2020

Aplicación de análisis de Actividades Deportivas

TRABAJO FIN DE GRADO

LUIS ÁNGEL RODRÍGUEZ GARCÍA

Resumen

El presente documento define los cimientos para el desarrollo de una aplicación que tiene como base temática la actividad física. En él se establecen los requisitos, el diseño, la implementación y las pruebas del sistema que permiten que sea posible el análisis, la visualización, la edición y la exportación de actividades deportivas contenidas en ficheros *xml*, con esquemas TCX o GPX, procedentes de aparatos electrónicos dotados de tecnología GPS.

Este escrito explora las diferentes opciones que existen en el mercado actual de empresas que permiten a través de sus aplicaciones el análisis y visualización de actividades deportivas. Para ello, se han analizado las cualidades y los defectos o carencias de cada una de ellas, contestando a preguntas como: ¿Qué plataformas son compatibles? ¿Con qué tipo de ficheros funcionan? ¿Permite la exportación de una actividad deportiva? ¿Existe la posibilidad de editar una ruta? ¿Qué información muestran los gráficos?

Posteriormente, se definen todos los requisitos que el sistema actual cubre y se muestra una estimación tanto económica como de tiempos de cuánto ha costado entregar el producto. Esta parte del documento se sustenta, en gran parte, por la implantación de una metodología ágil que mejora la adaptabilidad a posibles contratiempos o cambios que pueden suceder a lo largo del proyecto, entre otras cosas.

Otros apartados fundamentales son aquellos que abarcan temas como la arquitectura del sistema, las tecnologías usadas en el desarrollo del producto, el diseño planteado para solucionar cada problema, la implementación de todas las funcionalidades mediante un lenguaje de programación concreto y la batería de pruebas que cubrirá la validación del correcto funcionamiento de la aplicación y la posibilidad de realizar pruebas de regresión cuando el producto evolucione o sea modificado en el futuro.

Abstract

This document lays the foundation for the development of an application based on a physical activity as the thematic base. It defines the system's requirements, design solutions, implementation and tests allowing to create the application responsible for analysing, visualizing, modifying and exporting some activities contained in an xml file which has been provided by any GPS devices.

It explores many current market alternatives which manipulate sport activities. A study has been made to know what are the advantages and problems of all these company's applications, replying questions such as: What platforms are available in? What are the accepted file schemas? Is there the possibility to export an activity? Can users modify the route? What is the data visualized in the charts?

Afterwards, all needs covered are defined and the economic and time cost estimation to deliver the product explained. All of this is supported by the implementation of an agile methodology which makes easy the adaptability and maintainability of the project.

Other main fields of the document are those related to the technologies used in the project's development, the considered architecture, each design solution to solve some problem, the implementation of all the functionalities using a programming language and all the test cases created to validate the behaviour of the application and the possibility to run the regression tests in the future when there will be more changes or new requirements.

Tabla de contenido

Introducción

Existen multitud de aplicaciones que permiten el procesamiento y análisis de actividades deportivas tanto gratuitas como de pago. Sin embargo, no suelen reunir todas las cualidades que un deportista espera de este tipo de aplicaciones. Por lo tanto, este proyecto trata de reunir todas estas carencias y ofrecer al usuario final una experiencia increíble. Al final de este apartado, en el punto *Conclusiones*, se definen los resultados obtenidos tras haber analizado algunas de las alternativas en el punto *Alternativas de mercado*.

La aplicación que se ha de desarrollar tiene la funcionalidad de poder importar cualquier fichero *xml* con un esquema TCX o GPX, explicados más adelante en el apartado *Aspectos Teóricos*. Una vez importados estos datos, la información es convertida a un modelo común del sistema, explicado en el punto *Modelo* del apartado *Diseño*, y cuyo contenido es visualizado y puede ser editado o exportado.

La visualización de la ruta es llevada a cabo, por un lado, en una tabla que muestra los valores de cada segmento de la actividad deportiva y, por otro, a través de un mapa geográfico que permite visualizar dónde se encontraba el usuario en cada momento de la trayectoria. Los datos también son visualizados mediante diferentes tipos de gráficos, como se comentará más adelante, que permiten visualizar en cada momento ciertos valores cruciales de una actividad física como la altitud, la velocidad o las pulsaciones por minuto. Además, para dar valor añadido a la aplicación, la visualización entre el mapa y los gráficos está sincronizada, así pues, se consigue que el usuario sepa en todo momento los valores que obtuvo en un punto en concreto que ha sido seleccionado en alguno de los anteriores elementos.

La edición es posible desde el propio mapa o en la tabla donde se visualizan los segmentos de la ruta, descrito todo esto en el anterior párrafo. Desde el mapa se puede separar la ruta o un determinado segmento en dos trozos o eliminar un punto de la trayectoria. Desde la tabla se pueden borrar segmentos enteros o unir dos contiguos.

Por último, se permite exportar la ruta deportiva modificada, que está almacenada en el sistema, a alguno de los formatos comentados al inicio de este mismo aparatado llamado *Introducción* y que serán descritos más adelante. También es posible obtener la versión original del documento, es de gran ayuda para que el usuario pueda recuperar la actividad original si las modificaciones no satisfacen sus necesidades.

Alternativas de mercado

A continuación, se describen diferentes empresas que se dedican a este sector. Explicando de forma clara y concisa cuales son los puntos fuertes y las desventajas de cada una de ellas.

Adidas Runtastic

Una de ellas es Runtastic[1] de origen austriaco, fue concebida inicialmente con la idea de monitorizar las competiciones de barco de vela, no obstante, como estaba enfocada para un grupo muy reducido de usuarios los creadores decidieron abrir las funcionalidades a

[1] https://es.wikipedia.org/wiki/Runtastic

más deportes. En agosto de 2015, Adidas compra la empresa por 239 millones de dólares según indica Wikipedia. Después de cuatro años de la adquisición de la compañía, en septiembre de 2019 Adidas decide cambiar el nombre de la aplicación a *Adidas Running by Runtastic*[2].

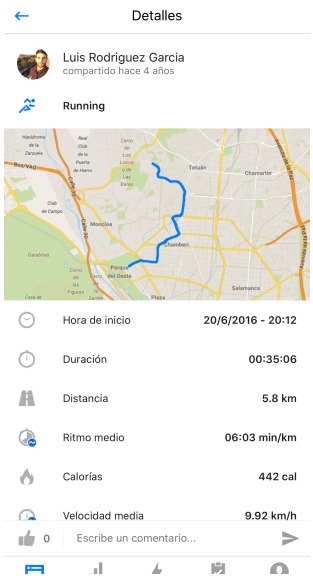


Imagen 1

En el pasado la aplicación disponía de versión web (en la Imagen 2 se muestra una captura de la aplicación web del año 2018), sin embargo, según indica la propia compañía en este enlace[3], a fecha de 2020 han decidido mantener solo las aplicaciones para dispositivos móvil en iOS y Android. Probablemente, esta sea la tendencia de un mercado en el que cada vez se hace un mayor uso de este tipo de aplicaciones.

La empresa sigue un modelo de negocio *freemium*[4] desde sus inicios, ofreciendo los servicios básicos sin tener que hacer ningún gasto monetario. Para tener funcionalidades de mayor calidad, se ha de pagar una cuota que puede ser anual, semestral o bien, trimestral (en abril de 2020 los precios son: 49,90€, 35,90€ y 19,90€, respectivamente). La suscripción te da derecho a los siguientes servicios:

- Tener un plan de entrenamiento personalizado, con distintos fines enfocados a aquellas personas que empiezan a correr, que quieren perder peso o que lo que buscan es preparar carreras.
- Disponer de un plan nutricional que mejore tus hábitos alimenticios.
- Descargar y guardar rutas de entrenamiento de otros usuarios de la aplicación.

También usa técnicas de *gamification*[5] para fomentar el uso de la aplicación empleando mecánicas de juego que fomenten la motivación de los usuarios. Gracias a la red social que integra, es posible añadir los datos de las carreras deportivas a una clasificación con el grupo de amigos del usuario para poder ver quien tiene mejores resultados, retando a los usuarios a superarse cada día.

En ambos planes, tanto el de pago como el gratuito, la aplicación permite la visualización de la ruta en el mapa terrestre y muestra ciertos valores que resumen cómo ha ido la actividad en el transcurso del tiempo. Permitiéndole al usuario analizar las estadísticas de un entrenamiento, compartir sus resultados con la comunidad e ir alcanzando las metas que se haya ido fijando.

Antes de que la versión que se representa en la Imagen 2 fuese eliminada, si era posible exportar las rutas en otros los formatos *gpx*, *tcx* y *kml* para su posterior uso, sin embargo, esta funcionalidad dejó de estar disponible tras haberse eliminado la aplicación web. En la nueva versión es posible solicitar la exportación de toda la información de un usuario

[2] https://help.runtastic.com/hc/en-us/articles/360008944520-WHAT-EXACTLY-CHANGES-IN-THE-APPS-
[3] https://www.runtastic.com/es/web-update
[4] https://es.wikipedia.org/wiki/Freemium
[5] https://en.wikipedia.org/wiki/Gamification

que tienen los sistemas almacenados en formato *json*[6], no obstante, esta acción tarda uno o dos días en llevarse a cabo.

No obstante, la posibilidad de importar actividades deportivas se ha mantenido en la versión web reducida que existe actualmente, en el apartado "*Importar actividad*" es posible subir actividades creadas por aplicaciones de terceros mediante la subida ficheros en formatos *gpx* o *tcx*.

Tras analizar todas las características de la aplicación, se han identificado las siguientes carencias que hacen que el usuario esté limitado a la hora de utilizar la aplicación:

- No se permite modificar la ruta
- No se puede dividir la actividad deportiva en los segmentos que más le interese al usuario de la aplicación para poder concentrarse en ciertos intervalos de tiempo específicos.
- No se permite exportar la actividad física

Como se mencionó al principio, la aplicación solo tiene versiones para aplicaciones para móviles. En la Imagen 1 se puede ver el menú de la aplicación para iOS, en este ejemplo se visualiza la información relativa a una carrera: la hora de inicio de la actividad física, la duración, la distancia, el ritmo medio, el número de calorías gastadas, la velocidad media, entre otros.

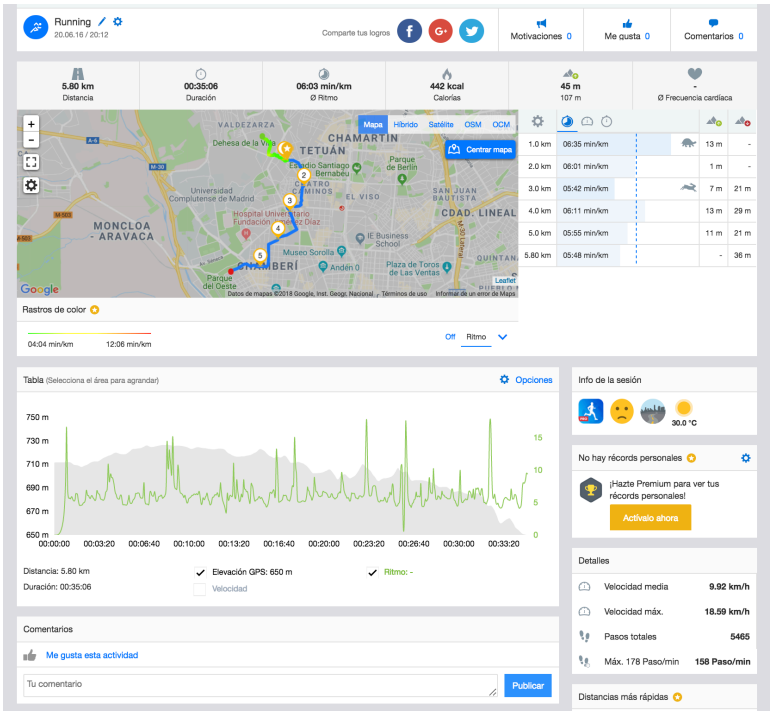


Imagen 2

[6] https://www.json.org/json-en.html

Strava

La empresa *Strava*[7] es otra de las alternativas existentes en el mercado. Fundada en San Francisco en el 2009, al igual que la empresa anterior, también sigue un modelo de negocio *freemium*. Existen la posibilidad de pagar una cuota anual o mensual (en abril de 2020, los precios eran de 59,99€ y 7,99€, respectivamente) que da derecho a disfrutar de las funcionalidades incluidas en el paquete que la empresa denomina Summit, concentradas en tres grandes bloques:

- Entrenamiento:
  Te permite definir unos objetivos personalizados, conseguir planes de entrenamiento, acceder a análisis exhaustos de tus carreras y permite visualizar la tabla de clasificación filtrados los datos por edad o peso.
- Seguridad:
  Disfrutar de la funcionalidad que se llama "*beacon*" que permite enviar la posición en tiempo real y de forma privada en tus carreras a familiares o amigos y la posibilidad de ver mapas donde el usuario ha hecho deporte.
- Análisis:
  Visualizar el esfuerzo relativo con el que se ha realizado la actividad física. Controlando la tendencia en cada actividad física, permitiendo entrenar de forma más constante, analizar la potencia que se ha tenido en una carrera y visualiza el ritmo y los datos por vuelta.

Sin embargo, a diferencia de Adidas, esta compañía ha tratado de diferenciarse desde un principio intentando enfocarse hacia un mercado más profesional. Como se podía ver en la página web en 2018, la cuenta de pago llamada por aquel entonces Premium estaba publicitada con los siguientes eslóganes:

«*A los atletas Premium les apasiona su deporte. No necesitas ser el mejor, solo necesitas la voluntad de serlo. La suscripción Premium es para el atleta que sabe exprimir al máximo su deporte favorito*»

«*La suscripción Premium es nuestra solución definitiva para garantizar la seguridad de los atletas, mejorar su rendimiento y hacer que disfruten al máximo de cada actividad*»

Por lo tanto, esta compañía quiere que su mercado esté orientado hacia el mundo del deporte de alto rendimiento, para aquellos deportistas que buscan mejorar y tener un seguimiento con todo detalle de su actividad física. Así pues, mientras que Strava sigue estas estrategias de marketing, Runtastic enfoca su versión Premium al público en general.

En la Imagen 3 se muestra cómo es la interfaz web de esta aplicación. En este caso, la versión web de esta aplicación sí se sigue manteniendo y sigue disponible a fecha junio de 2020. Se puede visualizar más información acerca de la actividad deportiva que en la versión web que existía de Runtastic en el pasado. En este caso, se da la posibilidad de seleccionar la altitud (seleccionada por defecto), el ritmo, el RAP[8], el ritmo cardíaco y la

[7] https://es.wikipedia.org/wiki/Strava

[8] Ritmo Ajustado en Pendiente (permite comparar fácilmente carreras llanas con carreras con desnivel): https://support.strava.com/hc/es-mx/articles/216917067-Ritmo-ajustado-en-pendientes-RAP-

cadencia. Mientras que en la Imagen 2 tan solo se puede visualizar la altitud, la velocidad y el ritmo. Además, también permite poder exportar cualquier tipo de actividad a formato *gpx* o en el formato original con el que se subió.

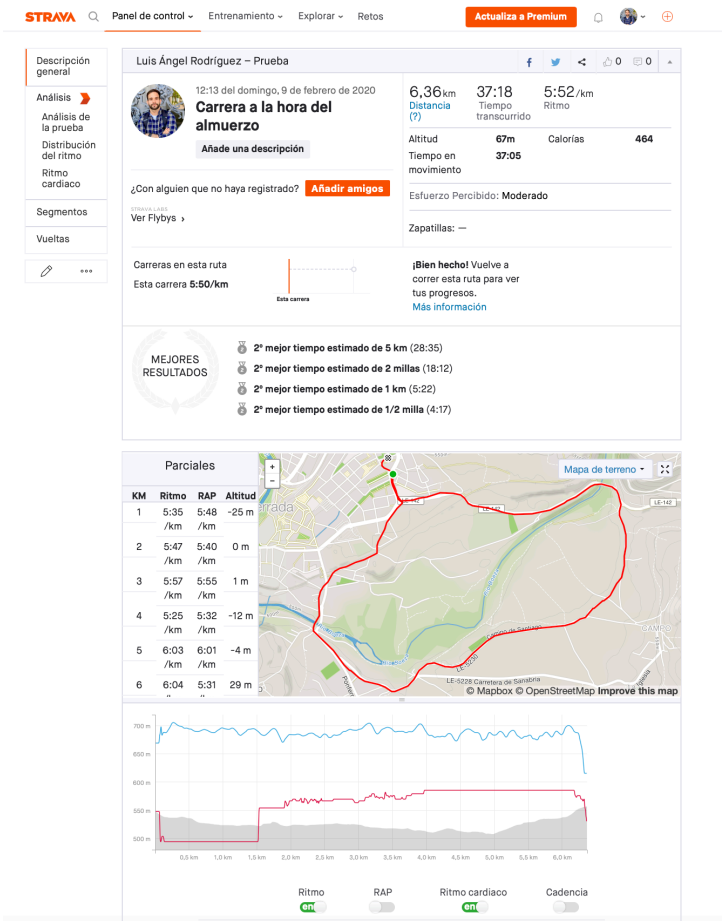


Imagen 3

Esta empresa también dispone de versión para aplicaciones móviles, en la Imagen 4 se puede visualizar como es la aplicación para el sistema operativo iOS. Hay que añadir que en ambas capturas de pantalla, se muestra la misma actividad deportiva, es decir, la información de la Imagen 3 es la misma que la de la Imagen 4. Además, hay que aclarar que ambas aplicaciones tienen un diseño de interfaz muy similar. A continuación, se va a explicar qué datos se muestran:

- En ambas aplicaciones se divide la actividad deportiva en segmentos de un kilometro de longitud. No existe la posibilidad de modificar la longitud de estos tramos. En Runtastic se proporciona la elevacion ganada, la elevación perdida, el ritmo y la frecuencia cardíaca de media en cada tramo, mientras que en Strava se proporciona el desnivel acumulado (un valor negativo, positivo o cero).
- Las calorias que se han consumido a lo largo de la actividad física son mostradas en ambas aplicaciones.

- En Runtastic se muestra un valor orientativo de la deshidratación generada tras finalizar la carrera. En Strava no hay ningún campo que dé información sobre este aspecto.
- El análisis cardíaco solo está disponible en la versión de pago de Strava, mientras que Runtastic proporciona una herramienta sin necesidad de tener la versión premium. La Imagen 5, que representa la aplicación móvil de Runtastic, muestran dos gráficos que representan de una manera muy buena cómo han ido las pulsaciones durante una carrera deportiva, viendo como en esta compañía sí es posible obtener el analisis de las pulsaciones de la carrera. En función de la edad y el peso, la aplicación define unos intervalos que relacionan pulsaciones por minuto con tipos de intensidad de entrenamiento, para este ejemplo:
    - -130 ppm: fuera de zona de ejercicio
    - 131-141 ppm: entreno fácil
    - 142-152 ppm: entreno quema de grasa
    - 153-163 ppm: entreno aeróbico
    - 164-174 ppm: entreno anaeróbico
    - 175+ ppm: entreno en línea roja

  Por lo tanto, con esta información se puede saber qué nivel de intensidad se consiguió en un determinado entrenamiento, cosa que no es posible saber con Strava.

Imagen 4

Tras el anuncio del cierre de la versión web de Adidas Runtastic, esta empresa ya no permite la creación de rutas deportivas ya que esta funcionalidad no esta activa en las aplicaciones móviles. No obstante, en 2018 sí era posible crear rutas usando la herramienta *online* que proporcionaba la compañía a través de la web mostrada en la Imagen 2.

A diferencia de Adidas, Strava sí permite la creación de carreras en la versión web de la aplicación. En la Imagen 6 se visualiza la herramienta mediante la cual se puede crear una ruta de manera muy intuitiva y sencilla. Según se van introduciendo diferentes posiciones en el mapa, la herramienta calcula el recorrido óptimo entre estos dos puntos y así se va generando el resto del recorrido. Para realizar los cáculos oportunos, se pueden seleccionar el tipo de trayectoria para ciclismo o para carrera. La principal diferencia es que el primero contará como válidas las calzadas para automóviles o vías habilitadas para bicicletas mientras que para las carreras solo se tendrán en cuenta los caminos o senderos reservados para que las personas puedan andar o correr.

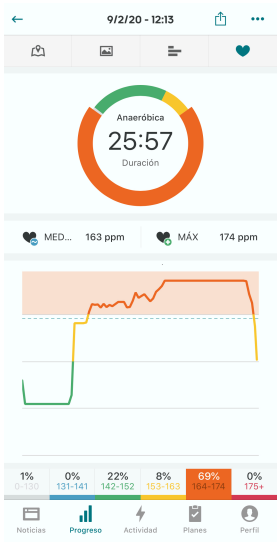


Imagen 5

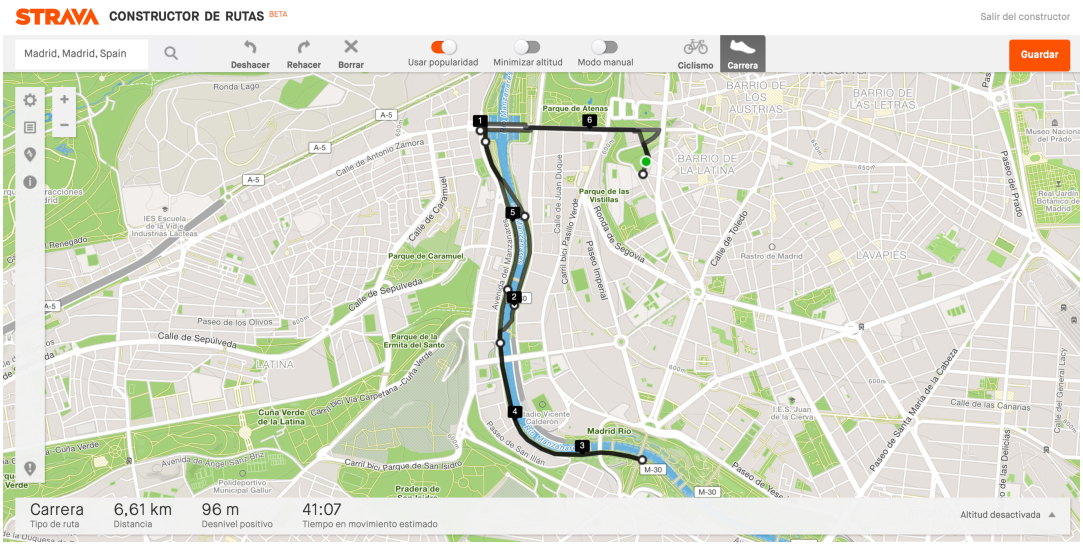


Imagen 6

Por otro lado, la opción de popularidad tiene en cuenta rutas de otros usuarios de Strava para trazar el camino entre dos puntos diferentes. También es posible que este trayecto sea calculado minimizando la altura si se activa esta casilla y, ya por último, existe la posibilidad de seleccionar caminos en línea recta si se activa la casilla con la opción modo manual.

Apple

Imagen 7

Por otro lado, en la WWDC[9] de 2014 Apple lanzó al mercado Salud[10], en inglés Health. Una aplicación que venía integrada con la versión 8 de iOS y se presentaba al público con el fin de tener recopilados todos los datos de salud y bienestar del usuario. Proporcionando un único punto donde poder registrar esta información y ver cómo evoluciona a lo largo del paso del tiempo. Otra de las grandes características de esta aplicación, desde sus origines, es la posibilidad de incluir los datos médicos del dueño del iPhone, siendo muy importante esta información para posibles emergencias que puedan ocurrir. Por otro lado, con esta nueva aplicación se introdujo la posibilidad de cuantificar los pasos y la distancia de la persona que llevaba el iPhone sin que el usuario tenga que llevar ningún otro dispositivo electrónico.

El siguiente paso de Apple fue presentar en 2015 el Apple Watch junto a su sistema operativo WatchOS. Este reloj revolucionó el mercado de wearables, gracias a que disponía de capacidades para la monitorización de la actividad física, funcionalidades orientadas a la salud y la integración total con el sistema operativo iOS y con otros sistemas operativos de la compañía. A diferencia de las otras dos empresas anteriormente mencionadas, esta

[9] https://es.wikipedia.org/wiki/Worldwide_Developers_Conference
[10] Health app: https://www.apple.com/es/ios/health/

compañía no proporciona ninguna aplicación web con la que acceder a todos estos datos, ni tampoco una aplicación que permita importar información procedente de otros sistemas.

Junto a estos dos hitos que han marcado la historia reciente de la empresa, Apple saco la aplicación Actividad[11], denominada en inglés Activity, que ha permitido guardar todos los datos relacionados con el seguimiento de los movimientos del usuario a lo largo del día y motivarlo para conseguir todos los objetivos definidos. La aplicación guarda la siguiente información del usuario relativa a un día:

- Cuanto tiempo está de pie
- Cuantas calorias ha gastado
- Cuantos tiempo de ejercicio has hecho

En la Imagen 7 se puede visualizar el diseño de la interfaz de usuario de esta aplicación. En ella se visualizan tres circulos, si el circulo está completo significa que los objetivos fijados se han logrado para ese día. El circulo rojo corresponde a la cantidad de calorías, el circulo verde corresponde al tiempo destinado a hacer ejercicio físicos y el circulo azul es el tiempo que el usuario ha estado de pie. Los datos de esta captura, según se indica en la parte superior de la captura, corresponden al viernes 24 de enero del 2020.

Además de este apartado, también es posible visualizar:

- Las tendencias: este apartado compara los últimos 90 días de actividad del usuario con los datos almacenados para el último año. Por lo tanto, mostrara indicadores (representados gráficamente con una flecha) donde se ha de hacer más incapié para mejorar los resultados globales.
- Los entrenamientos almacenados se han sincronizado con la aplicación Workout [12], llamada en español Entreno, del Apple Watch o con aplicaciones de terceros (por ejemplo, las que se han mencionado anteriormente como Strava o Adidas Runtastic) haciendo uso del SDK[13] identificado como HealthKit[14] que Apple hizo público cuando presentó la aplicación Salud. Además de permitir subir entrenamientos a la aplicación de Activity, a su vez, estos datos también son recopilados en la aplicación de Salud simplemente dando permisos a las aplicaciones de terceros para que puedan escribir en los datos de la aplicación Salud. En la Imagen 8 se puede ver como se muestra la información de un entrenamiento, además, se aprecia que el diseño esta totalmente integrado con el *look and feel*[15] del sistema operativo iOS. Los datos que se visualizan son muy similares a los que se visualizaban en las anteriores aplicaciones: la duración de la carrera, la distancia recorrida, las

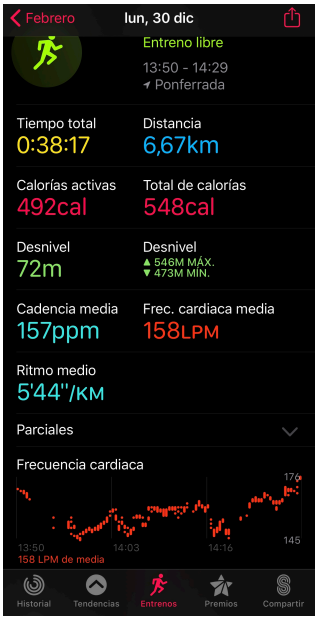


Imagen 8

[11] https://support.apple.com/en-ie/guide/watch/apd3bf6d85a6/watchos
[12] https://support.apple.com/en-us/HT204523
[13] https://en.wikipedia.org/wiki/Software_development_kit
[14] https://developer.apple.com/healthkit/
[15] https://en.wikipedia.org/wiki/Look_and_feel

calorías actividas[16], el total de calorias, el desnivel, desnivel máximo y mínimo, la cadencia media (en pasos por minuto), la frecuencia cardíaca media y el ritmo medio (en minutos y segundos por kilometro, respectivamente). En la Imagen 9 se puede ver en más detalle la parte de debajo de la Imagen 8. En esta ultima captura se visualiza un gráfico el cual las anteriores aplicaciones analizadaque no mostraban, este diagrama muestra la recuperación de la frecuencia cardiaca a lo largo del tiempo una vez que la actividad física ha finalizado. Por otro lado, también se puede acceder a la ruta visualizada en el mapa geográfico o ver con qué condiciones climatológicas el usuario realizó el entrenamiento para el que se están mostrando datos.

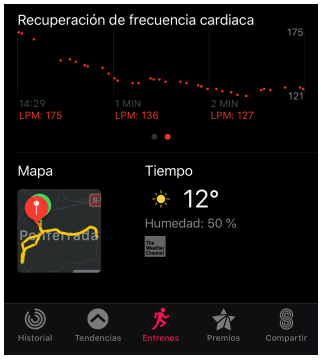


Imagen 9

- Los premios que se han conseguido, es decir, aquí Apple usa la técnica de gamificación para motivar a los clientes a que superen retos.
- Las estadísticas para ese día de los amigos que el usuario haya agregado. Por lo tanto, verá la información de esos usuarios, al igual que, estos usuarios también tendrán acceso a su información. Los datos mostrados seran los porcentajes de los circulos comentados más arriba que representan los objetivos para cada usuario en ese día.

Garmin

Es otra de las empresas punteras en el sector de la monitorización de actividades físicas, creada en Kansas en el 1989, la empresa no solo está presente en el mundo deportivo sino que también está especializada en sectores como la automoción, la aviación y la marina. Al igual que Apple y, a diferencia de las dos primeras que se han descrito, Garmin[17] diseña y desarrolla dispositivos tecnólogicos para permitir la monitorización de los datos relativos a una carrera o un ejercicio físico. Por lo tanto, además de competir con empresas de software como Strava o *Runtastic*, también compite en hardware con Apple, entre otras.

Al igual que sucede con *Strava*, esta empresa permite aceder a todos los datos a través tanto de sitios web como de aplicaciones móviles. Sin embargo, la primera permite importar otras actividades grabadas con otros dispositivos que no sean los propios de la marca.

Se empezará a describir la aplicación web que esta diseñada con una interfaz de usuario tipo *dashboard*[18]. En la Imagen 10 se muestra como es este menú inicial que, basicamente, contine un layout[19] con tres elementos diferentes. Por un lado, la barra de navegación a la izquierda, en el centro los elementos visuales y en la pate superior derecha se visualiza la información relativa al usuario que ha accedido al sistema, entre dichas funcionalidades se encuentra la posibilidad de importar una actividad, de visualizar las

[16] Las calorías activas, a diferencia de las calorías en reposo, son las que quemas al permanecer de pie o cuando te mueves.
[17] https://en.wikipedia.org/wiki/Garmin
[18] https://en.wikipedia.org/wiki/Dashboard_(business)
[19] https://en.wikipedia.org/wiki/Page_layout

notificaciones o solicitudes, ver los dispositivos de la compañía asociados a la cuenta y poder acceder a los datos de la cuenta.

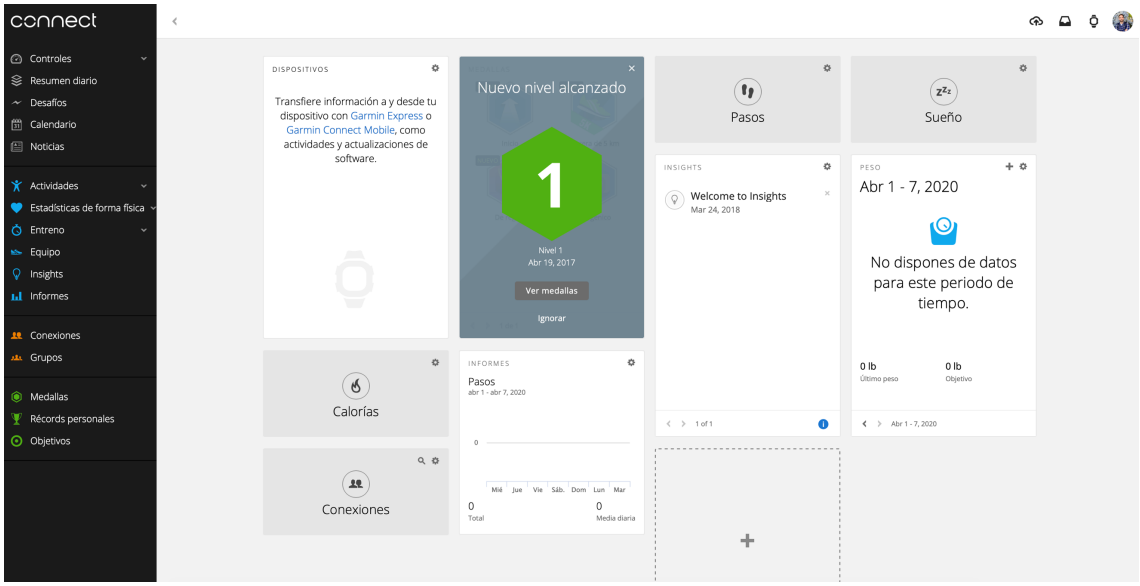


Imagen 10

Si nos centramos en la página principal, se puede ver que esta formada por diferentes widgets[20] que proporcionan información al usuario. En el ejemplo, muchas de estos elementos se encuentran minimizados porque no hay información a mostrar. Sin embargo, cabe destar alguno de ellos:

- El primero de la parte superior izquierda permite visualizar los dispositivos Garmin asociados. En este caso no existe ninguno registrado.
- El segundo proporciona el número de medallas obtenidas por las actividades que el usuario ha realizado.
- El tercero hace referencia al número de pasos realizados.
- La información sobre el sueño la cual ha sido registrada con algún dispositivo electrónico.
- Las calorias que se ingieren gracias a la colaboración con la aplicación MyFitnessPal, propiedad de la empresa Under Armour, y que da la posiblidad al usuario de registrar qué es lo que come y cuánta cantidad.
- Descubrir amigos dentro del ecosistema Garmin a través de la conexión con redes sociales como Facebook o de los contactos de alguna cuenta de Google.
- La información del peso del usuario.
- Visualizar el análisis y los resultados de la aplicación tras el analisis de las carreras o ejercicos subidos a la aplicación.

Por otro lado, el menú de la derecha da la opción de poder acceder al menú principal descrito en el párrafo anterior, ver todos los entrenamientos que se han realizado, el resumen diario (los datos resultantes de la monitorización con algun dispositivo de la marca), visualizar y crear retos o desafios, el calendario, todas las actividades realizadas,

[20] https://en.wikipedia.org/wiki/Graphical_widget

estadisticas de la forma física o del estado de bienestar (sueño, peso, hidratación, calorías, frecuencia cardíaca y estrés), entreno, el equipo con el que se realizan las actividades deportivas, informes, conexiones, grupos, medallas, records y objectivos.

En la Imagen 11 se visualiza una de las partes desde donde se pueden subir actividades físicas, en este caso, de forma múltiple. Además, el sistema detecta los posibles ficheros duplicados que se estan tratando de subir, mostrando un mensaje como el que aparece en la captura de pantalla en rojo.

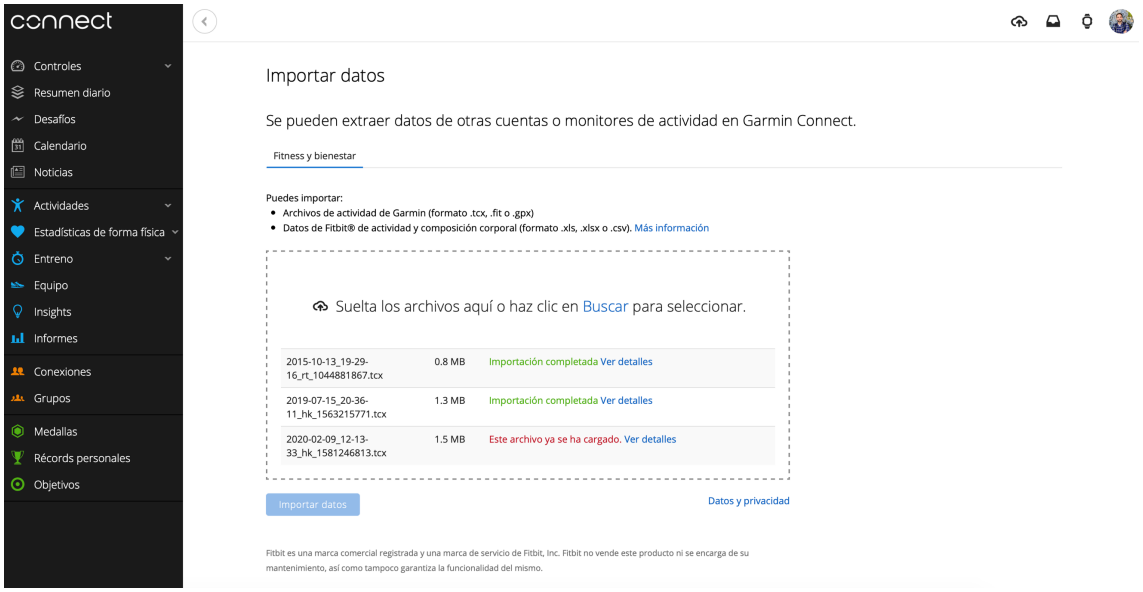


Imagen 11

Para poder acceder a esta vista, tan solo hay que hacer click sobre el icono de la parte superior derecha con forma de nube. Por otro lado, aparece de forma informativa una lista con los formatos *tcx*, *fit*, *gpx* y un formato especial de la empresa FitBit con la extensión de ficheros Excel o csv. Todos estos tipos de archivos siguen un esquema compatible con la aplicación Garmin.

A continuación, vamos a ver como analiza esta compañía una actividad física, tanto en la versión web como en la para dispositivos móviles, en particular para dispositivos iOS. Como se ha comentado anteriormente, la carrera se muestra en el panel principal, a través de la Imagen 12 se aprecia que la descripción de la actividad física se representa en la parte superior izquierda y la trayectoria con el mapa en el centro de la pantalla. Además, se puede reproducir un video que muestra la ubicación del usuario en cada instante de la carrera.

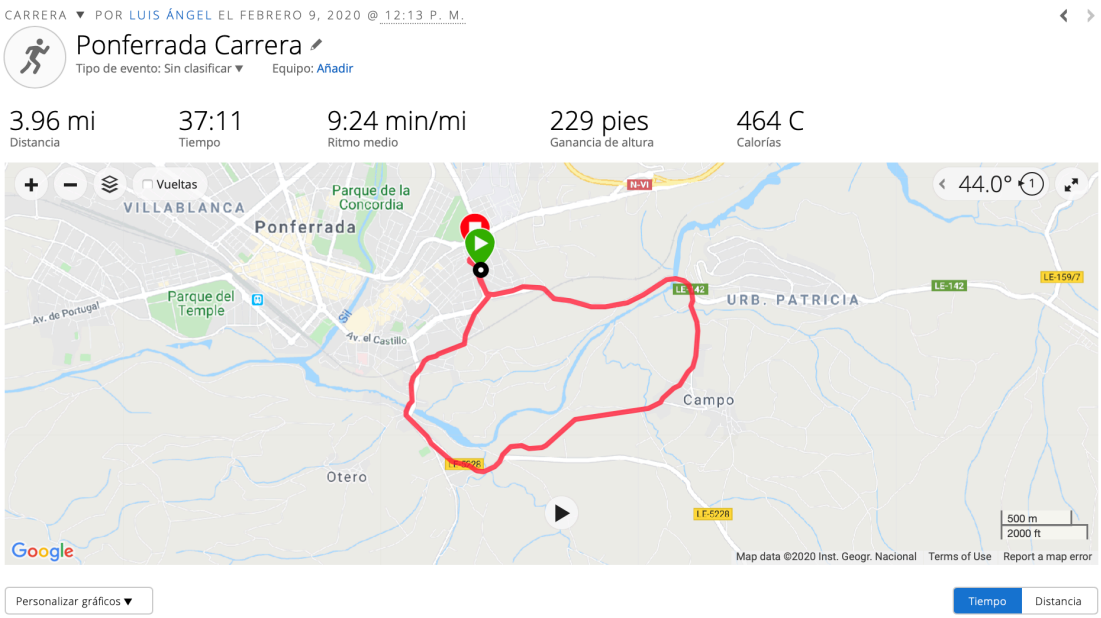


Imagen 12

La Imagen 13 corresponde otra parte de esta actividad mostrada en el panel principal de la aplicación web, justo a la derecha del mapa que se ha mostrado en la Imagen 12. Aquí se da la posibilidad de añadir fotografías que el usuario haya tomado de la carrera, a lo largo de toda la trayectoria. También se muestran las medalles que se han logrado tras finalizar esta carrera. En este caso, el usuario logró la medalla de su primera carrera, la que se identifica con el nombre "Inicio", y también consiguió la medalle de haber superado una carrera de 5 kilómetros. Más abajo de estos dos apartados, existe la posiblidad de añadir comentarios sobre cómo ha ido la carrera, como el deportista se ha sentido al acabar la actividad o cualquier otra cosa que se le ocurra al usuario con respecto a la ruta deportiva. Por último, en esta sección aparece el dispositivo con el que fue creada esta ruta. En este caso, como la ruta fue importada por el método que se comentó más arriba, no aparecerá ningún dispositivo electronico que realizó el registro.

Imagen 13

Debajo del mapa visualizado en la Imagen 12, se sitúa la Imagen 14 que contiene una serie de gráficas que muestran diferentes datos de la carrera. Se pueden ver valores que hacen referencia a la altura, el ritmo o la frecuencia cardíaca. Lo que tiene de diferente con respecto a las anteriores compañías es que muestra la cadencia de la carrera en un gráfico, además ha recopilado información de una gran cantidad de usuarios para asociar unos percentiles a los valores de cadencia, identificándo a cada uno de estos grupos con un color diferente. Por lo tanto, en el ejemplo de la Imagen 14, el color rojo corresponde al percentil menor de 5, el verde entre el percentil 30-69 y el morado al percentil mayor de 95.

Imagen 14

Más abajo de estos gráficos, como se muestra en la Imagen 15, aparece una tabla con todos los datos estadisticos relacionados con la actividad, las vueltas, y los valores de cuánto tiempo estuvo el atleta porcentualmente hablando en cada frecuencia cardíaca.

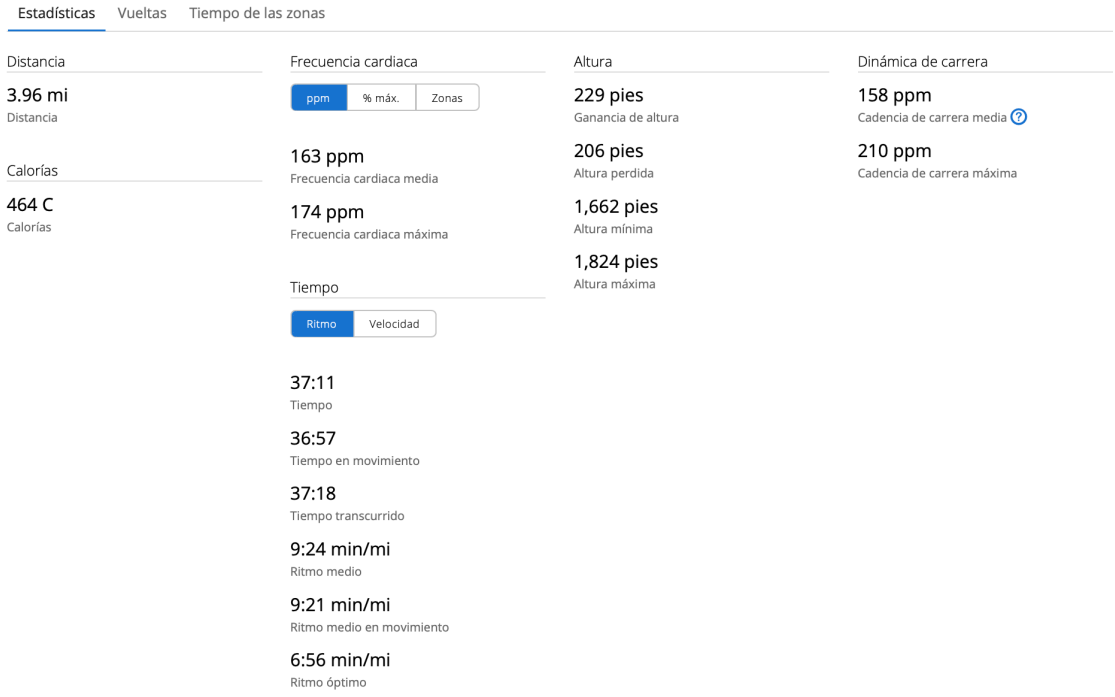


Imagen 15

En la aplicación móvil para iOS, visualizada en la Imagen 16, se visualizan los mismos elementos que en el sitio web pero con distinto orden y un *look & feel* diferente ya que en esta ocasión el *layout* es para una aplicación móvil donde se puede representa toda la información comentada anteriormente.

Por un lado, lo primero que se ve al acceder a la aplicación es un menù principal que contiene los datos que resumen de forma general cómo ha ido la actividad física, el mapa de la carrera y las medallas obtenidas. Existen otras tres pestañas más. La segunda permite visualizar una tabla con los datos sobre el ritmo, la velocidad, el tiempo, la frecuencia cardiaca, la dinámica de la carrera, la altura y las calorías. El tercer apartado contiene otra tabla con cada uno de los segmentos de la actividad y los valores para cada uno de ellos sobre la duración, la distancia y el ritmo medio para cada vuelta. Por último, se muestran las gráficas que se comentaron en la aplicación web añadiendo la que muestra cuánto tiempo estuvo en cada frecuencia cardiaca.

Imagen 16

Otros

Además de estas herramientas, también existen otras en el mercado como por ejemplo Sunto, Nike+, Fitbit, RunKeeper, Endomondo, Polar, Zwift, TrainingPeaks, TomTom MySports, Smashrun, 2Peak, RunAnalyze, Wikiloc, RideWithGps, MyFitnessPal, entre otros.

Exportación de actividades

Una de las cosas dificiles de realizar a la hora de manipular actividades deportivas es la exportación de datos de forma fácil para que puedan ser utilizados por otras aplicaciones. En dispositivos iOS se pueden dar situaciones en las que los datos no se han sincronizado con la aplicación Salud por lo que la información de terceras partes no se puede importar en esta aplicación nativa. Por lo tanto, para solventar este problema se ha indagado sobre las diferentes soluciones existentes en el mercado, basicamente, existen dos aplicaciones

que intentan proporcionar esta funcionalidad, una de ellas es RunGap [21] y la otra HealthFit[22]. El único inconveniente es que las dos aplicaciones requieren del pago de 8,99€ al año o 3,49€ en un único pago, respectivamente, para que puedan ser utilizadas. Siguiendo las conclusiones de este enlace [23], se puede decir que RunGap tiene funcionalidades más completas porque permite:

- Sincronizarse con una multitud de aplicaciones de registro de actividades físicas.
- Exportar la ruta a ficheros con extensión: *fit*[24], *tcx*, *gpx* y una extensión propia de la aplicación.
- Editar la actividad física: título, notas, tipo de actividad, distancia recorrida, duración y calorías en kcal.
- Compartir con la red social Facebook los resultados.
- Adjuntar fotos.

Una vez analizado lo anterior, se puede concluir que HealthFit solo puede exportar actividades a ficheros *gpx*, por ello, como la aplicación que se va a desarrollar trabaja con ficheros tanto con ficheros *gpx* como con archivos con una extensión *tcx*, está claro que la que interesa más es RunGap al proporcionar una mayor funcionalidad. Tras su uso, se puede decir que esta última aplicación no transfiere con total exactitud todos los datos almacenados en cada una de las aplicaciones de las compañías anteriormente mencionadas al formato que se le solicita.

Conclusiones

Con todo este análisis y teniendo en cuenta toda la información definida en este apartado, se va a desarrollar una aplicación que cubra alguna de las carencias que se han identificado en los párrafos anteriores. Por lo tanto, la aplicación ha desarrollar permitirá las siguientes funcionalidades:

- El cliente accedera a la aplicación a través de un navegador web. De esta forma, se podrá acceder desde un ordenador o desde un dispositivo móvil, indistintamente.
- El deportista podrá importar cualquier tipo de actividad deportiva contenida en alguno de los esquemas de los que se hablará más adelante.
- El usuario podrá realizar ciertas operaciones como separar la ruta en segmentos, seleccionando el punto en el que se dividirá el tramo que lo contien, borrar los posibles puntos que se hayan incluido por error, juntar dos vueltas contiguas o eliminar algún segmento.
- El cliente podrá exportar actividades deportivas a los formatos más comunes.
- El usuario tendrá la opción de recuperar el fichero que contine la actividad original que se subió.
- El cliente de la aplicación podrá ver los valores que resumen como ha ido la actividad física mediante el uso de gráficas.

[21] https://www.rungap.com
[22] https://twitter.com/HealthToFit
[23] https://theapplewatchtriathlete.com/blog-1/2017/10/6/exporting-your-apple-workout-data-to-fit-files-with-the-healthfit-app
[24] FITness file format: https://fileinfo.com/extension/fit

Planificación

Especificación de Requisitos

Tras haber analizado las diferentes opciones que hay en el mercado y haber definido cuáles son las necesidades de negocio en el apartado de *Introducción*, se procederá a enumerar todos los requisitos que el sistema ha de cumplir.

Para la obtención y definición de los requisitos se pueden:

- Realizar entrevistas con los *stakeholders*[25], si así lo permite el marco de trabajo. En este caso, la parte interesada en desarrollar la aplicación somos nosotros mismos.
- Mediante encuestas, definiendo una serie de preguntas que serán formuladas a los usuarios finales de la aplicación. Así pues, seleccionando una audiencia bien establecida se podrían obtener conclusiones de cuales son los puntos más fuertes o los que más valora el cliente final del sistema.

Lo requisitos pueden ser divididos en dos grupos. Por un lado, los funcionales que son aquellos que se centran en las necesidades que el software debe satisfacer y, por otro, los no funcionales que representan aquellas propiedades que el producto deberá cumplir pudiéndose dividir en tres subgrupos: de rendimientos del sistema, interfaces y procesos de desarrollo. A continuación, se listan cada requisito agrupado por cada uno de estos grupos:

- Requisitos funcionales:
    1. **RQ-FN-01**: El sistema en una primera versión no proporcionará ningún mecanismo para que los usuarios se autentiquen.
    2. **RQ-FN-02**: El sistema leerá actividades físicas a través de un fichero xml.
    3. **RQ-FN-03**: El sistema podrá leer una serie de esquemas xml.
        1. **RQ-FN-03-01**: Ficheros con un esquema gpx.
        2. **RQ-FN-03-02**: Ficheros con un esquema tcx.
    4. **RQ-FN-04**: El sistema procesará la información que hay en el fichero y lo transformará a un modelo común que entienda.
    5. **RQ-FN-05**: El sistema persistirá todas las actividades deportivas contenidas en el fichero de entrada que sean correctas.
    6. **RQ-FN-06**: El sistema almacenará en un repositorio el fichero xml original.
    7. **RQ-FN-07**: El sistema visualizará la ruta deportiva en un mapa que permita ver cual fue el rumbo recorrido por el usuario de la aplicación.
    8. **RQ-FN-08**: El sistema permitirá eliminar un determinado punto seleccionándolo desde el mapa en el que se ha mostrado la actividad deportiva.
    9. **RQ-FN-09**: El sistema tendrá la funcionalidad de dividir la ruta deportiva desde el mapa seleccionando el punto especifico.
    10. **RQ-FN-10**: El sistema mostrará la altitud a lo largo de todo el recorrido en una gráfica de áreas.

[25] https://en.wikipedia.org/wiki/Stakeholder_(corporate)

11. **RQ-FN-11**: El sistema mostrará la frecuencia cardiaca a lo largo de todo el recorrido en una gráfica de barras.
    1. **RQ-FN-11-01**: El sistema también mostrará una línea con la frecuencia media llevada en cada vuelta o segmento.
12. **RQ-FN-12**: El sistema mostrará la velocidad a lo largo de todo el recorrido en una gráfica de líneas.
    1. **RQ-FN-12-01**: Además, se mostrará una línea con la velocidad media llevada en cada vuelta o segmento.
13. **RQ-FN-13**: El sistema permitirá visualizar en cada momento el punto en los tres gráficos anteriores y en el mapa, según se mueve el puntero del ratón a lo largo de los datos de un gráfico o de la trayectoria visualizada en el mapa.
14. **RQ-FN-14**: El sistema mostrará una tabla que resuma cómo ha ido la actividad física y permita realizar operaciones sobre los segmentos.
    1. **RQ-FN-14-01**: El sistema visualizará los datos para cada segmento o vuelta:
        - La fecha en la que empezó ese segmento.
        - La intensidad con la que se entrenó en ese tramo específico.
        - La duración en recorrer ese trayecto medida en segundos.
        - La distancia total de ese trayecto medida en metros.
        - La velocidad media llevada en este segmento expresada en metros por segundo.
        - La frecuencia cardiaca media en número de pulsaciones por minuto.
        - La máxima velocidad que se ha alcanzado en este recorrido expresada también en metros por segundo.
        - La frecuencia cardiaca más alta que ha tenido el usuario en este segmento también expresado en pulsaciones por minuto.
        - El número de calorías que se han obtenido en este segmento.
    2. **RQ-FN-14-02**: El sistema permitirá borrar una vuelta, seleccionando la fila de la tabla que contiene la información del segmento a borrar.
    3. **RQ-FN-14-03**: El sistema tendrá la funcionalidad de poder unir dos filas contiguas seleccionando cada una de ellas en la tabla.
15. **RQ-FN-15**: El sistema permitirá exportar la aplicación modificada a un fichero xml.
    1. **RQ-FN-15-01**: Ficheros con un esquema tcx.
    2. **RQ-FN-15-02**: Ficheros con un esquema gpx.
16. **RQ-FN-16**: El sistema permitirá exportar el fichero original de la ruta que se ha procesado. Se obtendrá el fichero de entrada aun habiéndose modificado la actividad deportiva.
17. **RQ-FN-17**: El sistema leerá de base de datos las actividades deportivas mediante su identificador único.

- Requisitos no funcionales:

1. **RQ-NF-01**: El sistema mostrará una página web al usuario una vez que acceda al enlace donde estará alojada la aplicación a través de una petición GET HTTP.
2. **RQ-NF-02**: El sistema no tendrá versión para dispositivos móviles, al menos en una primera versión.
3. **RQ-NF-03**: El sistema expondrá una interfaz pública que proveerá a aplicaciones de terceros usar todos sus servicios.
4. **RQ-NF-04**: El sistema no tendrá ninguna tecnología de autenticación. No hay necesidad en una primera versión de algún servicio como Cognito[26] o OAuth[27].
5. **RQ-NF-05**: El sistema almacenará los datos usando un modelo común en una base de datos NoSQL. En concreto se usará una base de datos basada en documentos llamada MongoDB[28].
6. **RQ-NF-06**: El sistema almacenará los archivos originales usando el servicio S3[29] de AWS.
7. **RQ-NF-07**: La aplicación web será sencilla, tendrá un diseño limpio. Tan solo aparecerá una cabecera con el nombre de la aplicación, el formulario de entrada y el panel principal.
8. **RQ-NF-08**: La aplicación mostrará un símbolo que indicará que se está procesando una acción.
9. **RQ-NF-09**: El sistema usará los mapas que proporciona Google.

Casos de uso

Todos estas características, que han de ser cumplidas por parte de la aplicación, están descritas desde la perspectiva del sistema. Si queremos ver las interacciones que tienen lugar entre el usuario y la aplicación tenemos que recurrir a los casos de uso.

[26] https://aws.amazon.com/es/cognito/
[27] https://es.wikipedia.org/wiki/OAuth
[28] https://www.mongodb.com
[29] https://aws.amazon.com/es/s3/

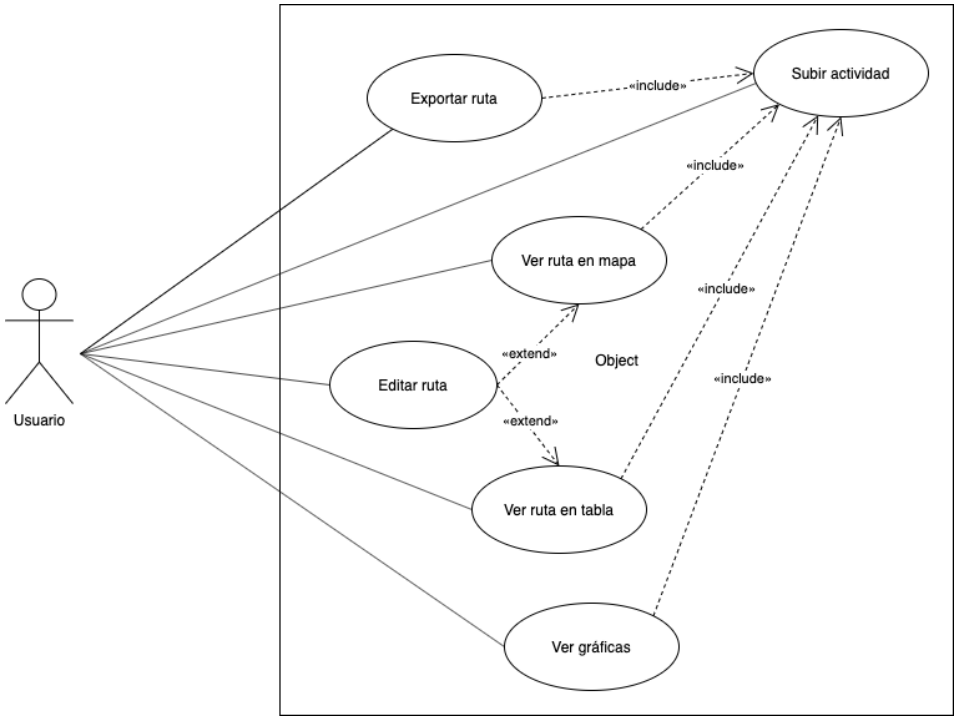


Ilustración 1

En el Ilustración 1 se representan los casos de uso del sistema a desarrollar. En las tablas CU 1, CU 2, CU 3, CU 4, CU 5 y CU 6.

