Java* para desarrollar aplicaciones *Android* está quedando cada vez más obsoleto, siendo *Kotlin* el sucesor perfecto que proporciona características muy útiles a la hora de implementar aplicaciones *Android*. Otras razones por las que se ha escogido este lenguaje son:

- Codificación de código asíncrono mediante las corrutinas y las funciones *suspend*.
- Azúcares sintácticos, como las *sealed class*, constructores simplificados, gestión de la nulidad de las propiedades, delegados, concatenación de *strings* y variables…
- Funciones de extensión y operadores.

9.2.2 SQL (*ROOM* v2.3.0)

El lenguaje *SQL* se utiliza en las **consultas** de la base de datos **local** de los dispositivos que tengan una versión oficial de la aplicación móvil de *Contact Tracker*. Este lenguaje específico del dominio permite definir las operaciones de inserción, borrado, actualización y consulta para las entidades de la base de datos ***SQLite*** integrada en la aplicación. Para trabajar con *SQL* se utiliza la librería ***ROOM*** de *Google* que constituye una capa intermedia de abstracción para desentenderse de las peculiaridades de bajo nivel para definir el esquema y la conexión con la base de datos.

Las consultas *SQL* se definen de manera declarativa a través de **anotaciones** en el código que después son implementadas por la propia librería de *ROOM*. Esta además permite definir las **restricciones** de **integridad** y las relaciones entre las tablas directamente a través de las clases planas que representan las entidades del dominio.

9.2.3 *Gradle (v6.7.1)*

Gradle es un lenguaje específico del dominio o ***DSL*** que permite definir los aspectos de la configuración de un proyecto a través de *scripts* automatizados. *Gradle* se utiliza como gestor de paquetes y dependencias en el proyecto ***Android*** dentro del *IDE* de ***Android Studio***. De este modo, permite automatizar las tareas de **construcción** y **empaquetado** del código a través de su lenguaje integrado, facilitando además la gestión de las versiones de las dependencias del proyecto.

Para integrarse con el *framework* de *Android*, se utiliza el ***plugin*** de *Gradle* para *Android*, que en este caso se corresponde con la versión ***4.2.1***.

9.2.4 *Javascript (ECMAScript 2016)*

JavaScript es un lenguaje diseñado para la ***web*** con el objetivo de dotar de dinamismo y actividad a los ficheros estáticos *html* y *css*. Se trata de una de las diversas **implementaciones** del **estándar *ECMASCRIPT***, es decir, es uno de sus dialectos. Actualmente, la gran mayoría de navegadores de *internet* soportan completamente la versión ***ECMAScript* 2016.** Existen versiones más recientes de este estándar, aunque no todos los navegadores las soportan.

Se trata de un lenguaje de tipado **dinámico**, es decir, no hay comprobación de tipos, además es un lenguaje **interpretado**, de manera que cada línea de código se va interpretando y ejecutando sobre la marcha, en lugar de compilar los ficheros previamente para generar el código ejecutable. Además, también soporta el paradigma orientado a objetos desde *ECMAScript 2015*, cuando se introdujeron las **clases** como una evolución basada en los prototipos.

En el contexto de este proyecto, el lenguaje de *JavaScript* se encuentra esparcido por multitud de *frameworks* y herramientas, como las que se describen en las siguientes subsecciones.

9.2.4.1 *Node.js (v14.15.5)*

Node.js es un entorno de ejecución basado en *JavaScript* y que permite definir la parte de **servidor** de una aplicación *web*. Está construido sobre el motor *V8* de *Google* y sigue una arquitectura basada en **eventos** con un solo hilo encargado de gestionar las operaciones de manera **asíncrona**. En el contexto de este proyecto, *Node.js* es utilizado como la base para la capa ***backend*** que da soporte al **servidor *web***, trabajando en armonía con el *framework* de *Express* que se describe en la siguiente sección.

Esta herramienta permite escribir código *JavaScript* que forme parte del servidor, sin necesidad de utilizar otros lenguajes de tipado fuerte como *Java*.

9.2.4.2 *Express.js (v4.17.1)*

Express.js es un *framework* pensado para trabajar con *Node.js*, escrito en *JavaScript* y que permite trabajar fácilmente con las peticiones de red y las respuestas a los clientes desde un entorno de servidor. *Express.js* se utiliza junto con el entorno de ejecución de *Node.js* para escuchar las peticiones de los clientes *web* y móvil del contexto de este proyecto. De este modo, permite gestionar las **rutas** expuestas por la ***API REST*** del servidor *web* y redirigir el tráfico entrante y saliente de las peticiones de red.

En este proyecto, se utiliza *Express* para definir el *router* que gestiona el enrutado del servidor *web* y así trabajar fácilmente con las peticiones y sus parámetros.

9.2.4.3 *Firebase Admin SDK (v.9.6.0)*

Para poder acceder a los servicios de la plataforma de ***Firebase***, se hace uso de una dependencia de *Node.js* para trabajar con las *APIs* de esta plataforma. Concretamente, se utiliza el ***SDK Admin*** de *Firebase* el cual permite trabajar con las funcionalidades de *Firebase* desde el entorno de servidor *node.js*.

En este proyecto, el *SDK* de *Firebase* se instala en el proyecto *Node.js* para poder trabajar con la base de datos documental en la nube, es decir, con ***Firestore***, y también para acceder a la *API* de ***FCM (Firebase Cloud Messaging)*** que permite enviar mensajes *push* a los clientes *Android*.

9.2.4.4 *Vue.js (v2.6.11)*

Para desarrollar la parte del *frontend* del sistema, es decir, el panel de control *web,* también se utiliza una tecnología basada en *JavaScript*, en este caso el *framework* de ***Vue.js***. Mediante este *framework*, se construyen los componentes *web* de la interfaz de la aplicación que se conecta con el servidor *web* a modo de panel de control para los administradores. Para ello, el código *javascript*, el marcado *html* y los estilos *css* se encapsulan en unas entidades denominadas **componentes**, de forma que estos contienen tanto el comportamiento, como la estructura y los estilos de la interfaz de usuario. Agrupando varios de estos componentes se van construyendo las vistas que conforman la aplicación *web*.

Como ya se mencionó, se utiliza ***HTML5*** y ***CSS3*** dentro de estos componentes para definir el marcado y los estilos de los elementos *html* que conforman el componente, aunque también se hace uso de una librería de interfaz de usuario denominada ***Vuetify*** que será descrita más adelante.

9.3 Herramientas y Programas Usados para el Desarrollo

El apartado anterior recoge todos los lenguajes de programación y derivados empleados en la codificación del sistema, mientras que en esta sección se recopilan aquellos **programas**, **herramientas**, **plataformas**, **entornos de desarrollo** y cualquier otra utilidad significativa que se haya utilizado tanto para la codificación del sistema como para el resto de tareas transversales relacionadas con el desarrollo, como puede ser la planificación y el despliegue del sistema.

Es importante destacar que en la sección 3.3 HERRAMIENTAS Y PLATAFORMAS UTILIZADAS ya se han cubierto gran cantidad de estas herramientas y por ello no se entrará en excesivo nivel de detalle en esta sección. Además, el objetivo de esta sección es describir los programas y herramientas de alto nivel involucradas en el desarrollo, con lo cual **no** se describirá la multitud de **dependencias** y **librerías** utilizadas, pues además es un número bastante elevado y carece de sentido exponerlas todas aquí. Muchas de estas librerías también fueron cubiertas en el capítulo 3 dedicado a los aspectos teóricos del proyecto.

9.3.1 *Android Studio* (v2020.3.1)

El entorno de desarrollo de *Android* por excelencia es ***Android Studio***, y es el que se ha utilizado para desarrollar y probar el subsistema más complejo e importante del presente proyecto: la aplicación móvil de rastreo. Este *IDE* fue desarrollado por *Google* y constituye el entorno de desarrollo oficial para la plataforma de *Android*. *Android Studio* da soporte tanto al desarrollo mediante el lenguaje *Java* como el lenguaje *Kotlin*. Este último, como ya se mencionó, es el utilizado en el contexto del proyecto. Además, contiene multitud de herramientas como el **editor** de ***layouts XML***, visualización de recursos, **depuración** de código, visualización de la base de datos interna de un dispositivo, inspector de *layout* de una vista concreta en ejecución, explorador de archivos de los dispositivos y gestor de **emuladores** *Android* para ejecutar las aplicaciones, entre otras funcionalidades.

Dentro de *Android Studio* residen el resto de las tecnologías y *APIs* de *Android* a las que se accede a través del código de *Kotlin*, algunas de las más importantes se describen en las siguientes subsecciones.

9.3.1.1 *Hilt (v2.33)*

Hilt **[54]** es una librería de **inyección de dependencias** que facilita la implementación de este patrón de diseño en la estructura interna de clases de la aplicación móvil. *Hilt* está desarrollado y mantenido por *Google* y permite definir de manera declarativa las

dependencias que utilizan las distintas clases de código, de manera que estas se inyecten automáticamente en tiempo de ejecución. *Hilt* gestiona por sí mismo el **árbol de dependencias** a través de un motor interno que representa las clases y sus dependencias mediante un grafo. De esta forma, *Hilt* conoce exactamente qué clases hay que inyectar y cómo se construyen esas clases para simplificar el código necesario para construir e inicializar los objetos utilizados en la aplicación.

9.3.1.2 *Retrofit (v2.9.0)*

Retrofit es una librería para *Android* que permite realizar peticiones ***http*** a una ***API*** remota para consumir sus recursos. Se trata de una capa de abstracción sobre el código plantilla que se utiliza para establecer conexión con un servidor por medio del protocolo *http*. De este modo, todo el código necesario para entablar la conexión, construir las peticiones, enviarlas y procesar las respuestas obtenidas se simplifica enormemente gracias a esta librería, la cual permite definir de manera **declarativa** las peticiones y convertir las respuestas a objetos del dominio automáticamente.

En la aplicación móvil de rastreo, se utiliza *Retrofit* junto con la librería ***GSON v2.8.6***, la cual gestiona la transformación de los objetos *JSON* a entidades del dominio tanto en el envío de las peticiones como en la recepción de las respuestas. Así, todas las peticiones salientes de la aplicación móvil que se lanzan contra el servidor *web* se realizan a través de *Retrofit*, integrado con *GSON* para procesar las respuestas del servidor.

9.3.1.3 *Jetpack Navigation (v2.3.5)*

Jetpack Navigation es una librería de la *suite* de bibliotecas de ***Android Jetpack***, la cual proporciona una serie de componentes que facilitan la implementación de la **navegación** entre los fragmentos de la aplicación móvil. Con *Jetpack Navigation*, el flujo de navegación entre la jerarquía de fragmentos que componen la aplicación se modela por medio de un **grafo de navegación**, en donde los nodos representan los fragmentos destino y los arcos las **acciones** que pueden llevarse a cabo para navegar de un nodo a otro, incluyendo los posibles **argumentos** asociados a dichas acciones.

La navegación de la aplicación móvil de rastreo está definida por medio de un grafo de navegación el cual está vinculado a un componente *NavHost* que es el que hospeda los distintos fragmentos que se van mostrando al usuario. Este componente trabaja junto con un ***NavController*** que actúa de controlador para gestionar las acciones que harán que se sustituya un fragmento por otro en el contenedor principal de la aplicación.

9.3.1.4 *Google Play Services Location (18.0.0)*

Los servicios de **localización** de *Google Play* son uno de los muchos servicios que proporciona esta plataforma a los desarrolladores de *Android* para integrar distintas funcionalidades en las aplicaciones móviles. En este proyecto relacionado con la geolocalización, los servicios de localización constituyen una dependencia vital para poder obtener información sobre la posición geográfica del dispositivo móvil, es decir, es la **fuente** de información a partir de la cual se obtienen las actualizaciones de coordenadas del dispositivo. Esta dependencia es

necesaria en el caso de que se utilice el ***fused provider*** descrito en la sección 3.2.2.2.2 FUSED LOCATION PROVIDER API en el capítulo de Aspectos Teóricos, pues si se hace uso de la otra alternativa del *framework* de *Android*, el ***location manager***, no es necesario incluir esta librería. Además, es fundamental que los dispositivos tengan acceso a los servicios de ***Google Play*** dado que se trata de un servicio proporcionado por esta plataforma, lo cual constituye un desventaja frente al *manager* de localización integrado en el *framework* de *Android*, como ya se comentó en su momento en el capítulo 3.

9.3.1.5 *ROOM (v2.3.0)*

ROOM es la librería de *Google* para realizar las operaciones de persistencia sobre una base de datos *SQLite* en las aplicaciones móviles *Android*. Como ya se comentó, consiste en una capa intermedia que simplifica el código para conectarse a la base de datos integrada en los dispositivos y definir el esquema compuesto por las tablas, junto con sus relaciones y restricciones de integridad. En este proyecto, *ROOM* se utiliza en la aplicación móvil para implementar las consultas que recuperan los datos almacenados en local, así como para insertar, actualiza o eliminar dichos datos. El lenguaje específico de dominio que se utiliza para definir las consultas es el clásico ***SQL***, que es procesado en tiempo de compilación por *ROOM*.

9.3.1.6 *Material Design (v1.4.0)*

Para seguir el estándar de *Material Design,* definido en el apartado 9.1 ESTÁNDARES Y NORMAS SEGUIDOS*,* sobre las interfaces de usuario en la aplicación móvil *Android*, se hace uso de la librería de *Google* ***Material Design***, la cual proporciona diversos *widgets* y componentes de interfaz de usuario que siguen las pautas establecidas por este estándar. Gracias a esta librería, se garantiza que la interfaz de usuario resultante de la aplicación móvil cumpla con los estándares definidos.

9.3.2 *Visual Studio Code (v1.61.2)*

Visual Studio Code es el *IDE* o entorno de desarrollo gratuito creado por *Microsoft* para dar soporte a multitud de lenguajes y *frameworks*. Incluye depuración de código, resaltado de sintaxis e integración con el control de versiones de *Git*, además de la capacidad de ampliar la funcionalidad por medio de ***plugins*** y **extensiones**. Para el caso de este proyecto, *Visual Studio Code* se ha utilizado para desarrollar tanto el **servidor *web*** de *node.js* como la **aplicación *web*** mediante el *framework* de ***Vue.js***. Además, se ha integrado con las consolas de *node* para ejecutar las aplicaciones en el entorno de desarrollo, así como para ejecutar las pruebas de la *API REST*.

9.3.3 GitHub (Git v2.28.0)

Como sistema de control de versiones (*VCS*) se ha empleado ***Git*** junto con los repositorios en remoto hospedados por la plataforma de ***GitHub***. *Git* es un *software* de control de versiones **descentralizado** que permite gestionar los diferentes estados de las clases de código de un proyecto, de forma que se mantenga un **histórico** de todos los cambios realizados en los

distintos ficheros de código fuente. *Git* es descentralizado porque permite gestionar los cambios en cada equipo **local** creando copias del repositorio remoto distribuidas entre los distintos clientes que cooperan en el código, de forma que antes de publicar los cambios definitivos en el repositorio remoto, primero se trabaja en el repositorio local.

Para el caso de este proyecto, se ha utilizado una licencia *Pro* de *GitHub* para albergar el código de los tres subsistemas en repositorios remotos en la nube, integrando *Git* con *Android Studio* y *Visual Studio Code* de manera que se puedan gestionar estos repositorios, así como los cambios en el código desde el equipo local de trabajo.

9.3.4 *Firebase (Admin SDK v9.6.0)*

Firebase es una plataforma en la nube desarrollada y gestionada por *Google* y que se encuentra fuertemente ligada a la consola de *Google Cloud*. *Firebase* proporciona servicios en la nube para aplicaciones móviles de *Android* e *iOS*, así como para aplicaciones *web* escritas en *JavaScript*. En el contexto de este proyecto, *Firebase* se utiliza en el **servidor *web*** para acceder a la base de datos documental ***Firestore*** y así poder almacenar y leer los datos del sistema, además de trabajar con ***Firebase Cloud Messaging*****,** otro de sus servicios para enviar mensajes *push* a los clientes *web* y móviles.

Para poder trabajar con *Firebase* desde el entorno de *node.js*, se utiliza el ***Firebase Admin SDK*** que se instala como una dependencia y se configura a través de un fichero *JSON* generado por la plataforma a modo de autenticación. Para obtener este fichero, es necesario crear previamente un proyecto en *Firebase* que se vincule con los subsistemas de la aplicación móvil y el servidor *web*.

9.3.5 *Azure DevOps* y *Azure Portal*

La plataforma en la nube de *Azure* de *Microsoft* proporciona una gran cantidad de funcionalidades y servicios para distintos tipos de aplicaciones. Por un lado, se encuentra ***Azure DevOps*** que es una *suite* de funcionalidades enmarcadas en una metodología concreta, en este caso ***Scrum***, y por otro está el **portal** *Azure* que alberga diversos servicios en la nube como *hosting* u otros.

En este proyecto, *Azure DevOps* se utiliza como un **panel de control** para gestionar la planificación del proyecto y sus entregables, además de visualizar su **evolución** temporal mediante diagramas de quemado. En este panel también se agrupan las propias **tareas** del proyecto en formato de **historias de usuario**, de modo que la especificación del sistema está registrada en esta plataforma. El **despliegue continuo** también se configura a través de *Azure DevOps* gracias al uso de las ***pipelines*** que permiten configurar flujos de operaciones de empaquetado, *testing* y despliegue de manera automática cada vez que se realiza un *push* a una rama del repositorio. Esto implica que los repositorios de *GitHub* estén vinculados con la plataforma de *Azure DevOps*.

Por otro lado, en el portal de *Azure* se encuentra el ***Azure App Service*** sobre el cual se despliega el servidor *web* de *Contact Tracker* y la **cuenta** de **almacenamiento estático** donde residen los recursos de la aplicación *web* hospedada en la nube. De este modo, tanto la

aplicación *web* como la *API REST* serán accesibles en remoto desde cualquier equipo a través de *internet*.

9.3.6 *Google Maps* (GCC)

La consola de *Google Cloud* permite gestionar multitud de servicios en la nube proporcionados por *Google* para distintos tipos de aplicaciones. En este caso, esta consola se utiliza solo para consumir los servicios de los ***mapas*** de ***Google***, por medio de distintas ***API KEYs*** que se utilizan en la aplicación móvil *Android* y en la aplicación *web* de *Vue.js*. Para ello, se ha creado un proyecto de *GCC* donde se ha contratado el servicio gratuito de mapas para mostrar de manera visual los itinerarios de los positivos, así como los contactos de riesgo detectados mediante la aplicación móvil de rastreo.

Para la aplicación móvil, la versión del ***SDK*** de *Google Maps* para *Android* que se utiliza actualmente es la **v18.0.0**, mientras que para la aplicación *web Vue*, se hace uso de la versión ***v0.10.7*** de la librería ***vue2-google-maps***.

9.3.7 *Vuetify* (v2.4.0)

Para cumplir con las pautas definidas en el estándar de ***Material Design***, la interfaz de usuario de la **aplicación *web Vue*** se ha construido utilizando la librería ***Vuetify***, la cual proporciona distintos tipos de componentes y *widgets* de interfaz de usuario que siguen el estándar de *Material* y que ya vienen preparados para ser utilizados e integrados en el *framework* de *Vue*, en forma de componentes. *Vuetify* permite construir infraestructuras *web* elegantes de manera sencilla y rápida, sin entrar en detalles de bajo nivel de implementación que distraen el flujo de trabajo de las operaciones de negocio prioritarias.

De nuevo, hay que recalcar que este listado de herramientas y programas utilizados en el desarrollo no pretende ser exhaustivo y cubrir por completo todo el amplio abanico de dependencias del proyecto, sino que más bien trata de mostrar aquellas más importantes y que mayor impacto han tenido a la hora de construir los diferentes subsistemas.

9.4 Creación del Sistema

Esta última sección de la implementación del sistema describe los aspectos más relevantes de la creación y construcción de los diferentes subsistemas basándose en el **diseño** previo presentado en el capítulo anterior. En primer lugar, se relatan los principales problemas encontrados durante el desarrollo y cómo fueron solventados. En segundo lugar, se describe el funcionamiento interno de alguno de los aspectos más interesantes del sistema. Por último, se presenta una descripción detallada de las principales clases de la aplicación móvil.

9.4.1 Problemas Encontrados

Las siguientes subsecciones describen los principales problemas que se fueron encontrando durante la construcción del sistema.

9.4.1.1 *Rastreo de ubicación en segundo plano*

Uno de los principales retos del sistema fue lograr que el rastreo de las coordenadas del usuario funcionase de manera constante y consistente, es decir, el objetivo era que se registrasen las localizaciones del usuario de manera uniforme incluso con la aplicación en segundo plano, cerrada o con el móvil bloqueado. Además, el rastreo debía ser consistente en el sentido de que las localizaciones se registraran periódicamente con un intervalo uniforme de tiempo lo más aproximado posible.

Con las nuevas versiones de *Android* y sus niveles de *API*, conseguir ejecutar el rastreo de manera constante sin importar el estado de la aplicación y del dispositivo era cada vez más difícil, pues las **políticas** de **privacidad** y **seguridad** del sistema operativo se volvían más fuertes. En la sección 3.2.4 VERSIONES: RESTRICCIONES Y LIMITACIONES se describen estas restricciones impuestas por el equipo de *Google* que limitan las operaciones que se pueden realizar en segundo plano y con el dispositivo bloqueado.

La **solución** a este problema fue utilizar un componente de *Android* que fuera análogo a una *Activity* en el sentido de que se ejecutase en primer plano de forma que no fuese suspendido por el sistema operativo. Este componente es el **servicio en primer plano** o ***foreground service***, descrito en la 3.2.3 EJECUCIÓN DE TAREAS EN SEGUNDO PLANO, el cual permite ejecutar fragmentos de código con la *Activity* principal en segundo plano, apagada o incluso con el dispositivo bloqueado, todo ello a costa de mostrar una **notificación** al usuario indicando su **presencia** para que este sea consciente de que hay un servicio ejecutándose. De este modo, la lectura de la posición del dispositivo se mantiene constante a lo largo del tiempo sin apagarse hasta que el usuario lo decida, y además, conservando un intervalo de tiempo uniforme entre las actualizaciones de coordenadas.

9.4.1.2 *Acceso a la base de datos desde el hilo principal*

En *Android*, una de las restricciones más notables es la prohibición de ejecutar **consultas** a la base de datos en el **hilo principal** (*main*), dado que en este hilo se procesan las interacciones con la interfaz de usuario, con lo cual aquellas operaciones pesadas o **bloqueantes** podrían

hacer que se congelara la interfaz de usuario hasta que estas se completasen. Al implementar estas consultas en el mismo hilo que la interfaz de usuario, se producían excepciones en tiempo de ejecución. No solo las consultas deben invocarse en un hilo separado, sino que todas aquellos procesos pesados que puedan tardar un tiempo considerable en terminar también deberían trasladarse fuera del hilo principal.

La **solución** fue emplear las **corrutinas** de *kotlin* para ejecutar el código bloqueante y las llamadas a la base de datos local en hilos separados e independientes del hilo principal, a través del ***Dispatcher*** de ***IO***, el cual representa los hilos dedicados a operaciones pesadas de entrada y salida de datos.

9.4.1.3 *Movimiento del mapa de rastreo en tiempo real*

En la pestaña del mapa en tiempo real del rastreo, dentro de la opción del rastreo de ubicación, el usuario puede ver las localizaciones que están siendo registradas por el servicio de rastreo activo en ese momento, de manera que cada localización obtenida se dibuja en el mapa con un marcador. Este mapa se encuentra dentro de un *TabLayout* que tiene varias pestañas, por lo que al hacer el gesto de ***swipe***, el usuario navega a las otras pestañas colindantes con el mapa. El problema surge cuando el usuario intenta desplazarse y **navegar** por el mapa, pues al realizar los gestos con el dedo que le permiten moverse por el mapa, este permanece inmutable y los gestos son **confundidos** como eventos de *swipe*, lo cual se traduce en un cambio de pestaña.

Para **solucionar** este problema, se **redefine** el comportamiento de la vista del mapa ***MapView*** de manera que los eventos de pulsación recibidos sean procesados por el propio mapa y no por su contenedor **padre**, siempre y cuando se encuentren dentro de la zona central del mapa, lo cual significa que el usuario quiere navegar por el mapa. Si el evento de pulsación se detecta en los **extremos**, el usuario quiere navegar entre las pestañas y por tanto se cancela el evento en el mapa y se redirige al contenedor padre para que este lo procese. La siguiente figura muestra estas zonas en la pantalla del dispositivo de manera visual.

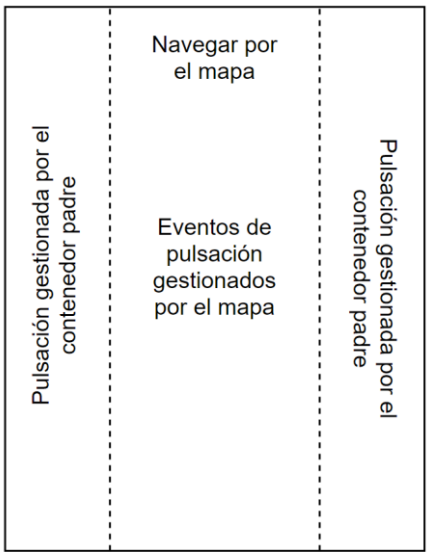


Figura 9.1. Zonas de gestión de los eventos de pulsación en la vista del mapa de rastreo en tiempo real.

9.4.1.4 *Timestamps almacenados en la base de datos central*

En un primer momento, los tiempos y fechas en la base de datos central se almacenaban como ***Strings*** a los cuales *Firebase* daba formato en función de la localización de los dispositivos. Sin embargo, al cabo del tiempo se detectó que almacenar las fechas como *Strings* formateados daba lugar a **inconsistencias** ya que algunos dispositivos de prueba (posiblemente mal configurados) subían los objetos fecha con distintas localizaciones a la de España, por lo que al leer las fechas de la base de datos las horas no coincidían con lo esperado debido al cambio de la zona horaria. Para **solucionar** esto, finalmente se optó por almacenar las fechas como milisegundos con formato ***Long*** en ***Firestore*** en lugar de *Strings* formateados, de forma que así se mantiene la consistencia entre zonas horarias pues siempre se contemplan los milisegundos. Así, los clientes procesan estos milisegundos para convertirlos en objetos de tipo fecha correspondientes a sus zonas horarias. Del mismo modo, las fechas de notificación de positivos que antes se almacenaban como ***timestamps***, también se pasaron a almacenar como valores de tipo *Long* que representan los milisegundos transcurridos desde 1970.

Estos milisegundos son procesados por los clientes para formatearlos y convertirlos a un objeto de tipo ***Date*** cuya zona horaria se corresponda con la zona horaria de los clientes.

9.4.1.5 *Problemas con los eventos de los LiveData*

Ya se explicó que los objetos *LiveData* son componentes observables que cada vez que son modificados notifican a sus observadores, quienes están a la escucha de estas actualizaciones. El problema es que los objetos *LiveData* están sujetos al **ciclo de vida** del componente *Android* al que están asociados, en la mayoría de los casos a los fragmentos a través de los *viewmodels*, por lo que cada vez que hay un cambio de configuración, las actualizaciones de los *LiveData* vuelven a dispararse, haciendo que los observadores reciban otro evento con cada cambio de configuración. Este comportamiento se observa, por ejemplo, cuando se muestra un *Snackbar* con un mensaje de error cada vez que se dispara un evento de un *LiveData* que representa dicho error. En este caso, al navegar a otro fragmento sin destruir el actual y volver atrás, el *Snackbar* se vuelve a mostrar porque el evento del *LiveData* se dispara con cada cambio de configuración.

La **solución** a este problema la proporciona el propio **equipo** de ***Google Android***, a través de un fragmento de código abierto que se ha convertido ya en un estándar en la comunidad de *Android* para solventar este problema que tienen los *LiveData*. Se trata de un *LiveData* modificado que recibe el nombre de ***SingleLiveEvent***. Funciona igual que un *LiveData* normal pero su evento solo se dispara una única vez, aunque haya múltiples cambios de configuración.

9.4.1.6 *Desfase temporal entre la hora local y el servidor desplegado en Azure*

Uno de los problemas más peculiares encontrados durante el desarrollo fue el desfase horario entre la hora local en España y la hora interna del servidor *Azure* desplegado en la nube, sobre el cual se ejecuta la *build* del servidor *web*. El empaquetado del servidor *web* se despliega sobre un servidor ***Linux*** gestionado por la plataforma de *Azure* que se encuentra desfasado **una hora** por **detrás** respecto de la hora local. Esto fue detectado debido a que, tras ejecutar

las pruebas de usabilidad con una serie de usuarios finales y el servidor desplegado en la nube, dejó de funcionar el envío de mensajes *FCM* y los *timestamps* de los positivos notificados no coincidían con la fecha actual. Tras volcar los ***logs*** del servidor, se vio que la fecha estaba atrasada una hora y esto era la causa de que no se enviaran los mensajes, ya que estos estaban programados para una hora después desde la perspectiva del servidor de *Azure*.

Para **solucionar** esto de la manera más rápida posible, se hizo uso de una **variable de entorno** llamada ***REMOTE_DEPLOY*** con el valor de "*Azure*" la cual se utiliza para distinguir entre un despliegue local, en la máquina utilizada para el desarrollo, y un despliegue en remoto, en este caso en los servidores de *Azure*. Dado que se trata de un servidor *Linux*, no es necesario utilizar la librería ***cross-env*** para definir las variables de entorno, sino que se pueden definir directamente en línea con el comando de inicio *npm start*. De este modo, en el código se utiliza un *if* para comprobar si se está en el entorno del servidor de *Azure* y así atrasar una hora la programación de las tareas síncronas para enviar las notificaciones, de manera que concuerde con la hora local.

Esto constituye una **solución temporal** que en un futuro debería revisarse para sincronizar la hora local con la del propio servidor. Para el caso de utilizar ***Heroku***, ocurre lo mismo, la fecha del sistema está desfasada una hora, así que también se aplica la variable de entorno ***REMOTE_DEPLOY="Heroku".***

9.4.1.7 *Superación del límite de recursos de Azure*

Otro de los problemas relacionados con la plataforma de *Azure* es que al utilizar un ***Free Tier***, es decir, un plan gratuito de servicios, existen diversas limitaciones en cuanto a la cantidad de recursos que se pueden consumir al día. Así, cuando se realizaron pruebas de la aplicación en producción, se detectó que el servidor *web* dejaba de funcionar debido a que se habían superado las **cuotas** de recursos.

La **solución** a este problema realmente fue dejar que se reiniciaran las cuotas y utilizar la plataforma de ***Heroku*** para desplegar el servidor *web* en la nube a modo de ***backup*** o **copia de seguridad*,*** por si se diera el caso de que se superaran los límites de recursos. Así, se disminuye el riesgo de fallo de **disponibilidad** del servidor *web* de *Contact Tracker* **escalando horizontalmente** los servicios de despliegue haciendo uso de *Azure App Service* y de los *Dynos* de *Heroku*.

9.4.1.8 *Base de datos en la nube para las pruebas de sistema*

Para poder ejecutar las pruebas del sistema destinadas a verificar el funcionamiento de la interfaz de usuario de la aplicación móvil, es necesario que la base de datos central y el servidor *web* estén disponibles, pues estos no se *mockean* para las pruebas. Esto tiene la desventaja de que los datos almacenados en la nube son destruidos y modificados como consecuencia de la ejecución de las pruebas. Por ello, se ha optado por dividir la base de datos *Firestore* en **dos partes,** una para las pruebas y otra para el entorno de producción. De este modo, las pruebas se ejecutarían utilizando la base de datos de prueba, mientras que al desplegar la aplicación real, el servidor *web* accedería a la base de datos en producción,

evitando así que se destruyan los datos reales. Para alternar entre una base de datos u otra se hace uso de las variables de entorno del servidor, lo cual se explica más adelante.

9.4.2 Algoritmo de comprobación

En el capítulo dedicado al diseño del sistema, se presentó la interfaz ***RiskContactDetector*** que definía el contrato entre las clases clientes y las implementaciones concretas del algoritmo. En esta sección se pretende explicar el funcionamiento de una de esas implementaciones concretas de este algoritmo, el cual **no** ha sido extraído de ninguna **fuente externa** sino que se trata de un **algoritmo de elaboración propia**. La interfaz del algoritmo contiene las cabeceras de los métodos que trabajan de manera conjunta para detectar contactos de riesgo entre dos **itinerarios** de dos usuarios, uno será el propio **usuario** que realiza la comprobación y el otro será el **positivo** que ha sido notificado en el sistema. Así, podemos representar estos itinerarios de la siguiente manera:

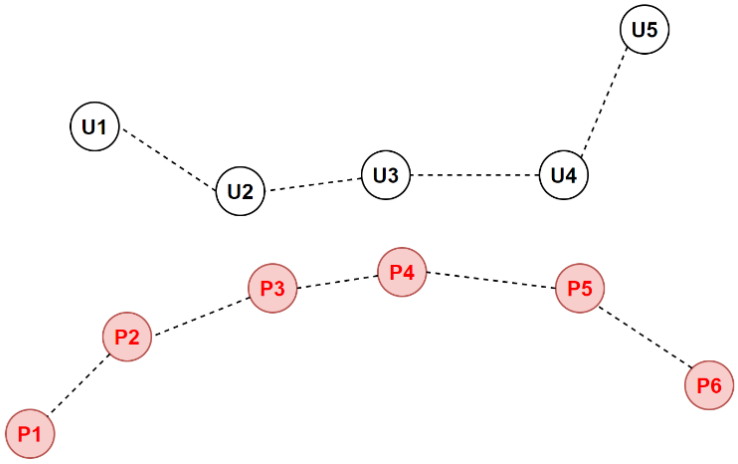


Figura 9.2. Representación de los itinerarios del usuario local y del positivo notificado en el sistema.

Los puntos anotados con la letra ***U*** representan los pares ***latitud-longitud*** del usuario que realiza la comprobación, mientras que los puntos con la letra ***P*** son los pares *latitud-longitud* del positivo con el cual se comparan las coordenadas. Como se puede observar en la figura, existen una serie de pares de puntos ***usuario-positivo*** que se encuentran próximos entre sí, basándose solamente en la **proximidad** en el **espacio**, concretamente son los pares ***{U2-P3, U3-P4, U4-P5}***. Estos tres pares de puntos representan lo que se denomina un **tramo de contacto**, es decir, una extensión en el espacio determinada por una ruta formada por los pares de puntos *usuario-positivo* que se encuentran relativamente próximos entre sí. Sin embargo, aquí solo se está contemplando la cercanía en el espacio, pero también es vital tener en cuenta la **cercanía** en el **tiempo**.

La **cercanía** en el **espacio** determina la proximidad entre dos pares de localizaciones medida en metros. Por otro lado, la **cercanía** en el **tiempo** define la proximidad entre las dos localizaciones medida en minutos o segundos, es decir, cuántos minutos y segundos de diferencia hay entre la **fecha de registro** de una localización y la otra. De este modo, la **marca** de **tiempo** en la que se registró la coordenada también juega un papel de vital importancia para el algoritmo, pues no tiene sentido considerar que dos personas han estado en contacto

basándose solamente en su posición geográfica, ya que pueden haber estado en el mismo lugar, pero en horas distintas.

El criterio utilizado para determinar cuándo dos pares de localizaciones están próximos entre sí, ya sea en el tiempo o en el espacio, es simplemente la comprobación de un **margen de diferencia**, es decir, a partir de qué valor de diferencia entre los dos puntos se considera que estos están cerca. El margen de diferencia se calcula tanto para la distancia como para el tiempo. En el caso del tiempo, se obtiene mediante una simple resta entre las fechas de registro de ambas localizaciones. Para la distancia, el cálculo se complica al tratarse de una distancia no euclídea, ya que la Tierra no es una superficie plana, sino que se asemeja más a una esfera. La distancia entre dos puntos se calcula mediante la **distancia** de ***Haversine*** lo cual se explicará en la siguiente sección. El margen de diferencia en el espacio también se conoce en el contexto del sistema como **distancia de seguridad** y se representa por ***Δd***. El margen de diferencia temporal se representa con ***Δt***. La siguiente figura muestra el cálculo de los ***deltas*** de espacio y de tiempo para los pares de localizaciones vistos en la Figura 9.2.

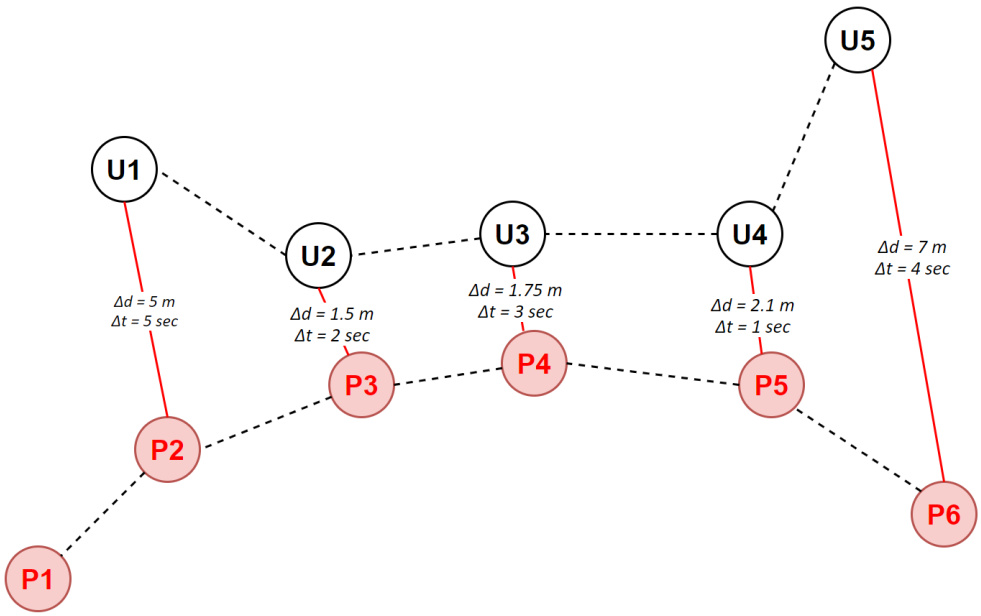


Figura 9.3. Marge de diferencia temporal y de distancia de seguridad entre las localizaciones de los itinerarios del usuario y del positivo.

Como se puede apreciar, el punto ***P1*** no se compara con ningún otro debido a que desde la perspectiva del punto ***U1***, el punto más cercano es el ***P2***. Por tanto, se toma como referencia la perspectiva del usuario local para ir calculando los márgenes de diferencia entre los puntos. De este modo, para poder determinar los pares de puntos que forman el tramo de contacto es necesario **comparar** los valores de los márgenes con un **valor fijo de referencia** que permita discernir entre si un par de localizaciones está próximo entre sí o no lo está. Hacen falta **dos** valores fijos de referencia, uno para comparar con la distancia de seguridad y otro para comprar con el margen de diferencia temporal. El valor fijo para la distancia se denota como ***D***, mientras que el valor fijo para el tiempo se denota con la letra ***T***.

Si se toma como valores fijos ***D = 3 m***, ***T = 4 sec*** ya se puede determinar qué pares de localizaciones pasarán a formar parte del tramo de contacto, pues serán todos aquellos cuyo **margen de diferencia temporal** sea menor o igual a ***T*** y al mismo tiempo su **distancia de**

seguridad sea menor o igual a ***D***. Este par de condiciones se puede representar de la siguiente manera:

$$\Delta d \leq D \;\&\&\; \Delta t \leq T$$

La siguiente figura muestra el tramo de contacto finalmente detectado entre los dos itinerarios del usuario local y del positivo, tendiendo en cuenta los anteriores valores fijos de referencia.

