



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

**GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN**

**ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE
COMPUTADORES**

**APLICACIÓN ANDROID PARA LA GESTIÓN DE UNA
PELUQUERÍA**

**DÑA. HIDALGO LÓPEZ, INES
TUTOR: D. ARIAS GARCIA, JOSE RAMON**

FECHA: Septiembre 2020

ÍNDICE

1. Introducción.....	5
2. Objetivo y alcance	6
3. Planificación temporal	7
4. Aplicación Android.....	9
4.1.- INTRODUCCIÓN ANDROID	9
4.1.1.- Versiones y dispositivos Android.....	9
4.2.- ANDROID STUDIO	11
4.2.1.- Lenguajes soportados	12
4.2.2.- Requisitos mínimos Android Studio	13
4.3.- CONCEPTOS BÁSICOS DE DESARROLLO.....	13
4.4.- CICLO DE VIDA DE UNA APLICACIÓN.....	14
5. BBDD.....	17
5.1.- INTRODUCCIÓN A FIREBASE.....	17
5.2.- DISEÑO DE BASE DE DATOS	19
6. Análisis de requisitos del sistema.....	23
6.1.- REQUISITOS FUNCIONALES.....	23
6.2 .- REQUISITOS NO FUNCIONALES	24
7. Casos de uso	25
7.1.- DESCRIPCIÓN DE LOS ACTORES DEL SISTEMA	25
7.2.- MODELO DE CASOS DE USO	26
7.3.- DESCRIPCIÓN DE LOS CASOS DE USO	28
8. Arquitectura.....	37
8.1.- PATRONES DE DISEÑO.....	38
8.2.- COMPONENTES DE LA ARQUITECTURA	41
9. Diseño	44
9.1.- DIAGRAMA DE NAVEGACIÓN	44
9.2.- DISEÑO DE LA INTERFAZ.....	45
9.3.- DISEÑO DE FUNCIONALIDADES	49
9.3.1- Registro e inicio de sesión.....	49
9.3.2- Gestión de empleados y tratamientos	51
9.3.3.- Gestión de reservas	52

9.3.4- Gestión de cola	54
10. Desarrollo	59
10.1.- COMPATIBILIDAD DE LA APLICACIÓN	59
10.2.- ESTRUCTURA DE LA APLICACIÓN	60
11. Pruebas.....	64
11.1.- PRUEBAS.....	64
10.2.- DISPOSITIVOS PARA LAS PRUEBAS	66
12. Presupuesto	67
12.1.- COSTE DE FIREBASE	67
12.2.- COSTE DEL PROYECTO	70
13. Trabajos futuros	72
14. Conclusiones	73
15. Bibliografía	74

Figuras

Figura 4.1.- Ciclo de vida de una actividad.....	14
Figura 4.2.- Estimación ciclo.....	16
Figura 5.1.- Creación Firebase	17
Figura 5.2.- Ejemplo Firebase	19
Figura 7.1. – Modelo casos de uso administrador	26
Figura 7.2.- Modelo casos de uso cliente	27
Figura 8.1.- Arquitectura recomendada por Android.....	37
Figura 8.2.- Modelo patrón MVC.....	39
Figura 8.3.- Modelo patrón MVP	39
Figura 8.4.- Ciclo de vida LiveData.....	41
Figura 8.5.- Dependencias del proyecto.....	43
Figura 8.6.- Dependencias de la aplicación.....	43
Figura 9.1.- Diagrama de navegación	45
Figura 9.2.Pantallas inicio y principal	46
Figura 9.3.- Menú aplicación.....	47
Figura 9.4.- Pantalla clientes	48
Figura 9.5.- Pantalla reserva.....	48
Figura 9.6.- Pantalla diálogo	49
Figura 9.7.- Correo verificación	50
Figura 9.8.- Pantalla olvido de contraseña	51
Figura 9.9.- Pantalla reserva.....	53

Figura 9.10.- Asignación.....	53
Figura 9.11.- Ejemplo gestión horas.....	54
Figura 9.12.- Pantalla gestión de cola.....	55
Figura 9.13.- Estados cliente.....	56
Figura 9.14.- Tiempos de espera.....	57
Figura 9.15.- Ejemplo cálculo tiempo de espera.....	58
Figura 10.1.- Estructura de la aplicación.....	60
Figura 10.2.- Estructura aplicación.....	61
Figura 10.3.- Estructura aplicación. Repositorios.....	61
Figura 10.4.- Estructura de la aplicación. Entidades.....	62
Figura 10.5.- Estructura del proyecto. Recursos.....	63
Figura 11.1.- Relación de pruebas.....	64
Figura 12.1.- Valores cálculo coste Firebase.....	69
Figura 12.2.- Coste Firebase al mes.....	70

Tablas

Tabla 3.1.- Diagrama de Gantt.....	8
Tabla 5.1.- Tabla usuarios.....	19
Tabla 5.2.- Tabla citas.....	20
Tabla 5.3.- Tabla tratamientos.....	20
Tabla 5.4.- Tabla empleados.....	21
Tabla 5.5.- Tabla parametrización.....	21
Tabla 5.6.- Tabla cola de espera.....	22
Tabla 6.1.- Requisitos funcionales.....	24
Tabla 6.2.- Requisitos no funcionales.....	24
Tabla 7.1.- Caso de uso: Iniciar sesión.....	28
Tabla 7.2.- Caso de uso: Cerrar sesión.....	28
Tabla 7.3.- Caso de uso: Alta usuario.....	29
Tabla 7.4.- Caso de uso: Registro.....	29
Tabla 7.5.- Caso de uso: Baja usuario (admin).....	30
Tabla 7.6.- Caso de uso: Baja de usuario (cliente).....	30
Tabla 7.7.- Caso de uso: Información.....	30
Tabla 7.8.- Caso de uso: Agenda administrador.....	31
Tabla 7.9.- Caso de uso: Agenda cliente.....	31
Tabla 7.10.- Caso de uso: Reservar cita.....	32
Tabla 7.11.- Caso de uso: Cancelar cita.....	32
Tabla 7.12.- Caso de uso: Modificación usuario.....	33
Tabla 7.13.- Modificar estados.....	33
Tabla 7.14.- Caso de uso: Añadir empleados.....	34
Tabla 7.15.- Caso de uso: Borrar empleados.....	34
Tabla 7.16.- Caso de uso: Modificar empleados.....	35
Tabla 7.17.- Caso de uso: Añadir tratamientos.....	35
Tabla 7.18.- Caso de uso: Modificar tratamientos.....	36

Tabla 9.1.- Tabla gestión de estados	55
Tabla 10.1.- Distribución de versiones por uso de usuarios.....	59
Tabla 11.1.- Plan de pruebas inicio de sesión	66
Tabla 12.1.- Almacenamiento por tipo de dato	68
Tabla 12.2.- Almacenamiento total aproximado de firebase.....	68
Tabla 12.3.- Ancho de banda por operaciones	69
Tabla 12.4.- Costes de equipo	71
Tabla 12.5.- Coste total proyecto.....	71

1. Introducción

En el presente documento se detalla el análisis y desarrollo de la aplicación móvil para dispositivos con sistema operativo Android, “La Marieta”. Se trata de una aplicación destinada a pequeñas peluquerías con el fin de ayudarlas automatizando la gestión de la misma, lo que será de gran ayuda para el empresario y para los clientes que dispondrán de una mayor información acerca de la peluquería y sus citas.

Este proyecto se entrega como Trabajo Fin de Grado de Ingeniería Informática en Tecnologías de la Información.

2. Objetivo y alcance

Desarrollar una aplicación para el sistema Android para la gestión de un pequeño negocio, en particular, una peluquería. Este sistema contemplará los perfiles de administrador y clientes, siendo el primero el encargado/empleador de la peluquería.

El perfil de administrador podrá parametrizar en el servidor los datos de sus servicios, como el número de empleados disponibles, tiempo de los diferentes servicios, reservas etc., con el fin de gestionar las citas y el tiempo de espera entre clientes.

El perfil del cliente podrá gestionar y consultar sus citas a través de la aplicación y obtener información de la peluquería como los tratamientos que se ofrecen o información de contacto.

Se desarrollará la gestión de una cola FIFO para la lista de espera de los clientes que estén en la peluquería esperando a ser atendidos, con cita en el mismo día, de manera que los clientes puedan visualizar el tiempo de espera aproximado que les resta para acceder a sus tratamientos. El administrador observará toda la lista ordenada mientras que el cliente tendrá limitada la información de forma que solo podrá observar el registro correspondiente a sí mismo.

3. Planificación temporal

A nivel del desarrollo de la aplicación, se ha considerado reunir un conjunto de actividades para elaborar una planificación de la misma mediante un diagrama de Gantt. Esta planificación ha ido cambiando con el tiempo a medida que se han encontrado errores durante el desarrollo, lo que impedía cumplir con los plazos establecidos en un primer momento.

Tareas planificadas:

- **Análisis**
Consiste en el análisis de los requisitos y funcionalidades de la aplicación,
- **Desarrollo Act.1**
Estructura de la aplicación
- **Desarrollo Act. 2**
Estudio y construcción de la base de datos de Firebase
- **Desarrollo Act. 3**
Construcción de la capa modelo de la aplicación
- **Desarrollo Act. 4**
Construcción de la capa modelo-vista
- **Desarrollo Act. 5**
Construcción de la capa vista
- **Pruebas**
Durante todo el desarrollo del proyecto se han llevado a cabo pruebas unitarias e integradas de la aplicación.
- **Mejoras**
Desarrollo de mejoras, en gran parte, localizadas durante las pruebas.
- **Calidad y Aseguramiento del código**
Limpieza del código de la aplicación de forma que se garantice una mayor eficiencia y legibilidad de esta.

N° Actividad	Inicio	Final	01-mar	10-mar	19-mar	28-mar	06-abr	15-abr	24-abr	03-may	12-may	21-may	30-may	08-jun	17-jun	26-jun	05-jul	14-jul	23-jul	01-ago	10-ago	19-ago	28-ago	06-sep	
Análisis	01/03/2020	31/03/2020	■	■	■	■																			
Desarrollo Act.1	01/04/2020	30/04/2020					■	■	■																
Desarrollo Act.2	01/05/2020	15/05/2020								■	■														
Desarrollo Act.3	10/05/2020	15/06/2020									■	■	■	■											
Desarrollo Act.4	15/05/2020	30/06/2020										■	■	■	■	■									
Desarrollo Act. 5	15/06/2020	30/07/2020													■	■	■	■	■						
Pruebas	15/04/2020	07/09/2020						■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Mejoras	10/08/2020	07/09/2020																			■	■	■	■	■
Calidad y Aseguramiento	01/09/2020	07/09/2020																							■

Tabla 3.1.- Diagrama de Gantt

4. Aplicación Android

En este capítulo se describe una pequeña introducción al sistema operativo Android desde su nacimiento y versiones hasta el desarrollo de aplicaciones para el sistema con Android Studio y los conceptos básicos de este.

4.1.- INTRODUCCIÓN ANDROID

Android es un sistema operativo multiplataforma utilizado en muchos de nuestros dispositivos, principalmente en los móviles además de las tablets, Google TV, etc. Actualmente cerca del 90% de los usuarios de móviles tienen un sistema operativo Android.

Una de las razones por las que Android es el preferido para los usuarios, es que se trata de un sistema de código abierto por lo que todos los desarrolladores pueden ver el código del sistema operativo y de la mayoría de las aplicaciones publicadas, lo que permite a los desarrolladores crear aplicaciones de forma más sencilla.

Actualmente Android pertenece a Google, puesto que la empresa Android Inc., creadora del sistema fue absorbida. Fue Google quien publicó la mayor parte del código fuente del sistema operativo mediante la fundación Apache la cual se encarga de dar soporte a proyectos software de código abierto.

4.1.1.- Versiones y dispositivos Android

La mayoría de los fabricantes de dispositivos móviles actuales, utilizan el sistema operativo de Android, como LG, Samsung, Huawei ... fuera se quedan fabricantes con su propio sistema operativo como Windows o Apple.

La primera versión de Android estaba orientada para móviles en negocios, en aquella versión ya se soportaban los widgets, Android Market o lo que hoy llamamos Google Play, y estaba orientado a teléfonos que tenían teclado.

Como curiosidad, desde 2009 todas las versiones de Android han recibido nombres clave en orden alfabético, muchos de ellos de dulces, como Cupcake, Donut, Éclair, Froyo...

Cupcake fue la primera versión que se sacó al público y empezó a orientarse a las pantallas táctiles.

A continuación, se muestra una tabla con las actualizaciones del sistema operativo a lo largo de los años.

Nombre código ↕	Número de versión ↕	Fecha de lanzamiento ↕	Nivel de API ↕
Apple Pie ⁵⁵	1.0	23 de septiembre de 2008	1
Banana Bread ⁵⁵	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 – 2.1	26 de octubre de 2009	5 – 7
Froyo	2.2 – 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 – 2.3.7	6 de diciembre de 2010	9 – 10
Honeycomb ⁵⁶	3.0 – 3.2.6	22 de febrero de 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.5	18 de octubre de 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9 de julio de 2012	16 – 18
KitKat	4.4 – 4.4.4	31 de octubre de 2013	19 – 20
Lollipop	5.0 – 5.1.1	12 de noviembre de 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5 de octubre de 2015	23
Nougat	7.0 – 7.1.2	15 de junio de 2016	24 – 25
Oreo	8.0 – 8.1	21 de agosto de 2017	26 – 27
Pie	9.0	6 de agosto de 2018	28
Android 10 ⁵⁷	10.0	3 de septiembre de 2019	29

	Última versión
	Versión antigua pero vigente
	Versión descontinuada

Figura 4.1.- Versiones SO Android

4.2.- ANDROID STUDIO

Android Studio es una plataforma de desarrollo integrado creada por Google. Está basado en el software IntelliJ IDEA de JetBrains, uno de los primeros IDE de Java disponibles con navegación avanzada de código y es publicado de forma gratuita a través de la licencia de Apache 2.0.

