ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DESARROLLO DE UNA APLICACIÓN MÓVIL ANDROID PARA EL SEGUIMIENTO DE LAS PRÁCTICAS DE CONDUCIR (DRIVINGLIA)

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Telemática

Autor: Luis Plasencia Pulido

Tutor: Dr. Luis Hernández Acosta

Fecha: Julio 2021

ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DESARROLLO DE UNA APLICACIÓN MÓVIL ANDROID PARA EL SEGUIMIENTO DE LAS PRÁCTICAS DE CONDUCIR (DRIVINGLIA)

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal **Secretario**

Fdo.: Fdo.:

Fecha: Julio 2021

Resumen

Las autoescuelas en toda España forman a la ciudadanía para aprender a conducir en vías públicas. En los últimos años y con el avance de la tecnología han surgido nuevas tendencias para apoyar en la labor de aprendizaje de los alumnos. Las aplicaciones móviles disponibles actualmente están orientadas a la parte teórica del permiso de conducir sin hacer mucho hincapié a la parte práctica.

Drivinglia es una aplicación móvil para dispositivos Android que surge con la premisa de facilitar el seguimiento de las prácticas de conducir de una forma interactiva y didáctica para reducir el índice de suspensos del examen práctico. Cuenta con perfiles de uso que facilitan la labor de calificación y seguimiento de las prácticas y nos permiten administrar de manera individualizada y ágil los distintos usuarios registrados en la base de datos.

De esta manera, el alumno será más consciente de los errores que comete en el desarrollo de las prácticas en función de los criterios de calificación de la DGT (Dirección General de Tráfico) y de las faltas de conducción recogidas en la normativa vigente. Por lo tanto, a la hora del examinarse estará más preparado para afrontar los retos que se le imponga.

Abstract

Driving schools all over Spain prepare citizens to learn how to drive in public roads. Over the last couple of years, mobile technology has skyrocketed and as such, new tendencies to further develop the learning experience of the students have appeared. Among the thousands of mobile applications available online at the moment, only a handful take the practical part of the driving test into consideration, the majority are aimed towards helping the students learn the theorical application for the multiple-choice test that they first have to pass.

Drivinglia is an Android based mobile application that will help teachers and students keep track of the practical driving lessons in a didactically rich and interactive environment with the ultimate intention being to reduce the number of failed practical exams. It has different profiles of use that help monitoring and evaluating the lessons as well as individually managing the different users registered in the database in an agile way.

At the end of the day, students will be more aware of the different errors that are committed in the driving lessons depending on the grading criteria of the DGT (Directorate-General of Traffic) and the different driving test faults gathered in the regulations in force. Hence, students will be more prepared to face the different challenges that arise at the time of having their practical exam.

Tabla de contenido

Índice de ilustraciones

Índice de Figuras

Índice de Tablas

1. Introducción

En este capítulo presentaremos el estudio previo a la elaboración de la aplicación. También estableceremos los objetivos iniciales del proyecto y detallaremos la estructura del documento.

1.1. Introducción

1.1.1. Contexto

Las autoescuelas en toda España preparan cada año a miles de alumnos para la obtención de los distintos permisos de conducir vigentes. A lo largo del periodo académico de enseñanza, se prepara al alumno para que consiga los conocimientos teóricos y prácticos recogidos por la Dirección General de Tráfico (DGT).

Eso sí, según estudios obtenidos en 2015 a partir de las más de 6.000 autoescuelas registradas en España, solamente alrededor de un 20% de los alumnos aprueban el examen práctico a la primera [1].

Desde nuestro punto de vista, los altos índices de fracaso en el número de aprobados se podrían mejorar si el alumno tuviera un seguimiento más detallado y organizado de las consecutivas prácticas que va realizando. De esta forma, el alumno estaría más familiarizado con sus puntos débiles y aspectos a mejorar. Además, se podrían recoger datos reales acerca de los puntos críticos y los fallos más comunes de los alumnos.

Actualmente, el profesor de prácticas de autoescuela recoge el seguimiento del alumno a mano en una libreta o cuadernillo. Sin embargo, gran parte de esta información no se puede tratar y digitalizar fácil y cómodamente. De forma que termina siendo un recordatorio para el profesor más que una pieza de material útil e informativo hacia el alumno del que podamos extraer datos tangibles para mejorar la enseñanza.

1.1.2. Motivación

Drivinglia surge con la premisa de mejorar el seguimiento de las prácticas de autoescuela para conseguir un sistema ágil y útil del que tanto alumno, profesor de prácticas y autoescuela puedan sacar beneficio.

El alumno será más consciente de sus errores y puntos débiles de forma que, como propósito final de este trabajo, se mejoren los índices de aprobado de los exámenes prácticos de conducir en las autoescuelas que utilicen Drivinglia en la enseñanza.

Además, aquellas autoescuelas que utilicen Drivinglia podrán presumir de unos índices de aprobado más elevados que mejorarían el prestigio de estas.

1.1.3. Estado del arte

Para reforzar y perfilar el planteamiento y los objetivos de nuestra aplicación, hemos tenido contacto con Alexis Ramírez González, profesor de la autoescuela Carmelo. Alexis nos ha contado de primera mano su experiencia dando prácticas en diferentes organismos a lo largo de su carrera profesional de más de 25 años. Nos comenta que, no hace muchos años, se utilizaba una aplicación multiplataforma dedicada y protegida para el seguimiento de prácticas de la que se podían extraer datos. No obstante, dicha aplicación no cumplía con los requisitos óptimos para un uso seguro en la carretera que mantenga las distracciones al mínimo. Dicha aportación nos ha servido para orientar nuestra aplicación a un entorno real en el que se dote al profesor de las herramientas necesarias para un uso ágil, eficiente y seguro de la aplicación en ruta.

La aplicación se desarrollará nativamente para Android con el lenguaje de programación Java [2]. Se barajaron otras opciones de desarrollo como la posibilidad de realizar la aplicación utilizando un framework multiplataforma como Angular o Ionic de forma que se pudiera utilizar la aplicación tanto en dispositivos iOS como Android [3] [4]. Sin embargo, la consistencia y flexibilidad que nos otorgaba el desarrollo nativo, así como el nivel de rendimiento que nos permitía alcanzar en el tracking nos hizo decantarnos por esta opción. Además, podríamos mejorar la experiencia de usuario implementando elementos de la vista con los que ya estábamos familiarizados [5].

Drivinglia utilizará la base de datos en tiempo real en la nube no relacional de Firebase, para tratar la información de alumnos y profesores [6]. Hemos optado por esta solución ya que los datos con los que tratamos suelen actualizarse con relativa frecuencia y ser susceptibles a modificaciones.

Cada práctica de conducir incluirá faltas o anotaciones apuntadas por el profesor según los criterios de calificación oficiales de la prueba de control de aptitudes y comportamientos en vías abiertas al tráfico general de la DGT [7].

Utilizaremos metodologías ágiles para el desarrollo del proyecto software. En concreto la metodología *scrum* basada en *sprints*. De modo que se analicen las tareas y se mantenga un seguimiento activo del desarrollo del proyecto entre el profesor y el alumno teniendo en cuenta nuestro posible cliente, la autoescuela [8]. Para ello se organizarán las tareas en un kanban haciendo uso de la plataforma *Trello* y utilizaremos un repositorio online libre de *Github* para tener un resguardo del código y llevar un seguimiento activo del proyecto.

Además, nos basaremos en el patrón arquitectónico Modelo-Vista-Presentador (MVP) a la hora de programar la aplicación. Con este patrón, podremos separar los datos de la vista a través del presentador para manejar la información y la lógica de la aplicación de forma eficiente y clara [9].

Por último, implementaremos algunos de los principios del Clean Code (Código Limpio) para escribir el código de forma mantenible a vistas futuras y mejorar la productividad del ciclo de trabajo [10].

1.2. Comparativa con aplicaciones de Google Play

Analizando las distintas aplicaciones del sector de las autoescuelas disponibles en la tienda de Google Play, destacamos las siguientes:

1.2.1. Autoescuela Clases Prácticas

Ilustración 1: Autoescuela Clases Prácticas

Autor: Confederación Nacional de Autoescuelas – AUCO S.I.

Fecha de publicación: 13 de junio de 2018

Última actualización: 5 de abril de 2021

Aplicación destinada a mantener el control de las clases prácticas desde el coche. Tiene las funciones de comprobar el calendario, acceder a ficha de alumnos e iniciar las clases. Almacena las firmas de alumno y profesor en el terminal y es apoyado por el programa eGestion para conocer en tiempo real la actividad de los profesores registrados desde la oficina. El logo lo podemos ver en la Ilustración 1.

Cuenta con alrededor de 1000 descargas por lo que no es una aplicación particularmente conocida y no parece ser capaz de apuntar las faltas cometidas en el momento de la práctica. En general, es una aplicación destinada al profesor para el seguimiento de las prácticas que van realizando sus alumnos y para tener contacto con ellos puntualmente o apuntar observaciones a modo de herramienta de gestión y planning [11].

1.2.2. TodoTest: Test de conducir

Ilustración 2: TodoTest

Autor: Autoinet

Fecha de publicación: 20 de julio 2012

Última actualización: 2 de junio de 2021

Todotest es una aplicación destinada a la realización de test teóricos con los contenidos oficiales de la DGT (Dirección General de Tráfico) de los diferentes permisos de conducir disponibles. La aplicación registra los fallos que vamos cometiendo y cuenta con una sección que recoge las últimas preguntas y casos publicados por el ministerio de Fomento de CAP (Competencia profesional y consejero de seguridad). Por otro lado, cuenta con una sección premium de pago con las preguntas de los exámenes más recientes del permiso B [12].

Cuenta con más de 1 millón de descargas por lo que nos encontramos ante una aplicación de notable éxito y repercusión. El logo lo podemos ver en la Ilustración 2.

Al estar orientada únicamente a la parte teórica del examen de conducir, guarda pocas similitudes con Drivinglia.

1.2.3. Dribo – La autoescuela en tu móvil

Ilustración 3: Dribo

Autor: Dribo

Fecha de publicación: 22 de diciembre de 2016

Última actualización: 23 de junio de 2021

Dribo es una aplicación que hace las funciones de una autoescuela de manera portátil. Está destinada a los alumnos que buscan conseguir el permiso de conducir B y les facilita clases y pruebas online desde la aplicación para la preparación de la parte teórica del permiso así como la capacidad de elegir el precio y la hora de práctica que más le convenga. La aplicación también facilita al alumno tener contacto con su profesor de prácticas y tramitar toda la documentación y pagos con una interfaz sencilla [13].

Cuenta con más de 100 mil descargas así que nos encontramos ante una aplicación ciertamente vanguardista y exitosa. Sin embargo, no es capaz de realizar seguimiento de las prácticas de conducción sino solamente tener contacto con el profesor y elegir el precio y la fecha de la clase. El logo lo podemos ver en la Ilustración 3.

Aparte de las 3 aplicaciones que hemos mencionado, la mayoría de las que encontramos guardan grandes similitudes con *TodoTest*. Es decir, la mayoría de las aplicaciones destinadas a los alumnos de autoescuela tienen un contenido educativo y están orientadas a la parte teórica del examen del permiso de conducir.

Por otro lado, hemos encontrado aplicaciones de gestión de las prácticas destinadas a la organización y contacto con el alumno por parte del profesor como la primera de todas e incluso aplicaciones que hacen de autoescuela como *Dribo*.

Sin embargo, no hemos encontrado aplicaciones que coincidan con los requisitos con los que cuenta Drivinglia o realicen un seguimiento detallado de las prácticas de conducir o tengan un registro de las faltas cometidas en el momento de la prueba.

1.3. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación móvil Android de tracking y seguimiento de los alumnos en autoescuela para la mejora de la enseñanza práctica en vías públicas. Para llevarlo a cabo, enumeramos una serie de objetivos más específicos en los que podemos dividir este objetivo principal más generalista:

O1. Análisis del dominio del problema y desarrollo de los mockups de la aplicación para determinar las interfaces de usuario y funcionalidad aproximada que podría tener la aplicación.

Intentaremos conseguir un diseño intuitivo y a la vez atractivo, en el que se pueda acceder a la información de la manera más cómoda posible.

O2. Familiarización con las tecnologías de trackeo en carretera y desarrollo de la parte de la aplicación que realiza el seguimiento de las prácticas.

Haremos uso de los criterios de calificación de la prueba de control de aptitudes y comportamientos en vías abiertas al tráfico general para establecer el listado de faltas que tiene que reconocer la aplicación.

O3. Diseño y programación de un sistema de perfiles e inicio de sesión acorde al alcance del proyecto.

Los perfiles de uso abordables serán los de alumno, profesor y administración. Cada uno de los perfiles tendrá funciones diferenciadas.

O4. Almacenamiento de los datos en la nube haciendo uso de la base de datos no relacional y en tiempo real Firebase.

Tendremos que implementar una estructura adecuada para manejar la información de las

prácticas que se realicen y de los distintos tipos de usuarios registrados. Para hacer uso de Firebase con Android Studio tendremos que importar las librerías de Google necesarias y añadir nuestra aplicación al proyecto Firebase.

O5. Puesta en marcha de un sistema de seguimiento de voz por manos libres opcional para el establecimiento de las faltas.

Este sistema será implementado con posterioridad para reducir en el mayor grado posible las distracciones que se puedan generar con el uso de Drivinglia. Utilizaremos las clases de reconocimiento de voz que nos proporciona Android [14].

1.4. Estructura del documento

La memoria del Trabajo de Fin de Grado está separada en 7 capítulos principales más la bibliografía, el presupuesto y el pliego de condiciones.

A continuación, se describen con brevedad cada uno de los apartados:

- **Capítulo 1: Introducción.** Capítulo introductorio en el que se presenta el estudio previo a la elaboración de la aplicación. También establece los objetivos iniciales del proyecto y detalla la estructura del documento.
- **Capítulo 2: Tecnologías software.** Se describen las tecnologías software, librerías para Android y servicios web de los que se han hecho uso.
- **Capítulo 3: Descripción general.** Esquema del planteamiento de la aplicación en esquemáticos y diagramas de bloques. También se explican los nuevos objetivos alcanzados a lo largo del desarrollo de la aplicación.
- **Capítulo 4: Diseño.** Se presenta el diseño previo a la elaboración del proyecto con los requisitos que debe cumplir la aplicación, el modelo de las pantallas de guía (mockups) y la estructura de la base de datos.
- **Capítulo 5: Implementación.** Se abarcan en detalle las consideraciones de la programación y el modelo de datos para llevar a cabo el proyecto software.

- **Capítulo 6: Guía de usuario.** Capítulo en el que se presenta una guía de uso de la aplicación con todas las funciones y características que ofrece a nivel de usuario (alumno, profesor y administrador).

- **Capítulo 7: Conclusiones.** Recopilación de las conclusiones alcanzadas en el trabajo de Fin de Grado tras la elaboración del proyecto software. Se incluyen objetivos cumplidos y líneas futuras en las que se proponen características y detalles que se podrían añadir posteriormente.

- **Bibliografía.** Referencias, páginas web y documentos consultados en la elaboración de la memoria del trabajo de Fin de Grado.

- **Presupuesto.** Estimación del coste aproximado de la elaboración del trabajo de Fin de Grado.

- **Pliego de condiciones.** Requisitos software y hardware necesarios en la elaboración del trabajo de Fin de Grado.

2. Tecnologías software

En este capítulo describiremos las tecnologías software, librerías para Android y servicios web de los que se han hecho uso.

2.1. Android Studio (v 4.1.2)

Ilustración 4: Android Studio

Entorno de desarrollo integrado (IDE) oficial para el especializado en desarrollo de aplicaciones Android. Está basado en IntelliJ IDEA y contiene múltiples funciones que aumentan la productividad del desarrollo software [15]. El logo lo podemos ver en la Ilustración 4.

2.2. Firebase

Ilustración 5: Firebase

Plataforma de desarrollo software en la nube propiedad de Google que nos ofrece la posibilidad de crear bases de datos en tiempo real no relacionales entre otras funciones. Firebase nos proporciona métodos y subrutinas que podemos utilizar en la programación para gestionar los distintos servicios que ofrece. El logo lo podemos ver en la Ilustración 5.

Para desarrollar Drivinglia, se ha hecho uso de los servicios de *Realtime Database* o base de datos en tiempo real para gestionar la información de usuario y *Firebase Storage* o almacenamiento online para guardar las imágenes de los distintos usuarios [16].

2.3. Trello

Ilustración 6: Trello

Trello es una aplicación con interfaz web que nos permite organizar el proyecto de manera flexible e interactiva. Nos da la posibilidad de crear tableros Kanban con tarjetas y etiquetas con los que podemos gestionar y analizar las tareas que tenemos pendientes [17]. El logo lo podemos ver en la Ilustración 6.

En la Ilustración 7, podemos ver una captura de pantalla del tablero Kanban de Drivinglia. Cuenta con distintas columnas dependiendo del progreso de las tareas que se van apuntando.