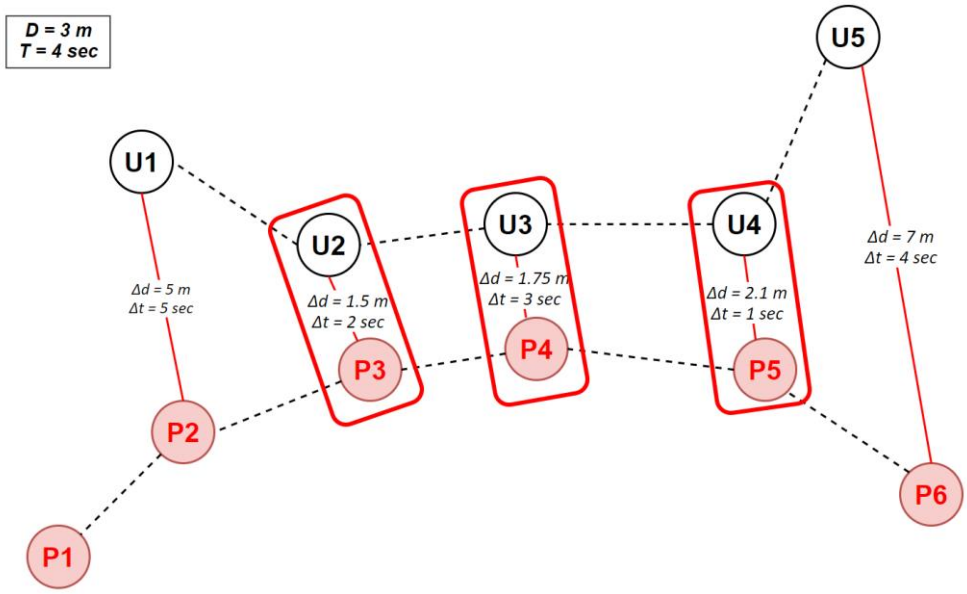


Figura 9.4. Tramo de contacto detectado entre los itinerarios del usuario local y del positivo.

Como se puede apreciar, los pares *U1-P2* y *U5-P6* no forman parte del tramo de contacto dado que no cumplen con alguna de las condiciones expuestas anteriormente. Así, este tramo de contacto da lugar a un **contacto de riesgo *RC1 = {U2-P3, U3-P4, U4-P5}*.**

Cuanto más cercanos a **0** sean los valores de *D* y *T*, más **estricta** será la comprobación, de forma que se admita menos **ruido** en los resultados al requerir que los puntos estén lo más próximos entre sí para considerar un nuevo contacto estrecho entre los dos usuarios.

Los valores fijos de tiempo y espacio constituyen dos de los principales **parámetros** de **configuración** que se pueden alterar para variar el comportamiento del algoritmo. Seleccionar estos valores será decisión del **personal sanitario** y dependerá de las características y de la evolución del virus *COVID-19*.

Con todo esto, el funcionamiento de esta implementación del algoritmo se resume en **iterar** entre los distintos positivos con los que hacer la comparación, y para cada uno de ellos:

1. Recorrer los puntos del itinerario del usuario local.
2. Para cada punto *Ux*, buscar el punto del itinerario del positivo más cercano al punto *Ux*.
3. Calcular los márgenes de diferencia temporal y de distancia de seguridad.
4. Comparar los márgenes con los valores fijos de referencia.
5. Si se cumplen ambas condiciones de cercanía temporal y espacial:

- Si ya había un tramo de contacto abierto, añadir el nuevo par de localizaciones *usuario-positivo* en el tramo.
- Si aún no había un tramo de contacto, crear uno nuevo y añadir el par de localizaciones.

6. Si no se cumplen, cerrar el tramo de contacto (si hay alguno abierto) y pasar al siguiente punto.

Esta serie de pasos constituye una primera aproximación del **pseudo-código** del algoritmo de comprobación empleado, a partir del cual se implementa la versión de código real.

Así, entre dos itinerarios puede haber 1 o más contactos de riesgo, los cuales determinarán el nivel de **gravedad** del contacto asignándole una **ponderación**. Esta ponderación determina un porcentaje entre 0 % y 100 % que representa el **riesgo** de haber sido contagiado de *COVID-19* por el positivo con el que se mantuvo contacto. Para calcular este nivel de riesgo y ponderar el contacto, se contemplan una serie de **parámetros descriptivos** de un tramo de contacto:

1. **Tiempo de exposición.** Tiempo total en minutos y segundos que dura la exposición al positivo. Se calcula con la diferencia entre los extremos que conforman la intersección entre las fechas de registro de los puntos extremos de contacto del tramo. La siguiente figura muestra un ejemplo de esta intersección.

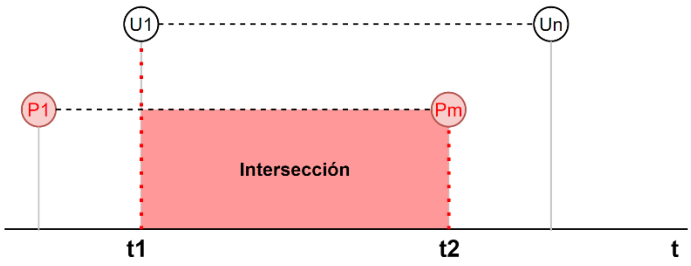


Figura 9.5. Intersección temporal entre los tramos de contacto de los itinerarios del usuario y del positivo.

Así, el tiempo de exposición se calcularía como $|\boldsymbol{t_1}_{-}\boldsymbol{t_2}|$.

2. **Proximidad media.** Distancia media en metros calculada como la media de los márgenes de distancia de seguridad de los puntos que conforman el contacto.

$$\frac{\sum_1^n \Delta d_i}{n}$$

3. **Intervalo de tiempo medio.** Media de las diferencias temporales entre las fechas de registro de cada localización. Representa el grado de **fiabilidad** de los itinerarios, pues cuanto más amplio es el intervalo de tiempo entre el registro de una coordenada y la siguiente más posibilidades hay de que por medio el usuario se haya desplazado con una mayor variabilidad. La media permite determinar cómo de fiable es la detección de un contacto de riesgo, pues si los puntos del tramo de contacto han sido registrados con un intervalo de tiempo muy amplio, el riesgo del contacto disminuirá

al decrementar la fiabilidad del rastreo. A continuación, se muestra un ejemplo de cómo se calcula este parámetro.

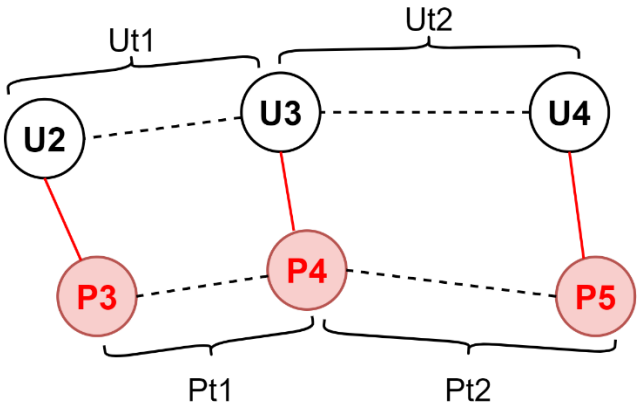
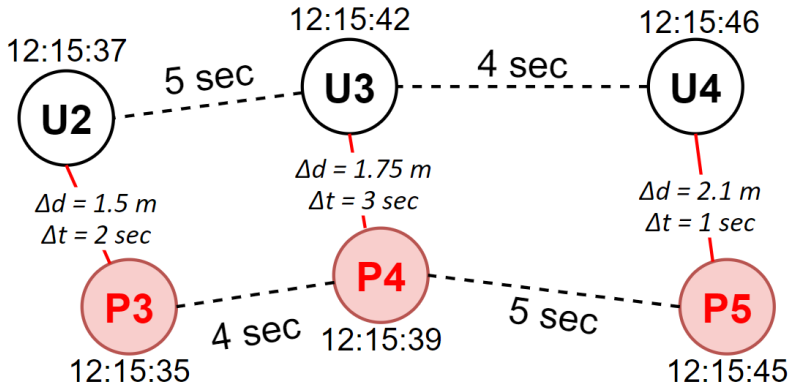


Figura 9.6. Cálculo del Intervalo de tiempo medio entre el registro de coordenadas.

En este caso, el intervalo de tiempo medio se calcularía como $\frac{Ut_1+Ut_2+Pt_1+Pt_2}{4}$. Aplicando el cálculo de estos parámetros al contacto de riesgo de ejemplo de la Figura 9.4 se obtienen los siguientes resultados.



Tiempo de exposición = **8 sec** Proximidad media = **1. 78 m** Intervalo de tiempo medio = **4.5 sec**

Figura 9.7. Ejemplo de cálculo de los parámetros descriptivos de un contacto de riesgo detectado por el algoritmo.

Una vez calculados estos parámetros, se proporciona un **peso de ponderación** a cada uno de ellos en formato de porcentaje para determinar cuánto ponderan en el porcentaje de riesgo final del contacto. De nuevo, estos pesos constituyen parte de los parámetros configurables por el personal sanitario, y variarán en función de las características y condiciones del virus, de manera que se le dé más importancia al tiempo de exposición, a la proximidad media, ambos por igual…*etc*. En cambio, antes de ponderar estos parámetros, es necesario **normalizar** sus valores para que estén dentro del rango ***[0,1]*** y así obtener un porcentaje de riesgo entre el 0% y el 100%.

Esto implica añadir nuevos parámetros de configuración que podrán ser modificados por el personal sanitario para determinar los **rangos** de valores **mínimos** y **máximos** admitidos por los parámetros descriptivos de un contacto de riesgo. Esto es necesario para poder aplicar la

fórmula de normalización de los parámetros, ya que es necesario que haya un valor mínimo y un valor máximo para cada parámetro de modo que el mínimo se corresponderá con una ponderación de **0** y el máximo con una ponderación de **1**.

Estos **rangos** de valores de los parámetros descriptivos, junto con los **pesos** asignados a cada parámetro descriptivo y los **márgenes** de distancia de seguridad y de diferencia temporal constituyen el conjunto de **parámetros configurables** por el administrador desde el panel de control *web* para controlar el comportamiento del algoritmo de comprobación.

9.4.2.1 *Distancia de Haversine*

A la hora de calcular distancias entre dos puntos en un plano vectorial, la trigonometría es un gran aliado que permite obtener distancias entre puntos en el espacio con una serie de operaciones matemáticas sencillas, utilizando, por ejemplo, la **distancia euclidiana** para calcular la distancia entre dos puntos en un espacio euclídeo. Sin embargo, cuando se trata de puntos situados en el **globo terrestre**, estas operaciones se complican debido a la forma de **esferoide oblato** que tiene la Tierra.

En este caso, existen diferentes aproximaciones para calcular la distancia entre dos puntos situados en una figura similar a la Tierra. En primer lugar, está la **Distancia** del **Gran Círculo** (***Great Circle Distance***) **[55]**, la cual contempla que los puntos se encuentran situados sobre el círculo central que se forma al seccionar transversalmente una esfera perfecta por la mitad. En cambio, una fórmula mucho más precisa y que tiene más similitudes con la forma del globo terráqueo es la **distancia** de ***Haversine*** **[56]**. Esta fórmula se deriva de la *Great Circle Distance* pero en este caso permite calcular la distancia entre cualquier par de puntos situados sobre la superficie de una esfera. Aunque no proporciona una precisión exhaustiva respecto a una medición real, la distancia de *Haversine* constituye un candidato casi perfecto para implementar el cálculo de distancias entre coordenadas mediante código, dada su sencillez y su buen nivel de precisión, además de que permite emplear **latitudes** y **longitudes** para realizar los cálculos. A continuación, se muestra la fórmula de *Haversine* para obtener la distancia entre dos puntos *A* y *B* extraída de **[56]**:

$$a = \sin^2(\phi B - \phi A/2) + \cos \phi A * \cos \phi B * \sin^2(\lambda B - \lambda A/2)$$
$$c = 2 * \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \cdot c$$

donde **Φ** representa la latitud, **λ** la longitud y ***R*** el radio de la Tierra (6.371 km).

9.4.2.2 *Código del algoritmo*

Una vez explicado el algoritmo en lenguaje natural y por medio de un pseudo-código, en este apartado se pretende mostrar el aspecto del propio código del algoritmo, implementado en lenguaje *Kotlin*. A continuación, se muestra el código del **cuerpo** del algoritmo de comprobación de contactos.

```
override fun startChecking(user: Itinerary, positive: Itinerary): List<RiskContact> {
        val result = mutableListOf<RiskContact>()
        usedLocations.clear()

        /* Recorrer localizaciones del itinerario del usuario por cada día */
        user.dates().forEach { date ->
            val uLocations = user[date] // Localizaciones del usuario
            val pLocations = positive[date] // Localizaciones del positivo
            var riskContact: RiskContact? = null // Contacto de riesgo actual
            // Comprobar si hay localizaciones del positivo para esa fecha
            uLocations.forEach { userLocation ->
                /* Buscar el punto más cercano del positivo (y que no haya sido ya comprobado) */
                val (closestLocation) = findClosestLocation(userLocation, pLocations)
                /* Comprobar Cercanía en el Espacio y en el tiempo */
                if(closestLocation != null && checkProximity(userLocation, closestLocation,
                        riskContactConfig.timeDifferenceMargin, riskContactConfig.securityDistanceMargin)) {
                    /* Iniciar nuevo tramo de contacto si aún no hay ninguno abierto. */
                    if(riskContact == null){
                        // Iniciar nuevo Tramo de contacto.
                        riskContact = RiskContact(config = riskContactConfig, positiveLabel = positive.label)
                    }
                    /* Actualizar Contacto de Riesgo */
                    riskContact?.let { update(it,userLocation, closestLocation) }
                } else {
                    /* Comprobar si existe un contacto de riesgo abierto */
                    riskContact?.let {
                        result.add(it)      /* Almacenar el Contacto de Riesgo en el resultado. */
                        riskContact = null  /* Cerrar Tramo de Contacto de Riesgo (si había uno existente) */
                    }
                }
            }
            /* Comprobar si existe un contacto de riesgo abierto */
            riskContact?.let {
                result.add(it)      /* Almacenar el Contacto de Riesgo en el resultado. */
                riskContact = null  /* Cerrar Tramo de Contacto de Riesgo (si había uno existente) */
            }
        }
        return result
}
```

Como se puede observar, se hace uso de una **lista auxiliar** en la que se van almacenando las localizaciones que ya han sido usadas, es decir, aquellas que ya han sido emparejadas con otra localización para formar un tramo de contacto. Dentro del bucle central que recorre las localizaciones del itinerario del usuario local se utilizan una serie de métodos que se describen a continuación:

- ***findClosestLocation(userLocation, pLocations)***: devuelve la localización más cercana del itinerario del positivo a la localización del usuario con la que se está comparando en esa iteración del bucle.
- ***checkProximity(userLocation, closestLocation, timeDiffMargin, securityDistance)***: comprueba la proximidad en el tiempo y en el espacio entre la localización del usuario y la del positivo que se ha detectado que está más cerca, teniendo en cuenta los valores fijos de diferencia temporal y distancia de seguridad.
- ***update(riskContact, userLocation, closestLocation)***: actualiza el contacto de riesgo pasado como parámetro añadiendo el par de localizaciones *usuario-positivo* que se ha detectado que están próximos en el tiempo y en el espacio.

Por último, cabe destacar que es importante **cerrar** un contacto de riesgo que esté abierto antes de pasar a la siguiente iteración. Un contacto de riesgo está abierto cuando en las iteraciones anteriores se han detectado pares de localizaciones próximas entre sí que han dado lugar a un tramo de contacto, por lo que en las siguientes iteraciones aún existe la posibilidad de que se detecten más puntos cercanos. De este modo, existen **tres condiciones** que se pueden dar para cerrar un contacto de riesgo, si es que hay alguno abierto:

1. No se encuentra ninguna localización del itinerario del positivo cercana a un punto del usuario, ya sea porque no existe o bien porque ya se hayan usado todas las localizaciones del positivo.
2. No existe proximidad en el tiempo o en el espacio entre las dos localizaciones de esa iteración.
3. Ya se han recorrido todas las localizaciones del usuario y se ha salido del bucle.

9.4.3 Aspectos interesantes de la implementación

En esta sección, se pretende describir con un mínimo de nivel de detalle, aquellos aspectos relativos a la implementación de las funcionalidades más destacables del sistema, es decir, aquellas cuya lógica de negocio es más compleja y que es interesante describirla en detalle para dejar constancia de su funcionamiento por escrito. El objetivo **no** es realizar una descripción exhaustiva de bajo nivel sobre todos los aspectos de implementación del sistema ya que sería demasiado extenso y no tiene sentido desviar la atención en tales detalles de código. Se trata más bien de enfatizar el funcionamiento interno, a nivel de componentes, de aquellos módulos o partes del sistema más complejas, o bien aquellos aspectos de código que resaltan sobre el resto.

9.4.3.1 *Llamadas a la API REST desde la aplicación móvil*

Anteriormente ya se comentó que para realizar las llamadas de red a la *API REST* desde la aplicación móvil se utiliza *Retrofit* junto con *GSON* para convertir los objetos *JSON* en entidades del dominio y viceversa. Sin embargo, para el caso de la aplicación móvil es necesario gestionar las posibles **excepciones** que se puedan producir en una petición de red, como un *timeout* o un error *http* enviado desde el servidor de la gama de 4xx. Para ello, se puede utilizar el típico bloque ***try catch***, aunque sería necesario repetirlo en todos los sitios en los que se invoque una llamada de red. Para hacer que las peticiones de red sean más mantenibles y reutilizables se utiliza una función de utilidad ***apiCall*** que es típica en la comunidad *Android* para gestionar las peticiones de red. El código fuente de esta función se puede consultar en los anexos, en el apartado 15.4.7. CÓDIGO DE LA FUNCIÓN DE UTILIDAD DE RED APICALL.

Se trata de una función ***suspend***, es decir, realiza llamadas asíncronas y se suspende hasta que estas se completan, que recibe como parámetro un ***dispatcher*** de corrutinas y una **función *lambda*** también de tipo *suspend*. Esta función pasada como parámetro, representa el bloque de código que invoca la llamada de red, en este caso, el método de la interfaz que implementa *Retrofit* por debajo. Así, el resultado devuelto por este método de la interfaz es envuelto en un objeto ***APIResult***, que representa una especie de enumerado especial. Se trata de una **clase sellada** de *Kotlin* que funciona a modo de enumerado y representa tres tipos de **estados:**

- **Éxito**. Se trata de una respuesta OK con código 200 que envuelve el valor específico obtenido como respuesta de la petición.
- **Error *HTTP.*** Modela cualquier error genérico *HTTP*, en el rango de los 4XX y 5XX, envolviendo un objeto *ResponseError* que contiene la información del error (su código, un mensaje y el *timestamp* en el que se produjo el error)

- **Error de red**. Modela cualquier otro error que surja en la transmisión de los datos, como, por ejemplo, un error de tiempo de espera agotado porque no se haya podido entablar conexión con el servidor.

Para construir estos objetos, desde la función de utilidad, se ejecuta la llamada a la *API* dentro de una corrutina del *dispatcher* pasado como parámetro, envuelta en un bloque ***try catch***. De este modo, si la petición se completa con éxito se devuelve un envoltorio de éxito. De lo contrario, se maneja cualquier excepción que se produzca en el bloque *catch* comprobando de qué tipo es la excepción. Si es una excepción de entrada/salida (*IOException*) se devuelve un error de red, si es una excepción *http* (*HttpException*) se devuelve un error *HTTP.* A continuación, se muestra un ejemplo del código cliente que utiliza esta función de utilidad *apiCall*.

```
return apiCall(dispatcher) {
    positiveAPI.notifyPositive(positive)
}
```

Como se puede observar, en *Kotlin* la función *lambda* se pasa como parámetro incluyendo el bloque de código entre llaves después de la llamada a la función de utilidad. La función de utilidad devuelve un *APIResult* que envuelve un objeto con el resultado, en caso de que la petición haya sido exitosa, o un objeto con el error si ha ocurrido algún problema.

9.4.3.2 *Variables de entorno*

En el caso de la **aplicación móvil**, las variables de entorno se utilizan principalmente para seleccionar entre distintas ***URLs* base** sobre las que se realizan las peticiones, en función de la variante de construcción seleccionada (*DEBUG, RELEASE, DEBUG_LOCAL, DEBUG_DEPLOY*). Para ello, se utiliza la herramienta ***Gradle*** junto con el fichero ***local.properties*** que es propio de cada espacio de trabajo y no se sube al control de versiones. Este fichero contiene los pares de clave-valor utilizados por la aplicación y que son información sensible que no debe ser revelada, por lo que estos valores solo se cargan en tiempo de compilación cuando se construye la aplicación. Así, el contenido del fichero con estas *URLs* tendría la siguiente forma:

localUrl = url local
productionUrl = url de producción

De este modo, desde el módulo de dependencias de *Gradle* se lee este fichero mediante las siguientes líneas.

```
// Local Properties
def localProperties = new Properties()
localProperties.load(new FileInputStream(rootProject.file("local.properties")))
```

Con ello, las variables ya estarán disponibles para ser utilizadas en el fichero *gradle* a través de la forma ***localProperties['variable']*** y así poder definir variables de entorno para las distintas variantes. La siguiente captura muestra un ejemplo de cómo se define una variable de entorno de la *URL* base para la variante ***debugLocalhost***.

```
debugLocalhost {
    initWith debug
    buildConfigField "String", "BASE_URL", "\"" + localProperties['localUrl'] + "\""
}
```

Ahora desde el código fuente se puede acceder a estas variables a través de la clase ***BuildConfig*** que se genera en la construcción de la aplicación, en este caso se accedería con ***BuildConfig.BASE_URL***.

En el caso del **servidor *web*,** el proceso es similar, pero creando ficheros con la extensión ***.env.[entorno]*** donde entorno es el nombre del entorno de ejecución, uno para producción y otro para *testing*, en este caso. Estos ficheros contienen los pares clave-valor de las variables, en este caso solo hay una variable ***FIRESTORE_ENV*** que especifica sobre qué documentos de la base de datos central se lanzan las operaciones de persistencia, y tampoco se suben al control de versiones. Para cargar estos ficheros en el servidor, se utilizan las librerías ***dot-env*** y ***cross-env***. La primera permite cargar las variables de entorno desde ficheros *.env* de tal manera que puedan ser accedidas desde el código mediante ***process.env.VARIABLE_DE_ENTORNO***. La segunda permite especificar en los *scripts* de ejecución del ***package.json*** el valor de una variable de entorno, de modo que se pueda utilizar para indicar qué fichero *.env* hay que cargar en función del entorno. Esto se muestra en la siguiente imagen de los *scripts* del *package.json*.

```
"scripts": {
  "test": "cross-env NODE_ENV=test mocha --timeout 10000",
  "startDev": "cross-env NODE_ENV=production nodemon app.js",
  "startTest": "cross-env NODE_ENV=test nodemon app.js",
  "start": "node app.js"
},
```

De este modo, las variables de entorno se encargan desde el código mediante esta instrucción utilizando la librería *dotenv*:

```
require('dotenv').config({path: `.env.${process.env.NODE_ENV}`})
```

En la aplicación *web*, se sigue el mismo procedimiento de utilizar ficheros *.env* para cargar las variables de entorno. En este caso, se utilizan para cambiar la *URL* base sobre la cual se lanzan las peticiones a la *API REST* con *axios.* Sin embargo, el proceso se simplifica ya que no es necesario cargar las variables, sino que el *framework* de *Vue* ya carga automáticamente un fichero u otro en función del entorno de ejecución.

El uso de variables de entorno permite gestionar diferentes variables usadas en el código que varían en función del entorno de despliegue del subsistema, además de garantizar un mayor nivel de seguridad al esconder el valor de estas variables en el espacio de trabajo local.

9.4.3.3 *Alarmas de localización y de comprobación*

Para que el rastreo de ubicación y la comprobación de contactos se ejecuten a una hora determinada se hace uso de una de las *APIs* del *framework* de *Android*, se trata del ***AlarmManager***. Esta *API* permite planificar tareas para que se ejecuten en un momento especificado. La precisión con la que se inician estas tareas varía en función de diversos factores, como el nivel de *API*, pues a partir de la *API 19* las alarmas pasan a ser **inexactas** dado que el SO ordena las alarmas próximas entre sí para ahorrar batería, o el número de procesos

que se estén ejecutando al mismo tiempo, porcentaje de memoria utilizado…*etc.* Estas alarmas son ejecutadas por la *CPU* incluso cuando la aplicación no se está ejecutando o si el dispositivo está bloqueado. Para alcanzar la máxima precisión en el disparo de estas alarmas se utiliza el método ***setExactAndAllowWhileIdle***, el cual permite planificar alarmas exactas que sean disparadas incluso con el dispositivo bloqueado.

El código a ejecutar cuando se dispara una de estas alarmas se incluye en un ***PendingIntent***, el cual representa un *Intent* que puede ser ejecutado por un agente externo a la aplicación (el *AlarmManager* en este caso). Mediante este *Intent* se pueden representar múltiples abstracciones de *Android* como *Activities*, *Services* o *Broadcast Receivers*. En el caso de este proyecto, las alarmas de localización disparan un ***foreground service*** el cual accede directamente al *location tracker* encargado de iniciar la obtención de actualizaciones de localización, mientras que las alarmas de comprobación hacen que se dispare un ***broadcast receiver*** desde el cual se pospone una nueva alarma para el siguiente día y además se ejecuta un *foreground service* que alberga el *manager* para ejecutar la comprobación de contactos.

9.4.3.4 *Notificaciones push periódicas*

Las notificaciones *push* se envían a los clientes *Android* por medio de *Firebase Cloud Messaging*, a través del *SDK Admin* de *Firebase* que se utiliza desde el servidor *web*, es decir, el servidor *web* es el encargado de iniciar la transmisión de los mensajes *push* a los clientes por medio de la *API* de *Firebase*. Para ello, se construye un objeto ***JSON*** que siga el esquema de un mensaje *push* tal y como se muestra a continuación:

```
Message: {
        Notification: {
                Title
                Body
        }
        Topic
}
```

Como se puede observar, el campo ***topic*** representa el tema al que está vinculado el mensaje de modo que todos aquellos clientes *Android* suscritos a dicho tópico recibirán el mensaje, siguiendo el patrón ***publish-subscribe***. Así, mediante el método ***send(message)*** de la *API* ***messaging*** de *Firebase* se dispara el envío de estos mensajes a los clientes *Android*.

Por otro lado, la periodicidad del envío de estas notificaciones se consigue por medio de la programación de **tareas síncronas** en *Node.js*, gracias a la librería ***node-cron***. Esta librería permite programar tareas en el tiempo para que sean ejecutadas a ciertas horas y se repitan periódicamente. Para ello, utiliza una nomenclatura peculiar basada en un *string* formado por seis asteriscos, de esta forma:

* * * * * *

Así, cada asterisco representa una **unidad** de **tiempo** a la que se le puede dar un valor, tal y como muestra la siguiente figura.

```
# ┌──────────── second (optional)
# │ ┌────────── minute
# │ │ ┌──────── hour
# │ │ │ ┌────── day of month
# │ │ │ │ ┌──── month
# │ │ │ │ │ ┌── day of week
# │ │ │ │ │ │
# │ │ │ │ │ │
# * * * * * *
```

El asterisco juega el papel de comodín, de manera que un asterisco en una unidad de tiempo significa que la tarea se repetirá tras cada iteración de esa unidad de tiempo. Para el contexto del sistema, solo se hace uso de los asteriscos de **minutos** y **horas** para programar el envío de notificaciones *FCM* a una hora del día determinada. Por ejemplo, la siguiente estructura representa una tarea síncrona que se ejecuta todos los días a las 12:35:

35 12 * * *

9.4.3.5 *Inyección de dependencias*

Como ya se mencionó anteriormente en las herramientas, la inyección de dependencias en la aplicación móvil se implementa a través de la librería ***Dagger-Hilt***, la cual se encarga de llevar un registro del árbol de dependencias de las clases de la aplicación. Así, por medio de anotaciones en las clases resulta muy sencillo inyectar y utilizar dependencias en las clases sin preocuparse de inyectarlas manualmente. Para ello, los constructores de aquellas clases que sean inyectables se anotan con ***@Inject***, de manera que *Hilt* sea consciente de los constructores para crear las clases más adelante. Esto permite que al declarar una de esas clases como un atributo en otro constructor, *Hilt* inicialice automáticamente dicha dependencia y la inyecte en la clase o clases donde es requerida.

Para aquellas dependencias que se inicializan de manera especial, como, por ejemplo, las instancias de los clientes de la *API REST* con *Retrofit,* surge el concepto de **módulos** de **dependencias**. Estos módulos son clases anotadas con ***@Module*** y ***@InstallIn*** que contienen la manera de inicializar y construir estas dependencias especiales que luego serán accesibles en un ámbito o *scope* determinado. Este ámbito se define mediante la anotación *@InstallIn*. *Hilt* hará uso de estos módulos en tiempo de ejecución para construir aquellas dependencias especiales e inyectarlas en los constructores y clases donde sean requeridas.

En el caso del servidor *web*, también se implementa la inyección de dependencias entre los *routers*, controladores y repositorios, aunque en este caso se implementa de manera manual manteniendo una clase global que contiene la inicialización de las dependencias y los métodos necesarios para acceder a ellas desde otras clases.

9.4.4 Descripción Detallada de las Clases

Para terminar con la creación del sistema, se describe el contenido y las responsabilidades de aquellas clases de código más relevantes que interactúan en el sistema. De nuevo, no se trata de describir todas las clases existentes junto con todas sus operaciones y miembros, pues eso sería demasiado excesivo y no aportaría nada útil de cara a la lectura del documento. El objetivo perseguido por este apartado es más bien reflejar un mayor nivel de detalle sobre la

implementación de aquellas clases o módulos del sistema que tengan un nivel de complejidad considerable y que formen parte de las funcionalidades más significativas del sistema.

Dado que el servidor *web* y el panel de control *web* contienen escasa **lógica de negocio,** gran parte de las clases aquí descritas serán aquellas que conforman la aplicación móvil de rastreo, que como se viene repitiendo a lo largo del documento, es el subsistema más complejo y que constituye el núcleo principal del sistema. Sin embargo, aquí no se describen todas las clases de la aplicación móvil definidas en el diseño, sino solo aquellas más importantes y complejas. De manera adicional, se muestran los módulos que representan los ***routers*** que conforman la ***API REST*** del servidor *web*.

9.4.4.1 *RiskContactResult*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContactResult	Clase de datos	Representa el resultado de una comprobación de contactos de riesgo.	*Parcelable*
Responsabilidades			
Número	**Descripción**		
1	Almacenar los tramos de contacto de riesgo entre dos usuarios.		
2	Calcular los datos estadísticos de un resultado.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*RiskContact*	*getHighestRiskContact*	
Público	*Double*	*getTotalMeanRisk*	
Público	*Long*	*getTotalMeanExposeTime*	
Público	*Double*	*getTotalMeanProximity*	
Público		*orderRiskContactsByRisk*	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público		*Long*	*resultId*
Público		*List<RiskContact>*	*riskContacts*
Público		*Int*	*numberOfPositives*
Público		*Date*	*timestamp*
Observaciones			

9.4.4.2 *RiskContact*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContact	Clase de datos	Representa un contacto de riesgo con un positivo.	*Parcelable*
Responsabilidades			
Número	**Descripción**		
1	Almacenar los puntos de contacto de un tramo de riesgo.		
2	Calcular los parámetros de un contacto de riesgo.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público		*addContactLocations*	userLocation: UserLocation positiveLocation: UserLocation
Privado		*calculateExposeTime*	
Privado		*calculateMeanProximity*	
Privado		*calculateMeanTimeInterval*	
Privado		*calculateRiskLevel*	
Privado	*Triple<Double, Double, Double>*	*normalize*	
Privado	*RiskLevel*	*getRiskLevel*	*riskPercent: Double*
Privado	*Pair<Date, Date>*	*getIntersection*	*date11: Date* *date12: Date* *date21: Date* *date22: Date*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público		*Long*	*riskContactId*
Público		*Long*	*riskContactResultId*
Público		*List<RiskContactLocation>*	*contactLocations*
Público		*RiskLevel*	*riskLevel*
Público		*Double*	*riskPercent*
Público		*Double*	*riskScore*
Público		*Long*	*exposeTime*
Público		*Double*	*meanProximity*
Público		*Long*	*meanTimeInterval*
Público		*Date*	*startDate*
Público		*Date*	*endDate*
Público		*RiskContactConfig*	*config*
Observaciones			

9.4.4.3 *RiskContactLocation*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContactLocation	Clase de datos	Representa un par de localizaciones que están próximas entre sí.	*Parcelable*
Responsabilidades			
Número	**Descripción**		
1	Almacenar el par de puntos que representan el contacto.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público		*Long*	*riskContactLocationId*
Público		*String*	*userContactPointName*
Público		*Point*	*userContactPoint*
Público		*String*	*positiveContactPointName*
Público		*Point*	*positiveContactPoint*
Público		*Long*	*riskContactId*
Observaciones			

9.4.4.4 *Positive*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Positive	Clase de datos	Representa un positivo en *Covid-19*	
Responsabilidades			
Número	**Descripción**		
1	Almacenar los datos del itinerario de un positivo.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público		*Long*	*positiveId*
Público		*String*	*positiveCode*
Público		*Date*	*timestamp*
Público		*List<UserLocation>*	*locations*
Público		*PersonalData*	*personalData*
Público		*Boolean*	*asymptomatic*
Público		*Boolean*	*vaccinated*
Observaciones			

9.4.4.5 *LocationForegroundService*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
LocationForegroundService	Componente *Android*	Modela un servicio *Android* para rastrear la ubicación.	*Service*
Responsabilidades			
Número	**Descripción**		
1	Rastrear la ubicación del usuario en primer y segundo plano.		
2	Almacenar las coordenadas en la base de datos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Privado		*Init*	
Privado		*startLocationService*	*alarmID: Long*
Privado		*stopLocationService*	*alarmID: Long*
Privado	*Notification*	*createNotification*	
Privado	*LocationTrackRequest*	*createLocationTrackRequest*	
Privado		*sendBroadcast*	*command: String*
Privado	*Boolean*	*checkLocationSettings*	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público		*LocationTracker*	*locationTracker*
Público		*LocationUpdateCallback*	*locationCallback*
Público		*ConfigRepository*	*configRepository*
Público		*LocationAlarmManager*	*locationAlarmManager*
Público		*Boolean*	*isActive*
Público		*SharedPreferences*	*sharedPrefs*
Observaciones			

9.4.4.6 *LocationTracker*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
LocationTracker	Interfaz	Modela el contrato del rastreador de ubicación.	
Responsabilidades			
Número	**Descripción**		
1	Representar la interfaz del rastreador de ubicación.		
2	Controlar el inicio y detención del rastreo de ubicación.		
3	Acceder a los servicios de geolocalización de *Android*.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*Boolean*	*start*	*mode: LocationUpdateMode*
Público	*Boolean*	*stop*	*mode: LocationUpdateMode*
Público		*setLocationRequest*	*request: LocationTrackRequest*

Público		*setCallback*	*callback: LocationUpdateCallback*
Público		*setIntentCallback*	*pendingIntent: PendingIntent*
Público		*setLocationProvider*	*provider: String*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			
Existen dos clases que implementan esta interfaz: • ***FusedLocationTracker***: accede a los servicios de geolocalización a través del *fused provider* de *Google Play*. • **LocationManagerTracker**: accede a los servicios de geolocalización a través del *location manager* de *Android.*			

9.4.4.7 *RiskContactDetector*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContactDetector	Interfaz	Modela el contrato de la comprobación de contactos de riesgo.	
Responsabilidades			
Número	**Descripción**		
1	Representar la interfaz de la comprobación de contactos.		
2	Ejecutar el algoritmo de comprobación para detectar contactos de riesgo.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*List<RiskContact>*	*startChecking*	*user: Itinerary* *positive: Itinerary*
Público	*Boolean*	*checkSpaceProximity*	*pointA: UserLocation* *pointB: UserLocation* *radius: Double*
Público	*Pair<UserLocation, Double>*	*findClosestLocation*	*location: UserLocation* *otherLocations: List<UserLocation>*
Público	*Boolean*	*checkTimeProximity*	*pointA: UserLocation* *pointB: UserLocation* *time: Int*
Público		*setConfig*	*config: RiskContactConfig*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			
Existen una clases que implementa esta interfaz: • ***RiskContactDetectorImpl:*** contiene una implementación concreta del algoritmo de comprobación de contactos y de sus métodos auxiliares.			