Android Studio posee numerosas funcionalidades que proporcionan una mayor productividad al desarrollador en comparación con otros entornos de desarrollo como, por ejemplo, Eclipse.

- Herramienta de compilación automatizada, Gradle. Que permite tiempos de compilación más bajos puesto que recoge las salidas de las ejecuciones anteriores para procesar solamente los cambios.
- Un emulador rápido.
- Un entorno unificado de desarrollo para todos los dispositivos.
- Aplicación de cambios para insertar cambios de código y recursos a la aplicación en ejecución sin reiniciar la aplicación
- Herramientas de lint para identificar problemas de rendimiento, errores de código, usabilidad y compatibilidad de la versión, etc. que deberemos ejecutar desde Android Studio para ir liberando de posibles errores la aplicación.
- Compatibilidad integrada para Google Cloud Platform, lo que facilita en nuestro caso la integración con Firebase.

Para el desarrollo de la aplicación se ha utilizado Android Studio 3.6.2, ya que es la última versión estable creada por Google, publicada el 1 de agosto de 2019 y cuya última actualización ha sido el 18 de marzo de 2020. Es importante desarrollar la aplicación con la última versión para poder beneficiarnos de las últimas novedades que se incluyen y para no preocuparnos de que el código de nuestra aplicación quede obsoleto en un corto espacio de tiempo.

4.2.1.- Lenguajes soportados

Java y Kotlin son los dos lenguajes más conocidos para programar en Android, pero realmente podemos programar Android en cualquier lenguaje que pueda compilarse y ejecutarse en la Java Virtual Machine (JVM).

Java es el lenguaje oficial y también el más antiguo lo que nos ofrece sus ventajas y desventajas a la hora de programar. Por la otra parte Kotlin es un lenguaje más nuevo creado por JetBrains, nacido en 2011 y cuyo código fuente fue liberado en 2012.

Aplicaciones como Pinterest o Uber están desarrolladas en Kotlin.

Ambos lenguajes pueden coexistir en un mismo proyecto, por lo que podríamos aplicar las ventajas de cada uno a la vez.

Ventajas de Kotlin

Aprendizaje sencillo. La sintaxis de Kotlin será muy familiar si controlamos el lenguaje Java.

Combina programación funcional y procedural.

Configuración sencilla en Android Studio.

Código más compacto que en Java.

Inconvenientes de Kotlin

Tiempo de ejecución extra.

Comunidad más pequeña y menos ayuda disponible.

Este proyecto finalmente ha sido desarrollado en Java, principalmente por ser un lenguaje conocido para el programador y por la mayor ayuda de la comunidad. Además, por el rendimiento de la aplicación, que, aunque no sea muy costosa es una ventaja considerable de cara al usuario de la aplicación.

4.2.2.- Requisitos mínimos Android Studio

- SO Windows 7 o superior
- 2 Gb de RAM (8 Gb recomendado)
- 2 Gb de espacio libre mínimo en disco (8Gb recomendado)
- Resolución mínima de 1280x800px
- Java 8
- 64 bits y procesador Intel

4.3.- CONCEPTOS BÁSICOS DE DESARROLLO

- **Activity**

Es uno de los elementos más comunes de Android. Las actividades son las encargadas de interactuar con el usuario por medio de una interfaz. Habitualmente cada actividad tiene una interfaz cargada. Por cada actividad es necesaria una clase para implementarla que extienda de la clase base Activity. Cada clase mostrará una interfaz de usuario, compuesta por una vista. Para cambiar de vista, se utiliza la clase base **Intent**.

- **View**

Es un el objeto que hace referencia a la interfaz o vista habitualmente escrita en XML. Las actividades cargan las vistas (View) para mostrárselas al usuario.

- **Layout**

El layout es el elemento XML en el que se programa la interfaz y que definen los parámetros para las vistas.

- **Service**

Los servicios son componentes en ejecución en segundo plano para operaciones de larga duración o trabajos en procesos remotos que queramos mantener de manera continua, aunque el usuario cambie de actividad.

- **Content Provider**

Gestiona un conjunto de datos de la aplicación para compartir. Es una clase que implementa un conjunto estándar de métodos que permite a otras aplicaciones guardar y obtener la información que maneja el Content.

- **Intent**

Es un mensaje que nos permite llamar a los componentes Activity o Service, lo que nos permite utilizar otras aplicaciones del sistema operativo. Se le indica al Intent que es lo que debe de realizar y la información de esta acción.

4.4.- CICLO DE VIDA DE UNA APLICACIÓN

Para poder seguir el ciclo de vida de la actividad de cualquier aplicación, estas incorporan un conjunto de llamadas. El sistema devuelve cada una de estas devoluciones de llamada cuando cambia de estado. En la mayoría de las aplicaciones no es imprescindible implementar cada una de las llamadas de la actividad.

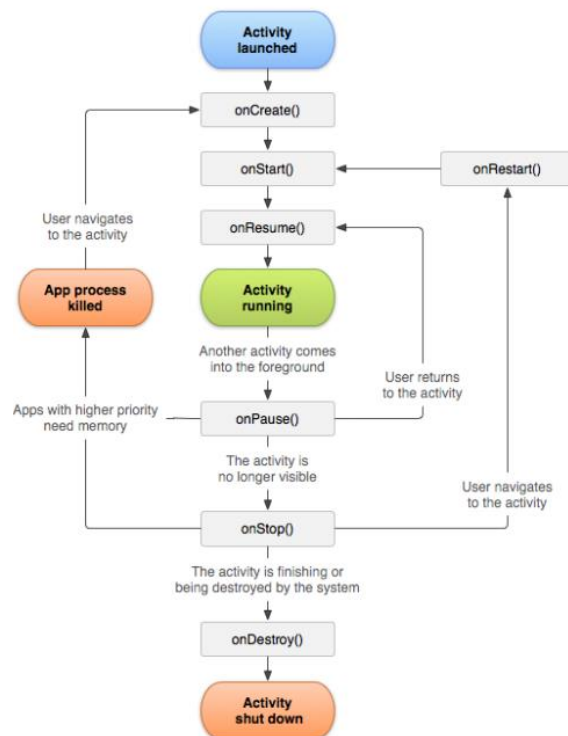


Figura 4.2.- Ciclo de vida de una actividad

onCreate() : Se llama cuando se crea por primera vez la actividad. En este método se implementa la lógica de arranque de la aplicación, que debe de ocurrir una única vez en toda la actividad. Una vez que se ha ejecutado la lógica de este método la actividad entra en el estado **onStart()** u **onResume()**.

onStart() : Esta llamada hace que el usuario pueda ver la actividad, una vez creada en la anterior llamada. Este método es donde la aplicación inicializa el código que mantiene la IU y la prepara para que el usuario pueda interactuar con ella. Una vez finalizado este método la actividad entra en el estado **onResume()**

onResume() : La aplicación entra en primer plano. Es el estado en el que la aplicación interactúa con el usuario. La aplicación permanece en este estado hasta que algún evento le quita el foco. Cuando se produce un evento de interrupción, la actividad entra en el estado **onPause()**.

onStop() : El sistema puede llamar a este método cuando la actividad va a finalizar o cuando un evento que cubre toda la pantalla interrumpe la actividad. En este método la aplicación libera los recursos que está utilizando mientras no sea visible para el usuario.

onPause() : Ocurre cuando un evento interrumpe la aplicación estando en el estado **onResume()**, por ejemplo, si el usuario recibe una llamada telefónica. El método indica principalmente que la actividad ya no se encuentra en primer plano o que su foco ha cambiado, como por ejemplo cuando se abre un dialogo. Cuando este evento termina, se vuelve al estado **onResume()**.

onDestroy() : Se invoca este evento antes de finalizar la actividad o antes de cambiar la configuración de la aplicación. En el primer caso, se llamara al método **isFinishing()**, en el otro caso, se volverán a inicializar los componentes con la nueva configuración y para ello volverá al evento **onCreate()**.

Probabilidad de que finalice	Estado del proceso	Estado de la actividad
Menor	Primer plano (en foco o por estar en él)	Created Started Resumed
Más	Segundo plano (foco perdido)	Paused
Mayor	Segundo plano (no visible)	Stopped
	Vacío	Destroyed

Figura 4.3.- Estimación ciclo

5. Base de datos

En este capítulo se realizará una breve introducción a Firebase y el diseño elegido en la base de datos para el desarrollo del proyecto.

5.1.- INTRODUCCIÓN A FIREBASE

Firebase es una base de datos en tiempo real localizada en la nube. Firebase permite interactuar con la BBDD cuando el dispositivo no tiene conexión mediante un sistema de cachés y colas de escritura locales. Cuando el dispositivo recupera la conexión, los datos se sincronizan automáticamente con la BBDD.

Para crear la BBDD desde Firebase se han seguido los siguientes pasos.

- Crear un nuevo proyecto en Firebase

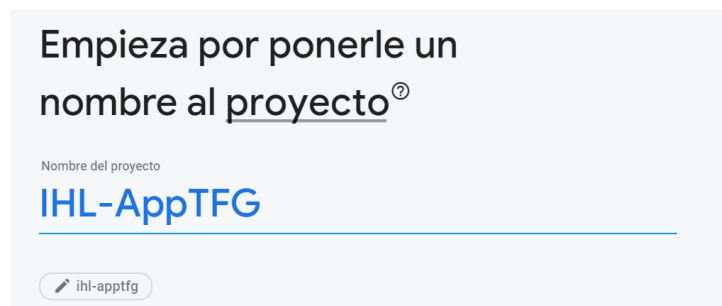


Figura 5.1.- Creación Firebase

Una vez creado el proyecto, podemos elegir entre dos tipos de base de datos Cloud Firestore o Realtime Database.

Para poder seleccionar entre estas decidimos agrupar los pros y los contras.

Realtime Database

- Datos jerárquicos y complejos difíciles de organizar
- Soporte sin conexión
- Las consultas pueden realizarse por ordenamiento o filtros, pero no encadenadas
- Las consultas no requieren un índice, pero se degrada su rendimiento cuanto mayores son los datos
- Escala a 200.000 conexiones simultaneas

Cloud Firestore

- Datos jerárquicos y complejos más fáciles de organizar con subcolecciones
- Soporte sin conexión
- Las consultas pueden encadenar filtros y ordenamiento
- Las consultas se indexan de forma predeterminada
- Mayor escalado de forma automática hasta 1 millón

Por lo puntos anteriormente citados, decidimos utilizar Cloud Firestore.

En la figura 5.2 se puede ver un ejemplo de cómo quedarían los datos en Firebase, una vez insertados. Es recomendable que tengamos los documentos vacíos e ir insertándolos desde la aplicación para que no haya problemas con los tipos de datos que se manejan, puesto que si se insertan directamente desde Firebase los datos serán de un tipo por defecto, que no tiene por qué ser el requerido para el desarrollo de nuestra aplicación.

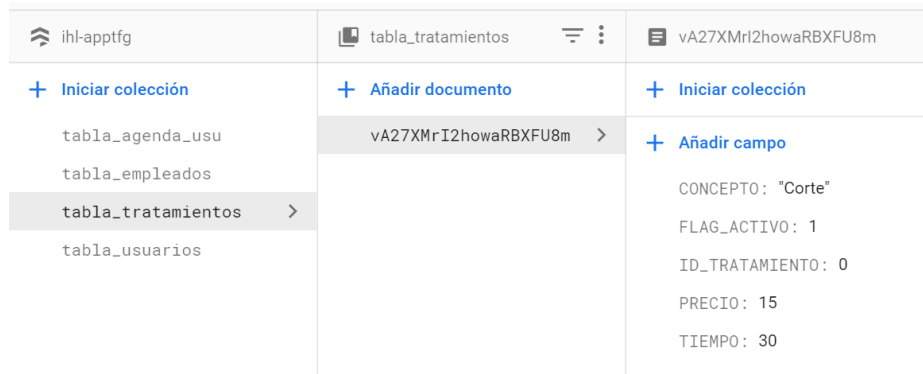


Figura 5.2.- Ejemplo Firebase

5.2.- DISEÑO DE BASE DE DATOS

La BBDD de Cloud Firestore estará formada por 6 tablas o colecciones y se describen a continuación.

TABLA_USUARIOS

En la colección TABLA_USUARIOS se almacenarán los datos personales de los usuarios de la aplicación, incluido el administrador de la misma ya que éste poseerá un perfil propio.

Nombre: tabla_usuarios		
Campo	Tipo	Observaciones
ID_USUARIO	String	ID Firebase
NOMBRE	String	Datos personales del usuario
APELLIDO1	String	Datos personales del usuario
APELLIDO2	String	Datos personales del usuario
NICKNAME	String	Alias/nickname del usuario
CORREO	String	Datos personales del usuario
DIRECCIÓN	String	Datos personales del usuario
TELÉFONO	Integer	Número de teléfono del usuario
ROL	String	Administrador o Cliente
FLAG_ACTIVADO	Integer	Indica si el usuario está activo. 1-Activo o 0-Desactivo

Tabla 5.1.- Tabla usuarios

TABLA_AGENDA_USU

En esta colección se almacenarán las citas de los usuarios.

Nombre: tabla_agenda_usu		
Campo	Tipo	Observaciones
ID_CITA	String	ID Firebase
CORREO	String	Correo del usuario de la cita
ID_TRATAMIENTO	Array	Array que almacena los IDs de los tratamientos de la cita
FECHA_CITA	Date	Fecha del inicio de la cita
FECHA_FIN	Date	Fecha fin de la cita, calculada por los tratamientos
FLAG_ACTIVADO	Integer	Indica si la cita está activa. 1-Activo o 0-Cancelada

Tabla 5.2.- Tabla citas

TABLA_TRATAMIENTOS

Almacena los datos de los tratamientos que el administrador/empleo ha registrado, tanto los que se encuentran activos y visibles al cliente como los que no.