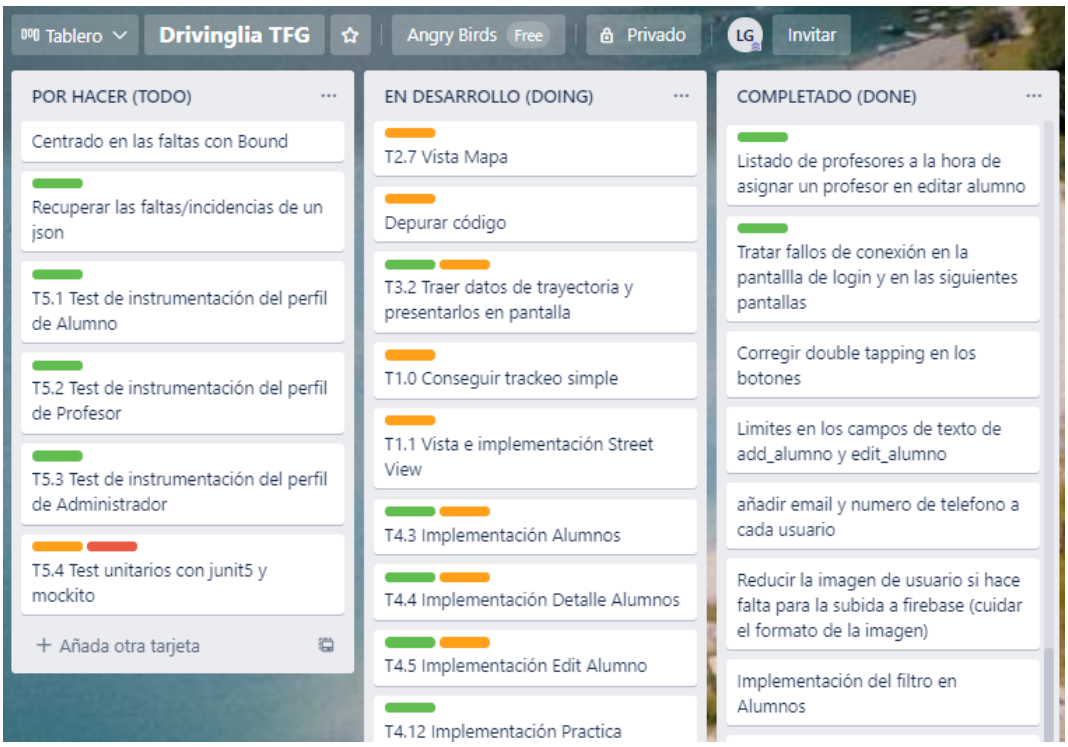


Ilustración 7: Tablero Kanban de Drivinglia

2.4. Figma

Ilustración 8: Figma

Figma es un software de edición gráfica con el que podemos diseñar el aspecto de las pantallas de nuestra aplicación (mockups). Ofrece una interfaz interactiva con una gran cantidad de herramientas y nos da la opción de visualizar la ejecución y composición de las distintas pantallas a modo de prototipo [18]. El logo lo podemos ver en la Ilustración 8.

2.5. Espresso

Ilustración 9: Espresso

Espresso es un framework que nos permite crear pruebas de la interfaz de usuario (IU) concisas, eficaces y confiables. También nos da la posibilidad de automatizar las pruebas desde Android Studio sobre un emulador o dispositivo virtual [19]. El logo lo podemos ver en la Ilustración 9.

2.6. GitHub

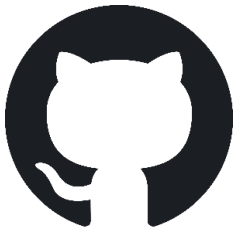


Ilustración 10: GitHub

GitHub es una plataforma online en la que podemos crear repositorios de nuestros proyectos. Un repositorio online nos permite gestionar el código que introducimos de forma organizada para mejorar la gestión de la aplicación y tener un respaldo fiable sobre la programación que se ha realizado [20]. El logo lo podemos ver en la Ilustración 10.

2.7. API

Una API o interfaz de programación de aplicaciones es un tipo de interfaz software capaz de ofrecer servicios en forma de subrutinas, funciones o métodos [21]. En esta ocasión, utilizaremos API de tipo REST (*Representational state transfer*) ya que obtenemos la información en formato JSON a través de peticiones http.

Las API REST empleadas han sido las siguientes:

2.7.1. Directions API (Google Maps)

Ilustración 11: Google Maps

Servicio web propiedad de Google que recoge peticiones HTTP y devuelve, en formato JSON o XML, la dirección entre puntos de ruta para distintos tipos de transporte como: conducción, tren, bicicleta o a pie [22]. El logo lo podemos ver en la Ilustración 11.

Hemos utilizado la API de direcciones de Google para realizar el trazado de las polilíneas del mapa en el perfil de profesor cuando se llevan a cabo prácticas de conducir.

2.7.2. MetaWeather API

Ilustración 12: MetaWeather

MetaWeather es un servicio automatizado de uso libre creado por Jason Cartwright. Colecciona información de múltiples servicios meteorológicos para determinar las predicciones del tiempo. Cuenta con una API que responde a peticiones HTTP (Hypertext Transfer Protocol) y devuelve datos en formato JSON sobre conexiones seguras HTTPS (Hypertext Transfer Protocol Secure) [23]. El logo lo podemos ver en la Ilustración 12.

Hemos utilizado la API de MetaWeather para obtener el id de la ciudad registrada más cercana y la obtención de las predicciones meteorológicas de la misma.

2.8. Librerías empleadas

Para llevar a cabo el proyecto Android, se han hecho uso de ciertas librerías externas que hemos importado para llevar a cabo funciones específicas.

2.8.1. Volley

Para realizar peticiones HTTP a las distintas API de las que hacemos uso en Drivinglia, utilizamos la librería Volley. Recomendada por Android, la librería HTTP Volley nos facilita y acelera las labores de networking para volúmenes de datos comedidos [24].

Las peticiones HTTP por Volley hacen uso de colas llamadas *RequestQueue* [25]

2.8.2. Gson

La librería Java *Gson* puede ser utilizada para convertir objetos Java a su correspondiente representación en formato JSON o convertir cadenas de caracteres JSON a su equivalente objeto Java [26].

Hemos hecho uso de esta librería para poder extraer la información del fichero de configuración JSON con la información de las incidencias de conducir que veremos en el apartado 5.3.

2.8.3. Android Image Cropper

Hemos hecho uso de la librería de recorte de imágenes *Android Image Cropper* para darle al administrador la posibilidad de seleccionar una imagen de la galería o realizar una foto y recortarla a la escala que definamos para los avatares al crear o modificar un profesor o alumno [27].

De esta forma, las imágenes de usuario siempre estarán a escala 1:1 y le damos al administrador la flexibilidad de poder elegir y recortar la imagen sin tener que recurrir a otras aplicaciones.

2.8.4. Google Play services

Haremos uso de las librerías *Google Maps SDK* y *Fused Location Provider* [28].

La librería de *Google Maps SDK* será utilizada para poder implementar los mapas de Google y la vista callejera o *Street View* en nuestra aplicación.

Por otro lado, la librería *Fused Locations Provider* nos permitirá acceder a la posición actual del dispositivo a través de forma periódica para realizar el tracking. Podemos especificar el nivel de precisión de las tomas, así como el intervalo de estas para ahorrar batería. El servicio accede a la posición a través del GPS del dispositivo o por triangulación de torres móviles y Wifi según las especificaciones de precisión que le indiquemos.

Hemos utilizado este método para calcular la posición y no otros como el propio de Android (*LocationManager)* debido a que es la forma recomendada por la documentación oficial de Android para recibir actualizaciones de posición cada cierto tiempo [29].

3. Descripción general

En este capítulo presentaremos el planteamiento de la aplicación de manera **generalizada** en forma de esquemáticos y diagramas. También se explicarán los nuevos objetivos propuestos a lo largo del desarrollo.

3.1. Funcionamiento

Para el desarrollo de la aplicación de Drivinglia se han diseñado 3 perfiles de uso: **administrador**, **profesor** y **alumno.** Cada perfil tiene funciones diferenciadas y para acceder a cada uno de ellos, primero hay que realizar un inicio de sesión con las credenciales del usuario en cuestión.

3.1.1. Inicio de sesión

Para realizar el inicio de sesión, la aplicación accede a los usuarios de la base de datos en tiempo real de **Firebase Real-time Database** y valida las credenciales introducidas (id o dni y contraseña). Si las credenciales coinciden, la aplicación entrará en la pantalla del perfil, en la que se accederá a **Firebase Storage** para descargar la imagen de usuario en cuestión. La contraseña ha de ser decodificada primero bajo el estándar **UTF_8** [30]. En la Figura 1, podemos ver un diagrama de bloques que ilustra la ejecución del inicio de sesión.

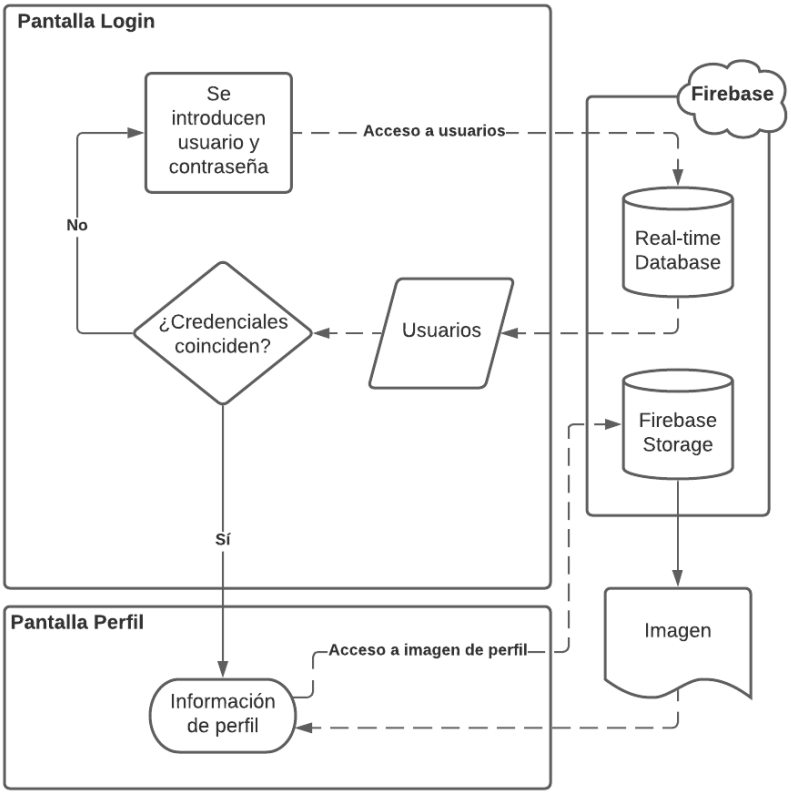


Figura 1: Diagrama de inicio de sesión

3.1.2. Perfil de Administrador

Para el perfil de administrador, la temática será el color **rojo**. Una vez el usuario administrador haya iniciado sesión, podrá navegar por las listas de alumnos y profesores. Para las listas de alumnos, para facilitar la navegación y mejorar la claridad, se da la posibilidad de ordenar los alumnos alfabéticamente o por id así como de filtrarlos en función de los permisos de conducir o si estuvieran o no listos para examen.

El administrador podrá editar o crear nuevos perfiles de alumno o profesor, así como asignar alumnos a los distintos profesores de manera que en el perfil del profesor aparezcan solamente los alumnos asignados por el administrador.

Si el administrador crea un nuevo usuario, se abrirá la aplicación de mensajería que tenga instalado el dispositivo para poder enviar un mensaje personalizado al correo electrónico de dicho usuario con la información de sus credenciales de inicio de sesión. Como mencionamos anteriormente, la contraseña en cuestión ha de ser cifrada con **UTF_8** antes de su escritura en la base de datos. En la Figura 2, podemos ver el esquemático del perfil de administrador.

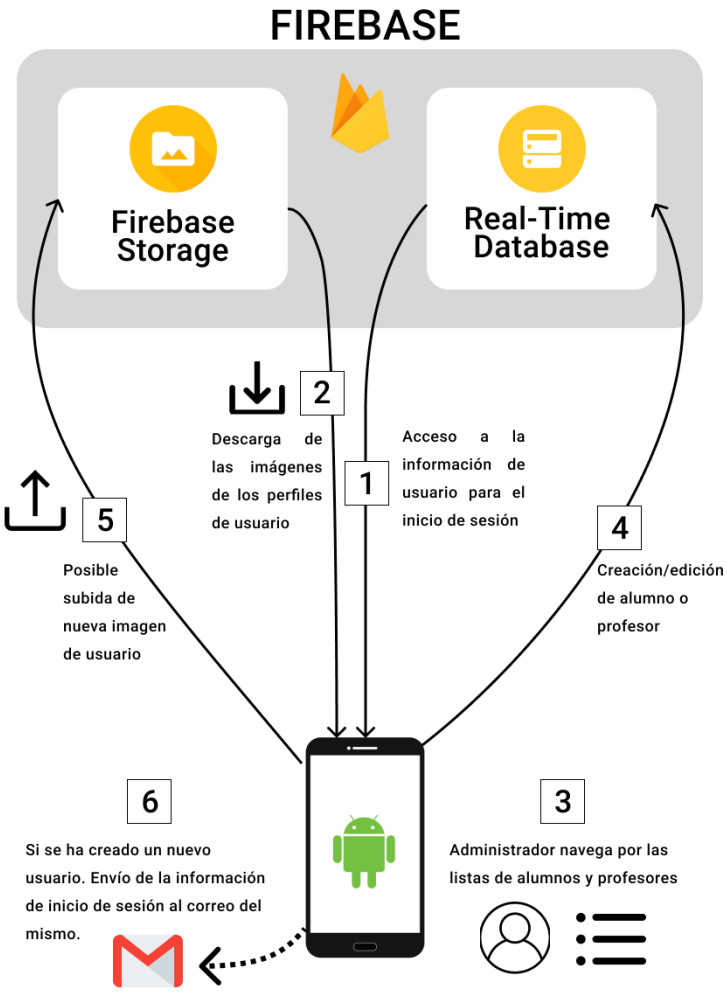


Figura 2: Esquemático del perfil de Administrador

En la Figura 3, podemos ver un diagrama de bloques que ilustra la ejecución de las pantallas del perfil de administrador.

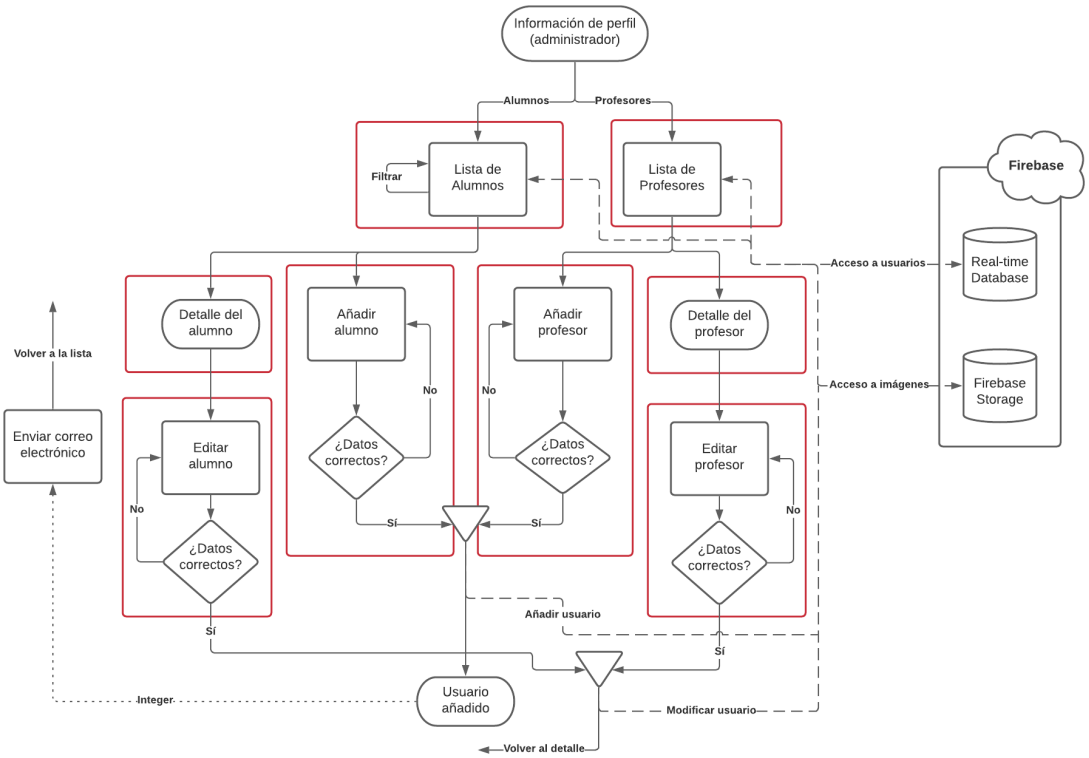


Figura 3: Diagrama del perfil de administrador

3.1.3. Perfil de Profesor

La temática del perfil de profesor es el color **azul**. Dicho perfil puede acceder a las listas de alumnos que tenga asignados para comenzar las prácticas de conducir pertinentes.

En el desarrollo de la práctica, la aplicación accederá a la API de **Google Maps Directions** cada 100 metros en movimiento para dibujar el trazado del recorrido en el mapa. También hará uso de la API de **MetaWeather** para recoger la predicción del tiempo.

Durante la prueba, el profesor podrá señalar faltas de conducción que se añadirán en el mapa. La información de dichas faltas se recoge de un archivo **JSON** local con las directrices de la DGT (Dirección General de Tráfico) para calificaciones en vías abiertas al tráfico general.

Una vez finalizada la práctica, el profesor podrá añadir anotaciones de seguimiento y señalar al alumno como listo o no para examen si así lo desea. En la Figura 4, podemos ver el esquemático del perfil de profesor.