9.4.4.8 *LocationAlarmManager*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
LocationAlarmManager		Gestiona y planifica las alarmas de localización.	
Responsabilidades			
Número	**Descripción**		
1	Gestionar y programar las alarmas de localización.		
2	Invocar a los repositorios para recuperar e insertar alarmas de localización.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*LiveData<List<LocationAlarm>*	*getAllAlarms*	
Público | *suspend*		*deleteAlarm*	*alarmID: Long*
Privado | *suspend*	*List<LocationAlarm>*	*checkAlarmCollisions*	*alarm: LocationAlarm*
Público | *suspend*	*ValueWrapper<Unit>*	*setAlarm*	*alarm: LocationAlarm*
Público | *suspend*		*toggleAlarm*	*alarmID: Long enable: Boolean*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*AlarmHelper*	*alarmHelper*
Privado		*LocationRepository*	*locationRepository*
Público		*Context*	*ctx*
Observaciones			

9.4.4.9 *RiskContactManager*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContactManager		Gestiona las comprobaciones de contactos.	
Responsabilidades			
Número	**Descripción**		
1	Ejecutar la comprobación de contactos y guardar los resultados.		
2	Recuperar las coordenadas locales y los positivos almacenados en la nube.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público		*checkRiskContacts*	*date: Date*
Público		*setCheckMode*	*checkMode: CheckMode*
Público	*CheckMode*	*getCheckMode*	
Privado	*List<Positive>*	*filterPositives*	*Positives: List<Positive>*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Privado		*RiskContactDetector*	*Detector*
Privado		*LocationRepository*	*locationRepository*
Privado		*PositiveRepository*	*positiveRepository*
Privado		*RiskContactRepository*	*riskContactRepository*
Privado		*ConfigRepository*	*configRepository*
Privado		*StatisticsRepository*	*statisticsRepository*
Privado		*InAppNotificationManager*	*notificationManager*
Observaciones			

9.4.4.10 *RiskContactAlarmManager*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RiskContactAlarmManager		Gestiona las alarmas de comprobación de contactos.	
Responsabilidades			
Número	**Descripción**		
1	Gestionar las alarmas de comprobación de contactos.		
2	Insertar, actualizar y recuperar alarmas de comprobación.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público | *suspend*	*ValueWrapper <RiskContactAlarm>*	*set*	riskContactAlarm: RiskContactAlarm
Público | *suspend*		*remove*	*alarmID: Long*
Público | *suspend*		*toggle*	*activate: Boolean*
Público | *suspend*		*reset*	*alarm: RiskContactAlarm*
Privado | *suspend*	*Boolean*	*checkAlarmCollision*	*alarm: RiskContactAlarm*
Privado | *suspend*	*Boolean*	*checkAlarmCount*	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*RiskContactRepository*	*riskContactRepository*
Privado		*AlarmHelper*	*alarmHelper*
Observaciones			

9.4.4.11 *PositiveManager*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
PositiveManager		Gestiona la notificación de positivos.	
Responsabilidades			
Número	**Descripción**		
1	Construir los objetos positivos a partir de las localizaciones locales.		
2	Subir los positivos a la nube.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público | *suspend*	*ValueWrapper <NotifyPositiveResponse>*	*notifyPositive*	*personalData: PersonalData* *answers: Map<String, Boolean>* *date: Date*
Público | *suspend*	*List<Positive>*	*getLocalPositives*	
Privado| *suspend*	*Boolean*	*checkNotifyLimit*	*days: Int* *date: Date*
Privado	*Boolean*	*checkLocations*	*locations: List<UserLocation>*
Privado| *suspend*	*ValueWrapper <NotifyPositiveResponse>*	*processNotifyResult*	*response: APIResult <NotifyPositiveResponse>*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*PositiveRepository*	*positiveRepository*
Privado		*LocationRepository*	*locationRepository*
Privado		*ConfigRepository*	*configRepository*
Observaciones			

9.4.4.12 *AlarmHelper*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
AlarmHelper		Gestiona la programación y cancelación de alarmas de *Android.*	
Responsabilidades			
Número	**Descripción**		
1	Programar y cancelar alarmas de *Android*.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*Boolean*	*setLocationAlarm*	*alarm: LocationAlarm*
Público		*cancelLocationAlarm*	*alarmID: Long*
Público	*Boolean*	*setRiskContactCheckAlarm*	*alarm: RiskContactCheckAlarm*
Público	*Boolean*	*cancelRiskContactCheckAlarm*	*Id: Long*

Privado	*PendingIntent*	*getPendingIntentForegroundService*	*Intent: Intent* *Id: Int*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*Context*	*ctx*
Privado		*AlarmManager*	*alarmManager*
Observaciones			

9.4.4.13 *APIConfig*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
apiconfig	Módulo JS	Contiene los *endpoints* de la *API* de configuración.	
Responsabilidades			
Número	**Descripción**		
1	Gestionar las peticiones entrantes y redirigirlas a los controladores.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*JSON*	*GET /config/:filename*	*filename: String*
Público	*JSON*	*POST /config/updateNotifyConfig*	*newConfig: JSON*
Público	*JSON*	*POST /config/updateRiskContactConfig*	*newConfig: JSON*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*ConfigController*	*configController*
Observaciones			

9.4.4.14 *APIPositive*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
apipositive	Módulo JS	Contiene los *endpoints* de la *API* de positivos.	
Responsabilidades			
Número	**Descripción**		
1	Gestionar las peticiones entrantes y redirigirlas a los controladores.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*JSON*	*POST /positive/notifyPositive*	*positive: JSON*
Público	*JSON*	*GET /positive/getPositives/:targetDate/:lastDays*	*targetDate: Long* *lastDays: Int*
Público	*JSON*	*GET /positive /getAllPositives*	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Privado		*PositiveController*	*positiveController*
Observaciones			

9.4.4.15 *APIStatistics*

Nombre	**Tipo**	**Descripción**	**Hereda de…**
apistatistics	Módulo JS	Contiene los *endpoints* de la *API* de estadísticas.	
Responsabilidades			
Número	**Descripción**		
1	Gestionar las peticiones entrantes y redirigirlas a los controladores.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	*JSON*	*POST /statistics/registerInstall*	*install: JSON*
Público	*JSON*	*POST /statistics/registerCheckResult*	*checkResult: JSON*
Público	*JSON*	*GET /statistics/positives/:targetDate/:lastDays*	*targetDate: Long lastDays: Int*
Público	*JSON*	*GET /statistics/checks/:targetDate/:lastDays*	*targetDate: Long lastDays: Int*
Público	*JSON*	*GET /statistics/installs/:targetDate/:lastDays*	*targetDate: Long lastDays: Int*
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado		*StatisticsController*	*statisticsController*
Observaciones			

Capítulo 10. Desarrollo de las Pruebas

Este capítulo engloba todo lo relacionado con la implementación de los casos de prueba especificados en el **Plan de Pruebas** presentado en la sección 8.6 ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS. Por un lado, se exponen las **tecnologías**, **herramientas** y **librerías** utilizadas para desarrollar todas las pruebas especificadas en los distintos entornos, tanto de la aplicación móvil como de la *API REST*. Por otro lado, se muestra el **resultado** obtenido tras la primera **ejecución** de las **pruebas**, contrastando la salida esperada con la salida obtenida. Por último, se incluye un apartado específico para explicar los ***bugs*** encontrados mediante la ejecución de las pruebas, indicando la causa del fallo y que se realizó para arreglarlo.

10.1 Herramientas y tecnologías

Antes de pasar a ver los resultados de los casos de prueba especificados en las secciones anteriores, en este apartado se describen a grandes rasgos las principales **tecnologías** y **herramientas** utilizadas en el desarrollo de las pruebas, además de presentar brevemente **aspectos de implementación** llevados a cabo para poder realizar las pruebas. A continuación, se enumeran estas tecnologías y herramientas empleadas:

- ***JUnit4.*** Es el *framework* de *Java* para implementar y ejecutar pruebas unitarias de clases.
- ***KotlinCoroutinesTest.*** Librería de *kotlin* para dar soporte al *testing* de las corrutinas en las pruebas unitarias. Las corrutinas son similares a los hilos de *Java* y proporcionan la capacidad de ejecutar código asíncrono, lo cual dificulta la implementación de las pruebas unitarias. Esta librería permite trabajar con las corrutinas de *kotlin* como si el código se ejecutara de manera síncrona.
- ***InstantTaskExecutor***. Librearía de *AndroidX* dedicada a dar soporte a las pruebas unitarias de los componentes *LiveData*. Estos componentes son objetos observables asociados al ciclo de vida de los componentes de *Android*. Por ello, suele ser difícil implementar pruebas cuando hay objetos *LiveData* de por medio. Esta librería facilita el *testing* con observables *LiveData*.
- ***Robolectric***. Se trata de una tecnología que permite realizar pruebas instrumentadas sin necesidad de ser ejecutadas sobre un dispositivo *Android*, es decir, se puede probar código que tenga dependencias del *framework* de *Android* en el entorno de una máquina virtual de *Java*, directamente sobre el equipo de sobremesa, lo cual hace que las pruebas se ejecuten con mayor rapidez.
- ***Mockito.*** Es una librería muy popular para *mockear* y simular dependencias, de forma que permite aislar los componentes que se desean probar de sus dependencias. Proporciona utilidades muy versátiles para simular las llamadas a métodos y verificar que un método ha sido invocado con unos parámetros determinados, entre otros.
- ***Espresso.*** Se trata de un *framework* desarrollado por *Google* que permite implementar pruebas de interfaz de usuario para simular las interacciones con aplicaciones *Android*.
- ***Mocha.*** *Framework* de *JavaScript* para el desarrollo de pruebas sobre la plataforma *Node.js* que incluye soporte para la ejecución de código asíncrono.

- ***Chai***. Es una librería de aserciones BDD/TDD para la plataforma de *Node.js* que se utiliza en conjunto con el *framework* de *Mocha.*
- ***Chai-http.*** Se trata de un *plugin* de la librería *Chai* para simular un cliente *web* y así realizar peticiones de red al servidor. Contiene aserciones para verificar aspectos *http* y comprobar las respuestas enviadas por el servidor.

Una vez vistas las tecnologías, en los siguientes apartados se muestran algunos aspectos clave relativos a la implementación de las pruebas que es interesante recalcar.

10.1.1 Observar los cambios de *LiveData* en las pruebas

Una de las dificultades a afrontar a la hora de desarrollar pruebas en *Android* siguiendo la arquitectura *MVVM* son los objetos *LiveData*. Estos objetos son componentes **observables**, lo cual quiere decir que siguen el patrón *Observer* de forma que sus subscriptores, los observadores, reciben actualizaciones cada vez que estos observables sufren algún cambio en su estado interno. El problema reside en que estas actualizaciones no se notifican cuando se ejecutan en las pruebas, ya que no existe un componente *Android* que tenga un ciclo de vida al cual estar ligado.

La solución a este problema es utilizar una **función de extensión** de *kotlin* que fuerce al *LiveData* a notificar la actualización. A continuación, se muestra el código de esta función que ha sido extraído de un artículo sobre *testing* en *Android* **[58]**.

```
fun <T> LiveData<T>.getOrAwaitValue(
    time: Long = 10,
    timeUnit: TimeUnit = TimeUnit.SECONDS,
    afterObserve: () -> Unit = {}
): T {
    var data: T? = null
    val latch = CountDownLatch(1)
    val observer = object : Observer<T> {
        override fun onChanged(o: T?) {
            data = o
            latch.countDown()
            this@getOrAwaitValue.removeObserver(this)
        }
    }
    this.observeForever(observer)

    try {
        afterObserve.invoke()

        // Don't wait indefinitely if the LiveData is not set.
        if (!latch.await(time, timeUnit)) {
            throw TimeoutException("LiveData value was never set.")
        }

    } finally {
        this.removeObserver(observer)
    }

    @Suppress("UNCHECKED_CAST")
    return data as T
}
```

Figura 10.1. Función de extensión Kotlin para obtener el valor de un LiveData en las pruebas.

Los propios desarrolladores de *Google* recomiendan utilizar este fragmento de código para recuperar el valor de un objeto *LiveData* en las pruebas.

10.1.2 Inyectar cualquier objeto con *Mockito* en *Kotlin*

La librería *Mockito* permite simular el comportamiento de un objeto cuando se invoca a alguno de sus métodos con unos parámetros determinados, si dicho método recibe parámetros. *Mockito* proporciona un método ***anyObject()*** que permite despreocuparse del tipo del objeto que se recibe como parámetro en un método cuando se quiere *mockear* dicho método. Sin embargo, en *kotlin* existe un problema con este enfoque, pues los métodos no pueden recibir ***null*** como parámetro a menos que se especifique explícitamente en la cabecera del método.

Para atajar este problema, se hace uso del siguiente código que utiliza **genericidad** para devolver un objeto *null* **casteado** al **tipo concreto** que se recibe como parámetro en el método que se desea *mockear*.

```
private fun <T> anyObject(): T {
    Mockito.anyObject<T>()
    return uninitialized()
}

private fun <T> uninitialized(): T = null as T
```

Figura 10.2. Código para inyectar objetos nulos con mockito en kotlin.

10.1.3 *Dispatcher* de *test* para las pruebas

A lo largo del código de la aplicación móvil de *Contact Tracker* se hace uso de las **corrutinas** de *kotlin* para realizar operaciones en hilos distintos al principal, el cual gestiona las operaciones de la interfaz de usuario, de manera que esta no quede bloqueada mientras se terminan de ejecutar otras operaciones de mayor carga computacional. Las corrutinas se ejecutan dentro de lo que se denomina el *scope*, el cual es proporcionado por un ***dispatcher***. Existen diferentes *dispatchers* en función de la operación que se desee realizar, sin embargo, para poder ejecutar código con corrutinas en las pruebas es necesario utilizar un *dispatcher* especial dedicado exclusivamente a los *tests*, se trata del ***TestCoroutineDispatcher***.

La siguiente captura muestra el fragmento de código de la **regla de *JUnit*** utilizada para **reemplazar** el *dispatcher* principal (*Main*) por este *dispatcher* de *test*.

```
class TestCoroutineRule : TestRule {

    val testCoroutineDispatcher = TestCoroutineDispatcher()
    val testCoroutineScope = TestCoroutineScope(testCoroutineDispatcher)

    override fun apply(base: Statement?, description: Description?) = object : Statement() {
        @Throws(Throwable::class)
        override fun evaluate() {
            Dispatchers.setMain(testCoroutineDispatcher)
            base?.evaluate()
            Dispatchers.resetMain()
            testCoroutineScope.cleanupTestCoroutines()
        }
    }

    fun runBlockingTest(block: suspend TestCoroutineScope.() -> Unit) =
        testCoroutineScope.runBlockingTest { block() }
}
```

Figura 10.3. Regla de JUnit personalizada para cambiar el dispatcher de Main por el dispatcher de test.

Como se puede observar, el método ***runBlockingTest*** recibe como parámetro un bloque de código que contenga alguna función *suspend*, es decir, que realice operaciones asíncronas, y lo ejecuta dentro del *scope* de *test* con el *dispatcher* de *test*.

De este modo, en las clases de prueba se puede inyectar esta regla para utilizar este método en cada prueba que ejecute código asíncrono como consecuencia de alguna función *suspend,* tal y como se aprecia en la siguiente captura.

```
@get:Rule
val testCoroutineRule = TestCoroutineRule() // Para utilizar el Scope de Test
```

Figura 10.4. Inyección de la regla de las corrutinas de test dentro de una clase de pruebas.

Los métodos de prueba que requieran ejecutar código asíncrono tendrán entonces la siguiente forma:

```
@Test
fun `test de prueba`() = testCoroutineRule.runBlockingTest {
    // Código asíncrono de prueba...
}
```

Figura 10.5. Ejemplo de método de prueba que ejecuta código asíncrono, envuelto por el scope de test.

De manera adicional, para garantizar que todo el código bajo pruebas se ejecuta sobre el mismo *dispatcher* de *test*, es necesario **inyectar** este *dispatcher* en todos aquellos componentes, *viewmodels,* repositorios… *etc* que ejecuten código mediante corrutinas. Para ello, es suficiente con pasar este *dispatcher* como parámetro en el constructor de dichos componentes. De lo contrario, surgirán comportamientos extraños al ejecutar las pruebas dado que una parte del código se estará ejecutando en un *dispatcher* y otra parte en otro *dispatcher* distinto.

10.1.4 Consultas en el hilo principal

Para acceder a la base de datos local *SQLite* del dispositivo se hace uso de un intermediario que sirve de abstracción para centrarse solo en las sentencias concretas a ejecutar sobre los datos en lugar de escribir grandes cantidades de código de conexión, modelado de las tablas *SQL…etc*. Este intermediario es la librería ***ROOM*** recomendada por *Google* y una de sus particularidades es que no permite ejecutar consultas *SQL* en el **hilo principal** de la aplicación (ya sean inserciones, borrados o consultas como tal). Esto lo hace para evitar bloquear el hilo de la interfaz de usuario con operaciones de lectura o escritura de datos en el almacenamiento local, que generalmente suelen ser bastante pesadas. De cara al usuario final, esta restricción es idónea para proporcionar una sensación de **fluidez** en la interfaz de usuario, permitiendo a su vez ejecutar operaciones pesadas.

En cambio, para la ejecución de las pruebas, la fluidez de la interfaz no es un requisito indispensable por lo que para facilitar la implementación y evitar conflictos entre fragmentos de código que se ejecutan en hilos distintos se le indica a *ROOM* que **relaje** esta restricción y permita ejecutar **consultas** en el **hilo principal**. Esto se realiza mediante el método ***allowMainThreadQueries()*** como se puede apreciar en el siguiente fragmento de código en la cláusula *@Before* que contiene los preparativos previos a cada método de prueba.

```
db = Room.inMemoryDatabaseBuilder(ctx, AppDatabase::class.java)
.allowMainThreadQueries() // Permitir consultas en el hilo principal.
.build()
```

Figura 10.6. Habilitando las consultas en el hilo principal para la base de datos en memoria de ROOM.

10.2 Pruebas Unitarias

Como se vio en la sección dedicada al diseño del plan de pruebas, las pruebas unitarias se realizan sobre componentes de la aplicación móvil, ejecutándolas directamente sobre el equipo de sobremesa en una máquina virtual de *Java*, las cuales residen en el **directorio *test*** dentro del proyecto de *Android Studio*, o bien sobre un dispositivo con *Android* instalado, las cuales se conocen como **pruebas instrumentadas** y residen en el **directorio *androidTest*.**

Los **códigos** de los casos de prueba se utilizan para mantener la **trazabilidad** entre los casos de prueba y los **métodos de prueba** implementados por código, de tal forma que cada método de prueba se corresponde con un caso de prueba. Los métodos de prueba están identificados mediante el código del caso de prueba que implementan.

En las siguientes subsecciones se mostrarán los casos de prueba diseñados en la sección 8.6 ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS para cada componente junto con los **resultados** obtenidos tras ejecutar las pruebas. Para no extenderse demasiado, no se mostrará el diseño previo con las condiciones y las situaciones de prueba correspondientes, esta información ya ha sido expuesta anteriormente y puede consultarse en la sección relativa a la especificación técnica del Plan de Pruebas.

10.2.1 Clases de utilidad

Para estas pruebas solo ha sido necesario utilizar el *framework* de *JUnit4* sin usar ninguna otra herramienta. Las siguientes tablas muestran los casos de prueba diseñados para probar las clases de utilidad junto con los resultados obtenidos.

Tabla 10.1. Casos de prueba con los resultados obtenidos para la clase DateUtils.

DateUtils				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
DU1	Dar formato a una fecha para convertirla en String.	Fecha y Formato: "dd-MM-yyyy HH-mm"	Fecha formateada a String con el formato indicado.	String con la fecha formateada.
DU2	Obtener un objeto fecha a partir de las horas y de los minutos.	Horas: 12 Minutos: 29	Objeto fecha "12:29"	Objeto fecha con las horas y minutos indicados.
DU3	Obtener un objeto fecha a partir de los milisegundos.	Milisegundos	Fecha correspondiente a esos milisegundos.	Fecha correspondiente.
DU4	Obtener hora y minutos a partir de una fecha con hora de mañana y otra de tarde.	Fechas con hora de mañana y de tarde	Horas y minutos correspondientes a ambas fechas.	Horas y minutos de las dos fechas.
DU5	Sumar un día a una fecha y luego restárselo.	Fecha y 1 día	Primero la fecha con un día más, y luego con un día menos.	Fecha con un día más y con un día menos.
DU6	Sumar un día a una fecha a finales de mes y luego restárselo.	Fecha y 1 día	Se incrementa un día en la fecha y además cambia al siguiente mes. Lo mismo, pero al contrario cuando se resta 1 día.	Fecha con un día más y un mes más y viceversa.
DU7	Sumar minutos a una fecha.	Fecha y 45 minutos	Fecha con 45 minutos más.	Fecha con 45 minutos más.
DU8	Obtener minutos y segundos a partir de milisegundos, pero sin completar un minuto.	54560 ms	54 segundos	54 segundos
DU9	Obtener minutos y segundos a partir de milisegundos completando un minuto.	83489 ms	1 minuto 23 segundos	1 minuto 23 segundos
DU10	Obtener minutos y segundos a partir de milisegundos completando varios minutos.	124402 ms	2 minutos 4 segundos	2 minutos 4 segundos
DU11	Obtener milisegundos a partir de minutos y segundos sin completar un minuto.	0 minutos 34 segundos	34000 ms	34000 ms
DU12	Obtener milisegundos a partir de minutos y segundos completando varios minutos.	2 minutos 80 segundos	200000 ms	200000 ms
DU13	Obtener los segundos de diferencia entre dos fechas iguales.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 11:20"	0 segundos	0 segundos

DU14	Obtener los segundos de diferencia entre dos fechas con la misma hora y minutos.	Fecha 1: "19/09/2021 11:20:00" Fecha 2: "19/09/2021 11:20:34"	34 segundos	34 segundos
DU15	Obtener los segundos de diferencia entre dos fechas distintas.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 11:35"	900 segundos	900 segundos
DU16	Modificar año, mes y día de una fecha.	Fecha: "19/09/2021 11:20" Año:2023 Mes:2 Día:15	Nueva fecha: "15/03/2023 11:20"	Nueva fecha: "15/03/2023 11:20"
DU17	Obtener los días de diferencia entre dos fechas con el mismo día.	Fecha 1: "19/09/2021 11:20" Fecha 2: "19/09/2021 12:23"	0 días	0 días
DU18	Obtener los días de diferencia entre dos fechas con distintos días, pero misma hora.	Fecha 1: "19/09/2021 11:20" Fecha 2: "15/09/2021 11:20"	4 días	4 días
DU19	Obtener los días de diferencia entre dos fechas con distintos días y distintas horas.	Fecha 1: "19/09/2021 11:20" Fecha 2: "15/09/2021 12:00"	3 días	3 días
DU20	Obtener los días de diferencia entre dos fechas de distintos meses.	Fecha 1: "19/09/2021 11:20" Fecha 2: "02/10/2021 12:00"	13 días	13 días

Tabla 10.2. Casos de prueba junto con los resultados obtenidos para la clase LocationUtils.

LocationUtils				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
LU1	Dar formato a una localización de usuario para transformarla en String.	Localización de usuario	Localización formateada a String con todos sus datos.	String con la localización formateada.
LU2	Convertir una localización de usuario a un objeto LatLng.	Localización de usuario	Objeto LatLng con la latitud y longitud de la localización.	Objeto LatLng con latitud y longitud de la localización indicada.
LU3	Obtener la distancia entre dos puntos cercanos representados mediante coordenadas (lat y lng) utilizando la distancia de Haversine.	Punto 1, Punto 2	Distancia en metros entre ambos puntos indicados.	Distancia aproximada en metros entre ambos puntos.
LU4	Obtener la distancia entre dos puntos lejanos representados mediante coordenadas (lat y lng) utilizando la distancia de Haversine.	Punto 1, Punto 2	Distancia en metros entre ambos puntos indicados.	Distancia aproximada en metros entre ambos puntos.

Tabla 10.3. Casos de prueba junto con los resultados obtenidos para la clase NumberUtils.

NumberUtils				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
NU1	Redondear un número a un decimal.	Número: 2,8776512 Decimales: 1	2,9	2,9
NU2	Redondear un número a varios decimales.	Número: 2,8776512 Decimales: 3	2,878	2,878
NU3	Redodear un número a ningún decimal.	Número: 2,8776512 Decimales: 0	3	3

10.2.2 Data Access Objects (DAOs)

Para implementar las pruebas de los *DAOs* solo se requiere el *framework* de *JUnit4* y una base de datos en memoria de *ROOM.* En este caso, estas pruebas se sitúan en el directorio ***androidTest*** ya que son instrumentadas y requieren de un dispositivo *Android* para ejecutarse. Para poder ejecutar código asíncrono, se establece como ejecutores de la base de datos los *dispatchers* de *test*. Las siguientes tablas muestran los casos de prueba para los distintos *DAOs* junto con los resultados obtenidos.

Tabla 10.4. Casos de prueba junto con los resultados obtenidos para el DAO de localizaciones.

UserLocationDao				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
ULDAO1	Insertar una localización de usuario.	Localización de usuario	Se inserta la localización de usuario correctamente.	Localización de usuario insertada.
ULDAO2	Obtener todas las localizaciones de usuario.		Lista con todas las localizaciones de usuario almacenadas y ordenadas descendentemente por fecha.	Lista con todas las localizaciones.
ULDAO3	Buscar una localización de usuario por ID.	ID de la localización	Localización de usuario de ID indicado.	Localización de usuario con el ID indicado.
ULDAO4	Obtener las localizaciones de usuario registradas en una fecha dada en la que no existen localizaciones.	Fecha: “2021-12-10”	Lista vacía sin localizaciones porque no existen en esa fecha.	Lista vacía.
ULDAO5	Obtener las localizaciones de usuario registradas en una fecha dada en la que existen varias localizaciones.	Fecha: “2021-09-19”	Lista con las localizaciones filtradas por la fecha indicada y ordenada descendentemente por fecha.	Lista de localizaciones filtradas.
ULDAO6	Eliminar todas las localizaciones de usuario.		Se eliminan todas las localizaciones de la base de datos.	Localizaciones eliminadas.

ULDAO7	Eliminar localizaciones por fecha en la que no existen localizaciones.	Fecha: "2021-12-10"	No se elimina ninguna localización porque en esa fecha no existen.	No se elimina ninguna localización.
ULDAO8	Eliminar localizaciones por fecha en la que si existen varias localizaciones.	Fecha: "2021-09-19"	Se eliminan 3 localizaciones de usuario que fueron registradas en esa fecha.	Localizaciones de usuario de la fecha indicada eliminadas.
ULDAO9	Obtener localizaciones entre dos fechas indicadas donde no existen localizaciones por la derecha.	Inicio: "2021-09-23" Fin: "2021-09-25"	Lista vacía porque no existen localizaciones registradas en ese rango de fechas.	Lista vacía.
ULDAO10	Obtener localizaciones entre dos fechas indicadas donde no existen localizaciones por la izquierda.	Inicio: "2021-09-11" Fin: "2021-09-14"	Lista vacía porque no existen localizaciones registradas en ese rango de fechas.	Lista vacía.
ULDAO11	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones por la izquierda del rango.	Inicio: "2021-09-19" Fin: "2021-09-20"	Lista con 3 localizaciones que han sido registradas dentro de ese rango de fechas.	Lista con 3 localizaciones filtradas.
ULDAO12	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en el interior del rango.	Inicio: "2021-09-16" Fin: "2021-09-20"	Lista con 4 localizaciones que han sido registradas dentro de ese rango de fechas.	Lista con 4 localizaciones filtradas.
ULDAO13	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en por la derecha del rango.	Inicio: "2021-09-20" Fin: "2021-09-22"	Lista con 1 localización que ha sido registrada dentro de ese rango de fechas.	Lista con 1 localización filtrada.
ULDAO14	Obtener localizaciones entre dos fechas indicadas donde existen localizaciones registradas en varias zonas del rango.	Inicio: "2021-09-14" Fin: "2021-09-17"	Lista con 2 localizaciones que han sido registradas dentro de ese rango de fechas.	Lista con 2 localizaciones filtradas.

Tabla 10.5. Casos de prueba junto con los resultados obtenidos para el DAO de positivos.

PositiveDao				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
PDAO1	Insertar un positivo con localizaciones.	Positivo	Se inserta el positivo y se asocia correctamente con sus localizaciones. Se devuelve su ID.	Positivo insertado y asociado con sus localizaciones.
PDAO2	Obtener todos los positivos con sus localizaciones.		Se obtiene una lista con todos los positivos almacenados y sus localizaciones asociadas ordenados por fecha de notificación descendente.	Lista con todos los positivos y sus localizaciones asociadas ordenados del más al menos reciente.
PDAO3	Obtener todos los códigos identificadores de los positivos.		Se obtiene una lista con todos los códigos identificadores de los positivos almacenados.	Lista con todos los códigos de los positivos.
PDAO4	Obtener el número de positivos notificados en una fecha en la que no haya positivos.	"2021-09-19"	Se obtiene 0 porque no hay positivos notificados en esa fecha.	0 positivos

PDAO5	Obtener el número de positivos notificados en una fecha en la que se hayan notificado varios positivos.	"2021-09-20"	Se obtiene 2 porque hay dos positivos notificados en esa fecha.	2 positivos
PDAO6	Obtener el último positivo que fue notificado de entre varios positivos.		Se obtiene el último positivo de todos los que se notificaron.	Último positivo notificado.
PDAO7	Obtener el último positivo que fue notificado cuando no existen positivos.		Se obtiene NULL porque no existen positivos.	NULL

Tabla 10.6. Casos de prueba junto con los resultados obtenidos para el DAO de alarmas de localización.

LocationAlarmDao				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
LADAO1	Insertar una alarma de localización y luego recuperarla.	Alarma de localización	Se inserta la alarma de localización correctamente.	Alarma de localización insertada.
LADAO2	Actualizar los datos de una alarma de localización existente y luego recuperarla.	Alarma de comprobación actualizada	Se actualiza la alarma de localización con los nuevos datos.	Alarma de localización actualizada.
LADAO3	Recuperar todas las alarmas de localización.		Lista con las alarmas de localización ordenadas descendentemente por fecha de creación.	Listado de las alarmas de localización ordenado.
LADAO4	Recuperar alarma de localización existente a partir de su ID.	ID de la alarma de localización	Se obtiene la alarma de comprobación correspondiente con sus datos.	Alarma de localización con el ID indicado.
LADAO5	Habilitar una alarma de comprobación y luego deshabilitarla.	Nuevo estado e ID de la alarma	Se actualiza el estado de la alarma de localización de ID indicado.	Alarma de localización actualizada.
LADAO6	Eliminar todas las alarmas de localización.		Se eliminan todas las alarmas de localización y se devuelve el número de eliminaciones.	Todas las alarmas de localización eliminadas.
LADAO7	Eliminar alarma por ID.	ID de la alarma de localización	Se elimina únicamente la alarma de localización de ID indicado.	Alarma de localización con el ID indicado eliminado.
LADAO8	Obtener colisiones con alarmas que coinciden exactamente.	Inicio: "12:00" Fin: "13:00"	Lista con las alarmas que coinciden exactamente con esas horas de inicio y fin.	Alarma de localización con la que colisiona.
LADAO9	Obtener colisiones con alarmas que coinciden por el interior.	Inicio: "12:20" Fin: "12:30"	Lista con las alarmas que coinciden por el interior con esas horas de inicio y fin.	Alarma de localización con la que colisiona.
LADAO10	Obtener colisiones con alarmas que coinciden por el exterior.	Inicio: "11:30" Fin: "13:30"	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.	Alarma de localización con la que colisiona.
LADAO11	Obtener colisiones con alarmas que coinciden por la izquierda.	Inicio: "11:30" Fin: "12:30"	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.	Alarma de localización con la que colisiona.
LADAO12	Obtener colisiones con alarmas que coinciden por la derecha.	Inicio: "12:30" Fin: "13:30"	Lista con las alarmas que coinciden por el exterior con esas horas de inicio y fin.	Alarma de localización con la que colisiona.

LADAO13	Obtener colisiones con múltiples alarmas, añadiendo una alarma auxiliar que genera otra colisión.	Inicio: "12:30" Fin: "13:30"	Lista con las dos alarmas que colisionan.	Dos alarmas de localización con las que se colisiona.
LADAO14	Obtener colisiones con alarmas que no coinciden con las horas indicadas.	Inicio: "11:00" Fin: "11:30"	Lista vacía porque no hay colisiones.	Lista vacía.

Tabla 10.7. Casos de prueba junto con los resultados obtenidos para el DAO de alarmas de comprobación.

RiskContactDao				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
RCADAO1	Insertar una alarma de comprobación y luego recuperarla.	Alarma de comprobación	Se inserta la alarma de comprobación correctamente.	Alarma de comprobación insertada.
RCADAO2	Actualizar varias alarmas de comprobación con nuevos valores.	Alarmas de comprobación actualizadas	Se actualizan correctamente las alarmas de comprobación indicadas con los nuevos valores.	Alarmas de comprobación actualizadas.
RCADAO3	Eliminar una alarma de comprobación a partir de su ID.	ID de la alarma de comprobación	Se elimina únicamente la alarma de comprobación de ID indicado.	Alarma con el ID indicado eliminada.
RCADAO4	Obtener todas las alarmas de comprobación existentes.		Listado con todas las alarmas de comprobación almacenadas en el dispositivo.	Lista con todas las alarmas de comprobación.
RCADAO5	Obtener todas aquellas alarmas establecidas para una hora indicada en la que no hay ninguna alarma.	Hora: "12:00"	Lista vacía porque no hay ninguna alarma establecida para esa hora.	Lista vacía.
RCADAO6	Obtener todas aquellas alarmas establecidas para una hora indicada en la que existen varias alarmas.	Hora: "13:00"	Lista con las alarmas que han sido establecidas en la hora indicada.	Listado con las dos alarmas de comprobación establecidas en la hora indicada.

Tabla 10.8. Casos de prueba junto con los resultados obtenidos para el DAO de contactos de riesgo.

RiskContactDao				
Código	**Procedimiento**	**Entradas**	**Resultado esperado**	**Resultado obtenido**
RCDAO1	Insertar un resultado de una comprobación con sus contactos de riesgo.	Resultado de comprobación	Se inserta el resultado correctamente junto con sus contactos de riesgo.	Resultado insertado correctamente con sus contactos de riesgo.
RCDAO2	Buscar un resultado de una comprobación a partir de su ID.	ID del resultado de la comprobación	Resultado de comprobación de ID pasado como parámetro.	Resultado de la comprobación de ID indicado.
RCDAO3	Obtener todos los resultados de comprobaciones con sus contactos de riesgo.		Listado con todos los resultados de comprobación y sus respectivos contactos de riesgo.	Lista con los dos resultados de comprobaciones almacenados en la base de datos.

10.2.3 Algoritmo de comprobación de contactos

Para verificar el funcionamiento del algoritmo de comprobación y de sus clases auxiliares simplemente se han utilizado los itinerarios ficticios cargados desde fichero y el *framework* de *JUnit4*, ya que se no es necesario ninguna dependencia de Android. Las siguientes tablas reflejan los casos de prueba para los componentes involucrados en la comprobación de contactos junto con los resultados obtenidos. Las columnas *SP* y *Cod* representan el código de la situación o situaciones de prueba cubiertas por el caso de prueba y el propio código del caso de prueba respectivamente.

Tabla 10.9. Casos de prueba junto con los resultados obtenidos para la clase Itinerary.

Clase Itinerary					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
I1	I1	Crear un itinerario con localizaciones divididas en un solo día.	Lista de localizaciones de un solo día.	Localizaciones agrupadas en un solo día.	Mapa con una sola entrada con las localizaciones agrupadas en ese día.
I2	I2	Crear un itinerario con localizaciones divididas en dos días.	Lista de localizaciones de dos días.	Localizaciones agrupadas en dos días.	Mapa con dos entradas con las localizaciones agrupadas en los dos días.
I3	I3	Crear un itinerario con localizaciones divididas en más de dos días.	Lista de localizaciones de más de dos días.	Localizaciones agrupadas en más de dos días.	Mapa con 4 entradas, una para cada día, agrupando las localizaciones correspondientes.

Tabla 10.10. Casos de prueba junto con los resultados obtenidos para la clase RiskContact.

Clase RiskContact					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
RC1	RC1	Añadir un punto de contacto con dos localizaciones que generen un riesgo amarillo.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo amarillo.	Contacto de riesgo con sus valores calculados correctamente.
RC2	RC2	Añadir un punto de contacto con dos localizaciones que generen un riesgo naranja.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo naranja.	Contacto de riesgo con sus valores calculados correctamente.
RC3	RC3	Añadir un punto de contacto con dos localizaciones que generen un riesgo rojo.	Configuración de la comprobación y punto de contacto.	Se recalculan los valores del contacto de riesgo para dar un riesgo rojo.	Contacto de riesgo con sus valores calculados correctamente.
RC4	RC4	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección exacta entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto con intersección exacta y con sus valores calculados correctamente.
RC5	RC5	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección interna entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto con intersección interna y con sus valores calculados correctamente.

RC6	RC6	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección externa entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto con intersección externa y con sus valores calculados correctamente.
RC7	RC7	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección derecha entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto con intersección derecha y con sus valores calculados correctamente.
RC8	RC8	Añadir dos puntos de contacto con 4 localizaciones que den lugar a una intersección izquierda entre sus fechas.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto con intersección izquierda y con sus valores calculados correctamente.
RC9	RC9	Añadir dos puntos de contacto con 4 localizaciones cuyas fechas no generen ninguna intersección.	Configuración de la comprobación y dos puntos de contacto.	Se recalculan correctamente los valores del contacto de riesgo.	Contacto de riesgo con dos puntos de contacto que no intersecan y con sus valores calculados correctamente.
RC10	RC10	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección interna.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.	Contacto de riesgo con tres puntos de contacto que generan intersección interna y con sus valores calculados correctamente.
RC11	RC11	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección externa.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.	Contacto de riesgo con tres puntos de contacto que generan intersección externa y con sus valores calculados correctamente.
RC12	RC12	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas generen una intersección derecha.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.	Contacto de riesgo con tres puntos de contacto que generan intersección por la derecha y con sus valores calculados correctamente.
RC13	RC13	Añadir 3 puntos de contacto con 6 localizaciones cuyas fechas no generen intersección.	Configuración de la comprobación y 3 puntos de contacto.	Se recalculan correctamente todos los valores del contacto de riesgo.	Contacto de riesgo con tres puntos de contacto que no generan intersección y con sus valores calculados correctamente.

Tabla 10.11. Casos de prueba junto con los resultados obtenidos para la clase RiskContactResult.

Clase RiskContactResult					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**

RCR1	RCR1	Calcular las medias de tiempo de exposición, proximidad y riesgo para un resultado con un solo contacto de riesgo. Luego obtener el contacto de mayor riesgo y ordenarlos por riesgo.	Resultado con un contacto de riesgo.	Medias de tiempo de exposición, proximidad y riesgo calculadas correctamente. Contacto de mayor riesgo y listado ordenado por riesgo.	Medias calculadas correctamente, contacto de riesgo más alto correcto y lista de contactos de riesgo ordenada de mayor a menor riesgo.
RCR2	RCR2	Calcular las medias de tiempo de exposición, proximidad y riesgo para un resultado con 3 contactos de riesgo. Luego obtener el contacto de mayor riesgo y ordenarlos por riesgo.	Resultado con 3 contactos de riesgo.	Medias de tiempo de exposición, proximidad y riesgo calculadas correctamente. Contacto de mayor riesgo y listado ordenado por riesgo.	Medias calculadas correctamente, contacto de riesgo más alto correcto y lista de contactos de riesgo ordenada de mayor a menor riesgo.

Tabla 10.12. Casos de prueba junto con los resultados obtenidos para el detector de contactos de riesgo.