Nombre: tabla_tratamientos		
Campo	Tipo	Observaciones
ID_TRATAMIENTO	String	ID Firebase
CONCEPTO	String	Concepto del tratamiento
TIEMPO	Double	Tiempo del tratamiento en minutos
PRECIO	Double	Coste del tratamiento en euros
FLAG_ACTIVADO	Integer	Indica si el tratamiento está activo. 1-Activo o 0-Desactivo

Tabla 5.3.- Tabla tratamientos

TABLA_EMPLEADOS

Registra los empleados y el horario de estos.

Nombre: tabla_empleados		
Campo	Tipo	Observaciones
ID_EMPLEADO	String	ID Firebase
HORA_INICIO	Date	Hora de inicio jornada laboral
HORA_FIN	Date	Hora de fin jornada laboral
NOMBRE	String	Nombre empleado
FLAG_ACTIVADO	Integer	Indica si el empleado está activo. 1-Activo o 0-Desactivo

Tabla 5.4.- Tabla empleados

TABLA_PARAMETRIZACION

Se almacenan diferentes registros parametrizables que no correspondan a una tabla/colección en concreto.

Nombre: tabla_parametrizacion		
Campo	Tipo	Observaciones
ID_PARAMETRIZACION	String	ID Firebase
CONCEPTO	String	Concepto identificativo de la parametrización
FLAG_ACTIVADO	Integer	Indica si el parámetro está activo. 1-Activo o 0-Desactivo
PARAMETRO_1	String	Parámetro

Tabla 5.5.- Tabla parametrización

TABLA_COLA_ESPERA

Se almacenan los clientes que se encuentran en la cola de espera.

Nombre: tabla_cola_espera		
Campo	Tipo	Observaciones
ID_COLA_FB	String	ID Firebase Cola
ID_CITA	String	ID Firebase de la cita que corresponde
CORREO	String	Correo del cliente
NOMBRE	String	Nombre del cliente
FECHA_ENTRADA	Date	Fecha completa en la que accedió a la espera
EMPLEADO	String	Empleado que atenderá la cita
ESTADO	String	Estado de la cita
FECHA_CITA	Date	Fecha original de la cita
FECHA_CURSO	Date	Fecha en la que el usuario está en curso
FECHA_FIN	Date	Fecha de finalización de la cita aproximada
PRESENCIA	Integer	Check de presencia en la sala de espera 1- Presente, 0 - Ausente
TIEMPO_ESPERA	Integer	Tiempo aproximado de espera para ser atendido

Tabla 5.6.- Tabla cola de espera

6. Análisis de requisitos del sistema

A continuación, se detallarán los requisitos necesarios para el desarrollo del sistema. Estos requisitos se dividirán en funcionales y no funcionales.

6.1.- REQUISITOS FUNCIONALES

Describen las iteraciones que tendrán los usuarios con el software.

RF1 – Gestión de usuarios	El usuario deberá identificarse como cliente o administrador
	Los clientes podrán registrarse en un formulario al inicio de la aplicación
	Si el usuario se olvida de su contraseña podrá recuperarla
	El sistema enviará un correo de confirmación al usuario al darse de alta en la aplicación
	El administrador tendrá una cuenta única
RF2 – Gestión de citas (cliente)	El cliente en la pantalla principal podrá ver su cita actual
	El cliente podrá elegir la hora y el tratamiento a realizar en su reserva
	Cuando el cliente entre en la peluquería, el administrador pondrá al cliente en la cola y el cliente visualizará el tiempo de espera aproximado y el número de personas que tenga delante
	El cliente puede consultar en cualquier momento el historial de sus citas
	El cliente podrá cancelar su cita hasta 24h antes
RF3 – Gestión de citas (administrador)	El administrador dispondrá de una agenda para ver las citas de los clientes
	El administrador podrá desde la aplicación reservar las citas de aquellos clientes que lo pidan presencial o telefónicamente
	El administrador podrá cancelar cualquier cita en cualquier momento
	El administrador podrá modificar el número de empleados en sala, con lo que se modificará el tiempo de espera de los clientes
RF4 – Gestión de cola	El administrador podrá cambiar el estado de las citas del día en la cola

	El cliente tendrá actualizado en tiempo real el tiempo de espera aproximado hasta que sea atendido
	El administrador tendrá actualizado en tiempo real el tiempo de espera aproximado de los clientes que están en la cola en el día actual
RF5 – Solicitud de alta	El cliente podrá darse de alta en la aplicación al acceder a esta
	El administrador podrá dar de alta a los clientes
RF6 – Solicitud de baja	El administrador puede gestionar las bajas de los clientes
	En la configuración de los clientes, pueden darse de baja automáticamente
	El administrador no podrá darse de baja
RF7 - Contacto	Los clientes podrán obtener el teléfono y horario de la peluquería en el apartado de contacto de la aplicación
RF8 – Configuración de la cuenta	El usuario podrá configurar los datos personales asociados a su cuenta
RF9 – Inicio de sesión	Al ingresar en la aplicación el usuario introducirá sus credenciales, si estas son válidas accederá al menú principal con el perfil al que pertenezca
RF10 – Cierre de sesión	En cualquier momento el usuario podrá cerrar sesión de la aplicación

Tabla 6.1.- Requisitos funcionales

6.2.- REQUISITOS NO FUNCIONALES

Especifican opciones del sistema como el rendimiento, seguridad, disponibilidad, etc.

RNF1 - Rendimiento	La aplicación podrá usarse en la mayoría de las versiones del S.O. Android actuales
RNF2 - Versiones	La versión de la aplicación podrá consultarse en cualquier momento en “ <i>Información</i> ”
RNF3 – Seguridad	El login del usuario se realizará mediante Firebase Authentication
	Los administradores nunca verán ni modificarán las contraseñas de los usuarios
RNF4 – Disponibilidad	El dispositivo debe tener conexión a Internet
	La aplicación estará disponible ininterrumpidamente, excepto por labores de mantenimiento

Tabla 6.2.- Requisitos no funcionales

7. Casos de uso

Mediante el modelo de casos de uso, explicaremos de forma gráfica la iteración entre los actores y el sistema.

En la aplicación habrá dos actores, ya que hay diferencias entre los dos tipos de usuarios del sistema.

7.1.- DESCRIPCIÓN DE LOS ACTORES DEL SISTEMA

Nombre:	Administrador
Descripción:	Representa al usuario encargado de la peluquería que podrá indicar a la aplicación todos los parámetros para la gestión de citas en la peluquería y su correcto funcionamiento, ver el historial de las citas, gestionar clientes, etc.

Nombre:	Cliente
Descripción:	Representa al cliente de la peluquería, el cual, deberá de estar registrado para poder hacer uso de la aplicación y disfrutar de las ventajas que ésta le ofrece.

7.2.- MODELO DE CASOS DE USO

Representación de los diagramas de casos de uso de la aplicación diferenciados por los actores de la misma.

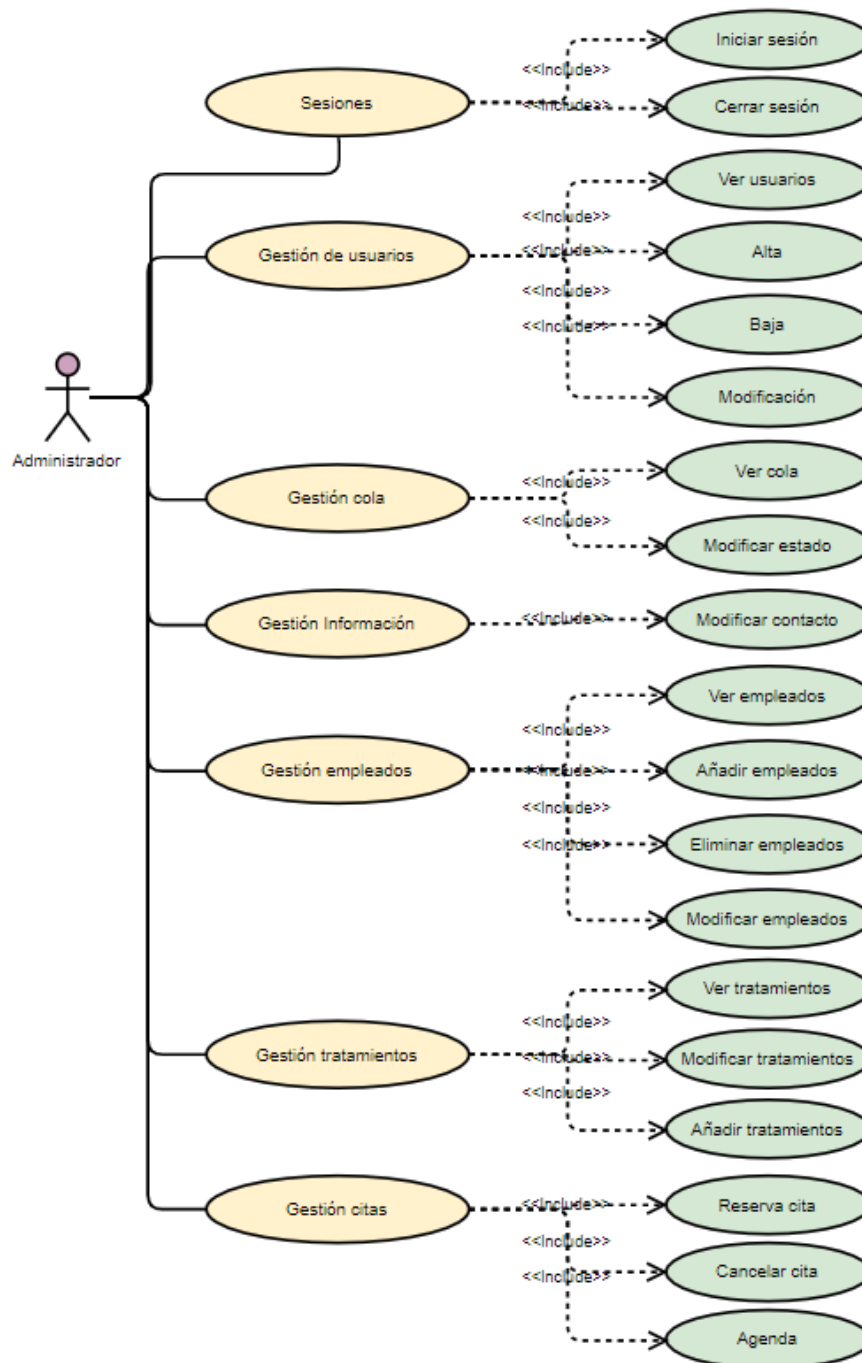


Figura 7.1. – Modelo casos de uso administrador

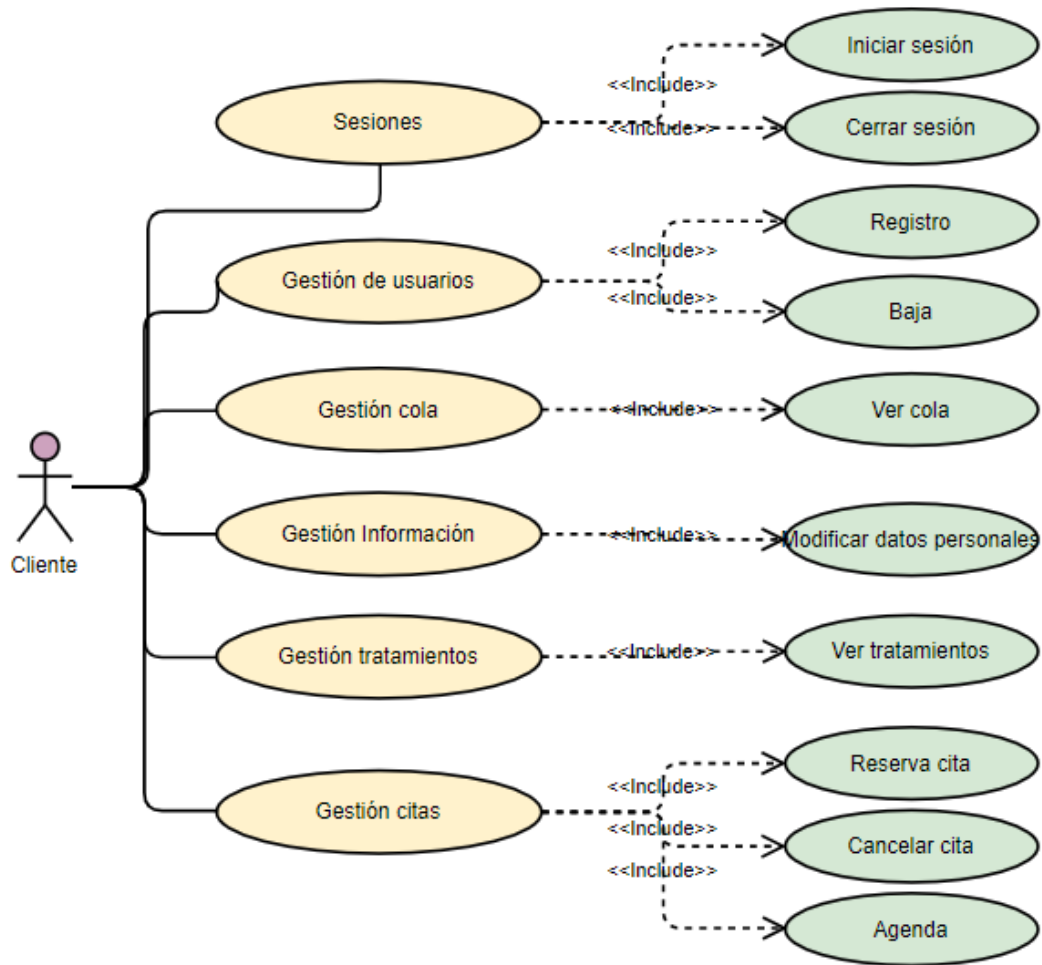


Figura 7.2.- Modelo casos de uso cliente

7.3.- DESCRIPCIÓN DE LOS CASOS DE USO

Nombre:	Iniciar sesión
Descripción:	Permite acceder al menú principal de la aplicación
Actores:	Administrador, Cliente
Precondiciones:	Estar dado de alta en la BBDD de la aplicación y en Authentication Firebase
Flujo normal:	<ol style="list-style-type: none"> 1. El actor inicia la aplicación desde su dispositivo 2. El actor introduce las credenciales de acceso 3. El actor accede al menú de la aplicación
Flujo alternativo:	2b. El sistema comprueba la validez de las credenciales contra la BBDD y Authentication, si las credenciales no son válidas, se notifica al actor que las credenciales son incorrectas
Postcondiciones:	