CU-01	**Subir Actividad**
Dependencias	**RQ-FN-01**: El sistema en una primera versión no proporcionará ningún mecanismo para que los usuarios se autentiquen. **RQ-FN-02**: El sistema leerá actividades físicas a través de un fichero xml. **RQ-FN-03**: El sistema podrá leer una serie de esquemas xml. **RQ-FN-03-01**: Ficheros con un esquema gpx. **RQ-FN-03-02**: Ficheros con un esquema tcx. **RQ-FN-04**: El sistema procesará la información que hay en el fichero y lo transformará a un modelo común que entienda. **RQ-FN-05**: El sistema persistirá todas las actividades deportivas contenidas en el fichero de entrada que sean correctas. **RQ-FN-06**: El sistema almacenará en un repositorio el fichero xml original. **RQ-NF-01**: El sistema mostrará una página web al usuario una vez que acceda al enlace donde estará alojada la aplicación a través de una petición GET HTTP.

	RQ-NF-02: El sistema no tendrá versión para dispositivos móviles, al menos en una primera versión. **RQ-NF-05**: El sistema almacenará los datos usando un modelo común en una base de datos NoSQL. En concreto se usará una base de datos basada en documentos llamada MongoDB. **RQ-NF-06**: El sistema almacenará los archivos originales usando el servicio S3 de AWS.	
Precondiciones	El fichero de entrada es de tipo xml. El fichero de entrada tiene un esquema tcx o gpx.	
Descripción	El sistema debe comportarse como se muestra en la siguiente fila de la tabla para permitirle a un usuario subir una actividad física.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación a través del enlace web.
	2	El usuario selecciona el fichero que contiene la actividad física que quiere subir al sistema.
	3	El usuario selecciona el tipo de esquema que es el fichero xml.
	4	El usuario envía el formulario a través del botón enviar.
	5	El sistema muestra una flecha que indica que se está procesando la acción
	6	El sistema recibe el fichero enviado
	7	El sistema procesa el fichero en función del tipo de esquema enviado como parámetro
	8	El sistema convierte los datos de entrada al modelo común del sistema
	9	El sistema guarda este modelo en la base de datos.
	10	Si se ha podido guardar la actividad física, entonces se almacena el fichero original en un repositorio de un tercero.

	11	El sistema devuelve al usuario el id autogenerado por el mismo con el que se ha guardado esta actividad física.
Postcondición	El usuario recibe el identificador único de la actividad subida con el que podrá realizar otras acciones futuras, imprescindible para realizarlas.	
Excepciones	Paso	Acción
	4	El sistema produce un error debido a que el fichero no tiene un esquema xml reconocido de tipo *tcx* o *gpx*.
Comentarios	Este caso de uso es imprescindible para que los demás se puedan desarrollar. Esto se indica gracias al uso de la relación «include».	

CU 1

CU-02	**Ver ruta en mapa**
Dependencias	**RQ-FN-07**: El sistema visualizará la ruta deportiva en un mapa que permita ver cual fue el rumbo recorrido por el usuario de la aplicación. **RQ-FN-13**: El sistema permitirá visualizar en cada momento el punto en los tres gráficos anteriores y en el mapa, según se mueve el puntero del ratón a lo largo de los datos de un gráfico o de la trayectoria visualizada en el mapa. **RQ-FN-17**: El sistema leerá de base de datos las actividades deportivas mediante su identificador único. **RQ-NF-05**: El sistema almacenará los datos usando un modelo común en una base de datos NoSQL. En concreto se usará una base de datos basada en documentos llamada MongoDB. **RQ-NF-09**: El sistema usará los mapas que proporciona Google.
Precondiciones	El parámetro de entrada es el identificador único de la actividad física obtenido en el CU 1. Debe existir una actividad deportiva asociada al identificador. Se debe haber subido un fichero con la actividad física previamente.
Descripción	El sistema debe comportarse como se muestra en las siguientes filas de la tabla para permitirle a un usuario ver una actividad física en el mapa geoespacial haciendo uso de la librería que

	proporciona la compañía Google, con su modulo para el tratamiento de mapas.	
Secuencia normal	Paso	Acción
	1	Después de haber ejecutado realizado el caso descrito en CU 1, se tendrá el identificador de la actividad física.
	2	El sistema consultará en la base de datos y obtendrá los datos de la actividad deportiva.
	3	El sistema procesa los datos del modelo obtenidos de base de datos.
	4	El modelo es visualizado en el mapa.
	5	El sistema muestra una visualización conjunta con las gráficas que serán descritas en el caso de uso CU 5, mostrando en todo momento el punto al que pertenece.
Postcondición	El usuario verá el mapa con la ruta deportiva relativa al identificador único. Los puntos tendrán que estar correctamente posicionados.	
Excepciones	Paso	Acción
	2	El identificador no corresponde con niguna actividad deportiva almacenada en base de datos.
Comentarios	Para que este caso de uso tenga sentido es necesario que se haya subido una actividad física y autogenerado un identificador asociado. Gracias a la relación «include» se puede indicar esta condición.	

CU 2

CU-03	**Ver ruta en tabla**
Dependencias	**RQ-FN-14**: El sistema mostrará una tabla que resuma cómo ha ido la actividad física y permita realizar operaciones sobre los segmentos. **RQ-FN-17**: El sistema leerá de base de datos las actividades deportivas mediante su identificador único. **RQ-NF-05**: El sistema almacenará los datos usando un modelo común en una base de datos NoSQL. En concreto se usará una base de datos basada en documentos llamada MongoDB.

Precondiciones	El parámetro de entrada es el identificador único de la actividad física. Debe existir una actividad deportiva asociada al id. Debe haberse subido una fichero con la actividad asociada.	
Descripción	El sistema debe comportarse como se muestra en las siguientes filas de la tabla para permitir a un usuario ver una actividad física con los valore de los segmentos desde una tabla.	
Secuencia normal	Paso	Acción
	1	Después de haber ejecutado la acción CU 1 y de recibir como respuesta el id de la actividad física.
	2	El sistema consultará en la base de datos y obtendrá los datos de la actividad deportiva.
	3	El sistema reconoce el modelo.
	4	El sistema identifica todos las vueltas asociadas a esa ruta.
	5	El sistema muestra para cada segmento la información asociada: fecha de inicio, intensidad, duración, distancia, velocidad media, frecuencia cardiaca media, velocidad máxima, frecuencia cardiaca máxima y el número de calorias consumidas.
Postcondición	El usuario verá todos los segmentos de la ruta deportiva asociada. Si no existen, entonces solo habrá un segmento que resumirá toda la actividad física.	
Excepciones	Paso	Acción
	2	Identificador no corresponde con niguna actividad deportiva almacenada en base de datos.
Comentarios	Para esta acción es necesita que se haya subido una actividad física y autogenerado un identificador asociado a ella. Como indica la relación «include».	

CU 3

CU-04	**Editar ruta**

Dependencias	**RQ-FN-08**: El sistema permitirá eliminar un determinado punto seleccionándolo desde el mapa en el que se ha mostrado la actividad deportiva. **RQ-FN-09**: El sistema tendrá la funcionalidad de dividir la ruta deportiva desde el mapa seleccionando el punto especifico. **RQ-FN-14-02**: El sistema permitirá borrar una vuelta, seleccionando la fila de la tabla que contiene la información del segmento a borrar. **RQ-FN-14-03**: El sistema tendrá la funcionalidad de poder unir dos filas contiguas seleccionando cada una de ellas en la tabla.		
Precondiciones	El parámetro de entrada es el identificador único de la actividad física. Debe existir una actividad deportiva asociada al id. Debe haberse subido una fichero con la actividad asociada.		
Descripción	El sistema debe comportarse como se muestra en las siguientes filas de la tabla para permitir a un usuario editar una actividad física. Habrá dos posibilidades, en función de qué caso de uso se haya implementado antes.		
Secuencia normal	Paso	Acción	
	Ver ruta en mapa		
	1	Después de CU 2 y de interactuar con el mapa.	
	2	**RQ-FN-08**: El sistema permitirá eliminar un determinado punto seleccionándolo desde el mapa en el que se ha mostrado la actividad deportiva.	**RQ-FN-09**: El sistema tendrá la funcionalidad de dividir la ruta deportiva desde el mapa seleccionando el punto especifico.
		El usuario podrá borrar un punto seleccionándolo en el mapa	El usuario podrá separar un segmento en dos vueltas a partir de un punto.
	3	El usuario hará click en el botón correspondiento que aparecerá sobre este punto	
	4	El sistema borrará el punto de los datos que tiene.	El sistema dividará el segmento seleccionado

			por el punto que se ha definido
	5	El sistema guardará esta información en la base de datos.	
	6	El sistema mostrará la información actualizada al usuario.	
	Ver ruta en tabla		
	1	Después de CU 3 y de interactuar con la tabla.	
	2	**RQ-FN-14-02**: El sistema permitirá borrar una vuelta, seleccionando la fila de la tabla que contiene la información del segmento a borrar.	**RQ-FN-14-03**: El sistema tendrá la funcionalidad de poder unir dos filas contiguas seleccionando cada una de ellas en la tabla.
		El usuario seleccionará la fila de la vuelta que quiere borrar	El usuario seleccionará los dos segmentos que quiere unir en uno.
	3	El usuario enviará la petición haciendo click sobre el icono de borrado en la tabla.	El usuario enviará la petición haciendo click en el icono de juntar que se mostrará en la tabla.
	4	El sistema borrar el segmento de los datos que tiene.	El sistema juntará los dos segmentos formando uno nuevo.
	5	El sistema guardará esta información en la base de datos.	
	6	El sistema mostrará la información al usuario actualizada.	
Postcondición	El usuario verá todas las modificaciones en la tabla y en el mapa. Es decir, tanto si edita la ruta en el mapa como en la tabla, el resultado se verá en ambos. El identificador nunca podrá ser cambiado, ya que identifica de manera unívoca a la actividad deportiva.		
Excepciones	Paso	Acción	

	3	Identificador no corresponde con niguna actividad deportiva almacenada en base de datos.
Comentarios	Para esta acción es necesita que se haya subido una actividad física y autogenerado un identificador asociado a ella. Como indica la relación «include». Con la relación «extend» se extiende, valga la redundancia, la funcionalidad de una acción, es por eso que se use para la visualización de la tabla y del mapa ya que desde estos elementos se puede editar la ruta.	

CU 4

CU-05	**Ver gráficas**	
Dependencias	**RQ-FN-10**: El sistema mostrará la altitud a lo largo de todo el recorrido en una gráfica de áreas. **RQ-FN-11**: El sistema mostrará la frecuencia cardiaca a lo largo de todo el recorrido en una gráfica de barras. **RQ-FN-11-01**: El sistema también mostrará una línea con la frecuencia media llevada en cada vuelta o segmento. **RQ-FN-12**: El sistema mostrará la velocidad a lo largo de todo el recorrido en una gráfica de líneas. **RQ-FN-12-01**: Además, se mostrará una línea con la velocidad media llevada en cada vuelta o segmento. **RQ-FN-17**: El sistema leerá de base de datos las actividades deportivas mediante su identificador único.	
Precondiciones	Debe existir una actividad deportiva asociada al identificador. Debe haberse subido una fichero con la actividad asociada. La información puede no tener datos relacionados con la altitud, frecuencia cardiaca y velocidad. En este caso, los driagramas no serán mostrados.	
Descripción	El sistema debe comportarse como se muestra en las siguientes filas de la tabla para permitir a un usuario de la aplicación ver las gráficas.	
Secuencia normal	Paso	Acción
	1	Después de haber ejecutado la acción CU 1 y de recibir como respuesta el id de la actividad física.
	2	El sistema consultará en la base de datos y obtendrá los datos de la actividad deportiva.

	3	El sistema reconoce el modelo.		
	4	El sistema identifica todos los datos relativos a una vuelta para crear la gráfica.		
	5	El sistema mostrará un grafico de areas con la elevación por la que ha pasado a lo largo de la ruta el usuario en cada momento.	El sistema mostrará un grafico de barras con la frecuencia cardiaca que ha mantenido el usuario en cada momento. Además, se mostrará una linea con frecuencia cardiaca media en cada vuelta.	El sistema mostrará un grafico de linea con la velocidad llevada a lo largo de todo el trayecto. Además, se mostrará una línea con la velocidad media en cada segmento.
	6	El sistema actualizará en todo momento en los otros gráficos y en el mapa el punto concreto con el que el ratón tiene el foco.		
	7	De manera complementaria, el sistema podrá calcular cierta información para la generación del gráfico. Por ejemplo, a partir de las cordenadas se puede saber la altura. O también se puede obtener la velocidad en funcion de coordenadas y tiempo.		
Postcondición	El usuario verá los tres gráficos siempre que exista información para cada uno de ellos. Puede haber casos en los que esta información esté incompleta o falte de forma total.			
Excepciones	Paso	Acción		
	2	Identificador no corresponde con niguna actividad deportiva almacenada en base de datos.		
Comentarios	Para esta acción es necesita que se haya subido una actividiidad física y autogenerado un identificador asociado a ella. Esta condición se indica con la relación «include».			

CU 5

CU-06	**Exportar ruta**

<table>
<tr><td>Dependencias</td><td colspan="3">RQ-FN-15: El sistema permitirá exportar la aplicación modificada a un fichero xml.
RQ-FN-15-01: Ficheros con un esquema tcx.
RQ-FN-15-02: Ficheros con un esquema gpx.
RQ-FN-16: El sistema permitirá exportar el fichero original de la ruta que se ha procesado. Se obtendrá el fichero de entrada aun habiéndose modificado la actividad deportiva.
RQ-NF-06: El sistema almacenará los archivos originales usando el servicio S3 de AWS.</td></tr>
<tr><td>Precondiciones</td><td colspan="3">El parámetro de entrada es el identificador único de la actividad física.
Debe existir una actividad deportiva asociada al id.
Debe haberse subido una fichero con la actividad asociada.</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema debe comportarse como se muestra en las siguientes filas de la tabla de secuencia de pasos para permitir a un usuario exportar una actividad física.</td></tr>
<tr><td>Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td rowspan="4"></td><td>1</td><td colspan="2">Después de haber ejecutado la acción CU 1 y de recibir como respuesta el identificador de la actividad física.</td></tr>
<tr><td>2</td><td colspan="2">El sistema buscará la actividad física correspondiente al identificador que se ha enviado como parámetro.</td></tr>
<tr><td rowspan="2">3</td><td>RQ-FN-15: El sistema permitirá exportar la aplicación modificada a un fichero xml.</td><td>RQ-FN-16: El sistema permitirá exportar el fichero original de la ruta que se ha procesado. Se obtendrá el fichero de entrada aun habiéndose modificado la actividad deportiva.</td></tr>
<tr><td>El usuario podrá elegir el tipo de esquema tcx o gpx del fichero xml que quiere como resultado de la exportación de la actividad con la que se ha estado trabajando.</td><td>El usuario podrá descargarse el fichero original asociado con la actividad deportiva</td></tr>
</table>

	4	El sistema transformará el modelo de actividades deportivas al esquema que el usuario haya seleccionado.	El sistema obtendrá el fichero original del repositorio distribuido y lo descargará.
	5	El fichero será enviado al usuario.	
Postcondición	El usuario podrá ver y modificar el fichero descargado en el formato que haya seleccionado o los datos originales de la actividad física con la que se ha estado trabajando.		
Excepciones	Paso	Acción	
	2	Identificador no corresponde con niguna actividad deportiva almacenada en base de datos.	
Comentarios	Para esta acción es necesario que se haya subido una actividad física y autogenerado un identificador asociado. Esta condición se indica mediantel la relación «include».		

CU 6

Los requisitos y los casos de uso son formas de descripción de requerimientos tipicos de los procesos tradicionales poniendo énfasis en las operaciones del sistema y en las interacciones entre el usuario y el sistema, respectivamente. No obstante, también pueden ser utilizados en metodologías ágiles ya que proporcionan un nivel de detalle muy alto de cada cualidad que el sistema a desarrollar ha de cumplir. Por otro lado, si estamos trabajando en un equipo multidisciplinar y queremos fomentar el nivel de colaboración y comunicación es mejor definir los requisitos con lo que en el mundo ágil se denomina historias de usuario.

Historias de usuario

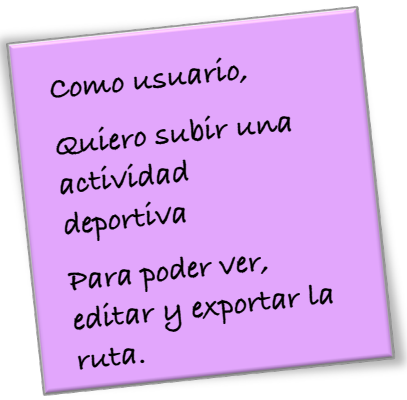
Las historias de usuario representan una forma corta y sencilla de describir características de un producto software desde la perspectiva de la persona que quiere esa funcionalidad, normalmente el usuario o cliente del sistema. La estructura para formular estos elementos siguen este patrón:

Como ..., quiero ... para ...

Las historias se escriben en pósits y se colocan en la pared o en una pizarra facilitando de esta forma la planificación y el debate entre todos los desarrolladores y las personas involucradas en el proyecto que se ha de realizar. Para que una historia sea lo suficientemente buena ha de cumplir la regla INVEST, acrónimo en inglés de:

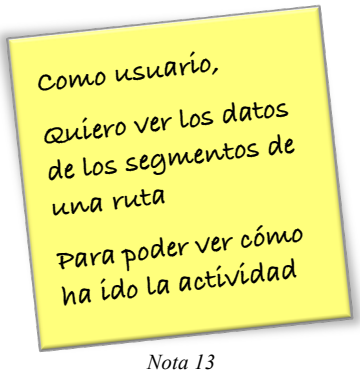
- *Independant*: independientes entre sí para que se puedan coger en cualquier iteración que el Product Owner crea conveniente.
- *Negotiable*, debe ser acordada o negociada con el Product Owner para que se limite la funcionalidad a desarrollar.
- *Valuable*, es decir, que aporte valor para el usuario.
- *Estimable*: el equipo ha de definir el tiempo que le llevará realizarla.
- *Small*: lo suficientemente pequeñas para que el equipo lo pueda producir en menos de una iteración, a ser posible que el equipo pueda completar varias en un sprint.
- *Testable*: que se pueda comprobar que la funcionalidad que provee la historia responde a lo esperado.

Cuando una historia es demasiado grande se llama épica y conviene añadir detalles para que no haya dudas de qué hacer. Existen dos posibilidades para conocer en profundidad una épica: dividirla en múltiples historias de usuario o añadir condiciones de satisfacción. Se pueden identificar épicas como historias de usuario que no se pueden entregar en una sola iteración o que son lo suficientemente largas para poder ser expresadas en pequeñas historias. Por lo tanto, una épica puede ser un proyecto porque tiene un objetivo de ser definido en un espacio de tiempo finito con el resultado de mejorar o crear un producto. En la Nota 1 se puede ver cómo se escribe una épica en la práctica del producto que se necesita entregar.

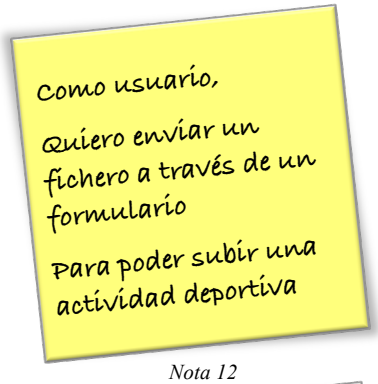


Nota 1

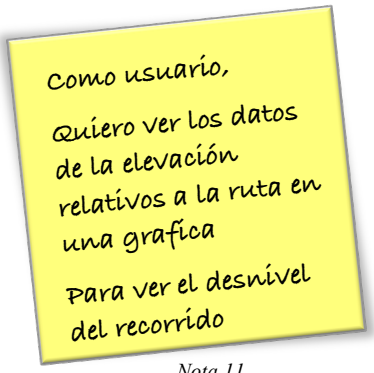
Para que sea más fácil la estimación de todo lo que hay que realizar, es recomendable aplicar el procedimiento "divide y vencerás" y así obtener una imagen más clara de todos los pasos que hay que completar para finalizar una épica o una historia demasiado grande.



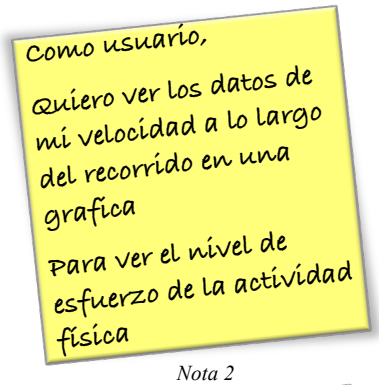
Nota 13



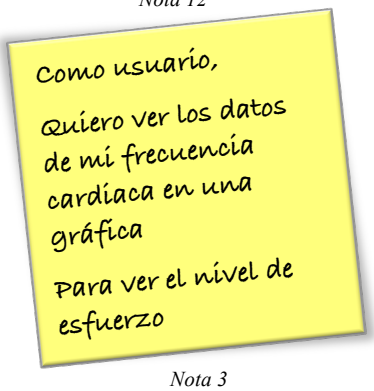
Nota 12



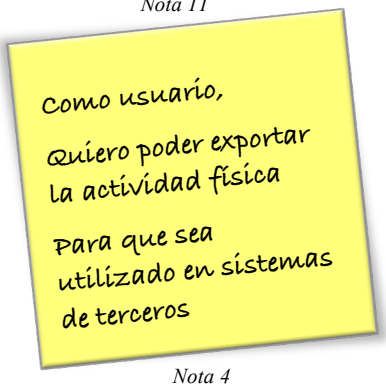
Nota 11



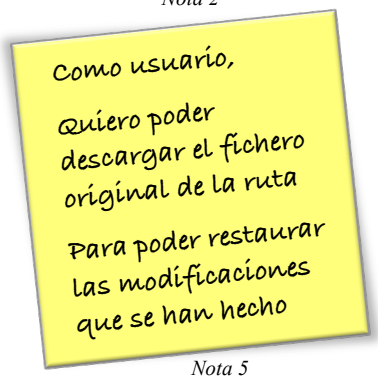
Nota 2



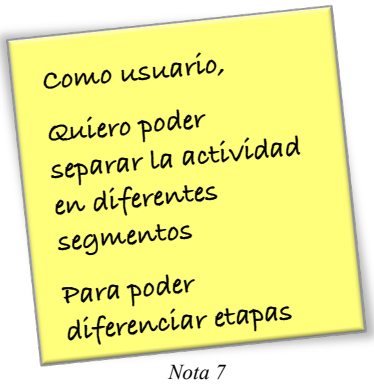
Nota 3



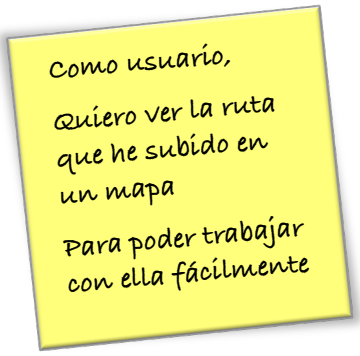
Nota 4



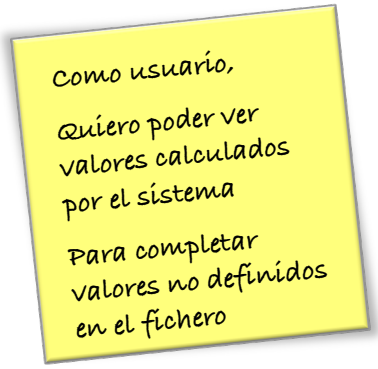
Nota 5



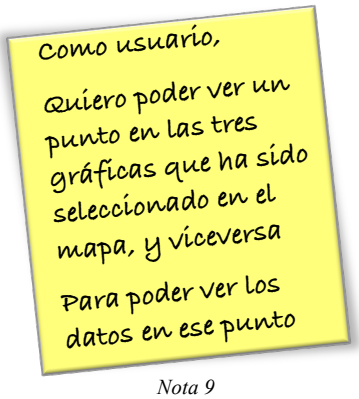
Nota 7



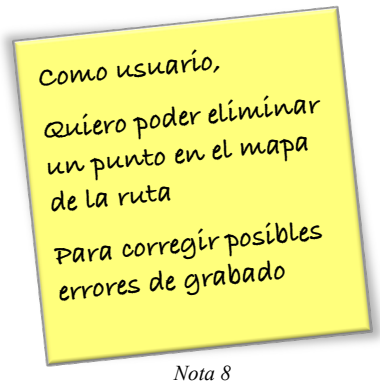
Nota 6



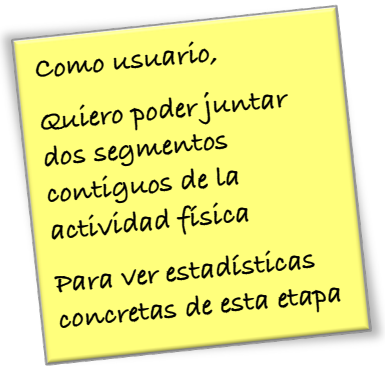
Nota 10



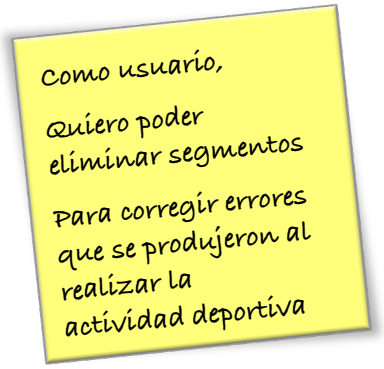
Nota 9



Nota 8



Nota 15



Nota 14

Más arriba, se visualizan todas las historias de usuario que podemos obtener mediante este proceso de la épica definida en la Nota 1 como se muestran en la Nota 13, la Nota 12, la Nota 11, la Nota 2, la Nota 3, la Nota 4, la Nota 5, la Nota 7, la Nota 6, la Nota 10, la Nota 9, la Nota 8, la Nota 15 y la Nota 14.

Además de definir cada historia de usuario, en la siguiente tabla se van a añadir los criterios de aceptación para cada una de ellas, incluyendo más detalle y, por tanto, que no haya duda de qué es lo que tiene que implementarse en cada historia de usuario.

Historia de usuario	**Criterios de aceptación**
Nota 13	Si no hay definidos varios segmentos, por defecto, el servicio mostrará el segmento que corresponderá al total de la ruta. Cada segmento será identificado con un color diferente. Cada segmento será identificado con un número único. La hora a la que se empezó un segmento tiene el formato: hh:mm:ss La intensidad con la que el usuario entrenó tendrá un formato libre. Si el valor viene vacío, se muestra un guion. La duración en segundos de ese segmento medida en segundos. La longitud de ese segmento medida en metros. La velocidad media del segmento medida en metros por segundo. La frecuencia cardiaca media en pulsaciones por minuto. La velocidad máxima medida en metros por segundo. La frecuencia cardiaca máxima medida en pulsaciones por minuto. El numero de calorías gastadas en cada segmento medidas en calorías.
Nota 12	La actividad deportiva está en un fichero xml. Solo se puede subir un fichero. Solo se aceptan ficheros xml con un esquema tcx o gpx. El tamaño máximo del fichero será de 10MB.

Nota 11	Los datos son mostrados en metros. La gráfica contendrá un gráfico de áreas que representa la altitud del terreno por donde el usuario ha hecho la ruta deportiva.
Nota 2	Los datos son mostrados en metros por segundo. La gráfica contiene una línea con la variación de la velocidad a lo largo de cada punto. La gráfica contiene una línea que conecta las velocidades medias obtenidas en cada segmento.
Nota 3	Los datos se muestran en pulsaciones por minuto. La gráfica contiene multitud de barras por cada dato de la actividad que representará la frecuencia cardiaca para ese punto. La gráfica contiene una línea que conecta las frecuencias cardiacas medias que el usuario ha tenido para cada segmento.
Nota 4	Al cargar la actividad deportiva, aparecen dos botones. El primer botón será para exportar la actividad en el estado actual en formato xml con el esquema tcx. El segundo botón será para exportar la actividad en el estado actual en formato xml con el esquema gpx. Al descargar la ruta, el usuario obtiene el fichero en su dispositivo electrónico.
Nota 5	Al cargar la actividad deportiva, aparece un botón para descargar el fichero. Al descargar la ruta, el usuario obtiene el fichero en su dispositivo electrónico.
Nota 7	Debajo de la descripción del punto en el mapa aparece un botón para dividir la ruta. El trayecto aparece dividido en dos segmentos a partir del punto seleccionado a través del mapa.
Nota 6	La ruta se visualiza con diferentes colores, si existen más de un segmento. Si solo existe un segmento, aparece un solo color. Si se desplaza el ratón sobre la ruta, se puede seleccionar cada punto. Al seleccionar un punto, se muestra una descripción con los datos para ese punto en la actividad deportiva (unidad de medida): tiempo (hh:mm:ss), latitud (grados), longitud (grados), altitud (metros), distancia (metros), velocidad (metros por segundo) y frecuencia cardiaca (pulsaciones por minuto).

Nota 10	Se proporciona la altitud cuando este parámetro no venga en el fichero de entrada. Se busca en función de la latitud y longitud usando servicio externo. Se proporciona la velocidad (valor aproximado) cuando este valor no tenga valor en función de las diferentes posiciones entre dos puntos (latitud y longitud).
Nota 9	Si desplazo el ratón sobre la trayectoria mostrada en el mapa, se identifica el valor que corresponde en las tres gráficas. Si desplazo el ratón sobre una de las gráficas, el punto correspondiente a las dos otras y al mapa también se muestra.
Nota 8	Al acceder a uno de los puntos del mapa, en la ventana informativa se muestra al final un botón para permitir borrarlo de la trayectoria. Si el punto a borrar esta al inicio o al final de la ruta, se borra y no se hace nada más. Si el punto a borrar esta en el medio, se juntan los otros dos puntos contiguos.
Nota 15	Al mostrarse los segmentos en la tabla, el usuario puede unir dos segmentos siempre y cuando los segmentos seleccionados sean contiguos. El botón se habilita en la parte superior derecha de la tabla.
Nota 14	Al mostrarse los segmentos en la tabla, el usuario puede seleccionar uno o más segmentos a borrar. El botón se habilita en la parte superior derecha de la tabla al seleccionar algún segmento.

Para obtener las historias de usuarios existen metodologías que facilitan la identificación de cada uno de estos elementos. Una de ellas es la *User Story Mapping* que permite generar historias de forma colaborativa entre el *Product Owner* y las demás partes involucradas en el proyecto. Para formar el conjunto de historias de usuario utiliza dos dimensiones, por un lado, la dimensión horizontal, denominada la espina dorsal del proceso (*backbone* en inglés), que se centra en épicas o grandes funcionalidades ordenadas por el flujo de actividad que el usuario tiene en el sistema. Por otro, la dimensión vertical que define cada historia de usuario priorizada por valor que proporciona al usuario, aquellas que más valor aportan estarán situadas en la parte superior de la tabla. Volviendo al ejemplo de la aplicación que se ha de desarrollar, en la Tabla 1 se define esta estructura.

Backbone	*Subida*	*Visualización*	*Modificación*	*Exportación*
Historias de Usuario	Nota 12	Nota 6	Nota 7	Nota 4
		Nota 13	Nota 9	Nota 5
		Nota 11	Nota 14	
		Nota 2	Nota 15	
		Nota 3		
		Nota 8		
		Nota 10		

Tabla 1

Es de gran utilidad este proceso para la obtención de historias de usuario en la migración de una aplicación o en el diseño de un nuevo sistema. Al producto resultante se le aplica un filtrado que determine cuales son las actividades que más valor le darán al usuario, lo que se denomina Producto Mínimo Viable (en inglés MVP[30]). En el ejemplo de arriba, en la Tabla 1, se ha omitido este paso intermediario, sin embargo, se van a mostrar en la Ilustración 2 algunas historias de usuario que no se encuentran definidas en esta tabla.

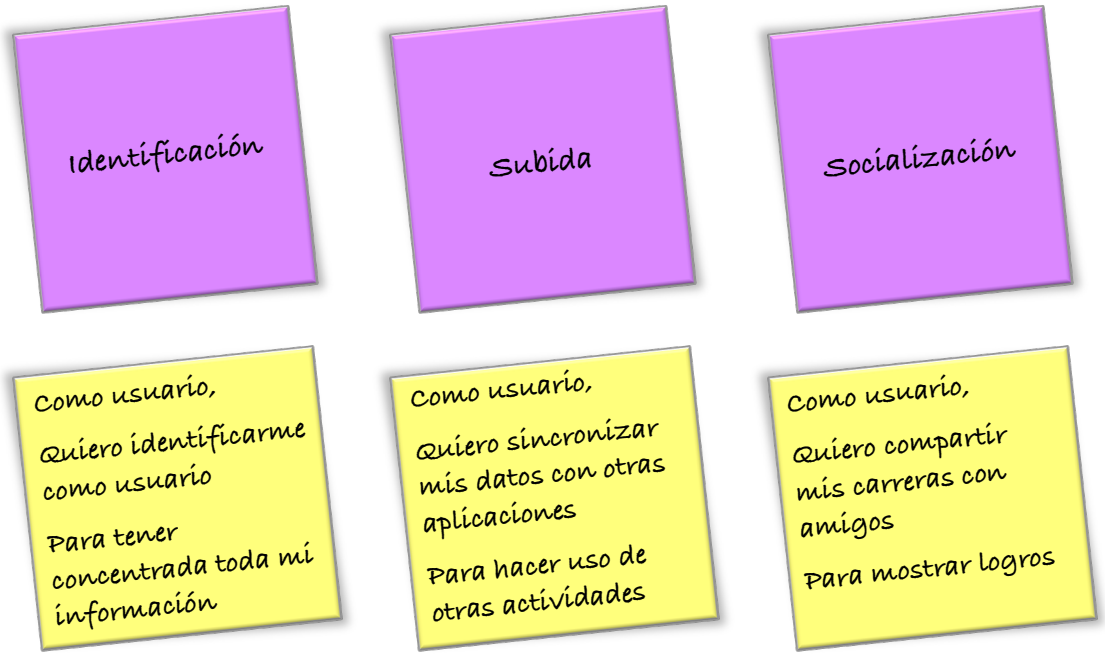


Ilustración 2

Estas tres historias formarían parte del *product backlog* de futuras versiones de la aplicación, todo ello dependiendo del *feedback* que los clientes proporcionasen.

[30] https://es.wikipedia.org/wiki/Producto_viable_m%C3%ADnimo

Todas las historias definidas en la Tabla 1 definen el producto con la funcionalidad básica que aportaría gran valor al usuario y que se define como *la versión más concisa y pequeña de un producto que desarrolla un equipo para obtener la mayor cantidad de feedback y validaciones con usuarios al menor esfuerzo* (Schelstraete, 2018).

Por lo tanto, con esta primera versión se le proporciona a los usuarios una buena experiencia, proveyendo a los interesados de opiniones tanto positivas como negativas y fomentando la comunicación que hace que se vaya mejorando la aplicación en futuras versiones.

En el siguiente punto se pondrán en practica todas estas historias de usuario y siguiendo una metodología ágil se definirá la planificación del proyecto.

Implementación

En el punto anterior denominado *Especificación de Requisitos*, se han comentado los diferentes elementos que existen a la hora de definir los requisitos tanto para un proceso tradicional como para uno ágil. Además, más adelante, en el punto *Planificación de proyectos* del apartado *Aspectos Teóricos*, se analizarán los diferentes tipos que existen y aunque las condiciones en las que se desarrolla todo este proyecto no son las más indicadas para un proyecto ágil, se va a escoger esta metodología para la planificación y gestión del proyecto presente. Se va a suponer que la figura de cliente recae en mí, además de la de desarrollador tanto *backend*, *frontend* como *devops* y la figura que gestiona al equipo. Por tanto, se va a implantar un tipo de planificación con una modelo ágil que permita entregas pequeñas al cliente y que mejore la posibilidad de manejar cualquier contratiempo o cambio que pueda ocurrir en la fase de creación del software.

Existen diferentes formas de implementar metodologías ágiles, una de las más conocidas en el mundo software es Scrum[31] que define un marco de trabajo iterativo en el que aplica buenas prácticas que permiten trabajar de forma colaborativa y en equipo, obteniendo, de forma eficiente, un buen resultado.

Esta metodología define una serie de roles imprescindibles:

- El dueño del producto (*Product Owner*) que se encarga de que el equipo trabaje correctamente, ayudando a escribir las historias de usuario y a priorizar el desarrollo de unas historias frente a otras.
- El *Scrum Master* es el que se encarga en solucionar cualquier contratiempo que pueda ocurrir en cada iteración. Como indican los principios del manifiesto ágil, en estas metodologías no hay ningún líder de equipo, tan solo es una figura para tratar de solucionar cualquier distracción que le ocurra al equipo a lo largo de todo el Sprint.
- El equipo estará formado de 3 a 9 personas y será el encargado de desarrollar y entregar el producto al cliente. Entre ellos puede haber diferentes roles: *backend*, *frontend*, diseñador *UX*, calidad, *devops*, …

Existen otros roles que no están tan presentes en todo el proceso de creación del producto, aunque si es importante que estén mucho tiempo con los demás integrantes para que el proyecto tome el mejor camino posible. Uno de ellos es el grupo de personas (*stackeholders*) que están interesados en la realización del proyecto, aportando diversas ideas y funcionalidades para desarrollar. A esto grupo de usuarios la finalización del proyecto en condiciones óptimas les aporta gran valor de retorno económico o de intereses.

Volviendo al proyecto que ha de desarrollarse, en este caso, la figura de *stakeholder* la realizo yo, basándome en todas las carencias obtenidas tras haber analizado en el punto *Alternativas de mercado* del apartado de *Introducción* el panorama actual que hay respecto a aplicaciones de análisis de actividades físicas.

Por otro lado, en base a todas estas funcionalidades que surgen del interés para desarrollar el sistema, también realizo la figura de *Product Owner* creando las historias de usuario y dando prioridad teniendo en cuenta el valor que aporta cada una de ellas al usuario final

[31] https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

de la aplicación. En la Tabla 1 se pueden ver las historias de usuario priorizadas de arriba abajo, en posiciones más altas una mayor prioridad, usando la técnica comentada más arriba.

La figura de *Scrum Master* también la realizo yo, por ejemplo, los bloqueos que han surgido a lo largo de la realización del proyecto los he ido solucionando, hablando con el director del trabajo fin de grado o buscando información en internet.

Por otro lado, el equipo de desarrollo ha estado concentrado también en mi persona, desarrollando los siguientes roles:

- Desarrollador *backend*: desarrollando del sistema interno que realiza el procesamiento de la información y provee una interfaz no gráfica, a través de peticiones http, para el acceso a todas estas funcionalidades.
- Desarrollador *frontend*: desarrollando la aplicación web que le proporciona al cliente el acceso mediante una interfaz gráfica a todas las herramientas del sistema.
- Diseñador *UX*: diseñando la parte visual de la aplicación, dónde debe colocarse y cómo debe visualizarse cada componente, además de definir la forma en la que el usuario interacciona con cada uno de estos elementos.
- *Devops*: Desplegando la aplicación en la infraestructura correcta y necesaria para el correcto funcionamiento del sistema: base de datos, servidor de aplicaciones para el *frontend* y el *backend*.

Por otro lado, como se comentó más arriba, esta metodología tiene un modelo de trabajo iterativo e incremental que favorece las entregas rápidas y la pronta salida a producción una vez que el desarrollo comienza.

Por lo tanto, existen diferentes etapas en el proceso de definir, desarrollar y finalizar cada iteración o Sprint. En esta metodología a cada iteración se le denomina Sprint y tienen una duración de entre 2 a 3 semanas, aunque puede ir variando en función del ritmo del equipo tras finalizar cada iteración. Cada vez que se finaliza un Sprint, se tiene una versión funcional de la aplicación que aporta valor a los interesados y que puede ser puesta en producción (si se ha establecido un flujo de despliegue continuo[32]) y, por tanto, entrar en contacto con el usuario final.

Las diferentes etapas de esta metodología son:

1. La planificación del *Sprint o Sprint Planning*. Como se definió en el punto *Especificación de Requisitos* de este mismo apartado, es necesario que antes de empezar se tengan todos los requisitos que han sido formulados en el punto *Historias de usuario* conformando, de este modo, lo que se llama en el mundo ágil el *Product Backlog*. Si no se ha llevado a cabo ningún Sprint todavía, este elemento contendrá todas las historias que quedan por implementar. El objetivo fundamental de esta fase es la definición del *Product Backlog Item* que contiene todas las historias que se van a realizar en esta iteración.
2. La *daily standup* que es una reunión muy breve que tiene como objetivo que cada integrante del equipo mantenga informado al resto del estado de la tarea que esta realizando y de los problemas que se ha encontrado. Seguramente otros usuarios

[32] https://en.wikipedia.org/wiki/Continuous_delivery

puedan echar una mano en determinados contratiempos a los que se hayan enfrentado con anterioridad. En esta fase también se definen las tareas que se van a llevar a lo largo del día.

3. La revisión del sprint donde se presenta el producto una vez que se han implementado todas las historias de usuario de la iteración.
4. La retrospectiva es donde se definen todos los conocimientos adquiridos durante esta vuelta, además de proponer mejoras en el proceso. Por lo tanto, el objetivo es retroalimentar la forma de trabajar, mejorándola y adaptándola al ritmo del equipo.

Una de las cosas más importantes que hay que hacer para la correcta planificación del Sprint es haber realizado las estimaciones de esfuerzo de desarrollo de todas las historias del *Product Backlog*. Al hablar de estimar seguramente se viene a la menta como unidad de medida el tiempo que se necesita para implementar y finalizar una historia de usuario, sin embargo, esta forma no emplea un criterio objetivo, sino que va íntimamente ligada a la persona encargada de realizarla. Por tanto, como el objetivo no es realizar estimaciones en función de qué persona va a desarrollar ese elemento, lo que se va a intentar medir es la complejidad, tamaño o dificultad de esa historia, teniendo en cuenta el punto de vista de todos los desarrolladores involucrados en el proyecto.

Para ello, se va a utilizar como medida los puntos de historias que representan un valor relativo, pudiendo ser diferentes para otros equipos, y que tienen como finalidad dar una idea del tamaño y esfuerzo para realizar una historia de usuario. Para ponerlo en práctica, se ha de coger una historia que sea bien conocida por todo el equipo, no siendo ni muy grande ni muy pequeña (en torno a 1-2 puntos de historia), y definir los puntos de historia que tiene, será la que se denomina pivote. En nuestro sistema, podríamos pensar en una historia que tenga diferentes tareas pero que su realización no sea ni muy difícil ni muy fácil. Por ejemplo, la historia pivote sería la descrita en la Nota 13 obteniendo 1 punto de historia. Se ha escogido porque la funcionalidad a implementar es sencilla y todo el equipo tiene claro cómo hay que realizarlo. Las tareas asociadas son: obtener los datos de base de datos, proporcionar esta información a la aplicación web y visualizar esta información según los criterios de aceptación estipulados.

Para generar los puntos de historia de los demás ítems, tan solo habrá que ver qué tan fácil o difícil será implementar las demás historias en comparación con la historia pivote. Para realizar esta comparación, se utiliza el *Scrum Poker*[33] que toma este nombre de la forma con la que actúan los jugadores de *Poker*, teniendo siempre la misma cara independientemente de las combinaciones que hayan conseguido. Por lo tanto, cada desarrollador tendrá una serie de cartas con una serie de números: 0, ½, 1, 2, 3, 4, 5, 8, 13, 20, 40, 100, ?, ∞ y ☕. De esta forma, los niveles de mayor dificultad tendrán más separación entre ellos ya que la incertidumbre será mayor. Por otro lado, si se entrega un valor ∞ la tarea será tan grande que se tendrá que descomponer en pequeñas historias para poder estimar mejor. El jugador nunca mostrará opinión positiva ni negativa sobre la historia que se está analizando. En el momento de mostrar las cartas, las respectivas personas con el valor más bajo y alto darán sus argumentos de por qué han elegido esos valores. Una vez concluido este paso, se volverá a votar y finalmente se cogerá el valor

[33] https://es.wikipedia.org/wiki/Planning_poker

más elevado, por la posibilidad de no saber con seguridad qué problemas podrán surgir en todo el proceso para completar la historia.

Historia de Usuario	**Historia de Usuario**	**Descripción**	**Puntos de Historia**
Nota 13	Como usuario, Quiero ver los datos de los segmentos de una ruta Para poder ver cómo ha ido la actividad	*Los segmentos se mostrarán en una tabla que contendrá todos los valores relativos a los segmentos que conforman la actividad deportiva.*	1
Nota 12	Como usuario, Quiero enviar un fichero a través de un formulario Para poder subir una actividad deportiva	*Acceso del usuario al sistema a través de un formulario donde puede enviar la actividad física en formato xml.*	4
Nota 11	Como usuario, Quiero ver los datos de la elevación relativos a la ruta en una grafica Para ver el desnivel del recorrido	*Mostrar el desnivel de la actividad física.*	3
Nota 2	Como usuario, Quiero ver los datos de mi velocidad a lo largo del recorrido en una grafica Para ver el nivel de esfuerzo de la actividad física	*Mostar la velocidad que ha llevado el usuario a lo largo de toda la actividad deportiva.*	3

Nota 3	Como usuario, Quiero ver los datos de mi frecuencia cardiaca en una gráfica Para ver el nivel de esfuerzo	*Mostar la frecuencia cardiaca del usuario que ha tenido a lo largo de la actividad deportiva.*	3
Nota 4	Como usuario, Quiero poder exportar la actividad física Para que sea utilizado en sistemas de terceros	*Posibilidad de exportar que tiene el sistema actual guardado en base de datos a formato xml.*	3
Nota 5	Como usuario, Quiero poder descargar el fichero original de la ruta Para poder restaurar las modificaciones que se han hecho	*Posibilidad de descargar el fichero que el usuario de la aplicación envió al inicio del uso del sistema, sin modificaciones.*	1
Nota 7	Como usuario, Quiero poder separar la actividad en diferentes segmentos Para poder diferenciar etapas	*Separación de un segmento de la trayectoria de la ruta en dos segmentos.*	4
Nota 6	Como usuario, Quiero ver la ruta que he subido en un mapa Para poder trabajar con ella fácilmente	*Visualizar la ruta deportiva, geoespacial en un mapa.*	5

Nota 10	Como usuario, Quiero poder ver valores calculados por el sistema Para completar valores no definidos en el fichero	*Calcular valores que no estén en el fichero originar como puede ser la velocidad en función del tiempo y la posición entre dos puntos.*	2
Nota 9	Como usuario, Quiero poder eliminar un punto en el mapa de la ruta Para corregir posibles errores de grabado	*Eliminar un punto de la ruta.*	2
Nota 8	Como usuario, Quiero poder ver un punto en las tres gráficas que ha sido seleccionado en el mapa, y viceversa Para poder ver los datos en ese punto	*Ver el punto que está seleccionado en una de las gráficas o en el mapa, en las otras gráficas o mapa. Que sea dinámica esta visualización.*	3
Nota 15	Como usuario, Quiero poder juntar dos segmentos contiguos de la actividad física Para ver estadísticas concretas de esta etapa	*Juntar dos segmentos contiguos.*	3
Nota 14	Como usuario, Quiero poder eliminar segmentos Para corregir errores que se produjeron al realizar la	*Eliminar un segmento de la ruta deportiva.*	2

	actividad deportiva		
Número total de Puntos Historia			**39**

Tabla 2

Una vez definidos todos los puntos asociados a cada historia de usuario en la Tabla 2, ya se puede comenzar a definir cuantas historias se desarrollarán en cada *Sprint*. Es lo que se realiza en la fase de *Spring planning*, definida más arriba en el punto número 1 de este mismo punto. Más o menos se ha acordado que cada iteración tenga 2 semanas de duración en los que se tendrán que completar todas las historias de la planificación de cada iteración.

Sprint	**Semanas**	**Historias de Usuario**	**Puntos de Historia**	**Puntos de Historia Restantes**
1	0-2	Nota 12, Nota 6	9	30
2	2-4	Nota 13, Nota 7, Nota 9	7	23
3	4-6	Nota 4, Nota 5, Nota 11	7	16
4	6-8	Nota 2, Nota 3, Nota 14	8	8
5	9-10	Nota 15, Nota 8, Nota 10	8	0

Tabla 3

Esta planificación se ha realizado suponiendo que van a trabajar en el desarrollo tres personas durante 8 horas diarias (5 horas efectivas) de lunes a viernes, se entrará más en detalle sobre todo esto en el siguiente punto identificado como *Estimación de costes*. Para ello, se han priorizado las historias que podrían aportar más valor al usuario con cada iteración generándose la Tabla 3.

El Gráfico 1, llamado gráfico *burndown*[34], muestra el trabajo que queda pendiente a lo largo de todo el proceso de creación del sistema permitiendo saber si el equipo podrá completar el trabajo en el tiempo estimado.

[34] https://en.wikipedia.org/wiki/Burn_down_chart

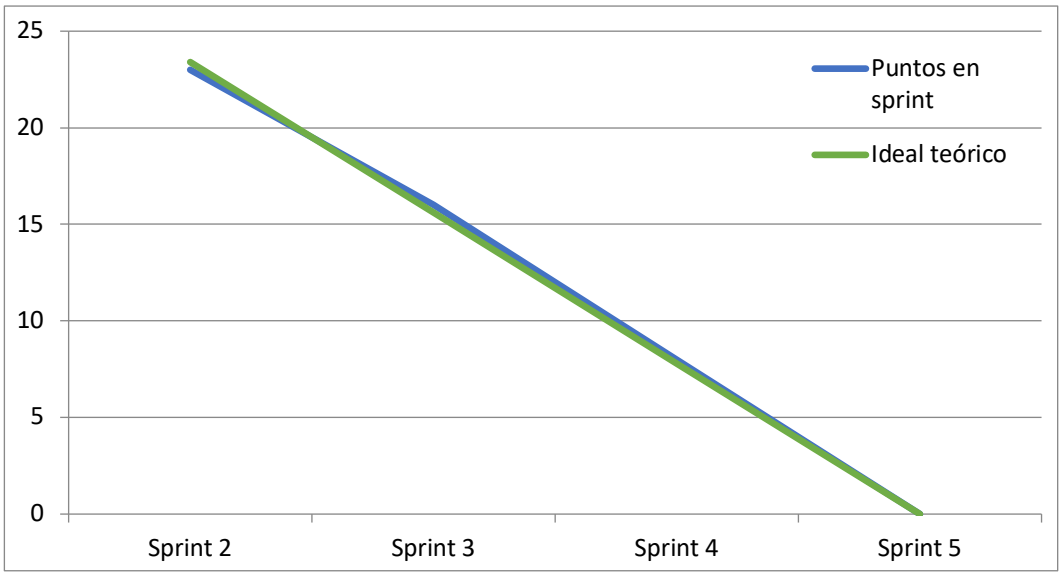


Gráfico 1

Este tipo de gráficos permiten similar diferentes escenarios en el caso de que, por ejemplo, se le añadan más historias al *Product Backlog*, más personas al equipo o se eliminen requisitos. Por lo tanto, con la estimación que hemos hecho, este gráfico nos muestra que la planificación se llevaría a cabo con éxito si se cumpliesen los plazos definidos. Por otro lado, al principio del proyecto, se puede ver que no se completan tantos puntos como en las demás iteraciones, se ha pensado así para permitir que el equipo se familiarice con el producto que hay desarrollar o modificar, además de sincronizarse con la forma de trabajar entre los diferentes integrantes.

Otro de los puntos que se han de realizar en la *Spring planning* es la identificación de las tareas técnicas que están asociadas a cada historia de usuario, haciendo que sean mas fáciles de manejar y que puedan dividirse entre un número determinado de personas del equipo. Es decir, facilitar la intervención de varias personas sobre una historia de usuario, por ejemplo, una persona que se dedique a hacer tareas de *backend*, otra de *frontend* y otra de *devops*, es decir, fomentar la división de tareas por función o tipo de rol. Por otro lado, la descomposición en tareas también facilita el seguimiento del Sprint para poder conocer en todo momento el avance del proceso. A continuación, en la Tabla 4 se van a descomponer cada historia de usuario en las tareas que tendrán que realizarse, entre paréntesis se identifica el rol al que pertenece cada tarea.

Una herramienta que sirve de gran utilidad, en la actualidad, para gestionar un proyecto ágil es Jira[35]. Esta aplicación permite administrar las tareas de un proyecto que están asociadas a cada historia de usuario o épica. Además, cada tarea puede ser dividida a su vez en tareas aun más pequeñas si eso ayudara a mejorar la comprensión y trazabilidad del proyecto.

[35] https://es.wikipedia.org/wiki/Jira

Historia de Usuario	Tareas Técnicas
Nota 13	▪ Crear servicio que obtenga datos de la actividad deportiva (*frontend*). ▪ Crear una tabla que contenga toda la información de la historia (*frontend*).
Nota 12	▪ Crear modelo de actividades deportivas (*backend*). ▪ Crear un lector de ficheros xml con esquem tcx o gpx (*backend*). ▪ Crear una base de datos (*devops*). ▪ Crear un *endpoint* que importe y almacene una actividad física (*backend*). ▪ Crear formulario para la subida de actividades deportivas (*frontend*).
Nota 11	▪ Visualizar datos de la elevación en una gráfica de área (*frontend*).
Nota 2	▪ Visualizar datos de la velocidad en una gráfica de líneas (*frontend*). ▪ Visualizar datos de la velocidad media con una línea (*frontend*).
Nota 3	▪ Visualizar los datos de la frecuencia cardiaca en una gráfica de barras (*frontend*). ▪ Visualizar datos de la frecuencia cardiaca media (*frontend*).
Nota 4	▪ Crear *endpoint* para exportar la actividad en formato tcx o gpx (*backend*). ▪ Crear botón para exportar actividad deportiva en tcx o gpx (*frontend*).
Nota 5	▪ Crear repositorio distribuido que permita almacenar ficheros originales (*devops*). ▪ Almacenar fichero original en el repositorio (*backend*). ▪ Crear botón para descargar fichero original (*frontend*).
Nota 7	▪ Adaptar el mapa de visualización de la ruta para permitir eliminar un punto (*frontend*). ▪ Crear *endpoint* para eliminar punto de la ruta deportiva (*backend*).
Nota 6	▪ Crear un mapa que visualice la ruta física (*frontend*).

	▪ Crear un *tooltip* [36] que de información del punto seleccionado en la ruta del mapa (frontend).
Nota 10	▪ Calcular los datos de la velocidad en función de la ubicación y el tiempo entre dos puntos (*backend*). ▪ Calcular los datos de la altitud usando un servicio de terceros (*backend*).
Nota 9	▪ Borrar un punto seleccionado en el mapa (*frontend*). ▪ Crear un *endpoint* para borrar un punto de la actividad física (*backend*).
Nota 8	▪ Visualizar el punto en el mapa relativo al punto seleccionado en alguna de las gráficas de la aplicación (*frontend*). ▪ Visualizar el punto en alguna de las gráficas que esta seleccionado en el mapa con el ratón (*frontend*). ▪ Visualizar el punto en las otras dos gráficas al mover el ratón sobre la otra restante (*frontend*).
Nota 15	▪ Seleccionar los dos segmentos a unir desde la tabla (*frontend*). ▪ Crear un *endpoint* para unir dos segmentos de la actividad (*backend*).
Nota 14	▪ Seleccionar los segmentos que se quieren eliminar (*frontend*). ▪ Crear un *endpoint* que permita el borrado de los segmentos seleccionados (*backend*).

Tabla 4

[36] https://es.wikipedia.org/wiki/Información_sobre_herramientas

Estimación de costes

Como es de esperar, todo proyecto software ha de tener una un documento donde se especifiquen los costes que tendrá la realización del producto. A lo lardo de la historia reciente, como se vio en el punto *Planificación de proyectos* del apartado *Aspectos Teóricos*, han surgido diferentes marcos de trabajo que enfocan dicha estimación de forma diferente. A continuación, se van a enumerar dos tipos diferentes y como influyen al proyecto en cuestión. Para realizar dicha tarea, se tendrá en cuenta lo que se ha descrito en este documento en los puntos anteriores identificados por el título *Especificación de Requisitos* e *Implementación* de este mismo apartado. Normalmente, todas estas estimaciones se elaboran en base a la experiencia previa y al conocimiento en cuestión del trabajo a implementar, por tanto, suele existir una cierta incertidumbre ya que ningún desarrollo software es exactamente igual a otro.