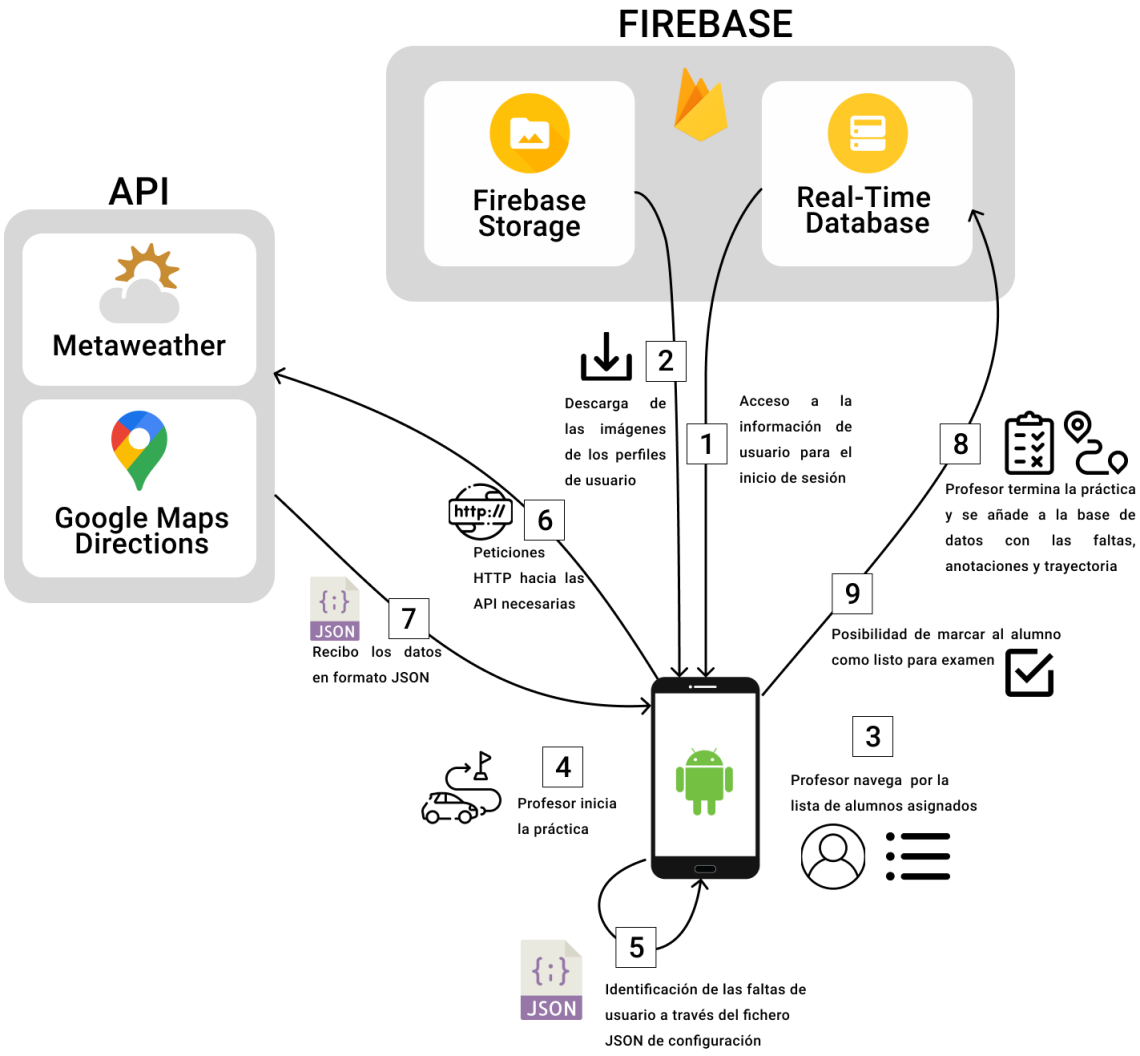


Figura 4: Esquemático del perfil de Profesor

En la Figura 5, podemos ver un diagrama de bloques que ilustra la ejecución de las pantallas del perfil de profesor.

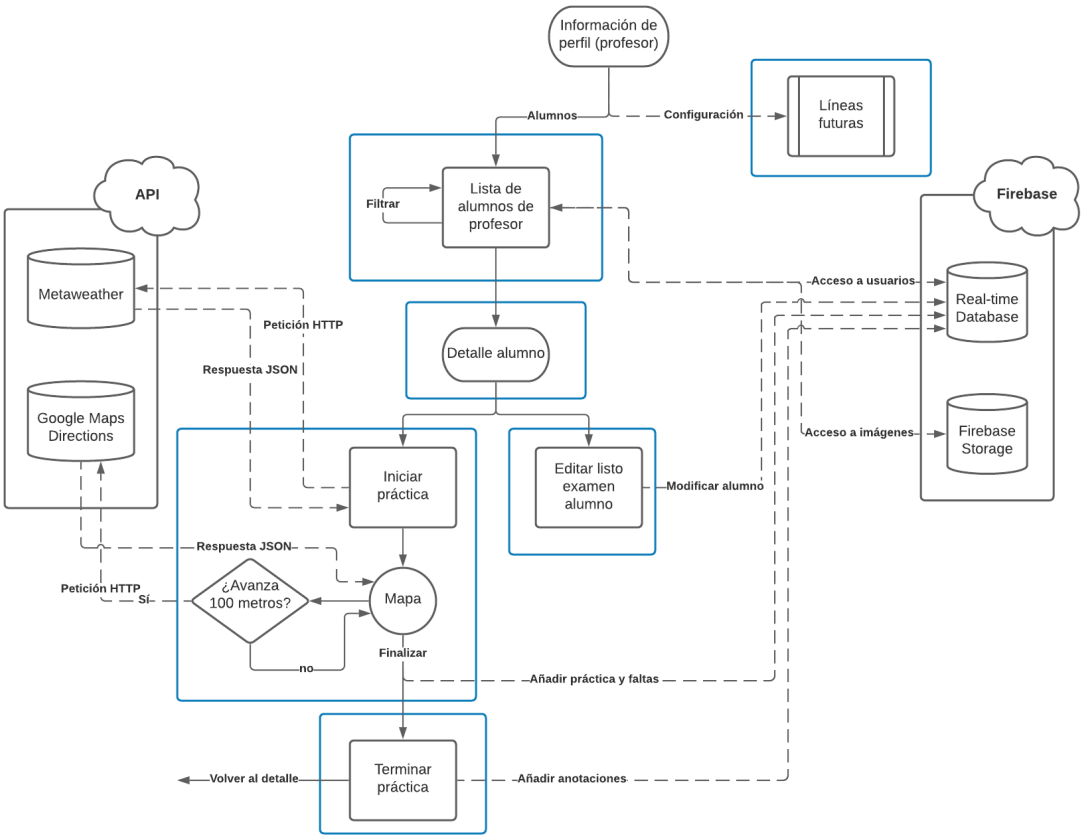


Figura 5: Diagrama del perfil de Profesor

3.1.4. Perfil de Alumno

La temática del perfil de alumno es el **verde**. Dicho perfil puede acceder a la lista de prácticas que tenga registrada en la base de datos, así como analizar las distintas faltas que haya cometido.

Las distintas prácticas que haya realizado se presentarán en forma de listas. Para cada una de las prácticas, el alumno tendrá la opción de comprobar las anotaciones y puntos de mejora, así como acceder al mapeado del recorrido realizado y las faltas cometidas.

Al acceder al compendio de faltas acumuladas, el alumno podrá ver el motivo y gravedad de cada una de ellas y compartir los resultados con quien quiera en forma de porcentajes a través de otras aplicaciones. En la Figura 6, podemos ver el esquemático del perfil de alumno.

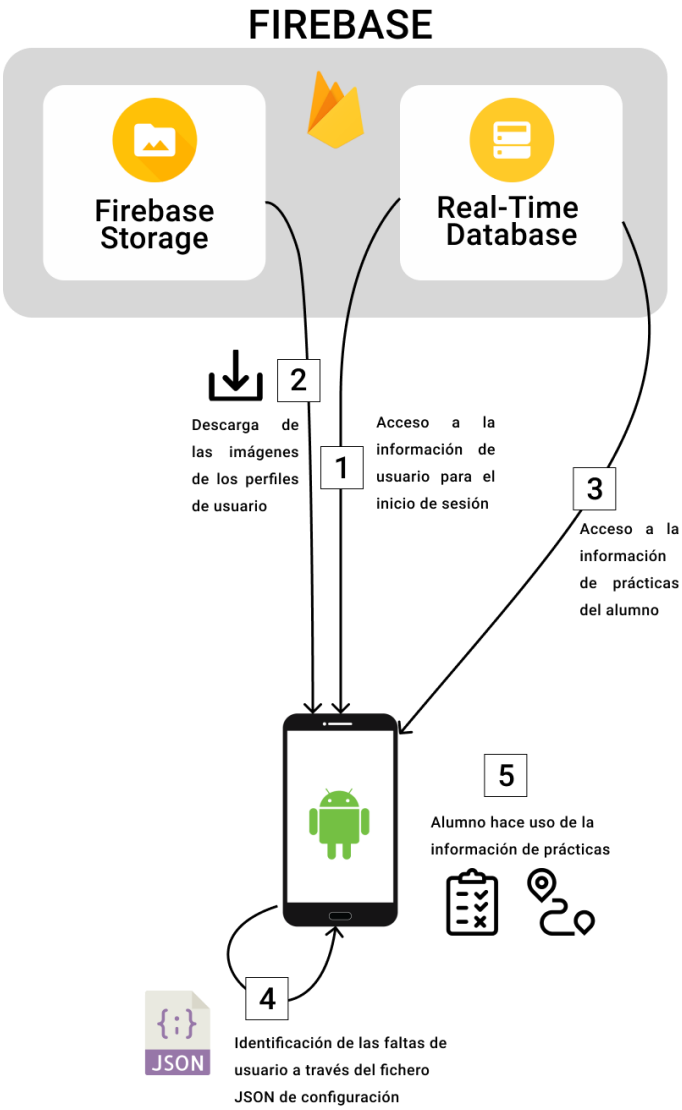


Figura 6: Esquemático del perfil de Alumno

En la Figura 7, podemos ver un diagrama de bloques que ilustra la ejecución de las pantallas del perfil de alumno.

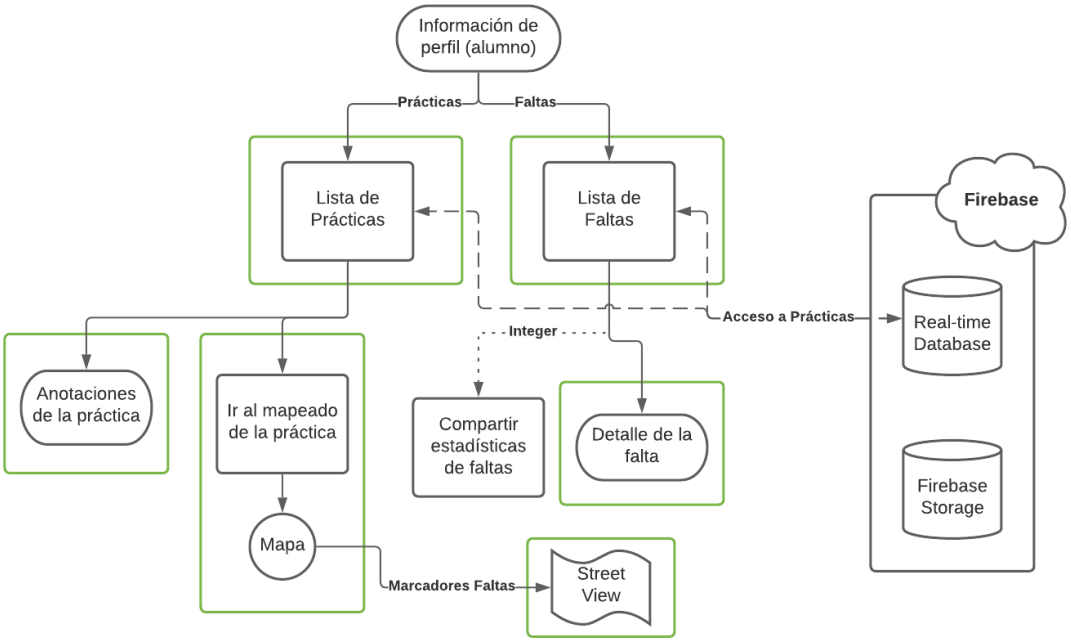


Figura 7: Diagrama del perfil de Alumno

3.2. Nuevos objetivos propuestos

A lo largo del desarrollo del proyecto hemos propuesto una serie de **objetivos adicionales** que enriquecerían la aplicación. Estos objetivos han sido los siguientes:

OA1. Almacenamiento y descarga de las imágenes de usuario desde la aplicación haciendo uso de la base de datos *Firebase Database* e implementación de un sistema de gestión de usuarios acorde.

Hemos logrado implementar un sistema en el que el administrador pueda editar y añadir imágenes de perfil e información de los usuarios de manera cómoda. Estos cambios se ven reflejados en las distintas pantallas de visualización de alumnos y profesores de las que dispone la aplicación.

OA2. Aplicación de las distintas herramientas de las que disponen los mapas de Google como pueden ser el Street View o la personalización del estilo del mapa, polilíneas y marcadores.

A lo largo del desarrollo del proyecto software hemos ido implementando un número de elementos que nos proporcionan los mapas de Google considerable y hemos conseguido personalizarlos a un nivel acorde a la aplicación. A su vez, hemos conseguido implementar el *Street View* o vista callejera para que el alumno pueda visualizar en un plano más cercano el lugar en el que cometió la infracción.

OA3. Implementar un sistema de acceso a las condiciones meteorológicas.

Hemos hecho uso de la API de *Metaweather* para determinar el id y las condiciones meteorológicas de la ciudad registrada en el servicio más cercana a la posición del dispositivo.

OA4. Diseño y programación de pantallas de visualización del compendio de faltas del alumno.

Conseguimos desarrollar una serie de pantallas en las que el alumno puede acceder a la información de las faltas que tenga acumuladas. También es posible visualizar el detalle de estas y compartir la gravedad de las faltas cometidas porcentualmente en función del número de prácticas realizadas.

OA5. Implementar mensajes de puntos de mejora automáticos en función de las faltas cometidas y realizar un balance de faltas en cada práctica.

El alumno puede acceder a las notas de cada práctica para encontrarse mensajes con los distintos puntos en los que debe mejorar en vez del motivo de las faltas y así no desalentarse. También encontrará un balance de faltas en cada práctica con el que puede comprobar si ha resultado apto o no en función de los criterios de calificación.

4. Diseño

En este capítulo vamos a analizar las decisiones tomadas en la etapa de diseño de la aplicación. Desde los requisitos a implementar hasta los elementos con los que debe contar cada una de las pantallas de la aplicación, así como la estructura de la base de datos y el patrón arquitectónico a seguir.

4.1. Escala del proyecto

Drivinglia será desarrollada exclusivamente para dispositivos Android. Por lo tanto, debido a la experiencia que tenemos con el lenguaje y para no comprometer el periodo de desarrollo de la aplicación será programada nativamente en Java.

Esta opción en el diseño nos permitirá:

- Ser más consistentes en la programación.
- Poder implementar funciones exclusivas de Android de forma eficaz.
- Limitar las excepciones y reducir el periodo de pruebas.
- Sacarle el máximo rendimiento al dispositivo.
- Mejorar la experiencia de usuario.

4.2. Requisitos

Un requisito es una función que ha de cumplir la aplicación. En el diseño de Drivinglia hicimos una lista con todos los requisitos que debería tener la aplicación y desestimamos algunas ideas que no llegaron a cuajar o que no tenían una prioridad suficientemente alta como para implementarlos.

Los requisitos con los que debía cumplir la aplicación al realizar el diseño de esta se dividieron en función del perfil de uso. A continuación, mostramos los requisitos a cumplir por cada uno de los perfiles en el orden en el que han sido pensados:

(Los requisitos desestimados, no implementados o postergados para líneas futuras aparecerán tachados)

4.2.1. Perfil Alumno

- Accede a su cuenta mediante código y contraseña
- Listado de las prácticas que ha hecho, el profesor y la fecha

- Interacción con el esquema o mapa de la práctica con las distintas infracciones anotadas, con colores para determinar si infracción leve (verde) grave (naranja) o eliminatoria (roja) y un código asociado, así el nombre de dicha infracción.
- Notas generales en cada práctica acerca de la evolución
- Indicativo en la práctica si está listo para examen o no.
- Mensajes automáticos de puntos en los que mejorar en función del número de incidencias, su tipo, o el momento o situación en la que ocurren en cada práctica (hora punta de alto tránsito, nervios al principio de la práctica, miras pocas veces al retrovisor, te sueles saltar muchos stops, te falta mejorar el manejo del coche…etc)
- ~~En el listado de prácticas se muestra también el permiso de conducir al que corresponde~~
- ~~Prácticas especiales que son simulacros de exámenes. Dichos simulacros tendrán ciertas diferencias en el color de la celda en la lista o se podrían poner en un listado aparte. Tienen un indicativo de apto o no apto y en el detalle todos los errores cometidos.~~
- Estadística general con número de faltas de un tipo u otro y si son graves, leves o eliminatorias.

4.2.2. Perfil Profesor

- Accede a su cuenta mediante código y contraseña
- Listado con los alumnos que tenga prácticas
- Para generar una práctica introduce el código del alumno o simplemente va a su detalle del listado y tiene un botón de iniciar práctica ~~y otro de iniciar simulacro de examen~~.
- En el detalle del alumno aparecerá información (nombre, apellidos, código, imagen, fecha de nacimiento)
- Al iniciar una práctica, el profesor apoya su móvil en un dock o soporte del coche y empezará el trackeo automáticamente de la posición del vehículo.
- En el trackeo aparecerán 3 botones que pueden estar ya en pantalla para su fácil alcance e indican la gravedad de la falta (rojo es eliminatoria, naranja es deficiente y verde es leve)
    - Cuando el profesor selecciona uno de los 3 botones se registrará en el mapa, en la posición donde se encuentra el vehículo una huella o marca indicando que en ese tramo se cometió una infracción y un color dependiendo de la gravedad. Entonces se abrirá un listado con los códigos de infracción y deberá de seleccionarse uno de ellos ~~o simplemente no seleccionar nada y cerrar el desplegable o seleccionar la opción de no definir~~.

  - Cuando el profesor selecciona uno de los 3 botones (rojo, naranja o amarillo), a parte del listado con las infracciones, tendrá opción de pulsar un botón con forma de micrófono. En ese momento, el profesor podrá decir el código de la infracción en voz alta de modo que se quede registrada automáticamente y no tenga que distraerse buscando en el listado. En pantalla aparecerá el código interpretado ~~(o lo puede reproducir el altavoz así evitamos distracciones)~~, si la interpretación fuera errónea podrá borrarlo y seguir buscando en el listado o aceptarlo o simplemente no hacer nada y se aceptará automáticamente.
- En el transcurso de la práctica aparecerá un timer con el número de minutos y horas que han transcurrido. De modo que el profesor sepa el tiempo de práctica.
- Habrá un modo noche para el mapa de la práctica.
- ~~Aparecerá el número de veces que se ha pasado por una calle (número o indicativo de colores en el dibujo del trackeo), ya que normalmente se suele recorrer el mismo lugar.~~
- Al final de la práctica, el profesor pulsará el botón de fin de práctica para ver el balance de esta.
- Al final de la práctica, el profesor puede añadir anotaciones escritas. Habrá mensajes predefinidos de puntos de mejora en función de las infracciones y el profesor podrá señalar si el alumno está o no listo para examen al volver.
- Si el alumno está listo para examen se añadirá a una lista de alumnos listos para examen a la que tendrá acceso el profesor y en el detalle de dicho alumno aparecerá dicha marca también.
- En el transcurso de la práctica, los puntos de infracción del mapa de trackeo se pueden modificar o eliminar.
- Al final de la práctica aparece el tiempo de la práctica, las faltas cometidas y puntos de mejora, ~~el municipio~~, las condiciones medioambientales ~~(se podrían especificar por el profesor al final de la práctica).~~
- ~~Con la opción de compartir práctica tenemos varias opciones como enviar por correo electrónico o generar un código QR para que el alumno lo lea.~~
- Con la opción de subir práctica, la práctica se subirá a la base de datos Firebase u otra base de datos en la nube con el código del alumno y se borrará del móvil del profesor. El alumno, al entrar a su aplicación rescatará los datos de dicha práctica automáticamente al estar referenciada con su código. Tanto el mapa con los puntos de infracción como las anotaciones aparecerán en su listado de prácticas con la fecha asociada.