Tabla 7.1.- Caso de uso: Iniciar sesión

Nombre:	Cerrar sesión
Descripción:	Permite cerrar sesión de la aplicación
Actores:	Administrador, Cliente
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El actor selecciona Cerrar sesión desde la aplicación 2. La aplicación cierra la sesión y se visualiza la pantalla de login
Flujo alternativo:	
Postcondiciones:	

Tabla 7.2.- Caso de uso: Cerrar sesión

Nombre:	Alta de usuarios
Descripción:	Permite dar de alta a los clientes de la aplicación
Actores:	Administrador
Precondiciones:	
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador selecciona “+” dentro del panel de Clientes 2. La aplicación muestra un formulario a rellenar por el administrador con los datos del cliente 3. El administrador selecciona enviar alta 4. El cliente recibirá un email para verificar la cuenta y deberá cambiar la contraseña por defecto que crea el administrador para el cliente
Flujo alternativo:	3b. El sistema comprueba que los datos introducidos son válidos, si no lo son, están incompletos o el correo del usuario ya está dado de alta se informa para que vuelva a rellenarlos correctamente
Postcondiciones:	

Tabla 7.3.- Caso de uso: Alta usuario

Nombre:	Registro usuario
Descripción:	Permite dar de alta a los clientes de la aplicación
Actores:	Cliente
Precondiciones:	
Flujo normal:	<ol style="list-style-type: none"> 1. El cliente desde la pantalla de login selecciona “Registrarse” 2. La aplicación muestra un formulario a rellenar por el cliente con los datos requeridos 3. El cliente selecciona guardar 4. El cliente recibe un email de confirmación y se da de alta en el sistema
Flujo alternativo:	3b. El sistema comprueba que los datos introducidos son válidos, si no lo son, están incompletos o el usuario ya existe se informa para que vuelva a rellenarlos correctamente
Postcondiciones:	

Tabla 7.4.- Caso de uso: Registro

Nombre:	Baja de usuarios
Descripción:	Permite dar de baja a los clientes de la aplicación
Actores:	Administrador
Precondiciones:	Estar dado de alta en la BBDD de la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador selecciona <i>un cliente</i> 2. El administrador selección <i>Baja cliente</i> 3. El administrador confirma la baja
Flujo alternativo:	
Postcondiciones:	El cliente se da de baja en la BBDD

Tabla 7.5.- Caso de uso: Baja usuario (admin)

Nombre:	Baja de usuario
Descripción:	Permite a el propio cliente darse de baja de la aplicación
Actores:	Cliente
Precondiciones:	Estar dado de alta en la BBDD de la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El cliente selecciona en <i>Ajustes, Baja cliente</i> 2. El cliente confirma la baja 3. El cliente sale de la aplicación
Flujo alternativo:	
Postcondiciones:	El cliente se da de baja en la BBDD

Tabla 7.6.- Caso de uso: Baja de usuario (cliente)

Nombre:	Gestión de la información
Descripción:	Permite al administrador modificar la información que se muestra al cliente como el contacto
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador selecciona la información que desea modificar 2. El administrador modifica la información
Flujo alternativo:	
Postcondiciones:	La información se actualiza en la BBDD de la aplicación

Tabla 7.7.- Caso de uso: Información

Nombre:	Agenda
Descripción:	Permite al administrador visualizar las citas concertadas por los clientes
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador selecciona <i>Agenda</i> 2. El administrador selecciona <i>Ver citas</i> para ver las citas del día seleccionado o <i>Historial</i> 3. El administrador ve las citas
Flujo alternativo:	3.b. No hay citas para el día seleccionado
Postcondiciones:	

Tabla 7.8.- Caso de uso: Agenda administrador

Nombre:	Agenda cliente
Descripción:	Permite al cliente visualizar todas las citas registradas de el mismo
Actores:	Cliente
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El cliente selecciona <i>Agenda</i> 2. El cliente selecciona <i>Ver citas</i> 3. El cliente ve las citas registradas
Flujo alternativo:	3.b. No hay citas para el día seleccionado
Postcondiciones:	

Tabla 7.9.- Caso de uso: Agenda cliente

Nombre:	Reservar cita
Descripción:	Permite al administrador reservar cita para cualquier cliente y a los clientes reservar citas para ellos mismos.
Actores:	Administrador, Cliente
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El actor selecciona el día que desea realizar la cita 2. El actor selecciona el/los tratamientos/s a realizar 3. El actor selecciona el hueco libre 4. El actor selecciona <i>Guardar</i>
Flujo alternativo:	<ol style="list-style-type: none"> 4b. En el caso del administrador selecciona el cliente al que va a reservar la cita 4c. El sistema valida que los datos introducidos son correctos
Postcondiciones:	Cita reservada y almacenada en la BBDD

Tabla 7.10.- Caso de uso: Reservar cita

Nombre:	Cancelar cita
Descripción:	Permite al administrador cancelar la cita para cualquier cliente y a los clientes cancelar citas para ellos mismos.
Actores:	Administrador, Cliente
Precondiciones:	Haber iniciado sesión en la aplicación y tener una reserva para el cliente
Flujo normal:	<ol style="list-style-type: none"> 1. El actor accede a la aplicación y visualiza sus próximas citas 2. Selecciona en la cita interesada 3. El actor desliza hacia la izquierda y selecciona <i>Cancelar cita</i>
Flujo alternativo:	<ol style="list-style-type: none"> 3c. El sistema valida que los datos introducidos son correctos y que el tiempo no supera las 24 horas de antelación a la cita en el caso del actor cliente.
Postcondiciones:	Cita cancelada y almacenada en la BBDD

Tabla 7.11.- Caso de uso: Cancelar cita

Nombre:	Modificación usuarios
Descripción:	Permite al administrador modificar los datos personales de los usuarios, a excepción del correo y la contraseña
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y accede a la pestaña de <i>Cientes</i> 2. Desliza hacia la derecha en el cliente que desea modificar 3. Al ir hacia atrás la información se guarda en la base de datos
Flujo alternativo:	
Postcondiciones:	Cliente actualizado en BBDD

Tabla 7.12.- Caso de uso: Modificación usuario

Nombre:	Modificar estado
Descripción:	Permite al administrador modificar el estado de los clientes en cola que tengan citas para el día.
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación 2. Desliza hacia ambos lados para observar las opciones de los estados 3. Selecciona la acción a realizar
Flujo alternativo:	3b. El cliente ya se encuentra en el estado seleccionado y no se actualiza
Postcondiciones:	Cola actualizada en BBDD

Tabla 7.13.- Modificar estados

Nombre:	Añadir empleados
Descripción:	Permite al administrador añadir un empleado nuevo a la peluquería
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y selecciona la pestaña <i>Empleados</i> 2. Visualiza los empleados y selecciona el botón “+” 3. Se abre la ventana para introducir los datos del empleado y seleccionamos el check para guardar
Flujo alternativo:	3b. Si los datos introducidos no son correctos, salta un aviso en la aplicación
Postcondiciones:	Empleado insertado en BBDD

Tabla 7.14.- Caso de uso: Añadir empleados

Nombre:	Borrar empleados
Descripción:	Permite al administrador eliminar empleados
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y selecciona la pestaña <i>Empleados</i> 2. Visualiza los empleados y desliza hacia la izquierda y selecciona “<i>Eliminar</i>” 3. Se abre un diálogo de confirmación, aceptamos y la aplicación nos informa de la eliminación del empleado
Flujo alternativo:	
Postcondiciones:	Empleado eliminado de BBDD

Tabla 7.15.- Caso de uso: Borrar empleados

Nombre:	Modificar empleados
Descripción:	Permite al administrador modificar los empleados de la peluquería
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y selecciona la pestaña <i>Empleados</i> 2. Visualiza los empleados y desliza hacia la derecha y selecciona “<i>Editar</i>” 3. Se abre la ventana para introducir los datos del empleado, vamos hacia atrás y se guardan los datos
Flujo alternativo:	3b. Si los datos introducidos no son correctos, salta un aviso en la aplicación
Postcondiciones:	Empleado actualizado en BBDD

Tabla 7.16.- Caso de uso: Modificar empleados

Nombre:	Añadir tratamientos
Descripción:	Permite al administrador añadir tratamientos para la peluquería
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y selecciona la pestaña <i>Tratamientos</i> 2. Visualiza los tratamientos y selecciona el botón “+” 3. Se abre la ventana para introducir los datos del tratamiento, los introducimos y seleccionamos el check del botón flotante para guardar
Flujo alternativo:	3b. Si los datos introducidos no son correctos, salta un aviso en la aplicación
Postcondiciones:	Tratamiento insertado en BBDD

Tabla 7.17.- Caso de uso: Añadir tratamientos

Nombre:	Modificar tratamientos
Descripción:	Permite al administrador modificar los tratamientos de la peluquería
Actores:	Administrador
Precondiciones:	Haber iniciado sesión en la aplicación
Flujo normal:	<ol style="list-style-type: none"> 1. El administrador accede a la aplicación y selecciona la pestaña <i>Tratamientos</i> 2. Visualiza los empleados y desliza hacia la derecha y selecciona “<i>Editar</i>” 3. Se abre la ventana para introducir los datos del tratamiento, vamos hacia atrás y se guardan los datos
Flujo alternativo:	3b. Si los datos introducidos no son correctos, salta un aviso en la aplicación
Postcondiciones:	Tratamiento actualizado en BBDD

Tabla 7.18.- Caso de uso: Modificar tratamientos

8. Arquitectura

La arquitectura en una aplicación Android es de las partes más importantes para tener en cuenta. Es ésta la que nos proporciona una aplicación estructurada con un código sólido, estable y escalable. Otros tipos de aplicaciones se ejecutan con un solo proceso monolítico que contienen una estructura fija, en cambio las aplicaciones Android tienen una estructura más complicada. En la arquitectura se tienen en cuenta diferentes componentes como las librerías de código, recursos, vistas, código fuente, servicios etc., de la aplicación. Estos componentes pueden ser utilizados en diferente orden, debido al uso general por parte de los usuarios de las aplicaciones. Es por ello, que es tan importante la arquitectura en el desarrollo de la aplicación.

Una buena arquitectura nos permite cumplir el principio más importante, **separación de problemas**. Para ello cada componente debe tener implementado la lógica de éste y los componentes no deben ser interdependientes entre ellos.

Para cumplir con todas estas características, la aplicación se ha basado en el modelo de arquitectura que propone **JetPack** en el que cada componente solamente depende del componente que se encuentra a un nivel inferior. *Figura 8.1.- Arquitectura recomendada por Android*

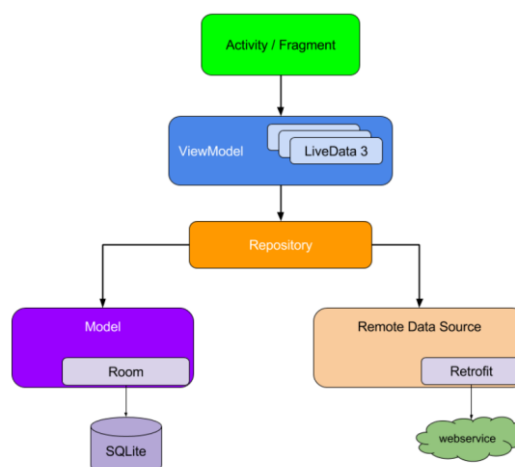


Figura 8.1.- Arquitectura recomendada por Android

La arquitectura recomendada en JetPack se ha adaptado al proyecto, puesto que en nuestro proyecto disponemos como proveedor de datos Firebase, el cual además incluye la persistencia de éstos en modo offline, el repositorio accederá a los datos de Firebase.

8.1.- PATRONES DE DISEÑO

Para apoyarnos en la construcción de esta arquitectura, ésta debe seguir un patrón que nos permita la separación de responsabilidades en nuestros componentes. Hay muchos patrones válidos para Android.

Los patrones de diseño nos ofrecen:

- Código más limpio y organizado
- Mayor claridad y mejor comprensión del proyecto frente a otros desarrolladores
- Reutilización de código
- Mantenimiento más rápido
- Independencia de la interfaz
- Independencia de la base de datos

Hasta hace poco tiempo las aplicaciones de Android utilizaban un patrón MVP (Modelo Vista Presentador), derivado del MVC (Modelo Vista Controlador). MVP solo modela la capa de presentación, hace que las vistas sean independientes de nuestra base de datos. Separa el modelo de la vista y estas interactúan por medio de un presentador.

Patrón MVC

Es el modelo-vista-controlador, muy conocido en diferentes tipos de aplicaciones. La vista contiene la interfaz de usuario y la lógica de negocio, el controlador, que está basado en los casos de uso responde a los eventos que le demanda la vista e invoca peticiones al modelo que administra esta información y la opera, devolviéndole al controlador un resultado y éste haciéndoselo llegar a la vista. De esta forma el modelo y la vista trabajan de forma separada gestionados por el controlador.