En las estimaciones de costes, en proyectos tradicionales, se han utilizado diferentes métodos, antiguamente se solía utilizar múltiples formas para analizar el coste de llevar a cabo un proyecto. Por ejemplo, por el número de líneas de código o por la cantidad de pantallas, informes o archivos que entregaba el software. Sin embargo, para la primera forma de hacer la estimación, la métrica era demasiado técnica lo que hacía que el cliente no pudiese comprender claramente el porqué de ese coste. Además, el hecho de que un software tenga más líneas de código no quiere decir que el producto ya tenga que ser más complicado o mejor que otro que tenga menos. Por lo tanto, hubo que buscar una forma de que el software fuera evaluado objetivamente. Es aquí donde nacen los puntos función, no confundir con los puntos de historia descritos en el punto anterior llamado *Implementación*, que permiten tener un conocimiento de la complejidad de lo que el equipo va a desarrollar. Los puntos función permiten resumir las funcionalidades de software de un requisito a un número gracias a la suma ponderada de diferentes cualidades, proporcionando una idea de qué compleja será su implementación. Existen diferentes implementaciones de los puntos función, una de las más usadas es la IFPUG[37] que define los siguientes puntos función no ajustados en función de la complejidad y el tipo:

Complejidad / **Tipo**	*Baja*	*Media*	*Alta*
Entrada Externa (EI)	3	4	6
Salida Externa (EO)	4	5	7

[37] International Function Point User Group: https://www.ifpug.org/

Consulta Externa (EQ)	3	4	6
Archivo Lógico Interno (ILF)	7	10	15
Archivo de Interfaz Externo (EIF)	5	7	10

Tabla 5

Los tipos se pueden agrupar en dos grandes subgrupos: de interacción y de almacenamiento. A continuación, se definen los tipos de la Tabla 5:

- Interacción con el usuario
    - La entrada externa (*EI*, External Input en inglés) es todo aquel componente que permite la interacción entre el usuario y el sistema. Por ejemplo, pantallas donde el usuario ingresa datos. En el sistema que se va a desarrollar, los formularios, los botones que permiten exportar la información o los botones en el mapa o en la tabla que permitirán editar la ruta.
    - La salida externa (*EO*, External Output en inglés) son aquellos componentes que producen valor agregado para el usuario de la aplicación. Por ejemplo: informes, gráficos y listado de datos. En el sistema a implementar, la tabla que tiene los datos agregados de cada segmento o los tres gráficos que visualizan la información de la velocidad, la altitud o la frecuencia cardiaca.
    - La consulta externa (*EQ*, External Query en inglés) es una característica de todo componente que permite visualizar información de archivos internos del sistema. Por ejemplo, la recuperación de datos y su visualización. En el sistema, serán componentes de consulta externa los que por ejemplo visualizan la ruta que ha llevado un atleta en la ruta deportiva y la visualización dé cada punto en el mapa en concreto.
- Almacenamiento de datos
    - Los archivos lógicos internos (*ILF*, Internal Logical Files en inglés) son archivos desde el punto de vista lógico que contienen información. Pueden ser tablas, documentos o otras estructuras de bases de datos. Por ejemplo, en la aplicación a desarrollar se utilizarán documentos para guardar toda la información en una base de datos *NoSQL*, además, también se guardarán documentos en un repositorio utilizando el servicio S3 que provee *AWS*.
    - Los archivos de interfaz externos (*EIF*, External Interface File en inglés) son datos provenientes y mantenidos por otros sistemas de terceros. Por ejemplo, esta aplicación se conectará a un sistema de Google para obtener el mapa donde se ubicarán los puntos a mostrar al usuario de la aplicación. También se usarán los servicios de Google para determinar la altitud en un punto especifico a partir de sus coordenadas espaciales de latitud y longitud.

Por lo tanto, se tienen que calcular para cada requisito funcional, descrito en el punto *Casos de uso* de la sección *Especificación de Requisitos* de este mismo apartado, los puntos función sin ajustar.

Caso de Uso	EI			EO			EQ			ILF			EIF		
	Núm	*P*	*C*	*Núm*	*P*	*C*	*Núm*	*P*	*C*	*Núm*	*P*	*C*	*Núm*	*P*	*C*
CU 1	1	3	B							2	10	M	1	7	M
CU 2							1	6	A				1	10	A
CU 3				*1*	4	B									
CU 4	2	4	M												
CU 5				3	5	M									
CU 6	2	3	B	2	4	B									

Tabla 6

En la Tabla 6 se resumen todos los puntos función por cada caso de uso de la aplicación a desarrollar. Por lo tanto, si se realiza la siguiente cuenta matemática:

*(1*3) + (2*10) + (1*7) + (1*6) + (1*10) + (1*6) + (1*4) + (2*4) + (3*5) + (2*3) + (2*4)*

Se obtienen 93 puntos función sin haber aplicado el ajuste.

A continuación, vamos a ajustar este resultado con las indicaciones que indica el IPFUG, organización definida en el pie del documento con el número 37. Para ello, se ha de calcular el factor de ajuste que viene definido por 14 características generales que se deben evaluar sobre el sistema que se va a desarrollar. En la Tabla 7 se muestra el factor de ajuste para este problema en concreto. En este enlace[38] se explican cada una de estas cualidades y sus valores posibles.

Característica	Valor	Puntuación
Comunicación de datos	La aplicación es más que una entrada on-line, y soporta apenas un protocolo de comunicación.	4
Procesamiento distribuido	Procesamiento distribuido y la transferencia de datos son on-line, en ambas direcciones.	4
Objetivos de rendimiento	Requisitos de rendimiento y de diseño fueron establecidos y previstos, sin embargo, ninguna acción especial fue requerida.	1
Configuración del equipamiento	Ninguna restricción operacional explícita o implícita fue incluida.	0

[38] https://docplayer.es/12389562-Determinacion-del-nivel-de-influencia.html

Tasa de transacciones	Se prevén picos diariamente.	3
Entrada de datos en línea	Más del 30% de las transacciones son entradas de datos on-line.	5
Interfaz con el usuario	El sistema posee: ▪ Scrolling vertical y horizontal. ▪ Selección de datos vía movimiento del cursor en la pantalla. ▪ Utilización del ratón. ▪ El menor número de pantallas posibles para ejecutar las funciones del negocio (*Single Page Application*). ▪ Utilización intensa de campos en video reverso, intensificados, subrayados, coloridos y otros indicadores. Tiene 5 elementos.	2
Actualizaciones en línea	Además de la actualización de la mayoría de los archivos lógicos internos, la protección contra pérdidas de datos es esencial y fue específicamente proyectado y codificado en el sistema.	3
Procesamiento complejo	Características presentes: ▪ Procesamiento lógico extensivo. (Varios algoritmos para modificar los valores de la ruta). ▪ Procesamiento complejo para manipular múltiples posibilidades de entrada/salida. (Controlar los diferentes tipos de esquemas xml: tcx o gpx) Tiene 2 características.	2
Reusabilidad del código	La aplicación fue específicamente proyectada y/o documentada para tener su código fácilmente reutilizable por otra aplicación y la aplicación es configurada por el usuario a nivel de código fuente.	4
Facilidad de implementación	Ninguna consideración especial fue establecida por el usuario y ningún procedimiento especial fue necesario en la implementación.	0

Facilidad de operación	Fueron desarrollados procedimientos de inicialización y respaldo, siendo necesaria la intervención del operador.	1
Instalaciones múltiples	Los requerimientos del usuario no consideran la necesidad de instalación de más de un local.	0
Facilidad de cambios	Están disponibles facilidades como consultas e informes flexibles para atender necesidades simples (contar 1 ítem).	1
	Factor de ajuste	**30**

Tabla 7

Una vez calculado el factor de ajuste según los niveles de influencia, se va a determinar cual es el valor de los puntos función total, con el ajuste ya realizado, de este proyecto siguiendo la fórmula:

$$PF = PFSA * (0{,}65 + (0{,}01 * \text{factor de ajuste})) = 93 * (0{,}65 + (0{,}01 * 30)) = 88{,}35 \cong 88$$

La cantidad de horas por hombre que se necesitan realizar para completar el proyecto a partir de los puntos función es difícil de valorar ya que no tiene en cuenta cosas como cuál es el lenguaje de programación que se va a utilizar o el nivel de experiencia de los trabajadores. Por lo tanto, la IFPUG ha desarrollado un equivalente para cada lenguaje de programación, de esta forma, lenguajes de bajo nivel tendrán más horas promedio por punto función que lenguajes más modernos. Por lo tanto, define que, para lenguajes de cuarta generación[39] se necesitan 8 horas por punto función o 20 líneas de código:

1 PF = 8 horas = 20 líneas de código (promedio)

Por tanto, se estiman 704 horas para completar todo el desarrollo del proyecto. Suponiendo que un trabajador tiene 5 horas productivas trabajando de lunes a viernes, se necesitan 140,8 días que se reparten entre 20 días que tiene un mes y saldría a 7 meses de duración del proyecto cuando trabaja en el desarrollo sólo una persona.

Si, por ejemplo, se incluyen tres trabajadores, como se especificó en el punto *Implementación* de este mismo apartado, tendríamos que se necesitan 234,67 horas paras completar el proyecto. Por lo tanto, con tres trabajadores se necesitarían 2,35 meses en los que trabajarían tres desarrolladores de lunes a viernes con 5 horas diarias de rendimiento. En la Tabla 8 se resumen todos estos cálculos matemáticos.

1 trabajador:

*88 * 8 = 704 horas*

704 horas / 1 trabajador / (5 horas / día / trabajador) = 140,8 días

140,8 días / 20 días / mes = 7,04 meses ≅ 7 meses

3 trabajadores:

704 horas / 3 trabajadores = 234,67 horas

[39] https://en.wikipedia.org/wiki/Fourth-generation_programming_language

234 horas / (5 horas /día / trabajador) = 46,93 días

46.93 días / (20 días / mes) = 2,35 meses ≅ 2 meses y medio

Tabla 8

En la Tabla 9 se incluyen más casos, con más trabajadores, además de los expuestos anteriormente, para ver como cambia la estimación de la duración del proyecto en función del número de trabajadores.

PF	Trabajadores	Horas promedio / PF [40]	Horas	Días	Meses proyecto
88	1	8	5	20	7
88	3	8	5	20	2,35
88	5	8	5	20	1,40
88	7	8	5	20	1

Tabla 9

Suponiendo que se han contratado a 3 trabajadores a cargo de la finalización del proyecto, tendremos que:

- Coste mensual de un desarrollador para la empresa: 1.433,33 € / mes
  - Salario neto: 1.201,98 € / mes
  - Cuota Seguridad Social (6,35%): 91,02 € / mes
  - I.R.P.F (9,86%): 141,33 € / mes
- Otros gastos como:
  - Pagar la luz: 140 € / mes.
  - Proveedores de internet: 80 € / mes.
  - Equipo (MacBook Pro 13”): 1.499,00 € / trabajador.

Si se aplica esta fórmula:

*Costes = (Número de desarrolladores * Duración de meses * sueldos) + Otros gastos*

*= (3 * 2,35 * 1.433,33) + ((140 * 2,35) + (80 * 2,35) + (3*1.499)) = 15.118,98 euros*

Se obtiene una estimación de 15.118,97 € para completar el proyecto con tres trabajadores y un marco de trabajo tradicional o en cascada.

Como se comentó en los puntos de *Especificación de Requisitos* y de *Implementación* de este mismo apartado, en la actualidad, es más común que las empresas adopten metodologías ágiles, así pues, este cambio de enfoque también afecta en el coste del proyecto y en la facilidad para empezar a desarrollarlo. Por tanto, será diferente el coste que la empresa tiene que afrontar ya que con cada iteración se realizarán pequeñas entregas totalmente funcionales. A diferencia del modelo tradicional que sigue la metodología de entregar el producto tras finalizar todas las etapas de análisis, diseño, desarrollo, pruebas e implementación.

[40] Horas promedio por punto función para lenguajes de cuarta generación

A la hora de realizar un presupuesto siguiendo una metodología ágil, cada entregable producirá valor al cliente y supondrá un beneficio para su economía, en el mejor de los casos. Es decir, se va a definir un coste por cada entregable producido, en lugar de cobrar por las horas trabajadas en el proyecto. Es por esto por lo que, se sigue un modelo de "*no money for nothing*", ya que cada vez que el cliente realice un pago, el equipo le proveerá de una entrega que implicará una rentabilidad económica. El cliente, al no estar ante una metodología rígida como son las tradicionales, podrá acogerse al lema de marketing "*change for free*" que posibilita en cualquier momento el cambio de una historia por otra o de finalizar el proyecto tras la superación de una determinada iteración.

Llegados a este punto, se va a tomar como hipótesis que la aplicación que se entrega tiene un modulo transversal de publicidad, desarrollador por una tercera empresa o el propio cliente, que genera una rentabilidad por cada clic que se realiza sobre algún anuncio. Por ejemplo, se genera 0.22 € por clic realizado, por lo que, a mayor número de usuarios, en general, habrá un mayor beneficio. Es decir, según se vaya incrementando la funcionalidad de la aplicación, el número de usuarios tenderá a crecer más y a que se produzcan más clics. Por otro lado, se van a definir los mismos *Sprints* que se definieron en el punto *Implementación* de este mismo apartado.

En la Tabla 10 se define los valores (coste por clic, número de iteraciones o Sprints, el modelo de ordenador, el valor y cuanto costará por Sprint) que tenemos como punto de partida:

CPC	Núm. Sprints	Trabajadores	Equipo	Valor/ Equipo	Equipo / Sprint
0.22 €	*5*	*3*	*MacBook Pro 13"*	*-1,499.00 €*	*-299.80 €*

Tabla 10

En la Tabla 11 se definen los costes fijos de cada mes (salario por cada trabajador, los gastos de la luz y de internet) con su valor asociado para cada iteración:

Salario Mes / Trabajador	Salario / Sprint
-1.433,33 €	*-1,800.00 €*
Luz / Mes	**Luz / Sprint**
-180.00 €	*-90.00 €*
Internet / Mes	**Internet / Sprint**
-80.00 €	*-40.00 €*
	Ordenador / Sprint / Total Trabajadores
	-899.40 €

Tabla 11

La Tabla 12 representa el gasto del proyecto para cada Sprint además del beneficio obtenido con cada entrega. En función de los gastos y los beneficios se obtiene el total de

beneficio que se produce con cada iteración. Se observa que, siguiendo este marco de trabajo, a partir del Sprint 4 la ganancia ya es positiva lo que hace que el proyecto ya comience a ser rentable a partir de este momento.

Sprint	Gastos	Clics Anuncios	Beneficio	Total
Semanas 0-2	-3.179,40 €	0	0,00 €	-3.179,40 €
Semanas 2-4	-6.358,79 €	8000	1.760,00 €	-4.598,79 €
Semanas 4-*6*	-9.538,19 €	23000	5.060,00 €	-4.478,19 €
Semanas 6-8	-12.717,58 €	55000	12.100,00 €	-617,58 €
Semanas 8-10	-15.896,98 €	108000	23.760,00 €	7.863,03 €
Semanas 10-12	-15.896,98 €	150000	33.000,00 €	17.103,03 €

Tabla 12

En cambio, para las estimaciones de costes que se ha hecho en este apartado, contando con una duración de dos meses y medio, como se definió antes, tendremos que el beneficio no será tangible hasta la finalización de la semana 10 del proyecto y aun así la rentabilidad del proyecto no será favorable. Siempre y cuando, no se haya producido retraso en la entrega, ni tampoco problemas a la hora de desplegar el producto o errores detectados una vez que el sistema ya está en producción. Es decir, estaríamos siendo optimistas con que todo haya ido como lo estipulado y esperado. Se ha hecho la hipótesis de que, una vez que se haya desplegado en producción, se conseguirán la media de todos los clics producidos a lo largo de todas las estregas de la Tabla 12.

En la Tabla 13 se muestran los valores (coste total, duración en meses y los gastos por cada semana) para la estimación que hemos hecho más arriba usando la técnica de Puntos Función.

Coste Total	Duración (meses)	Gatos / semanas
-15.118,97€	*2.35*	*-1,433.40 €*

Tabla 13

Y la Tabla 14 que contiene la inversión y el beneficio obtenido:

Semana	Gastos	Clics / Anuncio	Beneficio	Total
Semanas 0-2	-3.216,80 €	0	0,00 €	-3.216,80 €
Semanas 2-4	-6.433,60 €	0	0,00 €	-6.433,60 €
Semanas 4-*6*	-9.650,41 €	0	0,00 €	-9.650,41 €
Semanas 6-8	-12.867,21 €	0	0,00 €	-12.867,21 €

Semanas 8-10	-16.084,01 €	0	0,00 €	-16.084,01 €
Semanas 10-12	-16.084,01 €	57333,33333	12.613,33 €	-3.470,68 €

Tabla 14

En el Gráfico 2 se visualiza el beneficio o gasto para cada iteración, en función de si el valor es positivo o negativo, para cada estimación siguiendo un modelo ágil o uno tradicional. Es fácil apreciar gráficamente que la implementación con una metodología ágil producirá valor para el usuario mucho antes de lo que lo hará el mismo proyecto implementado con un marco de trabajo clásico. Ambos proyectos finalizarán al tercer mes, pero el tradicional tan solo tendrá unas perdidas de -3,470.68 €, mientras que, el mismo proyecto implementado con *Scrum* habrá generado 17.103,03 € de beneficio.

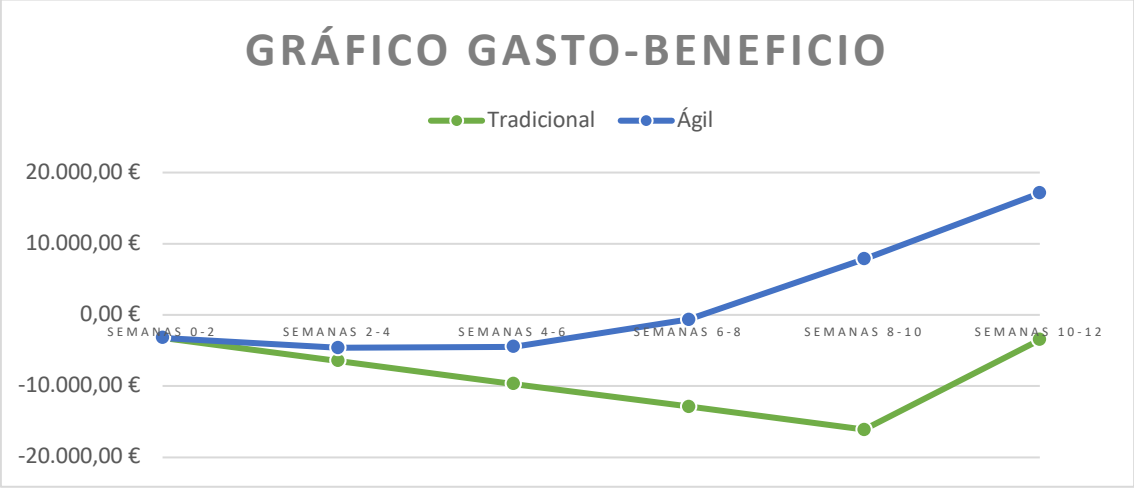


Gráfico 2

Aspectos Teóricos

Planificación de proyectos

La planificación de proyectos forma parte de la gestión de proyectos, la cual se vale de cronogramas tales como diagramas de Gantt para del progreso dentro del entorno del proyecto. Es el proceso para cuantificar el tiempo y recursos que un proyecto costará. La finalidad del planteamiento de proyecto es crear un plan de proyecto que un gestor (gestor de proyecto) pueda usar para acompañar el progreso de su equipo. (Wikipedia)

Tipos

Existen diferentes criterios en cuanto a qué tipo de proceso escoger a la hora de realizar la planificación de un proyecto software. Lo primero de todo, definir qué es exactamente eso de la planificación en el mundo software y las metodologías que hay. Básicamente la planificación es todo proceso de gestión para la creación o evolución de un sistema en el cual se suceden diferentes actividades. Por lo tanto, todas definen una metodología de desarrollo[41], es decir, un marco de trabajo usado para que la persona encargada de gestionar todo el proceso pueda estructurar, planificar y controlar el desarrollo de sistemas de la información.

Cascada

Este tipo de planificación se llama cascada[42] o *waterfall*, en inglés, y se caracteriza por tener un flujo secuencial entre sus diferentes fases. Su nombre hace referencia a que en este proceso se ve todo el flujo como desde abajo de una cascada, es decir, parece que cada paso cae "en cascada" haciéndose a continuación la siguiente fase. Las principales fases de este tipo son las típicas de un proyecto software: el análisis, el diseño, el desarrollo, las pruebas, el despliegue y el mantenimiento.

Las principales ventajas de este sistema son su facilidad de entender e implementar, se fundamenta en hacer una buena documentación, tiene un buen resultado en equipos débiles y producto maduros y, por último, es muy conocido en el mundo software y produce un marco de trabajo que se podría decir que es de sentido común: finalizar algo antes de comenzar con el siguiente paso.

Sin embargo, también existen diferentes desventajas como que, en la vida real, en lo que es la práctica, los proyectos no siguen un modelo secuencial por lo que no se pueden adaptar a cambios lo cual hace que muchas veces los procesos fracasen. Los procesos tardan mucho tiempo en salir a "producción", a ser mostrados al cliente, y todo se debe a que el proceso no puede finalizar la fase de pruebas hasta que todas las demás fases hayan finalizado. Por otro lado, cualquier error detectado en las pruebas implica un rediseño y las demás operaciones pertinentes hasta que vuelva a ser validado y se pueda llevar a "producción" y, por último, cualquier bloqueo en alguna de las etapas produce retrasos en completar las fases siguientes.

[41] https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software
[42] https://es.wikipedia.org/wiki/Desarrollo_en_cascada

Prototipos

Otro tipo es el modelo de prototipos[43] que pertenece al tipo de proceso de desarrollo evolutivo siendo el principal objetivo que el producto se construya en el menor tiempo posible, usando los recursos adecuados y gastando los menores recursos posibles. Las diferentes etapas para este proceso son: comunicación, planificación y diseño rápido, construcción del prototipo, desarrollo, entrega y retroalimentación y volver a empezar desde el principio. Al finalizar este proceso, una vez que se hayan logrado los requisitos de los interesados, el producto habrá finalizado.

A continuación, se van a definir las ventajas del uso de este marco de trabajo. Es eficaz cuando el cliente conoce los objetivos generales, pero no identifica o no conoce los detalles. Involucra más al cliente en el proceso de la creación del producto permitiendo que el cliente tenga una idea de cómo será el software una vez que el proyecto haya finalizado. Ofrece un mejor enfoque cuando el responsable del producto no esta seguro de las características del software como la fiabilidad de un algoritmo, la adaptabilidad de un sistema operativo o de la experiencia de usuario-máquina.

Por otro lado, el marco de trabajo también tiene algunas desventajas implícitas como que el cliente crea una imagen del producto final que usualmente no cumple ciertos aspectos importantes como la calidad, escalabilidad y mantenimiento a largo plazo lo que puede promover un replanteamiento del sistema que se diseñó que en ocasiones hace que el cliente se niegue a rediseñarlo y que finalmente se construya el proyecto final con el diseño inicial que no cumple todos los aspecto que debería. Por otro lado, la rapidez del desarrollo del prototipo (por ejemplo, elecciones transversales al proyecto como qué lenguaje de programación usar, qué base de datos elegir en un primer prototipo, …) puede dar lugar a decisiones erróneas que puedan ser arrastradas hasta la creación del producto final.

Incremental

El modelo incremental o iterativo-creciente[44] persigue suplir las carencias que presenta el modelo en cascada o secuencial. Se basa fundamentalmente en una serie de tareas agrupadas que se denominan iteraciones y es en lo que se basan la mayoría de las metodologías modernas.

El modelo consta de diferentes etapas de desarrollo software las cuales se inician con la planificación y finalizan con el despliegue de la aplicación. Cada requisito se ha de desarrollar en una iteración que debe incluir las fases de diseño, desarrollo e implementación y de generar toda la documentación entregada para que no quede nada pendiente a la hora de entregar el proyecto finalizado al cliente. Lo que se persigue es que en cada iteración el proyecto evolucione siempre dependiendo de las iteraciones anteriores y logrando una mayor funcionalidad de la aplicación con cada una de ellas. Una forma de priorizar los elementos que han de ser desarrollados en cada iteración es que se tenga en cuenta el valor que aporta a la experiencia de usuario cada uno de ellos.

Las ventajas que ofrece este modelo de trabajo es que los usuarios no tienen que esperar a que se finalice el proyecto para hacer uso del sistema ya que puede utilizarse en cada

[43] https://es.wikipedia.org/wiki/Modelo_de_prototipos
[44] https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

fase final de iteración. Con las principales versiones del sistema se puede obtener *feedback* del cliente para incluir esos nuevos requisitos en las demás iteraciones. Además, existe poco riesgo en que el proceso no finalice con éxito, por otro lado, la probabilidad de encontrar errores en las partes fundamentales del software es baja ya que las pruebas se encargarán de validar aquellas partes cruciales del software. Se da retroalimentación a los usuarios rápidamente y permite separar su complejidad en cada iteración lo que dará forma a un producto consistente y con una menor probabilidad de fallo. Por otro lado, en lo que a los desarrolladores respecta, obtendrán aprendizaje con cada iteración para el desarrollo posterior de otras iteraciones o de otros proyectos.

Las principales desventajas de esta metodología es que la entrega temprana después de las primeras iteraciones puede dar lugar a que los clientes o usuarios tengan la sensación de que el proyecto entregado es demasiado simple. Además, es necesario la disponibilidad de un cliente a lo largo de todo el desarrollo de software, siendo no siempre fácil que el cliente dedique el tiempo necesario en todo este proceso. Por otro lado, y a colación de esta desventaja, la relación entre el cliente y los demás interesados en el desarrollo del producto ha de ser de colaboración más que trabajar cada uno por su lado, dejando a un lado sus propios intereses. Puede que las primeras versiones no sean robustas como para ponerlas al alcance de todos los usuarios ya que podrían generar un riesgo seguridad, es decir, de ser potencialmente vulnerables y que finalmente sufran ataques cibernéticos.

Espiral

Otro es el desarrollo en espiral[45] es un modelo de trabajo en el que el flujo de actividades sigue una espiral en la que cada bucle representa un conjunto de actividades. La prioridad de las actividades no está definida con ningún valor, sino que se empieza por el inicio del bucle que sigue las diferentes etapas del desarrollo software, comentadas más arriba seguidas también por el modelo en Cascada. En lo que sí se hace hincapié es en evaluar los riesgos[46] que existen a la hora de desarrollar el software, es decir, se analizan las posibles alternativas que existen, se elige la que menor riesgo supone y se hace un ciclo de la espiral y así en cada vuelta de la espiral.

Las ventajas que presenta este modelo son que al realizarse un análisis de riesgos se reduce el riesgo del proyecto, incorpora cualidades que favorecen la calidad del producto, integra el desarrollo con el mantenimiento, el ciclo de vida no es rígido ni estático como sucede en el modelo secuencial lo que añade la posibilidad de incluir mejoras y nuevos requisitos en el futuro al proyecto uniendo las buenas prácticas de los modelos iterativos y secuenciales.

Sin embargo, al igual que los anteriores, presenta algunas desventajas como que el modelo es muy costo de llevar a cabo, genera mucho tiempo en el desarrollo del sistema y requiere una alta experiencia en el análisis de posibles riesgos y alternativas.

RAD

Por último, esta metodología tiene sus siglas producidas a partir del inglés, en español su traducción sería desarrollo rápido de aplicaciones e integra o mezcla dos de las metodologías anteriormente mencionadas: el desarrollo iterativo o incremental y el

[45] https://es.wikipedia.org/wiki/Desarrollo_en_espiral
[46] https://es.wikipedia.org/wiki/An%C3%A1lisis_de_riesgo

desarrollo guiado por prototipos. Por lo tanto, mezcla la rapidez de desarrollo de los prototipos con la entrega de diferentes versiones iniciales del sistema a realizar.

Las ventajas que tiene es que su implementación no implica un gran coste, se producen sistemas de gran calidad para el usuario o que dan gran valor al cliente, tienen su enfoque en la necesidad comercial en lugar de las buenas tecnologías a usar. Sin embargo, también se necesita que los usuarios estén presentes a lo largo de toda la fase de creación del software con ocurría en los modelos iterativos.

Popularidad

En los últimos tiempos, las empresas han adaptado en su día a día cada vez más metodologías ágiles porque se adaptan mejor a las diferentes situaciones del mundo real. Se enfocan a obtener de forma rápida resultados que satisfagan al cliente, prescindiendo de estructuras de organización clásicas en favor de estructura más plana que involucren a todos sus integrantes en el proyecto.

El enfoque tradicional y el ágil trata de forma diferente algunos aspectos cruciales en el desarrollo de software:

- Todos los responsables del equipo tratan de controlar la incertidumbre del proyecto para evitar contratiempos que puedan tener lugar en el futuro. En las metodologías tradicionales se intenta analizan las posibles incertidumbres inherentes al proyecto pudiendo tener que lidiar con problemas a lo largo de todo el ciclo de vida del proyecto. Sin embargo, en metodologías ágiles se propone un modelo más adaptativo respecto a la incertidumbre, contemplando la refactorización del proyecto sobre lo existente en cualquier momento del ciclo de vida del proyecto. En lugar de tener contratos estrictos con mucha documentación, se da más valor a la comunicación con el cliente a lo largo de todas las fases del proyecto para detectar y solucionar errores cuanto antes.
- En las metodologías tradicionales se cree que todos los cambios han sido contemplados en la fase de análisis del proyecto, sin embargo, siempre existen nuevos requisitos a lo largo de todo el proceso de desarrollo software. Además de esto, la baja calidad o la inexistente comunicación con el cliente hacen que surjan problemas a la hora de entregarle el producto ya finalizado. Por otro lado, los procesos ágiles contemplan el cambio como algo habitual en todo el proceso de desarrollo del proyecto sirviendo como herramienta para el aprendizaje continuo en el proyecto en curso y en futuros retos. Como se ha comentado en estas metodologías la continua relación con el cliente hacen que los cambios que puedan surgir se detecten cuanto antes y, por tanto, que el proyecto entregado cumpla todos los requisitos esperados.
- Otra de las diferencias entre los dos modelos es la forma de gestionar los equipos, en las metodologías ágiles lo más importante del proyecto son los recursos humanos que forma el equipo. Por otro lado, la forma de trabajar en este tipo de metodologías es diferente a las tradicionales ya que para el cliente representa una figura propia dentro del equipo, es decir, es uno más. En las metodologías tradicionales se intenta tener el control del equipo mientras que en las ágiles el equipo se gestiona y controla él mismo. También se fomenta el bien del equipo y no solo el interés y la competencia individual como sí están enfocadas las metodologías tradicionales. Cometer errores y fallos no está mal visto, sino que lo importante es identificar y corregir todo contratiempo que pueda surgir cuanto

antes. Los equipos, a diferencia de las estructuras tradicionales, son multidisciplinares y planos.

- Por último, la dirección de un proyecto en un modelo ágil es muy diferente, teniendo un marco de actuación que es flexible y adaptativo al entorno o al contexto en lugar de ceñirse al proceso como ocurre en los modelos tradicionales. Es decir, en los modelos ágiles la dirección del proceso esta fundamentada en las necesidades del cliente o del negocio en lugar de la visión propia de la dirección del negocio.

Ficheros de actividades deportivas

Existen multitud de formatos que permiten el intercambio de información relacionada con una actividad deportiva. Todos los modelos que se van a describir se encuentran disponibles en los ficheros adjuntos de la memoria, en el punto *Anexo I* se explica cómo encontrar cada documento. A continuación, se van a analizar algunos de los esquemas existentes en el mercado.

GPX

Estas iniciales proceden del inglés *GPS eXchange Format*[47] que en español sería algo así como formato de intercambio GPS. Es un tipo de fichero *xml* pensando para transferir datos GPS entre aplicaciones pudiéndolo usar para describir puntos (*waypoints*), recorridos (*tracks*) y/o rutas (*routes*) (Wikipedia). Tiene un tipo MIME[48] especifico identificado como **`application/gpx+xml`** aunque también acepta el tipo genérico **`application/octet-stream`**.

Su primera versión se dio a conocer en el 2002, posteriormente, en agosto de 2004 se liberó la versión 1.1 que supone la última versión lanzada al mercado. Anteriormente también se había lanzado la versión 1.0.1, sin embargo, pronto se supo que esta versión bloqueaba ciertos atributos de terceras aplicaciones por lo que se trató de lanzar rápidamente la última versión que hay en la actualidad.

De todas las versiones descritas en este punto, el sistema que se presenta en este documento tan solo usará la versión más reciente del esquema *xml*. En este enlace[49] se puede encontrar más información de cómo está definida la estructura de este tipo de archivos. Como se ha comentado al inicio de este punto, este tipo de ficheros permiten definir tres tipos de elementos:

- Los *wptType* que representan un punto en el mapa mediante el sistema WGS84[50] de coordenadas, ofreciendo la posibilidad de incluir más información descriptiva relativa a otro tipo de información.
- Los *rteType* que representan un conjunto de puntos alineados, es decir, contienen una secuencia de los elementos descritos en el primer punto de este listado. Este tipo de elementos sirven para la creación de rutas, es decir, un usuario no tiene por qué haber realizado la ruta especificada en este elemento.
- Los *trkType* que representan la trayectoria realizada por un usuario, estando formada por una serie de puntos identificados por elementos descritos en el primer punto de este listado.

Este tipo de esquemas también tienen la posibilidad de ser extendidos mediante otro tipo de ficheros. Para ello, se hace uso de una extensión llamada *TrackPointExtension*[51] creada por la compañía de la que se habló en el punto *Garmin* del apartado *Alternativas de mercado*. Este fichero permite incrementar los tipos de campos que se pueden guardar en el documento, posibilitando guardar información que no tiene en su descripción los ficheros con formato *gpx11*. Por lo tanto, se podrá almacenar información relativa a la

[47] https://www.topografix.com/gpx.asp
[48] https://en.wikipedia.org/wiki/Media_type
[49] http://www.topografix.com/GPX/1/1/
[50] https://en.wikipedia.org/wiki/World_Geodetic_System#A_new_World_Geodetic_System:_WGS_84
[51] https://www8.garmin.com/xmlschemas/TrackPointExtensionv1.xsd

temperatura que hace en el ambiente, la frecuencia cardiaca obtenida en cada punto de la trayectoria, el valor de la cadencia o la profundidad en cada punto.

TCX

Sus iniciales corresponden con el término en inglés de *Training Center XML*[52], lo que en español se puede traducir como centro de entrenamiento *xml*. Mayoritariamente este esquema es utilizado desde su lanzamiento en 2007 para el intercambio de datos entre dispositivos de la compañía que lo desarrolló, la que ha sido descrita en el punto *Garmin,* que lleva su nombre, del apartado *Introducción.* Al igual que ocurría en este punto anterior, también tiene un tipo MIME específico llamado **`application/vnd.garmin.tcx+xml`** siendo igualmente válido el valor **`application/octet-stream`**.

Teniendo una estructura parecida a la del esquema del punto anterior, identificado como *GPX*, y pese a que también es utilizado como un estándar para la transferencia de información de posicionamiento, su principal diferencia es que trata a la actividad como algo más que puntos geoespaciales, posibilitando incluir información relativa a la frecuencia cardiaca, la cadencia y las calorías, además de incluir datos agregados para cada segmento de la actividad.

Por lo tanto, la estructura de este esquema es mucho más amplia y abstracta permitiendo definir diferentes tipos de actividades. El elemento general donde se define los demás es el que se identifica como *TrainingCenterDatabase_t* y es aquí desde donde definir las diferentes actividades físicas o deportes que se hayan realizado. A continuación, se muestran algunos de los elementos que contiene el tipo de estructura comentado en este mismo párrafo:

- El tipo de elemento identificado como *Folders_t* que permite guardar identificadores de histórico de actividades realizadas, de tipos específicos de entrenamientos (bici, correr, etc.) y de curos.
- Los elementos de tipo *ActivityList_t* que permiten guardar:
    - Elementos del tipo *Activity_t* que contienen toda la información sobre una actividad física siendo éste el que se utiliza en el sistema que documenta esta memoria. Por lo tanto, este tipo almacena información propia de la actividad física como por ejemplo el tipo de entrenamiento, identificado por el tipo *Training_t*, que se ha realizado, almacenará todas las vueltas, identificadas por el tipo *ActivityLap_t*, y todos sus puntos geoespaciales relativos identificados por el tipo *Track_t*.
    - Elementos del tipo *MultiSportSession_t* que identifica al primer deporte realizado, identificado por el tipo *FirstSport_t del tipo Activity_t* descrito en el punto de arriba*,* y a los siguientes del tipo *NextSport_t* que tendrán una transición opcional y el deporte en sí también del tipo descrito más arriba e identificado como *Activity_t*.
- Elementos de tipo *WorkoutList_t* que contendrán una secuencia de elementos de tipo *Workout_t* y que a su vez contendrá un paso identificado como *AbstractStep_t* y que podrá tener como implementación *Step_t* o *Repeat_t*.

[52] https://en.wikipedia.org/wiki/Training_Center_XML

- Elementos de tipo *CourseList_t* que contendrán elementos del tipo *Course_t* teniendo la posibilidad de definir aquí las diferentes vueltas *CourseLap_t*, las diferentes pistas *Track_t* o los puntos del curso *CoursePoint_t* que lo conforman. El sistema descrito en esta documentación también está disponible para manejar información proveniente de este parte del esquema del fichero *tcx*.

Como ocurría con el anterior esquema descrito en este punto, para extender la funcionalidad o la capacidad del esquema para guardar información se hace uso de otro documento que en este caso sí ha sido creado por la misma compañía. Este esquema es la versión 2 del esquema llamado *ActivityExtension*[53] definiendo dos nuevos elementos:

- Un elemento identificado con el nombre *LX* y que tiene como tipo *ActivityLapExtension_t* permitiendo añadir información adicional a una vuelta como la velocidad media, la máxima cadencia de bicicleta/carrera, los pasos, la potencia media y máxima.
- Un elemento llamado *TPX* del tipo *ActivityTrackpointExtension_t* que permite añadir información a una pista o punto de la actividad como la velocidad, la cadencia de la carrera y/o la potencia.

[53] https://www8.garmin.com/xmlschemas/ActivityExtensionv2.xsd

Tecnología

A continuación, se va a enumerar las siguientes tecnologías utilizadas para la implementación de toda la funcionalidad del sistema. Todas las características de desarrollo serán detalladas más adelante en el apartado *Implementación* (no confundir con el punto del apartado de *Planificación*).

Backend

En la capa donde se realiza todo el procesamiento de datos, también llamada aplicación del lado del servidor se ha escogido como lenguaje de programación Java. Se usan, además, otras tecnologías como: Maven[54], Spring[55] o Mongo[56], entre otras, que serán detallas más en profundidad en los próximos puntos.

Java

Se ha escogido este lenguaje de programación por los siguientes motivos:

- Es orientado a objetos, lo que permite que el código sea
    - Reusable ya que permite que las clases se pueden usar en distintas partes del proyecto.
    - Mantenible ya que el código es más sencillo de leer debido a la capacidad de abstracción del dominio del problema.
    - Modificable ya que fácilmente se pueden añadir, suprimir o modificar nuevos objetos.
    - Extensible y flexible posibilitando añadir nueva funcionalidad a clases presentes en librerías ya existentes.
    - Fiable ya que es fácil testear el código de forma unitaria gracias a que el problema normalmente es dividido en componentes más pequeños.
- Es multiplataforma porque el código es compilado y ejecutado por la máquina virtual de Java (JVM) que es compatible con la mayoría de los sistemas operativos actuales como Windows, Mac o Linux. Uno de los principios con los que fue concebido en sus orígenes fue con el leitmotiv "*Write Once, Run Anywhere*".
- Es gratis, basta con descargar el JDK desde la página web de Oracle. Actualmente, existen diferentes versiones de Java, desde la versión 1, lanzada en 1996, hasta la versión 14, lanzada en abril de 2020. A partir de Java 9, la compañía decidió sacar versiones nuevas más frecuentemente, pero con menos funcionalidades.
- Es abierto, hay una extensa biblioteca de librerías abiertas, por lo tanto, el código fuente es público.
- Popularidad ya que es utilizado para desarrollar gran cantidad de programas en todo el mundo. Como se puede observar en el sitio web de la compañía Tiobe[57], desde el 2015 y hasta mayo de 2020 había sido el más popular a nivel mundial. Sin embargo, en junio de 2020 el lenguaje C le arrebató el primer puesto.

[54] https://maven.apache.org/
[55] https://es.wikipedia.org/wiki/Spring_Framework
[56] https://www.mongodb.com/
[57] https://www.tiobe.com/tiobe-index/

- Es potente, la máquina virtual de Java gestiona mejor ciertos aspectos que no estaban pulidos en lenguajes anteriores como C. Por ejemplo, el recolector de basura es totalmente transparente para el programador.

En 2014 la compañía Oracle, propietaria del lenguaje de programación tras la adquisición de la empresa Sun Microsystems en 2010, libera la octava versión en la que se incluyen entre otras cosas: conceptos de la programación funcional incluyendo la posibilidad de utilizar funciones lambda.

Un ejemplo práctico de la aplicación de la programación funcional es el uso de las interfaces funcionales genéricas, definidas en la librería nativa de Java:

- *Function<T, R>*
- *Predicate<T>*
- *Consumer<T>*
- *Supplier<T>*

La interfaz *Function<T, R>* define el método *apply* que recibe como parámetro un objeto de la clase T y genera como salida una instancia de la clase R. A continuación, se expone un caso práctico del uso de esta interfaz. El objeto *mapPosition* es una instancia de la clase *Function*, recibiendo un objeto de tipo Position y devolviendo un elemento equivalente pero correspondiente a la clase que modela un punto geoespacial del *xml* con un esquema *tcx*, este fichero contiene la información de una actividad deportiva:

```
120     private Function<Position, PositionT> mapPosition = position -> {
121         PositionT positionT = positionTSupplier.get();
122         positionT.setLatitudeDegrees(position.getLatitudeDegrees().doubleValue());
123         positionT.setLongitudeDegrees(position.getLongitudeDegrees().doubleValue());
124         return positionT;
125     };
```

Por otro lado, se ha incluido la interfaz *Predicate<T>* que define el método *test* y que tiene como parámetro de entrada un objeto de tipo T y devuelve el valor booleano verdadero o falso. Por ejemplo, el método *nonHasAltitudeValues* comprueba que las pistas de una vuelta no tengan el campo altitud con algún valor especificado:

```
91      Predicate<Lap> nonHasAltitudeValues = (lapParam) -> lapTrackPointValueHasCondition(
92                  lapParam, TrackPoint::getAltitudeMeters, Objects::isNull);
```

Para ello, se hace uso de un método *lapTrackPointValueHasCondition* que recibe una vuelta en concreto, siendo un objeto de tipo *Function* para obtener el valor de la pista con la que aplicar la condición y, por último, la condición en sí. Esto permite que se generalice para cualquier campo de una pista y para cualquier condición. El ejemplo anterior se utiliza concretamente para el campo altitud y cuando esta altitud no está presente:

```
481     private boolean lapTrackPointValueHasCondition(final Lap lap,
482                 final Function<TrackPoint, Object> function,
483                 final Predicate<Object> condition) {
484                 return getTrackPoints(lap).stream().map(function).allMatch(condition);
485     }
```

Snippet 1

Como se puede ver en el último ejemplo, gracias a estas interfaces, se puede llamar a métodos con parámetros del tipo de funciones lambda.

Otra de las interfaces funcionales introducidas en esta nueva versión es *Consumer<T>*, definiendo el método *accept* que, a diferencia de las anteriores, los objetos de esta clase no devuelven ningún valor y aceptan un parámetro del tipo T. Por lo que, haciendo un

equivalente con los métodos tradicionales, el valor de retorno en este caso es vacío, representado con la palabra reservada *void*.

Volviendo al código de la aplicación, un ejemplo práctico sería el método *resetHeartRateValues* que borra el pulso cardíaco de un punto geoespacial de la trayectoria:

```
248          Consumer<TrackPoint> resetHeartRateValues = trackPoint ->
249                  trackPoint.setHeartRateBpm(null);
```

Por último, la interfaz *Supplier<T>* que define el método *get* y cuya principal tarea es la creación de objetos de la clase T. Como se puede ver en la signatura del método, esta interfaz no admite ningún parámetro de entrada. De nuevo, se muestra un ejemplo del código del proyecto:

```
57       private Supplier<GpxType> gpxTypeSupplier = () -> new GpxType();
```

Esta nueva versión también incluye una de las novedades más destacas, una característica que abre la posibilidad de concatenar operaciones, mediante el uso de las interfaces *Function*, *Predicate, Consumer* o *Supplier*, por ejemplo, dependiendo de si la operación es un *map*, *filter, forEach* u *orElse*, respectivamente. Estas operaciones se aplicarán sobre un objeto siguiendo un estilo de programación declarativo, en lugar de imperativo. La clase que modela toda esta funcionalidad es la que está definida en la librería de Java 8 como *Optional<T>*, permitiendo acercar al lenguaje a la programación funcional siguiendo el patrón de diseño *Option*[58]. El principal objetivo de esta clase es encapsular un objeto para evitar los posibles *NullPointerException*[59] que puedan suceder. Para instanciar un objeto de esta clase se ha de llamar a los siguientes métodos estáticos: *of*, *ofNullable* y *empty* de la clase, ya que el constructor es privado. Pero lo que más interesante de trabajar con estos objetos, como se ha mencionado antes, es la posibilidad de ejecutar distintas operaciones, siguiendo el orden con que se definen, gracias a los métodos: *filter*, *map* y *flatMap*. Por ejemplo, en la aplicación se ha desarrollado una segunda versión con un código más próximo a la programación funcional. A continuación, se muestra un trozo de código utilizando un estilo de programación imperativo, es decir, se definen todos los pasos explícitamente de lo que ha de ejecutarse.

```
66   @Override
67   public Double getSpeedBetweenPoints(TrackPoint previous, TrackPoint current) {
68        if(!Objects.isNull(previous) && !Objects.isNull(current)
69            && !Objects.isNull(previous.getDate())
70            && !Objects.isNull(current.getDate())){
71                  long initTime = previous.getDate().getTime();
72                  long endTime = current.getDate().getTime();
73                  double totalTime = (endTime - initTime) / 1000;
74                  if (totalTime > 0)
75                            return (current.getDistanceMeters().doubleValue() -
76                                      previous.getDistanceMeters().doubleValue())
77                                      / totalTime;
79                  else
80                            return 0.0;
81        }else
82                  return 0.0;
91        }
92   }
```

Snippet 2

[58] http://www.codecommit.com/blog/scala/the-option-pattern

[59] Excepciones no comprobadas que se lanzan cuando se trata de realizer alguna operación sobre la instancia de una clase que no tiene valor definido, es *null*.

En el Snippet 2 se observa que se han creado variables auxiliares para guardar los tiempos y, aunque en este caso no ocurra, es muy normal que, en este estilo imperativo de programación, la ejecución pueda no tener siempre el mismo resultado debido a factores externos. Por ejemplo, que influya una variable global que usa el método con diferentes valores a lo largo del flujo de ejecución. Por lo tanto, estas funciones no son puras, ya que para los mismos valores de entrada no siempre devuelven el mismo valor. Sin embargo, la programación funcional toma como principal fundamento que las funciones sean puras y destaca este principio para que el entendimiento del código sea lo más rápido y fácil y que, además, no se cometan errores en el futuro. Para tratar de controlar que una función sea pura, es necesario que no se permita el uso de variables globales y si es necesario su uso, dichos valores sean inmutables. Con Java es recomendable que los parámetros de una función se declaren con la palabra reservada *final*.

Volviendo al Snippet 2, el valor que devuelve el método es un valor primitivo *double*, teniendo como valor de retorno por defecto 0,0. Es decir, esto implica que, si sucede cualquier error en la validación de los parámetros, por ejemplo, si el punto anterior y el actual son el mismo, el valor retornado será 0,0.

Además, también se observa que es un poco tedioso el hecho de comprobar que la información de entrada que se necesita viene con un valor definido, queda claro que no es la forma más pulcra de hacerlo.

A continuación, se muestra el Snippet 3 que tiene un paradigma más funcional y que realiza la misma funcionalidad que el Snippet 2, ambos calculan la velocidad que hay respecto a dos puntos de la trayectoria deportiva. Para ello, se hace uso de la clase *Optional*, creando una instancia con el constructor estático *ofNullable*; llamando a sus métodos públicos flatMap, map, filter y orElse; y haciendo uso de las funciones lambda que facilitan, en este caso, el filtrado de todos los tiempos positivos y que no sean cero.

```
66        @Override
67        public Double calculateSpeed(final TrackPoint origin, final TrackPoint end) {
68        return ofNullable(origin)
69                  .flatMap(__ -> ofNullable(end)
70                        .map(___ -> calculateTime(origin, end))
71                        .filter(MathUtils::isPositiveNonZero)
72                        .flatMap(totalTime -> ofNullable(calculateDistance(origin, end))
73                        .map(Math::abs)
74                        .map(distance -> distance / totalTime)))
75                        .orElse(null);
76        }
```

Snippet 3

En este ejemplo, lo que se devuelve es un tipo primitivo encapsulado en la clase *Double*. A diferencia del ejemplo anterior, ya no hace falta hacer comprobaciones en los parámetros de entrada, gracias a que se encapsula el valor en la clase *Optional* y automáticamente se ejecuta el *orElse* si se detecta que alguno de los métodos *ofNullable*, *map*, *flatMap* o *filter* ha devuelto un objeto vacío o nulo.

En la aplicación que se ha desarrollado, se han creado una serie de clases que contienen método usados por varias clases. Estas clases están divididas en función de las tareas que desempeña cada una: *CommonUtils*, *DateUtils*, *Encrypter*, *JsonUtils* y *MathUtils*. Además, también se ha creado una clase para guardar las constantes definida como *Constants*.

Si se pone atención sobre el último trozo de código, identificado como Snippet 3, se visualiza que se usa un *Predicate* en la operación de filtrado, llamada *filter*, definido en

una de las clases de utilidad anteriormente mencionadas que permiten comprobar que el valor sea positivo distinto de cero. En realidad, es un objeto que representa una función común pero que tiene la característica de recibir un objeto de tipo *Double* y de devolver un valor booleano, la misma peculiaridad que tiene el método *test* de la interfaz *Predicate*. Para invocar este método se usa el modo de referencia de métodos usando los dos puntos dobles, representados como "::", entre el nombre de la clase y el nombre del método que se desea llamar.

También se usa la operación *map* que invoca el método privado *calculateTime* de la misma clase que es producto de la refactorización de código siguiendo el principio de responsabilidad único[60]. A diferencia del trozo de código mostrado en el Snippet 2, el uso de *Optional* hace que cualquier error de validación o cualquier retorno interno inesperado de las demás operaciones utilizadas que se produzcan en el flujo de ejecución, conllevarán el retorno del valor vacío, identificado como *null*, y no el valor decimal 0,0 que podía dar lugar a confusión o conflicto en el entendimiento del método por otros desarrolladores en el futuro.

Otra de las funcionalidades incluidas en esta versión es la posibilidad de aplicar operaciones, recopiladas en una extensa biblioteca, sobre los objetos de tipo *Stream,* es decir, colecciones de objetos en los que se permite el procesamiento de datos en modo declarativo. Haciendo un símil con lo que se ha comentado anteriormente, de esta manera se permite aplicar una serie de operaciones a N objetos, mientras que, con la clase *Optional* tan solo se pueden aplicar operaciones a un único elemento. Si se pone atención a uno de los trozos de código, identificados por la etiqueta Snippet 1, mostrado más arriba se puede observar como se aplica para cada elemento las operaciones de *map* y *allMatch*, invocando sus respectivos métodos, sobre cada objeto del conjunto de elementos.

Para poder aplicar estas operaciones, como se puede ver en el código, tan solo hay que invocar al método *stream*, implementado en la interfaz *Collection* desde la versión 8 de Java, sobre cualquier objeto de alguna clase que implemente la interfaz como por ejemplo *List<T>* o *Set<T>*, entre otros. De esta forma se permite la creación de un objeto de la clase *Stream<T>* que facilitará el procesamiento de la información contenida. Haciendo un resumen, se podría decir que las instancias de esta clase tienen una secuencia de objetos que soportan varias operaciones con la posibilidad de encadenarlas para producir el resultado deseado. En este caso, lo que se aplica a cada elemento de tipo *TrackPoint* es la función pasada como parámetro, devolviendo los elementos resultantes de la aplicación de la función que contiene como parámetro cada operación. Además, como este elemento es del mismo tipo, como se dijo antes, se le pueden ir aplicando diferentes operaciones. Por último, se ejecuta sobre la secuencia de elementos el método *allMatch* que comprueba que todos los elementos cumplan el predicado pasado como parámetro.

Uno de los problemas que surgen siguiendo este estilo de programación es el que ocurre cuando alguna de las funciones lambda lanzan una excepción controlada y estas funciones son definidas como parámetros en las operaciones *map*, *flatMap*, o *filter*, entre otros, tanto en las instancias de clases *Optional* como de *Stream,* produciendo un error de compilación. Estos métodos no permiten en la versión 8 de Java el control de excepciones, por lo tanto, se ha usado una serie de clases definidas en una librería externa llamada

[60] https://en.wikipedia.org/wiki/Single_responsibility_principle

vavr[61]. Esta biblioteca permite al desarrollador utilizar conceptos de la programación funcional de una forma sencilla. En este caso en concreto, la librería permite gestionar las excepciones de forma fácil usando este paradigma de programación.

La clase *Try<T>* modela un contenedor de una instancia de la clase T o, si se ha producido algún error al evaluar alguna función, el contenedor tendrá el valor de esta excepción. Cada objeto de esta clase tendrá un estado que representa si el valor es el resultado de que las operaciones anteriores han ido bien, en ese caso *Success*, o si ha habido algún problema, en este otro caso el valor es *Failure*. En función de este estado se podrán concatenar más operaciones siempre y cuando haya habido éxito en las operaciones anteriores. Si ha fallado alguna, tan solo se podrán aplicar los métodos para devolver algún resultado por defecto, como es el método *getOrElse*, entre otros, devolviendo un objeto del último retorno del operador válido.

Por otro lado, existen diferentes métodos en función de si el valor de retorno está definido o es vacío. Si la operación no devuelve nada, tan solo realiza tareas de procesamiento, se usa el método estático *run* de la interfaz de la clase Try para inicializar el valor de la instancia. En contraste, si se quiere que el objeto devuelva un valor se usa el método estático *of* incluido en la clase. El comportamiento de los objetos de esta clase es muy similar a las clases comentadas más arriba que implementa de forma nativa la versión 8 de Java. Al igual que ocurría en las otras dos, para invocar ciertas acciones o funciones, también se puede hacer mediante el modo de referencia de métodos, explicado más arriba.

Esta clase funciona de *wrapper* cuando la función, sobre la que se esta aplicando el *map* o cualquier otro tipo de operación en cadena, puede producir una excepción. De esta forma se encapsula y procesa en función del valor del resultado de la operación. Por otro lado, esta clase también contiene otros métodos para convertir el contenedor junto con su valor en un objeto de las clases *Optional<T>*, *Stream<T>* o *List<T>*, según desee el desarrollador, utilizando los métodos *toJavaOptional*, *toJavaStream* / *toStream* o *toJavaList* / *toList*, respectivamente. Todo esto muestra una gran colaboración de esta librería para poder trabajar con las librerías de Java.

```
32      public Try<T> upload(final MultipartFile multiPartFile) {
33          return Try.of(() -> multiPartFile.getInputStream())
34                  .onFailure(err -> log.error("Error trying to get the input stream", err))
35                  .flatMap(xmlService::readXML);
36      }
```

Snippet 4

El Snippet 4 representa una forma de cómo utilizar la clase *Try*, el método *upload* que pertenece a la clase abstracta que modela el servicio que realiza la subida de ficheros al sistema.

Al margen de lo que estamos comentando, con Java 8 también se introducen las interfaces con métodos que pueden ser implementados de forma explícita, siempre y cuando, estén definidos con la palabra reservada *default* indicada al principio de la definición del método. Es por ello por lo que, si se desease, se podría crear una interfaz donde se definiese este método *upload*, en vez de, definirla como una clase abstracta. Volviendo al tema que se estaba comentado al inicio del párrafo, el uso de la clase *Try* posibilita el registro en el log, de una forma sencilla, cuando ha ocurrido algún error usando el método *onFailure*. Si la obtención del objeto InputStream ha ido correctamente, con la siguiente

[61] https://www.vavr.io/

operación *flatMap* se invoca al método *readXml*, perteneciente a un objeto inicializado mediante inyección de dependencias (se comentará más adelante sobre esta característica de Spring), que tiene como valor de retorno otro objeto *Try*, es por esto por lo que se utiliza el *flatMap* para "aplanar" el valor obtenido, es decir, convertir el objeto *Try<Try<T>>* a la instancia *Try<T>*.

```
26   @Override
27   public Try<T> readXML(InputStream inputFileXML) {
28        return Try.of(() -> getJAXBContext())
29                  .onFailure(err ->
30                            log.error("Error trying to get the jaxb context", err))
31                  .flatMap(jAXBContext -> Try.of( () -> jAXBContext.createUnmarshaller())
32                  .onFailure(err ->
33                            log.error("Error trying to create the un-marshaller", err)))
34                  .flatMap(unmarshaller -> Try.of(() ->
35                            unmarshaller.unmarshal(new StreamSource(inputFileXML), type))
36                  .onFailure(err ->
37                            log.error("Error trying to un-marshall", err))
38                  .map(JAXBElement::getValue)
39                  .map(type::cast));
40   }
```

Snippet 5

El Snippet 5 muestra la implementación del método *readXml* que pertenece a la clase abstracta que permite al sistema leer un fichero xml. Al igual que ocurría en el anterior caso, en el Snippet 4, también se podría utilizar una interfaz con la implementación de métodos por defecto. El método registra si ha ido mal alguna operación mediante el uso concatenado del método *onFailure* después de cada operación.