- ~~Opcionalmente se podrían almacenar las prácticas en el móvil del profesor y no borrarlas automáticamente para que las pueda modificar a posteriori o simplemente rellenar más tarde y no en el momento. Cuando quiera puede eliminarlas con un swipe para liberar memoria.~~

4.2.3. Perfil Administrador

- Accede a su cuenta mediante código y contraseña
- Listado de alumnos con la posibilidad de filtrar por permiso y ordenar alfabéticamente.
- ~~El listado de alumnos también cuenta con un buscador por nombre.~~
- Listado de profesores de prácticas.
- En el detalle de los alumnos, aparece su información y posibilidad de editarla o eliminar dicho alumno. También aparece el profesor de prácticas asociado que podemos también cambiar.
- En el detalle de los profesores de prácticas aparece su información, la posibilidad de editar dicha información o eliminar dicho profesor y listado de alumnos que tiene asociados de prácticas.
- Posibilidad de añadir alumnos a la base de datos.
- Posibilidad de añadir profesores de prácticas a la base de datos.
- Al añadir un profesor o alumno a la base de datos, se enviará un correo electrónico personalizado al mismo con la información de inicio de sesión.
- Los usuarios creados podrán iniciar sesión tanto con el id como con el dni.

4.3. Mockups

En el proceso de diseño de la aplicación intentamos conseguir unas pantallas lo más intuitivas posible con multitud de elementos interactivos que enriquezcan la aplicación a nivel de usuario.

Las pantallas que veremos a continuación se han diseñado con la herramienta Figma mencionada en el capítulo 2.4. Además, gran parte de los iconos de los que hacemos uso tanto en los mockups como en las pantallas finales han sido extraídos de la página web de iconos de libre uso *Flaticon* [31].

Pantalla de inicio de sesión

Pantalla principal de la aplicación en la que el usuario introducirá sus credenciales. No tendrá barra de herramientas y contará con un botón de iniciar sesión. En el momento de la validación de las credenciales aparecerá una barra de progreso. Podemos ver el mockup de la pantalla de inicio de sesión en la Figura 8.

Figura 8. Mockup: Inicio de sesión

Pantallas de perfil

La pantalla de perfil será la que le aparecerá al usuario una vez haya iniciado sesión. Contiene su nombre, avatar y 3 botones. El primer y segundo botón dependen del rol del usuario y el tercero cierra la sesión. En la Figura 9, Figura 10 y Figura 11, podemos ver los mockups de las pantallas de perfil de administrador, profesor y alumno.

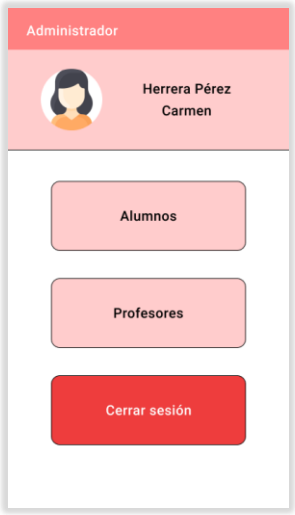


Figura 9: Mockup: Perfil Administrador

Figura 10. Mockup: Perfil Profesor

Figura 11. Mockup: Perfil Alumno

Si el usuario pulsa el botón de cerrar sesión aparecerá una ventana emergente de confirmación como la que podemos ver en la Figura 12.

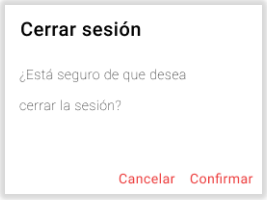


Figura 12. Mockup: Cerrar sesión

A continuación, mostraremos las pantallas que son exclusivas de cada perfil de uso:

4.3.1. Administrador

Las pantallas de administrador tendrán como temática el color rojo.

Pantalla de listado de alumnos

Mostrará una lista con los alumnos de la base de datos. Para cada alumno se presentará el nombre, id asociado, permiso de conducir del que está matriculado y avatar. Además, a aquellos alumnos que estén listos para examen les aparecerá adicionalmente un icono de un volante. En la parte inferior derecha habrá un botón circular que podemos pulsar para añadir nuevos alumnos. Podemos ver el mockup del listado de alumnos en la Figura 13.

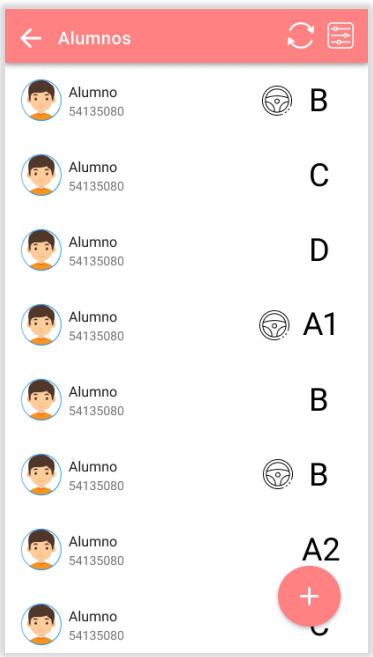


Figura 13. Mockup: Listado de alumnos

En la barra de herramientas habrá un botón de filtrado que nos abrirá una ventana emergente con la que podremos ordenar y filtrar los alumnos de la lista como la que podemos ver en la Figura 14, así como otro botón para recargar el listado.

Figura 14. Mockup: Ventana de filtrar y ordenar

Pantalla de listado de profesores

Muy similar a la del listado de alumnos, pero de profesores y sin la opción de filtrar ni ordenar. Tampoco aparecerán ni los iconos de listo para examen ni el permiso de conducir. Podemos ver el mockup del listado de profesores en la Figura 15.

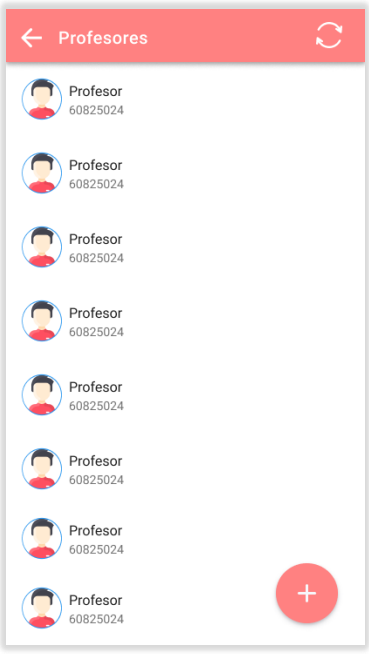


Figura 15. Mockup: Listado de profesores

Pantallas de añadir alumno y profesor

Pantallas similares entre sí en las que el administrador introducirá la información de

alumno o profesor. En el caso de la pantalla de añadir alumno, aparecerá también un botón para asignar profesor de prácticas que desplegará una lista de profesores a elegir. En la Figura 16 y Figura 17, podemos ver los mockups de las pantallas de añadir alumno y profesor.

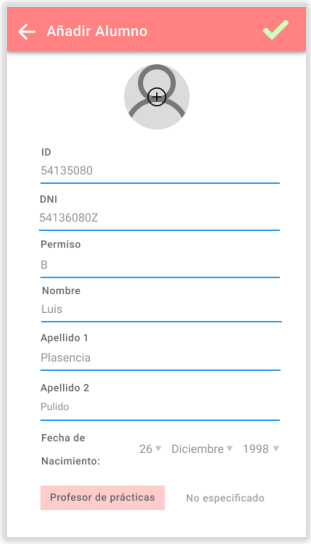


Figura 16. Mockup: Añadir alumno

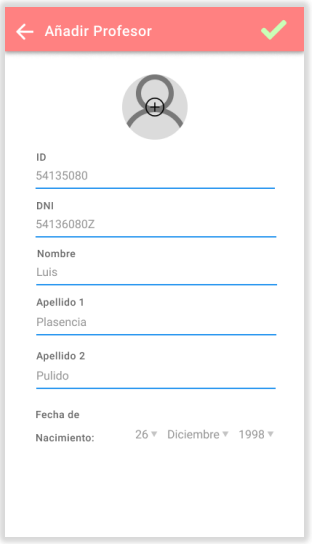


Figura 17. Mockup: Añadir profesor

Pantallas del detalle de alumno y profesor

Pantallas similares entre sí que muestran la información de alumno o profesor. En el caso del detalle del profesor habrá un botón de alumnos de prácticas que desplegará una lista de alumnos asignados. Ambas pantallas cuentan con un botón en la barra de herramientas para editar la información de dicho usuario. En la Figura 18 y Figura 19, podemos ver los mockups de las pantallas del detalle de alumno y profesor.

Figura 18. Mockup: Detalle del alumno

Figura 19. Mockup: Detalle del profesor

Pantallas de editar alumno y profesor

Estarán encargadas de editar la información de alumno o profesor. Contarán con un botón para eliminar dicho usuario y si queremos que los cambios se guarden tendremos que pulsar el botón de confirmación de la barra de herramientas.

Al igual que en la pantalla de añadir alumno, la pantalla de editar alumno contará con un botón para elegir el profesor de prácticas asignado. En la Figura 20 y Figura 21, podemos ver los mockups de las pantallas de editar alumno y profesor.

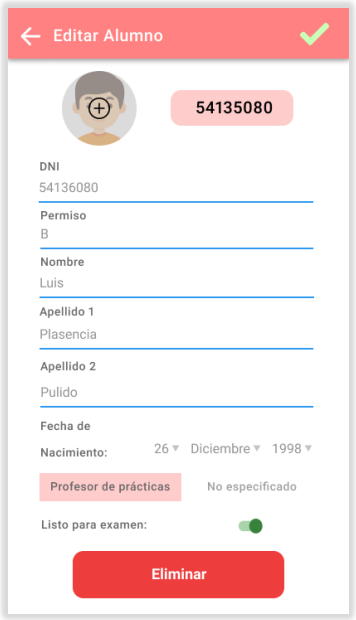


Figura 20: Mockup: Editar alumno

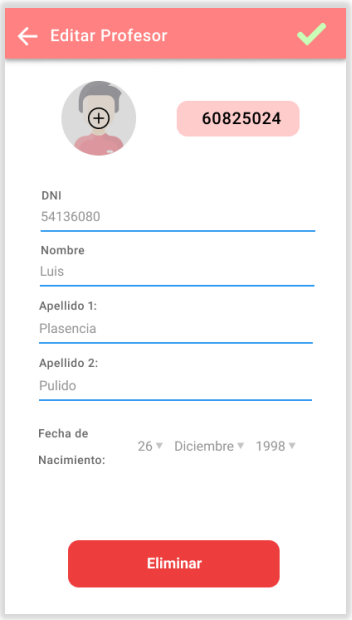


Figura 21: Mockup: Editar profesor

Si el administrador presiona el botón de eliminar alumno o profesor aparecerá una ventana emergente de confirmación similar a la de la Figura 22 y Figura 23.

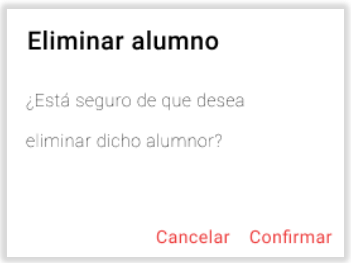


Figura 22: Mockup: Eliminar Alumno

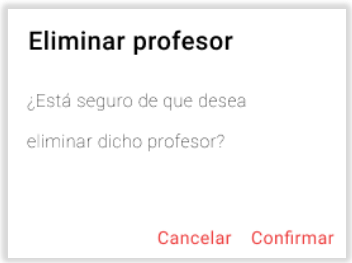


Figura 23: Mockup: Eliminar Profesor

Storyboard de los mockups del perfil de administrador

Podemos ver el storyboard de los mockups del perfil de administrador en la Figura 24.

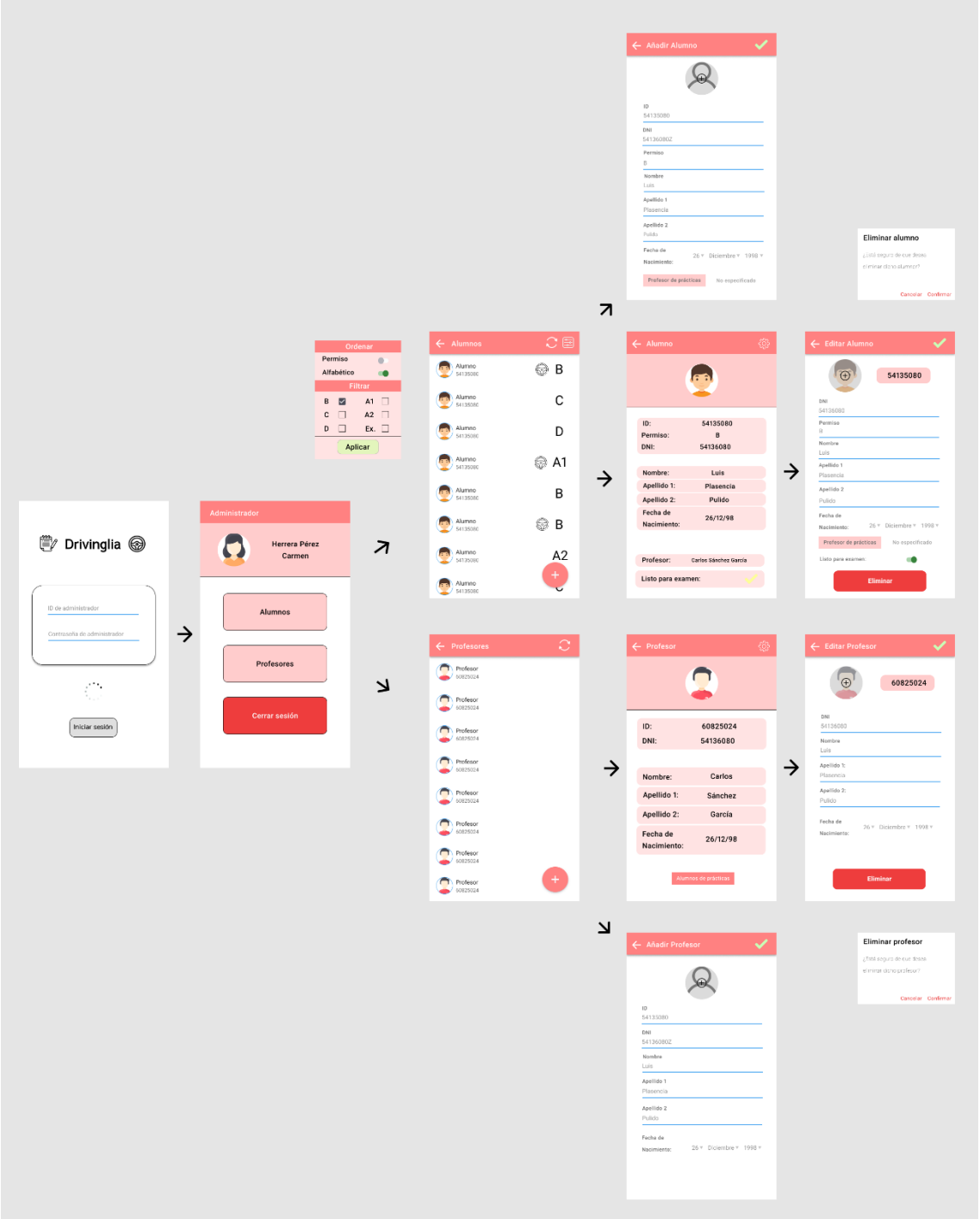


Figura 24. Mockups: Storyboard Administrador

4.3.2. Profesor

Las pantallas de alumno tendrán como temática el color azul.

Pantalla de configuración

Pantalla en la que el profesor podrá seleccionar sus preferencias de configuración.

Esta pantalla no se llegó a aplicar debido a que se relegó a líneas futuras. En la Figura 25, podemos ver el mockup que se había diseñado de la pantalla de configuración del perfil de profesor.

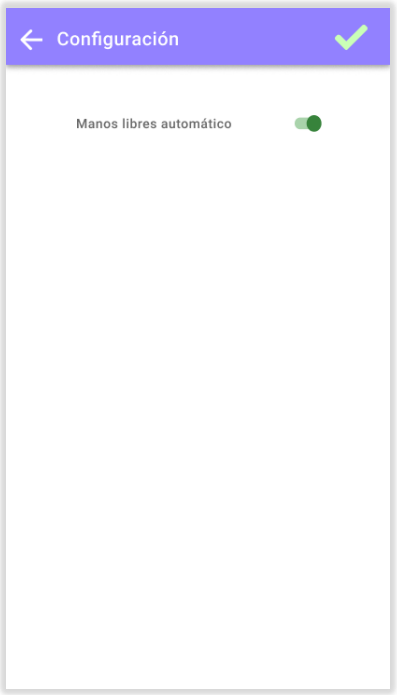


Figura 25: Mockup: Configuración

Pantalla de listado de alumnos asignados

Muy similar a la del listado de alumnos del perfil de administrador, como podemos ver en la Figura 26, pero solamente aparecerán los alumnos que tenga asignados dicho profesor y no aparecerá el botón de añadir alumno. Sí dispondrá del botón de actualizar el listado, así como del botón de desplegar la ventana emergente de filtrar y ordenar que vemos en la Figura 27.

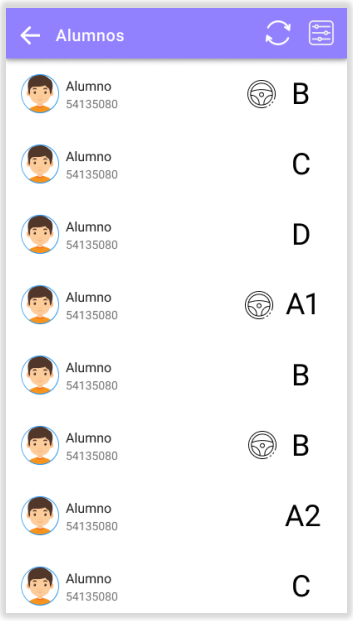


Figura 26. Mockup: Alumnos asignados

Figura 27. Mockup: Ventana de filtrar y ordenar

Pantalla del detalle de alumno asignado

Muy similar a la del detalle de alumnos del perfil de administrador, como podemos ver en la Figura 28, pero con la adición de dos botones extras para el comienzo de la práctica o examen. En este caso, el botón de la barra de herramientas nos dirigirá a una pantalla en la que podremos modificar ciertos campos de interés del alumno.