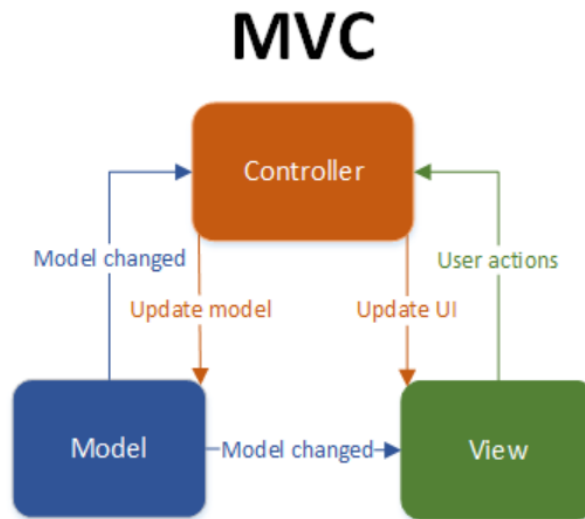


Figura 8.2.- Modelo patrón MVC

Patrón MVP

El modelo-vista-presentador es muy similar al MVC, con el añadido que no hay ningún contacto entre el modelo y la vista, la principal diferencia con el MVC es que en este caso puede haber un presentador por cada vista, siendo más sencillo utilizarlo para vistas muy complejas.

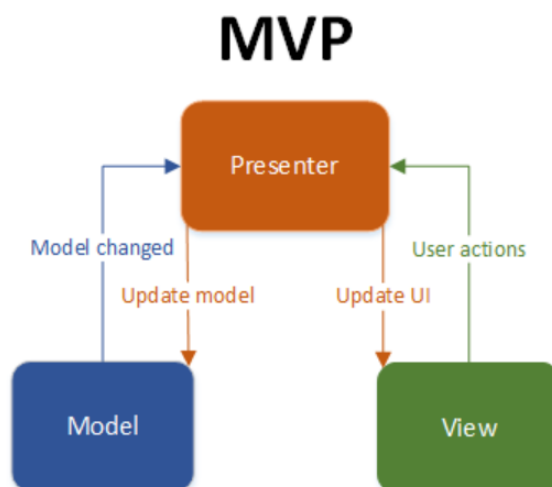


Figura 8.3.- Modelo patrón MVP

Patrón MVVM

El patrón Model-View-ViewModel nos ayuda a separar la lógica de negocios de la interfaz de usuario. Para ello utilizamos tres componentes.

El modelo, son las clases que representan el dominio de la aplicación. Al iniciar una aplicación es el primer componente tomado en cuenta ya que se definen los atributos que estaremos utilizando en nuestra aplicación. Las entidades creadas en el proyecto pertenecen al modelo.

El ViewModel contiene la lógica de presentación. Implementa propiedades y comandos que definen funcionalidades que tomará dicha aplicación.

Cuando sucede un cambio en el modelo, éste notifica al ViewModel que a su vez notifica al View y éste realiza los cambios necesarios para mostrar la información recibida al usuario.

Diferencias entre MVP y MVVM

- En MVP el presentador se comunica con la vista mediante una interfaz.
- En MVVM, el ViewModel se comunica con la vista mediante el patrón Observer.
- Gracias a ViewModel y LiveData no es necesario controlar el ciclo de vida de la aplicación.

MVVM es el patrón elegido para implementar la aplicación puesto que nos proporciona una mayor separación entre las capas dotando de mayor robustez a la aplicación.

Una gran ventaja de MVVM es el desacoplamiento de cada componente de la aplicación lo que nos proporciona un código más limpio, estructurado y nos facilita las pruebas.

La vista simplemente implementará lo que el usuario ve y tratará los datos recibidos del ViewModel.

El ViewModel realiza la transformación de los datos de la entrada para que el modelo los almacene. El ViewModel toma un repositorio en su constructor para poder comunicarse con el modelo.

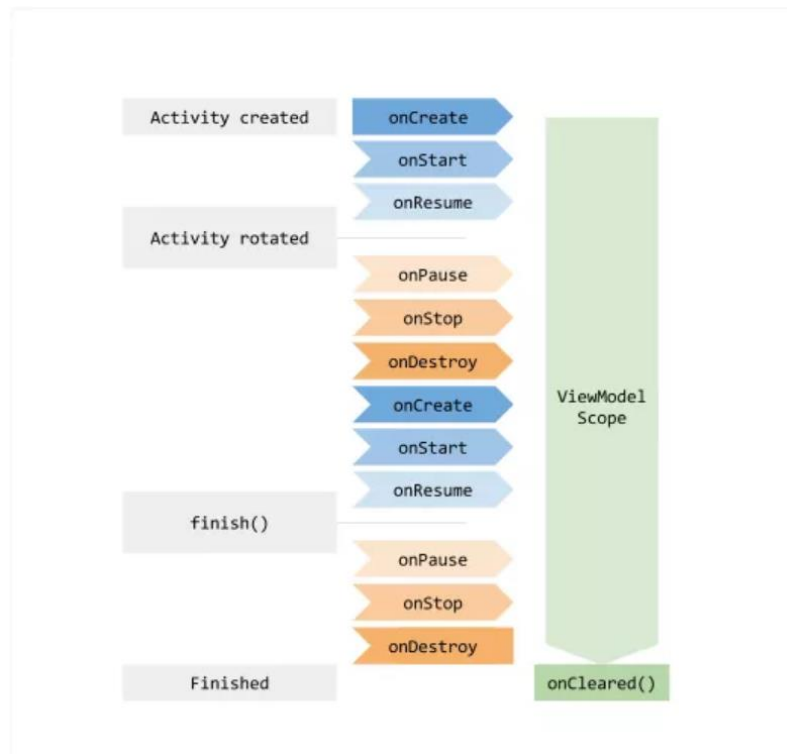


Figura 8.4.- Ciclo de vida LiveData

8.2.- COMPONENTES DE LA ARQUITECTURA

Utilizaremos algunos de los componentes recomendados por JetPack para el desarrollo de la aplicación.

Entre ellos destacan:

Room

Es una biblioteca de persistencia que permite crear y almacenar los datos en una base de datos SQLite.

Las partes de las que se compone Room son las siguientes:

- **Entity:** son clases que definen las tablas de la base de datos y de las entidades a utilizar.
- **DAO:** interfaces que definen los métodos utilizados para acceder a la base de datos.

- **RoomDatabase:** sirve de acceso a la base de datos SQLite a través de los DAOs definidos.

Además, es recomendable utilizar una clase intermedia a la cual denominamos Repository cuya finalidad es administrar las diferentes fuentes de datos.

LiveData

Se trata de una envoltura observable de nuestros datos que respeta el ciclo de vida de los componentes de la aplicación.

Ventajas:

- Datos actualizados. Cualquier cambio que se produzca en los datos LiveData será transmitido a cualquier vista o componente que lo esté observando. Esto significa que siempre se mantendrán actualizados.
- No más errores ni código defensivo para evitar activities/fragments inactivos. Como LiveData está asociado al ciclo de vida de la *activity* o *fragment*, no se producirán llamadas a sus observables si la vista no está activa.
- En el momento que ese objeto es destruido, los observadores se liberan de memoria, lo que evita fugas de memoria provocadas por pérdidas de referencia al área de memoria donde los objetos están almacenados y como consecuencia provoquen la falta de memoria en el dispositivo.

ViewBinding

ViewBinding es un mecanismo de acceso a las vistas de la aplicación que genera una clase de vinculación para cada XML de la aplicación que contiene referencias directas a los IDs de la vista, lo que hace que sea más seguro evitando punteros nulos, es decir que la referencia no tenga ningún retorno. Este problema lo tenía su antecesor, la función *findViewById()* (función para llamar a los IDs de la vista), además esta era mucho más costosa ya que cada vez que llamábamos a la función debía de recorrer todo el árbol jerárquico de la vista.

Dependencias

Las dependencias incluyen las bibliotecas al proyecto en tiempo de compilación

- Dependencias del proyecto

```
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.6.3'
        classpath 'com.google.gms:google-services:4.3.3'
        classpath 'com.android.databinding:dataBinder:1.0-rc1'
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
    }
}
```