Al hilo de todo esto, cuando en los métodos de las clases fachadas llega algún valor vacío, encapsulado en un *Optional*, dependiendo de la función que implementa se propagará alguna de las excepciones personalizadas creadas en el paquete **`com.routeanalyzer.api.model.exception`** y que serán detalladas más adelante en el punto *Modelo* del apartado *Diseño*:

- *ActivityColorsNotAssignedException*
- *ActivityNotFoundException*
- *ActivityOperationNotExecutedException*
- *FileNotFoundException*
- *FileOperationNotExecutedException*

Por ejemplo, el Snippet 6 muestra un ejemplo sencillo de la aplicación donde se utilizan estas excepciones cuando el método de obtener una actividad deportiva por su identificador falla y no devuelve el objeto esperado. En ese caso, se propaga la excepción hacia el controlador donde será recogida y convertida a la respuesta http asociada.

```
28       @Override
29       public Activity getActivityById(final String id) throws ActivityNotFoundException {
30           return mongoRepository.findById(id)
31                    .orElseThrow(() -> new ActivityNotFoundException(id));
32       }
```

Snippet 6

El controlador de excepciones es una funcionalidad proveída por Spring, de la que se hablará más adelante en profundidad, en el punto *Controlador* del apartado *Implementación*, y que realiza un mapeo entre una excepción personalidad y su respuesta. Un ejemplo donde se realiza este mapeo es en el ejemplo Snippet 7.

```
78       @ResponseBody
79       @ExceptionHandler(ActivityNotFoundException.class)
80       @ResponseStatus(code = HttpStatus.NOT_FOUND)
81       Response handleActivityNotFoundException(final Exception exception) {
```

```
82              log.warn("Params error happened: ", exception);
83              return createErrorBody(true, ACTIVITY_NOT_FOUND, exception);
84          }
```

Snippet 7

En el código de arriba, se muestra como existe para cada excepción personalizada una instancia de la clase *Response*. Esta clase es muy simple, como se puede apreciar en el Snippet 8, contiene cuatro objetos: el valor booleano si la respuesta ha generado un error, la descripción de la respuesta, el mensaje de error y el nombre de la excepción.

```
7   @Value
8   @Builder
9   @AllArgsConstructor
10  public class Response {
11
12      private Boolean error;
13      private String description;
14      private String errorMessage;
15      private String exception;
16
17  }
```

Snippet 8

Además, se inserta en el log el error ocurrido usando el registro de tipo "aviso" para que quede rastro de lo que se ha producido. También se define la excepción de entrada y el estado de la respuesta, a través del uso de anotaciones, en el caso del tipo de respuesta, se devuelve un 404 que indica que la actividad no se ha encontrado.

La librería *vavr* también permite realizar *pattern matching*[62] de manera fácil utilizando el método estático *Match* de la clase API. En realidad, es una forma de hacer el código más legible y fácil de leer, en lugar de tener múltiples *if/else if/else*. Es decir, incrementa el nivel de azúcar sintáctico[63] del lenguaje. En el Snippet 9 se muestra un ejemplo de como implementar este patrón para determinar cual es el caso, definido en el código con el método estático Case de la clase API, que más encaja con el valor de entrada. Por otro lado, si no encaja ninguno de los valores enumerados, se propagará una excepción que determine que en los valores de entrada algo ha fallado.

```
183 @Override
184 public String exportByType(final String type, final Activity activity) {
185     return Match(type.toLowerCase()).option(
186             Case($(is(SOURCE_TCX_XML)), tcxFile -> tcxExportService.export(activity)),
187             Case($(is(SOURCE_GPX_XML)), gpxFile -> gpxExportService.export(activity)))
188             .getOrElseThrow(() -> new IllegalArgumentException("Bad type file."));
189 }
```

Snippet 9

Otra de las incorporaciones a esta nueva versión de la librería de Java es la disponibilidad de nuevas clases para el manejo de fechas, tiempos, estos dos últimos elementos unidos, el anterior con la zona horaria, instantes y duraciones con las siguientes clases respectivamente: *LocalDate*, *LocalDateTime*, *ZonedDateTime*, *Instant* y *Duration*, entre otros. El principal motivo es que la clase *Date*, desde un primer momento, no tuvo un buen diseño lo que hizo que los desarrolladores no tuvieran un buen entendimiento de cómo funcionaba, dando lugar a mucha confusión. Por ejemplo, la clase no representa una fecha si no que es más bien un instante. Además, no tiene ni zona horaria, ni un formato establecido ni ningún tipo de calendario asociado.

[62] https://es.wikipedia.org/wiki/B%C3%BAsqueda_de_patrones
[63] https://es.wikipedia.org/wiki/Az%C3%BAcar_sint%C3%A1ctico

Así pues, en esta última versión de la aplicación que se ha desarrollador, el sistema ha pasado de tener los campos de las fechas como Date a:

```
39          private ZonedDateTime date;
```

El motivo principal de usar *ZonedDateTime*, en lugar de utilizar *LocalDateTime*, es porque el usuario de la aplicación puede haber grabado la ruta deportiva con las fechas en otra zona que no sea la misma que el servidor tiene donde está desplegada la aplicación. Por lo tanto, es necesario que, independientemente de todo esto, se procese toda la información con la misma zona horaria.

Gracias a estas clases, todo el proceso de manejo de variables temporales se hace mucho más sencillo. Por ejemplo, para saber la duración transcurrida entre dos fechas, tan solo hay que hacer uso de la clase *Duration* y los métodos que la interfaz provee. A continuación, en el Snippet 10, se muestra un ejemplo real de la aplicación en la que se usa el método *between* para obtener una instancia de la clase con el valor del tiempo transcurrido entre las dos fechas que han sido pasadas en los parámetros:

```
381  private void setTotalTimeSecondsLap(final Lap lap) {
382       getLastTrackPoint(lap)
383                 .map(TrackPoint::getDate)
384                 .flatMap(timeMillisLastTrack -> getFirstTrackPoint(lap)
385                 .map(TrackPoint::getDate)
386                 .map(time -> Duration.between(timeMillisFirstTrack, time))
387                 .map(Duration::toNanos)
388                 .map(Double::valueOf)
389                 .map(nanoseconds -> nanoseconds / 1E9))
390                 .ifPresent(totalTimeSeconds -> ofNullable(totalTimeSeconds)
391                           .ifPresent(lap::setTotalTimeSeconds));
392  }
```

Snippet 10

Para poder recuperar los segundos exactos que han transcurrido entre las dos fechas iniciales, se trabaja directamente con este valor devuelto. Primero se obtienen los nanosegundos invocando al método *toNanos* de la clase y luego se convierte a segundos dividiendo este valor entre 10^9.

Lombok

En la segunda versión de la aplicación se comienza a hacer uso de esta librería que permite, mediante anotaciones, la generación de código en tiempo de compilación. Las líneas de código generadas suelen ser aquellos elementos que para el programador son repetitivos o procesos muy mecánicos. Esto ayuda a que los desarrolladores se puedan centrar más en lo que es la lógica de negocio en sí que en la implementación de *setters*, *getters*, constructores, etc.

Por lo tanto, esta librería ayuda en la generación automáticamente de los métodos *get*, *set*, *toString*, *equals*, *hashCode* y constructores de clase, entre otros. A continuación, en el Snippet 11, se muestra la clase del modelo de datos *Position* en la primera versión de la aplicación.

```
6    public class Position implements Serializable {
7        private static final long serialVersionUID = 1L;
8
9        private BigDecimal latitudeDegrees, longitudeDegrees;
10
11       public Position(BigDecimal latitudeDegrees, BigDecimal longitudeDegrees) {
12           this.latitudeDegrees = latitudeDegrees;
13           this.longitudeDegrees = longitudeDegrees;
14       }
15
16       public BigDecimal getLatitudeDegrees() {
```

```
17          return latitudeDegrees;
18      }
19
20      public void setLatitudeDegrees(BigDecimal latitudeDegrees) {
21          this.latitudeDegrees = latitudeDegrees;
22      }
23
24      public BigDecimal getLongitudeDegrees() {
25          return longitudeDegrees;
26      }
27
28      public void setLongitudeDegrees(BigDecimal longitudeDegrees) {
29          this.longitudeDegrees = longitudeDegrees;
30      }
31
32      @Override
33      public int hashCode() {
34          final int prime = 31;
35          int result = 1;
36          result = prime * result + ((latitudeDegrees == null)
37                  ? 0 : latitudeDegrees.hashCode());
38          result = prime * result + ((longitudeDegrees == null)
39                  ? 0 : longitudeDegrees.hashCode());
40          return result;
41      }
42
43      @Override
44      public boolean equals(Object obj) {
45          if (this == obj)
46              return true;
47          if (obj == null)
48              return false;
49          if (getClass() != obj.getClass())
50              return false;
51          Position other = (Position) obj;
52          if (latitudeDegrees == null) {
53              if (other.latitudeDegrees != null)
54                  return false;
55          } else if (!latitudeDegrees.equals(other.latitudeDegrees))
56              return false;
57          if (longitudeDegrees == null) {
58              if (other.longitudeDegrees != null)
59                  return false;
60          } else if (!longitudeDegrees.equals(other.longitudeDegrees))
61              return false;
62          return true;
63      }
64
65      @Override
66      public String toString() {
67          return "Position [latitudeDegrees=" + latitudeDegrees.doubleValue()
68                  + ", longitudeDegrees=" + longitudeDegrees.doubleValue() + "]";
69      }
70  }
```

Snippet 11

El código equivalente que usaría la librería Lombok quedaría reducido a tan solo las líneas de código, se muestra en el Snippet 12. Para ello, se usan anotaciones como **`@Data`** que es una forma abreviada de definir las siguientes anotaciones:

- **`@Getter`** encargada de generar los métodos *get* de las variables de clase.
- **`@Setter`** encargada de generar los métodos *set* de las variables globales que no son inmutables (no se han definido con final).
- **`@ToString`** encargada de la definición del método que lleva su nombre con todas las variables definidas en la clase.
- **`@EqualsAndHashCode`** encargada de definir los dos métodos que llevan el nombre con las variables globales.
- **`@RequiredArgsConstructor`** que se encarga de generar un constructor para aquellas variables que son inmutables (definidas con final) y de aquellos otros valores definidos con la anotación **`@NonNull`**.

```
11  @Data
12  @Builder
13  @AllArgsConstructor
14  @JsonIgnoreProperties(ignoreUnknown = true)
15  public class Position implements Serializable {
16
17      private static final long serialVersionUID = 1L;
18
19      private BigDecimal latitudeDegrees;
20      private BigDecimal longitudeDegrees;
21
22  }
```

Snippet 12

En el Snippet 12 se muestra un ejemplo donde, además de la anotación **`@Data`**, se definen otras anotaciones de la librería Lombok. Por ejemplo, se crea un constructor que contenga como parámetros todas las variables globales de la clase. También usa la anotación **`@Builder`** para crear una serie de métodos que siguen el patrón de diseño *builder*[64] para la construcción de objetos. Esta anotación también permite que clases que son inmutables puedan ser modificadas de forma sencilla usando el método toBuilder (indicándoselo en las opciones de la anotación **`@Builder(toBuilder=true)`**). Este método crea una instancia equivalente, con distinta referencia a nivel interno, indicándole con cada método de la clase Builder autogenerada asociada a la clase en concreto el nuevo valor para cada variable global que tiene definidas. Al final, se tiene que llamar al método *build* que generará la nueva instancia de la clase.

Otra de las ventajas que te proporciona la librería Lombok es la posibilidad de generar una clase de utilidad de forma muy sencilla. Lo primero, hay que aclarar que una clase de utilidad es aquella que contiene métodos estáticos usados por muchas otras clases y que todos comparten el mismo paquete donde está alojada la clase. Esta funcionalidad está disponible con el uso de la anotación **`@UtilityClass`** al inicio de la declaración de la clase, sin embargo, esta anotación, pese a estar ya a punto de ser estable, aún está en fase experimental. En el Snippet 13 se muestra la clase de utilidad *CommonUtils* que indica como utilizar la anotación para la creación de una clase de utilidad, comentada en este párrafo. Entiendo que, como esta en una fase experimental, al realizar el desarrollo, he descubierto que a esta clase no se le generaba bien el código al compilar si cada método no era definido con la palabra reservada *static*.

Esta clase tiene los siguientes métodos:

- *toListOfType* que convierte una lista de *String* a la clase de retorno que se le indique en el parámetro de tipo *Function* identificado como *convertTo*.
- *not* que lo que hace es negar un predicado, una expresión booleana. En la versión de Java 11, este método estático se incluye en la interfaz *Predicate*.
- *toStringValue* es un método que se encarga de devolver *null* si el valor no está definido o el valor con convertido a clase *String*.

```
16  @Slf4j
17  @UtilityClass
18  public class CommonUtils {
19
20      // List Utils
21
22      public static <T> List<T> toListOfType(final List<String> listStrings,
                final Function<String, T> convertTo) {
23              Function<String, Optional<T>> checkConvertTo = str ->
                                Try.of(() -> convertTo.apply(str))
```

[64] https://es.wikipedia.org/wiki/Builder_(patr%C3%B3n_de_dise%C3%B1o)

```
24                              .onFailure(err -> log.error("Error {}", convertTo, err))
25                              .toJavaOptional();
26                      return IntStream.range(0, listStrings.size())
27                              .boxed()
28                              .map(listStrings::get)
29                              .map(checkConvertTo)
30                              .filter(Optional::isPresent)
31                              .map(Optional::get)
32                              .collect(toList());
33          }
34
35          // Boolean utils
36
37          public static <T> Predicate<T> not(final Predicate<T> t) {
38                      return t.negate();
39          }
40
41          // String utils
42
43          public static String toStringValue(final Object value) {
44                      return ofNullable(value)
45                              .map(String::valueOf)
46                              .orElse(null);
47          }
48
49  }
```

Snippet 13

Maven

Es una herramienta de software que permite gestionar y construir proyectos Java. El epicentro del sistema es el fichero POM[65] donde se describe el proyecto software a construir, las dependencias que tiene con otros módulos, los componentes externos y el orden de construcción de los elementos. El motor incluido en su núcleo puede descargar de forma dinámica *plugins* de un repositorio remoto.

En el siguiente punto, se mostrará como se configura este fichero para que, entre otras cosas, gestione todas las dependencias que usa la aplicación *backend* que se ha desarrollado.

Spring

Es un *framework* de código abierto creado para la plataforma Java que facilita el desarrollo de aplicaciones. Se fundamenta en el principio de Inversión de Control[66], frecuentemente usado en la programación orientada a objetos, en el que el flujo de ejecución se invierte respecto a los métodos de programación tradicionales. En vez de definir de forma imperativa los procedimientos o funciones, en su lugar, lo que se hace es especificar las respuestas deseadas o las solicitudes de datos. Por tanto, se delega el control de los métodos o procedimientos a una entidad externa que es la encarga de ejecutar y devolver el output al lugar desde donde se hizo la llamada a dicho servicio. Las ventajas de seguir esta arquitectura es que se desacopla las tareas de ejecución de las de implementación, se facilita el intercambio entre diferentes implementaciones, una mayor granularidad del código y mayor facilidad para probar el código de forma unitaria, aislando el código a probar o *mockeando* sus dependencias.

En este caso, Spring implementa el principio de inversión de control mediante el patrón de diseño llamado Inyección de Dependencias[67] que se encarga de suministrar objetos a

[65] Project Object Model: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html
[66] https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control
[67] https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias

una clase sin que sea ella la que los cree. Es decir, existirá una clase contenedora que cumpla con el contrato de interfaz previamente establecido entre ambas partes y que permita compartir la dependencia con la clase que usará el objeto con la funcionalidad descrita en el contrato. En las últimas versiones de Spring existen dos opciones para realizar la inyección de dependencias:

- Usando un documento *xml* de configuración donde se definen todos los *beans* que posteriormente, tras haberse inicializado el contexto, serán recuperados y provistos a las clases que los utilicen. Esta es la manera más tradicional de hacerlo, sin embargo, no es muy legible para el desarrollador.
- Mediante el uso de anotaciones.
  Las anotaciones que existen para instanciar *beans* en el contexto de Spring son:
  - **`@Bean`** usada en clases con la anotación **`@Configuration`**. como se mostrará más adelante en el Snippet 15, para la definición de *beans*, valga la redundancia, a través de la definición al inicio de estos métodos.
  - **`@Component`** es una anotación a nivel de clases. Genera un *bean* con el mismo nombre, pero la primera letra con minúscula. Los siguientes elementos son meta-anotaciones de esta, con diferente significado semántico:
    - **`@Service`** es una anotación para definir *beans* que residen dentro de la capa de servicios de la aplicación. En el sistema desarrollado hay varías clases, por ejemplo, la clase *TcxExportFileService* que se encarga de exportar actividades a ficheros *xml* con un esquema *tcx*.
    - **`@Repository`** es una anotación para representar objetos de la capa de acceso a base de datos en la aplicación. La ventaja que tiene frente a definirlo con la anotación **`@Component`** es que existe una traducción directa entre las excepciones que se pueden producir al trabajar con la base de datos y las subclases de la clase de Spring que maneja este tipo de errores, identificada como *DataAccessException*. En la aplicación existe una clase para el acceso a la base de datos *MongoDB* que se llama *ActivityMongoRepository* que está definida con esta anotación. Se usan las anotaciones del módulo *Spring Data* en los *dto*[68].
    - **`@Controller`** representaran las clases de las que se quieren instanciar *beans* que servirán como elementos de la capa del controlador. En la aplicación se usará una anotación, llamada **`@RestController`**, que es un alias de este elemento y que sirve para crear un controlador que proveerá a la aplicación de una API REST. Esta anotación es usada en estos dos controladores: *FileRestController* y *ActivityRestController*.
    - **`@Configuration`** es una clase que contendrá métodos que producirán la instanciación de *beans*, como se comentó más arriba. Servirá sobre todo para definir la configuración de diferentes aspectos del sistema.

[68] https://es.wikipedia.org/wiki/Objeto_de_transferencia_de_datos

Por otro lado, para obtener del contexto las dependencias que se han guardado en el contexto de Spring, se puede usar la anotación **`@Autowired`**, definida en la propia clase que use la dependencia, permitiendo inyectar el objeto directamente al crear la clase. Sin embargo, para una mayor facilidad a la hora realizar pruebas unitarias y, siguiendo las recomendaciones, tan solo se ha usado esta anotación en las variables globales de las clases de prueba. Por tanto, se ha usado una inyección de dependencias a través del constructor de cada clase que tiene dependencias con *beans* del contexto.

Lo primero de todo, la aplicación ha sido desarrollada con la versión 5 de Spring ya que es en esta versión en la que se implementa la versión 2 de *Spring Boot*[69]. Este es un módulo que permite centrarse en las tareas de desarrollo y dejar a un lado temas de configuración inicial y de despliegue, más adelante se entrará más en detalle. Volviendo al código de la aplicación, en ella se puede ver como se ha utilizado la funcionalidad comentada:

- En el Snippet 14 se muestra la clase que inicializa la aplicación Spring Boot gracias a la definición de **`@SpringBootApplication`** al inicio de la clase. Con esta anotación también se le indica el directorio raíz desde donde puede escanear los demás componentes, es decir, se analizarán todas las clases de los distintos sub-paquetes que tengan como raíz el directorio **`com.routeanalyzer.api`**. Además, también sirve para inicializar la configuración de la aplicación.

```
6   @SpringBootApplication
7    public class RouteAnalyzerApplication {
8
9        public static void main(String[] args) {
10           SpringApplication.run(RouteAnalyzerApplication.class, args);
11       }
12
13   }
```

Snippet 14

La clase solo define el método *main* con los argumentos y dentro se lanza la propia clase con el método estático *run* de la clase *SpringApplication*.

- En el paquete **`com.routeanalyzer.api.config`** se definen una serie de clases de configuración.

```
14   @Configuration
15   @EnableSwagger2
16   @EnableConfigurationProperties({GoogleMapsApiProperties.class})
17   public class RouteAnalyzerConfiguration {
18
19       @Bean
20       public RestTemplate restTemplate() {
21           return new RestTemplate();
22       }
23
24       @Bean
25       public Docket api() {
26           return new Docket(DocumentationType.SWAGGER_2)
27                   .select()
28                   .apis(RequestHandlerSelectors.any())
29                   .paths(PathSelectors.any())
30                   .build();
31       }
32
33   }
```

Snippet 15

[69] https://projects.spring.io/spring-boot/

En el Snippet 15 se muestra la configuración principal de la aplicación. En esta clase se crean los *beans* de la aplicación que serán inyectados al instanciarse otras clases que los usen. Se enumeran los dos objetos que se han creado:

- Una instancia de la clase *RestTemplate* de Spring que sirve para realizar peticiones http a servicios de terceros. En la aplicación, se utiliza esta instancia en la clase *GoogleMapsApiService* para realizar peticiones al servicio de Google que devuelve la altura que hay en un determinado punto definido por la latitud y la longitud. La anotación de la clase definida como **`@EnableConfigurationProperties`** permite que se active la clase *GoogleMapsApiProperties* como contenedor de los valores de las propiedades que vienen del fichero *application.properties*.
- Una instancia de la clase Docket que sirve para generar el *endpoint* con toda la documentación que provee la herramienta Swagger[70]. Para acceder al *endpoint* donde está documentada toda la API de la aplicación hay que concatenar a la dirección de la máquina donde esta desplegada la aplicación el *path /swagger-ui.html*. En relación con esta herramienta, en la aplicación de configuración también se declara la anotación **`@EnableSwagger2`** que sirve para activar esta herramienta.

En el Snippet 16 se muestra otra de las clases de configuración del sistema *backend*. En este caso es la que se encargará de configurar la conexión con el servicio S3[71] de la plataforma en la nube *Amazon Web Services* que proporciona un repositorio de documentos donde se almacenará el fichero original de la ruta deportiva.

```
21  @Configuration
22  @RequiredArgsConstructor
23  @EnableConfigurationProperties(AWSConfigurationProperties.class)
24  public class AWSConfiguration {
25
26      private final AWSConfigurationProperties properties;
27
28      @Bean
29      public AmazonS3 s3client() {
30          return createCredentials(properties.getDecodeAccessKeyId(),
31                      properties.getDecodeSecretAccessKey())
32                  .map(this::createAmazonS3HTTPProtocol)
33                  .orElse(null);
34      }
35
36      private Optional<BasicAWSCredentials> createCredentials(
37              final String accessKeyId,
38              final String secretAccessKey) {
39          return ofNullable(accessKeyId)
40                  .filter(StringUtils::isNotEmpty)
41                  .flatMap(__ -> ofNullable(secretAccessKey)
42                          .filter(StringUtils::isNotEmpty)
43                          .map(___ -> new BasicAWSCredentials(accessKeyId,
44                                  secretAccessKey)));
45      }
46
47      private AmazonS3 createAmazonS3HTTPProtocol(
48              final BasicAWSCredentials credentials) {
49          ClientConfiguration config = new ClientConfiguration();
50          config.setProtocol(Protocol.HTTP);
51          return AmazonS3ClientBuilder.standard()
52                  .withRegion(Regions.fromName(properties.getS3Region()))
53                  .withCredentials(new AWSStaticCredentialsProvider(credentials))
```

[70] https://swagger.io/
[71] Amazon Simple Storage Service

```
54                      .withClientConfiguration(config)
55                      .build();
56          }
57
58      }
```

Snippet 16

Como se comentó en el punto *Lombok*, la anotación **`@RequiredArgsConstructor`** generará el código que realice la función de constructor teniendo como parámetros los elementos con la palabra reservada *final*. En este caso, este parámetro es el *bean* inyectado por el contexto de Spring mediante el constructor de la clase. Para crear el objeto de la clase AmazonS3 se usan las librerías creadas por Amazon creadas para usar el servicio con el lenguaje de programación Java.

Por otro lado, en esta clase se usa un elemento que no se ha visto anteriormente, el uso de una clase como contenedor de propiedades de las que su valor ha sido obtenido del fichero *application.properties* alojado en el directorio *resources* de la aplicación. Para activar esta clase de propiedades de configuración es necesario que se habilite gracias a la anotación **`@EnableConfigurationProperties`**, indicando en el parámetro la clase de configuración llamada *AWSConfigurationProperties*, al principio de la definición de la clase de configuración donde se va a usar. En el Snippet 17 se muestra la clase de propiedades de la configuración del servicio de *AWS*. Para ello, es necesario que se defina a nivel de clase con la anotación **`@ConfigurationProperties(prefix = "aws")`** indicándole en el parámetro el prefijo de todos los nombre de las propiedades de esta clase.

```
10  @Data
11  @ConstructorBinding
12  @RequiredArgsConstructor
13  @ConfigurationProperties(prefix = "aws")
14  public class AWSConfigurationProperties {
15
16      private final String encryptedAccessKeyId;
17      private final String encryptedSecretAccessKey;
18      private final String s3Region;
19      private final String s3Bucket;
20
21      public String getDecodeAccessKeyId() {
22          return decrypt(encryptedAccessKeyId);
23      }
24
25      public String getDecodeSecretAccessKey() {
26          return decrypt(encryptedSecretAccessKey);
27      }
28
29  }
```

Snippet 17

Si se pone atención en el fichero de definición de propiedades, mostrado en el Snippet 18, se identifica a todas las claves de las propiedades por el prefijo “aws”, definido en el parámetro de la anotación que define la clase de configuración de propiedades. Por tanto, estos valores serán inyectados en las clases de configuración. Como el código está almacenado en un repositorio público en GitHub, se han encriptado los valores reales usando la clase de utilidad *Encrypter*, comentada anteriormente en el punto Lombok *Lombok*. Si se aplica el método estático de la clase utilidad *decrypt* sobre los primeros valores se obtendrían las claves planas para el acceso al servicio en la nube. Así es como se obtiene directamente desde esta aplicación, llamando a los métodos *getDecodeAccessKeyId* y *getDecodeSecretAccessKey*, de esta misma clase.

```
9    # amazon web service s3
10   aws.encrypted-access-key-id=DlM1pGPUM6CcV2SkAI6xMCD3C926uxdOsb1W3J2vkdc=
11   aws.encrypted-secret-access-key=
            JYNrVuPIMn2TAqI4QmRcR+VD42c5g4rsqWpoQDGc/x3h5oIOADqxok+vADS73ini
12   aws.s3-bucket=route-analyzer-bucket
13   aws.s3-region=eu-west-1
```

Snippet 18

Existe otra clase de configuración que sirve para configurar el acceso a la base de datos distribuida que se encuentra alojada en un servidor externo. Por tanto, para la configuración de esta base de datos se utiliza el módulo Spring Data MongoDB[72] en la aplicación de Configuración mostrada en el Snippet 19. Aquí se usa el mismo método de inyección de dependencias, mediante el uso de los parámetros del constructor gracias al uso de **`@RequiredArgsConstructor`**.

```
25   @Configuration
26   @RequiredArgsConstructor
27   @EnableConfigurationProperties(MongoProperties.class)
28   @EnableMongoRepositories(basePackages= "com.routeanalyzer.api.database")
29   public class MongoConfig extends AbstractMongoClientConfiguration {
30
31       private final MongoProperties properties;
32
33       private final List<Converter<?, ?>> converters = new ArrayList<>();
34
35
36       @Override
37       protected String getDatabaseName() {
38           return properties.getMongoDatabase();
39       }
40
41       @Override
42       public MongoClient mongoClient() {
43           return MongoClients.create(
44                   new ConnectionString(properties.getMongoUri()
45                           + getDatabaseName()));
46       }
47
48       @Bean
49       @Override
50       public MappingMongoConverter mappingMongoConverter() throws Exception {
51           DbRefResolver dbRefResolver =
52                   new DefaultDbRefResolver(mongoDbFactory());
53           MappingMongoConverter converter =
54                   new MappingMongoConverter(dbRefResolver, mongoMappingContext()) {
55               @Override
56               public void setCustomConversions(CustomConversions conversions) {
57                   super.setCustomConversions(conversions);
58                   conversions.registerConvertersIn(conversionService);
59               }
60
61           };
62           converter.setCustomConversions(customConversions());
63           return converter;
64       }
65
66       @Override
67       public MongoCustomConversions customConversions() {
68           converters.add(new ZonedDateTimeReadConverter());
69           converters.add(new ZonedDateTimeWriteConverter());
70           return new MongoCustomConversions(converters);
71       }
72
73       class ZonedDateTimeReadConverter
74               implements Converter<Date, ZonedDateTime> {
75           @Override
76           public ZonedDateTime convert(Date date) {
77               return date.toInstant().atZone(ZoneOffset.UTC);
78           }
79       }
```

[72] https://spring.io/projects/spring-data-mongodb

```
80
81      class ZonedDateTimeWriteConverter
82              implements Converter<ZonedDateTime, Date> {
83          @Override
84          public Date convert(final ZonedDateTime zonedDateTime) {
85              return Date.from(zonedDateTime.toInstant());
86          }
87      }
88  }
```

Snippet 19

Además, para que se pueda acceder a la base de datos mediante el uso del *bean* correspondiente, definido con la anotación **`@Repository`**, se ha de utilizar la anotación **`@EnableMongoRepositories`** indicándole en los parámetros el paquete donde esta alojada dicha clase. En el caso de la aplicación que se ha desarrollado, se ha incluido en el paquete **`com.routeanalyzer.api.database`**. Como en el caso de la aplicación para la configuración del servicio de *AWS*, también aquí se hace uso de una clase para la definición de lo valores de las propiedades de configuración. Esta clase es la que se muestra en el trozo de código definido en el Snippet 20, como ocurría con otras clases de propiedades de configuración mostradas más arriba, esta clase está definida con la anotación **`@ConfigurationProperties`** indicando en el parámetro el prefijo que tendrán las claves de cada variable. Adicionalmente, hay que añadir que esta forma de extraer las propiedades del fichero *application.properties*, sustituyendo en este caso a la anotación **`@Value`**, es relativamente nueva y tan solo se permite desde la versión 2.2 del módulo *Spring Boot*.

```
10  @Value
11  @ConstructorBinding
12  @RequiredArgsConstructor
13  @ConfigurationProperties(prefix = "route-analyzer")
14  public class MongoProperties {
15
16      private final String encryptedMongoUri;
17      private final String mongoDatabase;
18
19      public String getMongoUri() {
20          return decrypt(encryptedMongoUri);
21      }
22  }
```

Snippet 20

Esta clase también tiene el valor de la propiedad de la dirección de la base de datos encriptada. Por lo tanto, para obtener el valor real de esta propiedad, hay que invocar al método *getMongoUri* que utiliza la clase *Encrypt* para descodificar el valor del fichero de propiedades.

Por otro lado, en la clase de configuración mostrada en el Snippet 19 también se definen otra serie de métodos que básicamente se encargan de que se realice correctamente la transformación o mapeo entre una fecha guardada en la base de datos en una determinada zona horaria y el objeto que usa el sistema para trabajar con fechas que tengan el huso horario UTC[73], también denominado a veces como GMT[74] ya que es el que a traviesa la zona de Greenwich, cerca de Londres.

[73] Universal Time Coordinated
[74] Greenwich Mean Time

Si nos fijamos en el Snippet 21, se visualizan las dos claves asociadas a la clase de propiedades definidas en el Snippet 20 y que comienzan con el prefijo *route-analyzer*.

```
2    # mongodb configuration
3    route-analyzer.encrypted-mongo-uri=qmS7J/8srGrwFd36lvRTKhWBrQZd0/dXSaC50D0
4    route-analyzer.mongo-database=test
```

Snippet 21

Por último, se va a mencionar que en el código de la aplicación *backend* que se ha desarrollado también se ha definido otra clase de propiedades, la que sirve para obtener los valores de configuración de la llamada al servicio de terceros de Google y de la que se hablará más adelante.

Por lo tanto, estas serían, a grandes rasgos, las clases necesarias para la configuración de cualquier aplicación que quiera usar *Spring Boot*. Un módulo muy potente para facilitar la creación de aplicaciones y librando al usuario de pelear con configuraciones tediosas.

Por otro lado, para hacer uso de todas estas anotaciones es necesario que en el *pom* del Maven estén explícitamente descritas las dependencias a las librerías que hay que tener para poder usar Spring y todos los módulos que necesitará el sistema para su correcto funcionamiento. En el Snippet 22 se muestra una parte importante del fichero de configuración de Maven, localizada dentro de la etiqueta raíz del fichero, que se encarga de leer el fichero padre que se define en la etiqueta <*parent*> y de crear un *pom* efectivo que contenga los datos del padre y de la demás información contenida en el fichero del proyecto. Spring define unos *pom* predefinidos para agilizar el desarrollo de aplicaciones, en este caso se está haciendo uso del elemento predefinido con el nombre *spring-boot-starter-parent* de Spring de la versión *2.2.2*.

```
7    <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.2.2.RELEASE</version>
11   </parent>
```

Snippet 22

Por otro lado, en el Snippet 23, es donde se definen los valores que identifican al proyecto como el nombre, el identificador del grupo, el identificador del artefacto y la versión. Estos datos están definidos dentro de la etiqueta raíz <*project*>.

```
11        <name>Route Analyzer API Service</name>
12        <groupId>com.routeanalyzer</groupId>
13        <artifactId>api</artifactId>
14        <version>0.0.1</version>
```

Snippet 23

En segundo lugar, en el Snippet 24 se muestran las dependencias que tiene este proyecto con las librerías de Spring, en concreto. Todas ellas se encuentran definidas dentro de la etiqueta <*dependencies*> para poder hacer uso de los diferentes módulos del *framework*.

```
29        <dependency>
30                  <groupId>org.springframework.boot</groupId>
31                  <artifactId>spring-boot-starter-web</artifactId>
32        </dependency>
33        <dependency>
34                  <groupId>org.springframework.boot</groupId>
35                  <artifactId>spring-boot-starter-web-services</artifactId>
36        </dependency>
37        <dependency>
38                  <groupId>org.springframework.boot</groupId>
39                  <artifactId>spring-boot-starter-data-mongodb</artifactId>
40        </dependency>
```

Snippet 24

En tercer lugar, en el Snippet 25 se definen las dependencias de este marco de trabajo para el proceso de creación de pruebas que serán descritas en el apartado *Pruebas*.

```
136         <dependency>
137                 <groupId>org.springframework.boot</groupId>
138                 <artifactId>spring-boot-starter-test</artifactId>
139                 <scope>test</scope>
140         </dependency>
141         <dependency>
142                 <groupId>org.springframework.cloud</groupId>
143                 <artifactId>spring-cloud-contract-wiremock</artifactId>
144                 <version>${spring.wiremock.version}</version>
145                 <scope>test</scope>
146         </dependency>
```

Snippet 25

Por último, se debe definir el *plugin* de Spring que se empleará durante la construcción del ejecutable de la aplicación. En el Snippet 26 se muestra el código que realiza todo esto y que estará ubicado dentro de la etiqueta raíz del proyecto.

```
171 <build>
172   <plugins>
173      <plugin>
174        <groupId>org.springframework.boot</groupId>
175        <artifactId>spring-boot-maven-plugin</artifactId>
176        <configuration>
177            <mainClass>com.routeanalyzer.api.RouteAnalyzerApplication</mainClass>
178        </configuration>
179
180        <executions>
181            <execution>
182                <goals>
183                    <goal>repackage</goal>
184                </goals>
185            </execution>
186        </executions>
187      </plugin>
188    </plugins>
189 </build>
```

Snippet 26

En estos trozos del fichero *pom* se pueden ver definidas las declaraciones que tiene para hacer funcionar la aplicación con Spring. En el Snippet 25 se puede ver, entre otras cosas, la declaración de la versión en la etiqueta del mismo nombre con el valor **`${spring.wiremock.version}`**. El valor de esta declaración es definido en la etiqueta <*properties*> que es define al dentro de la etiqueta raíz del fichero y contiene la versión de la dependencia que se quiere descargar y seleccionar para la aplicación, en este caso la versión de la dependencia llamada *spring-cloud-contract-wiremock*. A diferencia de las demás dependencias de Spring que no tienen esta etiqueta *version* definida de forma explicita porque el valor ya esta definido en el *pom* padre que se ha incluido gracias a al contenido definido en el Snippet 22.

Frontend

Existen multitud de librerías implementadas con el lenguaje *JavaScript* que facilitan la creación de la capa del lado del cliente de una aplicación. Sin embargo, las que más popularidad han cogido en los últimos años son:

- ***Angular*** fue desarrollada por Google en 2010 y es la librería más antigua de las tres que se exponen aquí. Marca el origen de las *Single Page Application*[75], explicado más adelante lo que supuso este nuevo enfoque en el mundo web. Uno de los cambios más significativos se produjo en el 2016 con la versión 2, quitando el sufijo "JS" del nombre. La versión 9.1.4 es la última estable que se sacó al mercado el 29 de abril de 2020. En sucesivos años el equipo encargado del desarrollo ha prometido que habrá dos actualizaciones por año.
- ***React*** fue desarrollado por Facebook en 2013 y es usado, actualmente, para el desarrollo de sus propios productos (*Facebook*, *Instagram* y *WhatsApp*). La última versión que ha sido lanzada es la 16.13.1 en marzo de 2020.
- ***Vue*** fue desarrollado por un antiguo trabajador de Google en 2014 siendo el *framework* más reciente de los tres. Aunque no tiene el respaldo de una gran empresa tecnológica como los dos anteriores *frameworks*, en los últimos tres años ha cogido gran renombre a nivel internacional. La última versión estable es la 2.6.11 lanzada a finales del año 2019.

Teniendo en cuenta el número de estrellas de GitHub, habiendo usado esta herramienta[76], se ha identificado cuál de los *frameworks* anteriormente descritos es más popular para los usuarios de esta plataforma que provee de un repositorio de código distribuido y en línea.

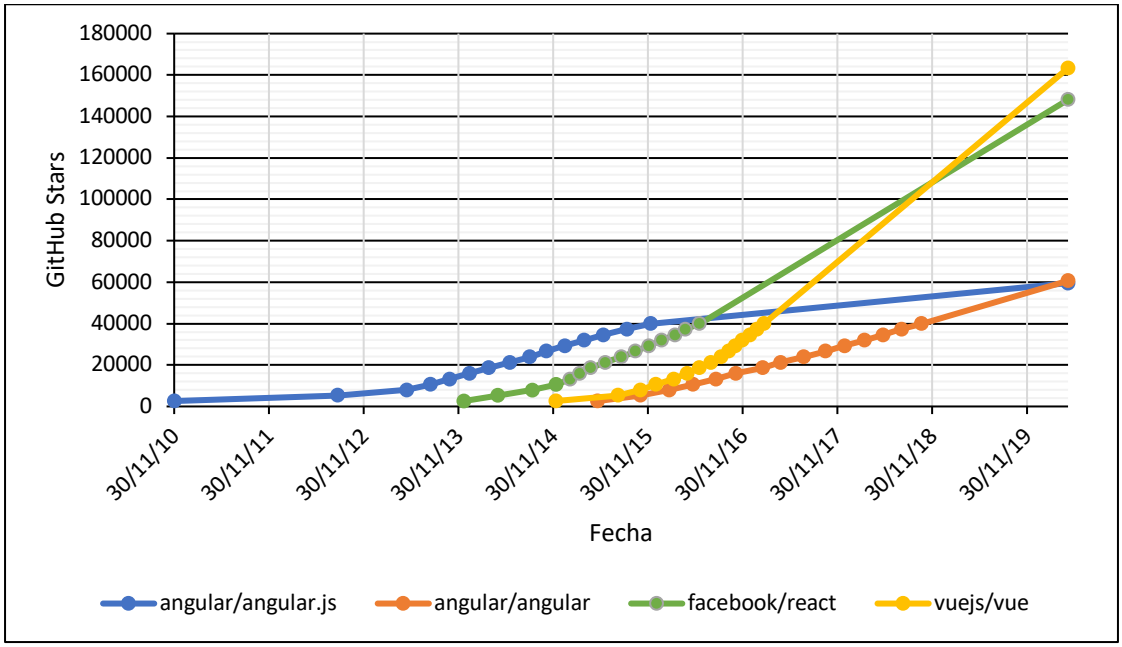


Gráfico 3

En la Gráfico 3 se puede ver como en los últimos meses *Vue* se ha convertido en el *framework* que tiene más popularidad. Sin embargo, la diferencia con respecto a React

[75] Single Page Application: https://es.wikipedia.org/wiki/Single-page_application
[76] https://star-history.t9t.io

no es muy significativa. No ocurriendo lo mismo con Angular, que ocupa una posición en la gráfica bastante más baja. Por otro lado, si se pone atención en las posiciones de trabajo de desarrollo *frontend* más solicitadas en los portales de empleo más populares se percibe que, aunque exista un auge en el interés por la librería *Vue*, las empresas aun no están utilizando esta librería de manera significativa, siendo la mayoría de las ofertas para roles de desarrollo web con *Angular* o *React*. Por último, si se toma como referencia el tamaño que ocupa cada librería, se comprueba que varía de unas a otras, mientras que *Angular* es el más pesado con más de 500kB, *React* y *Vue* tan solo pesan 100kB y 80kB respectivamente.

Tras este pequeño análisis, se ha decidido escoger *React* para el desarrollo de la capa de diseño. El principal motivo de haber elegido este *framework* es porque la curva de aprendizaje para una persona que se inicia en el mundo del *frontend* no es característica por tener una gran pendiente como la de *Angular*. Otro de los motivos, a diferencia de *Vue*, es que está apoyado por una gran comunidad ya que los empleados de Facebook realizan grandes contribuciones en los repositorios públicos.

La irrupción de estos tipos de *frameworks* en el mundo del desarrollo web ha permitido la creación de aplicaciones de una sola página, definidas en el pie de nota con el número 75, que suponen un gran paso adelante en lo que a desarrollo web se refiere. Al principio, la web se basaba en un sistema muy simple llamado SSR[77] que realizaba peticiones al servidor y, en el lado del navegador, se mostraban los resultados al usuario. Más tarde, surgió la idea de poder hacer estas peticiones en ciertas secciones especificas de la página web visualizada, realizándolas mediante una herramienta llamada *AJAX* de *JavaScript*. Al fin y al cabo, no dejaba de ser el mismo sistema anterior pero ligeramente modificado. De esta pequeña modificación evolucionan los sitios web tradicionales a las SPA, de la posibilidad de hacer pequeñas llamadas para actualizar determinados componentes y de la implementación del modelo MVC[78] en este nuevo sistema.

Hubo algún intento de hacer popular el uso de aplicaciones ejecutadas en el navegador web como son usadas las aplicaciones de escritorio, mediante un complemento o una maquina virtual que agregaran todas estas funcionalidades, es lo que se denominó RIA[79] que buscaban una mejora en la experiencia de usuario. Algunos ejemplos de estas aplicaciones son *Adobe Flash*, *JavaFX* y *Microsoft Silverlight*, entre otros. Sin embargo, en 2012 se incrementa el interés en favorecer del desarrollo web con HTML5 en lugar del uso de estos complementos para el desarrollo de aplicaciones enriquecidas. Sin embargo, llevar toda el procesamiento de datos y la carga de trabajo al navegador puede suponer que el navegador acabe colapsando, por eso, en la actualidad se está tratando de mezclar la tecnología SSR junto con las SPA produciendo una sinergia que produzca un correcto rendimiento de la aplicación web.

Como complemento a esta librería, también se usa *Redux*[80] que permite la gestión de un estado global a toda la aplicación sin que sea solo aplicable a un componente concreto. A continuación, veremos cada tecnología en concreto.

---

[77] Server-Side Rendering
[78] Model-View-Controller: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
[79] Rich Internet Application: https://es.wikipedia.org/wiki/Rich_Internet_application
[80] https://redux.js.org/introduction/motivation

JavaScript

Es un lenguaje de programación creado a mediados de los años 90, en concreto en el 1995, con la finalidad de permitir a los desarrolladores *embeber* pequeños trozos de código en las páginas HTML e interactuar con el DOM[81]. Es interpretado e implementa el estándar *ECMAScript* que ha tenido influencias de otros lenguajes de programación como: *Java*, *Python*, *C*, *Perl* y *Self*, entre otros. Por tanto, alberga ciertas características comunes de alguno de ellos como es el no ser tipado[82], influenciado por *Perl* que tampoco lo es, mientras que *Java*, comentado más arriba en el punto *Java*, y *Python* sí son fuertemente tipados.

Al igual que muchos de los lenguajes por los que está influenciado, también puede generarse código con un paradigma de orientación a objetos[83]. Sin embargo, está basado en prototipos[84], cualidad que hereda del lenguaje *Self* que, a diferencia de instanciar clases, en lo que consiste es en realizarla mediante la clonación de otros objetos o por la escritura de código del propio programador.

Otra de las peculiaridades heredadas de *Python* de este lenguaje es que es de tipado dinámico ya que una misma variable puede tomar valores de distinto tipo en distintos momentos del flujo de ejecución del programa. Esta característica hace que la comprobación de tipificación se realice durante la compilación o en tiempo de ejecución en función de si el lenguaje es de tipado estático o dinámico, respectivamente.

Mayoritariamente, su uso ha sido en el lado del cliente, desde el navegador web, permitiendo la posibilidad de tener páginas web dinámicas y, por tanto, mejorando la interfaz de usuario. Sin embargo, desde finales de los años 2000 su uso también se ha ido popularizando en el lado del servidor, siendo *Node.js*, plataforma desarrollada por Google, una de las implementaciones más populares en el lado del servidor de *JavaScript*.

Además de todo esto, su gran uso en todo tipo de plataformas hace que sea uno de los lenguajes más populares en la actualidad, según el ranking de popularidad de *TIOBE* se encuentra en la séptima posición frente a otros lenguajes de programación (datos de mayo de 2019).

Existen algunas dificultades a la hora de desarrollar una aplicación con este lenguaje para aquellas personas que vienen de otros lenguajes de programación que tienen tipado estático como *Java*, *C* y algunos otros. Uno de los problemas a los que se enfrenta un desarrollador la primera vez que empieza a programar con *JavaScript* es que hay que tener en cuenta de qué tipo de dato es una variable en cada momento, pudiendo no ser siempre del mismo tipo a lo largo del flujo de ejecución del programa.

Por lo tanto, se pueden realizar cosas que, a priori, para la gente que viene de otros lenguajes de programación, podrían resultar extrañas. Por ejemplo, se puede hacer lo siguiente:

- **`'hello' == true`** esta condición es evaluada a verdadero

[81] Document Object Model: https://es.wikipedia.org/wiki/Document_Object_Model
[82] https://es.wikipedia.org/wiki/Tipado_fuerte#Ejemplos_de_lenguajes_no_tipados
[83] https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
[84] https://es.wikipedia.org/wiki/Programaci%C3%B3n_basada_en_prototipos

- **`'0' == true`** esta condición es evaluada a false.

Por lo tanto, en este lenguaje cada valor tiene un valor *booleano* asignado por defecto. La mayoría de ellos son valores **`true`**, llamados *truthy values*, mientras que, los *falsy values* son, entre otros:

- la cadena de caracteres vacía: **`""`**
- el valor numérico **`NaN: Not a Number`**
- el tipo de dato **`null`**
- el valor *booleano* **`false`**
- el tipo de dato **`undefined`**
- el *array* vacío: **`[]`**

Como respuesta a alguno de los problemas que ocasionaba a veces este lenguaje y tratando de hacerlo menos flexible con el tipado, nace *TypeScript* en 2012 de la mano de Microsoft añadiendo el tipado estático y objetos basados en clases.

A lo largo de los últimos años se han lanzado diferentes versiones del estándar *ECMAScript*. La versión 6, lanzada en junio de 2015, será la que se ha usado para el desarrollo de la aplicación. A continuación, se van a describir una serie de características que ofrece la versión utilizada de este estándar y que se han implementado en la aplicación *frontend* del sistema:

- Se permite el uso de constantes o variables inmutables que no podrán ser reasignadas con un nuevo valor. Por ejemplo, en el Snippet 27, se muestra un trozo de código donde se utiliza la palabra reservada **`const`**:

```
4
5    const DIR_BASE = "https://route-analyzer-api.herokuapp.com";
6
```

Snippet 27

  Se define como una constante la URL donde está alojado el servidor de *backend*.
- Funciones flecha también llamadas en inglés *arrow functions* que hace más clara las clausuras o *closures* en las funciones, como se muestra en el Snippet 28.

```
66   setActivityObject(id){
67           get(id)
68           .then( activityObject => {
69                   this.handleActivityResponse(activityObject);
70           })
71           .catch( err => {
72                   alert(err.message);
73           });
74   }
```

Snippet 28

  Lo que está en negrita sería aquello donde se han usado las *arrow functions*. De haber utilizado una versión anterior, habría que haberlo implementado como se muestra en el Snippet 29:

```
66   setActivityObject(id){
67           get(id)
68           .then(function(activityObject) {
69                   this.handleActivityResponse(activityObject);
70           })
71           .catch(function(err) {
72                   alert(err.message);
73           });
74   }
```

Snippet 29

- Se permite expandir una expresión en ocasiones donde se esperan múltiples llamadas a argumentos (llamadas a funciones) o a múltiples elementos (*arrays* literales). En el Snippet 30 se puede ver un ejemplo hecho en la aplicación.

```
46     handleMarkClick(key){
47       let index = this.state.keys.indexOf(key);
48       if(index===-1){
49         this.setState({keys: [key, ...this.state.keys] });
50       } else{
51         this.setState({
52             keys: [...this.state.keys.slice(0,index),
53                  ...this.state.keys.slice(index+1)]
54           });
55       }
56     }
```

Snippet 30

Se utiliza para modificar el estado de un componente y así se visualicen los cambios en lo que es la vista. Para ello, es necesario que haya "cambios" y esto implica nuevos objetos, será más adelante explicado, en el punto denominado como React. Por lo tanto, en este caso se modifica el *array* con el nombre *keys* para que, si existe, se añada al *array* (*if*) y, si ya existía, se borre (*else*).

- Se permite importar valores hacia módulos sin un ámbito global. Por ejemplo, se muestra en el Snippet 31 un ejemplo de la aplicación.

```
1
2    import React, { Component } from 'react';
3
```

Snippet 31

Se permite exportar elementos a otros elementos mediante el uso de la palabra clave **`export`** como se muestra en el Snippet 32.

```
24
25   export class HeartRateChart extends React.Component {
26
```

Snippet 32

Si se quiere tener un elemento a exportar por defecto en una clase hay que definir la exportación con la palabra reservada **`default`** como se indica en el Snippet 33.

```
133
134  export default connect(mapStateToProps)(RouteMapContainer);
135
```

Snippet 33

- Una manera más intuitiva de aplicar la programación orientada a objetos en *JavaScript* con relación a versiones anteriores. Es decir, una mayor facilidad para declarar y extender clases. En la aplicación, cada componente heredará de la clase padre *Component* perteneciente de la librería de *React*, usando las palabras reservadas **`class`** y **`extends`**. En el Snippet 34 se visualiza la clase principal de la aplicación web desarrollada.

```
4   class App extends Component {
5     …
6   }
```

Snippet 34

Existen más características relativas al estándar EC6[85], tan solo se han mostrado algunas de las que han sido aplicadas en el proyecto ubicado en el repositorio de GitHub[86].

React

Es un marco de desarrollo creado por Jordan Walke en 2011, un desarrollador de software de Facebook, influido por XHP, un *framework* de componentes HTML para PHP. Sin embargo, no es hasta 2013 cuando el software se hace libre. Según la propia definición de la página web de React: «*A JavaScript Library for building user interfaces*», es decir, en líneas generales sería una librería JavaScript para la creación de interfaces de usuario.

Se usa *JSX* que es una extensión de *ECMAScript* sin definición semántica y con apariencia similar a HTML que proporciona una forma estructurada de describir el *rendering* de un componente. En resumidas cuentas, se aplica azúcar sintáctico, explicado más arriba, para la ejecución del siguiente comando:

```
20
21   React.createElement(component, props, ...children)
22
```

En la aplicación se usa, por ejemplo, para el método *render* de la clase App como se muestra en el ejemplo del Snippet 35:

```
26    export class App extends Component {
25
26     render() {
27       return (
28         <Router>
29         <div className="App">
30           {this.props.progress &&
31               <div className={"loading-div"}>
32               <CircularProgress style={{
33                                   color: blue[600],
34                                   }}
35                                 size={240}
36               />
37               </div>
38           }
39           <header className="App-header">
40             <h1 className="App-title">Route Analyzer</h1>
41           </header>
42           <div className="App-intro">
43             <FileUploaderContainer/>
44             {this.props.showRoute && <RouteContainer id={this.props.id}/>}
45           </div>
46         </div>
47         </Router>
48       );
49     }
50   }
```

Snippet 35

El texto marcado en negrita indica donde se ha hecho uso de *JSX*, por tanto, se aprecia que no es ni un *String* ni tampoco código *html*, es una descripción de cómo debería verse la interfaz gráfica[87] de la aplicación. Para distinguirse de las etiquetas incorporadas en *html*, los componentes creados con *JSX* siempre empiezan con letra mayúscula. Por otro lado, no forma parte de una propuesta para ser incorporado al estándar *ECMAScript* y tampoco de ser implementado en el motor de los navegadores web. En realidad, lo que

[85] http://es6-features.org/
[86] https://github.com/luisrodrigar/route-analyzer-web
[87] https://es.wikipedia.org/wiki/Interfaz_de_usuario

ocurre es que los preprocesadores, como Babel[88], lo transforman en lenguaje puro *ECMAScript*.

Una vez explicado cómo se puede describir el *renderizado* de un elemento, la base fundamental de este *framework* es el uso de componentes para representar cada parte de la interfaz de usuario que se va a desarrollar. Cada componente es independiente de los demás, es reusable y está aislado recibiendo una serie de atributos llamados propiedades y devolviendo elementos de React que describen lo que aparecerá en la interfaz.

A continuación, se muestra un componente, en el Snippet 36, que ha sido creado para permitir la descarga de ficheros. Este componente es reusable ya que es usado en dos ocasiones:

- Para poder descargar el archivo original de la ruta deportiva,
- Para exportar la ruta con los valores modificados a un determinado tipo de documento.

Las propiedades se definen en el constructor y se almacenan en la variable de clase *props*. Por otro lado, se define toda la lógica del componente, un método que permita la descarga del fichero desde la ruta de AS3 y, por otro, la manera de poder exportar la actividad deportiva con el formato que se seleccione.

```
13 export class DownloadFileComponent extends Component {
14
15   constructor(props){
16        super(props);
17        this.download = this.download.bind(this);
18.  }
19
20   download() {
21        get(this.props.id, this.props.type)
22           .then(response => {
23                  FileDownload(response, this.props.id + "_" + this.props.type + '.xml');
24        })
25        .catch(err =>{
26                  alert(err.message);
27        });
28.  }
29
30   export(type) {
31        exportAs(this.props.id, type)
32        .then(response => {
33                  FileDownload(response, this.props.id + "_" + type + '.xml');
34        })
35        .catch(err => {
36                  alert(err.message);
37        });
38   }
39
40   render() {
41        return(
42           <div>
43                  <button onClick={this.download}>Download file </button>
44                  <button onClick={()=>this.export('tcx')}>Export as TCX</button>
45                  <button onClick={()=>this.export('gpx')}>Export as GPX</button>
46           </div>
47        );
48   }
49 }
```

Snippet 36

Todo ello se realiza con un evento que generará el usuario de la aplicación como se puede ver en el método *render* (atributo *onClick*) donde se define toda la lógica de la capa UI.

[88] https://babeljs.io

Por lo tanto, el propio componente alberga una lógica de negocio y una lógica de la capa visual. Para poder añadir este elemento a otro componente, hacer uso de la composición de elementos, tan solo hay que definir, en este caso, el nombre del componente en *JSX* y definirlo en el *render* del componente donde se quiere añadir. En el ejemplo mostrado en el Snippet 36 será *<DownloadFileComponent/>*. Se entrevé que este componente no tiene ninguna propiedad ya que no tiene ningún atributo en la etiqueta.

Además de todo esto, los componentes también pueden tener un estado interno, con esto lo que se consigue es que se puedan actualizar sus valores en la UI sin que haya ningún elemento externo que lo gestione o realice. Se produce gracias a la variable *state*, una variable de la clase que almacena todos los valores del estado del componente. El estado puede ser definido en el constructor de cada componente de la siguiente forma en la que se especifica en el Snippet 37.

```
20   constructor(props){
21       super(props);
22       this.state = { keys: [], map:null }
23       this.calculateCenterZoom = this.calculateCenterZoom.bind(this);
24       this.handleInfoClose = this.handleInfoClose.bind(this);
25       this.handleMarkClick = this.handleMarkClick.bind(this);
26       this.handleDeletePoint = this.handleDeletePoint.bind(this);
27   }
```

Snippet 37

En negrita se marca la inicialización del estado del componente. Se realiza aquí la inicialización de los valores, por un lado, se define un *array* vacío llamado *keys* y, por otro, se crea el objeto mapa inicializándose con el valor *null*. Sin embargo, si se desea modificar alguna propiedad del objeto *state*, no se puede realizar de la misma forma como se hizo en el constructor. Se ha de utilizar *setState*, un método de la clase *Component* de React. Volviendo al ejemplo mostrado más arriba, en el Snippet 37, se podría modificar el valor de los objetos del estado de forma en la que se destaca en el Snippet 38.

```
37     handleInfoClose(key){
38         this.setState({
39             keys: this.getKeysWithoutKey(key)
40         });
41     }
42
43     handleMarkClick(key){
44       let index = this.state.keys.indexOf(key);
45       if(index===-1){
46         this.setState({
47           keys: [key, ...this.state.keys]
48         });
49       } else{
50         this.setState({
51             keys: [...this.state.keys.slice(0,index),
52                 ...this.state.keys.slice(index+1)]
53           });
54       }
55     }
56
57     handleDeletePoint(position, date, index, keyMarker){
58        this.props.handleRemoveMarker(position, date, index);
59        this.setState({
60                  keys: this.getKeysWithoutKey(keyMarker)
61       });
62     }
```

Snippet 38

Lo que ocurre después de la modificación de algún valor de objeto state, mediante el método *setState*, es la actualización de los campos a los que afecta que están presentes en la interfaz, en el *render*.