Figura 28. Mockup: Detalle del alumno asignado

Pantalla de configuración de alumno asignado

Pantalla, cuyo mockup podemos ver en la Figura 29, en la que el profesor puede modificar ciertos campos de interés del alumno. En principio solo podrá marcar al alumno como listo o no listo para examen.

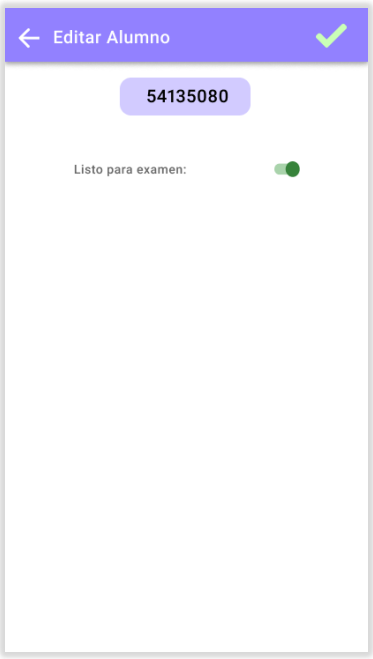


Figura 29: Mockup: Configuración de alumno asignado

Pantallas de inicio de practica e inicio de examen

Pantalla previa al inicio de la práctica o examen donde el profesor introduciría la información de esta, como podemos ver en los mockups de la Figura 30 y Figura 31.

Esta pantalla no llegó a ser implementada debido a que la información previa a la práctica la generaría la aplicación automáticamente. De esta forma, las prácticas se iniciarían directamente desde el detalle del alumno.

Figura 30. Mockup: Inicio de práctica

Figura 31. Mockup: Inicio de examen

Pantalla de la práctica

Las prácticas se llevarán a cabo en esta pantalla. Principalmente contará con 3 botones grandes y visibles para señalar la gravedad de la falta cometida, como podemos ver en el mockup de la Figura 32. En el momento en el que el profesor indica una falta se sobrepondrá un listado con los códigos de falta a especificar y un botón con forma de micrófono que podrá usar en caso de que quiera decirlo en voz alta, como podemos ver en el mockup de la Figura 33.

También cuenta con el tiempo de práctica en la parte superior y un botón para finalizar la práctica en la barra de herramientas. La posición actual estará señalizada en el mapa en todo momento y el mapa se centrará sobre la misma cada cierto tiempo.

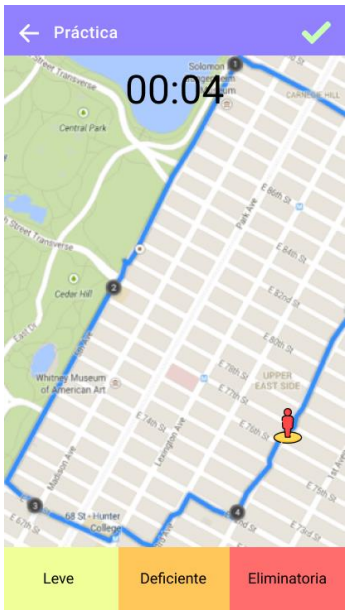


Figura 32. Mockup: Práctica

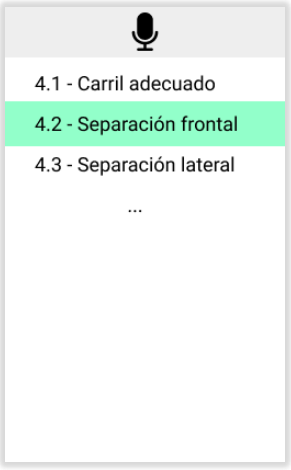


Figura 33. Mockup: Ventana de listado de faltas

Antes de finalizar o cancelar una práctica aparecerán ventanas emergentes de confirmación como las de la Figura 34 y Figura 35.

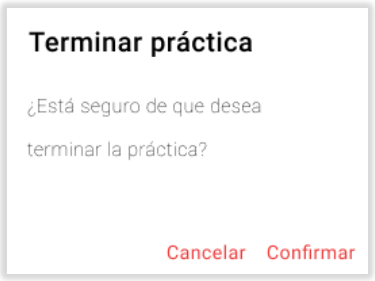


Figura 34: Mockup: Finalizar práctica

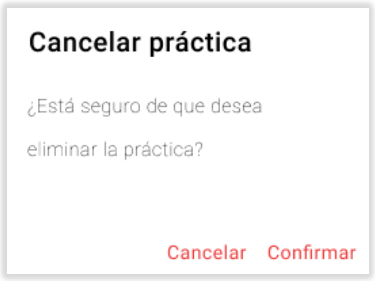


Figura 35. Mockup: Cancelar práctica

Pantalla del resultado de la práctica

Una vez se finalice una práctica, aparecerá la pantalla del resultado de práctica como la que podemos ver en el mockup de la Figura 36. En ella, el profesor podrá comprobar los mensajes de puntos de mejora que ha generado la aplicación automáticamente en función de las infracciones cometidas y podrá escribir anotaciones. Esta información le aparecerá al alumno en el detalle de la práctica.

Hay que comentar que la opción de elegir las condiciones meteorológicas no fue implementada debido a que esta información la genera la aplicación automáticamente a lo

largo de la práctica, no el profesor. En su lugar se incluyó un balance de faltas con las calificaciones de la práctica.

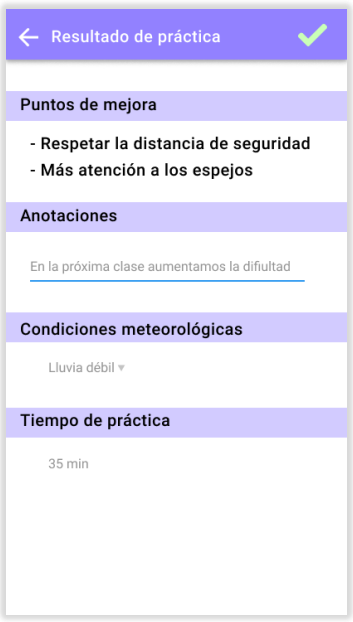


Figura 36. Mockup: Resultado de la práctica

Storyboard de los mockups del perfil de Profesor

Podemos ver el storyboard de los mockups del perfil de profesor en la Figura 37.

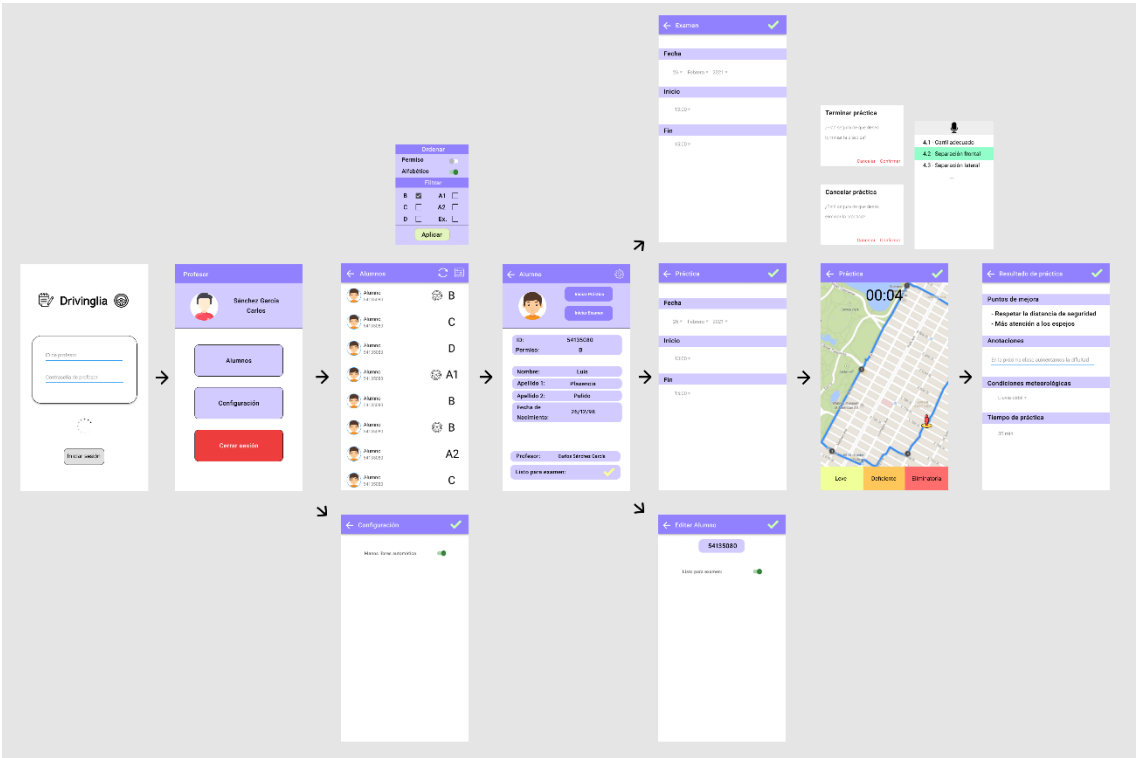


Figura 37. Mockups: Storyboard Profesor

4.3.3. Alumno

Las pantallas de alumno tendrán como temática el color verde.

Pantalla de faltas totales

El alumno podrá consultar el compendio de faltas totales que ha cometido a lo largo de las prácticas en forma de lista desplegable. Para cada infracción se indicará la gravedad, el código y el nombre. En la barra de herramientas superior habrá un botón de compartir. El mockup de la pantalla de faltas totales lo podemos ver en la Figura 38.

Figura 38: Mockup: Faltas totales

Pantalla del detalle de las faltas

Si en la pantalla anterior pulsamos alguna de las faltas nos aparecerá esta pantalla, cuyo mockup podemos ver en la Figura 39. Nos indicará el detalle de la falta en cuestión con una descripción del motivo y la sección o apartado al que pertenece.

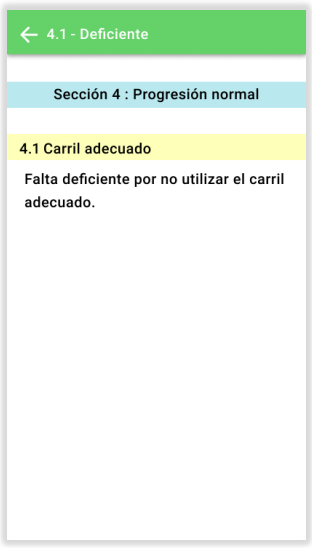


Figura 39. Mockup: Detalle de las faltas

Pantalla de prácticas

El alumno podrá consultar las distintas prácticas realizadas en un formato de lista. Para cada práctica podrá distinguir las fechas en las que se realizaron, el tiempo y el profesor asignado. A su vez, cada práctica contará con dos botones: uno accederá al mapa de la práctica y el otro a las notas o calificaciones de esta. En la barra de herramientas habrá un botón de actualizar similar al del listado de alumnos. El mockup de la pantalla de prácticas lo podemos ver en la Figura 40.

Hay que comentar que el botón de compartir práctica se terminó desestimando en la versión final.

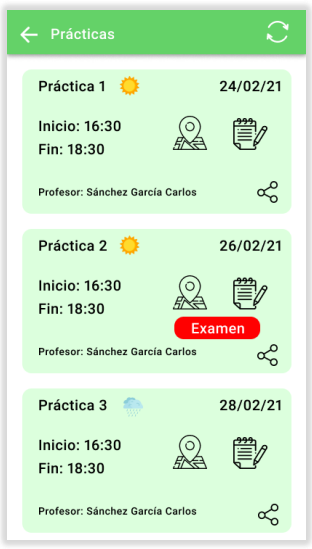


Figura 40. Mockup: Prácticas

Pantalla de notas de la práctica

El alumno podrá comprobar en esta pantalla tanto las anotaciones apuntadas por el profesor al final de la práctica como los puntos de mejora o la duración de esta. El mockup de la pantalla de notas de la práctica lo podemos ver en la Figura 41.

Hay que comentar que en la versión final también se incluye el balance de faltas y las calificaciones.

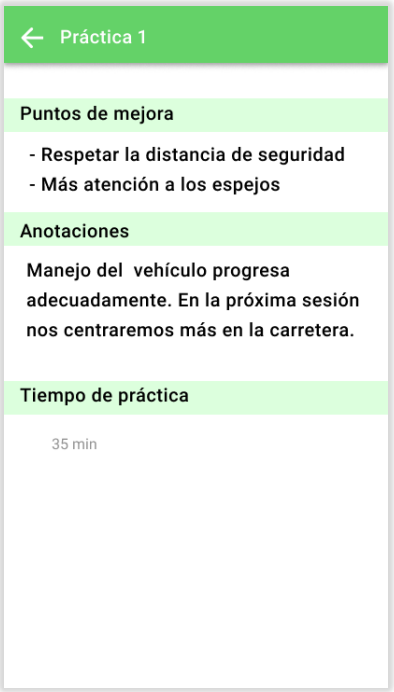


Figura 41. Mockup: Notas de la práctica

Pantalla de mapa de la práctica

Pantalla en la que el alumno podrá comprobar la trayectoria seguida por el vehículo en la práctica, así como las infracciones cometidas en forma de marcadores, como podemos ver en el mockup de la Figura 42.

En la barra de herramientas superior hay un botón para desplegar un listado expandible lateral con las faltas cometidas similar al del mockup de la Figura 43. Este listado nos servirá de navegación ya que, al pulsar en alguna de las faltas, el mapa se nos centrará en ella. Adicionalmente, si pulsamos sobre el detalle de una falta se nos mostrará información extra y podremos acceder al Street View.

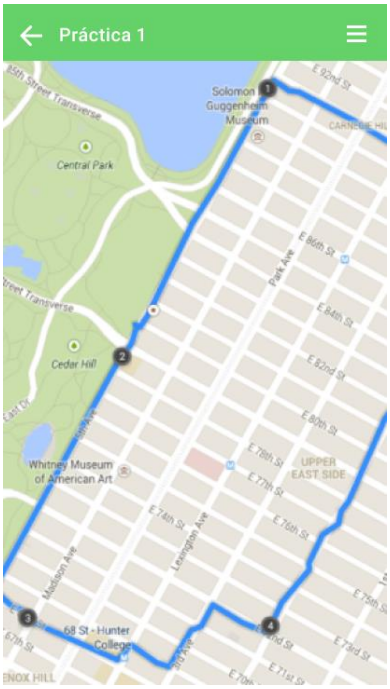


Figura 42. Mockup: Mapa de la práctica

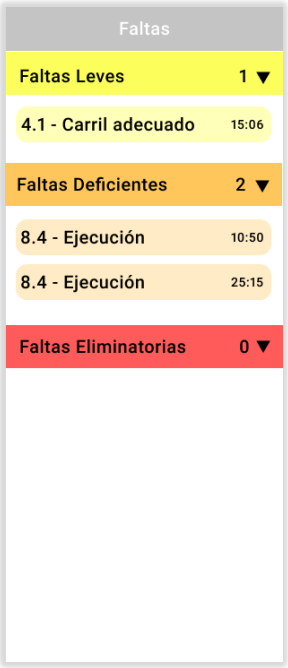


Figura 43. Mockup: Ventana lateral de faltas

Storyboard de los mockups del perfil de Alumno

Podemos ver el storyboard de los mockups del perfil de alumno en la Figura 44.

Figura 44. Mockups: Storyboard Alumno

4.4. Base de datos

Drivinglia debe poder tratar con los datos de usuarios de tipo alumno, profesor y administrador. Debido a la cambiante naturaleza de los datos con los que tratará Drivinglia nos decantaremos por una base de datos en tiempo real en la nube como la ***Real-Time Database*** de **Firebase**.

Firebase Real-Time Database funciona con registros de tipo **JSON**, por lo tanto, las tablas o colecciones de datos con las que trabajaremos, serán **no relacionales**. Principalmente contaremos con una **tabla de usuario** en la que estará contenida la información de alumno, profesor y administrador con ramificaciones en los roles de alumno para señalar las prácticas realizadas, el recorrido seguido en el mapa para cada una de ellas y las faltas cometidas. Esta opción nos dará una **estructura unificada** sobre los datos.

Como comentamos, habrá tres tipos de usuarios: alumno, profesor y administrador. Los alumnos tendrán así una tabla asociada de **prácticas** y para cada práctica habrá una tabla de **faltas** cometidas. Los **puntos de trayectoria** de una práctica los utilizaremos para mapear la trayectoria en el mapa e irán asociados a la práctica.

En la Figura 45, podemos ver los registros de cada tabla de manera esquematizada.