Figura 8.5.- Dependencias del proyecto

- Dependencias de la aplicación

```
implementation 'com.google.firebase:firebase-database:19.4.0'
debugImplementation 'androidx.fragment:fragment-testing:1.2.5'
implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
implementation 'com.google.firebase:firebase-firestore:21.6.0'
implementation 'androidx.legacy:legacy-support-v4:1.0.0'
implementation 'com.google.android.material:material:1.2.0'
implementation 'androidx.navigation:navigation-fragment:2.3.0'
implementation 'androidx.navigation:navigation-ui:2.3.0'
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
testImplementation 'junit:junit:4.12'
androidTestImplementation 'androidx.test.ext:junit:1.1.2'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
testImplementation 'org.robolectric:robolectric:4.3'
implementation 'com.android.support:multidex:1.0.3'
implementation 'android.arch.persistence.room:runtime:2.2.3'
annotationProcessor 'android.arch.persistence.room:compiler:2.2.3'
implementation 'com.google.android.gms:play-services-maps:17.0.0'
implementation 'com.firebaseui:firebase-ui-auth:4.3.1'
implementation 'com.google.firebase:firebase-messaging:20.2.4'
implementation 'com.google.firebase:firebase-auth:19.3.2'
implementation 'com.google.firebase:firebase-firestore:21.6.0'
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

Figura 8.6.- Dependencias de la aplicación

9. Diseño

En este capítulo se describe el proceso de diseño de la aplicación, de acuerdo con las funcionalidades y requisitos descritos en los capítulos anteriores. El diseño facilita la comprensión del desarrollo software y las decisiones tomadas para la realización de la aplicación.

9.1.- DIAGRAMA DE NAVEGACIÓN

El diagrama describe las pantallas que tendrá la aplicación y la transición entre ellas, mostrando una visión general de la misma.

El perfil cliente de la aplicación tiene algunas restricciones sobre las pantallas a las que tiene acceso, mientras que el administrador podrá realizar la navegación completa que describe el diagrama. Para una mejor comprensión se ha realizado el diagrama en dos colores principales, verde y azul. Las pantallas verdes son de acceso para ambos perfiles, mientras que las pantallas azules solo están permitidas para el perfil administrador.

Cuando se inicia la aplicación, se abrirá siempre una ventana “Splash”, es una ventana visible temporalmente mientras se encarga de comprobar si el usuario ha iniciado sesión previamente desde su dispositivo. En caso afirmativo la siguiente pantalla a la que accederemos será la pantalla principal. Si por el contrario no se ha realizado ningún inicio de sesión nos llevará a la ventana de inicio.

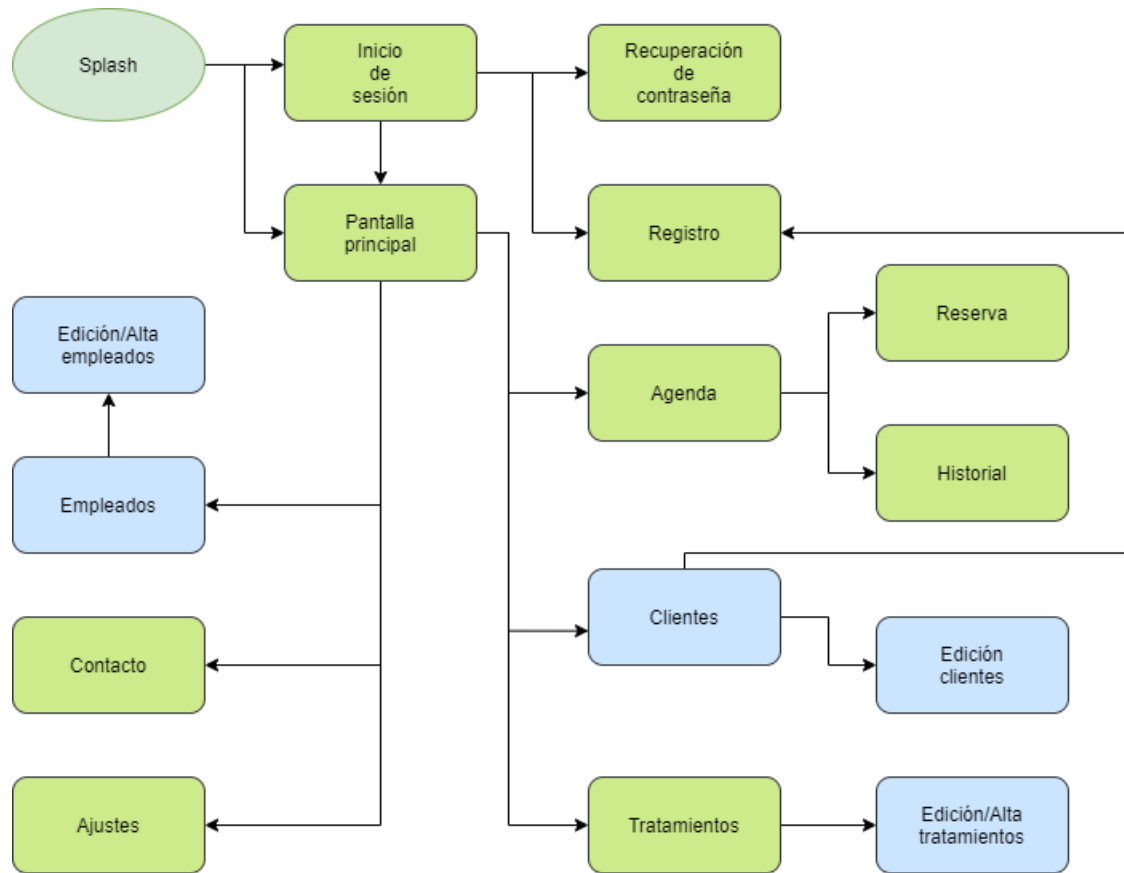


Figura 9.1.- Diagrama de navegación

9.2.- DISEÑO DE LA INTERFAZ

Toda la aplicación tiene un mismo estilo de diseño que la caracteriza. Se ha utilizado una gama de tonos tierra que transmitan tranquilidad y no generen demasiado ruido en la aplicación.

La única ventana que puede diferir del estilo del resto de la aplicación es “Inicio de sesión” que se ilustra en la figura 9.2, la cual tiene unos colores más oscuros, aunque sigue transmitiendo tranquilidad.

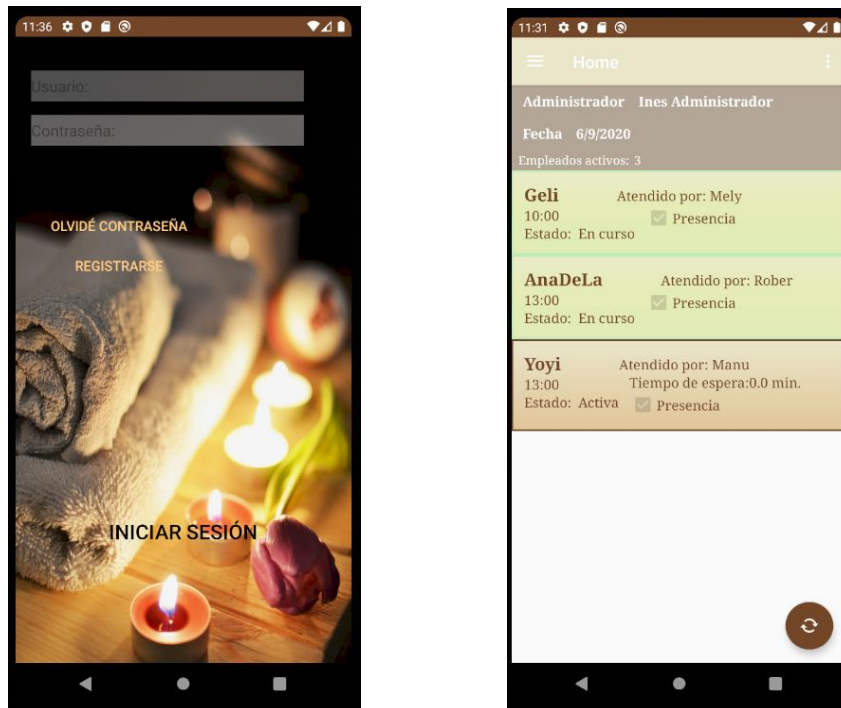


Figura 9.2. Pantallas inicio y principal

La navegación por la aplicación es intuitiva, en la parte superior izquierda tenemos tres líneas paralelas que si las seleccionamos se abrirá un menú mediante el cual accederemos a los distintos módulos de la aplicación.

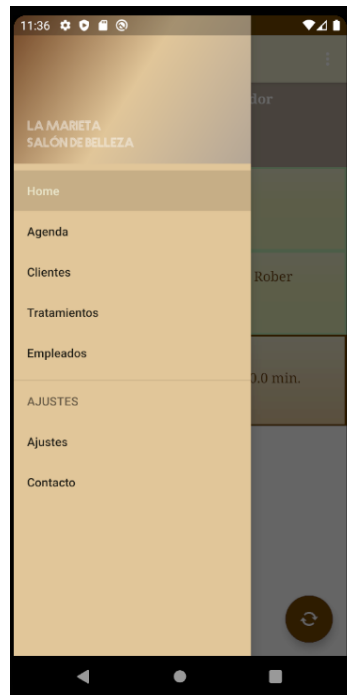


Figura 9.3.- Menú aplicación

Las pantallas que presentan datos, como empleados, tratamientos, clientes o el historial de citas, tendrán un modelo similar al representado en la pantalla que se observa a continuación en la figura 9.4.

Cuando en las pantallas se representen registros de información, se podrá editar o eliminar en el caso de que el perfil tenga permisos, deslizando hacia la izquierda o derecha.



Figura 9.4.- Pantalla clientes

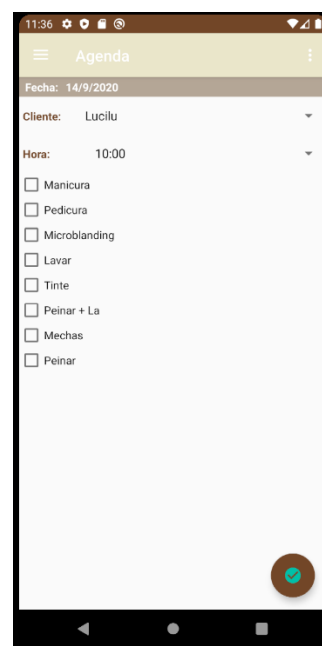


Figura 9.5.- Pantalla reserva

Las pantallas donde es necesario la introducción de datos, como la edición de clientes, empleados, registro, recuperación de contraseña, etc., por parte del usuario, tienen una estética más sencilla.

Durante la ejecución de la aplicación está nos ofrece avisos en forma de diálogos como en la figura 9.6 o con mensajes emergentes en la parte inferior de la aplicación.

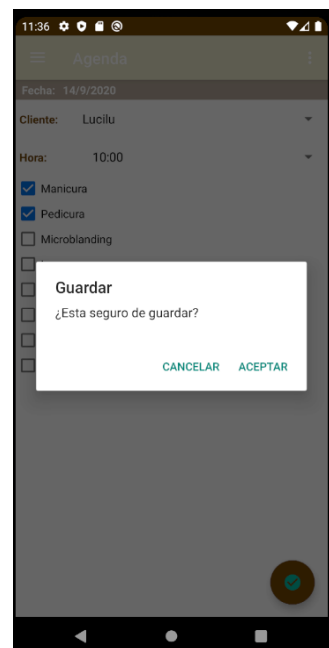


Figura 9.6.- Pantalla diálogo

9.3.- DISEÑO DE FUNCIONALIDADES

En este subapartado explicaremos el diseño aplicado a las funcionalidades de la aplicación en base a los requisitos y casos de uso planteados en los primeros capítulos.

9.3.1- Registro e inicio de sesión

Para explicar cómo se aplica el inicio de sesión en la aplicación, hablaremos principalmente de **Firestore Autenticación**.

Firestore Authentication es una biblioteca perteneciente a Firebase que nos permite autenticar usuarios mediante correo electrónico, teléfono o redes sociales y proporciona un alojamiento de los datos de sesión del usuario en la nube de forma segura. La implementación es muy sencilla puesto que contiene funciones propias que permiten realizar las operaciones más habituales.

Cuando un usuario se registra en la aplicación los datos introducidos se mandan al autenticador de firebase. Implementado las funciones correspondientes comprueba si el usuario ya existe. En caso afirmativo nos devuelve una excepción controlada con la que sacaremos el mensaje al usuario, mientras que si el usuario no estaba registrado se almacenará y se le enviará un correo electrónico de verificación de la cuenta que también gestiona el autenticador. Es imprescindible que el usuario se verifique en la dirección URL del correo recibido. Cuando el usuario inicia sesión si no se ha verificado no podrá acceder a la aplicación y saltará un mensaje indicando al usuario que debe verificarse en el correo electrónico, de esta forma controlamos que los usuarios de la aplicación son reales.

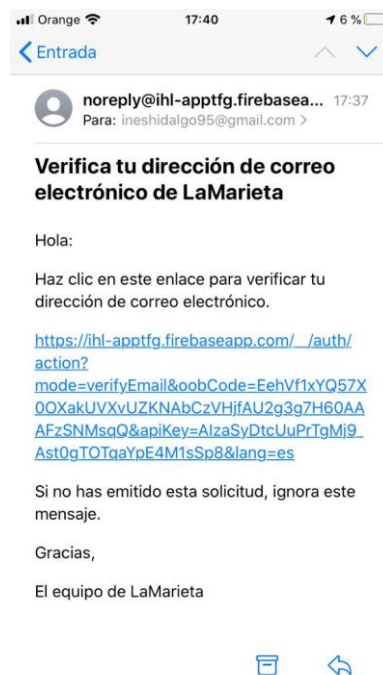


Figura 9.7.- Correo verificación

En resumen, para que el usuario inicie sesión es imprescindible que esté autenticado y verificado.

El resto de los datos personales que el usuario introduce durante el registro se almacenan en la base de datos de firebase. Cuando el inicio de sesión contra el autenticador es correcto, consulta la base de datos. Si el usuario no está en la base de datos no se iniciará sesión. En caso contrario obtendrá de ahí los datos necesarios para el inicio, como, por ejemplo, el perfil al que pertenece.

Si el usuario se olvida de su contraseña, el usuario puede solicitar a Firebase Authentication que le envíe un correo de recuperación de la misma.

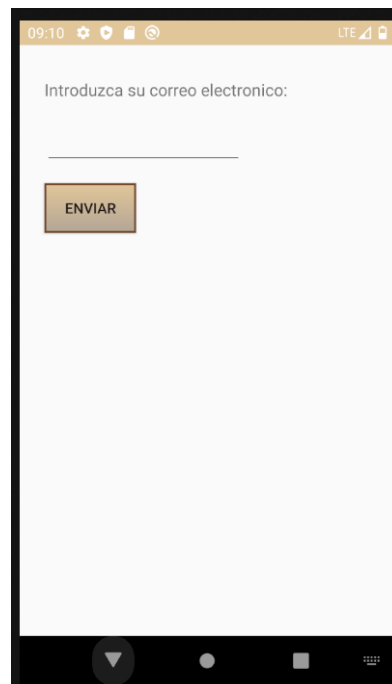


Figura 9.8.- Pantalla olvido de contraseña

9.3.2- Gestión de empleados y tratamientos

Los empleados y tratamientos serán parametrizados en la base de datos por el usuario con perfil administrador.

Cuando se quiere dar de alta o modificar un empleado, nos pedirá la hora de entrada y salida laboral. La aplicación solo permite introducir un horario continuo, puesto que es lo más habitual en las peluquerías ya que generalmente no cierran al mediodía. Si se quisiera introducir un horario partido, podemos introducir dos registros con diferentes horarios.

Los empleados pueden estar activos o no, de forma que, si un empleado se va de vacaciones y no queremos eliminarlo, podemos desactivarlo sin problema.

En el caso de los tratamientos nunca se podrán eliminar, pero si podremos desactivarlos y activarlos. Si un cliente ha realizado una reserva con un tratamiento y este es borrado, se perderían los datos del tratamiento a realizar.

Los clientes podrán consultar los tratamientos que hay activos, pero nunca modificarlos. Mientras que tal y como se expone en el diagrama de navegación, los clientes no accederán a consultar los empleados.

9.3.3.- Gestión de reservas

Cuando el cliente o el administrador quieren hacer una reserva estos acceden al día que seleccionan del calendario disponible en la agenda de la aplicación. Se abrirá una vista similar a la figura 9.9 , aunque en el caso del perfil cliente no se mostrará el desplegable para seleccionar el cliente, puesto que la cita se realiza para él mismo.

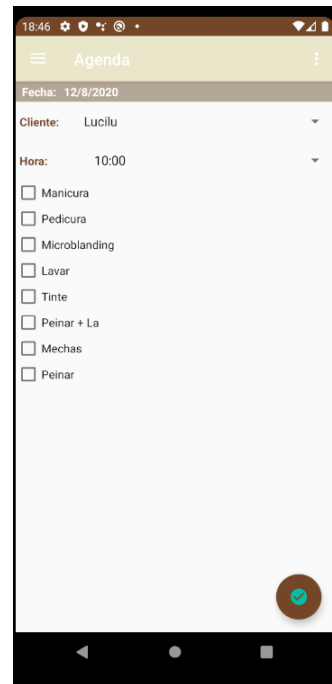


Figura 9.9.- Pantalla reserva

El desplegable de las horas, contiene solamente las horas disponibles para la cita para realizar esta comprobación es necesario obtener los siguientes datos:

- Horario comercial: El administrador podrá modificarlo en la sección de ajustes
- Empleados activos
- Citas del día seleccionado

Las citas se asignan cada media hora.

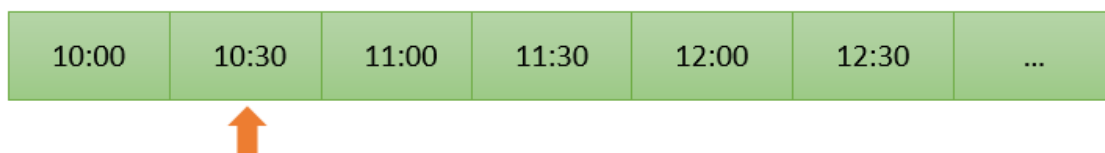


Figura 9.10.- Asignación

Para comprobar que horas están libres se recorre cada una de ellas y se comprueba cuántos empleados están activos en ese horario.

En cada hora se obtiene el número de citas que han reservado la misma hora o que continúen con su tratamiento. Si el número de citas calculadas es igual al número de empleados disponibles, ese hueco no estará libre. Por el contrario, si el número de citas es menor al de empleados, la hora estará disponible.

Por ejemplo, si nos fijamos en la **¡Error! No se encuentra el origen de la referencia.**, vemos que tenemos dos empleados disponibles. Hay dos citas a las 10:00 de la mañana. Si la cita del empleado 1 a las 10 de la mañana tiene una duración mayor a media hora y menor a 1 hora, el hueco de las 10:30 estará ocupado, pero en este caso para el empleado 2 hay una cita a las 10:00 de la mañana con una duración menor a media hora, por lo que, en el desplegable de horas de la aplicación, el cliente podrá seleccionar la hora de las 10:30.