Detector de contactos de riesgo					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
D1, D2	D12	Introducir dos localizaciones que no coinciden en el tiempo y otras dos que sí coinciden en el tiempo según el margen de diferencia temporal introducido. Introducir otras dos localizaciones que coinciden en el tiempo en el valor límite del margen de diferencia temporal. Comprobar otro par de localizaciones que coinciden en la hora, pero no en el día.	* Margen temporal: 5 segundos. * Pares de localizaciones.	Se devuelve true cuando ambas coinciden en el tiempo y false cuando no coinciden. En el valor límite se devuelve true.	True cuando ambas localizaciones coinciden en el tiempo y false en caso contrario.
D3, D4	D34	Introducir dos localizaciones que no coinciden en el espacio y otras dos que sí coinciden en el espacio según el margen de distancia de seguridad introducido. Introducir otras dos localizaciones que coinciden en el espacio en el valor límite del margen de distancia de seguridad.	* Margen de distancia de seguridad: 5 metros. * Pares de localizaciones.	Se devuelve true cuando ambas coinciden en el espacio y false cuando no coinciden. En el valor límite se devuelve true.	True cuando ambas localizaciones coinciden en el espacio y false en caso contrario.
D5	D5	Dada una lista de varias localizaciones, buscar la más cercana en el espacio a una localización dada. Buscar la localización más cercana a la localización objetivo con una lista vacía de localizaciones.	* Lista de localizaciones. * Localización con la que comparar la distancia.	Se devuelve la localización con menor distancia hasta la localización objetivo.	Localización más cercana en el espacio a la localización objetivo, de entre una lista de localizaciones específica.
D6	D6	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con un solo día que no coincide con el día del usuario, pero existe un posible contacto de riesgo porque hay coincidencias en el tiempo y en el espacio.	* Itinerario 1 * Itinerario 2	Lista vacía de contactos de riesgo porque son itinerarios de diferentes días.	Lista vacía de contactos de riesgo.
D7	D7	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con un solo día que no coincide con el día del usuario, y tampoco existen coincidencias en el tiempo y en el espacio en ninguno de	* Itinerario 1 * Itinerario 3	Lista vacía de contactos de riesgo porque son itinerarios de diferentes días.	Lista vacía de contactos de riesgo.

		los puntos.			
D8	D8	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, y donde existen varios puntos que coinciden solo en el tiempo.	* Itinerario 1 * Itinerario 4	Lista vacía de contactos de riesgo porque solo hay coincidencia en el tiempo.	Lista vacía de contactos de riesgo.
D9	D9	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, y donde existen varios puntos que coinciden solo en el espacio.	* Itinerario 1 * Itinerario 5	Lista vacía de contactos de riesgo porque solo hay coincidencia en el espacio.	Lista vacía de contactos de riesgo.
D10	D10	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con un solo punto de contacto entre dos localizaciones que coinciden en el tiempo y en el espacio.	* Itinerario 1 * Itinerario 6	Lista con un contacto de riesgo formado por un solo punto de contacto.	Un contacto de riesgo con un solo punto de contacto.
D11	D11	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no existen más localizaciones del positivo.	* Itinerario 7 * Itinerario 8	Lista con un contacto de riesgo formado por varios puntos de contacto.	Lista vacía de contactos de riesgo. Bug detectado en el algoritmo debido a que no se contemplaba el caso de que no existieran más localizaciones del positivo para cerrar el tramo de contacto.
D12	D12	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no existen más localizaciones del propio usuario.	* Itinerario 7 * Itinerario 9	Lista con un contacto de riesgo formado por varios puntos de contacto.	Lista con un contacto de riesgo formado por varios puntos de contacto.
D13	D13	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con varios puntos de contacto que conforman un contacto de riesgo, cuyo tramo se cierra debido a que no hay coincidencia en el tiempo y espacio en el siguiente par de localizaciones.	* Itinerario 7 * Itinerario 10	Lista con un contacto de riesgo formado por varios puntos de contacto.	Lista con un contacto de riesgo formado por varios puntos de contacto.
D14	D14	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por un solo punto de contacto y el otro formado por varios puntos de contacto que se cierra	* Itinerario 7 * Itinerario 11	Lista con dos contactos de riesgo formados por un punto y por varios respectivamente.	Lista con dos contactos de riesgo formados por un punto y por varios respectivamente.

		porque ya no hay coincidencia en el tiempo y en el espacio.			
D15	D15	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por varios puntos de contacto que se cierra porque ya no hay coincidencia en el espacio y tiempo, y el otro formado por varios puntos de contacto que se cierra porque ya no hay más localizaciones de usuario.	* Itinerario 7 * Itinerario 12	Lista con dos contactos de riesgo formados por varios puntos de contacto.	Lista con dos contactos de riesgo formados por varios puntos de contacto.
D16	D16	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con dos contactos de riesgo, uno formado por varios puntos de contacto que se cierra porque ya no hay coincidencia en el espacio y tiempo, y el otro formado por varios puntos de contacto que se cierra porque ya no hay más localizaciones de positivo.	* Itinerario 7 * Itinerario 13	Lista con dos contactos de riesgo formados por varios puntos de contacto.	Lista con dos contactos de riesgo formados por varios puntos de contacto.
D17	D17	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con el mismo día que el del usuario, con un contacto de riesgo, un par de localizaciones que coinciden en el tiempo y otro par que coincide en el espacio, pero no generan contactos de riesgo.	* Itinerario 4 * Itinerario 14	Lista con un solo contacto de riesgo porque el resto no coinciden en el tiempo y espacio.	Lista con un solo contacto de riesgo porque el resto no coinciden en el tiempo y espacio.
D18	D18	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con un solo día y otro itinerario del positivo con varios días donde uno coincide con el del usuario. En ese día existe un contacto de riesgo con varios puntos que se cierra debido a que no existen más coincidencias en el tiempo y espacio.	* Itinerario 5 * Itinerario 15	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.
D19	D19	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario con varios días y otro itinerario del positivo con un solo día que coincide con uno de los del usuario. En ese día existe un contacto de riesgo con varios puntos que se cierra debido a que no existen más coincidencias en el tiempo y espacio.	* Itinerario 15 * Itinerario 5	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.	Lista con un solo contacto de riesgo que se corresponde al del día que coinciden ambos itinerarios.
D20	D20	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario y otro itinerario del positivo ambos con varios días que coinciden. En algunos de los días que coinciden existen contactos de riesgo y en el resto de los días que coinciden no hay	* Itinerario 16 * Itinerario 17	Lista con dos contactos de riesgo, de días distintos.	Lista con dos contactos de riesgo, de días distintos.

		contactos de riesgo.			
D21	D21	Realizar la comprobación de contactos de riesgo entre un itinerario del usuario y otro itinerario del positivo ambos con varios días donde ninguno de ellos coincide y existen varios puntos de contacto que darían lugar a contactos de riesgo si coincidieran los días.	* Itinerario 16 * Itinerario 18	Lista vacía sin ningún contacto de riesgo.	Lista vacía sin ningún contacto de riesgo.

Tabla 10.13. Casos de prueba junto con los resultados obtenidos para la clase LocationAlarm.

Clase LocationAlarm					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
LA1, LA2	LA12	Crear una alarma inválida con fecha de inicio igual a la de fin, y luego crear otra alarma inválida con fecha de inicio posterior a la de fin.	* Fecha de prueba * Alarmas de localización	Se detecta que las dos alarmas son inválidas.	Se detecta que las dos alarmas son inválidas.
LA3	LA3	Crear una alarma válida que esté actualizada, con la fecha posterior a la fecha de prueba.	* Fecha de prueba * Alarma actualizada	No se modifica la alarma porque ya está actualizada.	No se modifica la alarma porque ya está actualizada.
LA4, LA5	LA45	Crear una alarma válida que esté desfasada, con la fecha igual a la fecha de prueba. Crear otra alarma desfasada con la fecha anterior a la fecha de prueba.	* Fecha de prueba * Alarmas desfasadas	Se modifican las alarmas posponiéndolas un día más que la fecha de prueba.	Se modifican las alarmas posponiéndolas un día más que la fecha de prueba.

10.2.4 Managers

Para las pruebas de los *managers* es necesario *mockear* los repositorios, por lo que se utiliza la librería de ***Mockito*** para ello. Además, se ejecuta código asíncrono con funciones *suspend* por lo que se utiliza el bloque ***runBlocking*** para envolver los métodos de prueba. Las siguientes tablas muestran los casos de prueba para los *managers* incluyendo las salidas obtenidas tras su ejecución.

Tabla 10.14. Casos de prueba junto con los resultados obtenidos para el manager de alarmas de localización.

Manager de alarmas de localización					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
LAM1, LAM2	LAM12	Establecer una alarma inválida con inicio igual a fin y luego otra alarma inválida con inicio posterior a fin.	* Alarma1 * Alarma2	Se devuelve un código de Error de alarma inválida en ambos casos.	Se devuelve un código de Error de alarma inválida en ambos casos.
LAM3	LAM3	Establecer una alarma válida y sin colisiones.	* Alarma3	Se devuelve un código de Éxito.	Se devuelve un código de Éxito.

LAM4	LAM4	Establecer una alarma válida que genera una colisión interna.	* Alarma3 * Alarma4(colisión)	Se devuelve un código de Éxito y se actualiza la alarma.	Se devuelve un código de Éxito y se actualiza la alarma.
LAM5	LAM5	Establecer una alarma válida que genera una colisión externa.	* Alarma4 * Alarma3 (colisión)	Se devuelve un código de Error de colisión de alarmas.	Se devuelve un código de Error de colisión de alarmas.
LAM6	LAM6	Establecer una alarma válida que genera una colisión izquierda.	* Alarma3 * Alarma5 (colisión)	Se devuelve un código de Error de colisión de alarmas.	Se devuelve un código de Error de colisión de alarmas.
LAM7	LAM7	Establecer una alarma válida que genera una colisión derecha.	* Alarma3 * Alarma6 (colisión)	Se devuelve un código de Error de colisión de alarmas.	Se devuelve un código de Error de colisión de alarmas.

Tabla 10.15. Casos de prueba junto con los resultados obtenidos para el manager de alarmas de comprobación.

Manager de alarmas de comprobación					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
RCAM1	RCAM1	Establecer una alarma sin que se supere el límite y sin que haya colisiones.	* Límite de alarmas * Nº de alarmas existentes: 1 * Alarma1	Se devuelve un código de Éxito.	Se devuelve un código de Éxito.
RCAM2	RCAM2	Establecer una alarma sin que se genere colisión y sin superar el límite igualándolo.	* Límite de alarmas * Nº de alarmas existentes: 2 * Alarma1	Se devuelve un código de Éxito.	Se devuelve un código de Éxito.
RCAM3	RCAM3	Establecer una alarma sin que se genere colisión y superando el límite sobrepasándolo.	* Límite de alarmas * Nº de alarmas existentes: 3 * Alarma1	Se devuelve un código de Error de límite de alarmas sobrepasado.	Se devuelve un código de Error de límite de alarmas sobrepasado.
RCAM4	RCAM4	Establecer una alarma generando colisiones.	* Límite de alarmas * Alarma que genera colisión. * Alarma11	Se devuelve un código de Error de colisión de alarmas.	Se devuelve un código de Error de colisión de alarmas.

Tabla 10.16. Casos de prueba junto con los resultados obtenidos para el manager de positivos.

Manager de positivos					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
PM1	PM1	Notificar un positivo superando el límite de notificación, es decir, aún no han transcurrido los suficientes días para permitir notificar de nuevo un positivo. Probar el valor límite de días transcurridos con la fecha del último positivo notificado.	* Configuración de la notificación. * Último positivo notificado en la fecha actual.	Se obtiene un código de Error indicando que se ha superado el límite de notificación de positivos.	Se obtiene un código de Error indicando que se ha superado el límite de notificación de positivos.
PM2	PM2	Notificar un positivo sin superar el límite de notificación porque no existen otros positivos notificados.	* Configuración de la notificación. * Localizaciones a subir.	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones subidas.

				subidas.	
PM3	PM3	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación, pero no existen localizaciones que notificar.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores.	Se obtiene un código de Error indicando la inexistencia de localizaciones que notificar.	Se obtiene un código de Error indicando la inexistencia de localizaciones que notificar.
PM4	PM4	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado exitoso.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones subidas.	Se obtiene un código de Éxito con el código del positivo y el número de localizaciones subidas.
PM5	PM5	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado de error de red.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Error de tiempo de espera agotado.	Se obtiene un código de Error de tiempo de espera agotado.
PM6	PM6	Notificar un positivo sin superar el límite de notificación porque ya han transcurrido suficientes días desde la última notificación y si existen localizaciones, obteniéndose un resultado de error HTTP.	* Configuración de la notificación. * Último positivo notificado en fechas anteriores. * Localizaciones a subir.	Se obtiene un código de Error genérico de que no es posible notificar el positivo.	Se obtiene un código de Error genérico de que no es posible notificar el positivo.

10.3 Pruebas de Integración

Las pruebas de integración de los *viewmodels* requieren del *framework* de *Android* para poder ejecutarse, sin embargo, se hace uso de ***Robolectric*** para *mockear* las dependencias de *Android* y así poder ejecutar las pruebas directamente en la máquina virtual de *Java*. También se utilizan las siguientes reglas de *JUnit4:*

- ***InstantTaskExecutorRule*** para poder trabajar con los objetos *LiveData*
- ***TestCoroutineRule*** para poder utilizar el *dispatcher* de *test* y el *scope* de *test* para ejecutar las corrutinas.
- ***MockitoRule*** para poder configurar el *runner* de *mockito* y utilizar al mismo tiempo el *runner* de *Robolectric*.

Se *mockean* los repositorios y los clientes de la *API REST* necesarios para poder ejecutar las pruebas. Además, se configura la base de datos **en memoria** de *ROOM* para que se permitan ejecutar **consultas** en el **hilo principal.**

Por otro lado, las pruebas de la *API REST* se implementan mediante el *framework* de ***Mocha***, la librería de aserciones ***Chai*** y el *plugin* ***Chai-http*** para simular las peticiones de red a la *API*. En este caso las pruebas residen en la carpeta ***test*** dentro de la raíz del proyecto, y se ejecutan en el entorno local de *Node.js*.

Como ya se comentó en la sección relativa a la ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS, estas pruebas se ejecutan utilizando una **base de datos** en la nube de ***test***, de manera que no se modifique el contenido de la base de datos de producción.

Al igual que con las pruebas unitarias, los métodos de prueba están etiquetados con los **códigos** de los casos de prueba que implementan, de forma que así se mantiene la trazabilidad entre casos de prueba y métodos de prueba.

10.3.1 ViewModels

Las siguientes tablas contienen los casos de prueba correspondientes a los distintos *viewmodels* incluyendo las salidas obtenidas una vez se han ejecutado.

Tabla 10.17. Casos de prueba junto con las salidas obtenidas para el viewmodel de notificación de positivos.

ViewModel para notificar positivos					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
NPVM1, NPVM3, NPVM4	NPVM 134	Notificar un positivo con éxito sin incluir datos personales y comprobar que se almacena correctamente. Intentar notificar otro positivo, no se podrá porque no han transcurrido suficientes días. Actualizar la fecha del positivo para que transcurran suficientes días y volver a notificar otro positivo incluyendo datos personales.	* Configuración de la notificación. * Localizaciones a subir. * Respuestas a las preguntas. * Datos personales.	Se notifica correctamente el primer positivo con código de éxito, el segundo no se notifica con código de error y el tercero se notifica con éxito incluyendo datos personales. Se filtran las localizaciones en función del Periodo de Infectividad.	Primer positivo notificado. Segundo no se notifica y el tercero se notifica.
NPVM2	NPVM 2	Notificar un positivo sin superar el límite y sin datos personales, pero sin existir localizaciones que subir.	* Configuración de la notificación. * Respuestas a las preguntas. * Lista vacía de localizaciones.	Se obtiene un código de Error indicando que no existen localizaciones para subir a la nube.	Se obtiene un código de Error indicando que no existen localizaciones para subir a la nube.
NPVM5	NPVM 5	Notificar un positivo generando un error de red.	* Configuración de la notificación. * Respuestas a las preguntas. * Localizaciones a subir.	Se obtiene un código de Error indicando Tiempo de Espera Agotado.	Necesario realizar el test a mano porque no se pueden simular las excepciones.
NPVM6	NPVM 6	Notificar un positivo generando un error HTTP.	* Configuración de la notificación. * Respuestas a las preguntas. * Localizaciones a subir.	Se obtiene un código de Error indicando que no se ha podido notificar un positivo.	Necesario realizar el test a mano porque no se pueden simular las excepciones.
NPVM7	NPVM 7	Cargar el periodo de infectividad de la configuración en el LiveData.	* Configuración de la notificación.	Se carga el LiveData con el periodo de infectividad establecido en la configuración.	Se carga el LiveData con el periodo de infectividad establecido en la configuración.

Tabla 10.18. Casos de prueba junto con las salidas obtenidas para el viewmodel de contactos de riesgo.

ViewModel para los contactos de riesgo					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
RCVM1, RCVM2, RCVM3, RCVM4	RCVM1234	Añadir una nueva alarma de comprobación con éxito. Añadir otra alarma de comprobación distinta que genere un error de inserción. Insertar una segunda alarma con éxito. Intentar añadir una tercera alarma de comprobación que genere colisión con la segunda alarma. Añadir una tercera alarma de comprobación distinta que no genere colisión. Por último, intentar añadir una cuarta alarma de comprobación que no genere colisión, no se debería insertar porque ya se ha superado el límite.	* Alarmas de comprobación. * Límite de alarmas de comprobación (3).	Se almacenan las tres alarmas de comprobación en la base de datos devolviéndose un código de Éxito. Para la alarma que genera error de inserción se devuelve el código de error de inserción. Para la alarma que genera colisión se devuelve un código de Error de colisión de alarmas y para la que supera el límite se devuelve un código de error de Límite superado.	Tres alarmas establecidas, el resto de las alarmas que generan errores, colisiones o superan el límite no son insertadas.
RCVM5, RCVM6	RCVM56	Añadir una nueva alarma de comprobación con éxito. Desactivarla y luego volver a activarla, comprobando que cambia de estado correctamente.	* Alarma de comprobación.	Se añade la alarma con éxito, luego se desactiva y activa correctamente.	Se añade la alarma con éxito, luego se desactiva y activa correctamente.
RCVM7	RCVM7	Añadir una nueva alarma de comprobación con éxito y luego eliminarla.	* Alarma de comprobación.	Se añade la alarma con éxito y luego se elimina por completo.	Se añade la alarma con éxito y luego se elimina por completo.
RCVM8	RCVM8	Ejecutar la comprobación de contactos de riesgo cuando no existen localizaciones del usuario registradas dentro del alcance de la comprobación.	* Configuración de la comprobación. * Fecha actual: “28/06/2021 17:49:00” * Itinerario 15 (usuario)	Se obtiene un resultado vacío sin contactos de riesgo.	Se obtiene un resultado vacío sin contactos de riesgo.
RCVM9	RCVM9	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local, pero no hay positivos notificados con localizaciones dentro del alcance de comprobación.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 2 (positivo)	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.

RCVM10	RCVM10	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y existe un positivo notificado con un itinerario de varios días dentro del alcance de la comprobación, pero no genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 16 (positivo)	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.	Se obtiene un resultado de comprobación vacío sin contactos de riesgo.
RCVM11	RCVM11	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y existe un positivo notificado con un itinerario de varios días dentro del alcance de la comprobación que genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo)	Se obtiene un resultado de comprobación con dos contactos de riesgo con el positivo.	Se obtiene un resultado de comprobación con dos contactos de riesgo con el positivo.
RCVM12	RCVM12	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Uno de esos positivos ha sido notificado por el propio usuario y genera contactos, pero debería ser ignorado y el otro genera contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo) * Itinerario 5 (positivo del usuario)	Se obtiene un resultado con dos contactos de riesgo porque el positivo notificado por el propio usuario genera contactos, pero no se tiene en cuenta en la comprobación.	Se obtiene un resultado con dos contactos de riesgo porque el positivo notificado por el propio usuario genera contactos, pero no se tiene en cuenta en la comprobación.
RCVM13	RCVM13	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Solo uno de los positivos genera contactos de riesgo, el resto no.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 19 (positivo que genera contactos) * Itinerario 14 (positivo) * Itinerario 1 (positivo)	Se obtiene un resultado con dos contactos de riesgo generados por un positivo porque el resto no genera contactos.	Se obtiene un resultado con dos contactos de riesgo generados por un positivo porque el resto no genera contactos.

RCVM14	RCVM14	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días almacenadas en local y hay varios positivos notificados con localizaciones dentro del alcance de la comprobación. Varios de estos positivos generan contactos de riesgo.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario) * Itinerario 20 (positivo que genera contactos) * Itinerario 5 (positivo que genera contactos) * Itinerario 19 (positivo que genera contactos)	Se obtiene un resultado con cuatro contactos de riesgo generados por varios positivos.	Se obtiene un resultado con cuatro contactos de riesgo generados por varios positivos.
RCVM15	RCVM15	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días, pero no hay positivos porque se genera un error de Tiempo de Espera agotado.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario)	No se obtiene ningún resultado porque no se ha ejecutado la comprobación debido a un error de Tiempo de espera agotado.	No se puede reproducir el error de Tiempo de Espera Agotado.
RCVM16	RCVM16	Ejecutar la comprobación de contactos de riesgo cuando existen localizaciones del usuario divididas en varios días, pero no hay positivos porque se genera un error HTTP.	* Configuración de la comprobación. * Fecha actual. * Itinerario 15 (usuario)	No se obtiene ningún resultado porque no se ha ejecutado la comprobación debido a un error HTTP.	No se puede reproducir el error HTTP.

10.3.2 API REST

Las siguientes tablas muestran los casos de prueba diseñados para las distintas *APIs REST* expuestas por el servidor *web*, incluyendo las salidas obtenidas una vez se han ejecutado.

Tabla 10.19. Casos de prueba junto con las salidas obtenidas para la API de configuración.

API de configuración					
SP	**Cod.**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
APIC1	APIC1	Recuperar la configuración de un fichero que no existe.	* Nombre de fichero que no existe: “no-existe”	* Código: 404 Not found	* Código: 404 Not found
APIC2	APIC2	Recuperar la configuración de la notificación de positivos.	* Nombre de fichero: “notify-config”	* Código: 200 OK * Objeto con la configuración de la notificación.	* Código: 200 OK * Objeto con la configuración de la notificación.

APIC3	APIC3	Recuperar la configuración de la comprobación de contactos de riesgo.	* Nombre de fichero: "risk-contact-config"	* Código: 200 OK * Objeto con la configuración de la comprobación.	* Código: 200 OK * Objeto con la configuración de la comprobación.
APIC4	APIC4	Intentar actualizar cualquier configuración con un fichero de configuración vacío y luego directamente sin enviar ningún parámetro en la petición.	* Objeto de configuración vacío.	* Código: 400 Bad Request	* Código: 400 Bad Request
APIC5	APIC5	Actualizar todos los campos de la configuración de la notificación con valores distintos.	* Objeto con la nueva configuración de la notificación de positivos.	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: 'Configuración actualizada correctamente.'}	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: 'Configuración actualizada correctamente'}
APIC6	APIC6	Actualizar todos los campos de la configuración de la comprobación con valores distintos.	* Objeto con la nueva configuración de la comprobación de contactos.	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: 'Configuración actualizada correctamente.'}	* Código: 200 OK * Respuesta con el objeto de éxito: { updated: true, msg: 'Configuración actualizada correctamente.'}

Tabla 10.20. Casos de prueba junto con las salidas obtenidas para la API de positivos.

API de positivos					
SP	**Cod**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
APIP1	APIP1	Recuperar todos los positivos notificados cuando en la base de datos hay varios positivos almacenados.	* Positivos almacenados en la BD.	* Código 200 OK * Lista con todos los positivos almacenados en la base de datos y sus datos correspondientes.	* Código 200 OK * Lista con todos los positivos almacenados en la base de datos y sus datos correspondientes.
APIP2	APIP2	Recuperar todos los positivos notificados cuando en la base de datos no hay ningún positivo almacenado.	* Base de datos sin positivos.	* Código 200 OK * Lista de positivos vacía.	* Código 200 OK * Lista de positivos vacía.
APIP3	APIP3	Intentar notificar un positivo enviando un objeto vacío en la petición.	* Objeto vacío en la petición.	* Código 400 Bad request	* Código 400 Bad request
APIP4	APIP4	Notificar un positivo enviando un objeto con localizaciones registradas en un solo día.	* Positivo con localizaciones registradas en un solo día.	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: "", uploadedLocations: 5}	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: "", uploadedLocations: 5}

APIP5	APIP5	Notificar un positivo enviando un objeto con localizaciones registradas en vario días.	* Positivo con localizaciones registradas en varios días.	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: “”, uploadedLocations: 5}	* Código 200 OK * Positivo insertado correctamente. * Objeto de respuesta: {positiveCode: “”, uploadedLocations: 5}
APIP6	APIP6	Obtener los positivos notificados con localizaciones registradas en los últimos días sin que coincida ningún positivo.	* Fecha de referencia: “05/10/2021 15:24:06” * N.º de días: 2	* Lista vacía porque no hay positivos con localizaciones registradas en esos dos últimos días.	* Lista vacía porque no hay positivos con localizaciones registradas en esos dos últimos días.
APIP7	APIP7	Obtener solo un positivo que tiene localizaciones registradas en los últimos días indicados.	* Fecha de referencia: “05/10/2021 15:24:06” * N.º de días: 3	* Lista con el positivo 2 porque tiene localizaciones registradas en el día 02 de octubre.	* Lista con el positivo 2 porque tiene localizaciones registradas en el día 02 de octubre.
APIP8	APIP8	Obtener positivos que tienen localizaciones registradas en los últimos días indicados tomando como referencia una fecha de un mes distinto.	* Fecha de referencia: “02/10/2021 15:24:06” * N.º de días: 3	* Lista con el positivo 1 y el positivo 2 porque tienen localizaciones registradas en los días indicados.	* Lista con el positivo 1 y el positivo 2 porque tienen localizaciones registradas en los días indicados.
APIP9	APIP9	Obtener varios positivos que tienen localizaciones registradas en alguno de los días indicados teniendo en cuenta la fecha de referencia.	* Fecha de referencia: “28/09/2021 15:24:06” * N.º de días: 5	* Lista con el positivo 1, el positivo 2 y el positivo 3, porque tienen localizaciones registradas en los días indicados.	Bug encontrado: al crear el array de fechas de filtro para realizar la query, no se tiene en cuenta el mes de la fecha de referencia para calcular las fechas en función del número de días atrás, sino que se crea una nueva fecha con la fecha actual. Se arregla teniendo en cuenta la propia fecha tomada como referencia para construir el resto de las fechas en función del número de días atrás a tener en cuenta.

Tabla 10.21. Casos de prueba junto con las salidas obtenidas para la API de estadísticas.

API de estadísticas					
SP	**Cod**	**Procedimiento**	**Entradas**	**Salida esperada**	**Salida obtenida**
APIS1	APIS1	Registrar una instalación enviando un objeto vacío en la petición POST.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Objeto vacío en el cuerpo de la petición.	* Código 400 Bad Request. * No se registra la instalación.	* Código 400 Bad Request. * No se registra la instalación.
APIS2	APIS2	Registrar una instalación enviando un objeto correcto en el cuerpo de la petición POST.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Nueva instalación a registrar.	* Código 200 OK. * Instalación registrada correctamente. * Objeto de respuesta: {registered: true, msg: “Instalación registrada.”}	* Código 200 OK. * Instalación registrada correctamente.

APIS3	APIS3	Registrar un resultado de comprobación enviando en el cuerpo de la petición POST un objeto vacío.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Objeto vacío en el cuerpo de la petición.	* Código 400 Bad Request. * No se registra el resultado.	* Código 400 Bad Request. * No se registra el resultado.
APIS4	APIS4	Registrar un resultado de comprobación enviando en el cuerpo de la petición POST un objeto que representa una comprobación correcta.	* Positivos notificados. * Instalaciones registradas. * Resultados registrados. * Nueva resultado a registrar.	* Código 200 OK. * Resultado registrado correctamente. * Objeto de la respuesta: {registered: true, msg: “Resultado de la comprobación registrado.”}	* Código 200 OK. * Resultado registrado correctamente. * Objeto de la respuesta: {registered: true, msg: “Resultado de la comprobación registrado.”}
APIS5	APIS5	Obtener estadísticas de positivos, instalaciones y resultados de comprobación cuando no existen datos filtrados en los últimos días indicados desde la fecha de referencia.	* Fecha de referencia: “05/10/2021 12:55:04” *Número de días: 2	* Código 200 OK * Resultados vacíos	* Código 200 OK * Resultados vacíos
APIS6	APIS6	Recuperar las estadísticas de los positivos notificados en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: “26/09/2021 12:00:00” *Número de días: 0	* Código 200 OK * Objeto con las estadísticas del número de positivos notificado hoy.	* Código 200 OK * Objeto con las estadísticas del número de positivos notificado hoy.
APIS7	APIS7	Recuperar las estadísticas de los resultados de las comprobaciones registrados en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: “26/09/2021 12:00:00” *Número de días: 0	* Código 200 OK * Un resultado de una comprobación.	* Código 200 OK * Un resultado de una comprobación.
APIS8	APIS8	Recuperar las estadísticas de las instalaciones registradas en los últimos días pasando como parámetro del número de días el valor 0, que se corresponde con el día de hoy, según la fecha de referencia.	* Fecha de referencia: “26/09/2021 12:00:00” *Número de días: 0	* Código 200 OK * Dos instalaciones registradas.	* Código 200 OK * Dos instalaciones registradas.

APIS9	APIS9	Recuperar las estadísticas de positivos notificados en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: 2	* Código 200 OK * Dos positivos notificados.	* Código 200 OK * Dos positivos notificados.
APIS10	APIS10	Recuperar las estadísticas de resultados de comprobaciones registrados en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: 3	* Código 200 OK * Dos resultados de comprobación.	* Código 200 OK * Dos resultados de comprobación.
APIS11	APIS11	Recuperar las estadísticas de instalaciones registradas en los últimos días pasando como parámetro del número de días un valor mayor que 0.	* Fecha de referencia: “28/09/2021 12:00:00” *Número de días: 2	* Código 200 OK * Tres instalaciones registradas.	* Código 200 OK * Tres instalaciones registradas.
APIS12	APIS12	Recuperar las estadísticas de positivos notificados en los últimos días pasando como parámetro del número de días un valor negativo igual a -1, que representa todos los días.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todos los positivos notificados.	* Código 200 OK * Todos los positivos notificados.
APIS13	APIS13	Recuperar las estadísticas de resultados de comprobaciones registrados en los últimos días pasando como parámetro del número de días un valor negativo igual a -1, que representa todos los días.	* Fecha de referencia: “01/10/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todos los resultados registrados.	* Código 200 OK * Todos los resultados registrados.
APIS14	APIS14	Recuperar las estadísticas de instalaciones registradas en los últimos días pasando como parámetro del número de días un valor negativo igual a -1 que representa todos los días.	* Fecha de referencia: “28/09/2021 12:00:00” *Número de días: -1	* Código 200 OK * Todas las instalaciones registradas.	* Código 200 OK * Todas las instalaciones registradas.

10.4 Pruebas del Sistema

El desarrollo de las pruebas del sistema se basa en verificar el correcto funcionamiento de la interfaz de usuario de la aplicación móvil, para lo cual se utiliza la librería de ***Espresso*** antes mencionada. Estas pruebas se ejecutan sobre el emulador *Android* cuyas características fueron

descritas en el apartado 8.6.1 DISPOSITIVOS DE PRUEBAS. Para su implementación, se utiliza la **grabadora** de *tests* de *Espresso* integrada en *Android Studio*, de forma que se genera una clase de *kotlin* con el código del *test*. Una vez grabado, se implementan los **asertos** más complejos de forma manual por código, utilizando los ***matchers*** proporcionados por la librería de *Espresso*.

Como ya se comentó, las pruebas del sistema se organizan por operaciones de alto nivel que se pueden realizar con la aplicación móvil, de manera que las clases de *test* se agrupan bajo directorios para cada caso de uso. Como se trata de pruebas **instrumentadas**, estas se almacenan en el directorio de ***androidTest***.

Antes de ejecutar los casos de prueba, se deben seguir unos pasos como **procedimiento previo** para establecer el estado inicial del sistema de modo que las pruebas sean consistentes. Estos pasos ya fueron descritos en la sección 8.6.4 sobre PRUEBAS DEL SISTEMA en el capítulo del diseño, por lo tanto, no se mostrarán de nuevo en las siguientes subsecciones.

Para simplificar, si un caso de prueba se ejecuta con éxito, es decir, la salida obtenida es igual a la esperada, se mostrará un ***OK*** en la salida obtenida, en lugar de describirla en detalle. Para los casos fallidos, sí se describirá en detalle cuál ha sido el resultado obtenido.

10.4.1 Rastreo de ubicación

La siguiente tabla muestra los casos de prueba para las pruebas del sistema de la pantalla del rastreo de ubicación, incluyendo las salidas obtenidas tras su ejecución.

Tabla 10.22. Casos de prueba junto con las salidas obtenidas para la interfaz del rastreo de ubicación.

Rastreo de ubicación				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
R1	1. Ir al rastreador. 2. Activar el servicio de rastreo. 3. Esperar 3 segundos. 4. Desactivar el servicio de rastreo.		Comprobar los textos de la vista. Comprobar que desaparece la etiqueta de lista vacía cuando se activa el servicio, así como que aparece la notificación correspondiente. Comprobar que aparecen los Snackbars de encendido y apagado del servicio.	OK
R2	1. Ir a la vista del mapa. 2. Comprobar que los marcadores están activados. 3. Desactivar los marcadores. 4. Volver a activarlos. 4. Pulsar sobre el FAB del menú de capas. 5. Comprobar que aparecen las tres capas posibles.		Se muestra la vista del mapa con las distintas opciones de capas. Comprobar el estado del Chip de marcadores.	OK

R3	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 11:10 5. Comprobar que se muestra el Snackbar de error de alarma inválida. 6. Comprobar que no se inserta ninguna alarma.	Hora de inicio: 12:15 Hora de fin: 11:10	No se inserta ninguna alarma de rastreo y se muestra un Snackbar con el error de alarma inválida.	OK
R4	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 12:45 5. Comprobar que se inserta la alarma correctamente. 6. Insertar como hora de inicio: 12:20 y como hora de fin 12:35 7. Comprobar que se muestra el Snackbar de colisión entre alarmas 8. Comprobar que no se ha insertado la alarma.	Hora de inicio: 12:15 Hora de fin: 12:45	Se inserta la primera alarma de rastreo, pero no la segunda.	OK
R5	1. Ir a la vista de alarmas de rastreo. 2. Comprobar que no hay alarmas programadas. 3. Insertar en como hora de inicio: 12:15 4. Insertar como hora de fin: 12:45 5. Comprobar que se inserta la alarma correctamente. 6. Pulsar sobre la cruz roja de la alarma. 7. Comprobar que se elimina la alarma y se muestra la etiqueta de que no hay alarmas.	Hora de inicio: 12:15 Hora de fin: 12:45	Se inserta la alarma de rastreo y luego se elimina.	OK

10.4.2 Notificación de positivos

La siguiente tabla muestra los casos de prueba para la pantalla de notificación de positivos en la aplicación móvil, incluyendo los resultados obtenidos.

Tabla 10.23. Casos de prueba junto con los resultados obtenidos para la pantalla de notificación de positivos.

Notificación de positivos				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
NP1	1. Ir a la pantalla de Notificación de Positivos. 2. Comprobar el contenido de los textos explicativos. 3. Comprobar que el Checkbox de adjuntar datos personales está desactivado. 4. Comprobar que se muestran los días del periodo de infectividad. 5. Pulsar sobre el CheckBox y comprobar que cambia su estado. 6. Pulsar sobre el botón de Política de Privacidad. 7. Comprobar que se muestra la pantalla con la política de privacidad. 8. Volver atrás.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos.	Se muestran los textos explicativos correspondientes, se muestra la política de privacidad y el número de días del periodo de infectividad establecido en la configuración del Backend.	OK

NP2	1. Ir a la pantalla de Notificación de Positivos. 2. Pulsar sobre el botón para notificar un positivo. 3. Comprobar que aparece el diálogo con las preguntas de asintomático y vacunado directamente y luego cancelar la notificación. 4. Pulsar sobre el CheckBox para adjuntar datos personales. 5. Volver a pulsar sobre el botón de Notificar un Positivo. 6. Comprobar que ahora aparece el formulario para rellenar datos personales. 7. Rellenarlo con los datos personales y pulsar Aceptar. 8. Comprobar que aparece la Cláusula de Consentimiento. 9. Cancelar la cláusula.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos. * Datos personales.	Se muestran los diálogos correspondientes con los formularios para adjuntar los datos personales solo cuando el Checkbox está activado, sino se muestra directamente las preguntas de asintomático y vacunado.	OK
NP3	1. Ir a la pantalla de Notificación de Positivos. 2. Pulsar sobre el CheckBox para adjuntar los datos personales. 3. Pulsar sobre el botón para notificar un positivo. 4. Pulsar Aceptar en el formulario de datos personales sin rellenar ningún campo. 5. Comprobar que no se avanza de diálogo y que se muestran los errores correspondientes debajo de cada campo de texto.	* Localizaciones del fichero del positivo. * Configuración de la notificación de positivos.	Comprobar que se muestran los errores de campos requeridos y que no se avanza de diálogo cuando se intenta notificar un positivo adjuntando datos personales vacíos.	OK
NP4	1. Hacer que las localizaciones leídas desde el fichero tengan fechas de hace 5 días atrás. 2. Ir a la pantalla de Notificación de Positivos. 3. Pulsar sobre el botón de Notificar un Positivo. 4. Pulsar el botón de Aceptar en el diálogo de las preguntas. 5. Comprobar que aparece un Snackbar indicando un error de que no existen localizaciones para subir a la nube.	* Configuración de la notificación de positivos.	Comprobar que se muestra el Snackbar con el error de inexistencia de localizaciones cuando se intenta notificar un positivo.	OK
NP5	1. Ir a la pantalla de notificación de positivos. 2. Pulsar sobre el botón para notificar un positivo. 3. Aceptar el diálogo modal con las preguntas. 4. Comprobar que aparece el Snackbar indicando el número de localizaciones que se han subido a la nube. 5. Pulsar de nuevo en el botón de notificar un positivo. 6. Comprobar que aparece el Snackbar indicando el error de que se ha superado el límite de notificación de positivos.	* Localizaciones del fichero de positivo. * Configuración de la notificación de positivos.	El primer positivo se notifica correctamente, mostrando un Snackbar con el número de localizaciones subidas a la nube. El segundo positivo no se notifica y se muestra un Snackbar indicando que se ha superado el límite de notificación.	OK

10.4.3 Comprobación de contactos

A continuación, se muestran los casos de prueba para las pantallas de comprobación de contactos incluyendo la salida obtenida tras su ejecución.

Tabla 10.24. Casos de prueba junto con la salida obtenida para las pantallas de la comprobación de contactos.

Comprobación de contactos				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
CC1	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Comprobar la presencia de los títulos y textos correspondientes. 3. Comprobar que el modo de comprobación manual está seleccionado. 4. Seleccionar el modo de comprobación periódica y comprobar que el otro modo se deshabilita. 5. Moverse a la pantalla de Notificación de positivos y volver a la de comprobación. 6. Comprobar que el modo de comprobación periódica sigue habilitado.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación.	Se muestran los títulos y textos explicativos de la comprobación de contactos. Se cambia de modo de comprobación correctamente, persistiendo entre cambios de pantalla.	OK
CC2	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Seleccionar el modo de comprobación periódica. 3. Insertar como hora de comprobación 10:30. 4. Comprobar que aparece el Snackbar de alarma programa correctamente. 5. Comprobar que aparece el Chip correspondiente a la alarma recién insertada. 6. Intentar insertar otra alarma con la misma hora 10:30. 7. Comprobar que aparece el Snackbar de error de colisión de alarmas. 8. Seleccionar como hora de comprobación las 11:00 e insertar una segunda alarma. 9. Comprobar que aparece el Snackbar y el Chip correspondiente a la alarma insertada. 10. Insertar una tercera alarma de comprobación con la hora 16:00. 11. Seleccionar como hora de comprobación las 18:00. 12. Comprobar que aparece el Snackbar de error de que se ha superado el límite de alarmas de comprobación permitido. 13. Eliminar la segunda alarma de comprobación. 14. Insertar la cuarta alarma con hora 18:00 y comprobar que se inserta correctamente.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación. * Horas de comprobación: [10:30, 11:00, 16:00, 18:00]	Las tres primeras alarmas se insertan correctamente. La alarma que genera colisión no se inserta y la cuarta alarma tampoco porque se supera el límite. Se borra la segunda alarma y se permite insertar la cuarta alarma.	OK

CC3	1. Ir a la pantalla de comprobación de contactos de riesgo. 2. Pulsar sobre el botón para realizar una comprobación de contactos de riesgo. 3. Ir al Tab con los resultados de la comprobación. 4. Comprobar que se muestran los valores correspondientes en el item del resultado. 5. Pulsar en el item del resultado y comprobar que se navega hacia los detalles del resultado. 6. Comprobar que todos los valores que se muestran se corresponden con los esperados. 7. Ir atrás en la aplicación. 8. Comprobar que sigue existiendo el item con el resultado.	* Localizaciones del fichero del usuario. * Positivo notificado en la nube. * Configuración de la comprobación.	Se realiza la comprobación de contactos de riesgo correctamente, mostrándose el resultado con los valores calculados correctamente. Se muestran los detalles esperados del resultado.	OK

10.4.4 Histórico de localizaciones

La siguiente tabla muestra los casos de prueba diseñados para la pantalla del histórico de localizaciones incluyendo las salidas obtenidas tras su ejecución.

Tabla 10.25. Casos de prueba junto con las salidas obtenidas para la pantalla del histórico de localizaciones.