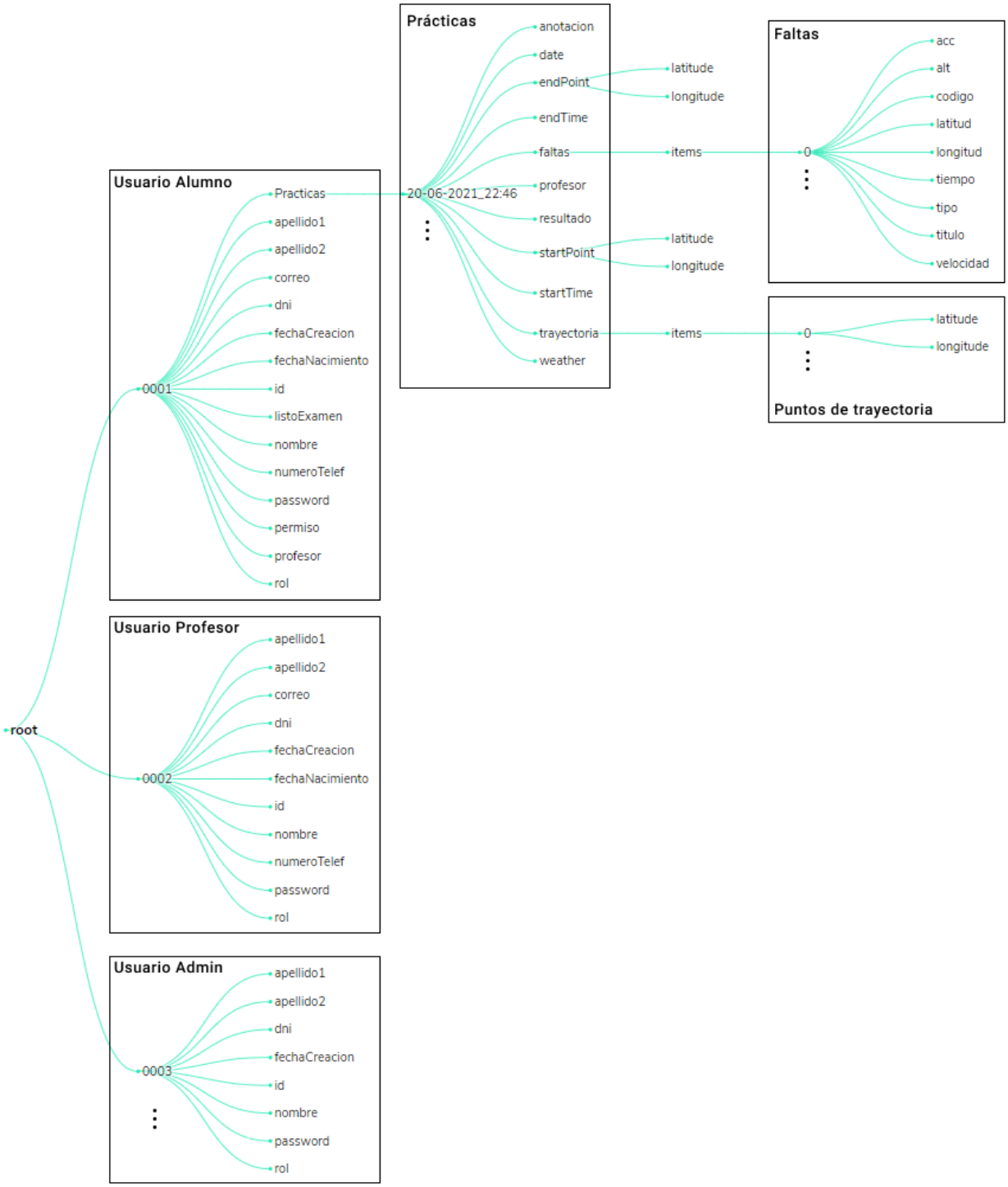


Figura 45. Esquema JSON de usuarios Firebase

Entraremos más en detalle acerca de los registros de la tabla de usuarios en el capítulo 5.1.

4.5. Patrón arquitectónico

Para cumplir con los principios de código limpio, el patrón arquitectónico que utilizaremos en la elaboración del proyecto será el de MVP (Modelo Vista Presentador). En términos generales, la vista es la parte de la aplicación que se encarga de manejar la interfaz de usuario de las pantallas, el modelo maneja los datos y el presentador la lógica de la aplicación.

Para ayudarnos a implementar esta arquitectura, utilizaremos una plantilla que dividirá las funciones de cada pantalla en diferentes clases. Tendremos que adaptar nuestra programación acorde a ellas.

A continuación, describimos las funciones de las clases que nos genera la plantilla MVP empleada para cada una de las pantallas:

- **Model.** Clase encargada de acceder a los datos y al repositorio.
- **View.** Recoge los eventos que ocurren en pantalla, modifica los elementos de la interfaz de usuario y accede a las funciones de Android.
- **Presenter.** Se encarga de la lógica de la aplicación y comunica a la vista con el modelo.
- **Screen.** Clase de configuración que es creada por la vista e inicializa y configura las otras clases del modelo MVP.
- **State.** Se encarga de mantener las variables del presentador.
- **ViewModel.** Extiende de *State* y almacena las variables que utilizaremos para actualizar los elementos de la vista.
- **Contract.** Interfaz que define los métodos que ha de implementar el modelo (*Model*), la vista (*View*) y el presentador (*Presenter*).

A parte de las clases mencionadas, tendremos otras dos clases de las que solo habrá una de cada para toda la aplicación:

- **Mediator.** Guarda el estado de la aplicación. Solo se crea una instancia del

mediador ya que lo programaremos como singleton, de forma que las referencias de las clases de estado que alberge se mantengan.

- **Repository.** Repositorio de la aplicación que contiene funciones que se repiten a las que puede acceder el modelo de las distintas pantallas.

En la Figura 46, podemos ver un esquema de las clases utilizadas en el patrón arquitectónico MVP de la plantilla.

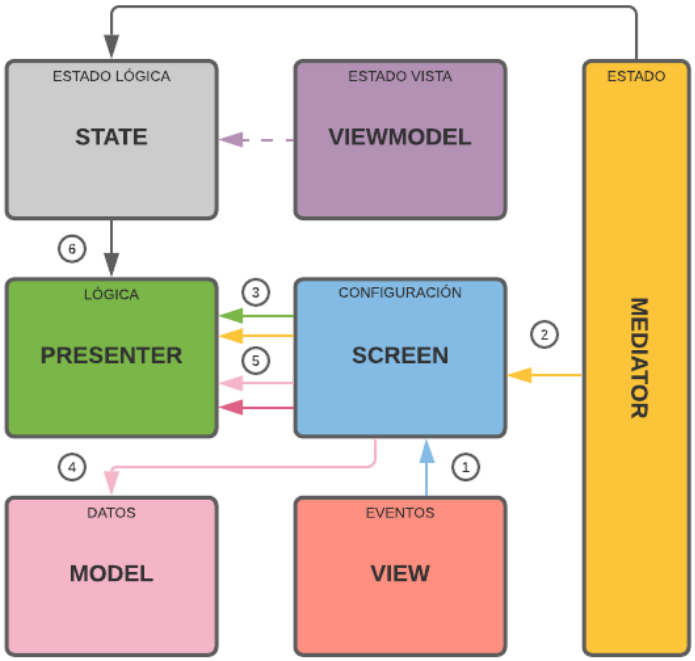


Figura 46: Esquema del patrón arquitectónico MVP empleado

Cuando se inicia la aplicación, se genera la vista (View), ya que es la clase que contiene las funciones Android. Esta vista, antes de actualizar los elementos en pantalla, inicializa la clase de configuración (*Screen*) **(1).** La clase de configuración (*Screen*) toma una referencia al mediador (*Mediator*) **(2)** y crea al presentador (*Presenter*) pasándole dicha referencia a través del constructor **(3)**. A su vez, crea el modelo (*Model*) **(4)** y se lo inyecta al presentador junto con la vista **(5)**. Es entonces cuando el presentador, a través de la referencia obtenida del mediador, accede a las variables de estado de la clase *State* y a las de la clase *ViewModel*, que extiende de esta **(6)**.

4.5.1. Interfaces de la plantilla

En este apartado vamos a analizar las clases, métodos y variables de relevancia que nos generará la plantilla MVP.

ACTIVITY (VIEW)

public class Activity
extends androidx.appcompat.app.AppCompatActivity
implements Contract.View

VARIABLES

presenter

private Contract.Presenter presenter

Referencia a la clase del presentador. Lo utilizaremos para establecer una comunicación entre la lógica (presentador) y la vista (*Activity*).

TAG

public static String TAG

Etiqueta que identifica la clase. Nos puede servir a la hora de imprimir un mensaje en la consola, por ejemplo.

MÉTODOS

injectPresenter

public void injectPresenter(Presenter presenter)

Setter que asigna la referencia del presentador a la variable global.

Parámetros		
Nombre	**Tipo**	**Descripción**
presenter	Presenter	Referencia a la clase Presenter.

navigateToNextScreen

public void navigateToNextScreen()

Inicia un *Intent* para navegar a la siguiente pantalla [32].

onBackPressed

public void onBackPressed()

Finaliza la *Activity* y se lo comunica al presentador.

Sobreescribe:

onBackPressed de la clase androidx.activity.ComponentActivity

onCreate

protected void onCreate(android.os.Bundle savedInstanceState)

Se llama al iniciar la pantalla. Vincula el archivo xml de la carpeta de recursos R.layout con la vista, llama al método *configure* de la clase *Screen* para que inicie la configuración de las otras clases de la plantilla y por último, comunica al presenter si nos encontramos en una pantalla de nuevo inicio (*onStart*) o no (*onRestart*).

Sobreescribe:

onCreate de la clase androidx.appcompat.app.AppCompatActivity

onDataUpdated

public void onDataUpdated(ViewModel viewModel)

Actualiza la vista con la información de las variables de estado del *ViewModel.*

Parámetros		
Nombre	**Tipo**	**Descripción**
viewModel	ViewModel	Referencia a la clase ViewModel, encargada de almacenar las variables que utilizamos en la vista.

onDestroy

protected void onDestroy()

Se llama justo antes de que se elimine la *Activity*. Comunica de ello al presentador.

Sobreescribe:

onDestroy de la clase androidx.appcompat.app.AppCompatActivity

onPause

protected void onPause()

Se llama justo antes de que se ponga en pausa la *Activity*. Comunica de ello al presentador.

Sobreescribe:

onPause de la clase androidx.fragment.app.FragmentActivity

onResume

protected void onResume()

Se llama justo antes de que la *Activity* entre o salga del primer plano [33]. Comunica de ello al presentador.

Sobreescribe:

onResume de la clase androidx.fragment.app.FragmentActivity

PRESENTER

public class Presenter
extends java.lang.Object
implements Contract.Presenter

CONSTRUCTOR

Presenter

public Presenter(AppMediator mediator)

En su creación, el presentador adquiere una referencia al mediador para acceder al estado de la aplicación.

Parámetros

Nombre	Tipo	Descripción
mediator	AppMediator	Referencia al mediador (Mediator) para acceder al estado de la aplicación.

VARIABLES

TAG

public static String TAG

Etiqueta que identifica la clase. Nos puede servir a la hora de imprimir un mensaje en la consola, por ejemplo.

view

private java.lang.ref.WeakReference<Contract.View> view

Referencia a la *Activity*. La utilizaremos para tener comunicación con la vista.

state

private State state

Referencia a la clase *State*. La utilizaremos para acceder a las variables de estado.

model

private Contract.Model model

Referencia a la clase *Model*. La utilizaremos para acceder a los métodos del modelo.

mediator

private AppMediator mediator

Referencia a la clase *Mediator*. La utilizaremos para acceder al estado de la aplicación.

MÉTODOS

getStateFromNextScreen

private NextToScreenState getStateFromNextScreen()

Accede al mediador para recuperar la información de la pantalla siguiente.

Retorno

Tipo	Descripción
NextToScreenState	Clase que almacena las variables que necesitamos de la pantalla siguiente.

getStateFromPreviousScreen

private PreviousToScreenState getStateFromPreviousScreen()

Accede al mediador para recuperar la información de la pantalla anterior.

Retorno	
Tipo	**Descripción**
PreviousToScreenState	Clase que almacena las variables que necesitamos de la pantalla anterior.

injectModel

void injectModel(Model model)

Setter que asigna la referencia del modelo a la variable global.

Parámetros		
Nombre	**Tipo**	**Descripción**
model	Model	Referencia a la clase *Model*, encargada de la comunicación con el repositorio y la gestión de los datos.

injectView

void injectView(java.lang.ref.WeakReference<Contract.View> view)

Setter que asigna la referencia de la vista a la variable global.

Parámetros		
Nombre	**Tipo**	**Descripción**
view	WeakReference	Referencia a la clase Activity, encargada de gestionar la vista (interfaz de usuario).

onBackPressed

void onBackPressed()

Gestiona la lógica cuando el usuario pulsa el botón de volver.

onDestroy

void onDestroy()

Gestiona la lógica cuando la pantalla se va a cerrar.

onPause

void onPause()

Gestiona la lógica cuando la pantalla entra en pausa.

onRestart

void onRestart()

Gestiona la lógica cuando la pantalla se vuelve a crear.

onResume

void onResume()

Gestiona la lógica cuando la pantalla vuelve o sale de primer plano.

onStart

void onStart()

Gestiona la lógica cuando la pantalla se crea por primera vez.

passStateToNextScreen

private void passStateToNextScreen(ScreenToNextState state)

Accede al mediador para pasarle información a la siguiente pantalla.

Parámetros		
Nombre	**Tipo**	**Descripción**
state	ScreenToNextState	Clase que almacena las variables que necesitará la pantalla siguiente.

passStateToPreviousScreen

private void passStateToPreviousScreen(ScreenToPreviousState state)

Accede al mediador para pasarle información a la anterior pantalla.

Parámetros		
Nombre	**Tipo**	**Descripción**
state	ScreenToPreviousState	Clase que almacena las variables que necesitará la pantalla anterior.

MODEL

public class Practica_setupModel
extends java.lang.Object
implements Practica_setupContract.Model

CONSTRUCTOR

Model

public Practica_setupModel(String data)

En su creación, el modelo adquiere algún tipo de dato. Por defecto, el constructor admite una simple variable con el nombre de la aplicación, pero en algunas pantallas lo modificaremos para que tenga una referencia al repositorio.

Parámetros		
Nombre	**Tipo**	**Descripción**
data	String	Datos que almacenar. Por defecto, el nombre de la aplicación.

VARIABLES

TAG

public static String TAG

Etiqueta que identifica la clase. Nos puede servir a la hora de imprimir un mensaje en la consola, por ejemplo.

data

private String data

Variable que almacena la información que contiene el modelo por defecto.

MÉTODOS

getStoredData

public String getStoredData()

Devuelve la información contenida en el modelo.

Retorno	
Tipo	**Descripción**
String	Datos del modelo. Por defecto, el nombre de la aplicación.

onDataFromNextScreen

public void onDataFromNextScreen(String data)

Gestiona la información de la pantalla siguiente que recibe el presentador a través del mediador.

Parámetros		
Nombre	**Tipo**	**Descripción**
data	String	Datos de la siguiente pantalla.

onDataFromPreviousScreen

public void onDataFromPreviousScreen(String data)

Gestiona la información de la pantalla anterior que recibe el presentador a través del mediador.

Parámetros		
Nombre	**Tipo**	**Descripción**
data	String	Datos de la anterior pantalla.

onRestartScreen

public void onRestartScreen(String data)

Gestiona la información de la pantalla una al volverse a iniciar.

Parámetros		
Nombre	**Tipo**	**Descripción**
data	String	Datos al volverse a iniciar la pantalla.

SCREEN

public class Screen
extends java.lang.Object

MÉTODOS

configure

public static void configure(View view)

Se encarga de iniciar las clases de la plantilla para que el modelo MVP pueda empezar a funcionar. Para ello genera una instancia del mediador y crea al presentador (*Presenter*). A su vez, crea el modelo y le inyecta al presentador tanto el modelo como una referencia a la vista (*Activity*). Finalmente, le inyecta a la vista (*Activity*) una referencia al presentador.

Parámetros

Nombre	Tipo	Descripción
data	View	Referencia a la vista (*Activity*).

STATE

public class Practica_setupState
extends Practica_setupViewModel

VIEWMODEL

public class Practica_setupViewModel
extends java.lang.Object

CONTRACT

public interface Practica_setupContract

APPMEDIATOR

public class AppMediator
extends java.lang.Object

VARIABLES

INSTANCE

private static AppMediator INSTANCE

Variable que almacena la instancia o referencia del mediador (*AppMediator*).

MÉTODOS

getInstance

public static AppMediator getInstance()

Si no se ha creado la clase (mediador) antes, genera una referencia a la misma, la asigna a la variable global y la retorna. Si ya existía, devuelve la que ya estaba creada. Este método sirve para conservar la información del mediador y así operar como un singleton, una clase de la que solo se genera una referencia o instancia.

Retorno

Tipo	Descripción
AppMediator	Referencia a la clase mediador (*AppMediator*).

resetInstance

public static void resetInstance()

Hace nula la variable global que almacena la referencia al mediador (*AppMediator*), liberando así cualquier información que haya almacenado.

5. Implementación

A continuación, vamos a describir el proceso de desarrollo de la aplicación. Describiremos los registros de la base de datos, el JSON de configuración y las interfaces de funcionalidad aplicadas junto con las decisiones de testing y verificación.

5.1. Tablas de la base de datos

Principalmente, para la base de datos en tiempo real, tenemos una tabla no relacional (clave-valor) de usuarios que pueden ser tanto administrador, profesor o alumno. Nuevos usuarios se crean con un id en incremento de 4 dígitos, permitiendo hasta 10.000 combinaciones, en un principio.

En la Figura 47, podemos ver la tabla de usuarios no desplegada en la consola de Firebase.

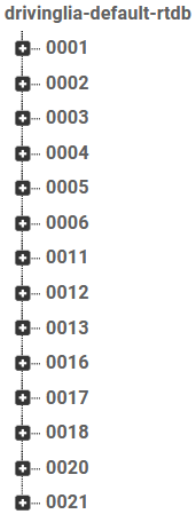


Figura 47: Tabla de usuarios no desplegada en Firebase

Para distinguir el perfil de los usuarios tenemos un campo de rol para cada uno. Este puede tomar los valores de *Profesor*, *Alumno* o *Administrador*.

5.1.1. Usuario administrador

Para los usuarios de rol *Administrador*, podemos ver en la Tabla *1* los diferentes registros que contienen.

Clave	Tipo	Descripción
apellido1	String	Valor del primer apellido.
apellido2	String	Valor del segundo apellido.
dni	String	Valor del dni sin letra.
fechaCreacion	String	Fecha en la que se creó dicho usuario en formato

		dd/MM/yyyy.
id	String	Valor del id del usuario. Debe coincidir con el valor del padre. Ha de tener 4 dígitos.
nombre	String	Valor del nombre.
password	String	Contraseña de usuario. Ha de estar cifrada en UTF-8 y su valor son las últimas 6 cifras del dni.
rol	String	Rol de usuario. En este caso "*Administrador*".

Tabla 1. Registros de Usuario Administrador

En la Figura 48, podemos ver a un usuario de rol administrador desplegado desde Firebase.

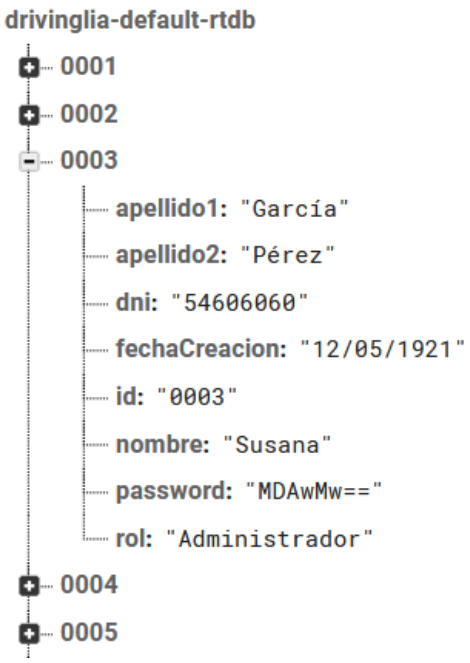


Figura 48: Tabla de usuario administrador desplegada en Firebase

5.1.2. Usuario profesor

Para los usuarios de rol *Profesor*, podemos ver en la Tabla 2 los diferentes registros que contienen.