Por lo tanto, en todo componente existe un ciclo de vida, con una serie de métodos, además del método *render*, asociados a *React.Component* (se destacan en negrita):

- Creación del componente:
  constructor → ***componentWillMount()*** → *render()* → crear los componentes hijos → ***componentDidMount()***
- Re-renderizado debido al re-renderizado del padre:
  componentWillReceiveProps()→***shouldComponentUpdate()***→
  {true}→ ***componentWillUpdate()***→*render()*→enviar las nuevas propiedades a los hijos→***componentDidUpdate()***
  {false}→ no se realiza nada
- Re-renderizado debido a un cambio en el estado interno, uso de *setState()*:
  shouldComponentUpdate()→
  {true}→ ***componentWillUpdate()***→*render()*→enviar las nuevas propiedades a los hijos→***componentDidUpdate()***
  {false}→ no se realiza nada
- Forzar el re-renderizado llamando a fo*rceUpdate()*:
  componentWillUpdate()→*render()*→enviar las nuevas propiedades a los hijos→***componentDidUpdate()***
- Re-renderizado debido a la captura de un error:
  Parent *render()* →
  {no error}→ se envían las nuevas propiedades a los hijos→algún error en algún *render()* de los hijos → *parent* ***componentDidCatch()*** – se almacena el error
  {error}→ mostrar el mensaje del error

Redux

Es una librería de código abierto de *JavaScript* influenciada por el lenguaje de programación *Elm*[89]. Su primera versión se lanza en 2015 permitiendo gestionar un estado global de las aplicaciones desarrolladas con *JavaScript*. En el proyecto que se ha desarrollado, se ha usado esta librería junto a *React*, sin embargo, también es compatible con *Angular*. La finalidad de esta librería en ambos *frameworks* es la de hacer más fácil el desarrollo de interfaces de usuario y gestionar un estado único. En el estado global se puede guardar una gran variedad de información: respuestas del servidor, datos cacheados y/o datos creados dinámicamente que aun no han sido persistidos.

Se apoya en tres principios básicos:

- Una única fuente de la verdad
  Posibilidad de serializar el estado de la aplicación lo que trae una mayor facilidad para intercambiar el objeto del estado entre el cliente y el servidor. Algunas funciones como el deshacer/rehacer se vuelven muy fáciles de implementar al tener guardado en un único punto todo el estado de la aplicación.
- El estado es de solo lectura

[89] https://es.wikipedia.org/wiki/Elm_(lenguaje_de_programación)

La única forma de poder realizar una modificación sobre el estado global de la aplicación es lanzando una acción que contenga la información de lo que ha ocurrido.

- Cambios realizados con funciones puras
  Para especificar como el árbol de estados es transformado por las acciones se utilizan funciones llamadas *reducers*. Son métodos simples que toman como parámetros el estado anterior al aplicar una determinada acción y la acción en sí, siendo el valor que devuelve el estado de cómo quedaría la aplicación tras aplicar la acción indicada en la entrada de la función.

En la aplicación se ha implementado esta librería de la siguiente forma, para que todos los componentes puedan acceder al *store* de Redux es necesario que este elemento sea pasado entre sus propiedades, sin embargo, se vuelve muy tedioso cuando es un componente que está dentro de otros tantos teniendo que especificar entre sus propiedades el *store*. Por tanto, una librería que se llama *react-redux* tiene un componente especial llamado *Provider* que permite que, como por arte de magia, todos los componentes del contenedor en el que se define tengan acceso. En el proyecto se ha definido en el fichero /src/*index.js*.

```
9    ReactDOM.render(
10         <Provider store={store}>
11                <App/>
12         </Provider>,
13         document.getElementById('root')
14   );
```

Snippet 39

El componente Provider tiene como propiedades el *store* que contiene el estado global de la aplicación definido en el fichero */src/store/index.js* como se indica más abajo, en el Snippet 40.

```
1    import { createStore } from "redux";
2    import rootReducer from "../reducers/index";
3    const store = createStore(rootReducer);
4    export default store;
```

Snippet 40

Este estado puede ser la combinación de varios *reducers* gracias al uso de un método de Redux llamado *combineReducers* definido en el archivo */src/reducers/index.js* como se muestra en el Snippet 41.

```
1    import { combineReducers } from 'redux';
2    import app from './appReducer';
3    import container from './containerReducer';
4
5    const rootReducer = combineReducers({
6      app,
7      container
8    })
9
10   export default rootReducer
```

Snippet 41

A su vez, cada *reducer* contiene las reglas de modificación del estado global de la aplicación indicando en los parámetros el tipo de acción, definidos todos en el fichero /src/*constants/action-types.js*, como se puede ver en el Snippet 42.

```
1    export const TOGGLE_PROGRESS = 'TOGGLE_PROGRESS'
2    export const SHOW_ROUTE = 'SHOW_ROUTE'
3    export const SET_ACTIVITY = 'SET_ACTIVITY'
4    export const UPDATE_TRACKPOINT = 'UPDATE_TRACKPOINT'
```

Snippet 42

Además, también se incluye en los parámetros los nuevos valores de los atributos que se han de haber modificado una vez ejecutada la acción. Los *reducers* se encuentran definidos en */src/reducers/appReducer.js*, definido en el Snippet 43, que contiene todas aquellas operaciones que se pueden realizar en la aplicación que por ejemplo son la visualización o no del icono de procesamiento mientras que se realiza alguna acción en el *backend*.

```
import { TOGGLE_PROGRESS, SHOW_ROUTE } from '../constants/action-types';

    const initialState = {
         showRoute: false,
         progress: false,
         id:null
    }

    export default function appReducer(state = initialState, action) {
      switch (action.type) {
        case TOGGLE_PROGRESS:
          return {
            ...state,
            progress: action.isEnable
          }
        case SHOW_ROUTE:
                    return {
                              showRoute: action.id?true:false,
                        id: action.id,
                        progress: false
                    }
        default:
          return state
      }
    }
```

Snippet 43

Por otro lado, si el valor de mostrar alguna actividad deportiva es verdadero entonces se hará uso del *reducer* definido en */reducers/containerReducer.js*, mostrado en el Snippet 44, que tiene definidas todas las operaciones que se pueden realizar en el contendor, como por ejemplo cambiar las vueltas que están asociadas al estado de la aplicación o actualizar el punto de la ruta deportiva sobre el que se está realizando una operación de consulta o de edición.

```
    const initialState = {
         activity: {},
      indexLap:null,
      indexTrackpoint:null,
      laps:[],
      currentTrackpoint: null
    }

    export default function containerReducer(state = initialState, action) {
      switch (action.type) {
        case SET_ACTIVITY:
          const laps = getLapsTrackPoints(action.activity.laps);
          return {
            ...state,
            activity: action.activity,
            indexLap:null,
            indexTrackpoint:null,
            currentTrackpoint: null,
            laps: [...laps]
          }
        case UPDATE_TRACKPOINT:
          return {
            ...state,
            indexLap: action.indexLap,
            indexTrackpoint: action.indexTrackpoint,
            currentTrackpoint:
                state.laps[action.indexLap].tracks[action.indexTrackpoint]
          }
        default:
          return state;
```

```
33      }
34  }
```

Snippet 44

Las acciones que pueden ocurrir en el sistema y que podrán modificar el estado global están definidas en el fichero */src/actions/index.js* como se indica en el Snippet 45.

```
1   import { SHOW_ROUTE, TOGGLE_PROGRESS, SET_ACTIVITY, UPDATE_TRACKPOINT }
2   from '../constants/action-types';
3
4   export const showRoute                                  = id => ({
5         type: SHOW_ROUTE,
6         id: id
7   });
8   export const toggleProgress                   = isEnable => ({
9         type: TOGGLE_PROGRESS,
10        isEnable:isEnable
11  });
12  export const setActivity                      = activity => ({
13        type: SET_ACTIVITY,
14        activity: activity
15  });
16  export const updateTrackpoint                 = (indexLap, indexTrackpoint) => ({
17        type: UPDATE_TRACKPOINT,
18        indexLap:indexLap,
19        indexTrackpoint:indexTrackpoint
20  });
```

Snippet 45

La primera de ellas es *showRoute* e indica la actividad deportiva que hay que visualizar, esta ruta viene determinada por el identificador con el que se almacenó en la base de datos. La segunda de ellas, *toggleProgress* indica cuando hay o no que mostrar el icono de la carga de una actividad deportiva. La tercera acción, llamada *setActivity* permite definir el objeto contenedor de toda la información deportiva actual en memoria. Por último, la acción definida como *updateTrackpoint* permite asignar el punto actual que se ha seleccionado en el mapa o que se ha sido seleccionado mediante el movimiento del ratón sobre alguno de los diagramas. Por tanto, para que cada componente pueda obtener los nuevos valores del estado global y aplicar acciones sobre ellos, es necesario definir dos funciones y usar el método *connect* de *react-redux* para enlazarlas al componente. Estas funciones son:

- *mapStateToProps* para poder definir como propiedades de un componente ciertos campos del estado de *Redux*.
- *mapDispatchToProps* que permite realizar algo similar a lo que realiza la función anterior con la peculiaridad de que ahora lo que se agregan a las propiedades del componente son las acciones del estado global de *Redux*.

Por ejemplo, en el componente *RouteContainer* es posible hacer uso de *Redux* gracias a la creación de las funciones anteriormente descritas y el uso de *connect* para añadir todos estos valores a las propiedades de este componente como se muestra en el Snippet 46.

```
14  const mapStateToProps = state => {
15    return {
16      activity:             state.container.activity,
17      idLap:                          state.container.idLap,
18      idTrackpoint:         state.container.idTrackpoint
19    };
20  };
21
22  const mapDispatchToProps = dispatch => {
23    return {
24      setActivity: activity => dispatch(setActivity(activity))
25    };
26  };
27
28  class RouteContainer extends Component {
```

```
29
...
171  }
172
173  export default connect(mapStateToProps,mapDispatchToProps)(RouteContainer);
```

Snippet 46

Arquitectura

La arquitectura propuesta para solucionar este problema es la que se basa en un modelo de capas, conectadas cada una de ellas a través de una API web REST[90]:

- La capa de *backend* que procesará toda la información, gestionando los procesos que implican una mayor carga de trabajo, es aquí donde se realiza la persistencia de datos, accediendo a la base de datos o a servicios en la nube para el almacenamiento de ficheros.
- La capa de *frontend* encargada de permitir la entrada de datos del usuario, de presentar toda la información al usuario y de gestionar la visualización de los datos de forma óptima.

Cada capa está desacoplada, es decir, la una no depende de la otra, tan solo ambas capas han de cumplir el contrato de interfaz que se ha acordado previamente para que se puedan comunicar. Es un diseño más flexible que permite que en un futuro se puedan desarrollar aplicaciones en otras plataformas, como por ejemplo para aplicaciones móviles, o con otros lenguajes; posibilitando la reusabilidad de algunas de las capas existentes.

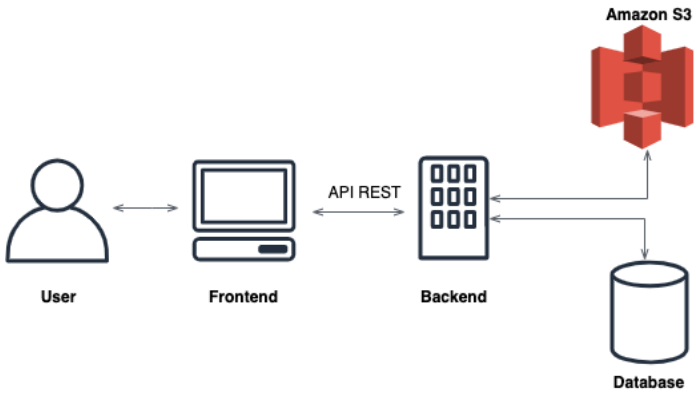


Ilustración 3

En la Ilustración 3 se muestra, de forma resumida, como está diseñada la arquitectura de la aplicación en términos generales, sin profundizar a nivel técnico, tan solo para tener una visión general del sistema. En este caso en particular, se ha decidido que el *backend* no esté formado por varios servicios pequeños ya que no tiene funcionalidades lo suficientemente bien diferenciadas para realizarlo, sin embargo, si en el futuro aumentaran este número de servicios, se podría dividir en dos o más microservicios[91] autónomos e independientes que faciliten el trabajo en equipo y la entrega continua[92], sin bloqueos. Más adelante se pondrá más énfasis en cada una de las partes del sistema.

[90] https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional
[91] https://es.wikipedia.org/wiki/Arquitectura_de_microservicios
[92] https://es.wikipedia.org/wiki/Entrega_continua

Vista del sistema

Para comenzar, con una visión muy abstracta, se va a explicar como está constituido el sistema. Principalmente, como ya se ha explicado, existen dos partes (los ficheros fuente de cada aplicación son descritos en el punto *Anexo II*):

- El *Componente Frontend* siendo el proyecto que contiene toda la lógica del diseño de la aplicación y la que le proporciona al usuario el acceso al sistema.
- El *Componente Backend* que es aquel proyecto que tiene toda la aplicación del lado del servidor.

Ambos sistemas están comunicados mediante los *endpoints* expuestos en la API que ha sido desarrollada en lado del servidor utilizando las herramientas descritas en el apartado *Tecnología*. La comunicación se realiza mediante el protocolo de comunicación http siendo un servicio web REST[90]. La propia aplicación de *backend* usa una herramienta llamada *Swagger*, descrita en dicho apartado de más arriba, que permite conocer a través de una interfaz web, accesible mediante el acceso a este enlace https://route-analyzer-api.herokuapp.com/swagger-ui.html, todos los *endpoints* definidos en la aplicación.

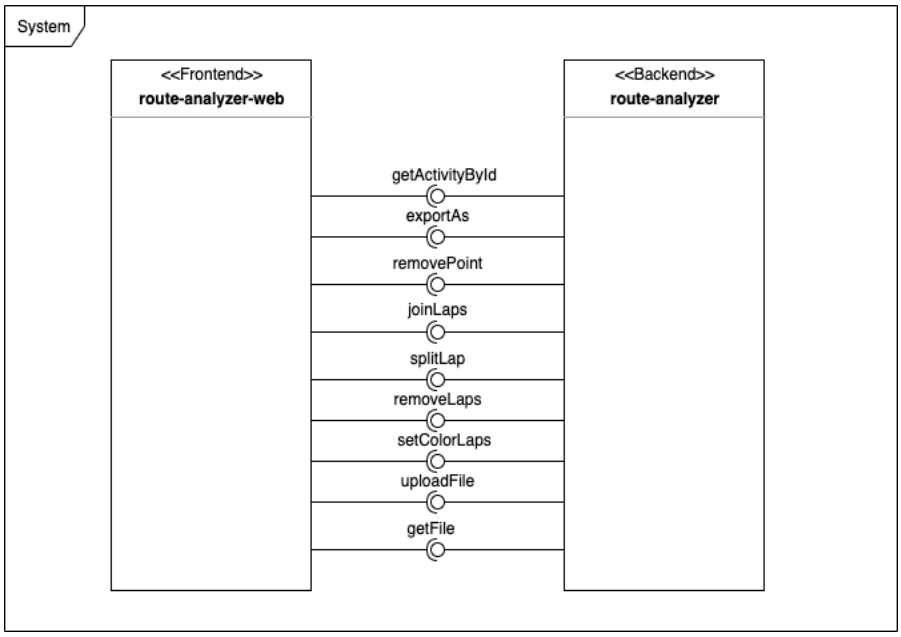


Ilustración 4

En la Ilustración 4 se muestra como la interfaz del lado del cliente consume los servicios proveídos por la aplicación del lado del servidor. Ofreciendo una determinada funcionalidad en función del servicio solicitado por parte del cliente. Para acceder a cada una de las interfaces es necesario utilizar el verbo HTTP apropiado y concatenar al *host* los siguientes *paths (en minúscula)*:

- getActivityById – GET
  /activity/{id}
- exportAs – GET
  /activity/ {id}/export/{type}
- removePoint – PUT
  /activity/{id}/remove/point?lat={latitud}&lng={longitud}&timeInMillis={time}&index={index}

- joinLaps – PUT
  /activity/{id}/join/laps?index1={indexLap1}&index2={indexLap2}
- splitLap – PUT
  /activity/{id}/split/lap?lat={latitud}&lng={longitud}&index={index}&timeInMillis={time}
- removeLaps – PUT
  /activity/{id}/remove/laps?date={startTimeLaps,...}&index={indexLaps,...}
- setColorLaps – PUT
  /activity/{id}/color/laps?data={data}
- uploadFile – POST
  /file/upload
- getFile – GET
  /file/get/{type}/{id}

A continuación, se detalla como se comporta cada uno de los componentes que conforman todo el sistema a nivel interno.

Componente Backend

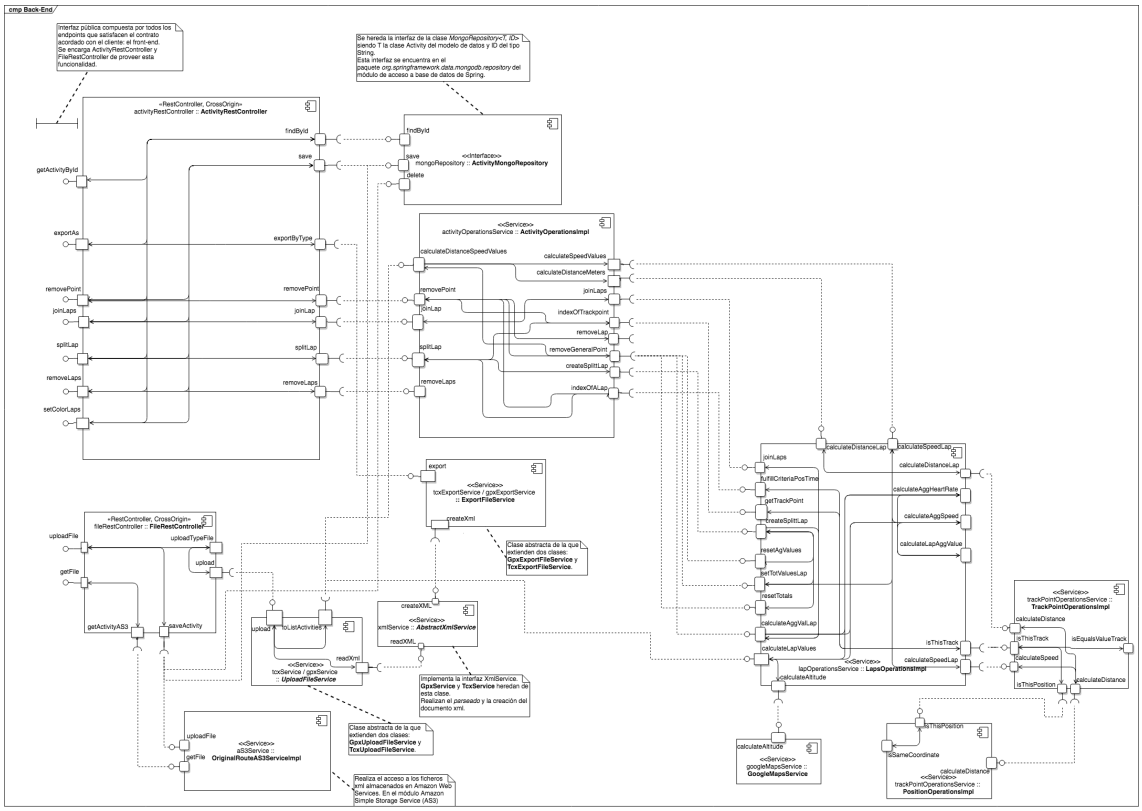


Ilustración 5

Más adelante se comentará en el apartado *Diseño* más cuestiones sobre toda esta arquitectura de componentes. Los componentes que modelan todos estos elementos son los que se muestran en la Ilustración 5:

- Proveedores de los servicios web:
  - *ActivityRestController*
  - *FileRestController*
- Procesan la información:
  - *ActivityOperationsImpl*
  - *LapsOperationImpl*

    - *TrackPointOperationsImpl*
    - *PositionOperationsImpl*
    - *UploadFileService*
        - *GpxUploadFileService*
        - *TcxUploadFileService*
    - *ExportFileService*
        - *GpxExportFileService*
        - *TcxExportFileService*
- Almacenan la información en base de datos:
    - *ActivityMongoRepository*
- Leen/escriben de/en el XML:
    - *AbstractXmlService*
        - *GpxService*
        - *TcxService*
- Acceden a servicios de terceros:
    - Persistencia de documentos
        - *OriginalActivityS3Service*
    - Servicios de localización
        - *GoogleMapsApiService*

Componente Frontend

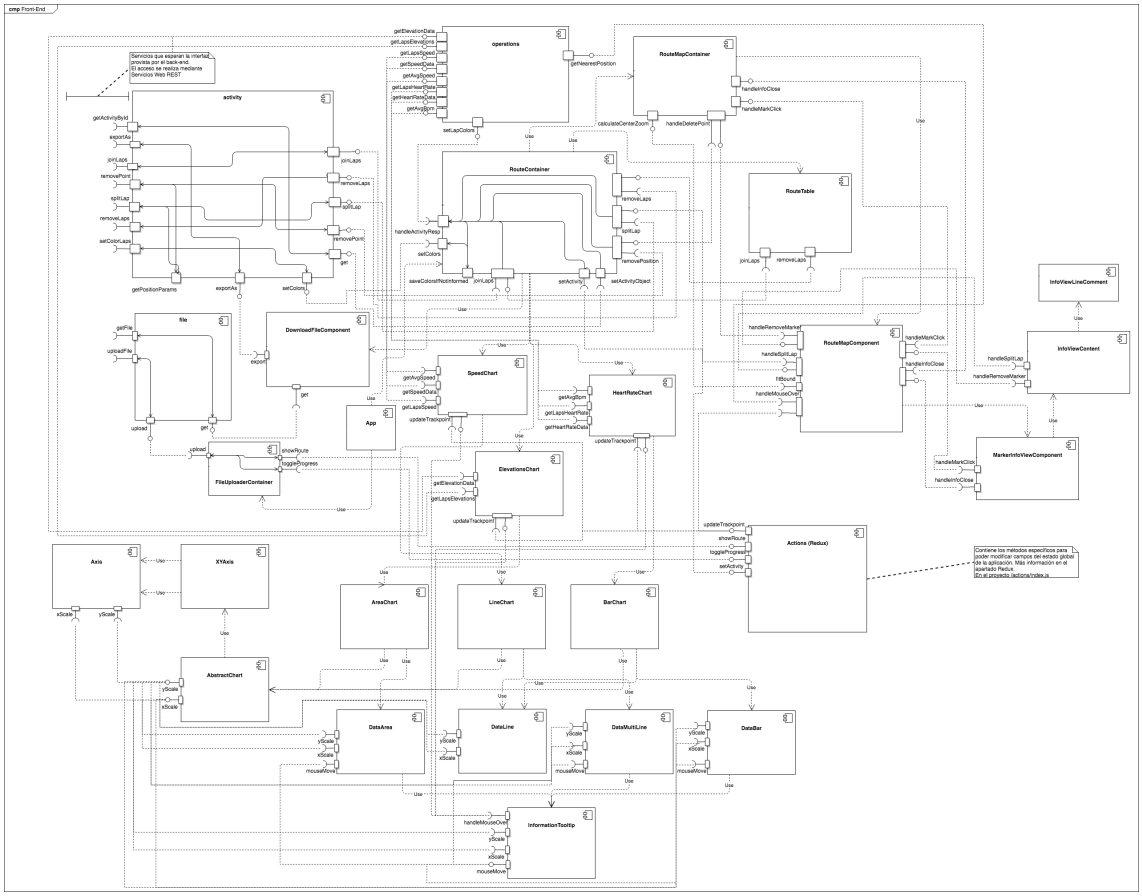


Ilustración 6

En este caso, como se muestra en la Ilustración 6, se ha representado el diagrama de componentes de la estructura de la aplicación de cliente desarrollada con *React*. Cada componente usará otros componentes (a nivel interno esto se traduce en que un componente puede estar formado por uno o varios componentes).

A continuación, se van a mostrar los elementos ordenados estructuralmente. Hay que recordar que uno de los puntos fuertes de este *framework* es la posibilidad de reusar elementos, por lo tanto, cada uno de ellos es independiente de los demás y se pueden probar de forma independiente facilitando el desarrollo de pruebas. El elemento *App* es el componente principal y el nodo raíz del que cuelgan los demás elementos que conformarán la aplicación:

1. *App*
    a. *FileUploaderContainer*
    b. *RouteContainer*
        i. *DownloadFileComponent*
        ii. *RouteMapContainer*
            1. *RouteMapComponent*
                a. *MarkerInfoViewComponent*
                    i. *InfoViewContent*
                        1. *InfoViewLineComment*
        iii. *RouteTable*
        iv. *ElevationsChart*
            1. *AreaChart*
                a. AbstractChart
                    i. XYAxis
                        1. Axis
                        2. Axis
                    ii. DataArea
                        1. InformationTooltip
        v. HeartRateChart
            1. BarChart
                a. AbstractChart
                    i. XYAxis
                        1. Axis
                        2. Axis
                    ii. DataLine
                    iii. DataBar
                        1. InformationTooltip
        vi. SpeedChart
            1. LineChart
                a. AbstractChart
                    i. XYAxis
                        1. Axis
                        2. Axis
                    ii. DataLine
                    iii. DataMultiLine
                        1. InformationTooltip

Existen elementos que no son considerados componentes, si no que su tarea principal es ofrecer funcionalidad o servicios a los componentes en sí. A continuación, se enumeran por funcionalidad algunos de los siguientes elementos que funcionan de esta manera y que aparecen en el diagrama de la Ilustración 6:

- Conexión con el Backend mediante servicios web REST:
    - *activity*
    - *file*
- Operaciones con los datos:
    - *operations*
- Proveedor de métodos que modifiquen estado global de la aplicación (*Redux*):
    - *Actions*

Diseño

A continuación, se va a mostrar el diseño sobre el que se ha trabajado para realizar la aplicación. Una vez que ya se ha visualizado en el apartado *Arquitectura* todo el sistema de forma abstracta, a continuación, se van a visualizar los diagramas u otros recursos para facilitar la transmisión de conocimiento de cómo se ha diseñado cada una de las partes del sistema.

Backend

Modelo

Una vez que tenemos la información extraída del documento, es necesario guardarla en un modelo de datos común para los diferentes tipos de esquema de ficheros con los que trabaja el sistema.

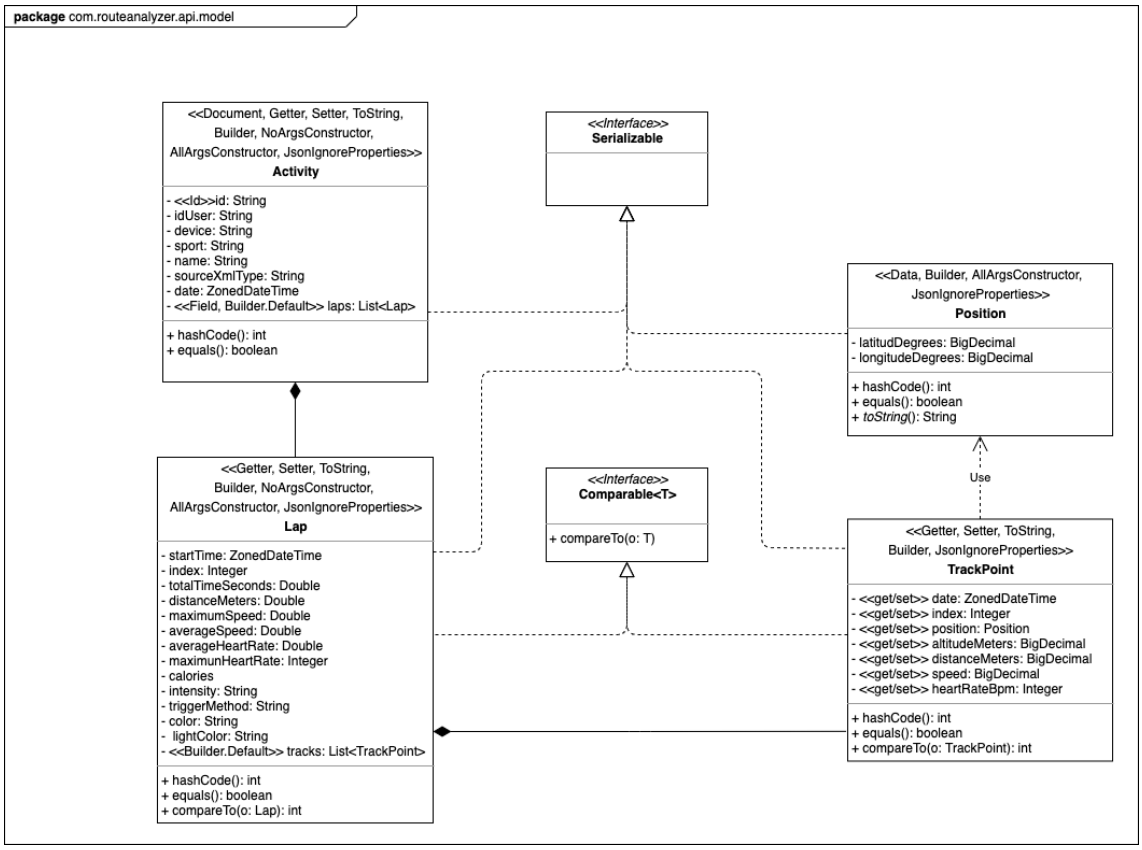


Ilustración 7

En la Ilustración 7 se muestran las clases que representan el dominio del problema. En concreto, toda la información será guardada en la clase principal *Activity*. Aquí se han añadido los siguientes estereotipos: *Document*, *Id* que representan anotaciones para saber cómo tiene que mapear los elementos el módulo de *Spring Data* para que coincidan con la información representados en la base de datos no relacional.

Además, también se definen estereotipos para los métodos *Getter* y *Setter*, al principio de la clase, ahorrando tiempo en codificar todos estos métodos que serán autogenerados por Lombok en tiempo de compilación.

Por otro lado, una actividad podrá contener tantas instancias de la clase *Lap* como vueltas se hayan definido en la actividad, dicha relación es de composición, es decir, si se borra una actividad, todas las vueltas asociadas dejan de tener sentido por sí solas. Además, cada *Lap* tendrá tantas instancias de la clase *TrackPoint* como puntos de localización haya sido definidos por el dispositivo electrónico encargado de guardar la información geoespacial en cada momento. Al igual que ocurría en el anterior caso, una vuelta tendrá un número determinado de instancias de la clase *TrackPoint*, pero si se borra una vuelta, todos estos puntos dejan de tener sentido por sí solos.

Las excepciones personalizadas de las que se hablo anteriormente están diseñadas como aparece en la Ilustración 8. Permiten identificar desde la clase definida con la anotación *@RestControllerAdvice*, descrita en el punto *Controlador* del apartado *Implementación*, como actuar en función del error que se haya producido.

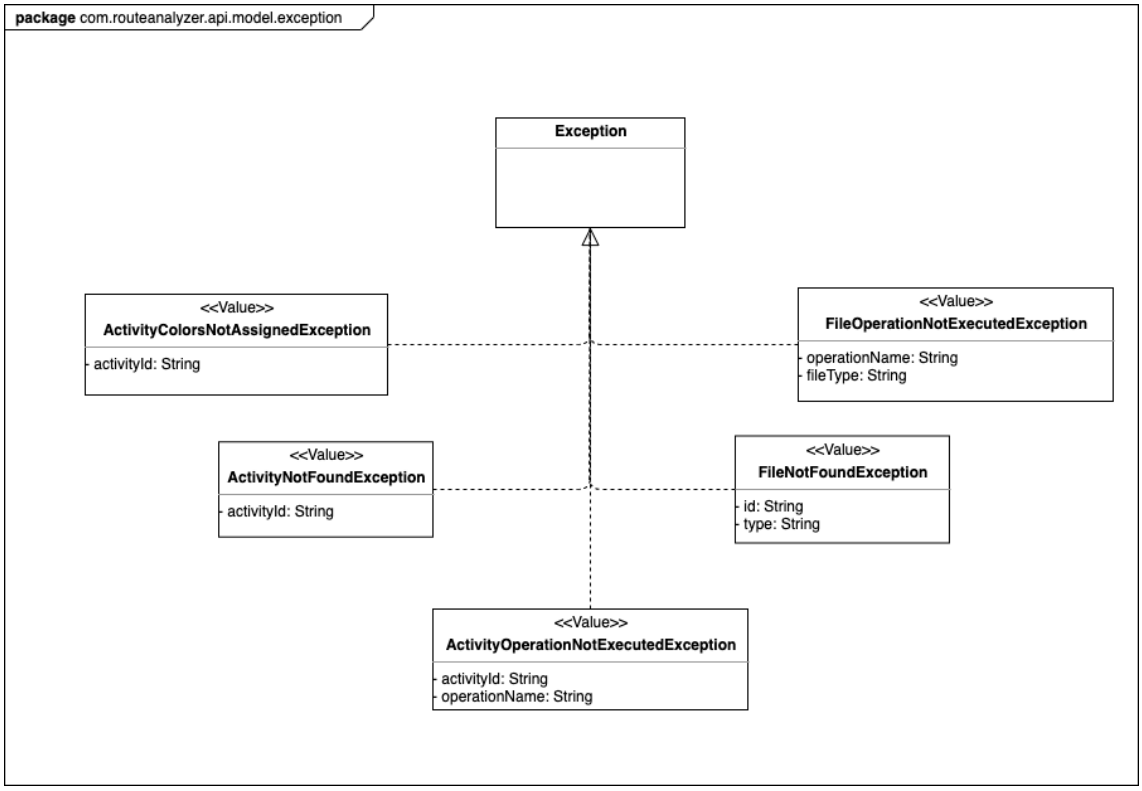


Ilustración 8

Controlador

El controlador es aquel componente que actúa como intermediario entre todas aquellas peticiones http que se realizan y la lógica de la aplicación. En el sistema existen dos controladores que responden en función del tipo de acción que se solicita. Para todas aquellas operaciones que tienen que ver con los ficheros existen el controlador llamado será *FileRestController*, mientras que, todas las acciones que se aplican a las actividades son gestionadas desde el controlador llamado *ActivityRestController*.

En un inicio el sistema estaba diseñado de la forma que aparece en la Ilustración 9, sin embargo, tras una actualización[93] en el repositorio, se modificó la forma en la que estaba diseñada esta capa de la aplicación.

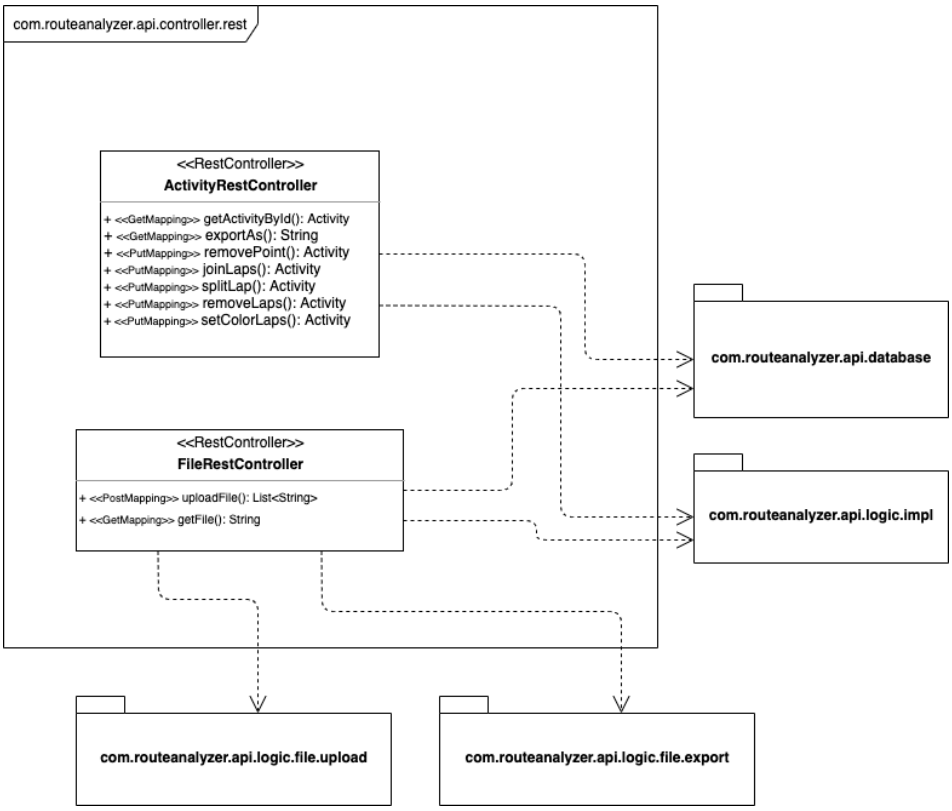


Ilustración 9

El problema que surge con el diseño definido en la Ilustración 9 es que en la capa del controlador se realiza mucha lógica que hace que quede mezclada la razón de ser de esta clase. Por tanto, para solventar este problema se hace uso del patrón fachada[94] que estructura el código para que las dependencias y todas las comunicaciones con otras clases estén focalizadas en una que implemente toda esta lógica, lo que permitirá que cualquier tarea de mantenimiento en el futuro sea más fácil y clara de llevar a cabo. Además, siguiendo el principio de Inversión de Dependencias de los principios SOLID[95]

[93] https://github.com/luisrodrigar/route-analyzer/commit/9c72f63e159e3d569b72d822867851f49ff0c9f7#diff-6be6dcd74e22cf7f1a07e0fedb3ba4ef
[94] https://es.wikipedia.org/wiki/Facade_(patrón_de_diseño)
[95] https://es.wikipedia.org/wiki/SOLID

que dice que hay que depender de abstracciones en vez de detalles, se realiza el diseño de la nueva estructura de clases como se representa en la Ilustración 10.

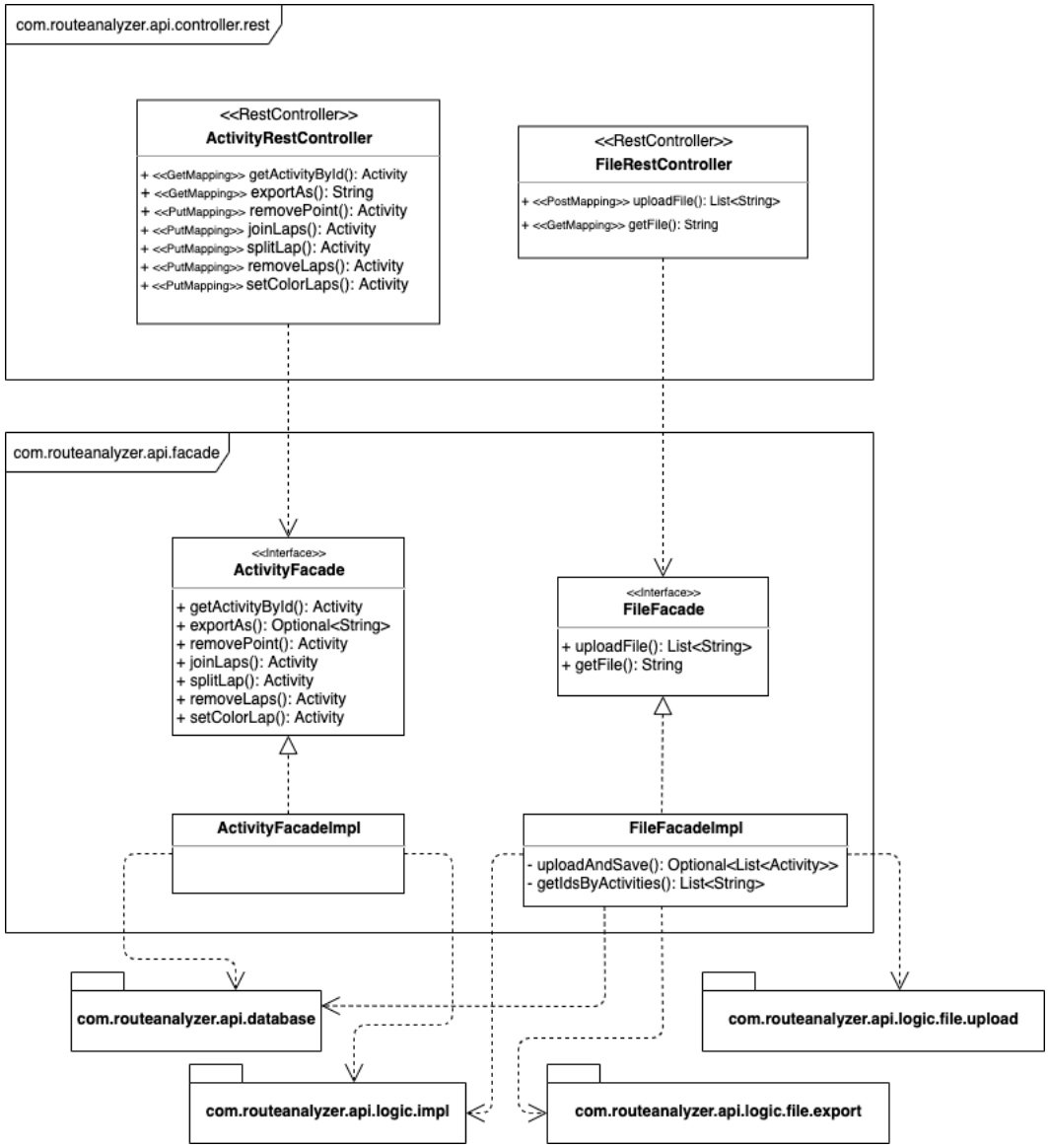


Ilustración 10

Lectura y Escritura

El Sistema tiene que hacer uso de los ficheros xml definidos en el punto *Ficheros de actividades deportivas* del apartado *Aspectos Teóricos*. Para ello, se ha de diseñar un modelo de clases que permita la lectura de actividades deportivas y la escritura de una ruta a un fichero xml. Con todo esto en mente, se ha de diseñado un lector que permita la lectura y escritura de los documentos con los que el sistema trabajará, como se comentará en el aparatado *Implementación*, se hará uso de la librería JAXB.

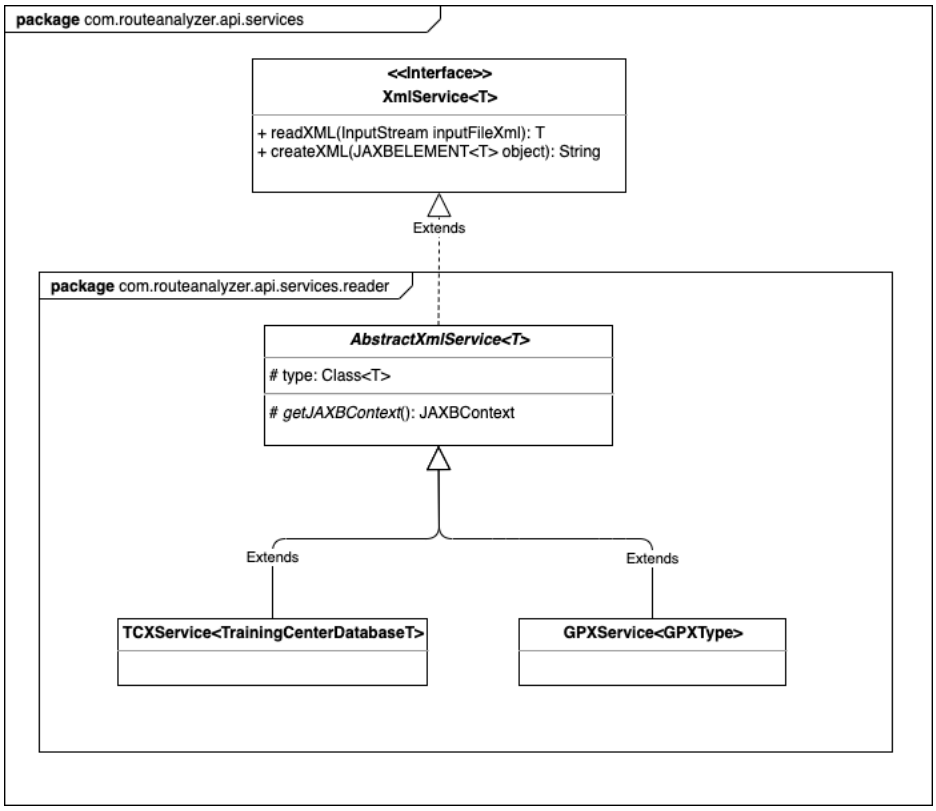


Ilustración 11

Como podemos ver en la Ilustración 11, la interfaz *XMLService* define la signatura de un método, llamado *readXml*, necesario para poder obtener el objeto de tipo T que será definido por las clases hijas que hereden de la genérica. Contendrá toda la información presente en el archivo *xsd*. Por otro lado, también se permitirá generar un documento *xml*, a través del otro método llamado *createXml* definido en la interfaz, que contenga toda la información de la ruta representada en los objetos de las clases del modelo de datos definido en el paquete **`com.routeanalyzer.api.xml`** para posteriormente ser convertido a texto plano.

Para ello, siempre pensando en maximizar la reutilización de código, se ha creado una clase abstracta que implementa el comportamiento de los métodos de la interfaz y que, por otro lado, define la signatura de un método abstracto denominado *getJAXBContext*. Gracias a este método se puede implementar el patrón de diseño Template[96], que permite, de una forma no compleja, definir un esquema común para que sus clases hijas redefinan

[96] https://es.wikipedia.org/wiki/Patrón_de_método_de_la_plantilla

el comportamiento interno de los procesos que contienen este esquema. el método abstracto lo único que hace es retornar el contexto con el que hay que trabajar y que estará definido por el tipo de esquema que tenga el documento *xml*. Estas clases genéricas son las clases raíz obtenidas anteriormente con el comando *xjc* de la librería de manipulación de *xml* de Java del que se hablara mas adelante en el punto *Modelo Ficheros XML* del apartado *Implementación*.

Si se desea en el futuro añadir otro tipo de esquema *xml* como descriptor de una actividad deportiva se necesitaría:

- Obtener el esquema que contiene la estructura del nuevo formato de documento.
- Generar las clases contenedoras de la información como se indicó al inicio de este punto.
- Crear una clase que extienda de la clase *AbstractXmlService*.
    - Definir las clases que utiliza en el constructor, siendo T la clase principal que contenga toda la información de la actividad deportiva.
    - Redefinir el método abstracto *getJAXBContext* para que devuelva las nuevas clases raíz que representan el nuevo esquema.
- Crear un Mapeador específico para este nuevo tipo de esquema que convierta los datos al modelo general de la aplicación.

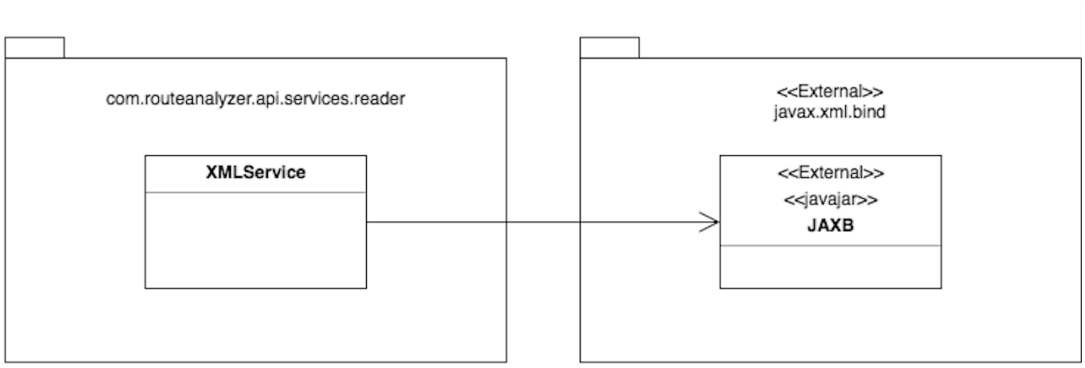


Ilustración 12

En la Ilustración 12, se observa que el sistema de lectura y escritura *xml* hace uso de la clase incluida en la librería nativa de Java usada para poder obtener la información de una actividad o para exportar alguna ruta que se haya estado modificando.

Mapeador

Una vez que se dispone de la lógica que permite la lectura de los diferentes tipos de esquemas de ficheros *xml* y de la creación de ficheros de este tipo a partir de las múltiples clases que modelan los diferentes esquemas, tan solo falta definir una serie de clases que permitan realizar el mapeo entre los datos de los ficheros *xml* y las clases del modelo generado descrito en el punto *Modelo* de este mismo apartado.

Fichero xml a modelo del sistema

Para poder convertir el fichero leído a las clases del modelo que definen una actividad física, independientemente del esquema que tenga, se usa un mapeador que permita convertir la información del fichero xml a las clases definidas en el paquete **`com.routeanalyzer.api.model`**. Ambas clases hijas, tienen métodos privados que ayudan a realizar todo el mapeo de las clases del esquema *xml* al modelo global del sistema, siguiendo el principio de una sola responsabilidad. Para ello, también hace uso de otras clases que permiten realizar operaciones de lógica y de instanciación: ActivityOperations, LapsOperations y TrackPointOperations definidas en el paquete **`com.routeanalyzer.api.logic`**.

En la Ilustración 13 se puede ver cómo está diseñado el diagrama de clases, contenidas en el paquete **`com.routeanalyzer.api.logic.file.upload`**, para la realización del procesado y subida de un fichero de actividad deportiva al sistema. Al igual que ocurrió antes, también se hace uso del patrón *Template Method* para que el código sea reutilizable y fácil de mantener, posibilitando la inclusión de otro tipo de esquema en el futuro. Por lo tanto, en la clase abstracta se define el método *toListActivities* abstracto y las clases hijas lo implementará, definiendo cada una de ellas un procesamiento diferente en función del tipo de esquema con el que se está tratando.

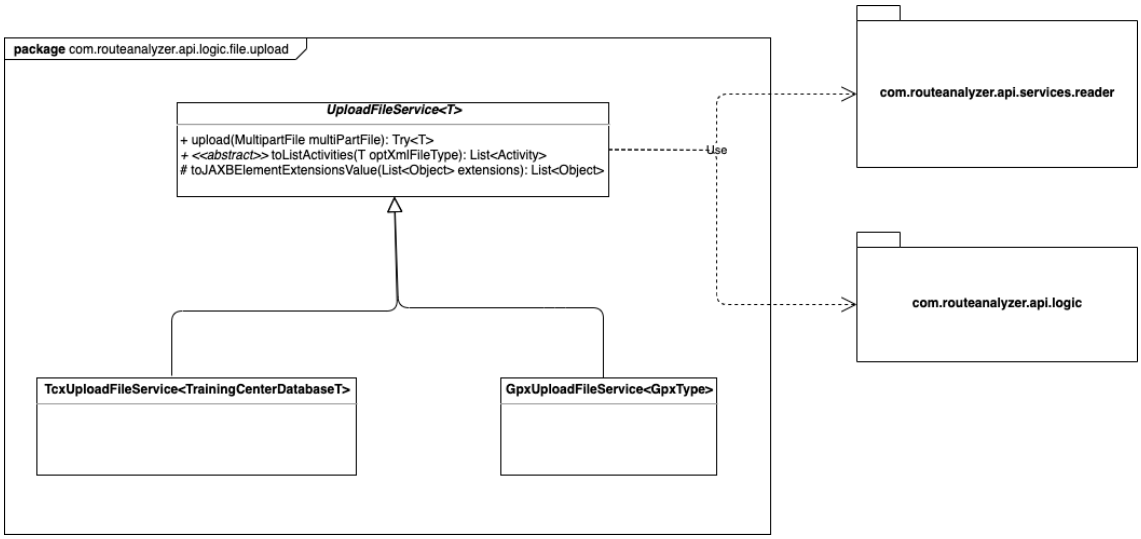


Ilustración 13

Modelo a fichero xml

Como consecuencia de que también se tiene que poder exportar una actividad modificada a los tipos de esquemas comentados en el punto *Ficheros de actividades deportivas*, es

necesario diseñar una forma que permita realizar el proceso inverso de lo que se ha comentado en el punto anterior *Fichero xml a modelo del sistema*.

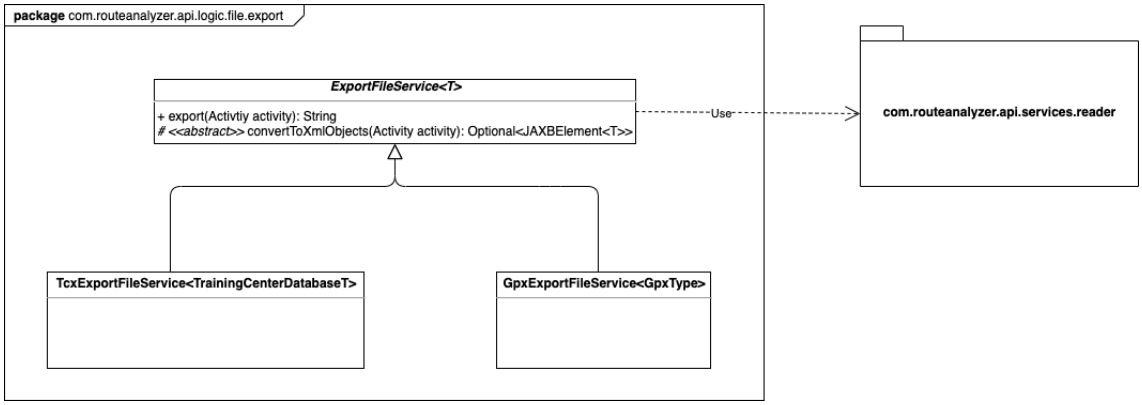


Ilustración 14

El modelo planteado, contenido en el paquete **`com.routeanalyzer.api.logic.file.export`**, es muy similar al que se ha desarrollado en el punto anterior, como se puede ver en el Ilustración 14. El diseño define una clase abstracta que tendrá la lógica común a la exportación de una actividad deportiva a un fichero *xml*. La lógica que cambia en función del tipo de fichero será implementada en el método abstracto *convertToXmlObjects* de cada clase hija que representa un tipo de fichero diferente.

Servicios Web de terceros

Obtener altitud en un punto

Los documentos del sistema contienen información de una actividad deportiva que pueden venir sin datos importantes, en estos casos, tomando como punto de partida los valores ya existentes se puede obtener la información que falta. Con esta premisa en mente, se obtiene la altitud en función de las coordenadas espaciales relativas a un punto geográfico, es decir, la latitud y la longitud. Para ello, se usa la API de *Google Maps* que proporciona una forma sencilla de obtener la altitud usando uno de los servicios públicos que tiene expuesto como *endpoint*.

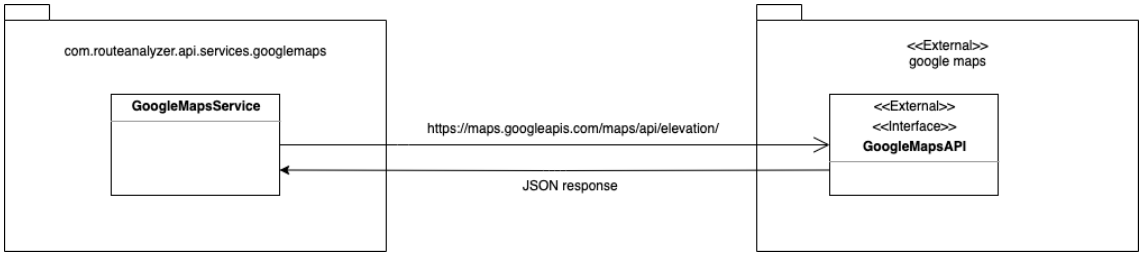


Ilustración 15

En el Ilustración 15 se representa un diagrama de paquetes en el que se visualiza la manera de obtener la información comentada más arriba. En la Ilustración 16 se define el diagrama de clases que permite tener la funcionalidad descrita en este punto. Para ello, se utiliza la clase de Spring *RestTemplate* para poder llamar al *endpoint* del servicio web de Google indicándole al método *getForObject* como parámetros la URL del servicio y la clase del objeto que representará la respuesta con la información relativa a las coordenadas y la altitud asociada, en este caso *GoogleMapsAPIResponse*. Este objeto contendrá tantas instancias de la clase *GoogleMapsAPIResult* como posiciones correctas a la entrada del método *getAltitude* existan.

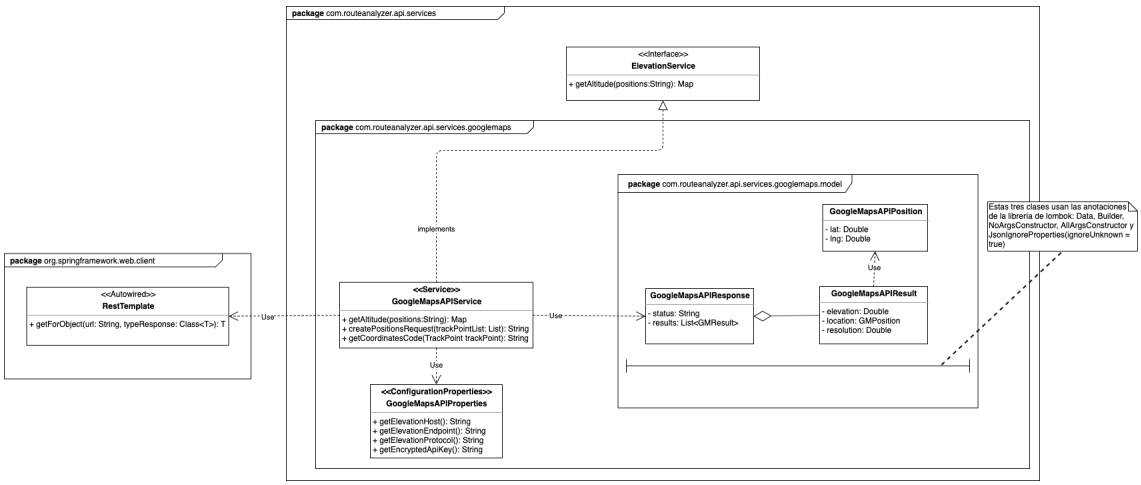


Ilustración 16

Guardar documentos originales

Una vez que la información de una actividad deportiva ha sido almacenada en la base de datos de forma satisfactoria, la tarea posterior es la persistencia del fichero original, sin ninguna modificación, para que en el futuro sea posible su descarga.