Figura 9.11.- Ejemplo gestión horas

9.3.4- Gestión de cola

La gestión de la cola FIFO establece que el primer cliente que llega será el primero en pasar. Es decir, en la peluquería el horario de la cita reservada es orientativo, pero que un cliente llegue antes o después dependerá para calcular el tiempo de espera que estará en sala.



Figura 9.12.- Pantalla gestión de cola

Para explicar correctamente el funcionamiento de la cola, explicaremos los estados por los que pasa el cliente.

Las citas de la cola de espera pueden pasar por 5 estados, pero solo estarán visibles en la pantalla principal las citas que corresponden a los 3 primeros estados.

ID	Estado	Presencia
1	Activa	X
2	Activa (En espera)	Presente
3	En curso	Presente
4	Finalizada	X
5	Cancelada	X

Tabla 9.1.- Tabla gestión de estados

Cuando se realiza una reserva todas las citas están activas (Estado 1).

Cuando el cliente llega el día de su cita a la peluquería el administrador deslizará su registro para añadirle la presencia (Estado 1), con lo que el registro quedará marcado como

presente igual que se puede ver en la figura 9.12. Esto indica que el cliente se encuentra en la sala de espera.

En el momento en el que haya un empleado libre se le asigna y el tiempo de espera pasa a ser 0, por tanto, el cliente podrá pasar a realizar sus tratamientos (Estado 3). El administrador es el encargado de realizar el cambio puesto que por varios motivos el cliente no tiene por qué pasar inmediatamente y de esta forma se aproximan más los tiempos de espera a los reales.

Como la duración del tratamiento es aproximada, cuando el cliente acaba, el administrador le cambiará a finalizada (Estado 4).

Aunque el cliente puede llegar a la peluquería en cualquier momento aproximado a su cita, no está permitido que llegue más de 1 hora tarde. Cuando un cliente no está presente y ha pasado más de 1 hora de la hora de su reserva, la cita se cancela automáticamente (Estado 5).

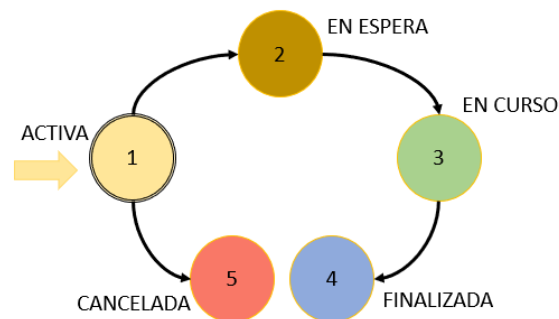


Figura 9.13.- Estados cliente

Cálculo del tiempo de espera

Para calcular el tiempo de espera de los clientes que están en espera necesitamos los siguientes datos.

- Empleados disponibles y activos
- Lista de citas en la cola: Ordenada de menor a mayor fecha de llegada. Los clientes que no hayan llegado estarán a continuación ordenados por fecha de reserva

- Lista de tiempos de espera

La base del cálculo la realiza una lista de tiempos de espera. Esta lista tendrá un tamaño máximo de los empleados que hay en ese momento. Cuando hay un empleado y llega el primer cliente del día a la peluquería, tras pasar por el estado de presencia **en la sala** de espera, ponemos al cliente **en curso**. En este momento se añade a la lista de tiempos de espera el tiempo que le queda al cliente por acabar y como consecuencia el empleado asignado quede libre para otro cliente. Además, se actualiza la fecha de fin aproximada, calculada por los tratamientos que tiene reservados. El tiempo que le queda al cliente para finalizar se calcula restándole a la hora de finalización la hora actual. De esta forma cuando otro cliente llegue a la peluquería su tiempo de espera lo recogerá de la lista de tiempos de espera, eliminará el tiempo que se ha asignado a él y se introducirá el nuevo valor. En los clientes que están en sala el **nuevo valor** se obtiene de la duración de los tratamientos más el tiempo de espera asignado.

Una vez explicado el concepto con un empleado, explicamos un ejemplo con tres empleados.

Si tenemos tres empleados disponibles, los tres primeros clientes pasan de forma casi directa a estar en curso y se guardan sus tiempos en una lista.

LISTA DE TIEMPOS DE ESPERA	45	15	30
----------------------------------	----	----	----

Figura 9.14.- Tiempos de espera

Cuando llega un cuarto cliente (sin que ninguno de los tres anteriores haya finalizado), escogerá de la lista el tiempo mínimo de espera, en este ejemplo 15.

Eliminará de la lista de espera el 15 y lo sustituirá por la suma de su espera más la duración de su tratamiento.



Figura 9.15.- Ejemplo cálculo tiempo de espera

A partir de este momento todos los clientes que vayan llegando a la peluquería realizarán la misma operación. Cuando un cliente finaliza o pasa su estado a en curso, no es necesario eliminarlo de la lista puesto que al actualizar el registro del cliente en firebase y ser la aplicación en tiempo real, los clientes de la cola se vuelven a recalcular y sería la misma operación.

10.Desarrollo

10.1.- COMPATIBILIDAD DE LA APLICACIÓN

Es importante que la aplicación sea compatible con el mayor número de usuarios posibles, según información proporcionada por Google en la tabla 10.1 podemos observar el porcentaje de uso de las diferentes versiones de los usuarios.

La aplicación ha sido auditada para funcionar al 100% de su capacidad en dispositivos con una API superior a la 21. Por tanto, la aplicación será compatible en un 84,9% con los dispositivos de los usuarios de la misma.

Versión	Nivel API	Porcentaje de usuarios
ICE CREAM SANDWICH (4.0)	15	0,2%
JELLY BEAN (4.1-4.3)	16-18	1,7%
KITKAT (4.4)	19	4%
LOLLIPOP (5.0 – 5.1)	20-21	9,2%
MARSHMALLOW (6.0)	22	11,2%
NOUGAT (7.0 – 7.1)	24-25	12,9%
OREO (8.0 – 8.1)	26-27	21,3%
PIE (9.0)	28	31,3%
ANDROID 10 (10.0)	29	8,2%

Tabla 10.1.- Distribución de versiones por uso de usuarios

10.2.- ESTRUCTURA DE LA APLICACIÓN

La estructura principal del proyecto se basa en una estructura típica en la que se encuentran los directorios generados automáticamente por Android Studio. Podemos observar esta estructura principal en la figura 10.1.

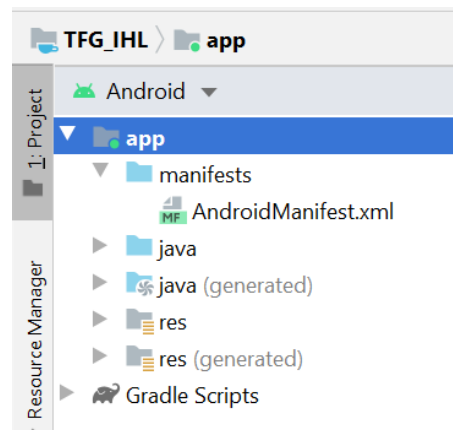


Figura 10.1.- Estructura de la aplicación

En el directorio /java se encuentra todo el código fuente de nuestra aplicación, tanto el código para la interfaz gráfica, como las clases necesarias para aplicar el patrón MVVM.

Entre los paquetes que podemos observar en la figura 10.2 , cabe destacar el **UI**. Éste contiene todos los fragmentos de las diferentes secciones de la aplicación y las clases que estos necesitan directamente.

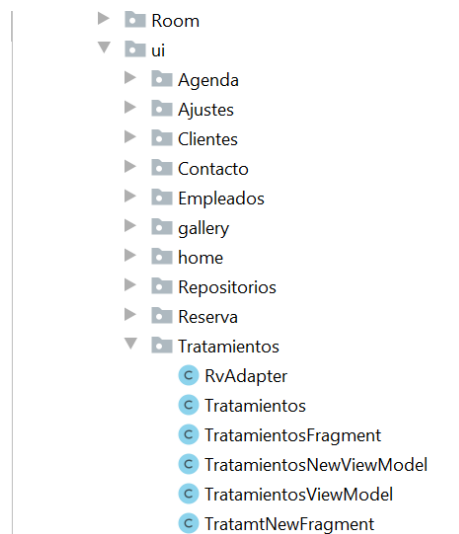


Figura 10.2.- Estructura aplicación

También nos encontramos con el paquete Repositorios, que incluye las clases repositorios que se comunican con Firebase para almacenar, actualizar o eliminar los datos.

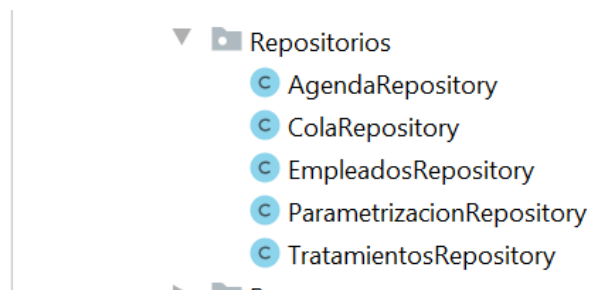


Figura 10.3- Estructura aplicación. Repositorios.

En la **¡Error! No se encuentra el origen de la referencia.**, remarcamos el paquete Room, que contiene las entidades de la base de datos.

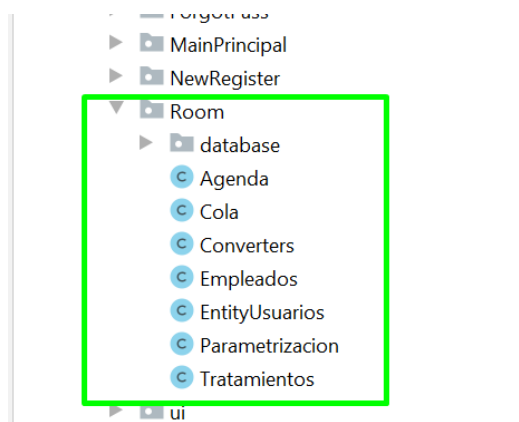


Figura 10.4.- Estructura de la aplicación. Entidades.

En el **directorio /res** se almacenan todos los recursos que utilizará la aplicación, podemos observarlo en la figura 10.5

Cada tipo de recurso se almacena en un subdirectorio diferente.

En **/drawable** se almacenan las imágenes u otros archivos gráficos xml como formas, animaciones, etc...

/font contiene los archivos fuentes del estilo de los textos.

/layout contiene los xml de diseño de la interfaz, los cuales están ligados a una actividad o fragment de la aplicación.

/menu contiene el menú de la aplicación, en nuestro caso tenemos un navigation drawer.

/mipmap archivos de diseño como iconos para diferentes densidades.

/navigation contiene el .xml que indica la navegación entre las interfaces de la aplicación

/values contiene ficheros xml de recursos fijos de la aplicación como strings, estilos, colores, etc ... que se utilizarán en los diferentes recursos.

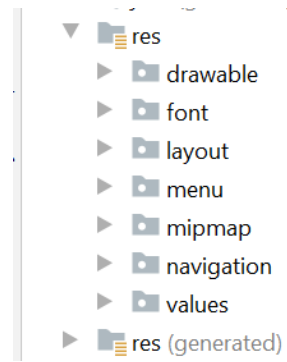


Figura 10.5.- Estructura del proyecto. Recursos

11.Pruebas

11.1.- PRUEBAS

Durante todo el desarrollo de la aplicación se han estado realizando pruebas de la misma según se añadían funcionalidades o porciones de estas. Se han realizado tanto pruebas estáticas, revisando el código desarrollado línea a línea, como dinámicas, ejecutando la aplicación y comprobando cada uno de los requisitos planteados en el análisis del proyecto.

Es importante realizar pruebas de manera iterativa según vaya aumentando el código.

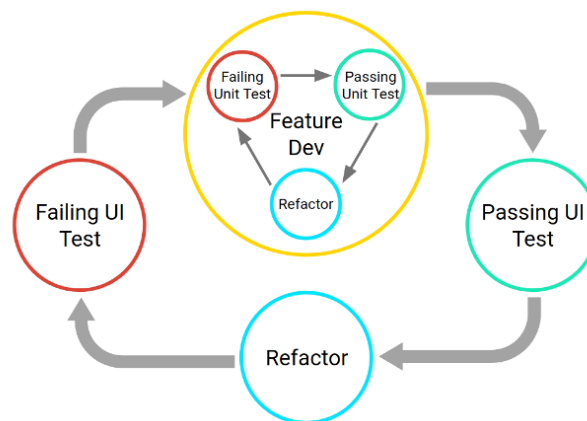


Figura 11.1.- Relación de pruebas

En la última fase del proyecto se han realizado las pruebas integradas con la aplicación al completo.

Antes de la ejecución de las pruebas se ha realizado un plan de pruebas, de manera que estas queden documentadas.

A continuación, a modo ejemplo, se muestra el plan de pruebas para las **Sesiones y usuarios**.

Nu me ro	Descripción	Steps	Resultado esperado	Resultado (OK/KO)
1	Registro de un nuevo usuario	1. Abrir aplicación	La aplicación se abre	OK
1		2. Seleccionar Registrarse	Se abre el formulario de registro	OK
1		3. Ingresar los datos y aceptar	La aplicación informa de que el usuario ha de verificarse por correo electrónico	OK
2	Registro de usuario existente y verificado	1. Abrir aplicación		OK
2		2. Seleccionar Registrarse		OK
2		3. Ingresar los datos y aceptar	La aplicación informa de que el usuario ya existe	OK
3	Registro de usuario existente y no verificado	1. Abrir aplicación		OK
3		2. Seleccionar Registrarse		OK
3		3. Ingresar los datos y aceptar	La aplicación informa de que el usuario existe y debe ser verificado	OK
4	Registro de usuario con correo falso	1. Abrir aplicación	La aplicación se abre	OK
4		2. Seleccionar Registrarse	Se abre el formulario de registro	OK
4		3. Ingresar los datos y aceptar	La aplicación informa que el usuario ha de verificarse por correo electrónico	OK
5	Iniciar sesión con un correo dado de alta y no verificado	1. Abrir aplicación		OK
5		2. Ingresar datos de usuario e iniciar sesión	La aplicación informa de que el usuario no está verificado	OK
6	Iniciar sesión con un correo no registrado	1. Abrir aplicación		OK