Histórico de localizaciones				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
H1	1. Ir a la pantalla del histórico de localizaciones. 2. Comprobar la presencia de la etiqueta de lista vacía. 3. Seleccionar en el campo de fechas la fecha 05/MM/2021 4. Comprobar que no hay localizaciones y que por tanto se muestra la etiqueta de lista vacía.	* Localizaciones del fichero del positivo.	No hay localizaciones en los días especificados, ni en el día actual ni en el día 05/MM/2021.	OK
H2	1. Ir a la pantalla del histórico de localizaciones. 2. Comprobar la presencia de la etiqueta de lista vacía. 3. Seleccionar en el campo de fechas la fecha 04/MM/2021 4. Comprobar la existencia de los botones de Mapa y Eliminar localizaciones. 5. Comprobar el contenido de la primera localización. 6. Seleccionar la fecha 05/MM/2021 y comprobar que se muestra de nuevo la etiqueta de lista vacía.	* Localizaciones del fichero del positivo.	Se muestran las localizaciones registradas en el día 04/10/2021 y al volver a seleccionar el día 05/10/2021 se vacía la lista mostrándose la etiqueta de lista vacía.	OK

H3	1. Ir a la pantalla del histórico de localizaciones. 2. Seleccionar en el campo de fechas la fecha 04/MM/2021 3. Comprobar la existencia de los botones de Mapa y Eliminar localizaciones. 4. Pulsar sobre el botón para Eliminar todas las localizaciones de ese día. 5. Comprobar que se vacía la lista y se muestra la etiqueta de lista vacía. 6. Cambiar de día y volver al día 04/MM/2021 para comprobar que se han eliminado correctamente.	* Localizaciones del fichero del positivo.	Se muestran las localizaciones registradas en el día 04/10/2021 y al pulsar sobre el botón de Eliminar se vacía la lista mostrándose la etiqueta de lista vacía.	OK

10.4.5 Política de privacidad

La siguiente tabla muestra los casos de prueba diseñados para la pantalla de la política de privacidad de la aplicación móvil, junto con los resultados obtenidos tras ser ejecutados.

Tabla 10.26. Casos de prueba junto con las salidas obtenidas para la pantalla de la política de privacidad.

Política de privacidad				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
PP1	1. Pulsar sobre el menú de la barra superior (tres puntos). 2. Comprobar que aparece la opción de Política de Privacidad. 3. Pulsar sobre la opción de Política de Privacidad. 4. Comprobar que aparecen los textos esperados, haciendo scroll hacia abajo cuando sea necesario.		Se muestra la Política de Privacidad de la aplicación, con las diferentes cláusulas.	OK

10.4.6 Ajustes

A continuación, se muestran los casos de prueba para la pantalla de ajustes generales de la aplicación móvil, incluyendo las salidas obtenidas tras su ejecución.

Tabla 10.27. Casos de prueba junto con las salidas obtenidas para la pantalla de ajustes generales.

Ajustes generales				
Código	**Procedimiento (Pasos)**	**Entradas**	**Salida esperada**	**Salida obtenida**
A1	1. Abrir el menú de la barra superior (tres puntos). 2. Pulsar sobre la opción de Ajustes. 3. Comprobar el texto y los valores de la configuración según los datos de entrada.	* Configuración establecida: - Intervalo de tiempo: 3 sec - Despl. Mínimo: 0 m - Alcance de la comprobación: 3 días - Notificaciones de positivos: false	Se muestra la pantalla de ajustes de la aplicación. Los textos y títulos de los parámetros de configuración son correctos y sus valores se corresponden con la configuración establecida.	OK

A2	1. Ir a los ajustes de la aplicación. 2. Cambiar el intervalo de tiempo y comprobar que se ha cambiado correctamente. 3. Cambiar el desplazamiento mínimo y comprobar que se ha cambiado correctamente. 4. Cambiar el alcance de la comprobación y comprobar que se ha cambiado correctamente. 5. Modificar la preferencia de envío de notificaciones y comprobar que se ha cambiado correctamente.	* Configuración establecida: - Intervalo de tiempo: 3 sec - Despl. Mínimo: 0 m - Alcance de la comprobación: 3 días - Notificaciones de positivos: false * Nueva configuración: - Intervalo de tiempo: 2 min 5 sec - Despl Mínimo: 3 m - Alcance: 1 día - Notificaciones de positivos: true	Se muestra la pantalla de ajustes de la aplicación. Al cambiar las preferencias, se actualizan correctamente los valores de las preferencias en las vistas.	OK

10.5 Pruebas de Usabilidad

En la sección 8.6.5. PRUEBAS DE USABILIDAD se diseñaron las pruebas de usabilidad para la aplicación móvil y el panel de control *web* a través de una serie de cuestionarios que trataban aspectos generales y relativos a la usabilidad. Dichos cuestionarios se han implementado por medio de la herramienta de ***Microsoft Forms*** para facilitar su distribución, así como la recopilación de las **respuestas** aportadas por los individuos de prueba. Estos individuos de prueba constituyen una **muestra** de la población final a la que va destinado el sistema, la cual se describe a continuación:

- 1 enfermera (personal sanitario)
- 1 administrador de sistemas
- 2 personas entre 18 y 30 años
- 2 personas entre 30 y 50 años

Las respuestas de estos individuos a los cuestionarios han sido recopiladas y se muestran en las siguientes subsecciones para finalmente contrastarlas y así obtener las **conclusiones** relativas a la usabilidad del sistema.

10.5.1 Resultado de las preguntas de carácter general

Tabla 10.28. Resultado del cuestionario de preguntas generales.

¿Usa un dispositivo móvil frecuentemente?
1. Todos los días – **83 %** 2. Varias veces a la semana – **0 %** 3. Ocasionalmente – **17 %** 4. Nunca o casi nunca – **0 %**
¿Qué tipo de actividades realiza con el dispositivo móvil?
1. Es parte de mi trabajo o profesión – **0 %** 2. Lo uso principalmente para ocio (juegos, música…) – **33 %** 3. Lo uso principalmente para las redes sociales y mensajería instantánea – **66 %** 4. Únicamente leo el correo y navego ocasionalmente por *internet* – **0 %**
¿Ha usado alguna vez software para el rastreo de COVID-19 (Ej. *Radar Covid*)?
1. Sí, he utilizado *Radar Covid* – **17 %** 2. Sí, he utilizado un software similar pero no *Radar Covid* – **0 %** 3. No, pero me lo he planteado – **33 %** 4. No, nunca – **50 %**
¿Qué busca Vd. Principalmente en una aplicación móvil para el rastreo de COVID-19?
1. Que sea intuitiva y fácil de usar – **17 %** 2. Que sea muy rápida, precisa y exhaustiva detectando contactos de riesgo a costa de un mayor consumo de batería – **33 %** 3. Que consuma menos batería a costa de ser menos precisa y exhaustiva detectando contactos de riesgo – **0 %** 4. Que respete lo máximo posible mi privacidad – **50 %**

Observando las respuestas a estas preguntas, se aprecia que prácticamente la mayoría de los entrevistados utiliza su dispositivo móvil **todos los días**, y aproximadamente la mitad de ellos lo emplea principalmente para **ocio** y la otra mitad para **mensajería** instantánea y **redes sociales**. Además, respecto a la tercera pregunta, la gran mayoría no ha utilizado nunca un *software* para el rastreo de *COVID*, aunque la mitad de ellos se lo ha planteado. Solo ha habido un entrevistado que ha hecho uso de ***Radar Covid***. Por último, las respuestas a la cuarta pregunta son interesantes ya que están bastante repartidas. Un 33 % de los entrevistados prefiere que la aplicación sea muy **rápida** y **exhaustiva** detectando contactos de riesgo a costa de consumir mayor batería. Otro 50 % prefiere que respete al máximo su **privacidad**, y solo un 17 % que sea **intuitiva** y **fácil** de usar. Es interesante ver como a ninguno de los entrevistados le interesa el **ahorro de batería** de su dispositivo móvil.

10.5.2 Resultado de las preguntas cortas acerca de la aplicación

Tabla 10.29. Resultados del cuestionario de preguntas cortas sobre la aplicación móvil.

Facilidad de Uso de la Aplicación Móvil	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?	**67 %**	**17 %**	**17 %**	**0 %**
¿Existe ayuda para las funciones en caso de que tenga dudas?	**67 %**	**33 %**	**0 %**	**0%**
¿Le resulta sencillo el uso de la aplicación?	**33 %**	**50 %**	**17 %**	**0%**
Funcionalidad de la Aplicación Móvil	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. espera?	**67 %**	**33 %**	**0 %**	**0%**
¿El tiempo de respuesta de la aplicación es muy grande?	**17 %**	**0 %**	**0 %**	**83%**
Calidad de la Interfaz de la aplicación móvil				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es	**50 %**	**50 %**	**0 %**	**0 %**
Los iconos e imágenes usados son	**33 %**	**67 %**	**0 %**	**0%**
Los colores empleados son	**33 %**	**67 %**	**0 %**	**0%**
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		**83 %**	**0 %**	**17 %**
¿El diseño de las pantallas es claro y atractivo?		**67 %**	**0 %**	**33 %**
¿Cree que la aplicación está bien estructurada?		**83 %**	**0 %**	**17 %**
Observaciones				
- **En la parte de alarma, faltaría una explicación de cómo funciona exactamente ese tipo de alarma.** - **Especificar mejor el número de veces que se ha estado con un contacto positivo(por ejemplo, poner número de coincidencias o número de veces en vez de contactos de riesgo)** - **Diseño muy dinámico, muy adecuado para una app móvil. Si tuviese que proponer algún cambio tal vez sugeriría añadir un efecto más llamativo en el momento en el que la alarma finaliza, por ejemplo, que apareciese algún símbolo más llamativo. Pero ya digo, app excepcional.**				

En cuanto a la aplicación móvil, los resultados sobre su facilidad de uso y de la calidad de la interfaz son bastante favorables, tal y como se aprecia en la tabla superior. En general, la aplicación móvil resulta más o menos fácil de utilizar para los entrevistados. Además, la calidad de la interfaz de usuario también parece satisfacer lo suficiente a los entrevistados. En cuanto a las **observaciones** aportadas, se pueden sacar varias conclusiones:

- En la programación de alarmas de localización, no existe ningún texto descriptivo que indique para qué sirven dichas alarmas, por lo que sería interesante incluir uno para que el usuario sepa para qué son esas alarmas.
- A la hora de visualizar los resultados de una comprobación, la **enfermera entrevistada** mencionó que el término ***contactos de riesgo*** puede llevar a un malentendido dado que en la jerga médica eso se entiende como las personas que han estado en contacto cercano con un positivo, mientras que dentro de la aplicación se entiende como una **coincidencia** en el **espacio** y el **tiempo** con un positivo. Sería interesante refactorizar esos términos y la lógica de negocio para aclarar esta ambigüedad.
- Cuando finaliza una alarma de localización, parece ser que no hay una señal o indicador llamativo de que esto ha sucedido. Se podría hacer más énfasis con animaciones y algún mensaje por pantalla que marquen el inicio y el fin de ciertas acciones.

Tabla 10.30. Resultados del cuestionario de preguntas cortas sobre el panel de control web.

Facilidad de Uso del panel de control *Web*	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?	**67 %**	**33 %**	**0 %**	**0 %**
¿Existe ayuda para las funciones en caso de que tenga dudas?	**67 %**	**17 %**	**17 %**	**0%**
¿Le resulta sencillo el uso de la aplicación?	**83 %**	**17 %**	**0 %**	**0%**
Funcionalidad del panel de control *Web*	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. espera?	**67 %**	**33 %**	**0 %**	**0 %**
¿El tiempo de respuesta de la aplicación es muy grande?	**17 %**	**17 %**	**0 %**	**67 %**
Calidad de la Interfaz del panel de control *web*				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es	**33 %**	**67 %**	**0 %**	**0 %**
Los iconos e imágenes usados son	**33 %**	**67 %**	**0 %**	**0%**
Los colores empleados son	**33 %**	**67 %**	**0 %**	**0%**
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		**83 %**	**0 %**	**17 %**
¿El diseño de las pantallas es claro y atractivo?		**83 %**	**0 %**	**17 %**
¿Cree que la aplicación está bien estructurada?		**83 %**	**0 %**	**17 %**
Observaciones				
- **Diseño intuitivo, fácil de usar incluso con un nivel bajo de informática.**				

En cuanto a la aplicación *web*, los resultados son muy similares a los obtenidos con la aplicación móvil observando las respuestas en la tabla superior. Los entrevistados parecen estar satisfechos en general con el manejo de la aplicación *web* y con su calidad de interfaz.

Solo hay una observación de uno de los entrevistados que indica que incluso con un nivel bajo de informática, el panel de control *web* resulta fácil e intuitivo de usar.

10.6 *Bugs* encontrados

Tras ejecutar los casos de prueba diseñados por primera vez, se detectan una serie de fallos en el sistema como consecuencia de errores de programación. Este apartado tiene como objetivo recopilar los ***bugs*** encontrados a través de los casos de prueba y exponer una posible solución para solventarlos. Cabe destacar que estos no son los únicos fallos detectados durante todo el desarrollo del proyecto, sino que son los más graves y que mayor impacto tienen en el comportamiento del sistema. El resto de fallos encontrados son insignificantes y apenas tienen un alcance notable en el sistema, por lo que fueron solucionados rápidamente.

En la siguiente página se muestra una tabla con los distintos fallos detectados, indicando para cada uno el **código** del caso de prueba mediante el cual fue detectado, una breve **descripción** explicativa y una posible **solución** para corregirlo.

Tabla 10.31. Bugs encontrados tras la ejecución de los casos de prueba diseñados.

Código del Caso de Prueba	Descripción	Posible solución
D11 – Detector	No se detectan los contactos de riesgo entre dos itinerarios si se ha abierto un tramo de contacto y uno de los itinerarios ya no tiene más puntos que examinar. Esto ocurre debido a que no se cierra el tramo de contacto porque ya no existen localizaciones para examinar entonces se finaliza el bucle del algoritmo sin cerrar el contacto de riesgo y por tanto no se añade a la lista. Solo se cierran los tramos de contacto cuando ya no hay proximidad en el tiempo y en el espacio entre dos puntos de los itinerarios.	Implementar un bloque *if-else* para comprobar si hay un tramo de contacto pendiente de ser cerrado, de forma que, aunque se salga del bucle, si se han detectado puntos de contacto, estos se tengan en cuenta para formar un contacto de riesgo. Así, aunque se terminen los puntos que examinar, siempre se garantizará que se cierran los contactos de riesgo pendientes que están abiertos.
APIP9 - apipositive	No se crea correctamente el *array* de fechas utilizado como filtro para recuperar los positivos notificados que tengan localizaciones registradas en las fechas especificadas en dicho *array*. Esto ocurre porque no se tiene en cuenta el mes de la fecha de referencia utilizada para comparar, sino que se crea un nuevo objeto *Date* con la fecha actual en ese momento, entonces el mes no se corresponde con el de la fecha de referencia sino con el de la fecha actual.	En lugar de crear un nuevo objeto *Date* con la fecha actual, tener en cuenta los días, el mes y el año de la fecha de referencia para construir el resto de las fechas del *array* utilizado para filtrar, contemplando el número de días atrás indicado.

Capítulo 11. Manuales del Sistema

En este capítulo se presentan los manuales del sistema que proporcionan la información necesaria para **instalar** y **desplegar** los subsistemas, así como una guía de **uso** de la aplicación móvil y del panel de control *web*.

11.1 Manual de Instalación

Las siguientes subsecciones describen cómo instalar y poner en marcha la aplicación móvil *Android* de rastreo y el servidor *web* junto con la aplicación *web*, los cuales se despliegan de manera muy similar.

11.1.1 Aplicación móvil de rastreo

Partiendo de los ***APK's*** contenidos en el entregable del proyecto, para instalar la aplicación móvil de rastreo, es necesario disponer de un dispositivo con sistema operativo ***Android***, con una versión comprendida entre la ***API 24 (Nougat)*** y la ***API 30 (Android 11)***. A continuación, se describen los diferentes *APK's* generados que se incluyen en el entregable:

- ***ContactTracker_DEBUG_LOCAL***. Versión de la aplicación móvil que realiza llamadas a un servidor desplegado en local (*localhost*) al puerto 8080. Para ello, utiliza una dirección *IP* especial diseñada para redirigir el tráfico desde los emuladores *Android* a la máquina *host* que los hospeda como máquinas virtuales, de modo que la *URL* base a la que se conecta tiene esta forma: ***http://10.0.2.2:8080***.
- ***ContactTracker_DEBUG_AZURE***. Versión de la aplicación móvil que se conecta al servidor *web* de *Contact Tracker* desplegado en remoto en la plataforma de *Azure*. Para ello, realiza las peticiones sobre esta *URL* base: ***https://contacttrackerbackend.azurewebsites.net***.
- ***ContactTracker_DEBUG_HEROKU***. Versión de la aplicación móvil utilizada a modo de ***backup*** o respaldo por si hubiera algún problema con el servidor de *Azure*. Se conecta al servidor *web* desplegado en remoto sobre la plataforma de *Heroku*. Realiza las peticiones sobre la *URL* base: ***https://contacttrackerbackend.herokuapp.com***.

Todos estos *APK's* están generados con la variante de ***DEBUG*** dado que no serán desplegados en la *Play Store*, pues es necesario disponer de una ***licencia*** de *Google Play* para firmar las versiones de *RELEASE* de la *APK*. Para instalar cualquiera de estos *APK's* en el dispositivo *Android* se deben seguir una serie de **pasos**:

1. Descargar o transferir el *APK* deseado al dispositivo.
2. Habilitar en los ajustes del dispositivo la instalación de *APK's* externos desde **fuentes desconocidas** (por seguridad esta opción suele estar desactivada por defecto).
3. Pulsar sobre el *APK* para instalarlo en el dispositivo.

11.1.1.1 *Generar un APK manualmente*

Si por algún casual se desea **generar manualmente** un *APK* de la aplicación móvil, es necesario partir del **código fuente** de la aplicación y abrirlo desde el *IDE* de *Android Studio* para seguir los siguientes pasos:

1. Seleccionar la variante de *build* deseada. La vista de variantes suele estar situada en la esquina inferior izquierda del *IDE* de *Android Studio* (está en una pestaña, puede estar colapsada).

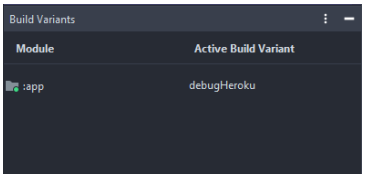


Ilustración 11.1. Vista de selección de variante de Build de la aplicación móvil.

2. En la opción de menú de ***Build***, situada en el menú superior de *Android Studio,* ir a ***Build Bundle(s) / APK (s)*** y pulsar sobre ***Build APK (s)***.

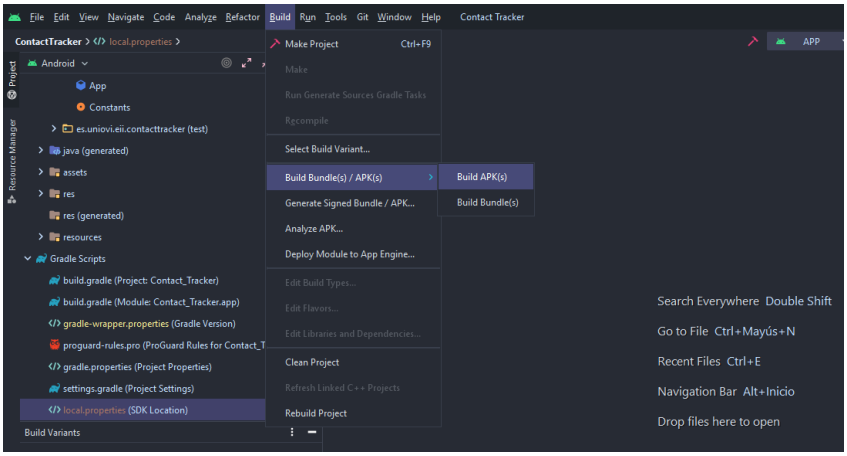


Ilustración 11.2. Opción de menú para generar una APK de la aplicación móvil.

3. Una vez generado, *Android Studio* mostrará un panel de alerta en la esquina inferior derecha con un enlace para localizar el *APK* recién generado en el explorador de archivos. Pulsar sobre ***locate***.

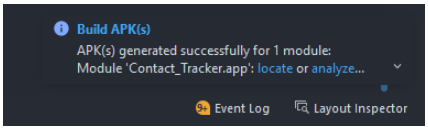


Ilustración 11.3. Alerta de Android Studio con el enlace para localizar el APK generado.

4. Al localizar el *APK*, se abrirá la carpeta contenedora que contiene el fichero *.apk* listo para ser distribuido.

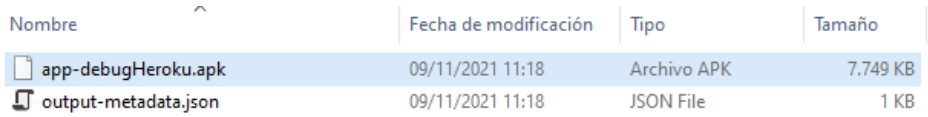


Ilustración 11.4. Contenido de la carpeta contenedora con el APK generado.

11.1.2 Despliegue del servidor *web* y de la aplicación *web*

Dado que el servidor *web*, que expone la *API REST,* y la aplicación *web*, que sirve de panel de control para el personal sanitario, se ejecutan sobre un entorno de ***node.js***, la forma de instalar y desplegar estos módulos es muy similar. Las siguientes subsecciones muestran, en primer lugar, como realizar un despliegue **local** de ambos subsistemas, y luego, cómo desplegarlos a través de la plataforma de *Azure*. Antes de proceder con la instalación y el despliegue, a continuación, se enumeran los **requisitos previos**:

- Instalar el entorno de ejecución de ***Node.js*** en el equipo.
- Instalar el *IDE* de *Visual Studio Code* u otro similar.

Requisitos parar el despliegue en la plataforma en la nube de ***Azure***:

- Disponer de una cuenta registrada de *Azure.*
- Contratar el plan gratuito *Tier-1* de *Azure* para consumir los servicios de despliegue en la nube.
- Crear un contenedor de *Azure* para conectarse con los servicios en la nube.
- Crear un ***Azure App Service*** para albergar el servidor *node.js* de *Contact Tracker* y una **cuenta de almacenamiento** para hospedar los recursos estáticos de la aplicación *web*.
- Tener el código del servidor *web* y de la aplicación *web* albergado en un repositorio de *GitHub*.

11.1.2.1 *Despliegue local*

Tanto para instalar la aplicación *web* como el servidor *web*, se deben seguir estos pasos comunes a ambas instalaciones:

1. Crear la carpeta contenedora que albergará el proyecto.
2. Descargar el *ZIP* con el código fuente y extraerlo en dicha carpeta.
3. Desde una *CLI* ejecutar el comando ***npm install***, el cual descargará e instalará todas las dependencias contenidas en el ***package.json***.

Tras ejecutar estos pasos para el servidor *web* y la aplicación *web*, ambos proyectos estarán ya instalados y listos para ser ejecutados.

Para ejecutar el **servidor *web*** existen diferentes comandos de ***script*** incluidos en el *package.json* que se describen a continuación (ejecutar uno de ellos):

- ***npm run test (****`cross-env NODE_ENV=test mocha --timeout 10000`****)***: Ejecuta los *tests* de mocha para probar la API REST, con un tiempo de espera máximo de 10

segundos. La variable de entorno *NODE_ENV* recibe el valor de *test* para indicar que se utilice la base de datos en la nube de *test*, para no alterar los datos de producción.

- ***npm run startDev (***`cross-env NODE_ENV=production nodemon app.js`***)***: inicia el servidor *node.js* con *hot-reloads* para el desarrollo, es decir, cada cambio realizado en el código obliga al servidor a actualizarse para reflejar los cambios automáticamente. La variable *NODE_ENV* recibe el valor de *production* para que se haga uso de la base de datos de producción, con los datos reales.
- ***npm run startTest (***`cross-env NODE_ENV=test nodemon app.js`***)***: mismo cometido que el comando anterior, pero utilizando la base de datos central de *test*, con los datos de prueba para evitar alterar los datos reales de producción.
- ***npm run start (***`REMOTE_DEPLOY=Heroku node app.js`***)***: comando ejecutado por la máquina remota para el despliegue del servidor en la nube. Se inicia el servidor *web* sin *hot-reloads* y con una variable de entorno *REMOTE_DEPLOY* indicando si el despliegue se realiza con *Azure* o con *Heroku*. Esta variable es necesaria debido a lo que se comentaba en el apartado 9.4.1.6 sobre el desfase temporal entre la hora local y la de los servidores.

En cuanto a la **aplicación *web***, existen los siguientes comandos de *script*:

- ***npm run serve (***`vue-cli-service serve`***)***: este es el comando que hay que ejecutar para iniciar la aplicación *web* en local. Inicia el servidor local de desarrollo de *vue* para proporcionar los ficheros y los recursos de la aplicación *web.*
- ***npm run build (***`vue-cli-service build`***)*: comando ejecutado por la máquina remota que hospeda la aplicación *web* en la nube para generar el empaquetado de la aplicación *web*.

Al ejecutar alguno de estos comandos en la máquina local, se puede acceder a la aplicación *web* a través de un **navegador *web*** conectándose a la *URL* indicada en la consola.

11.1.2.2 *Despliegue en Azure*

Para desplegar el servidor *web* y la aplicación *web* en la plataforma de *Azure*, además de los requisitos previos comentados antes, es necesario crear un **proyecto** en ***Azure DevOps***. Desde esta plataforma, se gestionan las diferentes ***builds*** y ***releases*** de ambos subsistemas a través de las denominadas ***pipelines*** de *Azure*. Una vez creado el proyecto, los pasos a seguir para crear los flujos de despliegue son análogos para la aplicación *web* y el servidor:

1. Crear una *pipeline* para que extraiga el código de *GitHub* y genere un empaquetado *ZIP*. Una vez empaquetado el código, publicarlo como un **artefacto** para que sea consumido por las *pipelines* de *release*.
2. Recoger el artefacto recién publicado con una *pipeline* de *release* que se encarga de realizar el despliegue en el *Azure App Service* o en la cuenta de almacenamiento en función de si se trata del servidor *web* o de la aplicación *web*.

11.1.2.2.1 Desplegar servidor *web*

La ***pipeline* de *build*** para extraer el código de *GitHub* y generar el fichero *ZIP* empaquetado para el servidor *web* debe tener este aspecto:

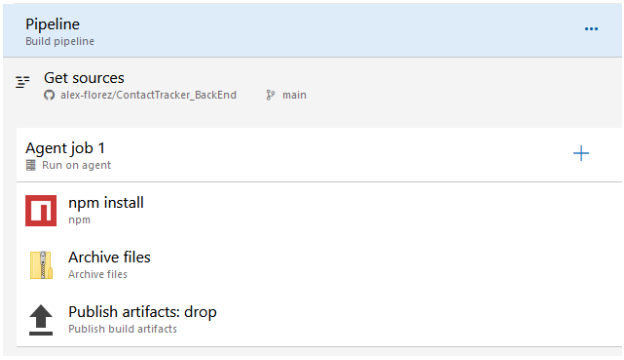


Ilustración 11.5. Pipeline de build para el servidor web.

En primer lugar, se ejecuta el comando ***npm install*** para instalar las dependencias del servidor *web*. Luego, se archivan los ficheros en un *ZIP* y, por último, este se publica como un artefacto. Para que estas operaciones se lleven a cabo automáticamente con cada ***push*** de código realizado sobre la rama principal (***main***) del repositorio *GitHub*, es necesario activar la **integración continua** dentro de la *pipeline* en el apartado de ***Triggers***, tal y como se muestra a continuación.

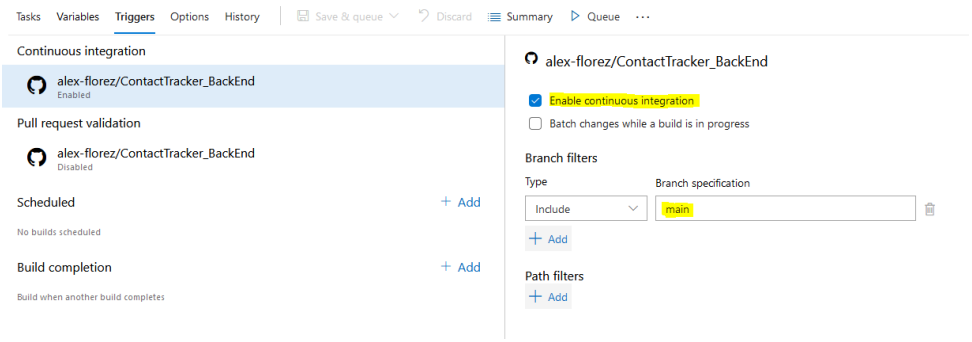


Ilustración 11.6. Habilitando la integración continua en la pipeline de build del servidor web.

La ***pipeline* de despliegue** para el servidor *web*, debe tener el siguiente aspecto:

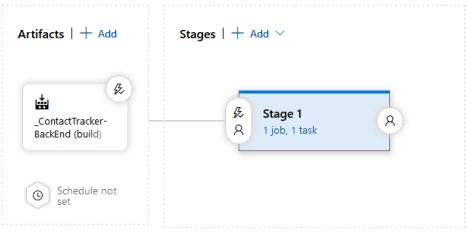


Ilustración 11.7. Pipeline de despliegue para el servidor web.

Dentro del ***job*** del ***stage***, se debe utilizar el servicio para desplegar el empaquetado sobre el *Azure App Service*, definiendo como **comando de inicio**, el comando ***npm run start*** descrito anteriormente.

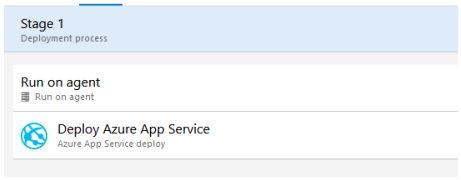


Ilustración 11.8. Tareas del job de despliegue del servidor web.

Así, con cada cambio en la rama principal que se suba al repositorio de *GitHub* se generará una nueva *release* que será desplegada automáticamente en la nube de *Azure* a través del *Azure App Service*. Para monitorizar el estado del servidor *web* y acceder a su ***URL* base**, se puede consultar el panel de control del *Azure App Service* sobre el cual está desplegado, dentro del **portal de *Azure*.**

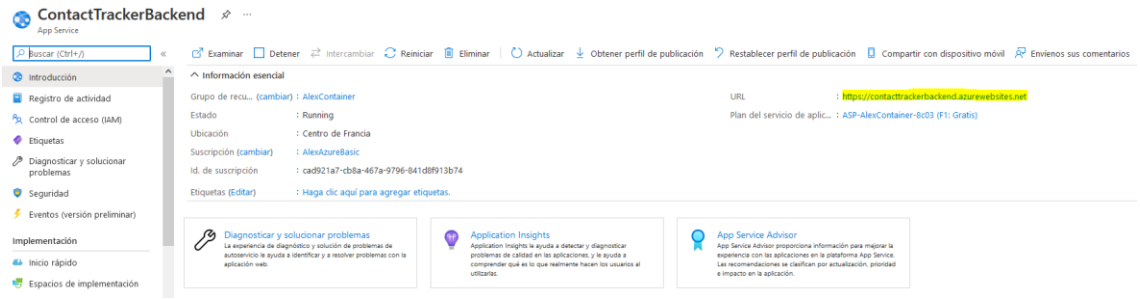


Ilustración 11.9. Panel de control del App Service de Azure para el servidor web de Contact Tracker.

11.1.2.2.2 Desplegar aplicación web

El proceso para desplegar la aplicación *web* es muy similar. A continuación, se muestra el aspecto que debería tener la ***pipeline*** de ***build*** para empaquetar el código de la aplicación *web* desde *GitHub* y publicar el artefacto correspondiente.

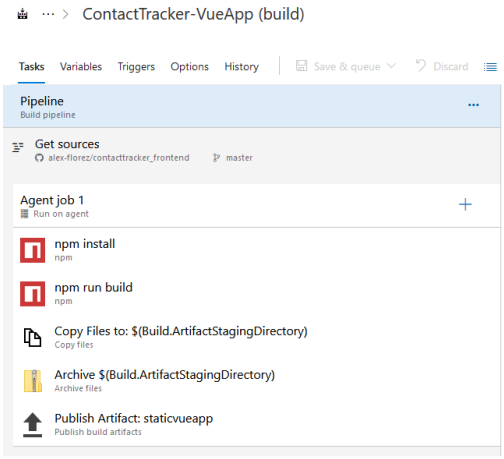


Ilustración 11.10. Pipeline de build para la aplicación web.

De nuevo, se ejecuta el comando *npm install* para instalar las dependencias, seguido del comando ***npm run build*** para generar la *build* de *Vue*, el cual fue descrito anteriormente. Luego se copian los ficheros a un directorio especial para generar el *ZIP* y publicar el artefacto.

Al igual que con el servidor *web*, es necesario activar la **integración continua** desde la pestaña de *Triggers*, para que cada *push* a la rama *master* genere un nuevo artefacto a través de esta *pipeline*.

El artefacto generado es consumido automáticamente por la ***pipeline* de *release*** que debe tener el siguiente aspecto:

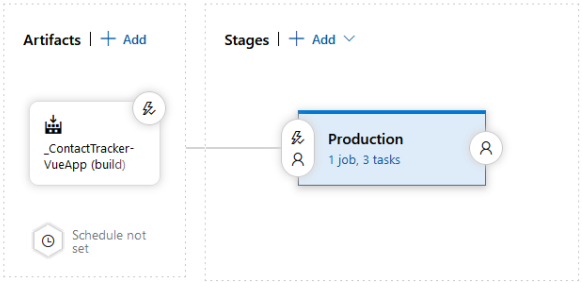


Ilustración 11.11. Pipeline de release para la aplicación web.

Dentro del ***stage* de *Production***, se debe crear un nuevo ***job*** con las siguientes tres **tareas:**

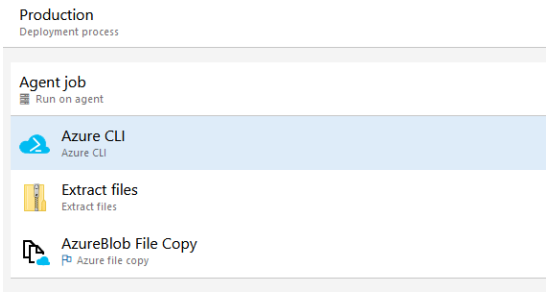


Ilustración 11.12. Tareas del Job para la pipeline de release de la aplicación web.

En primer lugar, se utiliza la consola de *Azure* para eliminar los recursos antiguos almacenados en la cuenta de almacenamiento de *Azure* creada a través del siguiente comando:

```
az storage blob delete-batch --account-name vuestaticapp --source "`$web"
```

Después, se extraen los nuevos ficheros del *ZIP* con la *build* de la aplicación *web* y por último se utiliza la tarea ***AzureBlob File Copy*** para copiar los ficheros extraídos que representan los recursos de la aplicación *web* en la **cuenta de almacenamiento** de *Azure.*

Tras ejecutarse estas dos *pipelines*, se puede monitorizar y consultar el contenido de la cuenta de almacenamiento desde el portal de *Azure*. Para acceder a la ***URL* base** del sitio *web* estático, basta con dirigirse a la opción de ***Sitio Web Estático*** dentro de la cuenta de almacenamiento, tal y como se muestra en la siguiente captura.

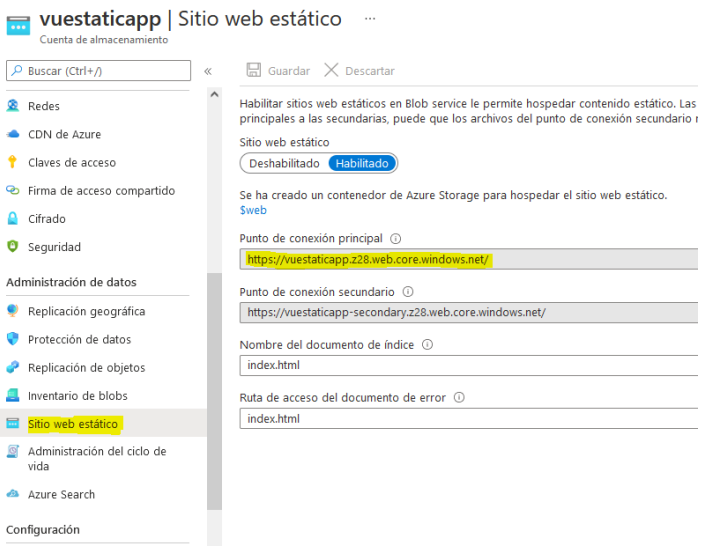


Ilustración 11.13. Panel de control del sitio web estático albergado en la cuenta de almacenamiento de Azure.

11.2 Manual de Usuario

Una vez visto la instalación y despliegue de los diferentes módulos que componen el sistema en su totalidad, en esta sección se presentan los manuales orientados a los **usuarios finales** de la aplicación móvil y del panel de control *web*. En ellos, se explicará con detalle las diferentes operaciones que se pueden realizar con la aplicación móvil y la aplicación *web* y cómo llevarlas a cabo paso a paso.

11.2.1 Usuario estándar de la aplicación móvil

Las presentes subsecciones describen cómo llevar a cabo los diferentes casos de uso de la aplicación móvil desde la perspectiva de un usuario estándar, es decir, un ciudadano de la población que se descarga la aplicación en su móvil.

11.2.1.1 *Rastrear la ubicación*

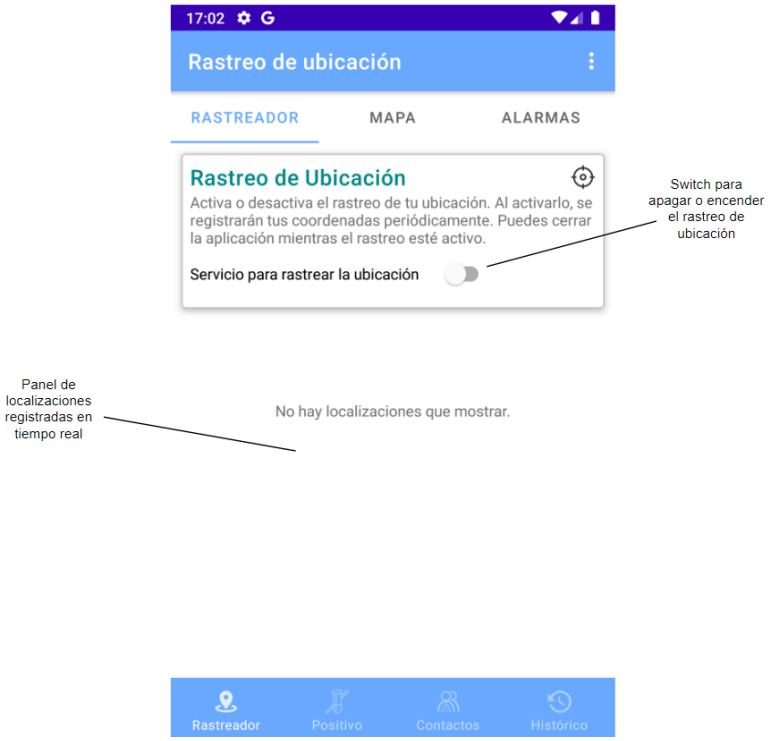


Ilustración 11.14. Pestaña del rastreador en la pantalla del rastreo de ubicación.

La pantalla principal que se muestra al abrir la aplicación es la del **rastreador,** accesible a través del **menú inferior** en la primera opción Rastreador, el cual permite llevar a cabo el registro de las coordenadas del usuario de manera periódica para guardarlas en el almacenamiento local del dispositivo. Esta pantalla está formada por **tres pestañas**:

1. **Rastreador.** Permite iniciar o detener el servicio de rastreo de ubicación.
2. **Mapa.** Permite visualizar el rastreo de ubicación en tiempo real en un mapa.
3. **Alarmas.** Permite programar alarmas para ejecutar el rastreo a horas determinadas.