Gracias al uso del servicio Amazon Simple Storage Service, abreviado como S3, de la plataforma en la nube Amazon Web Services[97] es posible persistir el documento usando, tan solo, una serie de clases pertenecientes a una librería externa desarrollada por Amazon que facilita las tareas de envío y recepción de ficheros. Existen multitud de implementaciones de este *sdk*[98], en inglés *Software Development Kit*, para los diferentes lenguajes de programación como por ejemplo Node.JS, Python, Java, entre otros. En el proyecto se ha usado el propio para el último lenguaje que es el que se está usando para desarrollar la aplicación *backend*.

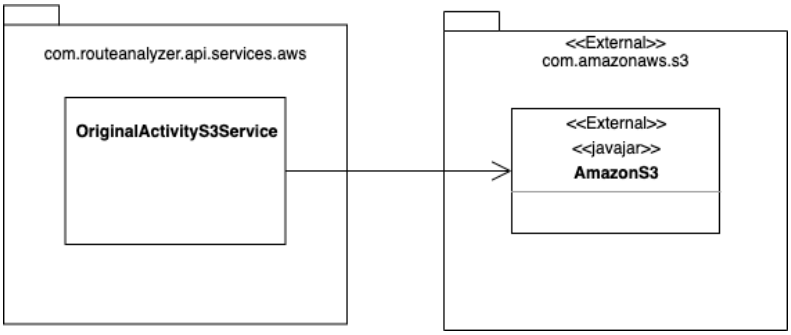


Ilustración 17

Como se puede ver en la Ilustración 17, la aplicación hace uso de una serie de clases pertenecientes a la librería externa de Amazon. Las clase *AmazonS3* se encuentra en el paquete de la librería **`com.amazonaws.s3`**, mientras que, las clases *GetObjectRequest*, *ObjectMetadata*, *PutObjectRequest* y *S3Object* pertenecen al paquete **`com.amazonaws.s3.model`**.

[97] https://es.wikipedia.org/wiki/Amazon_Web_Services
[98] https://es.wikipedia.org/wiki/Kit_de_desarrollo_de_software

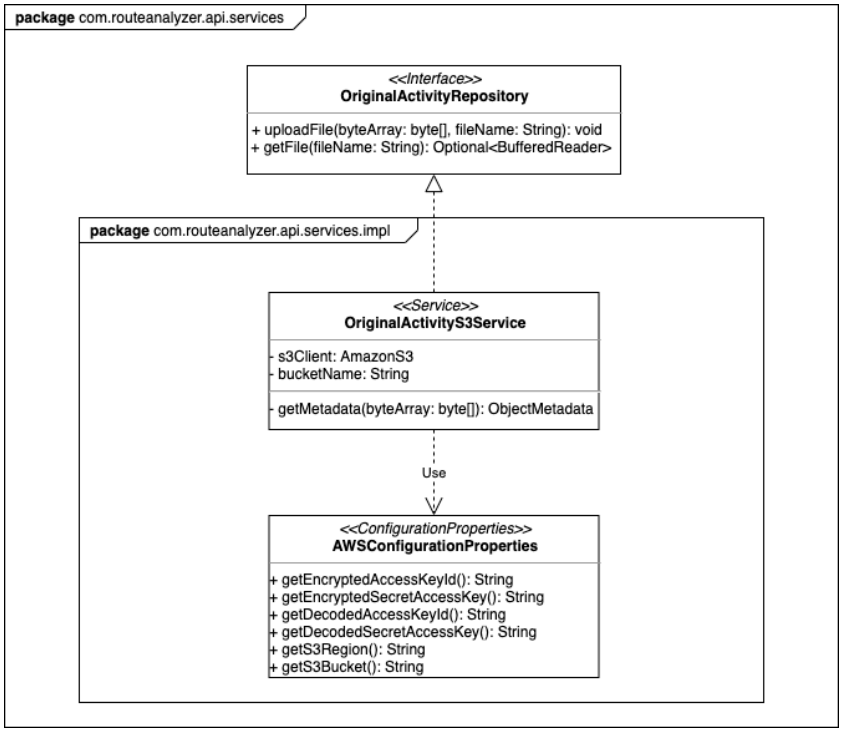


Ilustración 18

Además, también se tiene en cuenta que se puedan producir excepciones de la clase *AmazonClientException* en el paquete **`com.amazonaws`** para aquellos casos en los que no se haya podido acceder al servicio o que haya fallado la conexión de la aplicación con la plataforma de este cliente de la nube.

En la Ilustración 18 se muestra el diseño de esta parte de la aplicación, mostrándose la interfaz *OriginalActivityRepository* y la clase que hereda de ella denominada *OriginalActivityS3Service* que se encarga de usar las clases de Amazon S3 para realizar la subida y descarga de los ficheros previamente almacenados.

Para estar en disposición a realizar todo esto y en términos ya de *Implementación*, es necesario que se haya definido la clase de configuración definida en el Snippet 16, la clase en la que se inyectan los valores de las propiedades que se define en el Snippet 17 y el trozo de propiedades que se necesitan para hacer funcionar el sistema y que se definieron atrás en el Snippet 18.

Frontend

En el punto anterior se ha hablado del diseño de uno de los subsistemas de la aplicación que pertenece a la capa del servidor. A continuación, en la Ilustración 19, sin mostrar componente de librerías externas, se visualizan todos los componentes de la aplicación *frontend* del sistema.

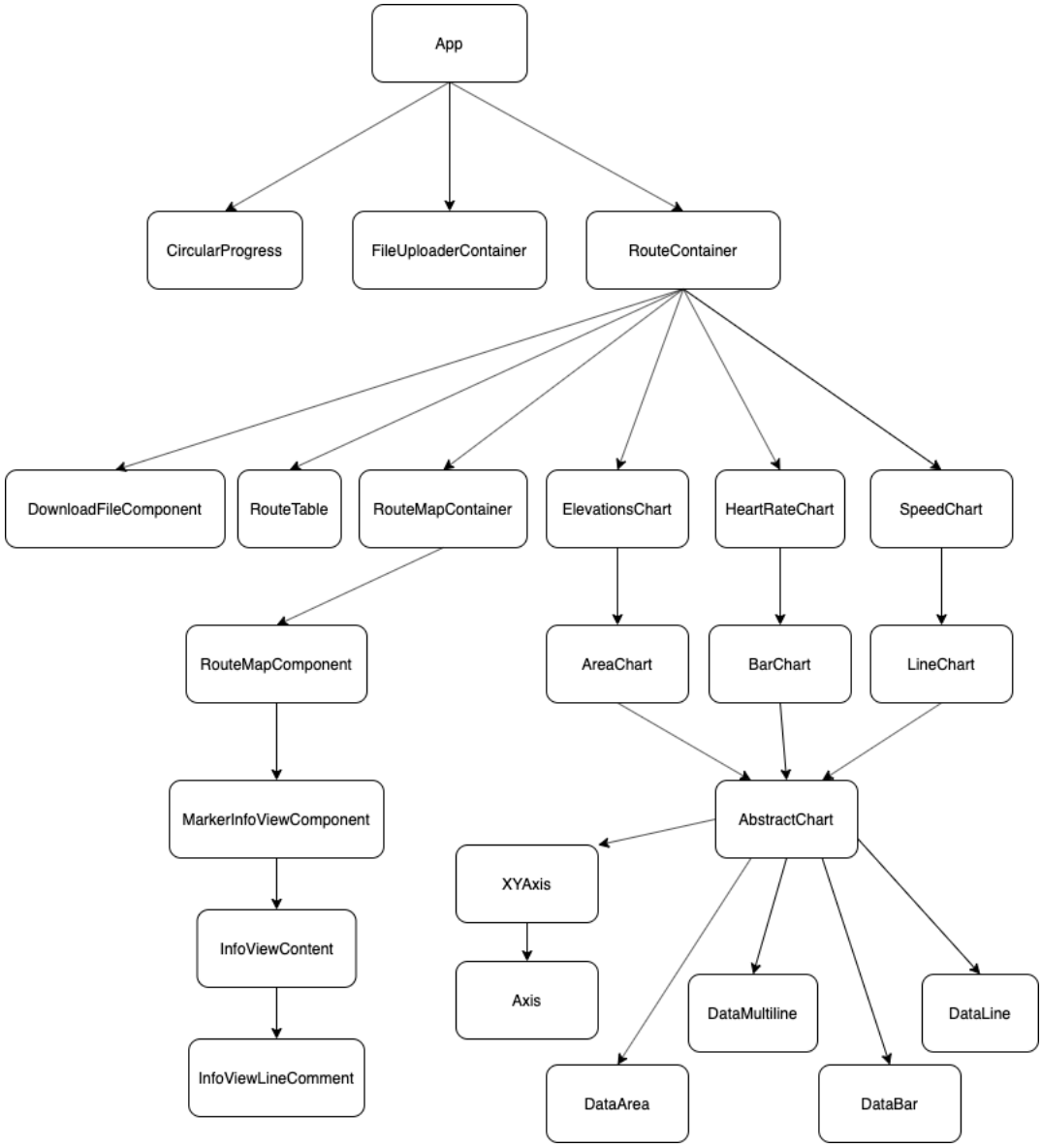


Ilustración 19

De los tres componentes hijos del elemento raíz que se muestran más arriba, el primero *CircularProgress* visualiza un icono cada vez que se está llevando a cabo alguna operación, el segundo, identificado como *FileUploaderContainer*, es el que se encarga de visualizar el formulario para poder enviar el fichero que contiene la actividad deportiva y el tercero, llamado *RouteContainer*, es el encargado de visualizar la ruta de diversas formas. En el punto *Frontend* del apartado *Implementación*, se va a hablar más en profundidad de las partes más cruciales de este último componente.

Implementación

Backend

Controlador

En el apartado de Diseño *Diseño*, en el punto *Controlador*, se ha definido la implementación del patrón Fachada. Por ejemplo, en el código antiguo de la clase *FileRestController* se definía toda la lógica que interactuaba con otras clases como se muestra en el Snippet 47. Las tareas que se realizaban en esta clase eran por un lado las comprobaciones del tipo de fichero en el método *uploadTypFile* o por otro se realizaba en el método *getActivityAS3* la llamada a la clase que gestiona el repositorio de ficheros de Amazon, devolviendo el fichero que se obtiene transformado en un *String*.

```
37      @PostMapping("/upload")
38      public List<String> uploadFile(@RequestParam("file") MultipartFile multiPart,
                                       @RequestParam("type") String type) {
39          return uploadTypeFile(type, multiPart)
40                  .orElseThrow(() -> new IllegalArgumentException(BAD_TYPE_MESSAGE));
41      }
42
43      @GetMapping("/get/{type}/{id}")
44      public String getFile(@PathVariable final String id, @PathVariable final String type,
45                            HttpServletResponse response) {
46          setExportHeaders(response, id, type);
47          return getActivityAS3(getFileName(id, type));
48      }
49
50      private Optional<List<String>> uploadTypeFile(String type,
                                       MultipartFile multipartFile) {
51          return Match(type).option(
52                  Case($(is(SOURCE_TCX_XML)), tcxType ->
                            activityOperations.uploadAndSave(multipartFile, tcxService)),
53                  Case($(is(SOURCE_GPX_XML)), gpxType ->
                            activityOperations.uploadAndSave(multipartFile, gpxService)))
54                  .toJavaOptional();
55      }
56
57      private String getActivityAS3(String fileName) {
58          return aS3Service.getFile(fileName)
59                  .map(InputStreamReader::new)
60                  .map(BufferedReader::new)
61                  .map(BufferedReader::lines)
62                  .map(streamLines -> streamLines.collect(joining("\n")))
63                  .orElse(null);
64      }
65
66      private String getFileName(String id, String type) {
67          return format("%s.%s", id, type);
68      }
```

Snippet 47

Por lo tanto, una vez que se ha aplicado el patrón de diseño se consigue extraer responsabilidades quedando clases más sencillas y fáciles de comprender. La clase donde se llevaron todos estos métodos será la clase fachada y en este caso se ha llamado *FileFacadeImpl*. Esta clase, por tanto, gestiona todas las comunicaciones con los objetos de las clases dependencia como se muestra en el Snippet 48. Como se comento en el apartado de *Diseño*, se hace uso de las abstracciones de las dependencias y no de las implementaciones. Es por ello por lo que la dependencia con la que se gestionan las actividades *tcx* se realizan con la interfaz *TcxUploadFileService*, al igual que ocurre con la gestión de las actividades *gpx* con la interfaz *GpxUploadFileService*. Sucede igual con el objeto el cual gestiona todas las actividades en la base de datos, la interfaz llamada *ActivityMongoRepository*, y con la clase que realiza todas las operaciones con las

actividades, identificada la interfaz como *ActivityOperations*. Las implementaciones serán inyectadas una vez que se inicialice la aplicación gracias al uso del *framework* de Spring.

```
32      private final TcxUploadFileService tcxService;
33      private final GpxUploadFileService gpxService;
34      private final ActivityOperations activityOperations;
35      private final ActivityMongoRepository mongoRepository;
36
37      @Override
38      public List<String> uploadFile(final MultipartFile multiPart, final String type)
39              throws FileOperationNotExecutedException{
40          return Match(type).option(
41                  Case($(is(SOURCE_TCX_XML)), tcxType -> uploadAndSave(multiPart,
                                                                tcxService)),
42                  Case($(is(SOURCE_GPX_XML)), gpxType -> uploadAndSave(multiPart,
                                                                gpxService)))
43                  .toJavaOptional()
44                  .filter(Optional::isPresent)
45                  .map(Optional::get)
46                  .map(this::getIdsByActivities)
47                  .orElseThrow(() -> new FileOperationNotExecutedException("uploadFile",
                                                                        type));
48      }
49
50      @Override
51      public String getFile(final String id, final String type) throws
                                                                FileNotFoundException{
52          return activityOperations.getOriginalFile(id, type)
53                  .orElseThrow(() -> new FileNotFoundException(id, type));
54      }
55
56      private Optional<List<Activity>> uploadAndSave(final MultipartFile multiPartFile,
57                                                     final UploadFileService fileService) {
58          return activityOperations.upload(multiPartFile, fileService)
59                  .map(activities -> mongoRepository.saveAll(activities))
60                  .map(activities -> activityOperations.pushToS3(activities,
                                                                multiPartFile));
61      }
62
63      private List<String> getIdsByActivities(final List<Activity> activities) {
64          return activities
65                  .stream()
66                  .map(Activity::getId)
67                  .collect(toList());
68      }
```

Snippet 48

Por lo tanto, la clase del controlador quedaría muy sencilla como se muestra en el Snippet 49. Tan solo tiene una dependencia con la clase Fachada que es inyectada al iniciarse la aplicación gracias al uso del inyector de decencias de Spring.

```
24  @Validated
25  @RestController
26  @RequiredArgsConstructor
27  @RequestMapping("/file")
28  @CrossOrigin(origins = "*", maxAge = 3600)
29  public class FileRestController {
30
31       private final FileFacade fileFacade;
32
33       @PostMapping("/upload")
34       @ResponseStatus(HttpStatus.CREATED)
35       public List<String> uploadFile(
36           @RequestParam("file") final MultipartFile multiPart,
37           @RequestParam("type")
38           @Pattern(regexp = "gpx|tcx",
39                 message = "Message type should be gpx or tcx.") final String type)
40           throws FileOperationNotExecutedException {
41                 return fileFacade.uploadFile(multiPart, type);
42       }
43
44       @GetMapping("/get/{type}/{id}")
45       public ResponseEntity<String> getFile(
46           @PathVariable  @Pattern(regexp = "^[a-f\\d]{24}$") final String id,
```

```
47              @PathVariable @Pattern(regexp = "gpx|tcx",
48                    message = "Message type should be gpx or tcx.") final String type)
49              throws FileNotFoundException {
50                    return ResponseEntity.ok()
51                              .contentType(MediaType.APPLICATION_OCTET_STREAM)
52                              .body(fileFacade.getFile(id, type));
53        }
54  }
```

Snippet 49

En el paquete **`com.routeanalyzer.api.controller`** se define un manejador de excepciones que convierte una determinada excepción en una respuesta http concreta. Para ello, como se muestra en el Snippet 50, se hace uso de la anotación **`@RestControllerAdvice`** en la definición de la clase, se crea un método por cada manejo de excepción a una respuesta concreta. Para ellos, se define cada método con las anotaciones **`@ResponseHandler`**, **`@ResponseStatus`** y **`@ResponseBody`** para definir la excepción a manejar, el código http de la respuesta y que el método devuelve el cuerpo de la respuesta, respectivamente.

```
33  @Slf4j
34  @RestControllerAdvice
35  public class ExceptionHandlingController {
36
37      @ResponseBody
38      @ExceptionHandler(SAXParseException.class)
39      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
40      Response handleSAXParseException(final Exception exception) {
41          log.warn("SAXParse error happened: ", exception);
42          return createErrorBody(true, SAX_PARSE_EXCEPTION_MESSAGE, exception);
43      }
44
45      @ResponseBody
46      @ExceptionHandler(JAXBException.class)
47      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
48      Response handleJAXBException(final Exception exception) {
49          log.warn("JAXB error happened: ", exception);
50          return createErrorBody(true, JAXB_EXCEPTION_MESSAGE, exception);
51      }
52
53      @ResponseBody
54      @ExceptionHandler(AmazonClientException.class)
55      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
56      Response handleAmazonClientException(final Exception exception) {
57          log.warn("Amazon Client error happened: ", exception);
58          return createErrorBody(true, AMAZON_CLIENT_EXCEPTION_MESSAGE, exception);
59      }
60
61      @ResponseBody
62      @ExceptionHandler(IOException.class)
63      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
64      Response handleIOException(final Exception exception) {
65          log.warn("Input/output error happened: ", exception);
66          return createErrorBody(true, IO_EXCEPTION_MESSAGE, exception);
67      }
68
69      @ResponseBody
70      @ExceptionHandler(IllegalArgumentException.class)
71      @ResponseStatus(code = HttpStatus.BAD_REQUEST)
72      Response handleIllegalArgumentException(final Exception exception) {
73          log.warn("Params error happened: ", exception);
74          return createErrorBody(true, BAD_REQUEST_MESSAGE, exception);
75      }
76
77      @ResponseBody
78      @ExceptionHandler(ActivityNotFoundException.class)
79      @ResponseStatus(code = HttpStatus.NOT_FOUND)
80      Response handleActivityNotFoundException(final Exception exception) {
81          log.warn("Params error happened: ", exception);
82          return createErrorBody(true, ACTIVITY_NOT_FOUND, exception);
83      }
84
85      @ResponseBody
86      @ExceptionHandler(ActivityColorsNotAssignedException.class)
```

```
87      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
88      Response handleColorsNotAssignedException(final Exception exception) {
89          log.warn("Params error happened: ", exception);
90          return createErrorBody(true, COLORS_ASSIGNED_EXCEPTION, exception);
91      }
92
93      @ResponseBody
94      @ExceptionHandler(ActivityOperationNotExecutedException.class)
95      @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
96      Response handleOperationNotExecutedException(final Exception exception) {
97          log.warn("Params error happened: ", exception);
98          return createErrorBody(true, OPERATION_NOT_EXECUTED, exception);
99      }
100
101     @ResponseBody
102     @ExceptionHandler(ConstraintViolationException.class)
103     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
104     Response handleConstraintViolationException(final Exception exception) {
105         log.warn("Params error happened: ", exception);
106         return createErrorBody(true, BAD_REQUEST_MESSAGE, exception);
107     }
108
109     @ResponseBody
110     @ExceptionHandler(MethodArgumentTypeMismatchException.class)
111     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
112     Response handleErrorTypeParamsException(final Exception exception) {
113         log.warn("Params error happened: ", exception);
114         return createErrorBody(true, BAD_REQUEST_MESSAGE, exception);
115     }
116
117     @ResponseBody
118     @ExceptionHandler(FileNotFoundException.class)
119     @ResponseStatus(code = HttpStatus.NOT_FOUND)
120     Response handleFileNotFoundException(final Exception exception) {
121         log.warn("Params error happened: ", exception);
122         return createErrorBody(true, FILE_NOT_FOUND, exception);
123     }
124
125     @ResponseBody
126     @ExceptionHandler(FileOperationNotExecutedException.class)
127     @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
128     Response handleFileOperationNotExecutedException(final Exception exception) {
129         log.warn("Params error happened: ", exception);
130         return createErrorBody(true, OPERATION_NOT_EXECUTED, exception);
131     }
132
133     private Response createErrorBody(
134             final boolean isError, final String description,
135             final Exception exception) {
136         return Response.builder()
137                 .error(isError)
138                 .description(description)
139                 .errorMessage(exception.getMessage())
140                 .exception(JsonUtils.toJson(exception)
141                         .getOrNull())
142                 .build();
143     }
144 }
```

Snippet 50

Modelo Ficheros XML

Como se comentó en el punto *Mapeador* del apartado *Diseño*, es necesario obtener los elementos que modelan cada uno de los tipos de ficheros *xml* documentados en el punto *Ficheros de actividades deportivas*. Los esquemas de cada modelo pueden encontrarse en el *Anexo I*.

Para ello, se utiliza una herramienta perteneciente a la librería *JAXB*[99] de Java que proporciona una forma fácil de obtener y guardar la información procedente de un

[99] Librería que permite asignar clases a representaciones XML.
http://www.oracle.com/technetwork/articles/javase/index-140168.html

documento *xml* en clases Java, es decir, es una librería que realiza un mapeo del fichero original y lo convierte a líneas de código con sus pertinentes anotaciones.

Gracias al comando *xjc*[100] de esta librería es posible obtener las clases relativas a un esquema *xsd*[101] ejecutando esta orden en la terminal como se detallará más adelante.

Tras haber definido esto, lo primero que se necesita es descargar los ficheros que describen los esquemas de los *xml* donde van a ir contenida la información.

Para los ficheros xml con un esquema GPX se crean el modelo asociado en el paquete **`com.routeanalyzer.api.xml.gpx11`** que es donde se declararán estas clases que representan los esquemas anteriormente descritos en el aparatado *Aspectos Teóricos*.

Para los ficheros de tipo *gpx* se necesita ejecutar los comandos de la Ilustración 20. Siempre suponiendo que nuestros ficheros que contienen la información relativa a cada esquema se llaman *gpx.xsd* y *trackpointextension.xsd*, respectivamente.

```
xjc –d src –p com.routeanalyzer.api.xml.gpx11 gpx.xsd
xjc –d src –p com.routeanalyzer.api.xml.gpx11.trackpointextension.garmin
trackpointextension.xsd
```

Ilustración 20

El paquete **`com.routeanalyzer.api.xml.tcx`** del proyecto incluye todas las clases equivalentes a los esquemas descritos en este párrafo.

Para los ficheros de tipo *tcx* se necesita ejecutar los comandos que aparecen en la Ilustración 21. Suponiendo que se han descargados los esquemas con los nombres *tcx.xsd* y *activityExtension.xsd*, respectivamente.

```
xjc –d src –p com.routeanalyzer.api.xml.tcx tcx.xsd
xjc –d src –p com.routeanalyzer.api.xml.tcx.activityextension activityExtension.xsd
```

Ilustración 21

Lectura y Escritura XML

Otra de las cosas que se comentaron en el punto *Lectura y Escritura* del apartado *Diseño* es la necesidad de poder leer o escribir ficheros xml. Por lo tanto, existirá una clase para cada tipo de esquema que extenderá de la clase *AbstractXmlService* y que tienen que definir las clases que hay que utilizar para la lectura o escritura de dichos documentos. Para trabajar con el esquema *tcx* se ha creado la clase *TCXService<TrainingCenterDatabaseT>* que proporciona el valor *TrainingCenterDatabaseT* para la clase T definida en la interfaz. En el Snippet 51 se puede ver como está implementada la clase.

```
10  @Service
11  public class TCXService extends AbstractXMLService<TrainingCenterDatabaseT> {
12
13      public TCXService() {
14              super(TrainingCenterDatabaseT.class);
15      }
16
17      @Override
18      protected JAXBContext getJAXBContext() throws JAXBException {
19              return JAXBContext.newInstance(
20                      TrainingCenterDatabaseT.class, ActivityLapExtensionT.class,
21                              ActivityTrackpointExtensionT.class);
22      }
```

[100] Comando que permite el análisis de un esquema XSD y su posterior transformación en clases JAVA. https://docs.oracle.com/javase/8/docs/technotes/tools/unix/xjc.html
[101] Documento que describe la estructura de un fichero XML. https://es.wikipedia.org/wiki/XML_Schema

```
23
24  }
```

Snippet 51

Para trabajar con el esquema *gpx* se ha creado la clase *GPXService<GpxType>* que proporciona el valor *GpxType* para la clase T definida en la interfaz. En la Snippet 52 se puede ver como está implementada la clase.

```
09  @Service
10  public class GPXService extends AbstractXMLService<GpxType> {
11
12      public GPXService() {
13              super(GpxType.class);
14      }
15
16      @Override
17      protected JAXBContext getJAXBContext() throws JAXBException {
18              return JAXBContext.newInstance(GpxType.class, TrackPointExtensionT.class);
19      }
20
21  }
```

Snippet 52

Mapeo xml a modelo

Como se comentó en el punto *Mapeador* del apartado *Diseño*, es necesario obtener la información de las actividades deportivas a través de los elementos que modelan cada uno de los tipos de ficheros *xml* documentados en el punto *Ficheros de actividades deportivas*. A continuación, se va a describir como se ha implementado cada caso en el sistema que ha sido desarrollado.

Fichero xml a modelo del sistema

Para poder realizar el mapeo del modelo xml al modelo del sistema se ha desarrollado una clase abstracta, llamada *UploadFileService*, que implementa la lógica común que hay que llevar a cabo, independientemente del esquema que sea, para poder convertir un fichero de entrada a una lista de instancias de la clase *Activity*. Esta clase, además, usará la clase *AbstractXmlService*, comentada más arriba, para realizar la lectura del fichero que contiene la actividad física.

```
17  @Slf4j
18  @RequiredArgsConstructor
19  public abstract class UploadFileService<T> {
20
21      protected final AbstractXMLService<T> xmlService;
22      protected final ActivityOperations activityOperationsService;
23      protected final LapsOperations lapsOperationsService;
24      protected final TrackPointOperations trackPointOperations;
25
26      /**
27       * Upload a specific xml file with the date of a sport activity
28       * @param multiPartFile object with the data
29       * @return A list with the data of every activity in the xml file.
30       * @throws RuntimeException encapsulates: IOException, JAXBException
31       */
32      public Try<T> upload(final MultipartFile multiPartFile) {
33          return Try.of(() -> multiPartFile.getInputStream())
34                  .onFailure(err -> log.error("Error trying to get the input stream", err))
35                  .flatMap(xmlService::readXML);
36      }
37
38      public abstract List<Activity> toListActivities(final T optXmlType);
39
40      protected List<Object> toJAXBElementExtensionsValue(final List<Object> extensions) {
41          return extensions.stream()
42                  .filter(JAXBElement.class::isInstance)
43                  .map(JAXBElement.class::cast)
44                  .map(JAXBElement::getValue)
45                  .collect(Collectors.toList());
```

```
46        }
47
48    }
```

Snippet 53

Básicamente, y como se puede ver en el Snippet 53, el método abstracto *toListActivities* realizará un mapeo entre un objeto de la clase T, representará el fichero especifico *xml*, y la lista de objetos de la clase *Activity*.

```
67        /**
68         * Get the list of activities
69         * @param tcxType: class which represents the info inside of a xml document (tcx)
70         * @return list of activities which contains the xml document.
71         */
72        @Override
73        public List<Activity> toListActivities(final TrainingCenterDatabaseT tcxType) {
74            return ofNullable(tcxType)
75                    .map(TrainingCenterDatabaseT::getActivities)
76                    .map(this::toActivities)
77                    .orElseGet(() -> ofNullable(tcxType)
78                            .map(TrainingCenterDatabaseT::getCourses)
79                            .map(this::toActivities)
80                            .orElseGet(Collections::emptyList));
81
82        }
```

Snippet 54

Por ejemplo, en el Snippet 54 se muestra la implementación de este método para el tipo de esquema *tcx*, contenido en la clase *TcxUploadFileService*. En el Snippet 55 se muestra lo mismo, pero para el esquema *gpx* en la clase *GpxUploadFileService*.

```
51        @Override
52        public List<Activity> toListActivities(final GpxType gpxType) {
53            return ofNullable(gpxType)
54                    .map(GpxType::getTrk)
55                    .map(trackLapsList -> trackLapsList.stream()
56                      .map(trackLap -> toActivity(gpxType.getMetadata(), gpxType.getCreator(),
trackLap))
57                            .map(activity -> {
58                                // calculating distance speed values
59                                activityOperationsService.calculateDistanceSpeedValues(activity);
60                                return activity;
61                            })
62                            .collect(toList()))
63                    .orElseGet(Collections::emptyList);
64        }
```

Snippet 55

Modelo a fichero xml

Para poder realizar el mapeo del modelo del sistema a cada uno de los modelos que acepta el sistema se ha desarrollado una clase genérica abstracta *ExportFileService* es definida en el Snippet 56, en ella se define el método *export* que hace uso del método abstracto que será definido en las demás clases hijas.

```
9     public abstract class ExportFileService<T> {
10
11        private AbstractXMLService<T> xmlService;
12
13        public ExportFileService(final AbstractXMLService<T> xmlService) {
14            this.xmlService = xmlService;
15        }
16
17        /**
18         * Export method to a specific file type
19         * @param act: activity to export
20         * @return String object with the data in the specific xml type.
21         */
22        public String export(final Activity act) {
23            return convertToXmlObjects(act)
24                    .map(xmlFile -> xmlService.createXML(xmlFile).get())
25                    .orElseThrow(() ->
```

```
                    new IllegalArgumentException("Not possible to convert activity to xml."));
26      }
27
28      protected abstract Optional<JAXBElement<T>> convertToXmlObjects(
                                                          final Activity activity);
29  }
```

Snippet 56

En el Snippet 56 también se puede ver como se propaga una excepción que será controlada más arriba, en el controlador de excepciones que sirve como mapeador entre excepcione y respuestas *http*.

Al igual que antes, cada hija redefine el método, en el Snippet 57 se puede ver el método para esquemas de tipo *tcx* que producirán elementos de tipo *TrainingCenterDatabaseT* y en el Snippet 58 se puede ver el mismo método pero creado para producir ficheros de tipo *gpx*, es decir, de la clase *GpxType*.

```
49  @Override
50  protected Optional<JAXBElement<TrainingCenterDatabaseT>> convertToXmlObjects(
                                                          final Activity activity) {
51          return ofNullable(activity)
52                  .map(Activity::getLaps)
53                  .map(this::toTcxLaps)
54                  .map(addActivityLaps)
55                  .map(getSetterActivityField::apply)
56                  .map(setActivityField -> setActivityField.apply(activity))
57                  .flatMap(createTrainingCenterDatabase)
58                  .map(objectFactorySupplier.get()::createTrainingCenterDatabase);
59  }
```

Snippet 57

```
42  @Override
43  protected Optional<JAXBElement<GpxType>> convertToXmlObjects(final Activity activity) {
44          return ofNullable(activity)
45                  .map(Activity::getLaps)
46                  .flatMap(this::toOptionalTrkType)
47                  .map(addTrkType)
48                  .map(addGlobalData)
49                  .map(adderGlobalData -> adderGlobalData.apply(activity))
50                  .map(objectFactorySupplier.get()::createGpx);
51  }
```

Snippet 58

Ambas clases usan diferentes métodos privados para poder lograr esta funcionalidad, siguiendo el principio de modularidad y de responsabilidad única.

Servicios Web de terceros

Obtener altitud en un punto

Para posibilitar la obtención de un determinada altitud en función de las coordenadas geoespaciales, se definen las propiedades, como se muestra en el Snippet 59, en el fichero *application.properties*. Estas propiedades serán inyectadas en la clase *GoogleMapsAPIProperties* gracias a la "magia" que realiza el *framework* Spring. Como se mencionó anteriormente, la clave del servicio irá encriptada y su valor se obtendrá fácilmente a través del método estático *decrypt* de la clase de utilidad *Encrypter*. Como se dijo anteriormente, al estar el código subido a un repositorio público de GitHub lo mejor es que los valores sensibles estén codificados.

```
15  # google maps elevation service
16  google-maps-api.elevation-protocol=https
17  google-maps-api.elevation-host=maps.googleapis.com
18  google-maps-api.elevation-endpoint=/maps/api/elevation/json
19  google-maps-api.encrypted-api-key=
                    uQp1FZQnVlXmwrp7me9xmcOfrp+CvXIAPmm0c2hyGdzjNnFtxpMiGvA+fDYxJAUk
```

Snippet 59

Para hacer uso del servicio público de Google, hay que indicar los siguientes valores:

- El método *http* get, no cambia.
- El host definido en el Snippet 59 como **`google-maps-api.elevation-host`**.
- La ruta definida con su valor en el Snippet 59 como **`google-maps-api.elevation-endpoint`**.
- Lo parámetros definidos como clave-valor.
    - Las localizaciones identificadas por su latitud y longitud separadas por un `|` siguiendo esta estructura:
      `latitud1,longitud1|latitud2,longitud2|…|latitudN,longitudN`
    - La clave descifrada para poder hacer uso del servicio y que proviene del valor definido en el Snippet 59 como **`google-maps-api.encrypted-api-key`**.

El servicio devuelve una respuesta representada con el modelo que se define en el paquete **`com.routeanalyzer.api.services.googlemaps.model`**.

Frontend

En el Snippet 35 se puede ver, de forma generalizada, la clase principal de la aplicación que incluirá al resto de componentes como se mostro en el punto *Frontend* del apartado *Diseño*.

Visualización de la ruta

Toda la lógica para la visualización, edición y exportación de una actividad física esta contenida en el componente de la aplicación llamado *RouteContainer*, para ello, es necesario que se haya seleccionado alguna ruta existente en el sistema. Este componente, como se mostró en la Ilustración 19, está formado por otros cuatro grupos de componentes, definidos en su método *render* que se puede visualizar en el Snippet 60.

```
115  render(){
116      if(this.props.activity && this.props.activity.laps){
117          return (
118              <div>
119                  <Grid container spacing={8}>
120                      <Grid item xs={12}>
121                          <DownloadFileComponent/>
122                      </Grid>
123                  </Grid>
124                  <Grid container spacing={16}>
125                      <Grid item xs={7}>
126                          <div className="RouteTable">
127                              <RouteTable handleRemoveLaps={this.removeLaps}
128                                          handleJoinLaps={this.joinLaps}
129                              />
130                          </div>
131                      </Grid>
132                      <Grid item xs={5}>
133                          <div className="RouteMap">
134                              <RouteMapContainer handleRemoveMarker={this.removePosition}
135                                                 handleSplitLap={this.splitLap}
136                                                 handleMouseOver={this.selectTrackpoint}
137                              />
138                          </div>
139                      </Grid>
140                  </Grid>
141                  <Grid container spacing={24}>
142                      <Grid item xs={12}>
143                           <ElevationsChart yTitle={'Altitude (m)'}
144                                            xTitle={'Time (hh:mm:ss)'}
145                           />
146                      </Grid>
147                  </Grid>
148                  <Grid container spacing={24}>
149                      <Grid item xs={12}>
150                          <HeartRateChart yTitle={'Heart Rate (bpm)'}
151                                          xTitle={'Time (hh:mm:ss)'}
152                          />
153                      </Grid>
154                  </Grid>
155                  <Grid container spacing={24}>
156                      <Grid item xs={12}>
157                          <SpeedChart yTitle={'Speed (m/s)'}
158                                      xTitle={'Time (hh:mm:ss)'}
159                          />
160                      </Grid>
161                  </Grid>
162              </div>
163          );
164      } else
165          return(
166              <div>
167
168              </div>
169          );
170  }
```

Snippet 60

Los dos componentes *DownloadFileComponent* y *RouteTable* son más pequeños que los otros dos y lo que hacen es permitir exportar una actividad (supliendo la funcionalidad de los requisitos **RQ-FN-15**: El sistema permitirá exportar la aplicación modificada a un fichero xml. y **RQ-FN-16**: El sistema permitirá exportar el fichero original de la ruta que se ha procesado. Se obtendrá el fichero de entrada aun habiéndose modificado la actividad deportiva.) y mostrar los datos agregados para cada segmento en una tabla (cumpliendo con el requisito **RQ-FN-14**: El sistema mostrará una tabla que resuma cómo ha ido la actividad física y permita realizar operaciones sobre los segmentos.), respectivamente. A continuación, se van a detallar los demás componentes.

Mapa

Para cumplir con la funcionalidad del requisito definido anteriormente en el punto *Especificación de Requisitos* del apartado *Planificación*: **RQ-FN-07**: El sistema visualizará la ruta deportiva en un mapa que permita ver cual fue el rumbo recorrido por el usuario de la aplicación., **RQ-FN-08**: El sistema permitirá eliminar un determinado punto seleccionándolo desde el mapa en el que se ha mostrado la actividad deportiva., **RQ-FN-09**: El sistema tendrá la funcionalidad de dividir la ruta deportiva desde el mapa seleccionando el punto especifico. y **RQ-FN-13**: El sistema permitirá visualizar en cada momento el punto en los tres gráficos anteriores y en el mapa, según se mueve el puntero del ratón a lo largo de los datos de un gráfico o de la trayectoria visualizada en el mapa.; la aplicación tiene que hacer uso de algún sistema de terceros que le proporcione un mapa donde poder visualizar la ruta deportiva. Para ello, existe una librería desarrollada por Google que proporciona todas estas herramientas de una manera sencilla para *JavaScript*, el mismo lenguaje que sobre el que está fundamentado el *framework* React.

Si volvemos al anterior ejemplo, el Snippet 60, se puede ver donde está definido este elemento haciendo uso del componente interno identificado como *RouteMapContainer*. Este componente tendrá todos los atributos que definirán el contenedor donde se alojará el mapa que presente la ruta deportiva. Por ejemplo, si se pone atención al Snippet 61, se muestra el método *render* del contenedor donde se definen las dimensiones del componente que representa al mapa y también el tipo de elemento visual donde irá incluido, en este caso se identifica como *Paper* y es un componente de una librería de Google llamada Material[102] que persigue encontrar una analogía entre el diseño de la interfaz visual y los materiales físicos reales. También es en esta clase donde se define el método, llamado *calculateCenterZoom* que centrará el mapa en función de los puntos que se muestren en el recorrido de la actividad deportiva; donde se definen las funciones para borrar un punto especifico (*handleDeletePoint*), visualizar (*handleMarkClick*) y cerrar (handleInfoClose) la información del entreno en un determinado punto. El resto de las propiedades que se le pasan al componente hijo *RouteMapComponent* pertenecen a los valores de las propiedades del componente en sí.

```
89  render() {
90    if(this.props.laps.length){
91     return (
92         <RouteMapComponent
93           loadingElement= {<div style={{ height: '100%' }} />}
94           containerElement= {<Paper style={ { height: '424px', width: '100%' ,
95                                                'marginTop': '24px'}} /> }
96           mapElement= { <div style={ { height: '100%' }} /> }
97           fitBound={this.calculateCenterZoom}
```

[102] https://es.wikipedia.org/wiki/Material_Design

```
          handleMarkClick={this.handleMarkClick}
          handleInfoClose={this.handleInfoClose}
          handleRemoveMarker={this.handleDeletePoint}
          handleSplitLap={this.props.handleSplitLap}
          currentTrack={this.props.idLap + "_" + this.props.idTrackpoint}
          track={this.props.currentTrack}
          keys={this.state.keys}
          laps={this.props.laps}
        />
    );
   } else {
    return( <p> No data map </p> );
   }
  }
```

Snippet 61

Usando una librería, llamada *react-google-maps*, desarrollada por un usuario de Github[103] para este *framework* se puede hacer uso de todo el sistema de Google Maps para representar la actividad física en el mapa. Es decir, esta librería define una capa más de abstracción que se adapta al marco de trabajo de React. Por lo tanto, como se puede ver en el Snippet 62, esta clase se define, por mayor comodidad, usando una librería, llamada *recompose*, creada por otro usuario de Github[104] y que permite crear componentes de *React* mediante la composición de valores, entre otras cosas.

```
  const RouteMapComponent =
   compose(
     withProps({
       googleMapURL: "https://maps.googleapis.com/maps/api/js" +
           "?key=AIzaSyDEtc96UC9co31AFUNuNsPZ1xV4SYEMwfA" +
           "&v=3.exp&libraries=geometry,drawing,places"
     }),
     lifecycle({
       componentDidUpdate(prevProps, prevState) {
         if(prevProps.laps !== this.props.laps){
           this.props.fitBound(this.props.map);
         }
       }
     }),
     withScriptjs,
     withGoogleMap
   )(props =>

     <GoogleMap
       ref={props.fitBound}
       defaultZoom={12}
       mapTypeId= {'terrain'}>
       {
         props.laps.map((lap) => {
           const strokeColor = lap.color;
           const path = lap.tracks.map((track)=>track.position);
           return <Polyline
               key={lap.index}
               path={path}
               options={ { strokeColor, strokeOpacity: 3.0, strokeWeight: 8 } }
               onMouseOver={(event)=>handleMouseOver(event, props)}
               onMouseOut={(event)=>handleMouseOver(event, props)}
           /> })
       }
       {
         props.laps.map( (lap) => {
           return lap.tracks.map((track)=>{
             return <MarkerInfoViewComponent
               key={lap.index + "_" + track.index}
               trackPoint={track}
               indexLap={props.laps.indexOf(lap)}
               indexTrackpoint={lap.tracks.indexOf(track)}
               sizeLaps = {props.laps.length}
               sizeTrackpoints = {lap.tracks.length}
```

[103] https://tomchentw.github.io/react-google-maps/
[104] https://github.com/acdlite/recompose

```
                handleMarkClick={props.handleMarkClick}
                handleInfoClose={props.handleInfoClose}
                handleSplitLap={props.handleSplitLap}
                handleRemoveMarker={props.handleRemoveMarker}
                currentTrack={props.currentTrack}
                keys={props.keys}
              />
            })
          })
        }
      </GoogleMap>
    );
```

Snippet 62

Por lo tanto, se utiliza los siguientes métodos en la composición del componente llamado *RouteMapComponent* de la aplicación:

- El método *withProps* en el que se definen las propiedades, en este caso la *url* de la API de Google para obtener todas las herramientas para la creación y gestión de mapas.
- El método *lifecycle* en el que se definen el comportamiento en cada fase del ciclo de vida. En este caso se define el comportamiento cada vez que la información se actualiza, con la llamada al método *calculateCenterZoom*, definido e implementado en el componente contenedor, se actualiza el punto de vista del mapa en función de todos los puntos existentes de la actividad.
- Los métodos withScriptjs y withGoogleMap de la librería que proporciona toda la configuración necesaria para hacer funcionar a todas estas herramientas. Entre otras cosas, con el primero se carga toda la API de *Google Maps* para *JavaScript* y con el segundo se inicializa el *RouteMapComponent* con instancias del DOM.
- La definición del mapa se realiza a través del componente **`GoogleMap`** de la libería *react-google-maps*. Dentro de este elemento se define otro componente de esta librería llamado **`Polyline`** que representa la ruta de la actividad física formada por todos los puntos de la ruta identificados por la latitud y la longitud. Por otro lado, también se ha definido un componente propio llamado *MarkerInfoViewComponent* que representa cada punto de la ruta identificado por un marcador y un *tooltip* asociado que se activa cada vez que se hace clic sobre alguno de estos elementos y que muestra la información para el punto que se ha seleccionado.

```
render(){
    const {indexLap,indexTrackpoint,sizeTrackpoints} = this.props;
    // key marker
    const keyMarker = this.props.indexLap +"_"+this.props.indexTrackpoint;
    const isStartPoint = this.isStart(indexLap, indexTrackpoint);
    const isEndPoint = this.isEnd(indexLap, indexTrackpoint);
    let label = null;
    let icon = null;
    // Start
    if(isStartPoint)
      label = "A";
    // End
    else if(isEndPoint)
      label = "B";

    // Only split lap functionality if it's between start and end (not included) of a lap
    let handleSplitLap = this.props.handleSplitLap;
    if(isStartPoint || isEndPoint || (indexTrackpoint === (sizeTrackpoints-1) ||
                                                    indexTrackpoint === 0) )
       handleSplitLap = null;
    if(this.isRenderedPoint())
      return (
          <Marker
              position={this.props.trackPoint.position}
              onClick={()=>this.props.handleMarkClick(keyMarker)}
```

```
157                 icon={icon}
158                 label={label}
159             >
160             { this.props.keys.length > 0 && this.props.keys(keyMarker) &&
161                 <InfoWindow onCloseClick={()=>this.props.handleInfoClose(keyMarker)}>
162                 <InfoViewContent  trackPoint={this.props.trackPoint}
163                                   handleRemoveMarker={this.props.handleRemoveMarker}
164                                   handleSplitLap={handleSplitLap}
165                                   keyMarker={keyMarker}/>
166                 </InfoWindow>
167             }
168             </Marker>
169       )
170     else
171       return null;
172   }
```

Snippet 63

En el Snippet 63 se visualiza el código que representa el método *render* del componente que indica donde se encuentra cada punto ubicado en el mapa. Se puede visualizar hasta tres puntos simultáneamente con el marcador comentado anteriormente: por defecto, se muestra el punto de inicio que se etiqueta con la letra *A*, el punto que indica el final de la carrera con la letra *B* y, de manera opcional, si se ha seleccionado algún punto mediante el movimiento del ratón sobre la trayectoria o si se ha movido en alguna de las gráficas, se mostrará el punto correspondiente a su latitud y longitud. Para obtener este punto, si se ha movido el ratón sobre los anteriores elementos, se usa un método, definido en el fichero **`/src/Utils/operations.js`** y mostrado en el Snippet 64, con el nombre *getNearestPosition* que indica cual es el punto de la ruta más cercano al punto sobre el que esta el ratón posicionado sobre el componente **`Polyline`**, identificado mediante su latitud y longitud. Esta funcionalidad permite saber en todo momento cuales son los datos relativos al punto sobre el que se encuentra el puntero del ratón. Por otro lado, la información del punto seleccionado en las otras tres gráficas procederá del estado global de la aplicación almacenado en una propiedad llamada *currentTrackpoint*, gracias al uso de Redux.

Otra de las cosas que se ven en el Snippet 63, es que, al hacer clic sobre el componente, se desplegará una ventana con la información para ese punto.

```
160 export function getNearestPosition(laps,position){
161   let indexLap =null, indexTrackpoint = null, minDistance = Number.POSITIVE_INFINITY;
162   laps.forEach((lap, eachIndexLap)=>{
163     lap.tracks.forEach((track,eachIndexTrackpoint)=>{
164       let eachPosition = {
165         lat: track.position.lat,
166         lng: track.position.lng,
167       };
168       let currentMin = getDistanceBetweenPoints(eachPosition,position);
169       if(currentMin<minDistance){
170         indexTrackpoint = eachIndexTrackpoint;
171         indexLap = eachIndexLap;
172         minDistance = currentMin;
173       }
174     })
175   })
176   return {
177     indexLap,
178     indexTrackpoint
179   }
180 }
```

Snippet 64

Para que se visualice la información relativa a cada punto, además de ciertos botones que permiten eliminar y dividir la ruta, se ha añadido otro componente,

como se ve en el método *render* del Snippet 63, llamado *InfoViewContent* como se muestra en el Snippet 65.

```
175  class InfoViewContent extends Component{
176    render(){
177      return(
178        <div>
179          <h2>Track Point</h2>
180          <InfoViewLineComment title={"Time"}
181                   value={new Date(this.props.trackPoint.date).toLocaleTimeString()} />
182          <InfoViewLineComment title={"Latitude"}
183                               value={this.props.trackPoint.position.lat} />
184          <InfoViewLineComment title={"Longitude"}
185                               value={this.props.trackPoint.position.lng} />
186          <InfoViewLineComment title={"Altitude"}
187                               value={this.props.trackPoint.alt} />
188          <InfoViewLineComment title={"Distance"}
189                               value={this.props.trackPoint.dist} />
190          <InfoViewLineComment title={"Speed"}
191                               value={this.props.trackPoint.speed} />
192          <InfoViewLineComment title={"Heart Rate"}
193                               value={this.props.trackPoint.bpm} />
194          <input type="button"
195                 onClick={()=>
196                 this.props.handleRemoveMarker(this.props.trackPoint.position,
197                 this.props.trackPoint.date, this.props.trackPoint.index,
198                  this.props.keyMarker)}
199                 value="Remove"/>
200         { this.props.handleSplitLap &&
201          <input type="button"
202                 onClick={()=>
203                     this.props.handleSplitLap(this.props.trackPoint.position,
204                     this.props.trackPoint.date, this.props.trackPoint.index)}
205                 value="Split Lap"/>
206          }
207        </div>)
208    }
209  }
```

Snippet 65

En este componente también se incluyen los botones comentados anteriormente para posibilitar la de edición de la ruta desde el mapa, permitir el borrado de un punto y la división de un tramo en dos segmentos. Por otro lado, como se puede ver, también se usa un componente propio, llamado *InfoViewContent*, muy simple que realiza la tarea de visualización de las propiedades pasadas como parámetros siguiendo la estructura: **`{title}: {value}`**.

Gráficas

Para poder cubrir los requisitos **RQ-FN-10**: El sistema mostrará la altitud a lo largo de todo el recorrido en una gráfica de áreas., **RQ-FN-11**: El sistema mostrará la frecuencia cardiaca a lo largo de todo el recorrido en una gráfica de barras. y **RQ-FN-12**: El sistema mostrará la velocidad a lo largo de todo el recorrido en una gráfica de líneas. es necesaria la creación de tres gráficas que representen los diferentes tipos de datos a cubrir. A la hora de diseñar e implementar esta funcionalidad se ha decidido usar la librería D3.js[105] ya que tiene mucha popularidad en el mundo del desarrollo web. Es una librería *JavaScript* que permite la manipulación de documentos basados en datos, es decir, hace posible enlazar los datos en el DOM y posteriormente aplicarles algún tipo de transformación que los convierta en otro tipo de representación siendo fundamentada por tecnologías sustentadas como *svg*[106], *html* y *css*.

[105] https://d3js.org
[106] https://es.wikipedia.org/wiki/Gráficos_vectoriales_escalables

Para la realización de cada gráfico, se ha adaptado esta librería a la forma de trabajar de *React* mediante el uso de componentes independientes y reutilizables. Para ello, se han creado los componentes, enumerados a continuación, que permiten definir diferentes tipos de gráficos con la idea maximizar la reutilización de código.

```
3  export default class Axis extends React.Component {
4      componentDidMount() {
5        this.renderAxis();
6      }
7
8      componentDidUpdate() {
9        this.renderAxis();
10     }
11
12     renderAxis() {
13       let node = this.refs.axis;
14       let axis = null;
15       switch(this.props.orient){
16         case 'left': axis = d3.axisLeft(); break;
17         case 'bottom': axis = d3.axisBottom(); break;
18         case 'right': axis = d3.axisRight(); break;
19         case 'top': axis = d3.axisTop(); break;
20         default: axis = d3.axisBottom(); break;
21       }
22       if(this.props.isTimeFormat)
23         axis = axis.tickFormat(d3.timeFormat(this.props.timeFormat));
24       axis = axis.ticks(this.props.ticks).scale(this.props.scale);
25       d3.select(node).call(axis);
26     }
27
28     render() {
29       return (
30         <g className={'axix ' + this.props.className} ref="axis"
31            transform={this.props.translate}>
32           <text transform={this.props.textTranslate} y={this.props.y} dy={this.props.dY}
33                 x={this.props.x}
34                 dx={this.props.dX} style={{textAnchor:'end'}} fill={"#000"} >
35             {this.props.text}
36           </text>
37         </g>
38         )
39     }
40 }
```

Snippet 66

Por un lado, se han creado el componente que representa a cualquier eje de la gráfica, llamado Axis y representado en el Snippet 66.

```
3   export default class XYAxis extends React.Component{
4      render(){
5        const xSettings = {
6          translate: `translate(0, ${this.props.height - this.props.padding})`,
7          scale: this.props.xScale,
8          orient: 'bottom',
9          isTimeFormat: true,
10         timeFormat:"%H:%M:%S",
11         text: this.props.textXAxis,
12         dX:'.71em',
13         y:(this.props.padding-10),
14         x:(this.props.width - this.props.padding - this.props.textXAxis.length)
15       };
16       const ySettings = {
17         translate: `translate(${this.props.padding}, 0)`,
18         scale: this.props.yScale,
19         orient: 'left',
20         isTimeFormat: false,
21         text: this.props.textYAxis,
22         textTranslate: 'rotate(-90)',
23         dY:'.71em',
24         x:-1,
25         y:(-1*(this.props.padding-5))
26       };
27       return <g className="x-y-axis">
28         <Axis className={'x'} {...xSettings} ticks = {this.props.ticks}/>
29         <Axis className={'y'} {...ySettings} ticks = {this.props.ticks}/>
```

```
30        </g>
31      }
32  }
```

Snippet 67

El componente *Axis* es utilizado dos veces para crear el eje de abscisas y el eje de ordenadas como se puede ver en el componente *XYAxis*, definido en el Snippet 67, donde también se definen las propiedades de cada eje como la orientación, el tipo de datos que representa, el margen, el texto, entre otros.

Lo siguiente a definir es un componente que represente cualquier gráfico independiente de su tipo y que posibilite la reutilización de otros componentes. Para ello se define el componente llamado *AbstractChart* con el método *render* definido en el Snippet 68.

```
34        render() {
35            let scales = {};
36            if(this.props.data && this.props.data.length)
37                scales = {xScale: this.xScale(this.props), yScale: this.yScale(this.props)};
38            let childrenWithProps = [];
39            if(this.props.children && Array.isArray(this.props.children)
40                && this.props.children.filter(child=>child).length>0){
41
42                childrenWithProps = React.Children.map(
43                    this.props.children
44                        .filter(child => child), (child,index) =>
45                        React.cloneElement(child, {
46                            data:this.props.data,
47                            laps:this.props.laps,
48                            dataLine: this.props.dataLine,
49                            height:this.props.height,
50                            track:this.props.track,
51                            handleMouseOver: index===0 ? this.props.handleMouseOver : null,
52                            xScale: scales.xScale,
53                            yScale: scales.yScale
54                        })
55                );
56            } else if(this.props.children && !Array.isArray(this.props.children)) {
57                childrenWithProps = React.cloneElement(this.props.children, {
58                    data:this.props.data,
59                    laps:this.props.laps,
60                    dataLine: this.props.dataLine,
61                    height:this.props.height,
62                    track:this.props.track,
63                    xScale: scales.xScale,
64                    yScale: scales.yScale
65                });
66            }
67
68            return(
69                <Paper  style={{overflowX:'auto'}}>
70                    <svg  height={this.props.height} width={this.props.width} >
71                        {childrenWithProps}
72                        <XYAxis {...this.props} {...scales}/>
73                    </svg>
74                </Paper>
75            );
76        }
```

Snippet 68

Esta clase lo que realiza en el *render* es el visionado del elemento *svg* teniendo los ejes de abscisas y ordenadas, definidos gracias a la clase *XYAxis*, además de visualizar todos los componentes hijos de esta etiqueta. Para ellos realiza la copia usando el método *React.cloneElement* con las propiedades propias de esta clase. Por tanto, hay que definir los posibles datos que pueden existir en cada gráfico: *DataArea* para los diagramas de área; *DataBar* para los diagramas de barras; *DataLine* para los diagramas con una sola línea y *DataMultiLine* para diagramas con múltiples líneas.

```
12    renderBarLap(lap, xScale, yScale){
13        return lap.tracks.map((track,index) =>
14            <rect className={'lap ' + lap.index + ", trackpoint " + index }
```

```
15                x={xScale(track[0])} y={yScale(track[1])} fill={lap.color}
16                height={this.props.height - this.props.padding - yScale(track[1])}
17                width={0.5} key={lap.index + "_" + index}/>)
18    };
19
20    render(){
21        let date = null, data=null;
22        if(this.props.track){
23            date = this.props.track.date;
24            data = this.props.track.bpm;
25        }
26        return(
27            <g className={'bars'}>
28            { this.props.laps.map(lap =>
29                  this.renderBarLap(lap, this.props.xScale, this.props.yScale))}
30            <InformationTooltip ref="tooltip" xScale={this.props.xScale}
31                yScale={this.props.yScale} width={this.props.width}
32                height={this.props.height} padding={this.props.padding}
33                handleMouseOver={this.props.handleMouseOver} laps={this.props.laps}
34                data={this.props.data} legend={this.props.legend}
35                trackpoint={{ date, data }}
36            />
37            <rect className="overlay" width={this.props.width-(2*this.props.padding)}
38                height={this.props.height-(2*this.props.padding)}
39                style={{fill: 'none', pointerEvents: 'all'}}
40                onMouseOver={(event)=>
41                          this.refs.tooltip.mouseMove(event)}
42                onMouseMove={(event)=>
43                          this.refs.tooltip.mouseMove(event)}/>
44            </g>
45        );
46    }
```

Snippet 69

Por ejemplo, en el Snippet 69 se puede visualizar como es la codificación del componente *DataBar* que representa la información de un gráfico de barras. Para ello, se define un método *renderBarLap* que crea un objeto *rect* de la librería *svg* que representa cada valor del conjunto de datos. Este conjunto de elementos *rect* irá incluido dentro de la etiqueta *g* de la librería *svg* que representa un gráfico de barras. Por otro lado, también se incluye dentro de esta etiqueta un componente creado en la aplicación para la visualización de la información de en ese punto en concreto una vez que el puntero del ratón se posiciona encima de dicha posición. Lo demás tipos de datos son parecidos a este, se encuentran implementados en la aplicación por lo que se pueden consultar en el repositorio de GitHub, sin embargo, para simplificar solo se muestra el Snippet 69 como ejemplo.