6		2. Ingresar datos de usuario e iniciar sesión	La aplicación informa de que el usuario no existe	OK
7	Dar de baja a un usuario desde el administrador	1. Acceder a clientes		OK
7		2. Seleccionar cliente para dar de baja		OK
7		3. Comprobar estado en bbdd		OK

8	Dar de baja a un usuario desde el cliente	1. Acceder a clientes		OK
8		2. Editar cliente	Se abre una nueva ventana con los datos del cliente	OK
8		3. Seleccionar baja cliente y aceptar	La aplicación informa de que el cliente ha sido dado de baja	OK
9	Iniciar sesión con un usuario dado de baja	1. Abrir aplicación		OK
9		2. Ingresar datos de usuario e iniciar sesión	La aplicación informa de que el cliente ha sido dado de baja	OK
10	Iniciar sesión con un correo dado de alta y verificado	1. Abrir aplicación		OK
		2. Ingresar datos de usuario e iniciar sesión	La aplicación accede al home	OK

Tabla 11.1.- Plan de pruebas inicio de sesión

10.2.- DISPOSITIVOS PARA LAS PRUEBAS

En este proyecto todas las pruebas realizadas se harán en un dispositivo virtual, estos tipos de dispositivos proporcionan una mayor velocidad de prueba, a coste de una menor calidad que en un dispositivo físico.

En el uso diario de las pruebas utilizamos un Nexus de 5" con una API 24.

Se realizaron pruebas integradas de la aplicación con diferentes dispositivos simulados, con el fin de controlar el diseño de la interfaz y el correcto funcionamiento en las diferentes versiones de la API.

- Nexus 6 API 30
- Pixel 3a API 27
- Pixel 3XL API 26
- Pixel 2 API 29
- Galaxy Nexus API 22

12. Presupuesto

En este capítulo se detallará el presupuesto total del proyecto. Dándose en detalle el coste de la parte de firebase y del análisis y desarrollo del mismo.

12.1.- COSTE DE FIREBASE

La base de datos de Cloud Firestore tiene un coste que se puede clasificar en tres aspectos.

- Operaciones
- Almacenamiento
- Ancho de banda de red

La mayor parte de las operaciones serán de lectura. El administrador abrirá la aplicación de continuo. Un cliente es probable que abra la aplicación una vez a la semana de media, cuando vaya a reservar la cita o cuando asista al negocio. Algunos clientes visitan la peluquería una vez cada tres meses, mientras otros suelen venir cada semana. Teniendo además en cuenta que el día de la cita se realiza un mayor consumo.

Almacenamiento

El **tamaño de un documento** es la suma de lo siguiente:

- El tamaño del nombre del documento
- La suma del tamaño de string de cada nombre de campo
- La suma del tamaño de cada valor de campo
- 32 bytes adicionales

Almacenamiento por tipo de dato.

Valor de cada campo	Tamaño
String	Cantidad de bytes con codificación UTF-8 + 1
Integer	8 bytes
Date	8 bytes
Double	8 bytes
Array	Suma de los tamaños de sus valores

Tabla 12.1.- Almacenamiento por tipo de dato

Con estos valores calculamos el almacenamiento de un documento de cada colección/tabla y con el número aproximado de documentos que tendremos, obtenemos el tamaño total aproximado.

Nombre colección	Documentos aproximados	Tamaño medio por un documento	Tamaño total aproximado (bytes)
TABLA_AGENDA_USU	6000	191 bytes	1146000
TABLA_COLA_ESPERA	100	325 bytes	32500
TABLA_EMPLEADOS	20	139 bytes	2780
TABLA_PARAMETRIZACION	20	123 bytes	2460
TABLA_TRATAMIENTOS	50	145 bytes	7250
TABLA_USUARIOS	5000	253 bytes	1265000
TOTAL			2455990

Tabla 12.2.- Almacenamiento total aproximado de firebase

Coste total

En condiciones normales el negocio admite citas durante 10 horas al día. Habitualmente no habrá más de 10 empleados trabajando y de media los clientes realizan tratamientos de 1 hora. Por tanto, el negocio tendrá un **máximo aproximado de 100 citas diarias**. Los usuarios de las 100 citas accederán el mismo día a la aplicación para controlar sus tiempos de espera y suponemos que otros 100 accederán para reservas u otros motivos.

Citas diarias	Usuarios diarios	Lecturas	Escrituras	Eliminaciones
---------------	------------------	----------	------------	---------------

100	200	2000000	200000	100000
-----	-----	---------	--------	--------

Tabla 12.3.- Ancho de banda por operaciones

Con los datos obtenidos en la tabla anterior, accedemos a la URL del cloud de Google se puede obtener el precio por mes basándonos en el número de lecturas y escrituras diarias.

<https://cloud.google.com/products/calculator#id=0db1cd62-ef61-4be2-b914-582bbbce6f5f>

Añadimos 1GB de memoria, para obtener un coste máximo del servicio basándonos en las predicciones realizadas.

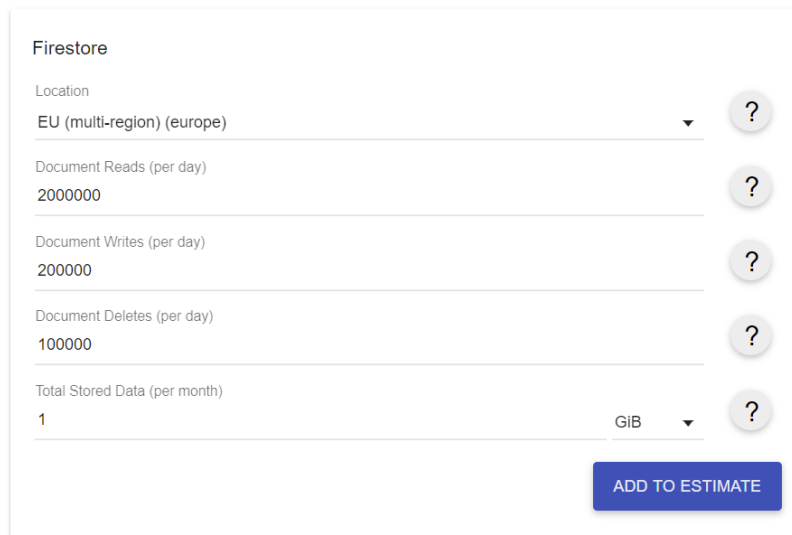


Figura 12.1.- Valores cálculo coste Firebase

El coste total será de 55,10€ /mes

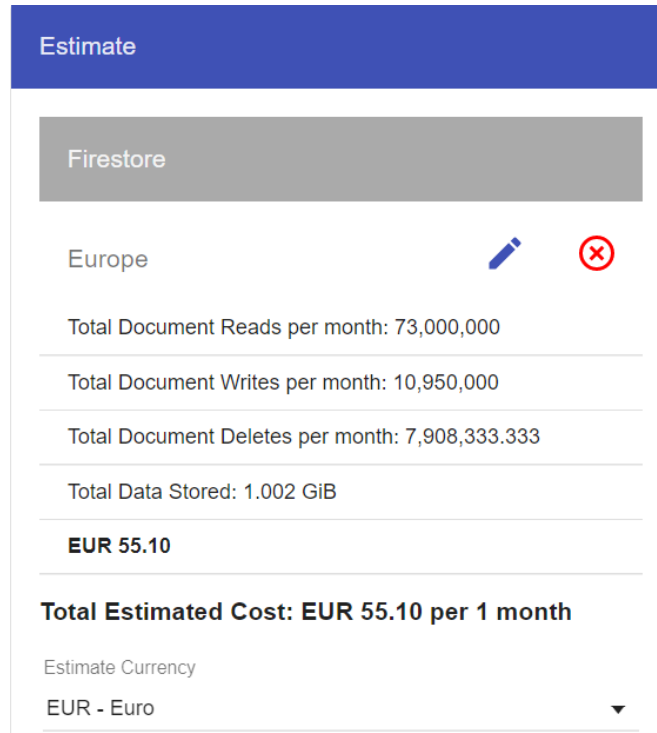


Figura 12.2.- Coste Firebase al mes

12.2.- COSTE DEL PROYECTO

Para el **cálculo del personal** se ha tenido en cuenta el salario de un profesional cuya categoría se encuentra entre programador-analista, percibiendo este en su cuenta un salario bruto de 1300€ mensuales. El gasto que tiene la empresa es mayor que el que percibe este puesto que deberá añadir al sueldo percibido los costes de la seguridad social, por tanto, calcularemos el coste real del mismo.

De la siguiente formula obtenemos el coste de la seguridad social.

$$\text{Coste de la Seguridad Social} = (\text{Sueldo bruto} * 0,2360) + (\text{Sueldo bruto} * 0,055) + (\text{Sueldo bruto} * 0,0020) + (\text{Sueldo bruto} * 0,0070)$$

La empresa pagará en total 1690€/mes por el profesional.

$\begin{aligned} \text{Coste del trabajador} &= \text{Sueldo bruto} + \text{Coste de la Seguridad Social} \\ 1690 &= 1300 + 390 \end{aligned}$

El cálculo del equipo se obtiene de todos aquellos elementos físicos para realizar el desarrollo del proyecto. Estos costes serán fijos.

Ordenador	600€
Herramientas de oficina	100€

Tabla 12.4.- Costes de equipo

Calculamos el coste total, con la suma de los anteriores cálculos. En el caso de los costes humanos, se multiplicará el gasto mensual por los 6 meses del desarrollo completo del proyecto.

El cálculo del coste de software implica el coste de Firebase durante un año.

Coste humano	Coste de equipo	Coste software	Total
1690€ x 6	700€	55,10x12	11501,2€

Tabla 12.5.- Coste total proyecto

13. Trabajos futuros

Como continuación de este trabajo la cola actual de clientes en la sala de espera se controlará mediante un sensor como Arduino, lo que automatizará la entrada de clientes y el acceso a sala de éstos en la peluquería cuando se levanten.

De esta manera el administrador se desprenderá del trabajo manual de notificar a la aplicación cada salida y cambio de estado de los clientes.

Para el desarrollo de esta funcionalidad tan solo se deberá de obtener los cambios del sensor y en base a estos actualizar el estado del cliente. Una vez realizada la modificación, el resto del código será compatible puesto que no es dependiente de los métodos que se realizan para actualizar el tiempo de espera y asignar el empleado en el momento del acceso a sala.

Por otro lado, otros trabajos a futuro pueden ser:

- Trabajo de mantenimiento: El sistema operativo se actualiza asiduamente por lo que serán necesarios proyectos de mantenimiento de la aplicación.
- Trabajo de seguridad: Es necesario realizar un proyecto para implementar una capa más alta de seguridad.
- Trabajo interfaz gráfica: Actualizar la interfaz de la aplicación para obtener un acabado más profesional.

14. Conclusiones

Finalmente se han cumplido los objetivos principales del proyecto. El principal objetivo era crear una cola FIFO para la lista de espera de los clientes en la peluquería, basándonos en este objetivo principal, se ha desarrollado una aplicación completa para que tanto los clientes como el administrador puedan manejar sus datos y hacer todas las acciones que rutinariamente harían por teléfono, mediante la aplicación, tales como la reserva de citas, la consulta de tratamientos o el tiempo de espera.

Durante el análisis del proyecto se han intentado aplicar conocimientos adquiridos durante el grado, de los cuales aprendí a analizar los requisitos de la aplicación, la creación de casos de uso y sus flujos e incluso la planificación con el diagrama de Gantt.

La etapa del desarrollo fue en mi caso la más complicada, puesto que elegí este proyecto como un “reto” ya que nunca había trabajado con Android. Aunque el lenguaje si lo manejaba, la arquitectura de las aplicaciones móviles o el desarrollo de la interfaz era totalmente desconocida. Sin duda considero que el conocimiento más importante que otorga el grado es la capacidad de adaptación y de aprender apoyándonos en los recursos que tenemos.

15. Bibliografía

Android Hive - Firebase Realtime Database. (01 de 05 de 2020). Obtenido de <https://www.androidhive.info/2016/10/android-working-with-firebase-realtime-database/>

Developer Android - Compatibilidad. (07 de 03 de 2020). Obtenido de <https://developer.android.com/guide/practices/compatibility>

Developer Android - JetPack. (01 de 04 de 2020). Obtenido de <https://developer.android.com/jetpack>

El baúl del programador. (05 de 2020). Obtenido de <https://elbauldelprogramador.com/fundamentos-programacion-android/>

Firebase. (20 de 06 de 2020). Obtenido de <https://firebase.google.com/?hl=es>

Five - Clean Architecture. (25 de 03 de 2020). Obtenido de <http://five.agency/android-architecture-part-3-applying-clean-architecture-android/>

Hermosa programación - Estructura Android. (15 de 03 de 2020). Obtenido de <http://www.hermosaprogramacion.com/2014/08/android-studio-proyecto-en/>

Platzi - Arquitectura. (20 de 05 de 2020). Obtenido de <https://platzi.com/blog/arquitecturas-de-software-en-android-mvc-mvp-y-mvvm/>

Xataka Android. (08 de 2020). Obtenido de <https://www.xatakandroid.com/sistema-operativo/android-10-esta-1-cada-10-dispositivos-ultimos-datos-distribucion-versiones>