Clave	Tipo	Descripción
apellido1	String	Valor del primer apellido.

apellido2	String	Valor del segundo apellido.
correo	String	Valor del correo electrónico. Ha de tener un formato válido. Ej: (someone@example.com)
dni	String	Valor del dni sin letra.
fechaCreacion	String	Fecha en la que se creó dicho usuario en formato dd/MM/yyyy.
fechaNacimiento	String	Fecha en la que nació dicho usuario en formato dd/MM/yyyy.
id	String	Valor del id del usuario. Debe coincidir con el valor del padre. Ha de tener 4 dígitos.
nombre	String	Valor del nombre.
numeroTelef	String	Valor del número de teléfono.
password	String	Contraseña de usuario. Ha de estar cifrada en UTF-8 y su valor son las últimas 6 cifras del dni.
rol	String	Rol de usuario. En este caso "*Profesor*".

Tabla 2. Registros de Usuario Profesor

En la Figura 49, podemos ver a un usuario de rol profesor desplegado desde Firebase.

Figura 49: Tabla de usuario profesor desplegada en Firebase

5.1.3. Usuario alumno

La tabla de usuarios, a diferencia de las anteriores, cuenta con ramificaciones o subtablas asociadas que analizaremos.

Para los usuarios de rol *alumno*, podemos ver en la Tabla 3 los diferentes registros que contienen.

Clave	Tipo	Descripción
apellido1	String	Valor del primer apellido.
apellido2	String	Valor del segundo apellido.
correo	String	Valor del correo electrónico. Ha de tener un formato válido. Ej: (someone@example.com)
dni	String	Valor del dni sin letra.
fechaCreacion	String	Fecha en la que se creó dicho usuario en formato dd/MM/yyyy.
fechaNacimiento	String	Fecha en la que nació dicho usuario en formato dd/MM/yyyy.
id	String	Valor del id del usuario. Debe coincidir con el valor del padre. Ha de tener 4 dígitos.
listoExamen	boolean	Nos indica si el alumno está listo para realizar. examen o no. Se crea falso por defecto.
nombre	String	Valor del nombre.
numeroTelef	String	Valor del número de teléfono.
password	String	Contraseña de usuario. Ha de estar cifrada en UTF-8 y su valor son las últimas 6 cifras del dni.
permiso	String	Permiso de conducir del que se matricula el usuario. Puede ser B, A, A1, A2, D o C.
rol	String	Rol de usuario. En este caso "*Alumno*".

Tabla 3. Registros Usuario Alumno

De uno de los hijos de los usuarios de tipo *Alumno*, puede colgar una **tabla de prácticas** de nombre *Practicas* en el caso de que haya realizado alguna. Los hijos de esta tabla se crearán con el valor de la fecha de creación de la práctica en formato *dd-MM-yyyy* más la hora en formato *horas:minutos*.

La tabla *Practicas* puede tener los registros que vemos en la Tabla 4.

Clave	Tipo	Descripción
anotacion	String	Valor de la anotación del profesor al final de la práctica.
date	String	Fecha en la que se inició la práctica. Ha de tener el formato dd-MM-yyyy. Al igual que el título del padre.
endPoint	LatLng	Valores de *latitude* (latitud) y *longitude* (longitud) de la posición en la que se terminó la práctica. Ambos campos son de tipo double.
profesor	String	Nombre del profesor que ha registrado la práctica.
resultado	String	Calificación de la práctica. Se calcula en función de las faltas y puede ser APTO o NO APTO.
startPoint	LatLng	Valores de *latitude* (latitud) y *longitude* (longitud) de la posición en la que se comenzó la práctica. Ambos campos son de tipo double.
startTime	String	Valor de la hora en la que se inició la práctica. Coincide con el nombre del padre y tiene el formato de *horas:minutos*.
trayectoria	LatLng[]	Array de posiciones tomadas en el tracking. Cada elemento tiene valores de *latitude* (latitud) y *longitude* (longitud) de tipo double.
weather	String	Condiciones climatológicas en las que se desarrolló la práctica. Ha de coincidir con uno de los estados aceptados por la API MetaWeather.

Tabla 4. Registros de Prácticas de Alumno

De uno de los hijos de la tabla de *Practicas*, puede colgar una **tabla de faltas** de nombre *faltas* en el caso de que el profesor haya registrado alguna a lo largo de la práctica.

La tabla *faltas* puede tener los registros que vemos en la Tabla 5.

Clave	Tipo	Descripción
acc	double	Valor de la precisión de la toma de posición GPS

		asociada a la falta.
alt	double	Valor de la altitud de la toma de posición GPS asociada a la falta.
codigo	String	Código asociado a la falta.
latitud	double	Valor de la latitud de la toma de posición GPS.
longitud	double	Valor de la longitud de la toma de posición GPS.
tiempo	String	Tiempo de práctica en el que se registró la falta. (horas y minutos)
tipo	String	Gravedad de la falta. Puede ser leve, deficiente o eliminatoria.
titulo	String	Título de la infracción cometida.
velocidad	double	Valor de la velocidad de la toma de posición GPS asociada a la falta.

Tabla 5.Registros de Faltas de las Prácticas de Alumn

En la Figura 50, podemos ver a un usuario de rol alumno desplegado desde Firebase.

Figura 50: Tabla de usuario profesor desplegada en Firebase

5.2. Almacenamiento de imágenes en la nube

Para almacenar las imágenes de usuario, utilizamos **Firebase Storage**. Hemos creado una carpeta de nombre *fotosPerfil* que alberga las distintas imágenes en formato jpg.

El título de las imágenes representa el id del usuario al que se asocian. En la Figura 51 y Figura 52 podemos ver, desde la consola de Firebase, la carpeta en la que se encuentran las imágenes de usuario.

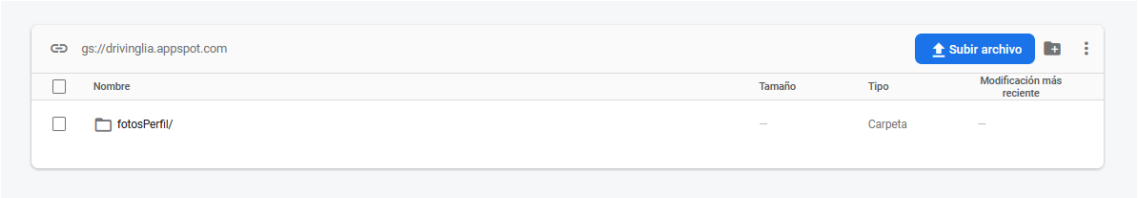


Figura 51: carpeta fotosPerfil en la interfaz de Firebase Storage

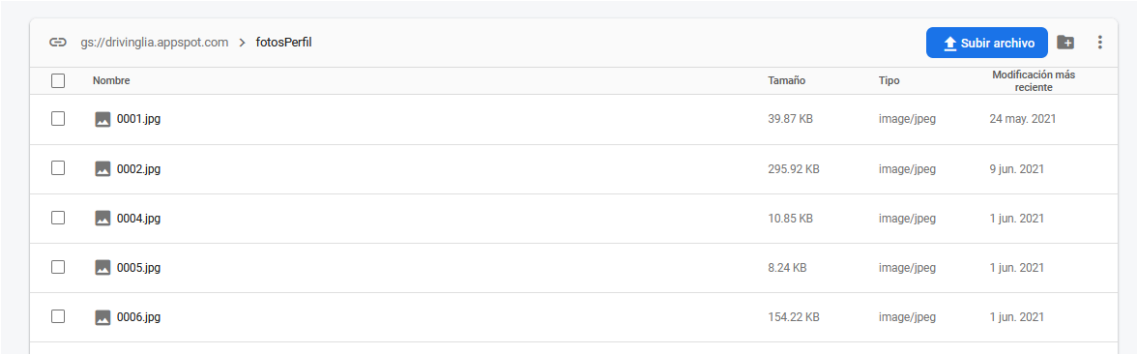


Figura 52: imágenes de usuario de la carpeta fotosPerfil

5.3. JSON de configuración

Para recuperar la información de las faltas de conducción disponibles, hemos creado un archivo JSON de configuración titulado *faltas*. Dicho archivo se encuentra en la carpeta de recursos *assets* del proyecto [34].

En la Figura 53, podemos ver la ubicación de la carpeta *assets* desde Android Studio.

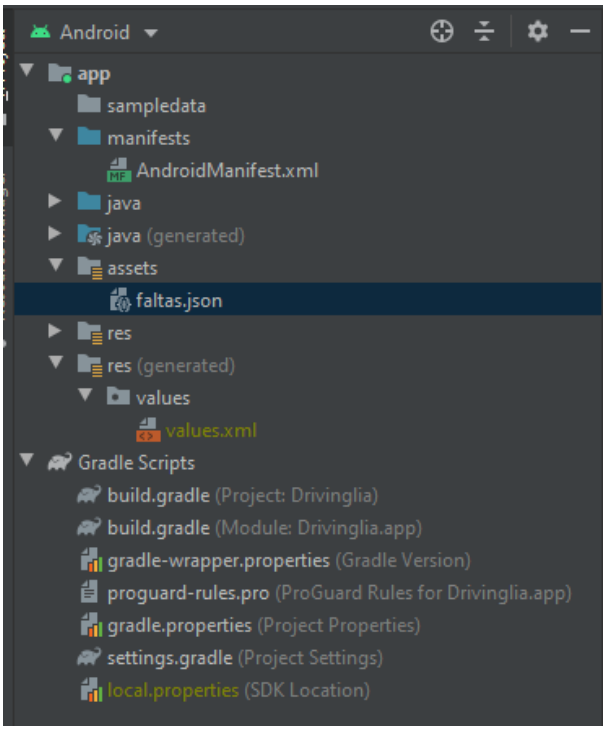


Figura 53: Archivo JSON faltas de la carpeta assets

Los objetos escritos en el archivo JSON de configuración intentan plasmar la información del documento de *criterios de calificación en vías abiertas* de la DGT mencionado en capítulos anteriores.

Como vemos en la Figura 54 las faltas se dividen en secciones (raya azul de la tabla) que van desde la 1 hasta la 15 dependiendo de la situación en la que ocurra la infracción (adelantamientos, cruces, etc.). Por otro lado, la falta en cuestión (raya verde de la tabla) puede ser leve, deficiente o eliminatoria.

EPÍGRAFE	F. ELIMINATORIAS	F. DEFICIENTES	F. LEVES	NOTAS ACLARATORIAS
4.- PROGRESIÓN NORMAL.				
4.1.- Carril adecuado.		No circular por el carril derecho cuando sea reglamentario, obstaculizando. En vías fuera de poblado con 3 o más carriles en el mismo sentido, circular por un carril no autorizado para ese vehículo o conjunto de vehículos, obstaculizando.	No circular por el carril derecho cuando sea reglamentario. En vías fuera de poblado con 3 o más carriles en el mismo sentido, circular por un carril no autorizado para ese vehículo o conjunto de vehículos.	**Solamente** para permisos de las clases C1, C y conjuntos de más de siete metros.

Figura 54: Documento de criterios de calificación en vías abiertas de la DGT

En el archivo *JSON* de configuración tenemos un array principal que contiene las faltas leves, deficientes y eliminatorias del reglamento de la DGT y otro con las secciones disponibles

en el epígrafe del documento que comentamos en el párrafo anterior (raya azul de la tabla).

En cada objeto JSON de **falta leve, deficiente o eliminatoria** hay 4 pares clave/valor que podemos ver en la Tabla 6.

Clave	Tipo	Descripción
codigo	String	Código de la falta en cuestión (1.1, 2.4…etc). El primer dígito representa la sección y el segundo el número de falta en dicha sección.
titulo	String	Nombre de la falta.
descripcion	String	Motivo de la falta.
pauta	String	Mensaje de punto en el que puede mejorar el alumno dependiendo del motivo de la falta.

Tabla 6. Registros JSON de faltas

En la Figura 55, podemos ver en el JSON de configuración, el array de faltas y algunas faltas leves del array de faltas leves.

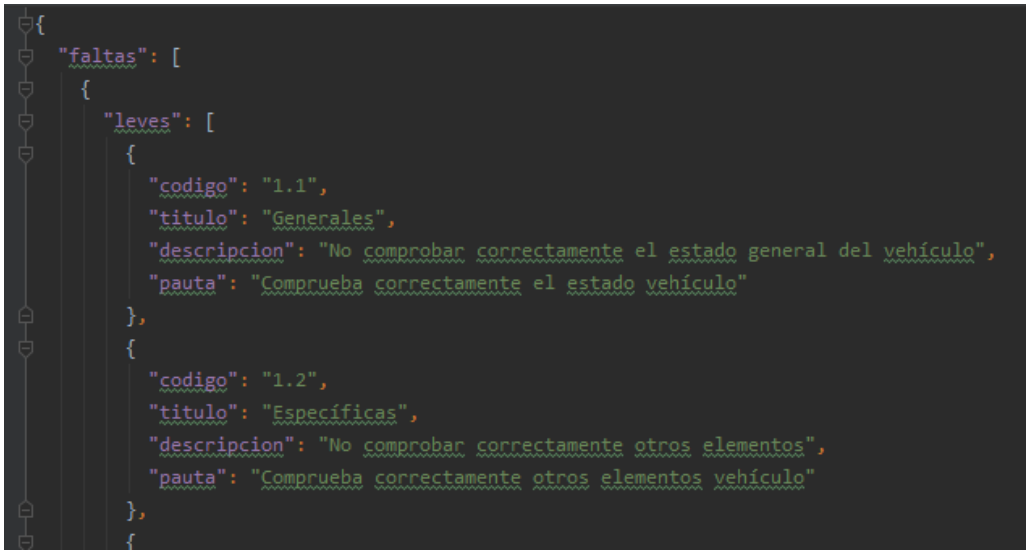


Figura 55: Faltas en el archivo JSON de configuración

Por otro lado, los objetos JSON **secciones** tienen 5 pares clave/valor que podemos ver en la Tabla 7.

Clave	Tipo	Descripción
codigo	String	Código de la sección. Corresponde al primer dígito de una falta.

titulo	String	Nombre de la sección.
leves	boolean	Nos indica si hay faltas leves en el reglamento pertenecientes a dicha sección. Tiene como propósito facilitarnos la selección en la pantalla de práctica.
deficientes	boolean	Nos indica si hay faltas deficientes en el reglamento pertenecientes a dicha sección. Tiene como propósito facilitarnos la selección en la pantalla de práctica.
eliminatorias	boolean	Nos indica si hay faltas eliminatorias en el reglamento pertenecientes a dicha sección. Tiene como propósito facilitarnos la selección en la pantalla de práctica

Tabla 7, Registros JSON de secciones

En la Figura 56, podemos ver en el JSON de configuración, el array de secciones desplegado.

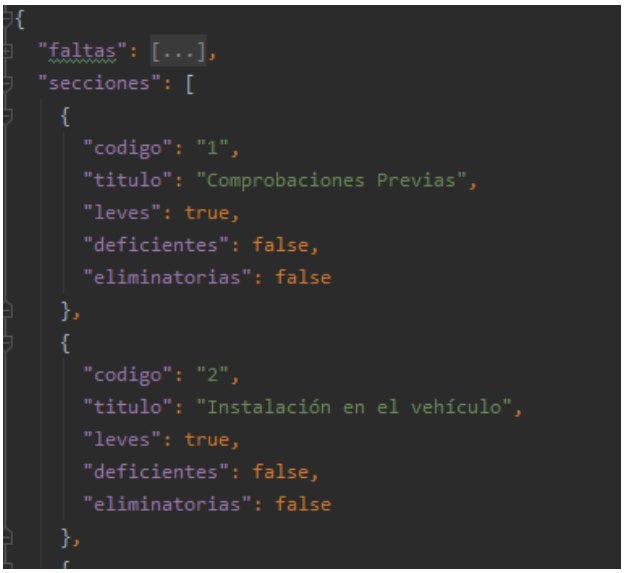


Figura 56: Secciones en el arhivo JSON de configuración

La información del archivo JSON de faltas la trataremos como listas de objetos Java. En el apartado 5.6 entraremos en detalle acerca del procedimiento de mapeo o *parsing* de los datos de este archivo.