Por lo tanto, ya solo quedaría definir el componente en función del tipo de dato que contenga reutilizando los componentes definidos más arriba:

- AreaChart que usa el componente *AbstractChart* con el tipo de dato DataArea como se muestra en el Snippet 70.

```
5   export default class AreaChart extends React.Component{
6
7       render(){
8                 return(
9                           <AbstractChart {...this.props}  >
10                                { this.props.data && this.props.data.length &&
11                                          <DataArea   {...this.props}/>
12                                }
13                          </AbstractChart>
14                );
15      }
16  }
```

Snippet 70

- *BarChart* para mostrar un gráfico de barras se usa el componente *AbstractChart* con el tipo de dato *DataBar* y, si la aplicación tiene más de un segmento, se

mostrará una línea que conecte la frecuencia cardiaca media de cada tramo gracias al uso del tipo de dato *DataLine* como se muestra en el Snippet 71.

```
7   export default class BarChart extends React.Component{
8
9         render(){
10           let existInformedData = this.props.data &&
11                                   this.props.data.filter(data=>data[1]).length>0;
12                return(
13                                   <AbstractChart {...this.props} >
14                                             { existInformedData &&
15                                                  <DataBar   {...this.props}/>
16                                             }
17                                             { existInformedData &&
18                                                  <DataLine   {...this.props}
19                                                      color={getColorObject('Red')[500]}
20                                                      strokeWidth={1}/>
21                                             }
22                                   </AbstractChart>
23                );
24        }
25  }
```

Snippet 71

- *LineChart* para visualizar un gráfico de líneas como se muestra en el Snippet 72. Al igual que en el anterior caso, se usa *AbstractChart* para la creación de la gráfica, *DataMultiLine* para definir la línea de velocidad en cada tramo y, si existe mas de uno, con *DataLine* la velocidad media en cada segmento.

```
6   export default class LineChart extends React.Component{
7
8         render(){
9                 let color = getColorObject('Red')[200];
10                let existInformedData = this.props.data &&
11                                  this.props.data.filter(data=>data[1]).length>0;
12                return(
13                                  <AbstractChart {...this.props} >
14                                      { existInformedData &&
15                                            <DataMultiLine   {...this.props}/>
16                                            }
17                                      { existInformedData &&
18                                            <DataLine   {...this.props} color={color}/>
19                                      }
20                                  </AbstractChart>
21                );
22        }
23  }
```

Snippet 72

Todos estos elementos que se han descrito, como pertenecen a la adaptación de la librería D3.js en el *framework* React, se han definido en el directorio **`/src/Charts`**.

Por tanto, si volvemos a poner atención al Snippet 60, podemos ver que la aplicación usa tres componentes para representar las gráficas con los datos de la elevación, la frecuencia cardiaca y la velocidad. Estos componentes, a su vez, se hace uso de los componentes definidos en los Snippet 70, Snippet 71 y Snippet 72. En cada uno de ellos, como se mostrará a continuación, se usará un componente externo desarrollado por un usuario de Github[107] que permite crear componentes reusables que puedan adaptarse al entorno en donde son visualizados, es decir, se adaptan por ejemplo al cambio de tamaño de ventana del navegador donde se está ejecutando la aplicación web. Entre otras cosas, como se verá más adelante, a cada uno de ellos se le pasa como parámetros:

[107] https://github.com/ctrlplusb/react-sizeme

- Los datos que se va a visualizar gracias al envío de un *array* que contiene para cada momento la altitud a la que el usuario estaba identificado con la propiedad *data*.
- Los colores con los que se identifica cada segmento identificado con la propiedad *laps*.
- En el caso de las gráficas de la frecuencia cardiaca y de velocidad, se enviará un *array* que identifique a cada segmento con el valor medio que se está representando. Este campo se identifica con la propiedad *dataLine*.
- El texto del eje de abscisas identificado con la propiedad *testXAxis*.
- El testo del eje de ordenadas identificado con la propiedad *textYAxis*.
- El margen identificado con la propiedad *padding*.
- La anchura y la altura del componente con la que se tiene que visualizar identificado con las propiedades *width* y *height*, respectivamente.
- El punto actual que proviene del estado global que ha de ser visualizado a través de el *tooltip* identificado con la propiedad *track*.
- El método que hay que ejecutar cada vez que el puntero del ratón se encuentre sobre alguna zona de la gráfica identificado con la propiedad *handleMouseOver*.
- La unidad del valor en cada punto de la gráfica identificado con la propiedad *legend*.
- El número de marcadores en el eje horizontal identificado con la propiedad *ticks*.

```
71      render(){
72        return(
73            <MySizeGrid
74                ref={ref=>this.chartGrid = ref} item
75                xs={12} style={{maxHeight:'200px', minWidth:'500px'}}>
76                <AreaChart  data={this.state.elevations}
77                            laps={this.state.laps}
78                            textYAxis={this.props.yTitle}
79                            textXAxis={this.props.xTitle}
80                            padding = {40}
81                            width = {this.state.width}
82                            height = {this.state.height}
83                            track={this.props.currentTrackpoint}
84                            handleMouseOver={this.props.updateTrackpoint}
85                            legend={'m'}
86                            ticks={5}/>
87            </MySizeGrid>
88        );
89      }
```

Snippet 73

Por lo tanto, el componente *ElevationChart*, en el Snippet 73 se visualiza su método *render*, es compuesto por una gráfica de área que representa las elevaciones del terreno por las que el usuario ha pasado durante la actividad física haciendo uso del componente *AreaChart* definido en el Snippet 70. Se puede ver como la propiedad *legend* es definida con la etiqueta m de metros, ya que cada dato representa la altitud en esta unidad.

Por otro lado, el componente *HeartRateChart*, en el Snippet 74 se visualiza su método *render*, representa la frecuencia cardiaca que ha tenido el usuario a lo largo de la actividad física haciendo uso del componente *BarChart* definido en el Snippet 71. Se puede apreciar como la propiedad legend es definida como *bpm* en inglés *"beats per minute"* (en español ppm: pulsaciones por minuto), ya que cada dato será representado en esta unidad de medida.

```
75      render(){
76          return(
77              <MySizeGrid
```

```
78                ref={ref=>this.chartGrid = ref} item
79                xs={12} style={{maxHeight:'200px', minWidth:'500px'}}>
80                <BarChart   data={this.state.bpms}
81                            laps={this.state.laps}
82                            dataLine={this.state.avg}
83                            textYAxis={this.props.yTitle}
84                            textXAxis={this.props.xTitle}
85                            padding = {40}
86                            width = {this.state.width}
87                            height = {this.state.height}
88                            track={this.props.currentTrackpoint}
89                            handleMouseOver={this.props.updateTrackpoint}
90                            legend={'bpm'}
91                            ticks={5}/>
92            </MySizeGrid>
93        );
94    }
```

Snippet 74

Por último, el componente *SpeedChart*, el método *render* es definido en el Snippet 75, representa la velocidad que el usuario ha tenido a lo largo de la actividad física haciendo uso del componente *LineChart*, definido anteriormente en el Snippet 72. Se puede ver como la etiqueta, en este caso que estamos tratando con valores de velocidad, es definida como *m/s*, es decir, metros por segundo como es de esperar.

```
75    render(){
76      return(
77          <MySizeGrid ref={ref=>this.chartGrid = ref} item
78                      xs={12} style={{maxHeight:'200px', minWidth:'500px'}}>
79            <LineChart  data={this.state.speeds}
80                        laps={this.state.laps}
81                        dataLine={this.state.avg}
82                        textYAxis={this.props.yTitle}
83                        textXAxis={this.props.xTitle}
84                        padding = {40}
85                        width = {this.state.width}
86                        height = {this.state.height}
87                        track={this.props.currentTrackpoint}
88                        handleMouseOver={this.props.updateTrackpoint}
89                        legend={'m/s'}
90                        ticks={5}/>
91          </MySizeGrid>
92      );
93    }
```

Snippet 75

Pruebas

Las pruebas de software son investigaciones tanto empíricas como técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o *stakeholder*, tal y como define este término la Wikipedia. Por lo tanto, ayuda a evaluar la funcionalidad e identificar posibles fallos del producto desarrollado, siendo el objetivo principal un producto sin defectos y con la mayor calidad posible. Existen muchas formas de categorizar las pruebas de software, a continuación, se enumeran algunas de ellas.

Hay una forma de separar pruebas de software en función de si la ejecución de la prueba es llevada a cabo por una persona:

- Pruebas automatizadas que son aquellas codificadas por una persona y ejecutadas por la máquina. Sin embargo, estas pruebas no suelen representar la visión de usuarios reales.
- Pruebas manuales en las que interfieren directamente un usuario que probará la aplicación directamente. La desventaja de este enfoque es que es necesario un gran esfuerzo e invertir mucho tiempo en probar las diferentes opciones.

Existen dos formas de categorizar las pruebas en función de si su evaluación necesita de la ejecución del sistema:

- Pruebas estáticas que son aquellas que se realizan sin ejecutar el código, por ejemplo, el seguimiento del flujo de ejecución de la aplicación o la revisión de la documentación.
- Pruebas dinámicas que para su evaluación es necesario que el sistema esté en funcionamiento, es decir, se haya iniciado previamente.

Para el caso de pruebas dinámicas, el último tipo mostrado en el párrafo anterior, existen dos tipos de pruebas en función del enfoque:

- Pruebas de caja negra con el punto de mira en las entradas y la salida de la aplicación que se está probando.
- Pruebas de caja blanca que son las que se enfocan en el código de la aplicación y de su estructura, es decir, la forma de comunicarse entre todos los componentes, y que se espera que tengan un comportamiento específico. En este tipo de pruebas es necesario que la persona que las vaya a desarrollar tenga pleno conocimiento del sistema.

Como el sistema está formado por dos partes, se van a comentar las pruebas realizadas en cada aplicación de manera separada. Hay que dejar bien aclarado que el desarrollo de estas pruebas ha estado centrado más en pruebas automatizadas, dinámicas y de caja blanca para los dos subsistemas del proyecto desarrollado. Sin embargo, como es de esperar, también he realizado pruebas de caja blanca como usuario del sistema.

Cabe destacar que, como se explicó en el apartado *Planificación*, el proceso de desarrollo de pruebas forma parte de cada Sprint, es decir, el producto entregado tras finalizar cada iteración ha de contener todas las pruebas oportunas que validen el correcto funcionamiento de la aplicación. A diferencia de lo que ocurre con los proyectos que siguen una planificación en cascada, donde las pruebas no son realizadas hasta que se han acabado todas las demás etapas anteriores, pudiéndose descubrir errores una vez que el proyecto ya ha acabado la fase de desarrollo y generando retrasos a la hora de finalizar el sistema. Existen algunas metodologías modernas de desarrollo que centran su modo de trabajo en la realización de pruebas antes de que el sistema esté finalizado, es lo que se llama desarrollo guiado por pruebas (o en inglés Test Driven Development).

Backend

Esta parte de la aplicación va a ser probada con una librería llamada *JUnit* [108] que introduce un marco de trabajo para realizar pruebas unitarias. Con este tipo de pruebas se consigue probar un trozo de código, en orientación a objetos como es el caso, de forma independiente y asilada. Para ello, se crea una clase en el mismo paquete que la clase a probar, pero en el directorio **`/src/test`**.

```
18      @Test
19      public void convertToListBoolean() {
20          // Given
21          List<String> inputList = Arrays.asList("true","false", "true","true");
22          // When
23          List<Boolean> result = CommonUtils.toListOfType(inputList, Boolean::valueOf);
24          // Then
25          assertThat(result).isEqualTo(Arrays.asList(true, false,
26                  true, true));
27      }
28
29      @Test
30      public void convertToListLong() {
31          // Given
32          List<String> inputList = Arrays.asList("8321312431412412422",
33                  "1043123412312312313", "4531231231233232",
34                  "32312312314231394");
35          // When
36          List<Long> result = CommonUtils.toListOfType(inputList, Long::valueOf);
37          // Then
38          assertThat(result)
39                  .isEqualTo(Arrays.asList(8321312431412412422L,
40                          1043123412312312313L, 4531231231233232L,
41                          32312312314231394L));
42      }
43
44      @Test
45      public void convertEmptyList() {
46          // Given
47          List<String> inputList = Collections.emptyList();
48          // When
49          List<Long> result = CommonUtils.toListOfType(inputList, Long::valueOf);
50          // Then
51          assertThat(result).isEqualTo(Collections.emptyList());
52      }
53
54      @Test
55      public void convertToNullList() {
56          // Given
57          // When
58          List<Long> result = CommonUtils.toListOfType(null, Long::valueOf);
59          // Then
60          assertThat(result).isNull();
```

[108] https://junit.org/junit4/

```
61        }
```

Snippet 76

En el Snippet 76, se muestran los métodos de test para el método de utilidad *toListOfType* definido en la clase *CommonUtils* definida en el paquete **`com.routeanalyzer.api.common`**. Por lo tanto, comprueban el correcto funcionamiento, validando que las salidas son las esperadas. Para ello, se anota con **`@Test`** cada método que representa la evaluación de un comportamiento esperado. En el primer método, se comprueba que la conversión de lista de cadenas de caracteres a valores booleanos se realiza correctamente, en el segundo método se realiza lo mismo, pero en este caso la lista de destino será de tipo *Long*, el tercero comprueba que de una lista vacía se devuelve otra igual y el cuarto comprueba que para una lista que no tenga valor definido se devolverá el valor nulo. La lógica de este método, mostrado en el Snippet 77, es bastante sencilla y fácil de entender por lo que no hay ningún tipo de dificultad a la hora de realizar todas las pruebas asociadas.

```
24  public static <T> List<T> toListOfType(
25                  final List<String> listStrings, final Function<String, T> convertTo) {
26      return ofNullable(listStrings)
27        .filter(__ -> nonNull(convertTo))
28        .map(__ -> listStrings
29                  .stream()
30                  .flatMap(stringObject -> Try.of(() -> convertTo.apply(stringObject))
31                  .onFailure(err -> log.error("Error trying to convert " +
32                            "to a List of types using {}", convertTo, err))
33                  .toJavaStream())
34                  .collect(toList()))
35        .orElse(null);
36  }
```

Snippet 77

Por otro lado, esta librería también permite ejecutar código antes (**`@Before`**) o después (**`@After`**) de cada método de la clase de pruebas. En el Snippet 78 se muestra un ejemplo de un trozo de código que se ejecuta antes de cada método de la clase *LapsOperationsImpTest*.

```
76        @Before
77        public void setUp() {
78            addLeftTracks();
79            addRightTracks();
80        }
81
82        private void addLeftTracks() {
83            createLeftTracks();
84            trackPointsLeft = Lists.newArrayList();
85            trackPointsLeft.add(trackPointLeft1);
86            trackPointsLeft.add(trackPointLeft2);
87            trackPointsLeft.add(trackPointLeft3);
88            trackPointsLeft.add(trackPointLeft4);
89        }
90
91        private void addRightTracks() {
92            createRightTracks();
93            trackPointsRight = Lists.newArrayList();
94            trackPointsRight.add(trackPointRight1);
95            trackPointsRight.add(trackPointRight2);
96            trackPointsRight.add(trackPointRight3);
97            trackPointsRight.add(trackPointRight4);
98        }
```

Snippet 78

En este caso es de gran utilidad para que el resultado de la ejecución de algunos de los casos de prueba no interfiera en otros, ya que habrá alguno de ellos que modifiquen el valor de los objetos definidos en este método de inicialización.

```
1047      @Test
1048      public void resetTotalValuesTest() {
1049          // Given
```

```
1050            lapRight = Lap.builder()
1051                    .totalTimeSeconds(40.55)
1052                    .distanceMeters(760.00)
1053                    .build();
1054
1055            // When
1056            lapsOperations.resetTotals(lapRight);
1057
1058            // Then
1059            assertThat(lapRight).isNotNull();
1060            assertThat(lapRight.getTotalTimeSeconds()).isNull();
1061            assertThat(lapRight.getDistanceMeters()).isNull();
1062        }
```

Snippet 79

Por ejemplo, en el Snippet 79 se puede ver como después de la ejecución de este método, el objeto *lapRight* se habría modificado, por lo que el valor definido inicialmente, si no se volviese a ejecutar el método definido en el Snippet 78, habría cambiado, poniendo a nulo los valores de los agregados del segmento. Lo que podría causar problemas en futuras ejecuciones de otros casos de pruebas.

Por otro lado, hay veces que el código se quiere que se ejecute una sola vez, al principio de la inicialización de la clase. Esto puede ser debido a que los objetos creados nunca son modificados, es decir, se aplican funciones puras sobre ellos o porque la modificación de estas instancias no implica condiciones de carrera en otros casos de prueba.

```
38      @BeforeClass
39      public static void setUpClass() throws Exception {
40          zonedDateTime = ZonedDateTime.of(2020, 05, 12,
41                  17, 36, 00, 00, ZoneId.of("Europe/Madrid"));
42          utcZonedDateTime = ZonedDateTime.of(2020, 05, 12,
43                  15, 36, 00, 00, ZoneId.of("UTC"));
44          timeMillis = 1589297760000L;
45
46          date = new Date(timeMillis);
47          gregorianCalendar = new GregorianCalendar();
48          gregorianCalendar.setTime(date);
49
50          xmlGregorianCalendar = DatatypeFactory.newInstance()
51                  .newXMLGregorianCalendar(gregorianCalendar);
52
53          utcDate = new Date(1589290560000L);
54          utcGregorianCalendar = new GregorianCalendar();
55          utcGregorianCalendar.setTime(utcDate);
56
57          utcXmlGregorianCalendar = DatatypeFactory.newInstance()
58                  .newXMLGregorianCalendar(utcGregorianCalendar);
59      }
```

Snippet 80

En el Snippet 80, se puede ver como se inicializan una serie de variables estáticas para su posterior uso. En este caso, este método es necesario que sea **`static`** para que se pueda ejecutar justo cuando se crea la clase. En el Snippet 81 se muestra un ejemplo de cómo se utilizan las instancias creadas en el método *setUpclass*.

```
61      @Test
62      public void toDate() {
63          // Given
64
65          // When
66          Optional<Date> result = DateUtils.toDate(zonedDateTime);
67
68          // Then
69          assertThat(result).contains(new Date(timeMillis));
70      }
71
72      @Test
73      public void nullToDate() {
74          // Given
75
76
```

```
77            // When
78            Optional<Date> result = DateUtils.toDate(zonedDateTimeNull);
79
80            // Then
81            assertThat(result).isEmpty();
82        }
```

Snippet 81

Estas variables son usadas a lo largo de toda la aplicación, todos los métodos que tienen la clase de utilidad *DateUtils* son funciones puras ya que nunca modifican el estado interno de los parámetros, es por esta razón por lo que no se necesita ejecutar antes o después de cada método. En el Snippet 82 se muestra el código de la función que se esta probando en el Snippet 81.

```
23        public static Optional<Date> toDate(final ZonedDateTime offsetDateTime) {
24            return ofNullable(offsetDateTime)
25                    .map(ZonedDateTime::toInstant)
26                    .map(Date::from);
27        }
```

Snippet 82

La dificultad puede ir en aumento cuando el código a testear tiene dependencias con otras clases del sistema que realizan, por ejemplo, el acceso a base de datos o llamadas a servicios de terceros. Por lo tanto, es aquí donde hay que realizar *mocks* de todos los elementos que son dependencias de la clase y que permiten que se sea independiente haciendo que los cambios en otras clases le afecten lo más mínimo, teniendo siempre en mente que el contrato no puede haber cambiado. *Mockito*[109] es una librería que permite realizar este tipo de cosas de forma sencilla y fácil para el programador. Volviendo al ejemplo del Snippet 78 que prueba la clase *LapsOperationsImpl*, podemos ver en uno de varios de sus casos de prueba que se define el comportamiento de los objetos que son dependencias de la clase, en concreto, de la instancia de la clase *GoogleMapsApiService* y de la otra instancia de la clase *TrackPointOperations*. Para ello, poder definir el comportamiento de estos objetos, es necesario hacer uso de la anotación **`@RunWith`** de la librería de JUnit que permite invocar todos los casos de prueba a través de la clase referenciada en la anotación. Por lo tanto, en el Snippet 83 se ejecutará la clase de *test* LapsOperationsImpTest a través de la clase *MockitoJUnitRunner* que representa el *runner* de *Mockito*. Además, se le indican las dependencias de las que fijaremos su comportamiento mediante la anotación **`@Mock`** que permite instanciar un objeto de la clase *Mock* a partir de la clase de la dependencia. Por último, todos estos elementos definidos con esta anotación son inyectados en la clase de pruebas a través del uso de la anotación **`@InjectMocks`**.

```
42  @RunWith(MockitoJUnitRunner.class)
43  public class LapsOperationsImpTest {
44
45      @Mock
46      private TrackPointOperations trackPointOperations;
47
48      @Mock
49      private GoogleMapsApiService googleMapsService;
50
51      @InjectMocks
52      private LapsOperationsImpl lapsOperations;
53
54      ...
55
```

[109] https://site.mockito.org

```
56  }
```

Snippet 83

En el Snippet 84 se puede ver como se utilizan la dependencia *mockeada* de la clase *GoogleMapsApiService*. En concreto, este caso de prueba que la respuesta del método *calculateAltitude* de la clase bajo prueba es el esperado. Para elos, es necesario definir el comportamiento que va a adquirir la llamada del método *getCoordinatesCode* de la dependencia. Para ello, se usan los métodos estáticos *doReturn* y *when* de la clase Mockito para definir cada comportamiento en función del punto de la ruta que se pase como parámetro utilizando el método estático *eq* de la clase *ArgumentMatchers* de la librería de Mockito. Una vez que se comprueban los resultados gracias al uso del método estático *assertThat* de la librería *AssertJ* creada por un usuario de GitHub[110], se verifica que efectivamente se han llamado a las dependencias *mockeadas* de las que el método hace uso. Para ello, se hace uso del método estático *verify* que tiene como parámetros, además de la instancia del *mock*, el número de llamadas que deberían haberse realizado a un determinado método. Por defecto, si no se le indica otra cosa, es solo una única vez. Así pues, se comprueba que el método que obtiene la altitud es llamado cuatro veces ya que es el número de puntos que tiene el segmento.

```
365  @Test
366  public void calculateAltitudeTest() {
367    // Given
368    Consumer<TrackPoint> resetAltValues = trackPoint -> trackPoint.setAltitudeMeters(null);
369    trackPointsLeft.stream().forEach(resetAltValues);
370    lapLeft = Lap.builder().tracks(trackPointsLeft)
371              .distanceMeters(100.0)
372              .startTime(toUtcZonedDateTime(timeMillisLeft1).orElse(null))
373              .index(8)
374              .maximumHeartRate(176)
375              .maximumSpeed(22.0)
376              .averageHearRate(123.03)
377              .averageSpeed(14.00)
378              .totalTimeSeconds(50.0)
379              .intensity(null)
380              .build();
381    BigDecimal ele1 = toBigDecimal(245.0);
382    BigDecimal ele2 = toBigDecimal(285.0);
383    BigDecimal ele3 = toBigDecimal(300.0);
384    BigDecimal ele4 = toBigDecimal(225.0);
385    String positions = trackPointLeft1.getPosition().getLatitudeDegrees() + ","
386              + trackPointLeft1.getPosition().getLongitudeDegrees() + "|"
387              + trackPointLeft2.getPosition().getLatitudeDegrees() + ","
388              + trackPointLeft2.getPosition().getLongitudeDegrees() + "|"
389              + trackPointLeft3.getPosition().getLatitudeDegrees() + ","
390              + trackPointLeft3.getPosition().getLongitudeDegrees() + "|"
391              + trackPointLeft4.getPosition().getLatitudeDegrees() + ","
392              + trackPointLeft4.getPosition().getLongitudeDegrees();
393    doReturn(positions).when(googleMapsService).createPositionsRequest(anyList());
394    Map<String, String> result = Maps.newHashMap();
395    result.put("status", "OK");
396    String key1 = trackPointLeft1.getPosition().getLatitudeDegrees() + ","
397              + trackPointLeft1.getPosition().getLongitudeDegrees();
398    result.put(key1, ele1.toString());
399    String key2 = trackPointLeft2.getPosition().getLatitudeDegrees() + ","
400              + trackPointLeft2.getPosition().getLongitudeDegrees();
401    result.put(key2, ele2.toString());
402    String key3 = trackPointLeft3.getPosition().getLatitudeDegrees() + ","
403              + trackPointLeft3.getPosition().getLongitudeDegrees();
404    result.put(key3, ele3.toString());
405    String key4 = trackPointLeft4.getPosition().getLatitudeDegrees() + ","
406              + trackPointLeft4.getPosition().getLongitudeDegrees();
407    result.put(key4, ele4.toString());
408    doReturn(result).when(googleMapsService).getAltitude(eq(positions));
409    doReturn(key1).when(googleMapsService).getCoordinatesCode(eq(trackPointLeft1));
```

[110] https://joel-costigliola.github.io/assertj/

```
410     doReturn(key2).when(googleMapsService).getCoordinatesCode(eq(trackPointLeft2));
411     doReturn(key3).when(googleMapsService).getCoordinatesCode(eq(trackPointLeft3));
412     doReturn(key4).when(googleMapsService).getCoordinatesCode(eq(trackPointLeft4));
413
414     // When
415     lapsOperations.calculateAltitude(lapLeft);
416
417     // Then
418     assertThat(lapLeft).isNotNull();
419     assertThat(lapLeft.getTracks().get(0).getAltitudeMeters()).isEqualTo(ele1);
420     assertThat(lapLeft.getTracks().get(1).getAltitudeMeters()).isEqualTo(ele2);
421     assertThat(lapLeft.getTracks().get(2).getAltitudeMeters()).isEqualTo(ele3);
422     assertThat(lapLeft.getTracks().get(3).getAltitudeMeters()).isEqualTo(ele4);
423     verify(googleMapsService).createPositionsRequest(anyList());
424     verify(googleMapsService).getAltitude(positions);
425     verify(googleMapsService, times(4)).getCoordinatesCode(any());
426 }
```

Snippet 84

Como aclaración, en todos los tests suelo hacer tres comentarios definidos en este orden: *given*, *when* y *then*. Esto me ayuda a saber que en el primer punto tengo que instanciar todos los objetos que se utilizarán en la prueba y definir el comportamiento de los objetos *mock*, como se muestra en el ejemplo de más arriba, para posteriormente utilizarlos en la llamada al método que se está probando después del segundo punto y, finalmente, en el último punto realizar las comprobaciones de las salidas esperadas y verificar que se han hecho las llamadas oportunas a los métodos de las dependencias que usa la funcionalidad bajo prueba.

Otra de las dependencias que usa la clase *LapsOperationsImpTest* es la se encarga de realizar todas las operaciones relativas a un punto de la trayectoria de la ruta definida toda la funcionalidad en la interfaz *TrackPointOperations*. Se utiliza por ejemplo para calcular la distancia que hay entre dos puntos de la ruta y este método es usado en la clase bajo pruebas para calcular la distancia total de un segmento en concreto.

```
546     @Test
547     public void calculateDistanceLapInTheMiddle() {
548         // Given
549         Consumer<TrackPoint> resetDistValues = trackPoint ->
550                 trackPoint.setDistanceMeters(null);
551         trackPointsRight.stream().forEach(resetDistValues);
552         lapRight = Lap.builder().tracks(trackPointsRight)
553                 .startTime(toUtcZonedDateTime(timeMillisRight1).orElse(null))
554                 .index(3)
555                 .totalTimeSeconds(50.0)
556                 .intensity("HIGH")
557                 .build();
558         double dist1 = 10.0;
559         double dist2 = 8.0;
560         double dist3 = 12.0;
561         double dist4 = 5.0;
562         double previousDistance = trackPointLeft4.getDistanceMeters().doubleValue();
563         doReturn(dist1).when(trackPointOperations)
564                 .calculateDistance(eq(trackPointLeft4), eq(trackPointRight1));
565         doReturn(dist2).when(trackPointOperations)
566                 .calculateDistance(eq(trackPointRight1), eq(trackPointRight2));
567         doReturn(dist3).when(trackPointOperations)
568                 .calculateDistance(eq(trackPointRight2), eq(trackPointRight3));
569         doReturn(dist4).when(trackPointOperations)
570                 .calculateDistance(eq(trackPointRight3), eq(trackPointRight4));
571
572         // When
573         lapsOperations.calculateDistanceLap(lapRight, trackPointLeft4);
574
575         // Then
576         assertThat(lapRight).isNotNull();
577         assertThat(lapRight.getTracks().get(0).getDistanceMeters()).isEqualTo(
578                 toBigDecimal(previousDistance + dist1));
579         assertThat(lapRight.getTracks().get(1).getDistanceMeters()).isEqualTo(
580                 toBigDecimal(previousDistance + dist1 + dist2));
581         assertThat(lapRight.getTracks().get(2).getDistanceMeters()).isEqualTo(
```

```
582                 toBigDecimal(previousDistance + dist1 + dist2 + dist3));
583         assertThat(lapRight.getTracks().get(3).getDistanceMeters()).isEqualTo(
584                 toBigDecimal(previousDistance + dist1 + dist2 + dist3 + dist4));
585         verify(trackPointOperations, times(1))
586                 .calculateDistance(trackPointLeft4, trackPointRight1);
587         verify(trackPointOperations, times(1))
588                 .calculateDistance(trackPointRight1, trackPointRight2);
589         verify(trackPointOperations, times(1))
590                 .calculateDistance(trackPointRight2, trackPointRight3);
591         verify(trackPointOperations, times(1))
592                 .calculateDistance(trackPointRight3, trackPointRight4);
593     }
```

Snippet 85

Por lo tanto, se necesitará definir el comportamiento de este objeto como se muestra en el Snippet 85 para cada dos puntos contiguos de la ruta. En este caso, se puede ver que el método que verifica las llamadas, se le pasa el método estático *times* de *Mockito* como parámetro indicándole que se ejecuta solo una vez. Sin embargo, si se omitiese este valor, la ejecución de esta prueba no cambiaría en absoluto.

Antes de pasar a describir las pruebas de integración, no hay que pasar por alto un elemento de la librería *spring-test* que permite comprobar la funcionalidad de los *endpoints* de un controlador web gracias al uso de la clase *MockMvc* incluida en el paquete **`org.springframework.test.web.servlet`**. Para poder usar esta dependencia, en el proyecto se añade el código mostrado en el Snippet 86 que representa la dependencia de Spring, llamada *spring-boot-starter-test*, que incluye todas las librerías que se necesitan para la realización de pruebas, tanto unitarias como de integración.

```
135     <dependency>
136             <groupId>org.springframework.boot</groupId>
137             <artifactId>spring-boot-starter-test</artifactId>
138             <scope>test</scope>
139     </dependency>
```

Snippet 86

Por lo tanto, la clase *MockMvc* es utilizada para probar el correcto funcionamiento de los *endpoints* que proporcionan las clases *ActivityRestController* y *FileRestController*. En el Snippet 87 podemos ver como se crean la clase que ejerce el papel de controlador para las funcionalidades relacionadas con las actividades deportivas. Además, posibilita la realización de un tipo de pruebas que se sitúa en la frontera entre las pruebas unitarias y las pruebas de integración. Esto es así, porque este tipo de prueba ni es una prueba unitaria puesto que los *endpoints* necesitan estar alojados en un contenedor ni tampoco es un test unitario ya que, si se definen correctamente, solo se probará el componente controlador instanciado dentro de una plataforma simulada. Para ello, es necesario utilizar la anotación **`@RunWith`** con la clase de Spring que ejecutará el test, en este caso identificado como *SpringRunner*. También es necesario indicar cuál es la clase que funciona como controlador, la cual va a ser probada, para ello, se ha de utilizar la anotación **`@WebMvcTest`** pasándole como parámetro dicha clase, en este caso *ActivityRestController*. Con esta última anotación logramos que se configure el objeto de la clase *MockMvc* que será inyectado en la instancia de la clase de prueba cuando sea ejecutada. En el código también se define un objeto de la clase *ActivityFacade* con una anotación llamada **`@MockBean`** que lo que hace es crear un *mock* de esta clase e instanciarlo dentro del contexto de Spring.

```
47  @RunWith(SpringRunner.class)
48  @WebMvcTest(ActivityRestController.class)
49  public class ActivityRestControllerTest {
50      @MockBean
51      private ActivityFacade activityFacade;
52
53      @Autowired
54      private MockMvc mockMvc;
```

```
55
56          @Value("classpath:expected/activity/split-lap-tcx.json")
57          private Resource splitTcxJsonResource;
58
59          ...
60      }
```

Snippet 87

Por otro lado, también se define un objeto de tipo *Resource*, una clase que permite la entrada y salida de ficheros en Spring, que mediante la anotación **`@Value`** permite inyectar el fichero que se encuentra en el valor del parámetro. Para ello, es imprescindible que se defina delante de la ruta del fichero el prefijo "**`classpath:`**", de esta forma se podrá definir la respuesta, en su mayoría en formato *json*, de las dependencias que tiene el controlador.

Por ejemplo, en el Snippet 88 se muestra el caso de prueba que valida que un tramo haya sido separado en dos segmentos separados. Para ello, como se hizo anteriormente, en el primer punto se definen las variables que serán utilizadas y el comportamiento de los *mocks* que tenga la clase que se esta probando, en este caso, el objeto *mockeado* al principio denominado *activityFacade*. Posteriormente se usa la instancia inyectada identificada como *mockMvc* para realizar la llamada al servicio *splitLap* a través del método *perform* que realizará la llamada http a la ruta indicada con el verbo que se le indique, en este caso, una operación *put* ya que se va a modificar el estado interno del objeto. También es necesario concatenar todos los parámetros y sus valores asociados, que el *endpoint* necesite, a la operación *perform*.

Posteriormente se comprueba con el método *andExpect* que el tipo de respuesta tenga un código 200 *http* y que el tipo de formato sea *json*. Por último, con el uso del método estático de la clase *Mockito*, se comprueba que se haya llamado al método *splitLap* de la dependencia *mockeada* con los parámetros definidos en el primer punto del método.

```
372  @Test
373  public void splitLapTest() throws Exception {
374       // Given
375       Activity splitActivity = toActivity(splitTcxJsonResource);
376       String lat = "42.6132170";
377       String lng = "-6.5733730";
378       Long timeMillis = 1519737378000L;
379       Integer index = 2;
380       doReturn(splitActivity).when(activityFacade)
381                 .splitLap(ACTIVITY_TCX_ID, lat, lng, timeMillis, index);
382       // When
383       // Then
384       mockMvc.perform(put(SPLIT_LAP_PATH, ACTIVITY_TCX_ID)
385                 .param("lat", lat)
386                 .param("lng",lng)
387                 .param("timeInMillis", String.valueOf(timeMillis))
388                 .param("index", String.valueOf(index)))
389                 .andExpect(status().isOk())
390                 .andExpect(content().contentType(MediaType.APPLICATION_JSON_VALUE));
391
392       verify(activityFacade).splitLap(eq(ACTIVITY_TCX_ID),
393                 eq(lat), eq(lng), eq(timeMillis), eq(index));
394       }
```

Snippet 88

Como se muetra en el Snippet 88, también es necesario definer el comportamiento de la fachada que usa el *endpoint*, para ello, se usa un método que convierte el objeto Resource, definido en el Snippet 87, que permite convertir el document *json* en una instancia de la clase Actividad como se muestra en el Snippet 89.

```
102  public static Activity toActivity(Resource resource)  {
103    return Try.of(() -> resource.getURL())
104       .onFailure(err -> log.error("It could not be possible to get the url"))
105       .flatMap(urlTcx -> Try.of(() -> Resources.toString(urlTcx, Charsets.UTF_8))
106       .onFailure(err -> log.error("It could not be possible to get to json string"))
```

```
107         .flatMap(jsonStr -> JsonUtils.fromJson(jsonStr, Activity.class)))
108         .getOrNull();
109     }
```

Snippet 89

Hasta ahora se han descrito cómo se han implementado, en líneas generales, los tests unitarios. Una vez que han sido implementados todos ellos, ya existe una base suficientemente grande de pruebas para dar paso al desarrollo de pruebas de integración que, como su nombre bien indica, comprueban que las interacciones entre todos los elementos funcionan como es esperado.

Para realizar estas pruebas, *spring-boot-test* integra una serie de utilidades que permiten desplegar el contenedor de *Servlets*, en este caso *Tomcat*[111]; levantar el contexto de Spring; ejecutar todos los tests integración de la clase en sí; y de la finalización del contexto de Spring y del contenedor de aplicaciones.

Para la realización de las dos pruebas de integración que se ha realizado, se ha creado una clase común, llamada *IntegrationTest*, que incluye toda la funcionalidad necesaria para ejecutar estos tests.

```
22  @TestConfiguration
23  @RunWith(SpringRunner.class)
24  @TestPropertySource("classpath:test.properties")
25  @DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_CLASS)
26  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
27  public abstract class IntegrationTest {
28
29      private static final String LOCALHOST_URL = "http://localhost";
30
31      private static final String
32              DOCKER_COMPOSE_MONGO_DB = "src/test/resources/mongodb/docker-compose.yml";
33      private static final String
34              MONGO_CONTAINER_NAME = "mongodb";
35      private static final int
36              MONGO_PORT = 27017;
37
38      protected TestRestTemplate testRestTemplate;
39
40      public static DockerComposeContainer mongoDbContainer =
41              new DockerComposeContainer(new File(DOCKER_COMPOSE_MONGO_DB))
42              .withExposedService(MONGO_CONTAINER_NAME, MONGO_PORT);
43
44      static {
45          mongoDbContainer.start();
46      }
47
48      @LocalServerPort
49      private int localPort;
50
51      @Autowired
52      private RestTemplateBuilder restTemplateBuilder;
53
54      @PostConstruct
55      public void initialize() {
56          this.testRestTemplate = new TestRestTemplate(restTemplateBuilder
57                  .rootUri(format("%s:%d",LOCALHOST_URL,localPort)));
58      }
59  }
```

Snippet 90

En el Snippet 90, se puede ver como se han creado todos los recursos compartidos por las clases de pruebas de integración identificadas como *ActivityRestControllerIntegrationTest* y *FileRestControllerIntegrationTest*. En esta clase abstracta también se indica la clase runner para pruebas de integración de Spring con la

[111] https://es.wikipedia.org/wiki/Tomcat

que se ejecutarán los tests de las clases hijas y el fichero *test.properties*, como se indica en el Snippet 91, con todos los valores de las propiedades para ejecutar las pruebas. Cabe destacar el uso de la propiedad **spring.main.allow-bean-definition-overriding=true** que permite sobrescribir *beans* de la aplicación por otros de prueba como por ejemplo aquel que sirve para conectarse al repositorio de documentos y del que se hablará más adelante.

```
route-analyzer.encrypted-mongo-uri=LizPpnuuH+o7bni+FMPccnz1n5fnhC0VPZVZkUbecz8=
route-analyzer.mongo-database=test
spring.main.allow-bean-definition-overriding=true
aws.s3-bucket=route-analyzer-bucket-test
logging.level.root=DEBUG
```

Snippet 91

Por un lado, como se muestra en el trozo de código, esta clase abstracta instancia un objeto de la clase *TestRestTemplate* con el tipo de modificador de acceso fijado a **protected** puesto que va a ser usado por las clases de prueba para realizar peticiones *http* reales a los *endpoints* de los controladores que se van a probar en cada caso. Para definir cual es el servidor que se levanta en cada aplicación de forma dinámica, es necesario definir cuale es el puerto en el que se desplegará en cada caso, para eso se hace uso del parámetro *WebEnvironment.RANDOM_PORT* en la anotación **@SpringBootTest** y de la anotación **@LocalServerPort** junto con la variable global *localPort* para definir en cada caso cuál es. Para la creación de este objeto también se hace uso de una clase creada por Sprint, llamada RestTemplateBuilder, que actua como parámetro para la creación de objetos *TestRestTemplate*.

Por otro lado, se utiliza la librería *testcontainers*[112], mostrada en el Snippet 92 cómo se define en el *pom*, que permite desplegar una base de datos real cada vez que se ejecutan las pruebas de integración. Como se muestra en este enlace[113], en el código se ha arrancado el contenedor dentro del elemento **static** para que se arranque una sola vez, como si de un *singleton* se tratase, de esta forma se consigue que sea compartido por todas las clases hijas. Esta librería hace uso de la herramienta Docker[114] que automatiza el despliegue de aplicaciones dentro de contenedores de software.

```
<dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>testcontainers</artifactId>
        <version>${testcontainers.version}</version>
        <scope>test</scope>
</dependency>
```

Snippet 92

Para definir la base de datos *mongodb* es necesario definir un fichero *docker-compose.yml*, definido en el Snippet 93, que permitirá indicarle a esta herramienta qué opciones se desean para el despliegue y la inicialización de la aplicación.

```
mongodb:
  image: 'mongo:4.0'
  environment:
    - MONGO_INITDB_DATABASE=test
  ports:
    - '27017:27017'

mongo_seed:
  build: ./mongo-seed
```

[112] https://github.com/testcontainers/testcontainers-java/
[113] https://www.testcontainers.org/test_framework_integration/manual_lifecycle_control/#singleton-containers
[114] https://www.docker.com

```
10      links:
11        - mongodb
12
```

Snippet 93

Por ejemplo, se define:

- La imagen que quiere usarse, en este caso, la identificada como mongo y la versión 4.
- El nombre de la base de datos a desplegar llamada *test*.
- El mapeo entre el puerto del contenedor (27017) y el puerto que corresponde a la máquina (27017) donde se están ejecutando las pruebas de integración.

  El directorio */mongo-seed* donde se encuentra toda la información para la inicialización de la base de datos indicándole el contenedor *mongodb* donde hay que aplicar esos cambios. Este directorio contiene un fichero, identificado como *Dockerfile*, con las instrucciones que se han de ejecutar, como se muestra en el Snippet 94, para la carga de datos. En este fichero también se especifica un documento *json* llamado *init.json* localizado en este directorio también que contiene todos los datos a importar. Por lo tanto, es necesario que Docker esté instalado y que se haya ejecutado antes de lanzar las pruebas de integración.

```
1    FROM mongo
2    COPY init.json /init.json
3    CMD mongoimport --host mongodb --db test --collection activities --type json
4        --file /init.json --jsonArray
```

Snippet 94

Una vez definida la clase abstracta *IntegrationTest* que permite definir los cimientos de las pruebas de integración, se van a enumerar las dos clases que realizan estos tests.

La clase que se muestra en el Snippet 95 contiene todas las pruebas de integración que comprueban la funcionalidad de los *endpoints* que realizan las operaciones de obtención, modificación y exportación de las actividades deportivas. Se ve que esta clase extiende la funcionalidad de la clase mostrada en el Snippet 90.

```
49   public class ActivityRestControllerIntegrationTest extends IntegrationTest {
50
51       @Autowired
52       private ActivityMongoRepository activityMongoRepository;
53
54       @Value("classpath:expected/json-activity-tcx.json")
55       private Resource tcxJsonResource;
56       @Value("classpath:expected/json-activity-gpx.json")
57       private Resource gpxJsonResource;
58       @Value("classpath:expected/activity/split-lap-tcx.json")
59       private Resource splitTcxJsonResource;
60       @Value("classpath:expected/activity/remove-point-tcx.json")
61       private Resource removePointTcxJsonResource;
62       @Value("classpath:expected/activity/join-laps-tcx.json")
63       private Resource joinLapsTcxJsonResource;
64       @Value("classpath:expected/activity/lap-colors-tcx.json")
65       private Resource lapColorsTcxJsonResource;
66       @Value("classpath:expected/activity/remove-lap-tcx.json")
67       private Resource removeLapTcxJsonResource;
68       @Value("classpath:expected/activity/remove-laps-tcx.json")
69       private Resource removeLapsTcxJsonResource;
70
71       private HttpEntity<Activity> requestEntity =
72               new HttpEntity<>(new Activity(), new HttpHeaders());
73
74       ...
75
76   }
```

Snippet 95

El objeto *requestEntity* es definido de manera global ya que será utilizado en múltiples pruebas de la clase para realizar llamadas *http* del tipo *put* mediante el objeto *testRestTemplate* de la clase padre.

Como se comentó más arriba, en el Snippet 87 de las pruebas unitarias, aquí también se hace uso de la inyección de documentos gracias al uso de la clase *Resource* de Spring y de la anotación **`@Value`**. Estos objetos se utilizan para realizar comprobaciones del estado de los objetos después de realizar las llamadas a los *endpoints*.

Por otro lado, también se inyecta del contenedor el objeto *ActivityMongoRepository* que realiza, como se comentó anteriormente, el acceso a base de datos mediante la extensión de la interfaz *MongoRepository* de Spring. Gracias a este objeto, se puede comprobar que los objetos han cambiado tras haber llamado a algún *endpoint* que modifica el estado de alguna actividad deportiva en concreto.

Por ejemplo, en el Snippet 96 se muestra uno de los métodos de prueba que comprueban que al llamar al endpoint, definido como **`/activity/{id}`**, se devuelve efectivamente la actividad física que se corresponde con el identificador pasado como variable en la definición del *path* del *endpoint*. Esto se comprueba con los objetos obtenidos gracias a la inyección de recursos de Spring y al método *toActivity* definido en el Snippet 89 de la clase *TestUtils*. Además, otra de las cosas que se comprueban es que la respuesta de llamada *http* tenga un valor 200 con el código *ok*.

```
75      @Test
76      public void getGpxActivityByIdTest() throws Exception {
77          // Given
78          Activity gpxActivity = toActivity(gpxJsonResource);
79          String getGpxUri = UriComponentsBuilder.fromPath(GET_ACTIVITY_PATH)
80                  .buildAndExpand(ACTIVITY_GPX_ID)
81                  .toUriString();
82          // When
83          // Activity with id 5ace8cd14c147400048aa6b0 exists in database
84          ResponseEntity<Activity> result =
85                  testRestTemplate.getForEntity(getGpxUri, Activity.class);
86          // Then
87          assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
88          assertThat(result.getBody()).isEqualTo(gpxActivity);
89      }
```

Snippet 96

Otra de las cosas que se comprueban en estos métodos, es que los comportamientos no esperados se producen de la forma correcta. Por ejemplo, en el Snippet 97 se comprueba que, dos actividades que no se encuentra en la base de datos, el servicio devuelve una respuesta http con un valor 404 con el código *not_found*.

```
107     @Test
108     public void getActivityDataBaseEmptyTest() {
109         // Given
110         String getActivityNotFound1 = UriComponentsBuilder.fromPath(GET_ACTIVITY_PATH)
111                 .buildAndExpand(NOT_EXIST_1_ID)
112                 .toUriString();
113         String getActivityNotFound2 = UriComponentsBuilder.fromPath(GET_ACTIVITY_PATH)
114                 .buildAndExpand(NOT_EXIST_2_ID)
115                 .toUriString();
116         // when
117         ResponseEntity<Activity> result1 =
118                 testRestTemplate.getForEntity(getActivityNotFound1, Activity.class);
119         ResponseEntity<Activity> result2 =
120                 testRestTemplate.getForEntity(getActivityNotFound2, Activity.class);
121         // Then
122         assertThat(result1.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
123         assertThat(result2.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
124     }
```

Snippet 97

Por otro lado, también se comprueba que las peticiones a algún servicio sean correctas, por ejemplo, que se trate de exportar una actividad deportiva a un formato que el sistema no soporta. En ese caso, como se muestra en el Snippet 98, el servicio debe enviar una respuesta con un valor 400 y el código *bad_request*.

```
173     @Test
174     public void exportAsUnknownXmlTest() {
175         // Given
176         String unknownXmlType = "kml";
177         String exportAsNonKnownFile = UriComponentsBuilder.fromPath(EXPORT_AS_PATH)
178                 .buildAndExpand(ACTIVITY_GPX_ID, unknownXmlType)
179                 .toUriString();
180         // When
181         ResponseEntity<String> result =
182                 testRestTemplate.getForEntity(exportAsNonKnownFile, String.class);
183         // Then
184         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
185     }
```

Snippet 98

Como se comentó más arriba, el objeto requestEntity es importante para realizar aquellas peticiones http que tienen el verbo *put*. Por ejemplo, en el Snippet 99, se muestra un trozo de código que comprueba el resultado que produce la aplicación cuando se trata de unir dos tramos de una actividad física. En este caso, se espera que el resultado sea correcto y la actividad física no sea igual a lo que había anteriormente en el objeto *beforeTestCase,* puesto que se espera que dos de los segmentos que había anteriormente estén unidos.

```
317     @Test
318     public void joinLapsTest() {
319         // Given
320         Activity joinLapsActivity = toActivity(joinLapsTcxJsonResource);
321         String index1 = "0";
322         String index2 = "1";
323
324         String removeLapExistentActivityPath = UriComponentsBuilder
325                 .fromPath(JOIN_LAPS_PATH)
326                 .queryParam("index1", index1)
327                 .queryParam("index2", index2)
328                 .buildAndExpand(ACTIVITY_TCX_4_ID)
329                 .toUriString();
330
331         Optional<Activity> beforeTestCase =
332                 activityMongoRepository.findById(ACTIVITY_TCX_4_ID);
333         // When
334         ResponseEntity<Activity> result = testRestTemplate
335                 .exchange(removeLapExistentActivityPath,
336                         HttpMethod.PUT, requestEntity, Activity.class);
337         Optional<Activity> afterTestCase =
338                 activityMongoRepository.findById(ACTIVITY_TCX_4_ID);
339         // Then
340         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
341         assertThat(result.getBody()).isEqualTo(joinLapsActivity);
342         assertThat(afterTestCase).isNotEqualTo(beforeTestCase);
343     }
```

Snippet 99

La clase que se muestra en el Snippet 101 pertenece a la otra clase de pruebas de integración de la aplicación que realizan las pruebas de toda la funcionalidad que tiene que ver con los ficheros de actividades deportivas. En esta clase se utilizan otras dos herramientas para poder simular toda la plataforma que necesitan la aplicación.

Una de ellas es *WireMockClassRule*, una clase que pertenece a una librería creada por un usuario de GitHub[115]. En el Snippet 100 se muestra cómo se define la dependencia de esta liberería en el *pom*.

```
140       <dependency>
141                 <groupId>org.springframework.cloud</groupId>
142                 <artifactId>spring-cloud-contract-wiremock</artifactId>
143                 <version>${spring.wiremock.version}</version>
144                 <scope>test</scope>
145       </dependency>
```

Snippet 100

Lo que permite esta clase es la posibilidad de levantar un servicio web en un puerto específico o aleatorio de la máquina donde se están ejecutando los tests y definir unas respuestas para unas peticiones específicas al servicio. En este caso, como se puede ver en el trozo de código, es necesario utilizar la anotación **`@ClassRule`** que manejará todo este objeto. Es aquí donde se le indica que el puerto será dinámico y que se va a desplegar en la dirección *localhost* del equipo. En el ejemplo, esta utilidad sirve para definir las peticiones que se le hacen a la API de Google para obtener la altitud en un determinado punto específico. Para ello, se han modificado las propiedades de forma dinámica, indicando que la dirección host y el puerto del servicio que obtiene la altitud ahora será el definido en el objeto estático *googleApiWireMockClass*. Todo esto se realiza mediante la clase que esta declarada dentro como *Initializer* y que extiende de la clase *ApplicationContextInitializer*. Para hacer uso de esta clase e inicializar los valores de las propiedades definidas dentro, hay que usar la anotación **`@ContextConfiguration`** definiéndola en el parámetro *initializers*.

```
60    @ContextConfiguration(initializers = FileRestControllerIntegrationTest.Initializer.class)
61    public class FileRestControllerIntegrationTest extends IntegrationTest {
62
63        @ClassRule
64        public static LocalStackContainer localStackS3 =
65                new LocalStackContainer().withServices(S3);
66
67        @ClassRule
68        public static WireMockClassRule googleApiWireMockClass =
69                new WireMockClassRule(options()
70                        .dynamicPort()
71                        .bindAddress(LOCALHOST_HOST_NAME));
72
73        @Value("classpath:input/coruna.gpx.xml")
74        private Resource gpxXmlResource;
75        @Value("classpath:input/oviedo.tcx.xml")
76        private Resource tcxXmlResource;
77
78        @Autowired
79        private OriginalActivityRepository originalActivityRepository;
80        @Autowired
81        private ActivityMongoRepository activityMongoRepository;
82
83        @AfterClass
84        public static void shutDown() {
85            localStackS3.stop();
86            googleApiWireMockClass.stop();
87            localStackS3.close();
88        }
89
90        static class Initializer implements
91                ApplicationContextInitializer<ConfigurableApplicationContext> {
92            @Override
93            public void initialize(ConfigurableApplicationContext ctx) {
94                TestPropertyValues.of(
95                        "aws.s3-bucket" + BUCKET_NAME_TEST,
96                        "google-maps-api.elevation-protocol:http",
```

[115] https://github.com/tomakehurst/wiremock

```
97                      "google-maps-api.elevation-host: " +
98                              format("%s:%d", LOCALHOST_HOST_NAME,
99                                      googleApiWireMockClass.port()),
100                     "google-maps-api.elevation-endpoint: "
101                             + GOOGLE_MAPS_ENDPOINT_PATH,
102                     "google-maps-api.encrypted-api-key: "
103                             + ENCRYPTED_API_KEY
104             ).applyTo(ctx);
105         }
106     }
107
108     @TestConfiguration
109     public static class AWSTestConfiguration {
110         @Bean
111         @Primary
112         public AmazonS3 s3client() {
113             AmazonS3 amazonS3 = AmazonS3ClientBuilder.standard()
114                     .withEndpointConfiguration(localStackS3.getEndpointConfiguration(S3))
115                     .withCredentials(localStackS3.getDefaultCredentialsProvider())
116                     .withPathStyleAccessEnabled(true)
117                     .disableChunkedEncoding()
118                     .enablePathStyleAccess()
119                     .build();
120             amazonS3.createBucket(BUCKET_NAME_TEST);
121             return amazonS3;
122         }
123     }
124     ...
125 }
```

Snippet 101

Otra de las herramientas que se utiliza en esta clase de pruebas de integración es el uso de la librería *localstack*[116] para desplegar toda la plataforma de AWS haciendo uso también de los contenedores de Docker. En el Snippet 102 se muestra la dependencia que está definida en el fichero *pom* para poder hacer uso de todas las clases que proporciona. Esta compañía, como indica en su página web[117], tiene diferentes modelos de pago, sin embargo, en este caso será más que suficiente con la versión gratuita.

```
157     <dependency>
158             <groupId>org.testcontainers</groupId>
159             <artifactId>localstack</artifactId>
160             <version>${testcontainers.version}</version>
161             <scope>test</scope>
162     </dependency>
```

Snippet 102

Para hacer uso de toda esta funcionalidad, también se declara el objeto estático *localStackS3* con la anotación **`@ClassRule`** para que sea gestionada por la clase de prueba. Además, se le indican los servicios que se van a utilizar para que pueda desplegarlos en el contenedor, en el caso de nuestra aplicación tan solo es necesario el servicio S3 de Amazon Web Services. Una vez levantado el servicio, hay que crear el vean que permita el acceso a toda esta funcionalidad a través de la librería de AWS para Java. Para realizar esto, es necesario activar la propiedad de sobreescritura de *beans* como se comentó anteriormente en el Snippet 91. Para la creación de este objeto, se define la clase *AWSTestConfiguration* con la anotación @**`TestConfiguration`**. En esta clase se define el objeto que será instanciado en el contexto de Spring gracias al uso de la anotación **`@Bean`**. Este objeto será creado pasándole las propiedades del contenedor donde está alojada la plataforma de AWS con el servicio S3 desplegado, como se dijo más arriba.

[116] https://github.com/localstack/localstack
[117] https://localstack.cloud

```
@Test
public void uploadGPXFileTest() {
     // Given
     LinkedMultiValueMap<String, Object> parameters = new LinkedMultiValueMap<>();
     parameters.add("file",
          new ClassPathResource(format("%s/%s", "input", CORUNA_XML_FILE)));
     HttpHeaders headers = new HttpHeaders();
     headers.setContentType(MediaType.MULTIPART_FORM_DATA);
     HttpEntity<LinkedMultiValueMap<String, Object>>
          requestEntity = new HttpEntity<>(parameters, headers);
     String uploadGpxXmlFile = UriComponentsBuilder.fromPath(UPLOAD_FILE_PATH)
          .queryParam("type", SOURCE_GPX_XML)
          .build()
          .toUriString();
     // When
     ResponseEntity<String> result =
          testRestTemplate.exchange(uploadGpxXmlFile, POST, requestEntity, String.class);
     List<String> ids = getIds(result.getBody());
     Optional<Activity> afterTestCase = activityMongoRepository.findById(ids.get(0));
     Optional<S3ObjectInputStream> fileStored =
          originalActivityRepository.getFile(
               format("%s.%s", ids.get(0), SOURCE_GPX_XML));
     // Then
     assertThat(result.getStatusCode()).isEqualTo(HttpStatus.CREATED);
     assertThat(afterTestCase).isNotEmpty();
     assertThat(fileStored).isNotEmpty();
}
```

Snippet 103

Una vez que han sido definidas todas estas herramientas, se puede definir cada prueba de integración. Por ejemplo, en el Snippet 103 se muestra el código que comprueba que la subida de una actividad con un formato *gpx* se lleva a cabo de forma correcta. En este caso, se comprueba que la respuesta del servicio tiene como valor un 201 y código *created*. Otra de las cosas que se comprueba es que exista en la base de datos creada anteriormente en un contenedor de Docker de la clase padre y también se haya guardado el fichero original en el servicio S3 de Amazon.