La ILUSTRACIÓN 11.14 muestra el aspecto de la primera pestaña del Rastreador. Para **activar o desactivar** el **servicio de rastreo de ubicación** basta con pulsar sobre el ***switch*** que se muestra en el panel superior. Al iniciarse el servicio de rastreo, se mostrará una **notificación** indicando que se está registrando la ubicación cada cierto tiempo aproximadamente. Esta notificación no se puede eliminar y se mostrará mientras el servicio de rastreo esté activo. Al detener el servicio esta notificación será eliminada. Además, mientras el rastreo esté activo, se pueden consultar las coordenadas que están siendo registradas en tiempo real a través del panel inferior tal y como se muestra en la siguiente imagen de la misma pestaña del Rastreador, pero ahora con el servicio encendido.

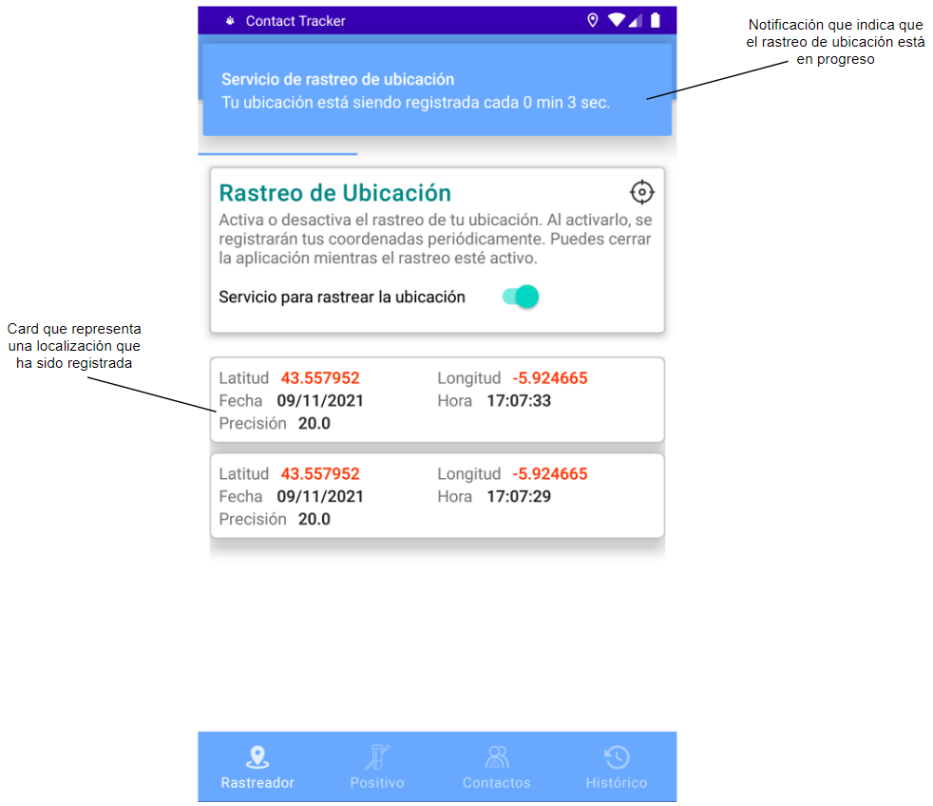


Ilustración 11.15. Pestaña del Rastreador con el servicio de rastreo de ubicación activo.

Como se puede apreciar, las coordenadas que se van registrando irán apareciendo en el listado inferior, mostrando para cada una de ellas su **latitud**, **longitud**, **fecha** y **hora** de registro y la **precisión** con la que se tomó esa coordenada. Además de poder visualizar las coordenadas en tiempo real en forma de lista, también se pueden visualizar en un **mapa** accediendo a la pestaña de **Mapa** mientras el servicio de rastreo sigue activo, tal y como se muestra a continuación.

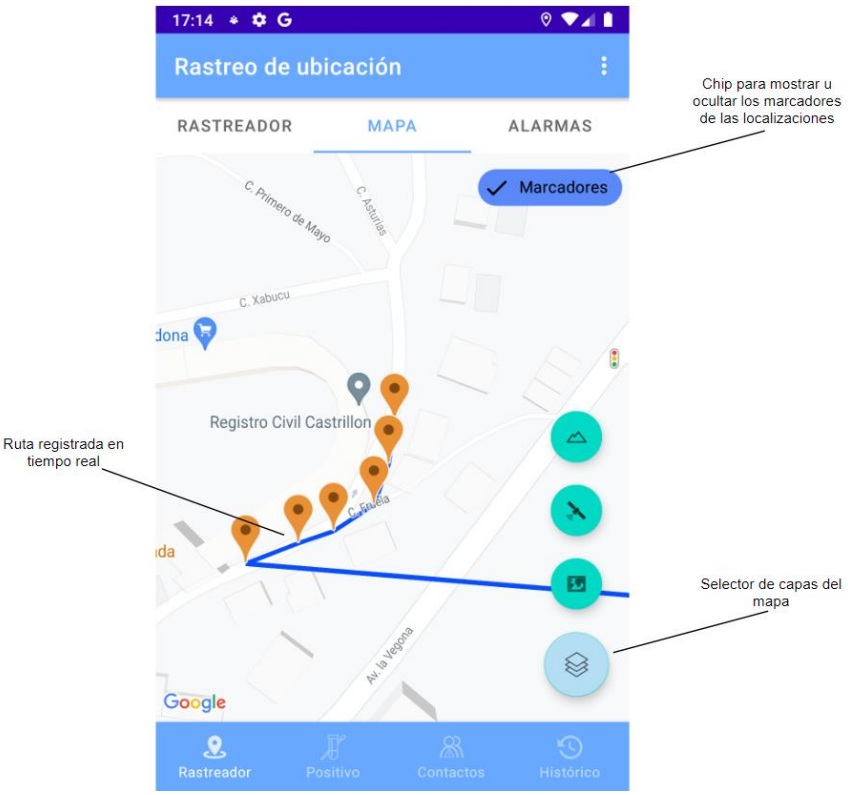


Ilustración 11.16. Pestaña del Mapa en tiempo real dentro del Rastreo de ubicación.

Como se puede ver, se muestran en el mapa todas las localizaciones que se van registrando en tiempo real mediante un **marcador**, además de que la aplicación va **trazando** una línea que representa la ruta seguida. El botón de **Marcadores** permite ocultar o mostrar los marcadores, y el botón situado en la esquina inferior derecha permite elegir entre las capas del mapa (**Normal, Satélite** y **Terreno** de abajo a arriba en el panel de botones).

11.2.1.2 *Alarmas de localización*

La última pestaña de **Alarmas** del rastreo de ubicación permite definir alarmas automáticas con una **hora de inicio** y una **hora de fin** para ejecutar el rastreo de ubicación. Así, a la hora de inicio, se ejecutará el servicio de rastreo de ubicación automáticamente sin necesidad de activarlo a mano a través de la pestaña de Rastreador, como se describió anteriormente, mientras que, a la hora de fin, se detendrá el servicio y la aplicación dejará de registrar las coordenadas del usuario. Esto es muy útil para programar diferentes alarmas al día para que se registren las rutas del usuario automáticamente. A continuación, se muestra el aspecto de esta pestaña de Alarmas.

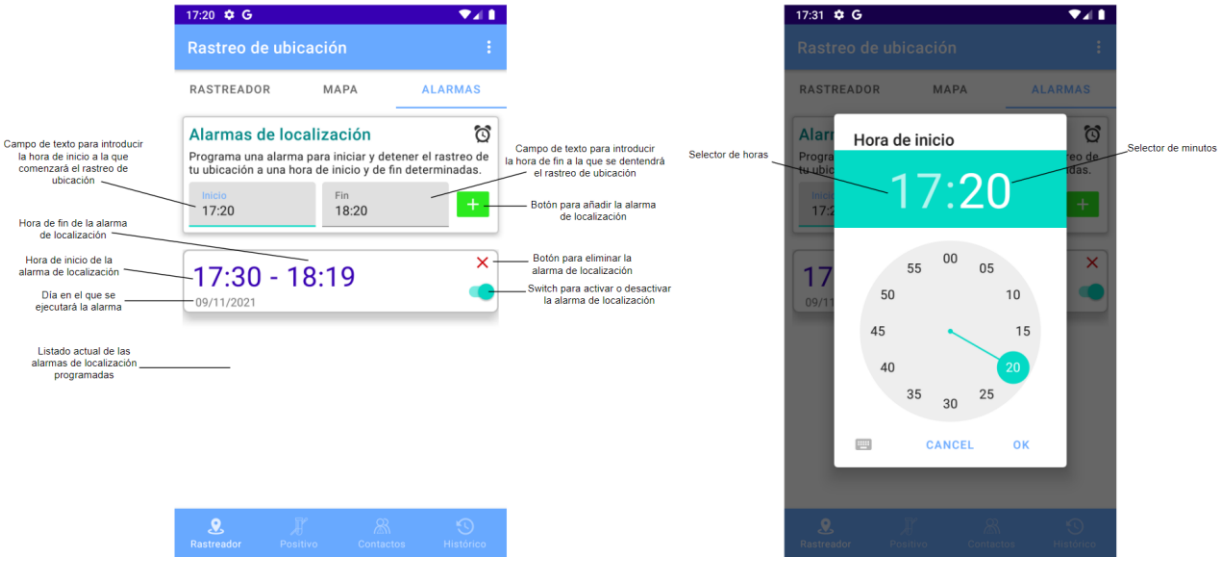


Ilustración 11.17. Pestaña de Alarmas dentro del rastreo de ubicación.

Como se puede ver, los campos de **inicio** y **fin** permiten seleccionar una hora del día para iniciar y detener el servicio de rastreo respectivamente. Al pulsar sobre uno de esos campos se abrirá el selector de horas y minutos mostrado en la pantalla derecha de la ilustración superior. Al pulsar sobre el botón de añadir, se insertará una nueva alarma en el sistema que se mostrará automáticamente en el listado inferior de alarmas programadas. Para cada alarma mostrada en este listado se indica su hora de **inicio** y de **fin** para la que fue programada, el **día** en el que se ejecutará, una **cruz** para eliminar por completo la alarma y un ***switch*** para activar o desactivar la alarma. Si una alarma está desactivada no se disparará a la hora indicada.

11.2.1.3 *Notificar un positivo*

Cuando un usuario da positivo tiene la oportunidad de notificarlo a través de la aplicación móvil, accediendo a la opción de **Positivo** en el menú inferior. La pantalla de notificación de positivos tiene el siguiente aspecto.

Ilustración 11.18. Pantalla de notificación de positivos.

El ***checkbox*** de la ilustración superior, permite al usuario decidir si **adjuntar** o no sus **datos personales** para facilitar las tareas de rastreo al personal sanitario. Si está marcado, al pulsar sobre el botón de ***He dado positivo***, se mostrará un formulario para insertar sus datos personales seguido de una **cláusula de consentimiento de tratamiento de datos**. Por otro lado, el botón de Política de Privacidad permite consultar la política de privacidad de la aplicación, también accesible desde el menú superior de la barra de herramientas. Encima del botón para notificar positivos se muestra el **Periodo de Infectividad**, el cual es configurado por las autoridades sanitarias y define el rango de días desde hoy hasta el n.º de días atrás especificados para recuperar las localizaciones registradas mediante el servicio de rastreo y subirlas a la nube. Al pulsar sobre el botón de ***He dado positivo*** comienza el proceso de notificación de positivos mostrado a continuación.

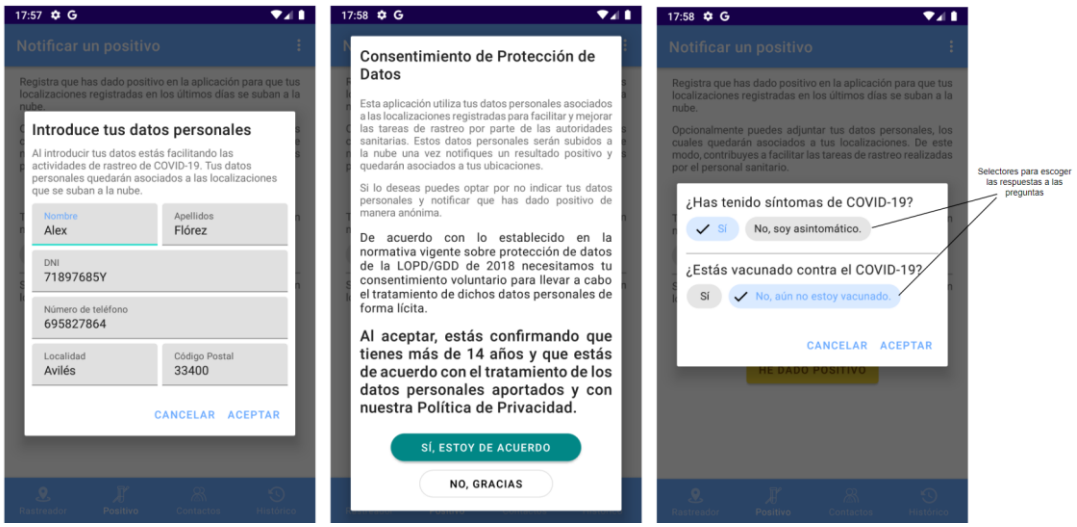


Ilustración 11.19. Proceso de notificación de un positivo incluyendo los datos personales.

Como se puede ver, en primer lugar, se muestra el **formulario** para insertar los datos personales. Al aceptar, se procede con la cláusula de **consentimiento** para solicitar al usuario su consentimiento sobre el tratamiento de sus datos personales. Si el usuario está de acuerdo, se muestra un último diálogo con una serie de **preguntas**, la primera sobre si el usuario ha manifestado algún síntoma y la segunda acerca de si está vacunado. Al aceptar este último diálogo, se mostrará una barra de progreso y si no surge ningún error, se mostrará un mensaje por pantalla indicando cuántas localizaciones fueron subidas a la nube.

Nota importante: el sistema no permitirá notificar un positivo si no existen localizaciones que subir a la nube registradas en los últimos días indicados por el periodo de infectividad o si ya se ha notificado un positivo y aún no han transcurrido los suficientes días para notificar otro.

11.2.1.4 *Comprobación de contactos*

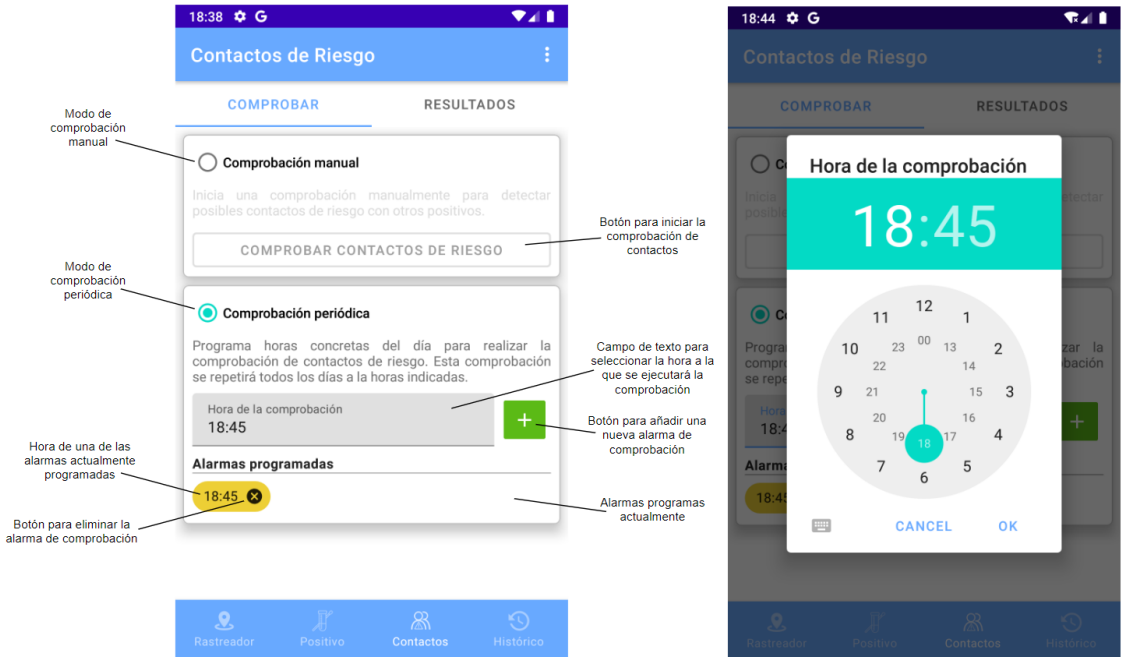


Ilustración 11.20. Pestaña de Comprobar dentro de la opción de comprobación de contactos.

La opción más interesante de la aplicación es la **Comprobación de Contactos**, accesible desde la opción Contactos en el menú inferior. Desde esta pantalla se pueden realizar comprobaciones para detectar si se ha estado cerca de algún positivo y de ser así **cuándo** se ha producido ese contacto y cuál es el **riesgo** de haber sido contagiado. La ILUSTRACIÓN 11.20 muestra el aspecto de esta pantalla. Como se puede ver hay **dos pestañas**:

1. **Comprobar**. Permite realizar una comprobación de contactos, ya sea manual o automáticamente a través de alarmas programadas a una hora.
2. **Resultados.** Muestra los resultados obtenidos de las comprobaciones realizadas por el usuario.

Desde la pestaña de Comprobar, se puede elegir entre la **comprobación manual** o la **comprobación periódica**, solo puede estar activado uno de los dos modos al mismo tiempo.

- Comprobación manual: al pulsar sobre el botón de Comprobar Contactos de Riesgo, se inicia una nueva comprobación en 1er plano, que solo funcionará mientras la aplicación esté encendida y en 1er plano. Mientras se comparan las coordenadas del usuario con las de otros positivos, se mostrará un indicador de progreso indeterminado. Una vez finalice la comprobación, el resultado se almacenará en el dispositivo y podrá consultarse desde la pestaña de Resultados, además de mostrarse una **notificación** con un resumen de los **resultados** obtenidos.
- Comprobación periódica: permite programar una alarma a una hora determinada para que se inicie la comprobación de contactos de riesgo a esa hora. Se pueden programar hasta un **máximo** de **tres alarmas.** Cuando se llegue a dicha hora, se disparará la alarma de comprobación iniciándose un **servicio** de **comprobación** el cual mostrará una **notificación** indicando que se está llevando a cabo una comprobación. Esta notificación permanecerá visible mientras dure la comprobación. A diferencia de la comprobación manual, en este caso se puede tener la **aplicación cerrada** e incluso el móvil bloqueado mientras se ejecuta la comprobación de contactos. Cuando termine la comprobación, su resultado se almacenará en el dispositivo y se mostrará una notificación con un resumen de este al igual que en el modo manual.

En la ILUSTRACIÓN 11.20, para añadir una nueva alarma de comprobación, primero se selecciona la **hora** pulsando sobre el campo de texto, tras lo cual se abrirá el selector de hora mostrado en la imagen derecha de dicha ilustración. Una vez seleccionada la hora, se pulsa sobre el botón de añadir denotado por un **+** para añadir la nueva alarma de comprobación.

Se pueden ver las alarmas de comprobación programadas actualmente en el apartado inferior de **Alarmas programadas**. Se muestra un ***chip*** para cada alarma programada, con una **cruz** que se puede pulsar para eliminar la alarma por completo.

11.2.1.5 *Visualización de resultados*

Los resultados de las comprobaciones de contactos se pueden consultar desde la **pestaña de Resultados** en la opción *Contactos* del menú inferior. En esta pestaña, los resultados aparecen ordenados por la **fecha** en la que fueron ejecutados, de **más reciente** a **más antiguo**, además de agruparse por día, tal y como se muestra en la siguiente ilustración.

Ilustración 11.21. Pestaña de Resultados dentro de la opción de Comprobación de Contactos.

En esta ilustración, se puede apreciar en la parte superior la **notificación** indicando un resumen con el resultado de una comprobación que estaba en curso. Justo debajo, se encuentra el título con la **fecha** del 10 de noviembre bajo la cual se agrupan todas las comprobaciones realizadas en ese día. En este caso, se han realizado tres comprobaciones, para cada una de las cuales se muestra un panel con la siguiente información:

- **Hora de la comprobación**
- **Porcentaje de riesgo**. Es el porcentaje de riesgo **más alto** de todos los contactos detectados en esa comprobación.
- **Positivos.** N.º de positivos con los que se entró en contacto.
- **Contactos de Riesgo**. Representa cuántas **coincidencias** en el espacio y en el tiempo se detectaron con los positivos con los que se entró en contacto.

Para **ver los detalles** de un resultado, basta con **pulsar** sobre el resultado deseado tras lo cual se mostrará una nueva pantalla con sus detalles, tal y como se muestra en la siguiente ilustración.

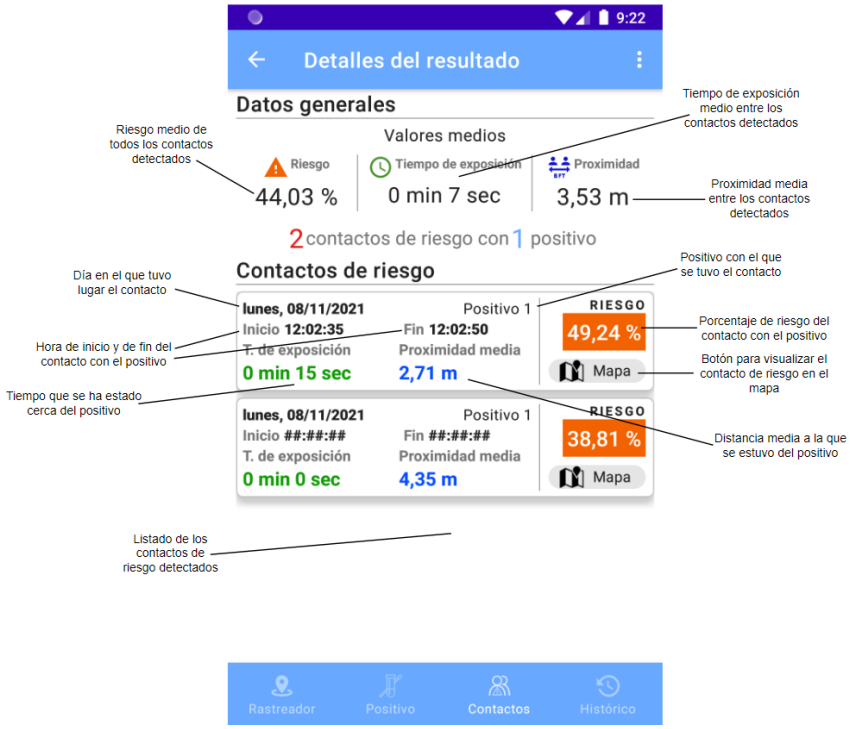


Ilustración 11.22. Pantalla con los detalles de un resultado de una comprobación.

En esta pantalla de detalles, se muestra en primer lugar una serie de **Datos generales**:

- **Valores medios.** Son los valores calculados como la media de todos los contactos de riesgo detectados, es decir, el porcentaje de riesgo medio, el tiempo de exposición medio y la proximidad media calculados entre todos los contactos detectados.
- **Número de contactos de riesgo** con el **número de positivos**. Indica cuántas veces se ha entrado en contacto con el número de positivos con los que se mantuvo contacto.

Justo debajo, se muestra un listado con todos los **contactos de riesgo** que fueron detectados, indicando para cada uno de ellos:

- **Día del contacto.** Día en el que tuvo lugar el contacto cercano con el positivo.
- **Etiqueta del positivo.** Indica el número de positivo con el que se tuvo este contacto.
- **Inicio / Fin.** Hora de inicio y de fin aproximadas durante las cuales se estuvo cerca del positivo en este tramo de contacto. En algunos casos, estas horas pueden ser mostradas como ***##:##:##***, en cuyo caso significa que solo se detectó un único punto de contacto con el positivo o bien que no se ha podido calcular una **intersección** en el tiempo entre ambos.
- **Tiempo de exposición.** Indica cuántos minutos y segundos se estuvo en contacto cercano con el positivo.
- **Proximidad media.** Indica los metros de distancia media a los que se estuvo del positivo.
- **Riesgo.** Indica la gravedad del contacto con el positivo, en base al tiempo de exposición, la proximidad y otros valores internos que no se muestran al usuario.
- **Botón de Mapa**. Permite visualizar el contacto de riesgo en un mapa.

Para visualizar un contacto de riesgo en un mapa se debe pulsar sobre el botón de *Mapa* del contacto deseado, tras lo cual este se mostrará en un mapa tal y como se indica en la siguiente ilustración.

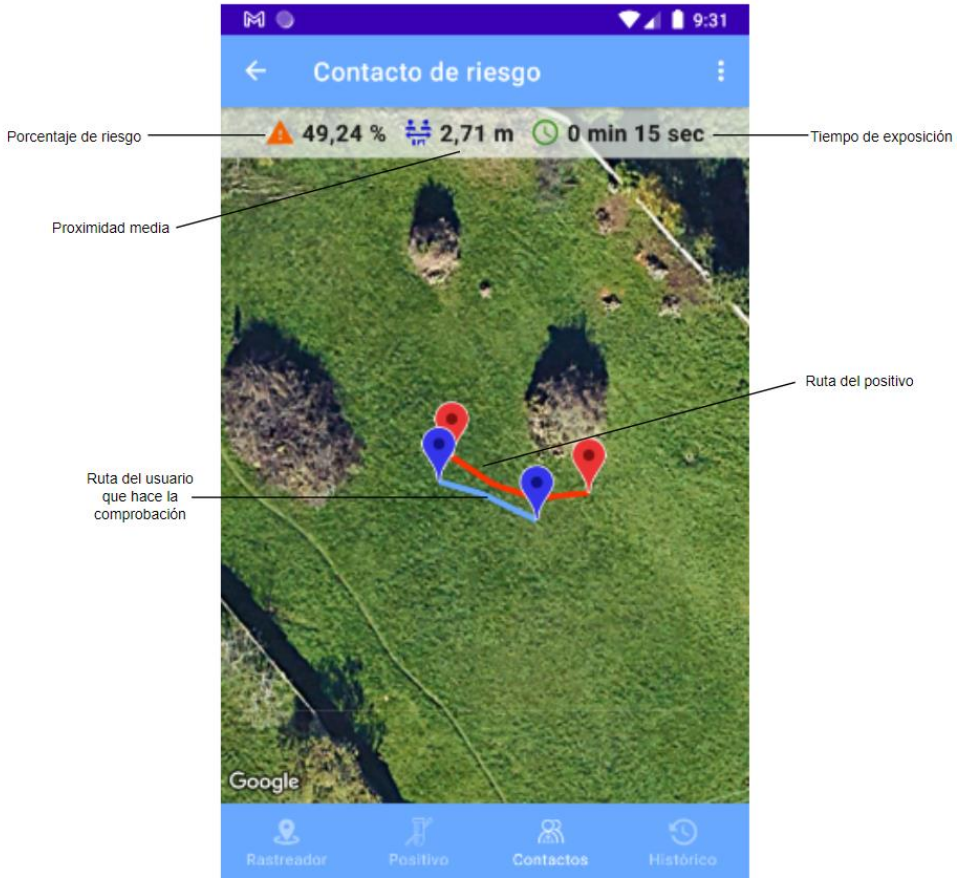


Ilustración 11.23 Visualización de un contacto de riesgo en un mapa.

Como se puede ver, en el mapa se dibujan las **rutas** del propio usuario en color azul y del **positivo** con el que se tuvo contacto en color rojo. Los puntos de inicio y de fin de ambos tramos está delimitado por marcadores. Además, en la parte superior también se indican los **valores** del contacto de riesgo, es decir, el **porcentaje de riesgo**, la **proximidad media** y el **tiempo de exposición** de izquierda a derecha en la imagen.

11.2.1.6 *Histórico de localizaciones*

La última opción del menú inferior es la del **Histórico**, desde el cual se puede consultar un historial de las localizaciones registradas por la aplicación agrupadas por días. Esta pantalla tiene el siguiente aspecto.

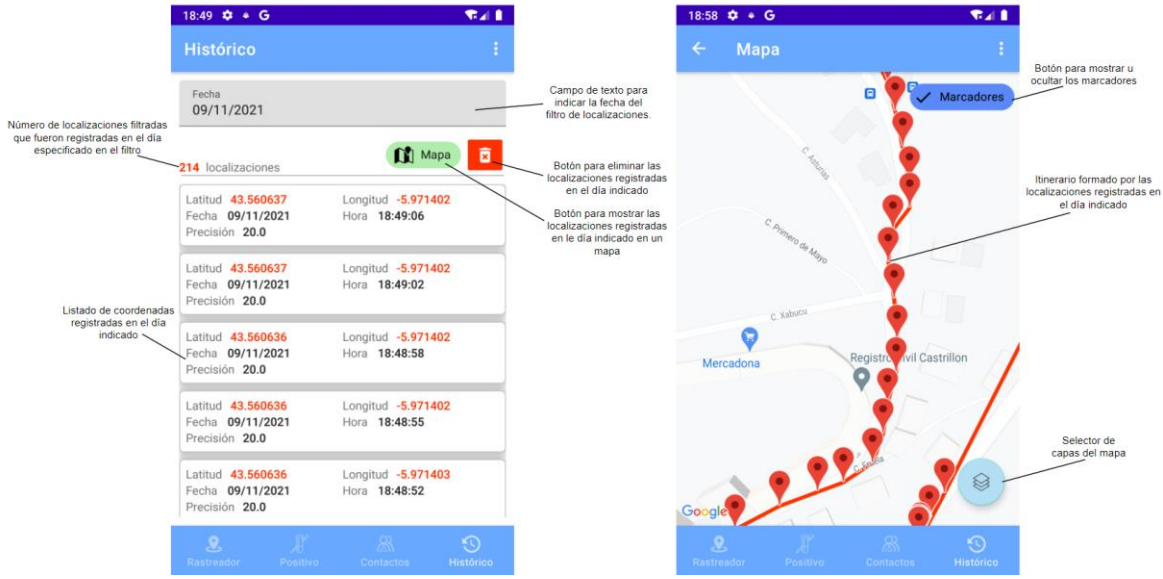


Ilustración 11.24. Pantalla del Histórico de localizaciones registradas.

En la imagen izquierda de la ilustración, el campo de texto **Fecha** permite seleccionar una fecha a través de un calendario que tiene el siguiente aspecto.

Ilustración 11.25. Calendario para seleccionar una fecha de filtro en el Histórico.

Al seleccionar una fecha concreta, en el listado de la imagen izquierda se mostrarán todas aquellas localizaciones que fueron registradas en ese día específico. Encima del listado se muestra el **número de localizaciones** registradas en ese día junto con un botón para **mostrarlas en un mapa** y otro botón para **eliminarlas**. Estos componentes solo son **visibles** si existe alguna localización registrada en el día especificado.

Si se pulsa sobre el botón de Mapa, se mostrará la pantalla derecha de la ILUSTRACIÓN 11.24, donde se puede ver el itinerario representado por estas coordenadas filtradas del día especificado. De manera similar al mapa en tiempo real del rastreo, también se pueden ocultar los marcadores y seleccionar la capa concreta del mapa.

Al pulsar sobre el botón de la **papelera**, se eliminarán todas las localizaciones registradas en el día especificado.

11.2.1.7 *Ajustes generales*

Para terminar con el manual de la aplicación móvil, se encuentran los **ajustes generales** de la aplicación, accesibles desde el **menú superior** situado en la barra superior de herramientas, representado con **"tres puntos alineados verticalmente"**, tal y como muestra la siguiente ilustración.

Ilustración 11.26. Menú superior de ajustes generales de la aplicación móvil.

Para consultar la **política de privacidad**, se debe pulsar sobre la opción Política de Privacidad, tras lo cual se mostrará la siguiente pantalla.

Ilustración 11.27. Política de Privacidad de la aplicación móvil.

Para ir a los **Ajustes** del sistema, basta con pulsar sobre la opción de Ajustes en el menú superior. La siguiente ilustración muestra el aspecto que tiene esta pantalla de ajustes.

Ilustración 11.28. Pantalla de ajustes de la aplicación móvil.

En cuanto al **rastreo**, se pueden configurar los siguientes parámetros:

- **Intervalo de Tiempo**. Indica el n.º de minutos y segundos que transcurren entre el registro de una coordenada y la siguiente. Cuanto menor sea este intervalo, más **fiable** será el rastreo y por tanto más precisas serán las comprobaciones, a costa de un mayor **consumo de batería**.
- **Desplazamiento mínimo**. Indica cuántos metros de diferencia debe haber entre una coordenada y otra para que se registren las coordenadas. Cuando es mayor que 0 m evita que se registren múltiples coordenadas de la misma posición, por ejemplo, cuando el dispositivo está encima de una mesa quieto.

En cuanto a la **comprobación de contactos**, se pueden configurar los siguientes parámetros:

- **Alcance de la comprobación**. Define cuántos días atrás, desde que se inicia una comprobación, se tienen en cuenta para comprobar la cercanía con otros positivos. Por ejemplo, con un alcance de 3 días, se comprueba si se ha estado en contacto con algún positivo en los últimos 3 días.

En cuanto a las **notificaciones:**

- **Notificaciones de positivos.** Permite habilitar o deshabilitar el envío de notificaciones diarias relativas al número de positivos que se notificaron en el sistema en el día de hoy.

11.2.2 Usuario administrador del panel de control web

Al entrar por primera vez en el panel de control *web* de *Contact Tracker* se muestra la siguiente **pantalla de bienvenida**.

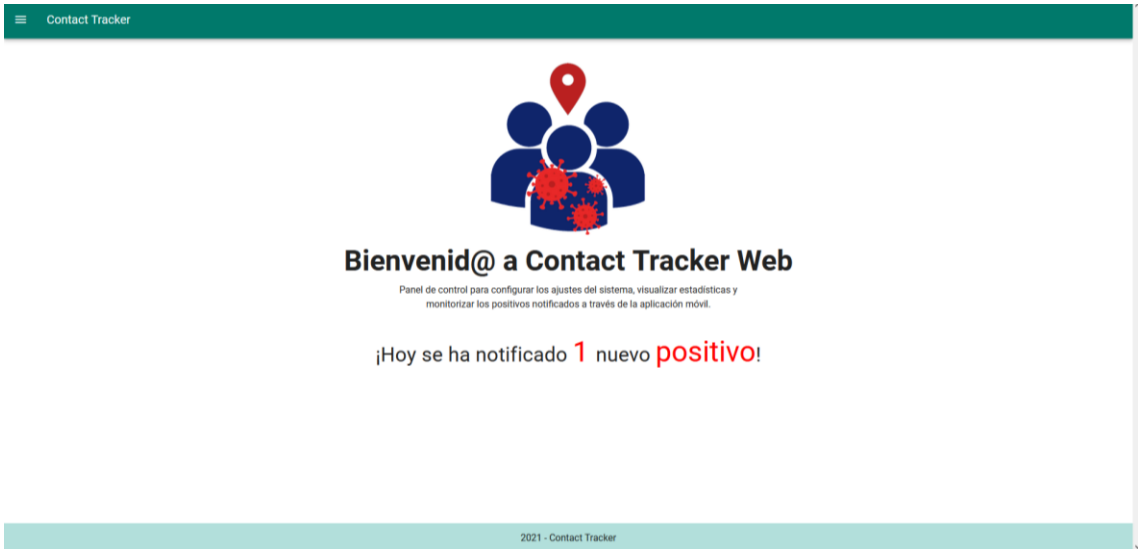


Ilustración 11.29. Pantalla de bienvenida del panel de control web.

En esta pantalla, a modo de información complementaria, se indica el **número de positivos** que fueron notificados en el sistema en el día de hoy.

El **menú desplegable** de la aplicación *web* está disponible en todo momento en la esquina superior izquierda, pulsando sobre el icono de las **"tres barras horizontales"**.

11.2.2.1 *Configuración del sistema*

Desde la opción de menú de **Configuración**, se pueden modificar los parámetros de configuración del sistema. Para ello, se debe desplegar el menú lateral y seleccionar la opción de Configuración. Dentro de esta pantalla, existen dos **pestañas**:

- **Notificación de positivos**. Contiene los parámetros de configuración de la notificación de positivos a través de la aplicación móvil.
- **Comprobación de contactos**. Contiene los parámetros de configuración relativos al algoritmo de comprobación de contactos de riesgo.

La siguiente ilustración muestra el panel de configuración de la **notificación de positivos**.

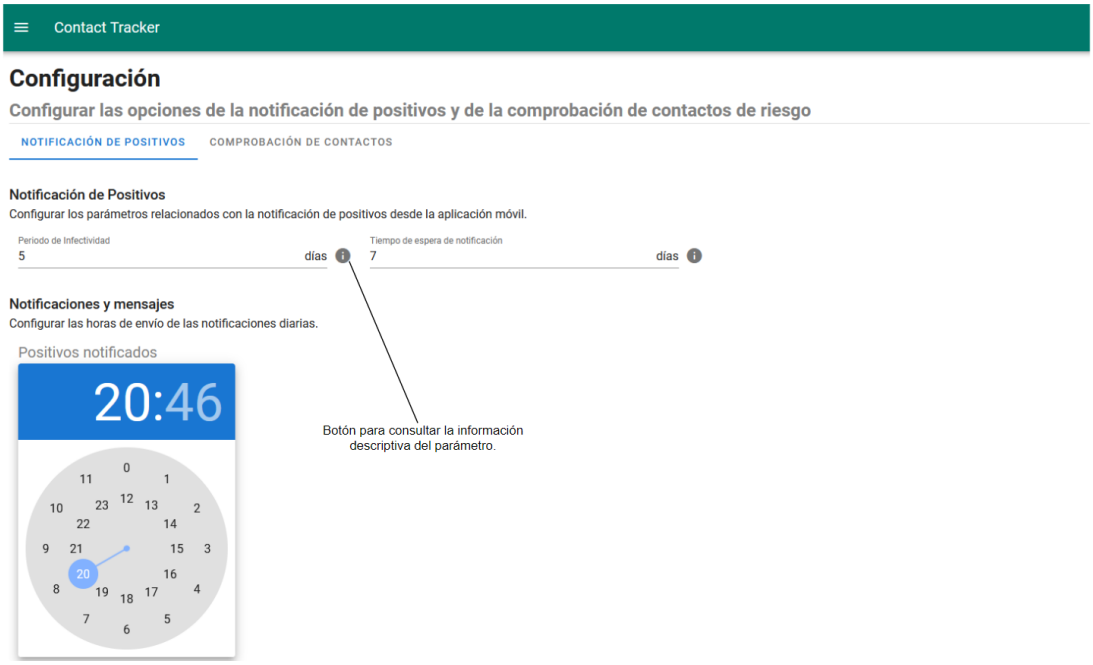


Ilustración 11.30. Panel de configuración de la notificación de positivos.

Desde este panel, se pueden configurar los siguientes parámetros:

- **Periodo de infectividad.** Representa el número de días atrás desde los primeros síntomas manifestados por un positivo durante los cuales este es contagioso. Este valor se utiliza en la aplicación móvil para subir las coordenadas registradas en los últimos días especificados por este periodo.
- **Tiempo de espera de notificación**. Indica el n.º de días que es necesario que transcurran después de haber notificado un positivo para poder volver a notificar otro desde la aplicación móvil.
- **Notificaciones sobre los positivos notificados.** Permite configurar la hora a la que se envían las notificaciones relativas al n.º de positivos notificados hoy. Estas notificaciones se envían a los dispositivos *Android* todos los días a la hora especificada.

Con los **botones de información (i)** se puede consultar una explicación sobre en qué consiste un parámetro de configuración determinado.

La siguiente ilustración muestra el panel de configuración de la **comprobación de contactos**.

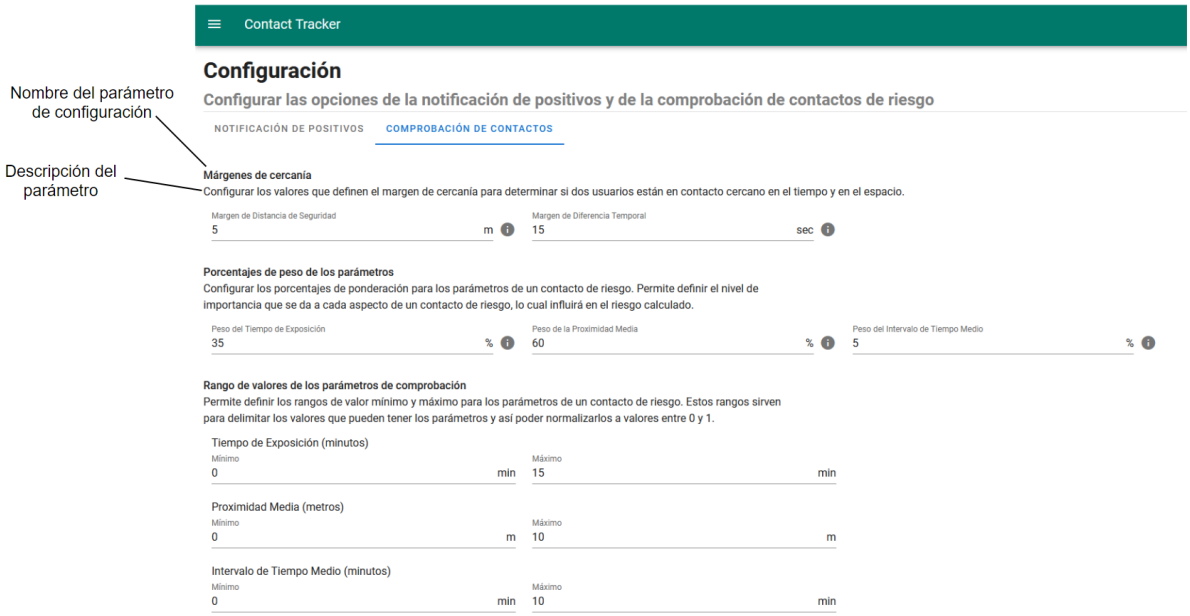


Ilustración 11.31. Panel de configuración de la comprobación de contactos de riesgo.

Desde este panel de configuración, se pueden configurar los siguientes parámetros que influyen en el comportamiento del algoritmo de comprobación ejecutado en los dispositivos móviles:

- **Margen de distancia de seguridad.** Permite indicar el número de metros de distancia a partir de los cuales se considera que dos usuarios están suficientemente próximos en el espacio como para considerar que están en contacto cercano.
- **Margen de diferencia temporal.** Permite indicar el número de segundos de diferencia entre una posición de un usuario y la de otro usuario para considerar que ambos han coincidido en el tiempo con una diferencia temporal lo suficientemente pequeña como para considerar que están cercanos en el tiempo.
- **Pesos de los parámetros descriptivos de un contacto de riesgo.** Permite variar la ponderación de pesos de los valores calculados para un contacto de riesgo, los cuales influyen en el cálculo del porcentaje de **riesgo** que se asigna a cada contacto de riesgo detectado.
    - **Peso del tiempo de exposición.** Porcentaje de peso que se le da al tiempo de exposición con un positivo.
    - **Peso de la proximidad media.** Porcentaje de peso que se le da a la proximidad en metros con el positivo.
    - **Peso del intervalo de tiempo medio.** Porcentaje de peso que se le da al valor del intervalo de tiempo medio entre las fechas de registro de las coordenadas. El intervalo de tiempo medio representa la fiabilidad del rastreo de ubicación, pues cuanto menor sea este intervalo más fiables y exhaustivos serán los itinerarios registrados.
- **Rangos de valores de los parámetros de comprobación.** Permite definir los valores mínimos y máximos de cada parámetro descriptivo de un contacto de riesgo para que el algoritmo de comprobación pueda normalizarlos (convertirlos a un valor entre 0 y 1).

- **Rango del tiempo de exposición.** Define el valor mínimo y el valor máximo de exposición en minutos. A partir de este valor máximo, se considera que el peso que puede aportar el tiempo de exposición es el máximo, es decir, se normaliza a 1.
- **Rango de la proximidad media.** Define el valor mínimo y máximo de la proximidad en metros. A partir del valor máximo de metros, se considera que el peso que aporta la proximidad es nulo, es decir, se normaliza a 0 porque se considera que se está muy lejos del positivo.
- **Rango del intervalo de tiempo medio.** Define el valor mínimo y máximo del intervalo de tiempo de registro de coordenadas en minutos. A partir del valor máximo de intervalo de tiempo medio, se considera que este parámetro es nulo, es decir, se normaliza a 0 porque se considera que el registro de ubicación deja de ser fiable a partir de ese valor.

En ambos paneles de configuración, para **guardar los cambios realizados** basta con editar los parámetros deseados y pulsar sobre el botón de **GUARDAR CAMBIOS**, o bien, sobre el botón de **CANCELAR** si se desea revertir los cambios. Esta pareja de botones se muestra cada vez que se modifica alguno de los parámetros de configuración.

11.2.2.2 *Visualización de estadísticas*

Para visualizar las **estadísticas** de **uso** de la aplicación de *Contact Tracker*, se debe desplegar el menú lateral y pulsar sobre la opción de **Estadísticas**. La siguiente ilustración muestra el aspecto que tiene este panel de estadísticas.

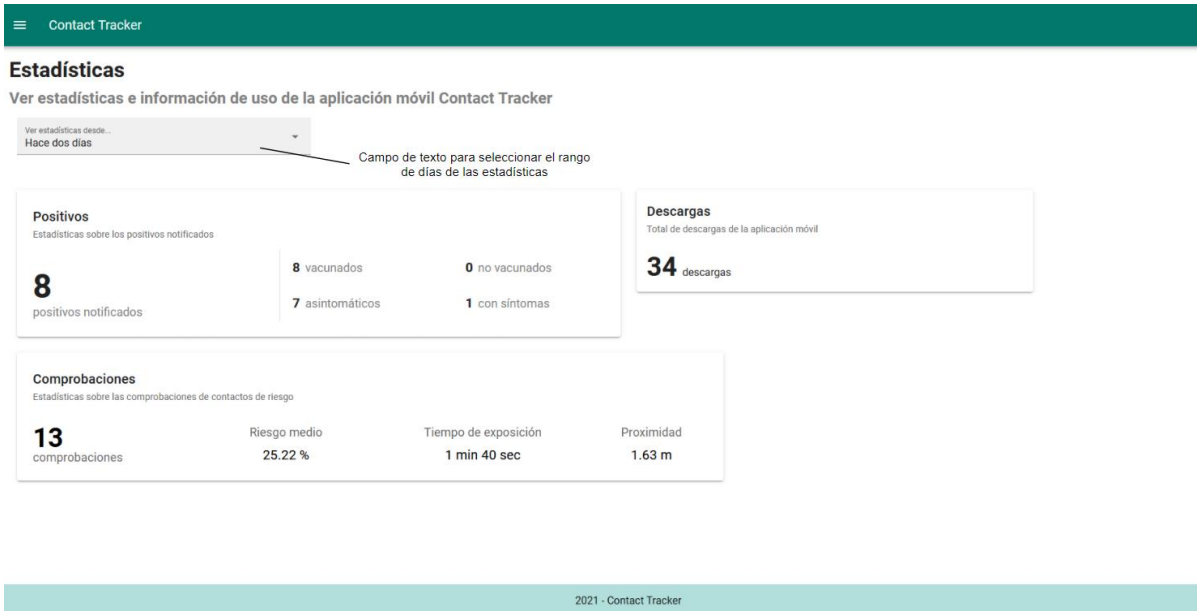


Ilustración 11.32. Panel de estadísticas de uso del sistema de Contact Tracker.

En la ilustración superior, se puede ver en primer lugar el **campo de texto** de **selección** del rango de días que se tienen en cuenta para recopilar las estadísticas mostradas, seguido de **tres paneles** de estadísticas.

El primer panel, muestra las estadísticas relativas a los **positivos** que se describen a continuación:

- **Número de positivos.** Es el n.º de positivos que fueron notificados en el sistema por los usuarios desde sus dispositivos móviles.
- **Asintomáticos / No Asintomáticos.** Recuento del n.º de positivos notificados que son asintomáticos y de los que sí presentan síntomas.
- **Vacunados / No Vacunados.** Recuento del n.º de positivos notificados que están vacunados frente a los que no lo están.

El segundo panel, muestra las estadísticas relativas a las **descargas** de la aplicación móvil, es decir, cuántos usuarios se han instalado la aplicación móvil en su dispositivo.

El tercer y último panel, muestra las estadísticas relativas a las **comprobaciones** de **contactos** realizadas por los usuarios:

- **N.º de comprobaciones.** Representa cuántas comprobaciones se han realizado con las aplicaciones móviles en los últimos días.
- **Riesgo medio.** Porcentaje de riesgo medio calculado entre todos los resultados de las comprobaciones realizadas.
- **Tiempo de exposición medio**. N.º medio de minutos y segundos calculado como la media de los tiempos de exposición de los resultados de las comprobaciones realizadas.
- **Proximidad media**. N.º medio de metros de distancia calculado como la media de las proximidades de los resultados de las comprobaciones realizadas.

11.2.2.3 *Ver positivos notificados*

Para ver los positivos notificados en el sistema, se debe desplegar el menú lateral y pulsar sobre la opción de **Positivos**. Se mostrará entonces la siguiente pantalla con el listado de positivos notificados.

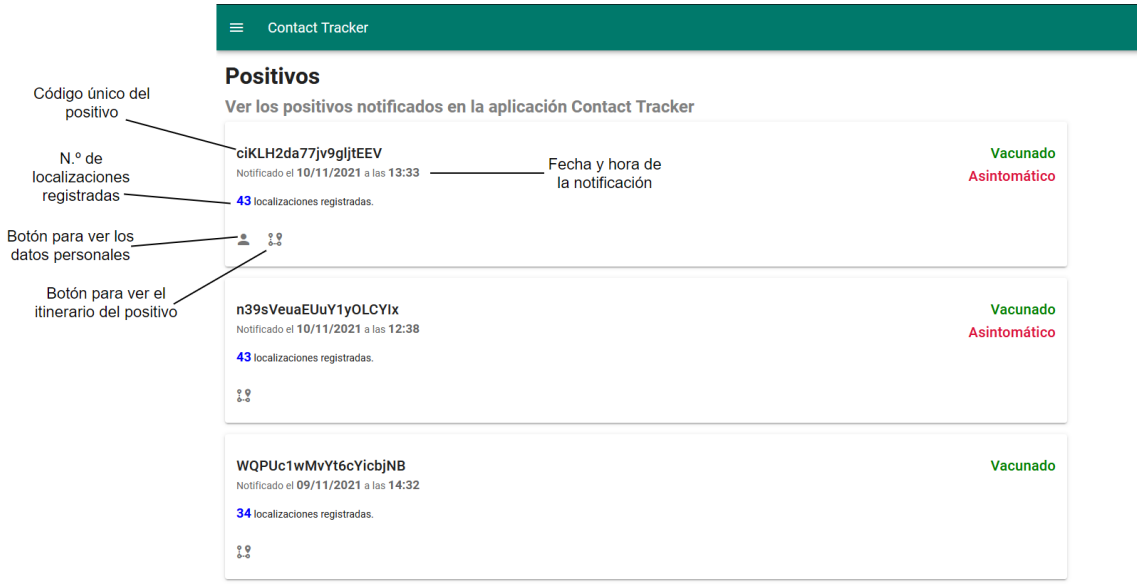


Ilustración 11.33. Pantalla de visualización de los positivos notificados.

Para cada positivo notificado en el sistema se muestran los siguientes datos:

- **Código del positivo**. Es un código alfanumérico que identifica unívocamente a cada positivo registrado en el sistema.
- **Fecha y hora de notificación**. Indica la fecha y la hora a la que fue notificado el positivo.
- **N.º de localizaciones**. Indica el número de localizaciones que fueron subidas a la nube y que conforman el itinerario del positivo.
- **Etiquetas de Vacunado / Asintomático**. Etiquetas que indican si el positivo está vacunado o si es asintomático.
- **Botón de datos personales**. Muestra un diálogo con los datos personales del positivo. Este botón solo aparecerá en aquellos positivos que tengan asociados datos personales.
- **Botón de itinerario**. Muestra en un diálogo el itinerario del positivo en un mapa dibujándolo con una línea definida por las localizaciones del positivo.

Para ver los datos personales de un positivo, basta con pulsar sobre el botón con el icono de un usuario, tras lo cual se mostrará el siguiente diálogo.

Ilustración 11.34. Diálogo con los datos personales de un positivo.

Para visualizar el **itinerario** de un positivo, basta con pulsar sobre el botón con el icono de un itinerario, tras lo cual se abrirá un diálogo que muestra las localizaciones del positivo en un mapa, tal y como se muestra en la siguiente ilustración.

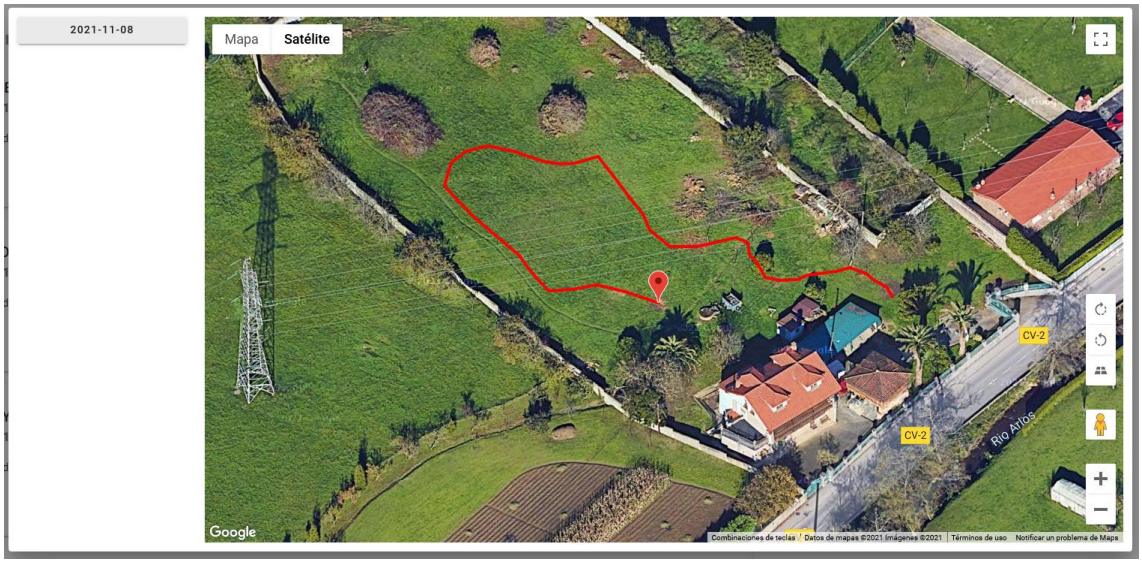


Ilustración 11.35. Diálogo con el mapa que muestra el itinerario de un positivo.

En el panel izquierdo de este diálogo, se muestran los días que componen el itinerario y en los cuales se agrupan las localizaciones del positivo. Al pulsar sobre uno de los días, el **foco del mapa** se traslada hacia el tramo registrado con las localizaciones de ese día.

Capítulo 12. Conclusiones y Ampliaciones

A lo largo de este documento, se ha visto en profundidad el proceso de planificación, definición, diseño y construcción del sistema, además de otros aspectos transversales del proyecto, tras lo cual se han obtenido una serie de conclusiones que se discuten en este capítulo. Además, existen algunas funcionalidades que no han sido implementadas, ya sea por falta de tiempo o porque quedan fuera del alcance del proyecto, las cuales se pueden posponer de cara al futuro para otro sistema similar basado en este.

12.1 Conclusiones

Una vez completadas las cuatro iteraciones planificadas junto con las tres iteraciones extra, necesarias debido a los contratiempos, se ha conseguido implementar y desplegar con éxito los cuatro módulos especificados en el apartado 7.3.1 DESCRIPCIÓN DE LOS SUBSISTEMAS, de tal forma que se han obtenido como productos la **aplicación móvil** de **rastreo**, la **aplicación *web*** del **panel de control**, el **servidor *web*** con la ***API REST*** y la base de datos ***Firestore*** hospedada en la nube. La aplicación *web* y el servidor están desplegados en los servicios de *Azure*, de manera que se puede acceder a ellos desde una máquina remota sin necesidad de hacerlo en local. Por otro lado, se han alcanzado prácticamente la mayoría de los objetivos establecidos en la sección 2.2 relativa a los OBJETIVOS DEL PROYECTO, a excepción de los objetivos 9 y 13 dado que se trata de un prototipo que aún no ha sido implantando en un entorno real y aún no se ha dado el tiempo suficiente para determinar si el uso extendido de la aplicación disminuye la transmisión del virus y si puede ser utilizado con otras enfermedades contagiosas.

Sin embargo, bajo un entorno ficticio donde se simulan los contagios, el sistema parece ser **efectivo** en casi el 100% de las casuísticas. Solamente en entornos cerrados se disminuye la precisión de la comprobación de contactos debido al aumento del error en las coordenadas proporcionadas por *GPS* o *WiFi*, lo cual hace que se registren ubicaciones que no se corresponden con la posición real del usuario. Esto permite sacar la conclusión de que el correcto funcionamiento del sistema depende en gran parte de la **calidad** de las coordenadas registradas, es decir, en la **precisión** del *GPS*/*WiFi*, por lo que en aquellas situaciones, ya sea un entorno cerrado o al aire libre, en las que el registro de coordenadas no sea lo suficientemente preciso debido a la imprecisión del *GPS* o de la geolocalización por *internet*, la comprobación de contactos apenas será efectiva, existiendo la posibilidad de que se detecten **falsos positivos** (el sistema detecta erróneamente que se ha estado en contacto con un positivo) o **falsos negativos** (el sistema detecta erróneamente que no se ha entrado en contacto con ningún positivo). Esta imprecisión y error en el registro de coordenadas, está **fuera del alcance** del proyecto, al tratarse de algo intrínseco a la naturaleza de la geolocalización y del *GPS*, pues la aplicación móvil poco puede hacer para mejorar la calidad de las localizaciones. Tal vez sería posible utilizar algoritmos inteligentes mediante **heurísticos** para detectar posibles errores en las coordenadas y tratar de corregirlos alineando aquellas ubicaciones fuera de lugar respecto

de la tendencia común del itinerario que siguen el resto de las coordenadas, pero esto también queda fuera del alcance del proyecto, como una posible ampliación.

Todos estos logros han sido completados a costa de incurrir en un **retraso** de unos **tres meses** respecto a la fecha de entrega planificada inicialmente, tal y como se comentó en el capítulo de Control y seguimiento, lo cual influye en el presupuesto final estimado para el proyecto como se explica en la sección 13.5 sobre el seguimiento del proyecto. Las pérdidas aproximadas resultantes de estos contratiempos ascienden a los **9.518,37 €,** considerando que el coste estimado de una iteración es de unos 3.172,79 €.

Como conclusión, se ha experimentado con la geolocalización en *Android* para rastrear contactos de riesgo obteniéndose unos buenos resultados, más allá de los errores de precisión de la instrumentación *GPS*. El sistema funciona como se esperaba detectando los contactos de riesgo cercanos a los itinerarios de los usuarios, calculando además diversos parámetros de interés como el tiempo de exposición, proximidad…*etc*, cuyos valores son usados para calcular el **nivel de riesgo** del contacto y así dar una idea aproximada al usuario sobre la gravedad de su encuentro con un positivo. El usuario tiene la capacidad de configurar algunos aspectos del rastreo y de la comprobación de contactos. Además, el sistema es configurable por los administradores desde el panel de control *web*, pudiendo gestionar la notificación de positivos y el comportamiento del algoritmo de comprobación según las necesidades sanitarias. También se permite visualizar los itinerarios de los positivos notificados en un mapa, además de una serie de estadísticas de uso de la aplicación, como descargas, positivos notificados, comprobaciones realizadas o incluso valores medios de los parámetros de los contactos, es decir, tiempo de exposición…*etc.* Así, el sistema funciona a la perfección siempre y cuando el registro de coordenadas sea lo suficientemente preciso, lo cual es difícil de conseguir en un entorno real donde los *chips GPS* y las condiciones de rastreo son muy variadas. Esto hace que la **alternativa** del **Bluetooth** sea más llamativa e interesante que utilizar la geolocalización para rastrear a los usuarios, pues el proceso comprobación de contactos es mucho más rápido y sencillo, además de obviar los errores de localización al tratarse simplemente de ***tokens*** únicos que se intercambian con los dispositivos cercanos, tal y como se explicó en el apartado 2.4.1.1 dedicado a RADAR COVID (ESPAÑA). Además, en la mayoría de los casos, el consumo de **batería** será mayor al utilizar la geolocalización pues es necesario registrar periódicamente las coordenadas, a diferencia del escaneo de dispositivos por *bluetooth*, más ligero para la vida de la batería.

Con todo ello, la geolocalización como herramienta para el rastreo de contactos de riesgo supone una buena alternativa que cumple con su cometido cuando la calidad del rastreo de ubicación es aceptable, aunque, sin duda alguna, el *bluetooth* es una alternativa preferible a la hora de detectar individuos cercanos, dada su simpleza, además de que no involucra aspectos de posición y por tanto errores de precisión.

12.2 Ampliaciones

En esta sección se describen las distintas ampliaciones propuestas para el sistema, las cuales no se han podido llevar a cabo para el prototipo. Otras son meras ampliaciones que, aunque no fueron planificadas desde un principio, se incluyen en este apartado como posibles mejoras de cara a implementar en un futuro sistema similar.

12.2.1 Autenticación como administrador

Esta ampliación consiste en añadir un simple sistema de autenticación para los usuarios de tipo administrador, de forma que así se controle el acceso al panel de control *web*. Al intentar acceder a dicho panel, se mostraría un formulario para indicar un nombre de usuario y una contraseña, ambos proporcionados por un administrador del sistema certificado, y una vez autenticado, el administrador tendría acceso a todas las características del panel de control.

La autenticación podría implementarse mediante un ***token JWT*** al cual se le asigna una validez temporal, de modo que al cabo de cierto tiempo caducase su validez y el administrador tendría que repetir la autenticación. Con cada petición se envía en *token* el cual se comprueba en el servidor antes de acceder al recurso solicitado.

Este sistema aporta gran seguridad para limitar el acceso a las características de administrador solamente a aquellas personas que dispongan de unas credenciales válidas. La autenticación estaba planificada desde un principio para ser implementada, sin embargo, debido a los diversos contratiempos no llegó a llevarse a cabo para no extender el desarrollo más de la cuenta.

12.2.2 JWT y certificados para proteger los *endpoints* de la *API REST*

Está ampliación está relacionada con la anterior pues consiste en proteger las rutas expuestas por el servidor a través de *tokens* privados generados por el sistema. Concretamente, se trata de generar un ***token*** válido ***JWT*** que se envía al cliente recién autenticado mediante sus credenciales. El cliente guarda este *token* para utilizarlo en las futuras peticiones, por ejemplo, a través de una *cookie*, y lo envía con cada petición al servidor, el cual comprueba su validez antes de conceder acceso al recurso solicitado. La validez del *token* depende del *secreto* mediante el cual fue generado y la fecha de caducidad del mismo.

Por otro lado, en la aplicación móvil, no existe autenticación para los usuarios, sino que estos acceden libremente a las funciones de la aplicación. Para proteger los *endpoints* de posibles atacantes y de peticiones que no provengan de dispositivos móviles con una versión oficial de *Contact Tracker*, se propone el uso de **certificados** que se instalarían automáticamente al instalar la aplicación en el dispositivo. De este modo, con cada petición realizada por el cliente *Android* al servidor *web*, se enviaría este certificado cuya validez se comprobaría antes de acceder al recurso, de manera similar al *token JWT*.

Las ventajas que aportan estas ampliaciones son obvias, una mayor **seguridad** para el servidor *web* de forma que no se pueda consumir la *API REST* a menos que se esté certificado para ello, ya sea por medio de un *token JWT*, obtenido tras autenticarse en el sistema como administrador, o por medio de un certificado móvil, instalado automáticamente cuando se dispone de una versión oficial de la aplicación en el dispositivo.

Ambos aspectos estaban programados para ser implementados desde un principio, aunque al igual que con la autenticación, no se llevaron a cabo debido a los contratiempos y por no extender de más el desarrollo del sistema.

12.2.3 Acotar los positivos con los que hacer la comprobación

A la hora de realizar la comprobación de contactos de riesgo, la aplicación móvil recupera de la base de datos central todos los positivos existentes cuyos itinerarios tengan localizaciones que hayan sido registradas en alguno de los días en los que el usuario, que realiza la comprobación, haya registrado sus ubicaciones. Este filtro de días limita la comprobación de forma que solo se comparen las localizaciones de días iguales, ignorando el resto ya que carece de sentido. Así, el algoritmo de comprobación es más eficiente y requiere menor tiempo de procesamiento. Sin embargo, este filtro se podría mejorar aún más **clasificando** los **itinerarios** por zonas, utilizando, por ejemplo, la ***API Places*** de *Google*, de tal manera que se tenga en cuenta, no solo la temporalidad de las localizaciones, sino también la propia posición geográfica para descartar positivos que no tengan ubicaciones registradas en los mismos días ni en las mismas zonas que el usuario.

Esta ampliación no ha sido planificada inicialmente, sino que se propone como una mejora de cara a futuros sistemas similares.

12.2.4 Comprobación automática con cada positivo notificado

En la aplicación móvil resultante, existen dos modos de ejecutar la comprobación de contactos. Por un lado, está el modo manual, el cual implica ejecutar la comprobación manualmente cuando lo decide el usuario, y por otro, el modo periódico, donde se permite programar comprobaciones de contactos para que se ejecuten a ciertas horas. Una posible alternativa de cara a futuros sistemas basados en este sería la posibilidad de añadir un tercer modo, por ejemplo, **modo automático**, en el cual la comprobación de contactos se realizase automáticamente después de que se hubiera notificado un **nuevo positivo** en el sistema. Esto tendría como ventaja que la efectividad del sistema sería más elevada, ya que las comprobaciones de contactos se ejecutarían justo en el momento de conocer el itinerario de un nuevo positivo, más cerca de la hora de la notificación, en lugar de a una hora más alejada.

Como desventaja principal, el hecho de tener que ejecutar una comprobación con cada nuevo positivo notificado puede derivar en una **saturación** del sistema, si se considera que en un entorno real se notifican miles de positivos diarios. Una posible solución sería limitar el número de comprobaciones que se realizan por hora, por ejemplo, además de ayudarse del filtro de positivos para acotar la comprobación, lo cual se comentó en la anterior ampliación.

12.2.5 Inteligencia artificial para la detección de contactos y para reducir el error del rastreo de coordenadas

Una alternativa al algoritmo de comprobación clásico, implementado actualmente, sería utilizar alguna técnica de inteligencia artificial para detectar contactos de riesgo. Podrían utilizarse **heurísticos** junto con un espacio de **estados** que representen el problema a resolver de detectar puntos cercanos. También se podrían utilizar **redes neuronales** o **algoritmos genéticos** para comparar las coordenadas de los itinerarios y así detectar contactos estrechos.

Este enfoque no se llevó a cabo desde un principio dada su complejidad y alto consumo de tiempo, pues es necesario diseñar previamente el espacio de estados, la red neuronal…*etc.*, además de dar soporte a dichas estructuras por código. Sin embargo, sería una alternativa más eficiente que la implementación clásica.

12.2.6 Desarrollo multiplataforma para otros dispositivos

Actualmente, la versión móvil de *Contact Tracker* solo está disponible para algunos dispositivos que tengan instalado *Android*, lo cual fue planificado desde un principio. Sin embargo, sería interesante aumentar el alcance de la aplicación para que funcionase con otros sistemas operativos, es decir, una aplicación **multiplataforma**. De este modo, la aplicación estaría disponible para *iOS* e incluso *Windows Phone*, cubriendo una mayor parte de la población y haciendo el rastreo más efectivo.

Para el desarrollo multiplataforma se pueden utilizar *frameworks* como ***Flutter*** o ***Xamarin Forms***, los cuales permiten desarrollar versiones de la misma aplicación para distintas plataformas móviles, todo ello abstrayéndose de los detalles de bajo nivel de los *frameworks* nativos.

12.2.7 Pruebas de rendimiento y de accesibilidad

Las pruebas de rendimiento y accesibilidad no estaban previstas desde un primer momento, dado que el contexto del proyecto hace que no sean excesivamente necesarias. En cambio, podrían realizarse pruebas de rendimiento en diferentes puntos de la aplicación. Por un lado, se podría probar el límite del sistema en la **notificación** de **positivos** entre varios usuarios simultáneos con pruebas de carga y de estrés. Estas pruebas también podrían realizarse para probar el límite de recuperación de positivos para realizar comprobaciones. Por otro lado, también sería interesante realizar pruebas de carga para el **algoritmo de comprobación**, de forma que se compruebe cómo crece el tiempo de ejecución a medida que se va aumentando la carga, que en este caso coincidiría con el número de localizaciones que se comparan.

Las pruebas de accesibilidad se pueden aplicar a las interfaces gráficas de usuario tanto de la aplicación *móvil* como de la aplicación *web*, aunque deberían de ser más exhaustivas en la

aplicación móvil, ya que será la que utilicen los ciudadanos. Por ello, se debe garantizar la máxima accesibilidad en esta aplicación para abarcar a todas las personas posibles, mientras que el panel de control *web* será utilizado por un número más reducido de usuarios, y por tanto estas pruebas no son tan cruciales.

12.2.8 Pruebas de interfaz de la aplicación *web*

Actualmente, la aplicación *web* carece de pruebas de interfaz de usuario ya que no son excesivamente necesarias, pues el público al que está destinada está limitado a los usuarios administradores.

Podría emplearse algún *software*, como, por ejemplo, el *framework* de *Selenium* para realizar pruebas de interfaz que integren la interfaz de usuario con el resto del sistema del *frontend*. De este modo, se garantizaría una mayor calidad del panel de control *web* además de una manera automatizada de verificar su comportamiento esperado, una vez se han realizado cambios en el código.

12.2.9 Pruebas de código

Para complementar la batería de pruebas desarrollada para los distintos módulos, se propone como ampliación incluir **pruebas de código** mediante *software* que analice el código estáticamente para detectar posibles vulnerabilidades de seguridad, riesgo de punteros nulos u otros aspectos relacionados con la calidad del código. Para ello, podría integrarse la herramienta ***SonarQube*** que evalúa la calidad del código y de la seguridad del mismo, de forma que con cada compilación se puedan comprobar que tipo de errores o advertencias de código ha detectado la herramienta y así poder subsanarlas.

Estas pruebas aportan como ventaja una mayor **calidad** del código y una mayor **seguridad** para evitar exponer vulnerabilidades que puedan ser explotadas por individuos externos que accedan al código fuente.

Capítulo 13. Presupuesto

En este capítulo se describe el proceso llevado a cabo para obtener una estimación aproximada del presupuesto del proyecto, comenzando por el modelo de empresa que realiza el proyecto seguido de las distintas partidas en las que se estructura. Inicialmente, se muestran las partidas correspondientes a las etapas iniciales de investigación y preparativos del proyecto, para luego presentar las partidas correspondientes a las etapas de implementación e implantación del sistema descritas en la sección 5.1.1 sobre las etapas del proyecto. De forma adicional, en otra partida se calcula lo que costaría mantener y mejorar el sistema de *Contact Tracker* durante un periodo de un año.

Las partidas del proyecto están desglosadas según la EDT presentada en la sección 5.1.1.1 EDT Y DIAGRAMA DE GANTT dedicada a la planificación del proyecto. Como ya se comentó en dicha sección, el proyecto se enmarca dentro de una metodología de desarrollo ágil, por lo que el presupuesto no es tan exhaustivo como en un desarrollo tradicional, sino que el coste de las partidas se obtiene de manera aproximada en función de los meses de desarrollo de los *sprints* y el coste mensual que implica el trabajo del personal como se verá más adelante. Sin embargo, las primeras etapas del proyecto, previas al desarrollo en *sprints*, están más detallas y desglosadas en tareas cuya duración se ha estimado, por lo que en esos casos si se calcula el coste del trabajo en función de las horas de duración de las tareas planificadas y el coste por hora calculado.

Por último, es importante remarcar que el proyecto no está enfocado para ser entregado a un cliente en concreto, aunque se presenta por un lado el **presupuesto de costes,** es decir, el coste total del proyecto para la empresa (en este caso el único trabajador), y por otro, el **presupuesto de cliente**, que incluye los **beneficios** perseguidos que serán cargados al cliente que recibe el producto *software*.

13.1 Modelo de empresa

El desarrollo del proyecto es llevado a cabo por un equipo formado por un solo **ingeniero informático** que engloba los diversos roles que se pueden encontrar en un proyecto de *software*. Entre estos roles, está el rol de director de proyecto, arquitecto de *software*, analista de negocio, ingeniero de pruebas, *Scrum Master* u otros. Además, el trabajador debe estar formado en las principales tecnologías y herramientas utilizadas en el proyecto, como son el desarrollo de aplicaciones móviles con el *framework* de *Android*, lenguaje *Kotlin*, *Node.js*, *Vue.js* y las plataformas de *Azure* y *Firebase*.

Para calcular el salario aproximado que puede tener un ingeniero de *software* con estos conocimientos, con **un año** de experiencia y situado en España se ha hecho uso de la **calculadora de salarios de *StackOverflow* [68]** obteniéndose los siguientes resultados.

Tabla 13.1. Salarios medios de un Ingeniero Informático obtenidos con la calculadora de Stack Overflow.

Parámetros para el cálculo del salario medio (Stack Overflow)			Resultado		
Tecnologías	Experiencia	Localización	Poco común	Media	Más común
Android, Kotlin, Vue.js, Node.js, Firebase	1 año	Madrid, España	27.000,00 €	35.000,00 €	47.000,00 €

Como se puede observar, se obtienen tres valores, salario poco común, medio y el más común. Para el cálculo del presupuesto se tomará como referencia el salario **medio** obtenido con esta calculadora.

Además, cabe destacar que estos resultados están basados en los datos obtenidos con la **encuesta anual de desarrolladores** del año **2019** realizada por *Stack Overflow* para programadores, por lo que estos salarios son una aproximación que puede diferir de los sueldos actuales.

Con este salario medio, y teniendo en cuenta una jornada laboral de lunes a viernes, 8 horas al día, 5 días por semana, con aproximadamente 20 días de trabajo mensuales, se obtiene la siguiente tabla.

Tabla 13.2. Sueldo por hora de un Ingeniero Informático con una jornada laboral de 40 horas semanales.

Personal	Sueldo Bruto Anual	Sueldo Bruto Mensual	Sueldo por hora
Ingeniero Informático	35.000,00 €	2.916,67 €	18,23 €

13.1.1 Productividad

En cuanto a la **productividad** del personal, el trabajador encargado de desarrollar el proyecto tiene otras tareas transversales externas por lo que no estará trabajando al máximo rendimiento en todo momento, además de los contratiempos que puedan surgir durante el avance del proyecto. Con ello, se ha estimado de forma optimista una productividad del **85 %**, aplicando este porcentaje al sueldo mensual del trabajador se calculan los **costes directos** e **indirectos** por **mes** y **anualmente**, tal y como se muestra en la siguiente tabla.

Tabla 13.3. Costes directos e indirectos en función de la productividad del Ingeniero Informático.

Personal	Sueldo Bruto		Productividad	Coste Directo		Coste Indirecto	
	Anual	Mensual		Anual	Mensual	Anual	Mensual
Ingeniero Informático	35.000,00 €	2.916,67 €	85%	**29.750,00 €**	**2.479,17 €**	**5.250,00 €**	**437,50 €**

Los costes directos e indirectos se calculan multiplicando el porcentaje de productividad por el sueldo bruto. Por otro lado, también se ha calculado el **total** de **horas productivas** al **año** durante las cuales el ingeniero informático genera trabajo real. Para ello, se contempla un total de **1920 horas** anuales de trabajo, calculadas multiplicando las 8 horas diarias de trabajo por 20 días mensuales y por 12 meses, a las cuales se aplica el porcentaje de productividad para obtener las horas de trabajo productivo anuales, lo cual se ve reflejado en la siguiente tabla.

Tabla 13.4. Facturación de la empresa en función del porcentaje de beneficio.

Personal	Productividad (%)	Horas/año	Horas productivas / año	Precio / hora		Facturación
				Sin beneficios	Con beneficios	
Ingeniero Informático	85%	1920	1632	**18,23 €**	**19,14 €**	31.237,50 €

En la tabla anterior, también se muestra el **precio/hora** correspondiente al sueldo del ingeniero informático, primero sin beneficios y luego aplicando el **porcentaje de beneficios** perseguido por la **empresa**, el cual se verá en las necesidades de facturación. Multiplicando el número de horas productivas al año por el precio/hora con beneficios se obtiene la **facturación** de la empresa, es decir, la estimación del dinero que produciría en un año.

13.1.2 Medios de producción

En esta sección se presentan los recursos, servicios o licencias utilizadas para el desarrollo de las tareas del proyecto y cuyo coste incurrirá en los **costes indirectos** del proyecto como se verá más adelante. Se distingue entre tres tipos de recursos:

- **Alquiler**. Recursos cuyo importe se abona por uso, es decir, se abona el coste del recurso mensualmente, o en función de la cantidad del recurso que se utilice. El coste de los recursos de alquiler se calcula en función de las tarifas establecidas por las entidades que alquilan el recurso.
- **Amortización**. Recursos como equipos o material adquirido que se puede amortizar a lo largo de un plazo, es decir, se puede aprovechar su uso para otros proyectos durante varios años. En este caso, el coste del recurso al año se calcula dividiendo el coste total del recurso entre el número de años que se estima será amortizado.
- **Material**. Recursos de un solo uso que son consumidos en el desarrollo del proyecto.

Aunque el desarrollo del prototipo de *Contact Tracker* se realizará mayoritariamente con herramientas y servicios **gratuitos**, para el presupuesto se considera el coste por uso de las herramientas o servicios *premium* que deberían ser utilizados en un supuesto real.

El coste anual de los medios de producción se sumará a los costes indirectos del proyecto para deducir las necesidades de facturación de la empresa. La siguiente tabla muestra los medios de producción y sus costes para el proyecto de *Contact Tracker*.

Tabla 13.5. Medios de producción.