Por otro lado, esta clase también comprueba que, al subir una actividad deportiva, si no hay datos de la altitud, la aplicación haga uso del servicio para obtener el número de metros de elevación en función de la latitud y longitud. En el Snippet 104 se puede ver como se define la respuesta cuando se llama al servicio mediante el método estático *stubFor* de la clase *WireMock*. Una vez que se ha llamado al servicio, se comprueba además de lo que se comprobaba en la prueba del Snippet 103, también los valores nuevos de la altitud para cada punto que define el tramo de la actividad.

```
@Test
public void uploadWithoutElevationDataGpxFileTest() {
    // Given
    stubFor(get(urlEqualTo(format(ELEVATION_ENDPOINT,
                POSITIONS_URL_ENCODED, API_KEY)))
        .willReturn(ok()
            .withHeader(HttpHeaders.CONTENT_TYPE,
                MediaType.APPLICATION_JSON_VALUE)
            .withBodyFile(ELEVATION_STUBBING_RESPONSE)));
    LinkedMultiValueMap<String, Object> parameters = new LinkedMultiValueMap<>();
    parameters.add("file", new ClassPathResource(
         format("%s/%s", "input", GPX_WITHOUT_ELEVATION_FILE)));
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.MULTIPART_FORM_DATA);
    HttpEntity<LinkedMultiValueMap<String, Object>>
         requestEntity = new HttpEntity<>(parameters, headers);
    String uploadGpxXmlFile = UriComponentsBuilder.fromPath(UPLOAD_FILE_PATH)
        .queryParam("type", SOURCE_GPX_XML)
        .build()
        .toUriString();
    // When
    ResponseEntity<String> result =
```

```
217             testRestTemplate.exchange(uploadGpxXmlFile, POST, requestEntity, String.class);
218         List<String> ids = getIds(result.getBody());
219         Optional<Activity> afterTestCase = activityMongoRepository.findById(ids.get(0));
220         Optional<S3ObjectInputStream> fileStored =
221             originalActivityRepository.getFile(format("%s.%s", ids.get(0),
222                 SOURCE_GPX_XML));
223         // Then
224         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.CREATED);
225         assertThat(afterTestCase).isNotEmpty();
226         assertThat(afterTestCase.get().getLaps()
227                 .get(0).getTracks().get(0).getAltitudeMeters())
228             .isEqualTo(new BigDecimal("2.200000047683716"));
229         assertThat(afterTestCase.get().getLaps()
230                 .get(0).getTracks().get(1).getAltitudeMeters())
231             .isEqualTo(new BigDecimal("1.600000023841858"));
232         assertThat(afterTestCase.get().getLaps()
233                 .get(0).getTracks().get(2).getAltitudeMeters())
234             .isEqualTo(new BigDecimal("1.2000000476837158"));
235         assertThat(fileStored).isNotEmpty();
236     }
```

Snippet 104

Otra de las cosas que comprueba esta clase es que los ficheros originales pueden ser descargados correctamente. Para ello, como se muestra en este enlace [118], se ha desactivado la comprobación MD5 para que la prueba funcione correctamente debido a que debe ser que se deja algún espacio en blanco que hace que no sea el mismo fichero. Por otro lado, al principio, se sube el fichero que posteriormente se va a descargar. Al final, en el punto *Then*, se comprueba que el valor de la respuesta sea un 200 con el código *ok*. También se comprueba que el tipo de dato que se ha obtenido sea del tipo **`application/octet-stream`** que es todo aquel fichero binario[119], valga la redundancia, sin identificarse con algún tipo de extensión.

```
294     @Test
295     public void getTcxFileTest() {
296         // Given
297         // Upload xml file
298         System.setProperty(SkipMd5CheckStrategy
299                 .DISABLE_GET_OBJECT_MD5_VALIDATION_PROPERTY,"true");
300         Try.of(() -> IOUtils.toString(
301                 tcxXmlResource.getInputStream(), UTF_8).getBytes())
302                 .forEach(arrayBytes ->
303                         originalActivityRepository.uploadFile(arrayBytes,
304                                 format("%s.%s", TCX_ID_XML, SOURCE_TCX_XML)));
305         String getTcxFilePath = UriComponentsBuilder.fromPath(GET_FILE_PATH)
306                 .buildAndExpand(SOURCE_TCX_XML, TCX_ID_XML)
307                 .toUriString();
308         // When
309         ResponseEntity<String> result =
310                 testRestTemplate.getForEntity(getTcxFilePath, String.class);
311         // Then
312         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
313         assertThat(result.getBody()).isNotEmpty();
314         assertThat(result.getHeaders().getContentType())
315                 .isEqualTo(MediaType.APPLICATION_OCTET_STREAM);
316     }
```

Snippet 105

Haciendo un resumen, se han realizado 286 pruebas en todo el proyecto *backend*. De las cuales, 27 corresponden a pruebas de integración y 259 a pruebas unitarias. El IDE[120] *IntelliJ* permite obtener el porcentaje de cobertura que aportan tus pruebas al total de las clases, de los métodos y de las líneas de código, como se puede ver en la Tabla 15.

[118] https://github.com/localstack/localstack/issues/869
[119] https://en.wikipedia.org/wiki/Bitstream
[120] https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

Paquete	Clases %	Métodos %	Líneas %
all classes	84.4% (341/ 404)	49.3% (1004/ 2038)	60% (2480/ 4134)

Tabla 15

También permite ver, como se muestra en la Tabla 16, estos porcentajes de cobertura, pero en función de cada paquete. Muchas de los paquetes que tienen modelo de objetos no tienen una alta cobertura porque no contiene ningún tipo de lógica.

Paquete	Clases %	Métodos %	Líneas %
com.routeanalyzer.api	100% (1/ 1)	50% (1/ 2)	33.3% (1/ 3)
com.routeanalyzer.api.common	100% (6/ 6)	87.5% (49/ 56)	95.8% (160/ 167)
com.routeanalyzer.api.config	100% (8/ 8)	84.4% (27/ 32)	90.9% (50/ 55)
com.routeanalyzer.api.controller	100% (5/ 5)	82.9% (29/ 35)	81.4% (48/ 59)
com.routeanalyzer.api.facade.impl	100% (2/ 2)	100% (14/ 14)	100% (47/ 47)
com.routeanalyzer.api.logic.file.export	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
com.routeanalyzer.api.logic.file.export.impl	100% (2/ 2)	100% (34/ 34)	100% (234/ 234)
com.routeanalyzer.api.logic.file.upload	100% (1/ 1)	100% (4/ 4)	100% (10/ 10)
com.routeanalyzer.api.logic.file.upload.impl	100% (2/ 2)	70% (35/ 50)	70.2% (198/ 282)
com.routeanalyzer.api.logic.impl	100% (4/ 4)	98.4% (184/ 187)	98.1% (622/ 634)
com.routeanalyzer.api.model	100% (8/ 8)	84.5% (49/ 58)	72.7% (64/ 88)
com.routeanalyzer.api.model.exception	100% (5/ 5)	85.7% (12/ 14)	85.7% (12/ 14)
com.routeanalyzer.api.services.aws	100% (1/ 1)	87.5% (7/ 8)	96% (24/ 25)
com.routeanalyzer.api.services.googlemaps	100% (2/ 2)	90.9% (20/ 22)	96.5% (55/ 57)
com.routeanalyzer.api.services.googlemaps.model	50% (3/ 6)	39.1% (9/ 23)	39.1% (9/ 23)
com.routeanalyzer.api.services.reader	100% (3/ 3)	100% (15/ 15)	100% (35/ 35)
com.routeanalyzer.api.xml.gpx11	91.6% (87/ 95)	36.8% (154/ 419)	39.6% (271/ 684)
com.routeanalyzer.api.xml.gpx11.trackpointextension.garmin	90% (9/ 10)	34.1% (14/ 41)	37.5% (24/ 64)
com.routeanalyzer.api.xml.tcx	77.6% (173/ 223)	32.5% (305/ 938)	36.3% (549/ 1512)
com.routeanalyzer.api.xml.tcx.activityextension	94.7% (18/ 19)	47.6% (40/ 84)	45.2% (61/ 135)

Tabla 16

Frontend

A la hora de probar la aplicación React que se ha desarrollado, existen diferentes formas para comprobar el correcto funcionamiento de cada componente del sistema que se han mostrado en la Ilustración 19. Por un lado, las pruebas "*end-to-end*" que se lleva a cabo a través del navegador web ejecutando la aplicación completa. Por otro, las que se realizan mediante el renderizado del árbol de componentes en un entorno de pruebas. Las pruebas que se van a exponer en este documento se centran en las últimas en las que se simula un entorno simplificado de testing.

Para ello, se utiliza una librería de JavaScript, llamada Jest[121], que introduce un marco de trabajo para la realización de pruebas de forma sencilla. Centrada en tener una configuración sencilla, una diversa interfaz con diferentes tipos de métodos, la posibilidad de hacer pruebas de snapshots y pruebas aisladas unas de otras. Estas pruebas permiten renderizar todos los componentes a probar mediante el uso de la librería jsdom[122] que permite emula la funcionalidad de un navegador a la hora de desplegar los elementos y, por tanto, permitiendo definir interacciones con el DOM y esperar que sucedan diferentes resultados causados por estas acciones.

Para configurar todo este software de testing tan solo hay que definir añadir las dependencias en el **`package.json`** como se muestra en el Snippet 106. En este caso habría que definir las dependencias en el campo *devDependencies* ya que solo van a ser necesarias en las fases previas a la construcción del fichero ejecutable. Para ello, se hace uso de las dependencias que **`@babel/preset-env`**, **`@babel/preset-react`**, **`babel-jest`**, **`jest`**, **`react-test-render`** y **`jsdom`**, entre otros.

```
40     "devDependencies": {
41       "@babel/plugin-proposal-class-properties": "^7.4.4",
42       "@babel/plugin-transform-async-to-generator": "^7.7.4",
43       "@babel/polyfill": "^7.0.0",
44       "@babel/preset-env": "^7.4.5",
45       "@babel/preset-react": "^7.0.0",
46       "@testing-library/react": "^8.0.4",
47       "babel-jest": "^24.9.0",
48       "babel-preset-env": "^1.7.0",
49       "babel-preset-react": "^6.24.1",
50       "blob": "0.0.5",
51       "enzyme": "^3.10.0",
52       "enzyme-adapter-react-16": "^1.15.1",
53       "enzyme-to-json": "^3.3.5",
54       "identity-obj-proxy": "^3.0.0",
55       "jsdom": "^15.2.1",
56       "jest": "^24.9.0",
57       "jest-cli": "^24.9.0",
58       "jest-dom": "^3.5.0",
59       "jest-environment-jsdom-fourteen": "^0.1.0",
60       "jest-enzyme": "^7.1.2",
61       "jest-mock-random": "^1.0.3",
62       "flush-promises": "^1.0.2",
63       "react-redux": "^5.0.7",
64       "react-test-renderer": "^16.8.6",
65       "redux": "^3.7.2",
66       "redux-mock-store": "^1.5.3",
67       "regenerator-runtime": "^0.13.2",
68       "supertest": "^4.0.2"
69     }
```

Snippet 106

[121] https://jestjs.io
[122] https://github.com/jsdom/jsdom

Una vez definidas las dependencias necesarias para hacer funcionar la librería, tan solo habrá que indicar el comando para poder ejecutar los tests desde el mismo fichero que se utilizó para definir las dependencias necesarias para las pruebas.

```
34     "scripts": {
35       "start": "react-scripts start",
36       "build": "react-scripts build",
37       "test": "react-scripts test --env=jsdom --config=jest-config.js",
38       "eject": "react-scripts eject"
39     }
```

Snippet 107

En el fichero **`package.json`**, como se define en el Snippet 107, se indica el entorno donde se realizarán las pruebas comentado anteriormente usando la librería jsdom y la configuración [123] que utilizarán los test definida en el fichero **`jest-config.js`** del directorio raíz del proyecto. En el Snippet 108 se muestra este fichero que contiene principalmente una línea donde se importan la configuración por defecto y se usa para definir las extensiones que usarán los módulos. Además de los de por defecto, también otros definidos en la propiedad *moduleFileExtensions*.

```
1  const {defaults} = require('jest-config');
2  module.exports = {
3    // An array of file extensions your modules use
4    moduleFileExtensions: [...defaults.moduleFileExtensions, 'ts', 'tsx', 'js', 'map'],
5    moduleNameMapper: {
6       "\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$":
7           "<rootDir>/__mocks__/fileMock.js",
8       "\\.(scss|sass|css)$": "identity-obj-proxy"
9    },
10   setupFilesAfterEnv: ["<rootDir>/src/setUpTests.js"],
11   roots: ["src"]
12 };
```

Snippet 108

Otra de las otras cosas que se realizan es que cualquier fichero multimedia sea *mockeado* usando el fichero JavaScript llamado *fileMock* que se encuentra en la carpeta **`__mocks__`** y tan solo indicando esta línea de código: `module.exports = 'test-file-stub';`. También se realiza lo mismo para cualquier hoja de estilo mediante el uso de una librería definida en el documento mostrado en el Snippet 106, identificada como **`identity-obj-proxy`**[124]. Por otro lado, también se indica el directorio raíz definido como *src* donde se tendrán que buscar las clases de prueba identificadas con el sufijo "*.test.js*". Por último, se indica un fichero que se ejecutará antes de que se lancen todas las pruebas pertenecientes a un fichero, se denominado *setUpTests*.

```
1    import { configure } from 'enzyme/build';
2    import Adapter from 'enzyme-adapter-react-16/build';
3    import 'jest-enzyme';
4    // react-testing-library renders your components to document.body,
5    // this will ensure they're removed after each test.
6    import '@testing-library/react/cleanup-after-each';
7    // this adds jest-dom's custom assertions
8    import 'jest-dom/extend-expect';
9
10   configure({ adapter: new Adapter() });
```

Snippet 109

Este fichero se encuentra definido en el Snippet 109, en el se hacen uso de dos librerías más, llamadas **`enzyme`**[125] y **`@testing-library/react`**, e importadas en las dependencias de

[123] https://jestjs.io/docs/en/configuration
[124] https://www.npmjs.com/package/identity-obj-proxy
[125] https://github.com/enzymejs/enzyme

desarrollo como se indicó en el Snippet 106. La primera permite comprobar de forma sencilla la salida de un componente de React, pudiendo realizar operaciones con esta salida. La segunda es una librería de utilidades de pruebas para React que proporciona una manera sencilla de comprobar componentes sin depender de detalles de implementación. En este caso, en el fichero del Snippet 109 se indica, por un lado, importando el cleanup-after-each que se borre todos los componentes renderizados que cuelgan del cuerpo del documento en el DOM y, por otro, se preparan los tests para poder hacer uso de la funcionalidad que proporciona la librería enzyme.

En el Snippet 110 se muestra la clase, llamada *App.test.js*, que prueba el componente raíz. En este ejemplo muy sencillo, tan solo se comprueba que para un estado concreto se renderizan los componentes oportunos. Es decir, se comprueba que se visualiza cuando no hay identificador de ruta ni está en progreso, cuando se está procesando la carga de una ruta y cuando existe una ruta a visualizar.

```
8    describe('App main component', () => {
9      it('Initial render without crashing', () => {
10         const props = {
11             showRoute: false,
12             progress: false,
13             id: null
14         };
15         const wrapper = shallow(<App {...props} />);
16         expect(wrapper.find(CircularProgress).length).toBe(0);
17         expect(wrapper.find(FileUploaderContainer).length).toBe(1);
18         expect(wrapper.find(RouteContainer).length).toBe(0);
19         expect(wrapper).toMatchSnapshot();
20     });
21     it('Loading some activity render', () => {
22           const props = {
23               showRoute: false,
24               progress: true,
25               id: null
26           };
27           const wrapper = shallow(<App {...props} />);
28           expect(wrapper.find(CircularProgress).length).toBe(1);
29           expect(wrapper.find(FileUploaderContainer).length).toBe(1);
30           expect(wrapper.find(RouteContainer).length).toBe(0);
31     });
32     it('Some activity loaded render', () => {
33           const idActivity = "1234567890";
34           const props = {
35               showRoute: true,
36               progress: false,
37               id: idActivity
38           };
39           const wrapper = shallow(<App {...props} />);
40           expect(wrapper.find(CircularProgress).length).toBe(0);
41           expect(wrapper.find(FileUploaderContainer).length).toBe(1);
42           const routeContainer = wrapper.find(RouteContainer);
43           expect(routeContainer.length).toBe(1);
44           expect(routeContainer.props().id).toBe(idActivity);
45     });
46   });
```

Snippet 110

Al igual que ocurría en las pruebas descritas en el punto *Backend* del apartado *Pruebas*, en este *framework* de desarrollo también se pueden definir ciertos trozos de código que se ejecutarán antes de cada método usando los métodos *beforeEach*/*afterEach* o antes de la suite de test con el método *beforeAll*, como se indica en el Snippet 111.

```
16       beforeAll(() => {
17           activityId = "5ace8cd14c147400048aa6b0";
18       });
19       beforeEach(() => {
20           toggleProgressMock = jest.fn();
21           actionShowRouteMock = jest.fn();
22           appendMockFn = jest.fn();
```

```
23          window.FormData = jest.fn(() => ({ append: appendMockFn }));
24          preventDefaultMock = { preventDefault: jest.fn(() => ({}))};
27      });
28      afterEach(() => {
29          jest.resetAllMocks();
30      });
```

Snippet 111

Este trozo de código corresponde a la clase que comprueba el correcto funcionamiento del componente *FileUploaderContainer*. En el se inicializa un identificador de una actividad física que será válido para toda la clase de prueba. Por otro lado, se mockean las acciones de Redux que utiliza el componente gracias al uso de la función *fn* de la librería Jest, después de que se haya ejecutado cada método de prueba, todos los mocks realizados se restaurarán.

```
31      it('Render initialization successfully', () => {
32          const component = shallow(
33              <FileUploaderContainer toggleProgress={toggleProgressMock}
34                                     actionShowRoute={actionShowRouteMock} />);
35          expect(toJson(component)).toMatchSnapshot();
36      });
37      it('FormData is called with right tcx param and file.', async () => {
38          const component = mount(
39              <FileUploaderContainer toggleProgress={toggleProgressMock}
40                                     actionShowRoute={actionShowRouteMock} />);
41          fileServices.upload.mockResolvedValueOnce(jest.fn());
42          const mockFileName = "Mock file name";
43          const fileInputMock = {
44              files: [mockFileName]
45          };
46          component.instance().fileInput = fileInputMock;
47          const tcxTypeMock = { value: "tcx" }
48          component.instance().type = tcxTypeMock;
49
50          const form = component.find('form');
51          await form.simulate('submit', preventDefaultMock);
52          expect(appendMockFn).toBeCalledTimes(2);
53          expect(appendMockFn).toBeCalledWith("file", mockFileName);
54          expect(appendMockFn).toBeCalledWith("type", tcxTypeMock.value);
55
56          const formData = new FormData();
57          formData.append('file', mockFileName);
58          formData.append('type', tcxTypeMock.value);
59          expect(fileServices.upload).toBeCalledWith(formData);
60      });
61      it('Handling submit file.', async () => {
62          const component = mount(
63              <FileUploaderContainer toggleProgress={toggleProgressMock}
64                                     actionShowRoute={actionShowRouteMock} />);
65          fileServices.upload.mockResolvedValueOnce(activityId);
66
67          const form = component.find('form');
68          await form.simulate('submit', preventDefaultMock);
69
70          expect(preventDefaultMock.preventDefault).toBeCalled();
71          expect(toggleProgressMock).toBeCalledWith(true);
72          expect(fileServices.upload).toBeCalled();
73          expect(actionShowRouteMock).toBeCalledWith("5ace8cd14c147400048aa6b0");
74          expect(actionShowRouteMock).not.toBeCalledWith(null);
75      });
76      it('Error trying to upload some file', async () => {
77          const component = mount(
78              <FileUploaderContainer toggleProgress={toggleProgressMock}
79                                     actionShowRoute={actionShowRouteMock} />);
80          window.alert = jest.fn(() => ({}));
81          const error = { message: "Error mocked" };
82          fileServices.upload.mockRejectedValueOnce(error);
83
84          const form = component.find('form');
85          await form.simulate('submit', preventDefaultMock);
86          await flushPromises();
87          expect(preventDefaultMock.preventDefault).toBeCalled();
88          expect(toggleProgressMock).toBeCalledWith(true);
89          expect(fileServices.upload).toBeCalled();
90          expect(actionShowRouteMock).not.toBeCalledWith("5ace8cd14c147400048aa6b0");
```

```
91            expect(window.alert).toBeCalledWith(error.message);
92            expect(actionShowRouteMock).toBeCalledWith(null);
93        });
```

Snippet 112

El primer método es útil para comprobar que la interfaz gráfica no ha cambiado, gracias a pruebas de *snapshot*, mediante la comprobación del resultado con un fichero *json* almacenando previamente, la primera vez que se ejecutó este método de prueba. Por otro lado, para poder operar con el resultado del método shallow, de la librería enzyme, y poder realizar las comprobaciones de que no se ha modificado, se ha de incluir una librería llamada **`enzyme-to-json`** en el package.json como se muestra en el Snippet 113. Esta librería permite mapear el objeto resultante de la función *shallow*, en este caso, a formato json. Este método de la librería *enzyme* lo que realiza es el renderizado de manera superficial, de forma unitaria sobre un componente, sin depender en los componentes hijos que contenga.

```
70      "jest": {
71        "snapshotSerializers": [
72          "enzyme-to-json/serializer"
73        ]
74      }
```

Snippet 113

En el Snippet 112 se ven otros tres métodos de prueba que comprueban otras cosas. Para ellos, en ambos casos, a diferencia del primero, se utiliza el método mount de la librería *enzyme* que lo que hace es un renderizado completo de todo el componente en el DOM, para ello, como se comentó anteriormente, es necesario que esté presente la librería *jsdom*. En las tres se mockea la respuesta de la función *upload* de la clase *file* del directorio Services para que pueda devolver diferentes respuestas en función de si todo ha ido bien o no. Por un lado, si la respuesta es satisfactoria se utiliza el método *mockResolvedValueOnce* del paquete **`jest-mock`**, de lo contrario, si se quiere simular una respuesta errónea se usa *mockRejectedValueOnce*. Al final ambas funciones lo que hacen es mockear la respuesta de una *Promise*, no deja de ser azúcar sintáctico para las funciones que se muestran en el Snippet 114, respectivamente.

```
1   jest.fn().mockImplementationOnce(() => Promise.resolve(value));
2   jest.fn().mockImplementationOnce(() => Promise.reject(value));
```

Snippet 114

En el primer método de prueba se comprueba que el formulario funciona correctamente llamando a la función *upload* de la clase con los parámetros correspondientes. En el segundo se mira que la función devuelve una actividad gracias al mockeo del método *upload* del fichero file que devuelve un identificador predefinido de una actividad. En el tercer método se comprueba que el componente funciona como es de esperar cuando se produce algún error cuando se trata de subir una actividad física. En este caso se necesita usar una librería llamada **`flush-promises`** para pue todas las promesas hayan sido ejecutadas antes de realizar las comprobaciones.

```
17  export function get(id, type) {
18     const path = DIR_BASE + "/file/get/"+type+"/"+id;
19      return axios
20        .get(path)
21        .then(res => res.data)
22        .catch(err => {
23            console.log(err);
24            throw new Exception(err.response.data.description);
25        });
26  }
```

Snippet 115

Es así porque, si vemos el método *upload* de la clase file que se muestra en el Snippet 115, existen dos promesas encadenadas usando la librería axios[126] que sirve para realizar llamadas *http* por medio de promesas. Si en la tercera prueba del Snippet 112 no se indica el método *flushPromises* con la palabra reservada **`await`**, la prueba falla porque no se llega a ejecutar nunca la promesa manejadora de los errores identificada en el método del Snippet 115 como *catch*.

Al igual que ocurría en las pruebas de *Backend*, jest ofrece una funcionalidad para proporcionar un reporte de la cobertura de las pruebas realizadas. Para ello, tan solo hay que ejecutar el comando test con el parámetro **`--coverage`**. Este comando genera una carpeta *coverage* en el directorio raíz del proyecto.

All files							
Statements		**Branches**		**Functions**		**Lines**	
59.97%	454/757	*36.03%*	98/272	*61.29%*	190/310	*58.87%*	415/705

Tabla 17

En la Tabla 17 se muestran datos generales de la cobertura de todas las pruebas dentro del proyecto. Para ello, se centra en definir qué porcentaje/número total de ciertos aspectos se han cubiertos con la ejecución de las pruebas:

- Lo que se indica como *Statements* corresponde a los valores de sentencias cubiertas.
- Lo que se define como *Branches* corresponde a lo valores de declaraciones lógicas que se han cubierto, si las pruebas abarcan todas las posibilidades.
- El que se define como Functions representan los valores para las funciones.
- Lo que se muestra como Lines representan los valores para las líneas de código.

File	**Statements**		**Branches**		**Functions**		**Lines**	
src	60%	3/5	100%	4/4	33.33%	1/3	75%	3/4
src/Components	26.73%	89/333	2.36%	3/127	26.39%	38/144	27.3%	89/326
src/Components/Charts	100%	36/36	93.33%	28/30	100%	19/19	100%	26/26
src/Components/Charts/Axis	69.57%	16/23	57.14%	4/7	80%	4/5	77.78%	14/18
src/Components/Charts/Data	77.78%	42/54	50%	3/6	80%	24/30	73.91%	34/46

[126] https://github.com/axios/axios

File	Statements		Branches		Functions		Lines	
src/Components/Charts/Tooltip	40.68%	24/59	9.38%	3/32	28.57%	2/7	35.29%	18/51
src/Services	100%	64/64	64.29%	9/14	100%	30/30	100%	64/64
src/Utils	98.55%	136/138	88.1%	37/42	100%	48/48	98.45%	127/129
src/__test__	100%	22/22	100%	0/0	100%	18/18	100%	22/22
src/actions	100%	8/8	100%	0/0	100%	4/4	100%	4/4
src/constants	100%	4/4	100%	0/0	100%	0/0	100%	4/4
src/reducers	90.91%	10/11	70%	7/10	100%	2/2	90.91%	10/11

Tabla 18

En la Tabla 18 se muestran todos estos indicadores, pero separados por directorios de la aplicación, desde el fichero raíz identificado como *src* hasta todos los demás hijos.

Despliegue

En la Ilustración 22 se puede ver como está el sistema desplegado. A simple vista, se pueden identificar 5 nodos que permiten que el sistema pueda funcionar con total normalidad. A continuación, se van a enumerar cada uno de ellos.

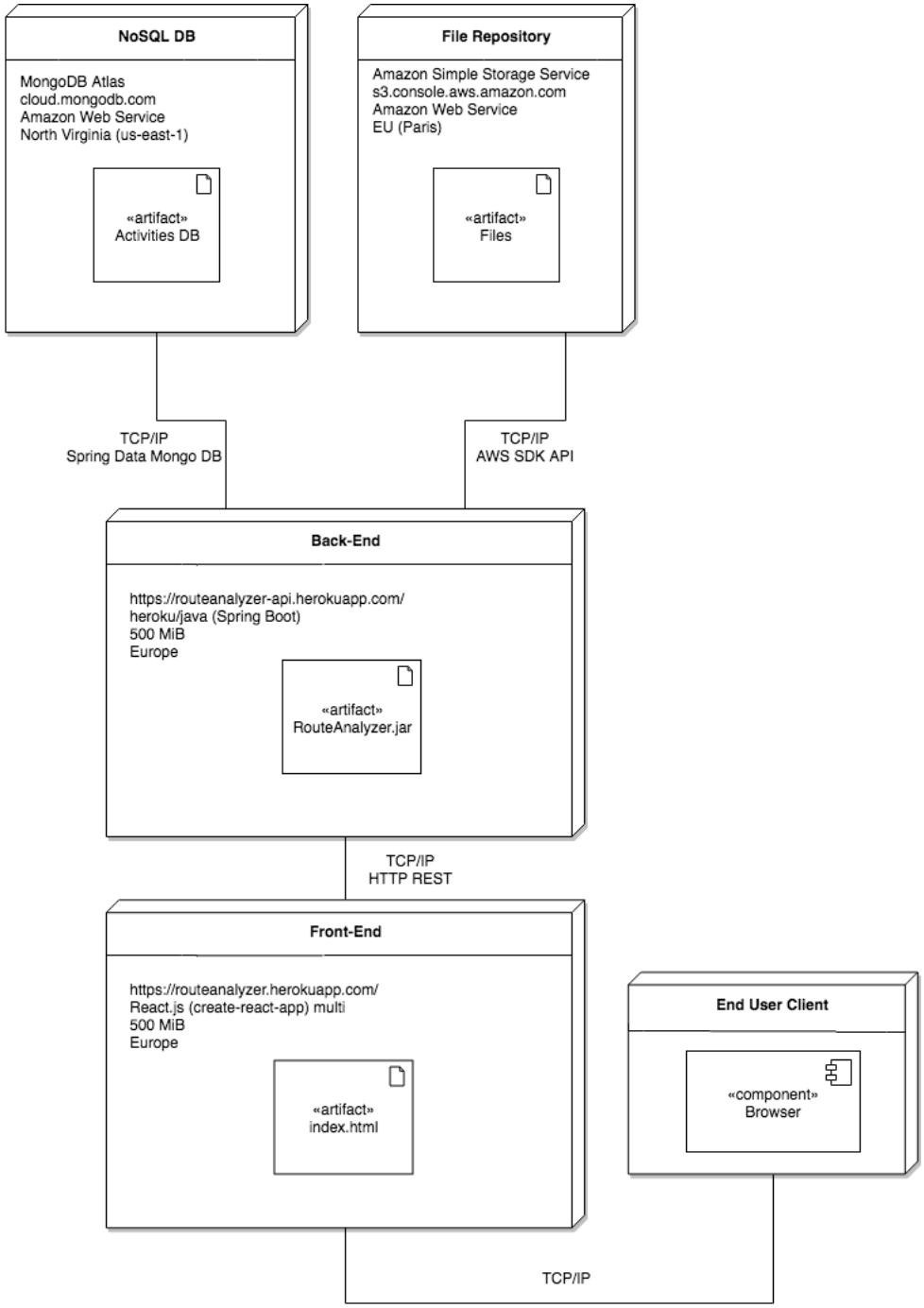


Ilustración 22

El nodo que hace referencia al servidor donde está alojada la base de datos *NoSQL*. Se ha desplegado con una plataforma llamada Mongo DB Atlas que proporciona una manera

simple de desplegar, gestionar y escalar una base de datos en la nube. La creación de un nuevo *cluster* es tan fácil como seguir los sencillos pasos que el propio sitio web muestra: lo primero de ello es seleccionar la región donde estará alojado físicamente el servidor y, además, seleccionar cuál será la compañía con la que se desea tener el servidor. En la actualidad, la plataforma ofrece tres marcas de *hosting* de servidores: Amazon Web Services, Google Cloud Platform y Microsoft Azure; sin embargo, tan solo en los dos primeros es posible tener un plan gratuito, sin tener que realizar ni un solo ingreso, es por esto, que se ha contratado el servicio con AWS. Una vez hecho este primer paso, tan solo hay que indicar la RAM y el tamaño de disco duro que se quiere tener para el servidor, en este caso, se ha escogido el plan gratuito, es decir, la memoria RAM es compartida y el tamaño de disco duro es de 512MB. Una vez hechos todos estos pasos en función de las características que más se ajusten a tus necesidades, es hora de especificar la versión de Mongo que se va a instalar y el nombre del *cluster* que se va a crear, en el caso de la aplicación que se ha desarrollado, el nombre del *cluster* establecido es *RouteAnalyzerDB* y la versión de Mongo es la 3.6.

El nodo que hace referencia al servidor donde está alojado el repositorio de ficheros de actividades deportivas. La plataforma seleccionada, en este caso, es Amazon Web Service usando el módulo AS3. La conexión a este modulo se realiza mediante conexión TCP/IP usando los diferentes SDK que existen para cada uno de los lenguajes más populares en la actualidad. En el caso de la aplicación que se ha desarrollado, se usa en el back-end la librería Java proporcionada por Amazon.

El nodo que contiene la aplicación del Back-End, es decir, la API de la aplicación a la que conectarse para poder realizar toda la lógica de negocio del sistema. Esta aplicación está desplegada en la plataforma llamada Heroku. La forma de conexión con ella es mediante TCP/IP usando los servicios web http REST.

El nodo que contiene la aplicación de la interfaz del sistema, también, al igual que la capa de la lógica, está desplegado en la plataforma Heroku, sin embargo, el contenedor de la aplicación está desarrollado para que funcione con aplicaciones de React. La forma de conectarse a este servicio es mediante TCP/IP usando el protocolo http desde un navegador web.

El nodo que hace referencia al usuario final el cual puede conectarse al sistema a través de un navegador web. Existiría la posibilidad de conectarse también al back-end directamente desde este nodo, sin embargo, no tiene una funcionalidad real a no ser que se conozca a la perfección cómo funcionan las “tripas” de los sistemas.

Conclusiones y ampliaciones

En lo que respecta a mejoras funcionales, como se definió en el punto *Especificación de Requisitos* en el apartado *Planificación*, podrían realizarse las tres historias de usuario comentadas en la Ilustración 2.

La primera de ellas pertenece a la creación de un registro de usuarios que permita la autenticación en el sistema a los usuarios, mostrándole a cada uno de ellos, las actividades subidas anteriormente. Para ello, este cambio también involucraría la redefinición del modelo de base de datos para que pueda identificarse cada actividad deportiva con el usuario que corresponda. A nivel técnico, una solución muy efectiva sería tener todo el proceso de autenticación externalizado como un servicio, lo que se conoce en inglés como Authentication as a Service. Al estar usando el servicio S3 para el almacenamiento de ficheros originales, sería muy sencillo implementar esta funcionalidad con el uso del servicio Cognito[127] que también pertenece a Amazon Web Services. Como se muestra en la página web de la compañía, este servicio permitiría incluir de manera sencilla el control de acceso, el registro y el inicio de sesión de los usuarios.

La segunda historia de usuario hace referencia a poder importar una actividad que se haya registrado en alguna aplicación de terceros, de las que se han descrito en el apartado *Introducción*, para el posterior uso en el sistema que se ha desarrollado. Por ejemplo, habría que investigar si cada empresa tiene una API pública para poder acceder de manera fácil a los recursos que contienen. Por ejemplo, la aplicación de Runtastic, como se indicó en este enlace[128] de 2016, no tiene una API pública establecida por la propia empresa, sin embargo, se puede monitorizar los *endpoints* que usa la aplicación móvil para obtener los ficheros que contienen actividades físicas. Por otro lado, existen otras empresas como Strava[129] o Garmin[130] que sí poseen una interfaz pública para desarrolladores.

La tercera funcionalidad, mostrada en la Ilustración 2, es aquella que permitiría el compartir actividades deportivas entre usuarios del sistema. Para ellos, es necesario que la primera historia de usuario mostrada en este apartado haya sido desarrollada previamente ya que sin un sistema de autenticación no podríamos compartir carreras con nadie más que el usuario actual de la aplicación. Básicamente, esta funcionalidad generaría una red social en el sistema.

Habría que añadir una historia de usuario que, aunque no fue descrita en la Ilustración 2, permitiría definir un nuevo formato de actividad física que el sistema reconocería, mapearía al modelo de datos general y guardaría toda esta información en la base de datos. El formato de la actividad deportiva sería el que es denominado fit, como se describe en la documentación[131] de desarrolladores de Strava, y que tiene este acrónimo ya que la definición en inglés es *Flexible and Interoperable data Transfer*. En este enlace, además, se explica cómo funciona este esquema de documentos xml y contiene un enlace[132] que

[127] https://aws.amazon.com/es/cognito/
[128] https://mijailovic.net/2016/10/21/reversing-runtastic-api/
[129] https://developers.strava.com
[130] https://developer.garmin.com
[131] https://developers.strava.com/docs/uploads/
[132] https://www.thisisant.com/resources/fit

descarga diferentes datos sobre este tipo de fichero. La historia de usuario a incluir en el *backlog* futuro vendría definida como se muestra en la Ilustración 23.

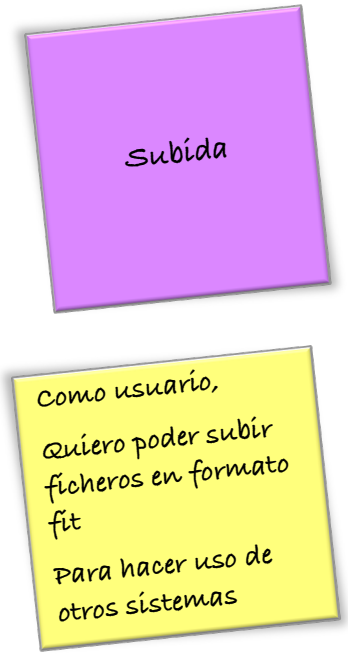


Ilustración 23

Por otro lado, también se pueden realizar mejoras no funcionales, es decir, aquellas que no dan valor añadido al sistema pero que sí mejoran características transversales como puede ser la escalabilidad o el rendimiento.

Por ejemplo, si el sistema creciese de manera significativa, es decir, se realizasen multitud de peticiones por segundo a la API del *backend*, se podría enfocar el sistema a un entorno más asíncrono donde las peticiones que se realizan cada minuto no fuesen bloqueantes ayudándose de la programación concurrente. En los últimos años se ha desarrollado una librería llama Project Reactor[133] que permite trabajar con un modelo de programación concurrente de forma muy sencilla. Es una implementación del estándar de *Streams* reactivos[134] y permite la creación de aplicaciones no bloqueantes en la máquina virtual de Java. El único inconveniente es que suele tener una mayor complejidad a la hora de realizar el *debugging* de la aplicación, sin embargo, trae consigo diferentes utilidades para lidiar con esta dificultad.

Para comprobar que efectivamente se ha mejorado el rendimiento con el uso de esta librería, es posible comprobarlo gracias al uso de Gatling[135] una herramienta que permite realizar test de carga sobre la aplicación donde se han realizado las mejoras, en este caso el *backend*.

[133] https://projectreactor.io
[134] https://www.reactive-streams.org
[135] https://gatling.io

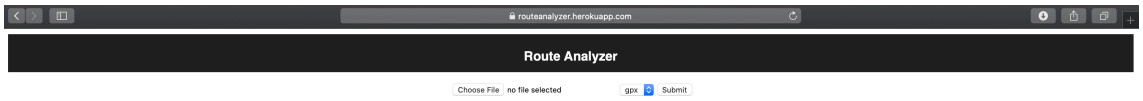
Manual de Usuario

Pantalla Inicial

En la Captura de Pantalla 1 se muestra el menú inicial de la aplicación. Tras haber accedido al sitio web donde está desplegado el sistema, descrito todo este proceso en el apartado *Despliegue*. La interfaz web es bastante sencilla, así se consigue que en estas primeras versiones el usuario pueda acceder directamente a la funcionalidad del sistema. Más adelante, en función del *feedback* obtenido por parte de los *stakeholders* y de los usuarios, se podría ir adaptando o mejorando a las necesidades solicitadas.

Así pues, hay que dejar claro que esta página web es una *Single Page Application*, concepto definido más arriba en el apartado *Tecnología*, y que quiere decir que la pantalla que se ve en la captura de más abajo también es la pantalla principal del sistema.

Esta primera pantalla de la aplicación contiene solamente un formulario donde se puede realizar la subida de una actividad física. Para ello, se ha de escoger uno de los ficheros compatibles con el sistema, descritos en el punto *Ficheros de actividades deportivas* del apartado *Aspectos Teóricos*, y seleccionarlo a través del elemento que aparece como un botón con el texto "*Choose File*". Una vez hecho esto y tras haber seleccionado el tipo de fichero xml que contiene la actividad deportiva que se desea subir desde el seleccionable, tan solo habrá que hacer clic sobre el botón identificado como "*Submit*". Tras esto, aparecerá un elemento que indicará que el elemento se está procesando. Una vez que haya finalizado, se visualizará toda la ruta como se muestra en el siguiente punto.



Captura de Pantalla 1

Pantalla Visualización Ruta

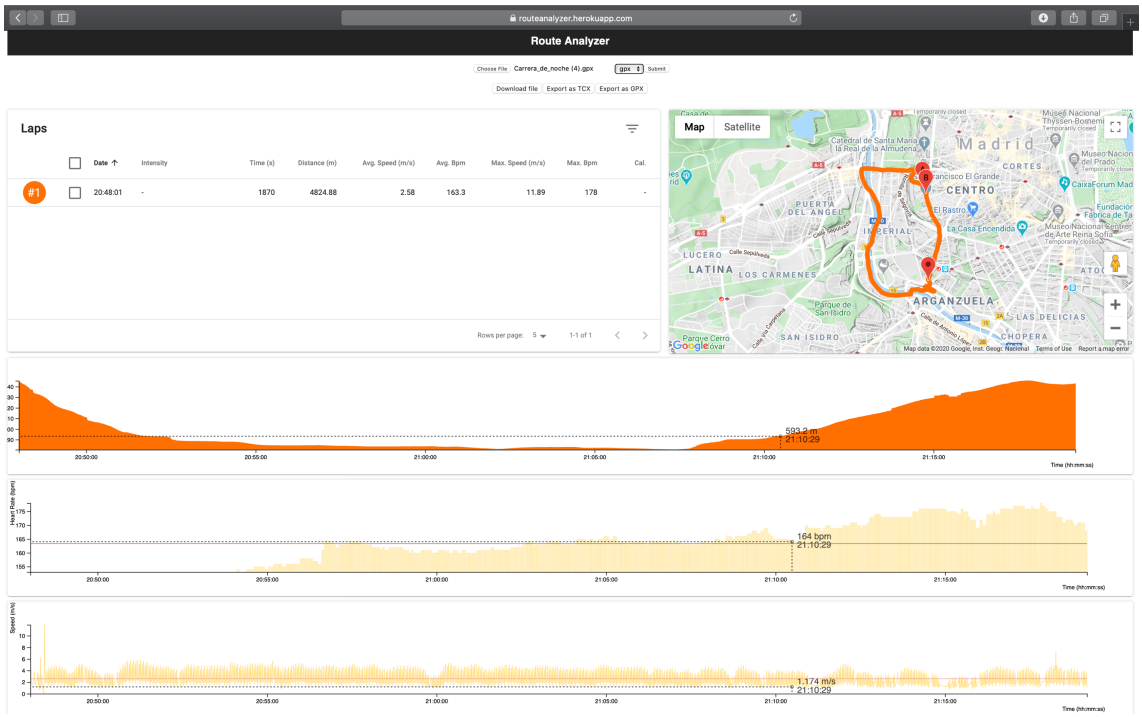
En la Captura de Pantalla 2 se muestra la situación de la aplicación web una vez que la actividad ha sido subida al sistema y no se ha producido ningún tipo de error.

Se pueden identificar tres zonas de la pantalla de visualización de la ruta: la pantalla donde se muestra la ruta deportiva en el mapa que se encuentra en la parte superior derecha, la pantalla donde se muestran los valores de cada vuelta de la ruta deportiva situada en la parte superior izquierda y la parte que contiene los gráficos que se encuentra ubicada en la parte inferior de los dos elementos descritos antes.

Los gráficos que se visualizan son tres y corresponden a los valores de la altura de ese punto geoespacial por donde ha pasado la trayectoria del usuario, la frecuencia cardiaca que se ha tenido en cada momento y la velocidad con la que el usuario ha realizado la actividad. Es importante destacar que cualquier punto seleccionado en el mapa también se visualizará en cada una de las gráficas, también ocurrirá lo mismo para cualquier punto seleccionado en alguna de los gráficos.

Por otro lado, al no estar segmentada la actividad deportiva, en la tabla solo se visualiza el segmento entero de la actividad deportiva con los valores que resumen cómo ha ido toda la actividad física. Estos valores vendrán muy ligados al tipo de formato que se haya

subido, en este caso, se muestra el resultado de haber subido una actividad física con formato GPX.



Captura de Pantalla 2

Elementos Edición de Rutas

Una vez que se tiene la actividad física en la pantalla de visualización se pueden realizar modificaciones sobre la propia ruta.

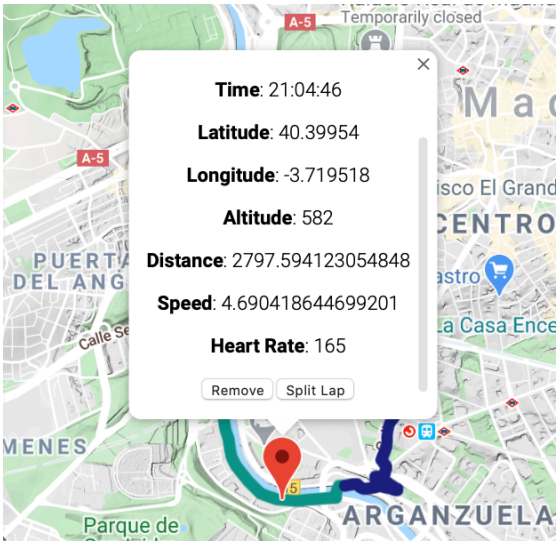
En la Captura de Pantalla 3 se muestra como se puede realizar la unión de dos segmentos o tramos a través de la tabla situada en la parte izquierda de la Captura de Pantalla 2. Para ello es necesario seleccionar los dos tramos (ni más ni menos) para que aparezca en la parte superior derecha el botón que permite unirlos. Una vez que se haya lanzado la acción, la ruta se volverá a cargar en el mapa siempre que todo haya ido bien. De no ser así, se mostrará un mensaje de que algo ha ido mal.

Captura de Pantalla 3

Otra de las funcionalidades que ofrece la aplicación a la hora de editar una ruta es la posibilidad de realizar el borrado de alguno de los segmentos a través de la misma tabla desde donde se visualizan. En la Captura de Pantalla 4 se muestra como se puede borrar un tramo siempre y cuando se seleccione solo un elemento. Tras haberlo seleccionado, aparecerá un icono que permite el borrado de todo ese tramo de la ruta deportiva. Como ocurría en la unión de segmentos, el párrafo anterior, si ocurriese algún error se visualizaría un mensaje y la actividad física no se cargaría con los nuevos valores ya que internamente permanecería como se visualiza.

Captura de Pantalla 4

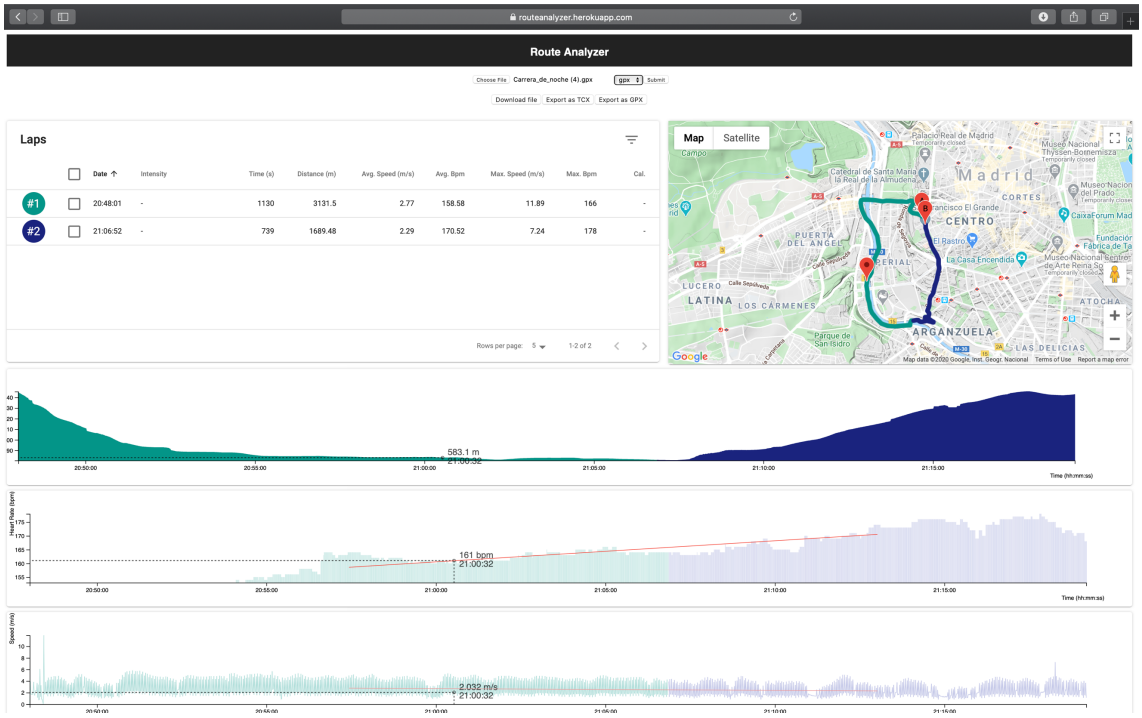
Otra de las posibilidades que brinda esta aplicación es poder editar la ruta deportiva a través del mapa. Seleccionando cualquier punto del mapa la aplicación web le permite al usuario poder borrar ese punto o separar la ruta en dos tramos, gracias a los botones que están identificados en la Captura de Pantalla 5 como "*Remove*" y "*Split Lap*", respectivamente. Como ocurría antes, si ocurriese algún error se mostraría un mensaje explicando qué ha pasado y la actividad física ningún otro elemento se actualizaría.



Captura de Pantalla 5

Pantalla Resultado de Modificación de Ruta

El resultado de aplicar la división de la ruta en dos segmentos desde el mapa geográfico tendría como resultado una pantalla similar a la mostrada en la Captura de Pantalla 6.



Captura de Pantalla 6

Con este ejemplo se puede apreciar que con esta funcionalidad que proporciona la aplicación se puede dividir la ruta en diferentes tramos teniendo en cuenta el usuario diferentes criterios. Por ejemplo, en este caso en concreto se ha tenido en cuenta la altitud ya que, si se pone atención a la captura de pantalla, se puede ver que el segundo tramo tiene una pendiente mucho más inclinada que el primero, que es bastante más llano o cuesta abajo. De esta forma, se puede enfocar los datos a diferentes contextos o escenarios.

Exportar Actividad Física

Por último, ya solo quedaría definir dónde se puede realizar la exportación de la actividad física. Tanto en la Captura de Pantalla 2 como en la Captura de Pantalla 6 se puede visualizar en el centro de la parte superior de la pantalla de la aplicación, una vez que se ha subido alguna actividad física, tres botones que se encuentran justo debajo del formulario.

El primero botón permite descargar el fichero original del repositorio en línea proveído por el servicio S3 de Amazon Web Service.

Los otros dos botones permiten exportar la actividad con la que se ha trabajado para poder trabajar, por ejemplo, en un sistema de terceros con ella. La diferencia que hay entre cada uno de ellos es que el segundo botón (contando al que descarga el fichero original) permite exportar la actividad a un fichero *xml* con un esquema *tcx* mientras que el tercero a uno con un esquema *gpx*.

Una vez que se ha hecho clic en alguno de los tres botones, automáticamente se descargará el fichero y ya estará disponible en la máquina desde donde has accedido al sitio web.

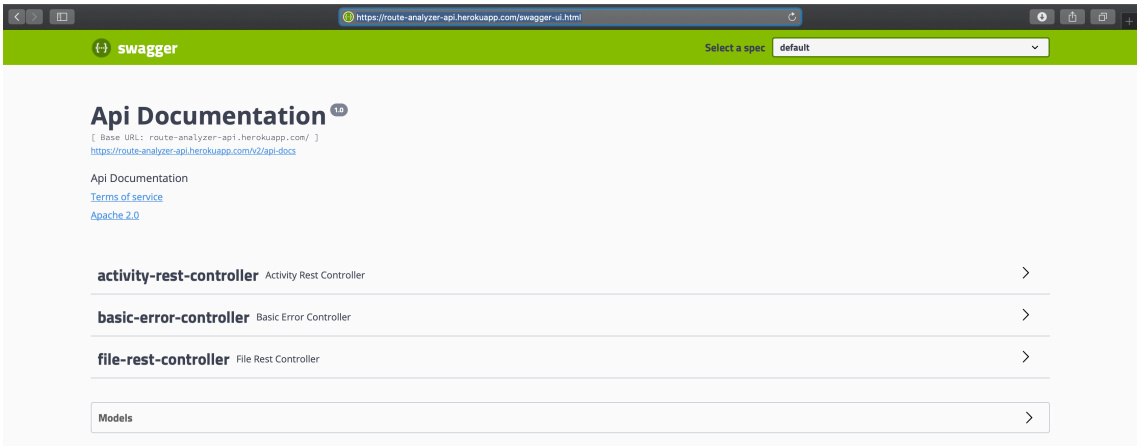
Acceso a documentación API

Como se comentó en el apartado identificado como *Tecnología*, la aplicación de *backend* usa la librería Swagger para generar la documentación de los *endpoints* de la aplicación del lado del servidor. Esta herramienta sirve de gran utilidad ya que permite que otros sistemas, creados o no, puedan hacer uso de esta librería y desarrollar otro tipo de aplicaciones, por ejemplo, para otro tipo de plataformas o con otra funcionalidad.

Por lo tanto, si se quiere consultar esta información, tan solo hay que acceder a:

https://route-analyzer-api.herokuapp.com/swagger-ui.html

El enlace llevará a una página web como el que se muestra en la Captura de Pantalla 7.



Captura de Pantalla 7

Anexo

Anexo I

Los ficheros se encuentran en un fichero comprimido denominado "Esquemas de Actividades Deportivas.zip". Una vez descomprimido, la carpeta resultante contendrá la siguiente información:

- El documento gpx1.xsd contiene el esquema de la versión 1.0 tratado en el punto *GPX* del apartado *Aspectos Teóricos*. Accesibles desde este enlace[136].
- El documento gpx.xsd contiene el esquema de la versión 1.1 tratado en el punto *GPX* del apartado *Aspectos Teóricos*. Accesible desde este enlace[137].
- El documento trackpointextension.xsd contiene el esquema de la versión 1 que extiende al esquema principal tratado en el punto *GPX* del apartado *Aspectos Teóricos*. Accesible desde este enlace[138].
- El documento tcx.xsd contiene el esquema de la versión 2.0 tratado en el punto *TCX* del apartado *Aspectos Teóricos*. Accesibles desde este enlace[139].
- El documento activityExtension.xsd contiene el esquema de la versión 2.0 tratado en el punto *TCX* del apartado *Aspectos Teóricos*. Accesibles desde este enlace[140].
- Fichero para realizar pruebas sobre la aplicación identificado como test.xml que contiene una actividad en formato *gpx*.

Anexo II

El proyecto que contiene la aplicación *frontend* se llama *route-analyzer-web* y está alojado en el repositorio https://github.com/luisrodrigar/route-analyzer-web. Se proporciona el código fuente identificado por un fichero comprimido llamado "*route-analyzer-web.zip*".

El proyecto que implementa todo el procesamiento del lado del servidor se llama *route-analyzer* y se encuentra almacenado en el repositorio *https://github.com/luisrodrigar/route-analyzer*. También se entregan los ficheros fuente en un fichero comprimido identificado como "*route-analyzer.zip*".

[136] https://www.topografix.com/GPX/1/0/gpx.xsd
[137] https://www.topografix.com/GPX/1/1/gpx.xsd
[138] https://www8.garmin.com/xmlschemas/TrackPointExtensionv1.xsd
[139] https://www8.garmin.com/xmlschemas/TrainingCenterDatabasev2.xsd
[140] https://www8.garmin.com/xmlschemas/ActivityExtensionv2.xsd

Referencias

http://www.ciberaula.com/articulo/ventajas_poo

https://openwebinars.net/blog/7-novedades-de-java-8-debes-saber-para-estar-al-dia/

https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias

http://es6-features.org

https://en.wikipedia.org/wiki/Adidas_Running_by_Runtastic

https://medium.com/grupo-carricay/cómo-escribir-un-buen-documento-de-especificación-de-requisitos-de-software-fd8bb3b5a39a

https://www.mountaingoatsoftware.com/agile/user-stories

https://josehuerta.es/gestion/agile/que-es-una-epica-y-que-no-es

https://urtanta.com/historias-de-usuario/

https://www.agilealliance.org/glossary/invest/

https://urtanta.com/user-story-mapping/

http://scrum.menzinsky.com/2018/11/como-se-hace-un-user-story-mapping.html

http://diegoacosta.net/blog/2017/12/10/scrum-con-jira-relacion-entre-epicas-historias-y-tareas-tecnicas/

https://medium.com/@ashera/qué-es-un-mbp-y-por-qué-deber%C3%ADamos-empezar-por-acá-b5324eeb0de7

https://es.slideshare.net/johnanthonyprza/planificacion-de-software-sistemas-ii

https://uv-mdap.com/programa-desarrollado/bloque-iv-metodologias-agiles/metodologias-agiles-vs-tradicionales/

https://www.youtube.com/watch?v=MeTn2LqnkKo

https://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html

https://proyectosagiles.org/graficos-trabajo-pendiente-burndown-charts/

https://www.youtube.com/watch?v=BeP6dXdLLo8&t=333s

https://www.baeldung.com/vavr-try

https://www.baeldung.com/vavr-pattern-matching

https://www.adictosaltrabajo.com/2015/03/02/optional-java-8/

http://www.codecommit.com/blog/scala/the-option-pattern

https://www.geeksforgeeks.org/stream-in-java/

https://medium.com/@Thoughtworks_es/qu%C3%A9-demonios-es-jsx-txt-f5841e51f664

https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

https://medium.com/@davidjguru/single-page-application-un-viaje-a-las-spa-a-trav%C3%A9s-de-angular-y-javascript-337a2d18532

https://medium.com/@Sugeesh/deploy-your-java-application-with-ec2-bd70de34c4a0

https://www.androidpolice.com/2019/09/25/runtastic-becomes-adidas-running/

https://blog.marcnuri.com/mockmvc-introduccion-a-spring-mvc-testing/

https://www.adictosaltrabajo.com/2015/07/27/tests-de-integracion-con-spring-boot-spring-security-en-servicios-rest/

https://medium.com/@krishankantsinghal/how-to-read-test-coverage-report-generated-using-jest-c2d1cb70da8b