Recurso, Servicio o Licencia	Uds	Tipo	Precio unitario	Coste Total	Coste Anual	Coste Mensual	Plazo
Tarifa de *Internet* Vodafone (Fibra óptica de 600 Mb)	1	Alquiler	35,00 €	35,00 €	420,00 €	35,00 €	
Equipo de sobremesa	1	Amortización	825,00 €	825,00 €	103,13 €	8,59 €	8
Monitor HD 60Hz	2	Amortización	120,00 €	240,00 €	30,00 €	2,50 €	8
Dispositivo móvil Android	2	Amortización	250,00 €	500,00 €	100,00 €	8,33 €	5
Cloud Firestore (Firebase)	1	Alquiler	24,70 €	24,70 €	296,40 €	24,70 €	
Licencia GitHub Team	1	Alquiler	3,40 €	3,40 €	40,80 €	3,40 €	
Azure App Service Premium V2 (Instancia P1V2)	1	Alquiler	154,53 €	154,53 €	1.854,36 €	154,53 €	
Azure Cuenta de Almacenamiento Estándar V2 para la App Web	1	Alquiler	17,30 €	17,30 €	207,60 €	17,30 €	

Vue.js							
Tarifa de cuenta de desarrollador de Google Play Store para publicar aplicaciones Android	1	Material	21,23 €	21,23 €	21,23 €	- €	
				Total	**3.073,52 €**	**254,36 €**	

En esta tabla, el precio de los recursos de *Azure* se ha calculado utilizando la **calculadora de precios** de los servicios de *Azure* **[65]**. Por un lado, la instancia del ***App Service*** para hospedar la *API REST* se encuentra en Francia Central y se ha escogido el nivel *Premium V2* con 1 núcleo de procesamiento (***P1V2***). Por otro, también se utiliza una **Cuenta de almacenamiento** para hospedar la aplicación *web* con un nivel de rendimiento estándar y de uso general *V2*. Además, para poder publicar el *apk* de la aplicación *Android* en la *Play Store* es necesario crear una **cuenta de desarrollador de *Google Play Store***, lo cual implica abonar una tarifa única.

Por último, la base de datos en la nube ***Firestore*** que se utiliza para la construcción del prototipo tiene un plan gratuito denominado ***Spark***, que presenta ciertas limitaciones de velocidad y capacidad de almacenamiento. Para un desarrollo en un entorno profesional se debería utilizar el plan ***Blaze*** de **pago por uso**, que es el que se contempla en la tabla anterior de medios de producción.

Para calcular el coste por uso de la base de datos *Firestore* se utiliza la tabla de precios del plan *Blaze* **[66]**. Estas tarifas de uso son diferentes en función de la operación que se realice (lectura, escritura…) y pueden verse en la siguiente tabla.

Tabla 13.6. Coste por uso de las operaciones de Firestore.

Lectura / 100k documentos	Escritura / 100k documentos	Borrados / 100k documentos	Almacenamiento / GiB / mes
0,05 €	0,15 €	0,01 €	0,15 €

Teniendo en cuenta los precios mostrados en esta tabla, se calcula el coste **mensual** de utilizar *Firestore* como si se tratase de un sistema real puesto en producción, estableciendo un número de lecturas, escrituras, borrados y *GiB* de almacenamiento aproximados que pueden variar mucho en función del uso en un mes determinado. Se estiman alrededor de 10 millones de lecturas y escrituras, 1 millón de borrados y unos 30 *GiB* de almacenamiento al mes con la aplicación en producción y siendo utilizada por la mayoría de la población española. Esto se puede ver en la siguiente tabla.

Tabla 13.7. Coste total de uso de Firestore al mes.

	Lecturas / mes	Escrituras / mes	Borrados / mes	GiB almacenamiento / mes	Total / mes
Documentos o GiB	10.000.000	10.000.000	1.000.000	30	**24,70 €**
Coste	5,10 €	15,00 €	0,10 €	4,50 €	

13.1.3 Necesidades de facturación

Con el sueldo bruto, las horas productivas y los medios de producción vistos en las secciones anteriores se derivan las **necesidades de facturación** de la empresa. Se contempla un **5 %** de **beneficio** que la empresa pretende alcanzar en un año. Aplicando este porcentaje a la suma de costes directos e indirectos se calcula el beneficio monetario que se obtendría. Estos datos se ven en la siguiente tabla.

Costes directos	29.750,00 €
Costes indirectos	8.323,52 €
CD+CI	38.073,52 €
Beneficio deseado (%)	**5%**
Beneficio monetario	**1.903,68 €**

Ilustración 13.1. Relación de costes directos e indirectos junto con el porcentaje de beneficio.

Con ello, las **necesidades** de **facturación** para que la empresa obtenga dichos beneficios se calculan sumando los costes directos, indirectos y el beneficio monetario. Haciendo el **cociente** entre las necesidades de facturación y la **facturación posible** calculada en la sección de Productivdad se obtiene el **margen**, tal y como se muestra a continuación.

Facturación posible	31.237,50 €
Necesidades de facturación	39.977,19 €
Margen entre la facturación posible y las necesidades de facturación	**-0,279782017**

Ilustración 13.2. Margen entre la facturación y las necesidades de facturación.

Como se puede ver, el margen es negativo, lo cual se debe a que la empresa está formada por un solo trabajador con una productividad limitada. Esto hace que sea difícil alcanzar la facturación anual necesaria para afrontar los gastos y así obtener el 5 % de beneficios deseados. Si se añaden trabajadores o se reduce el beneficio deseado, el margen comenzaría a aumentar pasando a ser positivo, lo cual significa que la cantidad que factura la empresa en un año es **superior** a las necesidades de facturación.

Para que los costes indirectos de los medios de producción incurran en el **precio/hora** del trabajador y así se tengan en cuenta en el coste del proyecto, se realiza el cociente entre la suma de **costes directos e indirectos** y el número total de **horas de trabajo** al **año** correspondiente a la jornada de 8 horas/día, 20 días al mes aproximadamente (teniendo en cuenta meses de 28 días), tal y como se muestra a continuación.

Horas / año	1920
Precio/hora	**19,83 €**

Ilustración 13.3. Cálculo del precio/hora del ingeniero informático en función de los costes directos e indirectos.

13.2 Partidas

En esta sección se describen brevemente las distintas partidas que componen el presupuesto sin entrar en detalles de las tareas en las que se desglosan y sus costes. Para conocer el máximo detalle sobre las partidas desglosadas véase los anexos correspondientes a cada partida en la sección 15.4 de Anexos en los apéndices de la documentación. A continuación, se describen las partidas del presupuesto.

- PARTIDA 1: ETAPA DE INVESTIGACIÓN. Engloba todas las tareas de investigación para recabar información sobre todos los aspectos involucrados en el proyecto, como aspectos legales, herramientas, tecnologías *Android*…
- PARTIDA 2: ETAPA DE PREPARACIÓN DEL ENTORNO Y ANÁLISIS/DISEÑO PRELIMINAR. Contiene las tareas relacionadas con los preparativos de los entornos de trabajo, repositorios, plataformas de *Firebase* y de *Azure*…etc., además del análisis y diseño preliminar para definir las principales funcionalidades del sistema.
- PARTIDA 3: ETAPA DE IMPLEMENTACIÓN. Está formada por las distintas iteraciones o *sprints* que conforman el desarrollo del sistema. Al utilizar una metodología ágil, no se desglosan las iteraciones en tareas pequeñas ni se planifican, por lo que el coste del desarrollo se estima en base al sueldo mensual del trabajador y al número de iteraciones necesarias.
- PARTIDA 4: ETAPA DE IMPLANTACIÓN DEL SISTEMA. Tareas de despliegue y puesta en marcha de los módulos que componen el sistema, incluyendo también la formación del personal sanitario para poder utilizar el panel de control *web*.
- PARTIDA 5: ETAPA DE MANTENIMIENTO DEL SISTEMA. Involucra las tareas de mantenimiento, mejora y ampliación del prototipo de *Contact Tracker* para ponerlo en producción durante un periodo de un año.

13.3 Presupuesto de Costes

Agrupando las partidas descritas anteriormente se obtiene el presupuesto de costes que representa el coste total del proyecto para la empresa que lo desarrolla, sin incluir los beneficios. La siguiente tabla muestra este presupuesto de costes para el proyecto de *Contact Tracker*.

Tabla 13.8. Presupuesto de costes del proyecto.

PRESUPUESTO DE COSTES

CÓDIGO	Partida	Subtotal	Total
1	**Etapa de Investigación**		1.368,27 €
2	**Etapa de Preparación del Entorno y Análisis/Diseño preliminar**		773,37 €
2.1	Preparación del Entorno	475,92 €	
2.2	Análisis/Diseño preliminar	297,45 €	
3	**Etapa de Implementación**		12.691,17 €
4	**Etapa de Implantación del Sistema y Formación**		297,45 €
	COSTE TOTAL DEL DESARROLLO DEL PROTOTIPO		**15.130,26 €**
5	**Etapa de Mantenimiento del Sistema**		38.073,52 €

COSTE TOTAL	53.203,77 €

Como se puede observar, el coste total del desarrollo del **prototipo** sin contemplar el **año de mantenimiento** es de unos **15.130,26 €**. Teniendo en cuenta el mantenimiento y ampliación del sistema durante un periodo de un año, el coste total asciende hasta los **53.203,77 €.**

13.4 Presupuesto de Cliente

Para obtener el presupuesto de cara al **cliente**, se toma como base el presupuesto de costes anterior y se realiza una **ponderación de costes** para el cliente en función del porcentaje de **beneficios** marcado como objetivo para el proyecto. En este caso, se ha marcado como objetivo alcanzar un **20 %** de beneficios sobre el coste del proyecto. Aplicando este porcentaje al coste total del proyecto, se obtiene que el beneficio monetario a alcanzar es de unos **10.640,75 €**, como se puede ver en la siguiente tabla.

Otros costes (costes de investigación, preparación y análisis/diseño preliminar)	**2.141,64 €**
Porcentaje de beneficio deseado sobre el proyecto	**20%**
Beneficio monetario sobre el proyecto	**10.640,75 €**

En la primera fila aparecen los costes de las partidas de **investigación, preparación** y **análisis/diseño preliminar** agrupados bajo ***Otros costes***. Esto es así porque estas partidas no se mostrarán al cliente, sino que aparecerán ocultas, aunque su coste tenga una repercusión sobre el presupuesto. Para ello, es necesario calcular el **valor a promediar**, es decir, la cuantía económica que se va a repartir entre el resto de las partidas visibles para el cliente de forma que estos costes incurran en el presupuesto.

El valor a promediar es la suma de ***Otros costes*** y el **beneficio monetario**, lo cual da lugar a un total de **12.782,39 €.** Para repartir esta cuantía, es necesario calcular el **coeficiente de ponderación** que se utilizará para calcular el coste de las partidas del presupuesto del cliente. Esto se consigue mediante una regla de tres entre el total de costes sin contar los *Otros costes* y el valor a promediar, tal y como se muestra a continuación.

Total de costes (Sin Otros costes) 51.062,14 € ⟶ 100 %

Valor a promediar (Otros costes + beneficio) 12.782,39 € ⟶ **25,033 %**

Ilustración 13.4. Relación de ponderación de costes en función del porcentaje de beneficio.

Con ello se obtiene un porcentaje de alrededor del **25 %**, es decir, un **coeficiente de ponderación** de **0,25.** Aplicando este porcentaje a las partidas visibles para el cliente, se obtiene el presupuesto del cliente que se muestra en la siguiente tabla.

Tabla 13.9. Presupuesto de cliente del proyecto.

PRESUPUESTO DE CLIENTE

CÓDIGO	Partida	Subtotal	Total
1	**Etapa de Implementación**		15.868,15 €
2	**Etapa de Implantación del Sistema y Formación**		371,91 €
	COSTE TOTAL DEL DESARROLLO DEL PROTOTIPO		**16.240,06 €**
3	**Etapa de Mantenimiento del Sistema**		47.604,46 €
		COSTE TOTAL	**63.844,53 €**

De este modo, el cliente debe abonar una cantidad total de **16.240,06 €** por el desarrollo del prototipo del sistema, y junto con el año de mantenimiento, el coste final del proyecto asciende hasta los **63.844,53 €.**

13.5 Seguimiento del presupuesto

Una vez se ha visto el presupuesto **planificado** para el proyecto, en esta sección se presenta el resultado final una vez se ha ejecutado el proyecto. Como se vio en el CAPÍTULO 6 dedicado a la Evolución y seguimiento de proyecto, hubo diversos contratiempos que obligaron a dedicar **tres iteraciones extra** además de las planificadas para poder completar el desarrollo del proyecto.

Si se tiene en cuenta que el coste planificado de cada iteración es de alrededor de **3.172,79 €**, esto supone unas **pérdidas** que ascienden hasta los **9.518,37 €**, ya que dichos *sprints* necesarios para completar el proyecto no estaban previstos en el presupuesto planificado inicialmente.

Para evitar estas posibles pérdidas debido principalmente a los contratiempos, hubiera sido interesante **reservar** una porción del dinero para posibles urgencias, es decir, lo que se denominan **reservas para contingencias.**

Capítulo 14. Referencias Bibliográficas

14.1 Libros y Artículos

1. **[MinisterioDeSanidad21]** "Enfermedad por coronavirus, COVID-19". Secretaria de Estado de Sanidad; Dirección General de Salud Pública, Calidad e Innovación. 2021.
2. **[MinisterioDeSanidad21]** "Estrategia de detección precoz, vigilancia y control de COVID-19". Instituto de Salud Carlos III. 2021.
3. **[BoletínOficialDelEstado]** "Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales". Cortes Generales. 2018.
4. **[DiarioOficialUE].** "REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos)". Parlamento Europeo. 2016.

14.2 Referencias en Internet

1. Google Inc. "Exposure Notifications". https://www.google.com/covid19/exposurenotifications/. Última consulta: 12/11/2021. Sistema de Contact Tracing de Google y Apple.
2. NYTimes. Mozur, Paul; Zhong, Raymond; Krolik, Aaron. "In Coronavirus Fight, China Gives Citizens a Color Code, With Red Flags". https://www.nytimes.com/2020/03/01/business/china-coronavirus-surveillance.html. Última consulta 12/11/2021. Aplicación de rastreo de China.
3. GitHub. "PEPP-PT". https://github.com/pepp-pt/pepp-pt-documentation. Última consulta: 12/11/2021. Modelo de rastreo centralizado.
4. GitHub. "DP3T". https://github.com/DP-3T/documents. Última consulta: 12/11/2021. Modelo de rastreo descentralizado.
5. OpenWebinars. Montero, Ortega, José Manuel. "La arquitectura MVVM y sus componentes". https://openwebinars.net/blog/la-arquitectura-mvvm-y-sus-componentes/. Última consulta: 12/11/2021. Arquitectura MVVM.
6. CDC. "Centros para el Control y la Prevención de Enfermedades". https://www.cdc.gov/spanish/index.html. Última consulta: 12/11/2021. Página oficial del CDC.
7. BMJ. R. Jones; Nicolas. "Two metres or one: what is the evidence for physical distancing in covid-19?". https://www.bmj.com/content/370/bmj.m3223. Última consulta: 12/11/2021. Artículo sobre el estudio de la distancia de seguridad.
8. Servicio de Salud del PA. "Manejo en Atención Primaria de COVID-19 en Asturias". https://www.astursalud.es/documents/35439/39435/Manejo+en+Atencion+Primaria+de+COVID-19+en+Asturias+042021.pdf/7bdf1db6-1882-76c2-0027-e8bec79be479?t=1621866639152. Última consulta: 12/11/2021
9. Guillermo Aldama. Aldama, Guillermo. "Fases de la Infección por Coronavirus, Carga Viral y Resultados de los Test". https://twitter.com/Guillermo4ldama/status/1340399228380733441/photo/1. Última consulta: 12/11/2021. Gráfica del periodo de infectividad.
10. Android Inc. "Android Open Source Project". https://source.android.com/. Última consulta: 12/11/2021. Documentación sobre AOSP.
11. Android Inc. "Android Docs". https://www.android.com/. Última consulta: 12/11/2021.
12. Android Inc. "Build location-aware apps". https://developer.android.com/training/location. Última consulta: 12/11/2021. Documentación sobre cómo utilizar la geolocalización en Android.
13. Android Inc. "LocationManager". https://developer.android.com/reference/android/location/LocationManager. Última consulta: 12/11/2021. Documentación sobre el Location Manager.
14. Google Inc. "Fused Location Provicer API". https://developers.google.com/location-context/fused-location-provider. Última consulta: 12/11/2021.
15. Android Inc. "Guide to background processing". https://developer.android.com/guide/background. Última consulta: 12/11/2021. Guía para las tareas en 2º plano.

16. Android Inc. "Services overview". https://developer.android.com/guide/components/services. Última consulta: 12/11/2021. Documentación sobre los servicios Android.

17. Android Inc. "Broadcasts overview". https://developer.android.com/guide/components/broadcasts. Última consulta: 12/11/2021. Guía sobre los Broadcast Receivers.

18. Android Inc. "Schedule alarms". https://developer.android.com/training/scheduling/alarms. Última consulta: 12/11/2021. Guía sobre programación de alarmas en Android.

19. Android Inc. "Optimize for Doze and App Standby". https://developer.android.com/training/monitoring-device-state/doze-standby. Última consulta: 12/11/2021.

20. Android Inc. "Background Execution Limits". https://developer.android.com/about/versions/oreo/background. Última consulta: 12/11/2021. Restricciones de ejecución en 2º plano.

21. Android Inc. "Access location in the background". https://developer.android.com/training/location/background. Última consulta: 12/11/2021. Localización en 2º plano.

22. Google Inc. "Firebase". https://firebase.google.com/?hl=es. Última consulta: 11/11/2021. Documentación relativa a la plataforma de Firebase.

23. OpenJS. "Nodejs". https://nodejs.org/es/. Última consulta: 12/11/2021. Documentación relativa a Node.js.

24. OpenJS. "Express.js". https://expressjs.com/es/. Última consulta: 12/11/2021. Documentación relativa a Express.js.

25. Evan You. "Vue.js". https://vuejs.org/. Última consulta: 12/11/2021. Página oficial de Vue.js.

26. Android Inc. "Android Studio". https://developer.android.com/studio. Última consulta: 12/11/2021. Página oficial del IDE de Android Studio.

27. Android Inc. "Save data in a local database using Room". https://developer.android.com/training/data-storage/room. Última consulta: 12/11/2021.

28. Square. "Retrofit". https://square.github.io/retrofit/. Última consulta: 12/11/2021. Documentación sobre Retrofit.

29. Axios. "Axios". https://axios-http.com/docs/intro. Última consulta: 12/11/2021. Documentación sobre AXIOS.

30. Google Inc. "Firebase Cloud Messaging". https://firebase.google.com/docs/cloud-messaging?hl=es. Última consulta: 12/11/2021. Documentación sobre FCM.

31. Android Inc. "Maps SDK for Android". https://developers.google.com/maps/documentation/android-sdk/start. Última consulta: 12/11/2021. Documentación sobre el SDK de Google Maps para Android.

32. OpenJS. "Mocha". https://mochajs.org/. Última consulta: 12/11/2021. Página oficial de Mocha.js.

33. Chai. "Chai Assertion Library". https://www.chaijs.com/. Última consulta: 12/11/2021. Página oficial de Chai.js.

34. Chai. "Chai HTTP". https://www.chaijs.com/plugins/chai-http/. Última consulta: 12/11/2021. Página oficial del plugin Chai-http.

35. Android Inc. "Espresso". https://developer.android.com/training/testing/espresso. Última consulta: 13/11/2021. Documentación sobre Espresso.

36. JUnit. "Junit 5". https://junit.org/junit5/. Última consulta: 13/11/2021. Página oficial de JUnit.

37. Mockito. "Tasty mocking framework for unit tests in Java". https://site.mockito.org/. Última consulta: 13/11/2021.

38. JetBrains. "Kotlin". https://kotlinlang.org/. Última consulta: 13/11/2021. Página oficial de Kotlin.

39. JetBrains. "Coroutines". https://kotlinlang.org/docs/coroutines-overview.html. Última consulta: 13/11/2021. Documentación sobre las corrutinas de Kotlin.

40. Android Inc. "Data Binding Library". https://developer.android.com/topic/libraries/data-binding. Última consulta: 13/11/2021. Documentación sobre Data Binding.

41. Android Inc. "View Binding". https://developer.android.com/topic/libraries/view-binding. Última consulta: 13/11/2021. Documentación sobre View Binding.

42. Android Inc. "Navigation". https://developer.android.com/guide/navigation. Última consulta: 13/11/2021. Documentación sobre la navegación en Android.

43. Microsoft Inc. "Azure DevOps". https://azure.microsoft.com/es-es/services/devops/es. Última consulta: 13/11/2021. Documentación sobre DevOps.

44. RedHat. "¿Qué son la integración/distribución continuas (CI/CD)?". https://www.redhat.com/es/topics/devops/what-is-ci-cd. Última consulta: 13/11/2021.

45. Android Inc. "Guía de arquitectura de apps". https://developer.android.com/jetpack/guide?hl=es-419. Última consulta: 13/11/2021.

46. Red Hat. ¿Qué es una API de REST?". https://www.redhat.com/es/topics/api/what-is-a-rest-api. Última consulta: 13/11/2021.

47. C. Martin, Robert; Fowler, Martin; Otros. "Manifiesto por el Desarrollo Ágil de Software". https://agilemanifesto.org/iso/es/manifesto.html. Última consulta: 13/11/2021.

48. Scrum. "Scrum". https://www.scrum.org/resources/what-is-scrum. Última consulta: 13/11/2021. Página oficial de Scrum.

49. AEPD. "Agencia Española de Protección de Datos". https://www.aepd.es/es. Última consulta: 13/11/2021. Página oficial de la AEPD.

50. GrupoAtico34. "Política de privacidad web". https://protecciondatos-lopd.com/empresas/politica-de-privacidad-web/#Que_debo_incluir_en_la_politica_de_privacidad_de_mi_pagina_web. Última consulta: 13/11/2021.

51. IEEE. "IEEE 830-1998". https://standards.ieee.org/standard/830-1998.html. Última consulta: 13/11/2021. Estándar para la documentación de requisitos de *software*.

52. MartinFowler. Fowler, Martin; "The practical test pyramid". https://martinfowler.com/articles/practical-test-pyramid.html. Última consulta: 13/11/2021.

53. AlistairCockburn. Cockburn, Alistair. "Hexagonal Arquitecture". https://alistair.cockburn.us/hexagonal-architecture/. Última consulta: 13/11/2021. Artículo sobre la arquitectura hexagonal.

54. Android Inc. "Dependency Injection with Hilt". https://developer.android.com/training/dependency-injection/hilt-android. Última consulta: 13/11/2021.

55. CueMath. *"Great Circle Formula"*. https://www.cuemath.com/great-circle-formula/. Última consulta: 13/11/2021.

56. ESRI. "Distance on a Sphere: The Haversine Formula". https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128. Última consulta: 13/11/2021.

57. Medium. Dejan, Giovanni. "The Magic of Haversine Distance". https://medium.com/ninjavan-tech/the-magic-of-haversine-distance-1c4e1641d880. Última consulta: 13/11/2021.

58. Medium. Sarkar, Prokash. "Unit testing with Kotlin Coroutines: The Android Way". https://medium.com/swlh/unit-testing-with-kotlin-coroutines-the-android-way-19289838d257. Última consulta: 13/11/2021.

59. AanandShekhar. Shekhar Roy, Aanand. "Know your sensors – Android Location Services". https://www.aanandshekharroy.com/articles/2018-01/android-location-sensors. Última consulta: 13/11/2021. Descripción de los sensores de localización de Android.

60. Medium. Bhandari, Nikit. "Using AlarmManger like a pro". https://medium.com/android-news/using-alarmmanager-like-a-pro-20f89f4ca720. Última consulta: 13/11/2021. Guía sobre el AlarmManager.

61. CDC. "Rastreo de contactos". https://espanol.cdc.gov/coronavirus/2019-ncov/daily-life-coping/contact-tracing.html. Última consulta: 13/11/2021.

62. CDC. "Síntomas del COVID-19". https://espanol.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html. Última consulta: 13/11/2021.

63. SimplifiedCoding. Khan, Belal. "Android ViewModel Test Tutorial". https://www.simplifiedcoding.net/android-viewmodel-unit-test/. Última consulta: 13/11/2021.

64. DigitalOcean. Zaza, Samuele. "Test a Node RESTful API with Mocha and Chai". https://www.digitalocean.com/community/tutorials/test-a-node-restful-api-with-mocha-and-chai. Última consulta: 13/11/2021.

65. Microsoft. "Calculadora de precios de Azure". https://azure.microsoft.com/es-es/pricing/calculator/. Última consulta: 13/11/2021.

66. Google Inc. "Understand Cloud Firestore Billing". https://firebase.google.com/docs/firestore/pricing. Última consulta: 13/11/2021.

67. EFF. Cyphers, Bennett; Gebhart, Gennie. "La API de notificación de exposición de COVID-19 de Apple y Google: Preguntas y respuestas". https://www.eff.org/es/deeplinks/2020/04/apple-and-googles-covid-19-exposure-notification-api-questions-and-answers. Última consulta: 17/11/2021. Información sobre la API de Contact Tracing.

68. StackOverflow. "Job Salary Calculator". https://stackoverflow.com/jobs/salary. Última consulta: 19/11/2021. Calculadora de salarios de puestos *software*.

Capítulo 15. Apéndices

15.1 Glosario y Diccionario de Datos

- **API:** de sus siglas *Application Programming Interface*, representa la interfaz que define el contrato de un sistema, servicio o componente *software* para poder hacer uso de sus funcionalidades y operar con este desde otro sistema, servicio o componente externo.
- **Base de datos documental:** tipo de base de datos en la que los datos se organizan en documentos flexibles agrupados en colecciones sin un esquema o restricciones determinadas.
- **Build:** código empaquetado que representa un hito o versión nueva de un producto *software* y que puede ser desplegado sobre una plataforma.
- **Clase:** unidad lógica que encapsula código con responsabilidades comunes y que sirve como una plantilla con atributos y métodos para crear instancias u objetos con esas operaciones.
- **Despliegue:** proceso llevado a cabo tras la instalación de un sistema para ponerlo en marcha y hacer que esté disponible para los usuarios finales.
- **EDT:** siglas de Estructura de Desglose de Tareas. Consiste en la descomposición jerárquica de las tareas identificadas dentro de un proyecto en otras más pequeñas y manejables.
- **Entorno de ejecución:** contexto en el que se ejecuta un sistema, pudiendo ser el entorno de desarrollo con datos de prueba o bien el entorno de producción que representa la ejecución del sistema en su ambiente real.
- **Git:** sistema de control de versiones descentralizado para gestionar el código fuente y sus cambios a través de repositorios remotos y locales.
- **HTTP:** de las siglas *Hyper Text Transfer Protocol*, es un protocolo de transferencia de datos a través de *internet* utilizado en los sistemas distribuidos para leer y escribir datos entre sistemas remotos conectados a *internet*.
- **IDE:** de sus siglas *Integrated Developmente Enviroment*, se trata de un conjunto de herramientas y facilidades que definen el entorno de desarrollo utilizado para codificar los programas.
- **Integración continua:** conjunto de procesos y actividades transversales a la implementación de un sistema para integrar el código fuente, es decir, someterlo a diversos tipos de pruebas y análisis, ejecutar los *tests*, unir componentes o desplegar *builds* entre otros, todo ello de forma automática con cada cambio en el código.
- **JSON:** de sus siglas *Javascript Object Notation* define un formato de organización de los datos en listas y objetos con campos, que a su vez pueden ser otras listas u objetos, para ser procesados por los sistemas distribuidos en *internet*.
- **Paquete:** organización lógica de varias clases que tienen unas responsabilidades comunes y que trabajan de manera conjunta para realizar una tarea determinada.
- **Patrón de diseño**: técnica o solución reutilizable que define un estándar adoptado por la comunidad para solventar problemas a la hora de implementar algunos aspectos que se repiten en muchos desarrollos de *software.*

- **Rastreador:** individuo, no necesariamente con conocimientos sanitarios, encargado de rastrear los contactos estrechos de los ciudadanos con positivos en *COVID-19.*
- **REST:** de sus siglas *Representational State Transfer*, es una arquitectura que define un estándar de conexión y transferencia de datos entre sistemas distribuidos en la red separando las responsabilidades de cliente y servidor.
- **SGBD:** de sus sigas Sistema Gestor de Base de Datos, es un sistema que actúa de intermediario entre un cliente y una base de datos para gestionar las peticiones y consultas realizadas, además de garantizar que se cumplan las restricciones de integridad y de velar por la seguridad de los datos.
- **Stakeholder:** persona, entidad, servicio o componente que influye directa o indirectamente en el sistema bajo construcción o bien que recibe su influencia o está relacionado con este.
- **UML:** de sus siglas *Unified Modeling Language,* es un estándar de modelado de *software* que define una serie de lenguajes, notaciones y símbolos estándar para dibujar diagramas estructurales y de comportamiento del código.
- **Verbo HTTP**: se refiere a una acción u operación dentro del protocolo *HTTP*, para recuperar recursos, actualizarlos o crearlos dentro de un servidor (*GET, UPDATE, POST…*)
- **WBS/PBS:** de sus siglas *Work Breakdown Structure* y *Product Breakdown Structure*, son unos diagramas empleados en la planificación de un proyecto para representar las tareas a realizar junto con sus productos derivados.

15.2 Contenido Entregado en el ZIP

15.2.1 Raíz del archivo comprimido

Directorio	Contenido
./Directorio raíz	Fichero "LEEME.txt" explicando esta misma estructura.
./Código fuente	Código fuente de los tres subsistemas que conforman el sistema de Contact Tracker.
./APKs	Contiene los APKs de la aplicación móvil de rastreo: - **ContactTracker_DEBUG_LOCAL**: apk que consume los servicios de un servidor web desplegado en local. - **ContactTracker_DEBUG_AZURE:** apk que consume los servicios de un servidor web desplegado en remoto en la plataforma de Azure.
./Documentación	PDF con la memoria del proyecto y las imágenes asociadas.

15.2.2 Código fuente

Directorio	Contenido
./Aplicación móvil	Contiene el fichero ZIP con el proyecto de Android Studio de la aplicación móvil. Contiene un fichero Instrucciones.txt que explica cómo abrir el proyecto en Android Studio y un fichero local.properties con las propiedades locales para copiar y pegar en el proyecto..
./Servidor web	Contiene la carpeta con el proyecto del servidor web para Visual Studio Code.
./Aplicación web	Contiene la carpeta con el proyecto de la aplicación web (panel de control web) para Visual Studio Code.

15.2.3 Documentación

Directorio	Contenido
./Directorio raíz de "Documentación"	Contiene el PDF de documentación con la memoria del proyecto y una carpeta con las imágenes asociadas. También contiene un fichero "Enlaces.txt" con las URLs de la aplicación web y de la API REST desplegada en Azure (servidor web), además del Excel utilizado para calcular el presupuesto del proyecto.
./Imágenes	Imágenes utilizadas en la memoria del proyecto.
./Imágenes/Diagramas/Análisis	Imágenes de los diagramas utilizados en el Análisis del sistema.
./Imágenes/Diagramas/Diseño	Imágenes de los diagramas utilizados en el Diseño del sistema.
./Imágenes/Pantallas/AppMóvil	Capturas de pantalla de la aplicación móvil de rastreo.
./Imágenes/Pantallas/AppWeb	Capturas de pantalla de la aplicación web.

15.3 Índice Alfabético

15.4 Anexos

15.4.1 Partida 1: Etapa de Investigación

Item	*Descripción*	*Duración*	*Unidades*	*Precio*	*Subtotal*
1	Estudio de Geolocalización en Android.	15	horas	19,83 €	297,45 €
2	Estudio de ejecución en 2º plano y consumo de batería en Android.	15	horas	19,83 €	297,45 €
3	Estudio de las versiones Android: limitaciones y restricciones de geolocalización y ejecución en 2º plano.	10	horas	19,83 €	198,30 €
4	Estudio de tecnologías y herramientas a utilizar (ROOM, MVVM, Firebase, Azure Dev Ops, Kotlin, AlarmManager, Corrutinas…).	15	horas	19,83 €	297,45 €
5	Estudio de antecedentes y sistema anterior.	2	horas	19,83 €	39,66 €
6	Estudio de la situación actual y sistemas existentes (RadarCovid, HealthKit).	4	horas	19,83 €	79,32 €
7	Estudio de la normativa, aspectos legales y protección de datos.	6	horas	19,83 €	118,98 €
8	Estudio del contexto sanitario: aspectos teóricos sobre la pandemia de COVID19.	2	horas	19,83 €	39,66 €
				Total	**1.368,27 €**

15.4.2 Partida 2: Etapa de Preparación del Entorno y Análisis/Diseño preliminar

I1	*I2*	*Descripción*	*Duración*	*Uds*	*Precio*	*Subtotal (2)*	*Subtotal (1)*
1		**Preparación del Entorno**					**475,92 €**
	1.1	Creación de los repositorios GIT.	2	horas	19,83 €	39,66 €	
	1.2	Creación y configuración del proyecto para la aplicación cliente Android.	4	horas	19,83 €	79,32 €	
	1.3	Creación y configuración del proyecto para la aplicación backend del servidor web (API REST).	3	horas	19,83 €	59,49 €	
	1.4	Creación y configuración del proyecto para la aplicación frontend (App Web).	3	horas	19,83 €	59,49 €	
	1.5	Creación y configuración del proyecto en Firebase (base de datos documental en la nube).	3	horas	19,83 €	59,49 €	
	1.6	Creación y configuración del proyecto en Google Cloud Console para usar la API de Google Maps.	1	horas	19,83 €	19,83 €	
	1.7	Creación del proyecto en Azure Dev Ops (tablero kanban, storymap).	3	horas	19,83 €	59,49 €	
	1.8	Configuración de la integración continua (CI) y despliegue (CD) con las pipelines de Azure DevOps.	5	horas	19,83 €	99,15 €	
2		**Análisis y Diseño preliminar**					**297,45 €**
	2.1	Análisis a alto nivel del alcance y objetivos del sistema.	3	horas	19,83 €	59,49 €	

2.2	Descripción a alto nivel de las principales funcionalidades del sistema.	4	horas	19,83 €	79,32 €	
2.3	Realización de Prototipos, Sketchs y bocetos iniciales de la arquitectura del sistema.	5	horas	19,83 €	99,15 €	
2.4	Diseño preliminar del modelo de datos.	3	horas	19,83 €	59,49 €	
					Total	**773,37 €**

15.4.3 Partida 3: Etapa de Implementación

Item	*Descripción*	*Duración*	*Unidades*	*Precio*	*Subtotal*
1	Sprint 1	1	meses	3.172,79 €	3.172,79 €
2	Sprint 2	1	meses	3.172,79 €	3.172,79 €
3	Sprint 3	1	meses	3.172,79 €	3.172,79 €
4	Sprint 4	1	meses	3.172,79 €	3.172,79 €
				Total	**12.691,17 €**

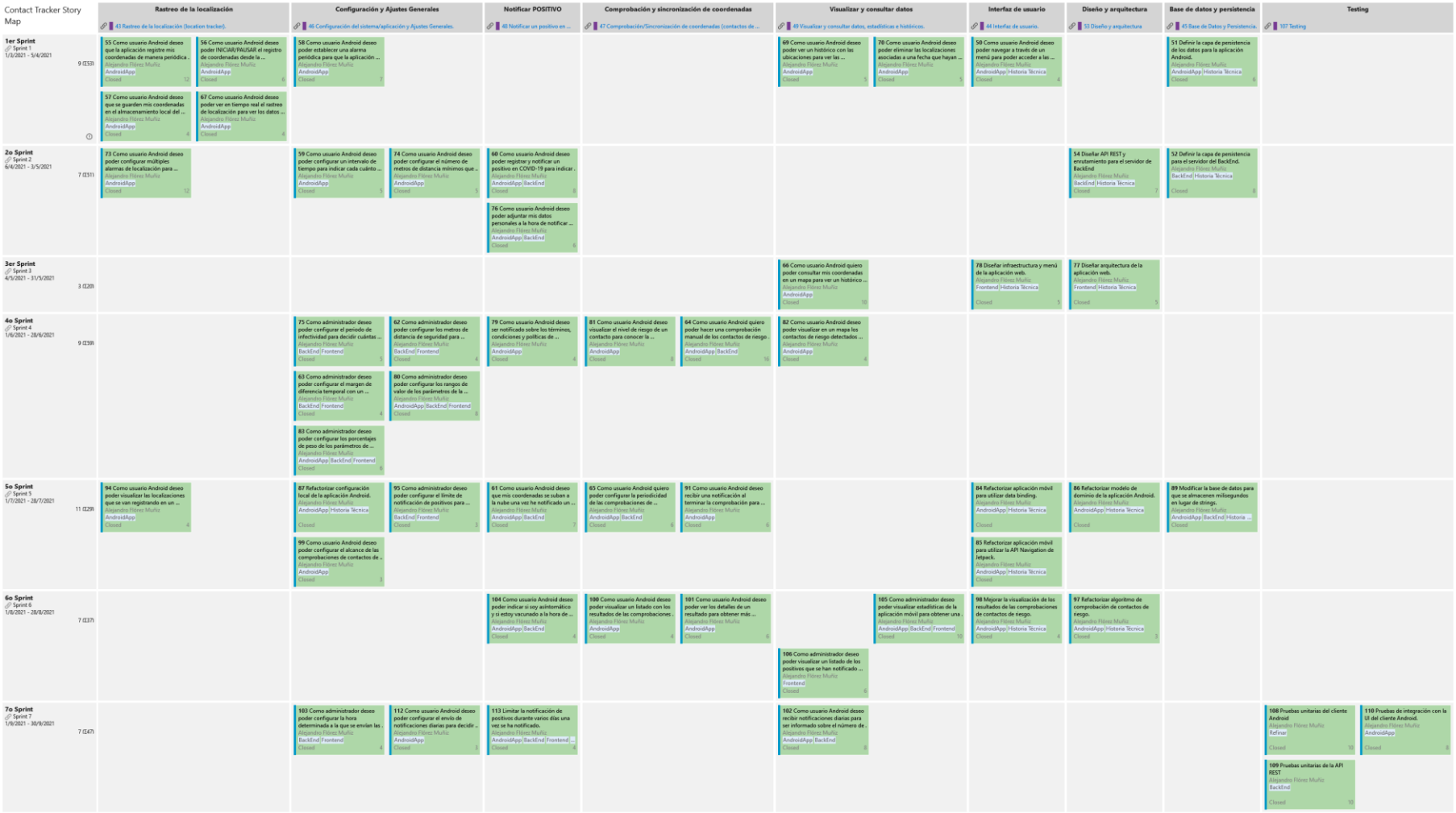
15.4.4 Partida 4: Etapa de Implantación del sistema

Item	*Descripción*	*Duración*	*Unidades*	*Precio*	*Subtotal*
1	Desplegar API REST en el servidor web de Azure.	1	horas	19,83 €	19,83 €
2	Desplegar Aplicación Web en el contenedor de Azure.	1	horas	19,83 €	19,83 €
3	Publicar APK de la aplicación Android en Google Play Store.	3	horas	19,83 €	59,49 €
4	Formar al personal sanitario para el uso del panel de control web.	10	horas	19,83 €	198,30 €
				Total	**297,45 €**

15.4.5 Partida 5: Etapa de mantenimiento del sistema

Item	*Descripción*	*Duración*	*Unidades*	*Precio*	*Subtotal*
1	Mantenimiento, ampliación y mejora del sistema de *Contact Tracker*.	12	meses	3.172,79 €	38.073,52 €
				Total	**38.073,52 €**

15.4.6 *Story mapping* del proyecto



15.4.7 Código de la función de utilidad de red *apiCall*

```
suspend fun <T> apiCall(dispatcher: CoroutineDispatcher, call: suspend () -> T): APIResult<T> {
    return withContext(dispatcher) {
        try {
            APIResult.Success(call.invoke()) // Éxito
        } catch (throwable: Throwable) {
            when (throwable) {
                is IOException -> APIResult.NetworkError // Error de RED
                is HttpException -> { // Excepción Genérica
                    // Parsear datos del error al objeto de Dominio
                    val code = throwable.code()
                    var responseError: ResponseError?
                    responseError = try {
                        Gson().fromJson(throwable.response()?.errorBody()?.charStream(), ResponseError::class.java)
                    } catch (e: JsonSyntaxException) {
                        null
                    }
                    APIResult.HttpError(code, responseError)
                }
                else -> APIResult.HttpError(null, null)
            }
        }
    }
}
```