En la Figura 57 podemos ver un esquema de la estructura del JSON de configuración:

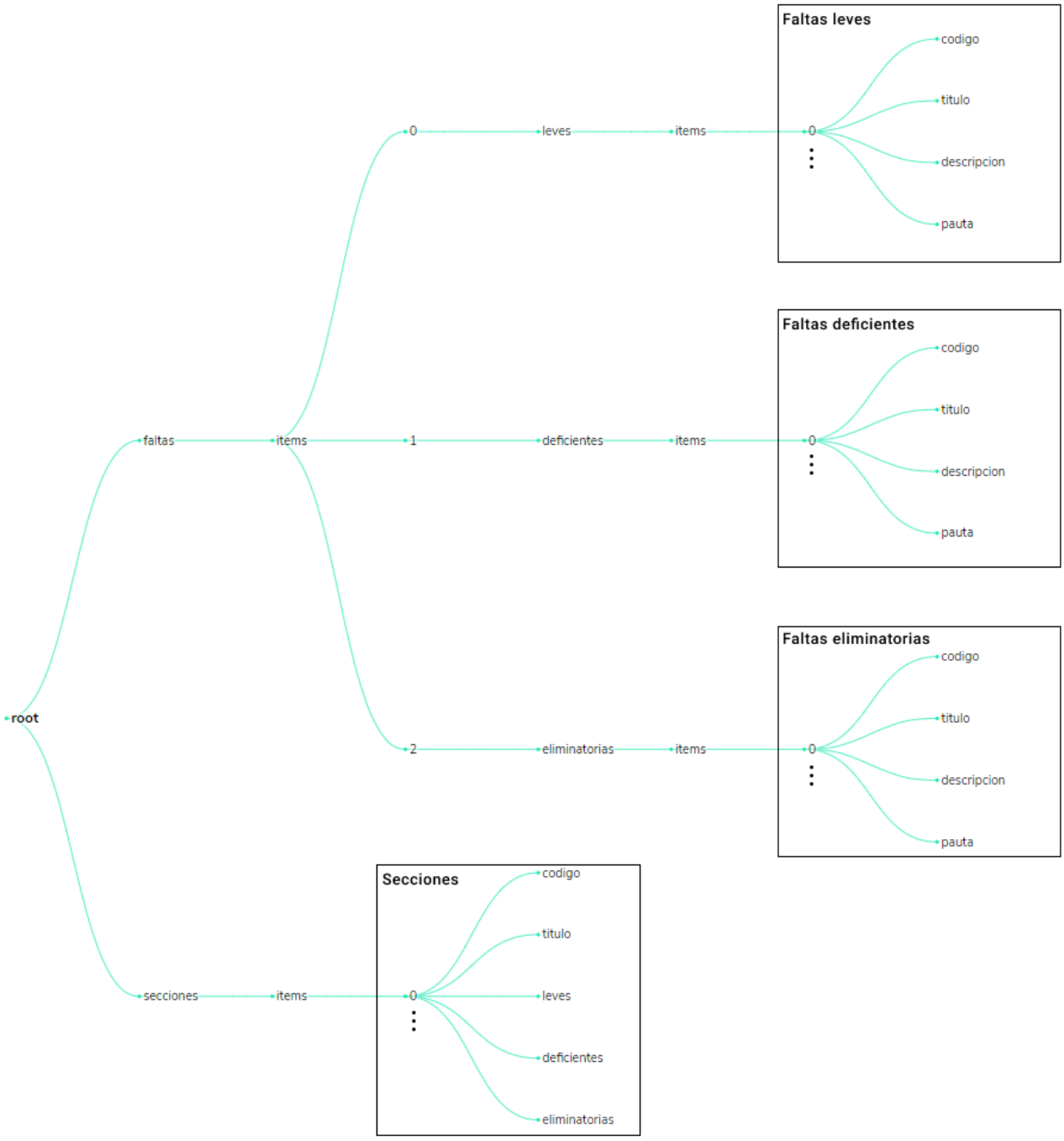


Figura 57. Esquema JSON de configuración

5.4. Interfaces de Funcionalidades

Este apartado, en el que describimos las interfaces de funcionalidades comunes y básicas, se ha movido a la sección Anexos debido a su extensión.

Las **interfaces de funcionalidades comunes** son aquellas a las que acceden las distintas clases de cada pantalla de manera indistinta. Entre ellas, contamos con el repositorio, las clases destinadas a realizar las peticiones http y otras clases de uso.

A continuación, vamos a comentar las distintas referencias de las que hemos hecho uso en este segmento de la memoria, ya sea para su creación en el periodo de desarrollo o como explicación.

En el método ***fetchPracticasFromFirebase*** de la clase **Repository**, hacemos uso de la referencia [35].

Para explicar el funcionamiento de la API de direcciones de Google en la clase ***DirectionsJsonParser***, hacemos uso de las referencias [36] y [37]. En ella, también comentamos que la clase fue extraída de la página **JournalDev**, encontrada en la referencia [38].

Para la explicación y creación de la clase **WeatherDataService** hemos hecho uso de las referencias [39] y [40].

Las **interfaces de funcionalidades básicas** son aquellas que desarrollan la lógica y el modelo de cada pantalla.

A continuación, vamos a comentar las distintas referencias de las que hemos hecho uso en este otro segmento de la memoria, ya sea para su creación en el periodo de desarrollo o como explicación.

En el método ***onLoginButtonClicked*** de la clase **LoginPresenter**, hacemos uso de la referencia [41].

En el método ***getAvatarImageFromFirebase*** de la clase **PerfilPresenter**, hacemos uso de la referencia [42].

En la clase **AlumnosPresenter**, el método ***onActionButtonClicked*** de la clase hace uso de la referencia [43], el método ***onFiltrarApplied*** hace uso de la referencia [44] y el método ***sortByAlphabet*** hace uso de las referencias [45] y [46].

En la clase **EditAlumnosPresenter**, el método ***onDataSet*** hace uso de las referencias [47] y [48], el método ***onOptionsItemSelected*** hace uso de las referencias [49] y [50]

En la clase **AddAlumnosPresenter**, el método ***onConfirmButtonClicked*** hace uso de la referencia [51] y el método ***generateEmail*** hace uso de la referencia [52].

En la clase **PracticaPresenter**, la constante ***FALTA_LEVE_CODE*** hace uso de la referencia [53], el método ***setupLocationRequest*** hace uso de la referencia [54], el método ***onNightModeClicked*** hace uso de la referencia [55], el método ***onTimerCreated*** hace uso de la referencia [56], el método ***onMicrophoneClicked*** hace uso de la referencia [57], el método ***onInfoWindowClicked*** hace uso de la referencia [58] y el método ***onFaltaListItemClicked*** hace uso de la referencia [59].

En el método ***setupExpandableListAdapter*** de la clase **StatisticsPresenter**, hacemos uso de la referencia [60]

5.5. Verificación y testing

Con el objetivo de poner a prueba la aplicación en un entorno real se han realizado **test de usabilidad.** En ellos, la aplicación se ha probado en un circuito en carretera de 5 kilómetros simulándola a través de un dispositivo móvil físico. Enfatizamos el aspecto físico ya que ciertas características críticas de la aplicación, como la realización de las prácticas de conducir, han de considerarse en un entorno real, donde pueden surgir excepciones que debemos tratar.

Gracias a estos test de usabilidad, se detectaron y corrigieron ciertos elementos que ocasionaban problemas como pueden ser la interacción con los marcadores del mapa al realizar la práctica. A su vez, ajustamos el intervalo de tiempo en el que debíamos tomar las muestras de posición para no comprometer a la batería y la distancia óptima entre muestras a la que llamaríamos a la API de direcciones de Google para trazar la polilínea del recorrido del vehículo sobre el mapa.

A su vez, se pulieron ciertas funciones, como la toma de la posición del dispositivo cuando

el profesor señala la falta y no cuando especifica el código de esta. De modo que el profesor pueda señalar la falta y corregir al alumno para que una vez todo esté controlado, poder especificar el código de la misma sin que el marcador se cree en el punto actual, sino en el que señaló la falta. También corregimos otro problema de los marcadores en el que se creaban uno encima de otro al especificar varias faltas en el mismo lugar añadiendo un offset sobre la coordenada de longitud.

A parte de detectar y corregir fallos gracias a la simulación en un dispositivo físico como puede ser el móvil personal, también se han realizado **test de instrumentación** orientados a la interfaz de usuario con la intención de poner a prueba la interacción a nivel de usuario con los elementos en pantalla y la navegación entre ellas [61].

Hemos utilizado la herramienta generadora de test **Espresso** ya que nos facilita la labor de escritura de los mismos [62].

Dichos test se ejecutan sobre el emulador que tengamos vinculado a Android Studio y pueden comprobar distintas funciones de la interfaz de usuario como:

- Elementos de la vista y su lógica. A través de los identificadores únicos y las distintas propiedades que pueden llegar a adoptar.
- Interacción entre pantallas.
- Giro de pantalla. En Android, un giro de pantalla implica gestionar el estado de la vista de manera correcta, debido a que los elementos en pantalla se tienen que volver a crear.
- Ciclo de vida de la aplicación.
- Etcétera…

Los test que hemos realizado prueban el funcionamiento completo de los 3 perfiles de uso de la aplicación: alumno, profesor y administrador.

A continuación, nombramos las interfaces con las que cuenta la aplicación para realizar el testing:

RecyclerViewMatcher

public class RecyclerViewMatcher
extends java.lang.Object

Utilizaremos esta clase para poder testear las listas que implementan *RecyclerView*. Podremos comprobar los elementos de las posiciones individuales de la lista y realizar acciones como clickear sobre ellos [63].

Constantes

recyclerViewId

private final int recyclerViewId

Id del *RecyclerView* que queremos testear.

Constructor

RecyclerViewMatcher

public RecyclerViewMatcher(int recyclerViewId)

Parámetros

Nombre	Tipo	Descripción
recyclerViewId	Int	Id del *RecyclerView* que queremos testear.

Métodos

atPositionOnView

public org.hamcrest.Matcher<View> atPositionOnView(int position, int targetViewId)

Utilizaremos este método para acceder a las posiciones de las listas *RecyclerView.*

Parámetros		
Nombre	**Tipo**	**Descripción**
position	int	Posición del elemento al que queremos acceder en la lista.
targetViewId	int	Id del elemento hijo de la lista al que queremos acceder y testear. Ej: Id de la *CardView*

Retorno	
Tipo	**Descripción**
Matcher<View>	Emparejador de la vista del elemento que queremos acceder o falso si no lo ha conseguido detectar [64].

INSTRUMENTEDTESTS

public class InstrumentedTests
extends java.lang.Object

Clase en la que ejecutaremos los test de implementación de la aplicación. Comprobaremos funciones de la interfaz de usuario separadas en los distintos perfiles de uso de la aplicación: alumno, profesor y administrador.

Para testear estos 3 roles, iniciaremos sesión programáticamente con las cuentas de testing haciendo uso de las funciones de Espresso, las cuales tienen las siguientes credenciales.

- Alumno

    - Id: 0001
    - Contraseña: 0001
- Profesor
    - Id: 0002
    - Contraseña: 0002
- Administrador.
    - Usuario: 0003
    - Contraseña: 0003

Para aquellas pantallas que necesiten acceder a la red de internet, pondremos a dormir al hilo de ejecución durante 3 segundos para que se completen los procesos. Esta técnica tiene un margen de error, pero la gran mayoría de veces no da problemas así que ha resultado la opción más cómoda.

Para cada test, las comprobaciones previas a la acción llevan el comentario "**GIVEN**", las acciones como el click de los elementos llevan el comentario "**WHEN**" y el resultado de la acción lleva el comentario "**THEN**". De esta forma podemos mantener una metodología de testing que va de la mano con los **requisitos** de la aplicación.

Por último, hay que comentar que los métodos que sean tests, llevan la anotación *@Test* delante para señalarle al compilador que se trata de un test a ejecutar.

CONSTANTES

mActivityTestRule

public androidx.test.rule.ActivityTestRule<LoginActivity> mActivityTestRule

Le indica a los test desde que pantalla deben lanzarse. En este caso, la pantalla de inicio de sesión (Login), debido a que necesitamos identificarnos con los distintos perfiles para comprobar las otras pantallas. Lleva la anotación @Rule para señalar una regla.

mGrantPermissionRule

public androidx.test.rule.GrantPermissionRule mGrantPermissionRule

Autoriza a los test a tener permisos de acceso a la ubicación. Lleva la anotación @Rule para señalar una regla.

MÉTODOS

setUp

public void setUp()

Realiza la inicialización y grabación de los *Espresso-Intents* para que ciertos test puedan probar los Intents de la aplicación. Se llama antes de cada test gracias a la anotación *@Before*.

tearDown

public void tearDown()

Limpia el estado de los Intents. Se llama después de cada test gracias a la anotación *@After*.

alumnoActivityIntentShareTest

public void alumnoActivityIntentShareTest() throws InterruptedException

Comprueba que la función de compartir el balance de faltas totales en el perfil de alumno esté lanzando el *Intent* para compartir a través de otras aplicaciones correctamente.

alumnoActivityFullTest

public void alumnoActivityFullTest( )throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de alumno comprobando e interactuando con los elementos de cada una.

alumnoActivityWithRotationFullTest

public void alumnoActivityWithRotationFullTest() throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de alumno comprobando e interactuando con los elementos de cada una y rotando la pantalla para comprobar que el estado de la aplicación se recupera correctamente ante cambios en el ciclo de vida de la aplicación.

profesorActivityIntentVoiceTest

public void profesorActivityIntentVoiceTest() throws InterruptedException

Comprueba que la función de indicar la falta oralmente en el perfil de profesor esté lanzando el *Intent* para realizar el reconocimiento de voz a través de las funciones de Android correctamente.

profesorActivityFullTest

public void profesorActivityFullTest() throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de profesor comprobando e interactuando con los elementos de cada una.

profesorActivityWithRotationFullTest

public void profesorActivityWithRotationFullTest() throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de profesor comprobando e interactuando con los elementos de cada una y rotando la pantalla para comprobar que el estado de la aplicación se recupera correctamente ante cambios en el ciclo de vida de la aplicación.

administradorActivityIntentImageSelectTest

public void administradorActivityIntentImageSelectTest() throws InterruptedException

Comprueba que la función de seleccionar imagen de perfil al editar usuario en el perfil de administrador esté lanzando el *Intent* para acceder a las funciones de la librería de recorte de

imágenes correctamente.

administradorActivityFullTest

public void administradorActivityFullTest() throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de administrador comprobando e interactuando con los elementos de cada una.

administradorActivityWithRotationFullTest

public void administradorActivityWithRotationFullTest() throws InterruptedException

Hace una ejecución completa de las distintas pantallas del perfil de administrador comprobando e interactuando con los elementos de cada una y rotando la pantalla para comprobar que el estado de la aplicación se recupera correctamente ante cambios en el ciclo de vida de la aplicación.

childAtPosition

private static Matcher<View> childAtPosition(Matcher<View> parentMatcher,
int position)

Método que genera Espresso para comprobar elementos de la vista que se encuentran contenidos dentro de otros, como puede ser un elemento botón (*Button*) dentro de un elemento *layout* (*LinearLayout*), etcétera.

Parámetros

Nombre	Tipo	Descripción
parentMatcher	Matcher<View>	Emparejador del elemento padre.
position	Int	Posición del elemento hijo que queremos comprobar

Retorno

Tipo	Descripción
Matcher<View>	Emparejador del elemento hijo.

withRecyclerView

public static RecyclerViewMatcher withRecyclerView(int recyclerViewId)

Inicia un objeto de tipo *RecyclerViewMatcher* con el id del *RecyclerView* que queremos testear a través de su constructor y lo retorna.

Parámetros

Nombre	Tipo	Descripción
recyclerViewId	int	Id del *RecyclerView* que queremos testear.

Retorno

Tipo	Descripción
RecyclerViewMatcher	Clase con los métodos necesarios para acceder a los emparejadores de los elementos hijo del *RecyclerView* que queremos comprobar.

rotateScreen

public void rotateScreen()

Realiza la acción de rotar pantalla. Cambia la orientación de la pantalla hacia la izquierda, espera 700 milisegundos y la devuelve a su estado natural para esperar otros 700 milisegundos y así darle a la aplicación un tiempo aproximado para que recupere el estado de los elementos.

Utilizamos este método cada vez que queremos rotar la pantalla en los distintos test de rotación de pantalla.

En la Tabla 8. Esquema de los test de mostramos un esquema de los test de instrumentación implementados.

Nombre	Definición
alumnoActivityIntentShareTest	Comprueba la función de compartir el balance de faltas totales en el perfil de alumno.
alumnoActivityFullTest	Ejecución completa de las distintas pantallas del perfil de alumno.
alumnoActivityWithRotationFullTest	Ejecución completa de las distintas pantallas del perfil de alumno, con rotación.
profesorActivityIntentVoiceTest	Comprueba la función de indicar la falta oralmente en el perfil de profesor.
profesorActivityFullTest	Ejecución completa de las distintas pantallas del perfil de profesor.
profesorActivityWithRotationFullTest	Ejecución completa de las distintas pantallas del perfil de profesor, con rotación.
administradorActivityIntentImageSelectTest	Comprueba la función de seleccionar imagen de perfil en el perfil de administrador.
administradorActivityFullTest	Ejecución completa de las distintas pantallas del perfil de administrador.
administradorActivityWithRotationFullTest	Ejecución completa de las distintas pantallas del perfil de profesor, con rotación.

Tabla 8. Esquema de los test de instrumentación

6. Guía de usuario

En este capítulo explicaremos cómo utilizar la aplicación y las funciones a las que puede acceder el usuario en función del rol al que pertenece: administrador, profesor o alumno.

Al ser un capítulo orientado al usuario, evitaremos tecnicismos en la medida de lo posible.

Ilustración 13: Guía de usuario

(Imagen extraída de SmartMockups) [65]

6.1. Pantallas finales

6.1.1. Inicio de sesión

Lo primero que necesitas para poder utilizar **Drivinglia** es un dispositivo móvil con sistema operativo **Android** versión **8.0** o superior y con los servicios de ***Google Play Services*** instalados.

Para lanzar la aplicación, simplemente has de dirigirte a la ventana de aplicaciones de tu dispositivo y presionar sobre el icono que corresponde a **Drivinglia** como puedes observar en la Figura 58.

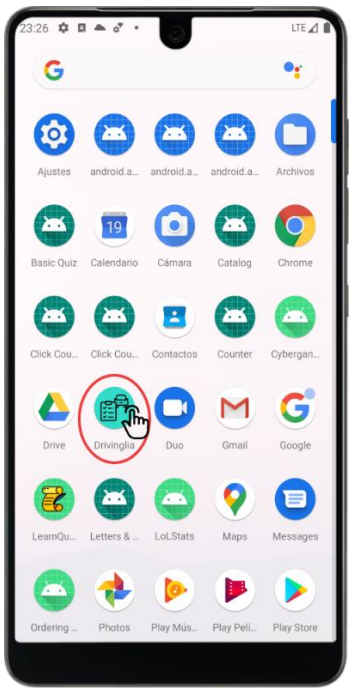


Figura 58: Ventana de Aplicaciones

Pantalla de Inicio de Sesión

Lo primero que verás al ejecutar la aplicación es la pantalla de **inicio de sesión**. Esta pantalla cuenta con dos **campos de texto** donde podrás introducir tus credenciales y un **botón** de confirmación. Por defecto, cada usuario puede iniciar sesión tanto con el **id** asignado en la base de datos como con el **dni** sin la letra. Puedes ver los elementos de la pantalla de inicio de sesión en la Figura 59.

Figura 59. Inicio de Sesión: Elementos

Cuando presionas el botón de confirmar, Drivinglia inicia el proceso de autenticación. Se bloquearán los elementos en pantalla momentáneamente y aparecerá una **barra de progreso**, como puedes ver en la Figura 60.

Figura 60. Inicio de Sesión: Iniciar

Si introduces de manera **errónea** las credenciales, no figuras en la base de datos o faltan campos que rellenar en el inicio de sesión, se cancelará el proceso y aparecerá un **mensaje** indicándonos el motivo. Podrás ver los distintos mensajes de error en la Figura 61.

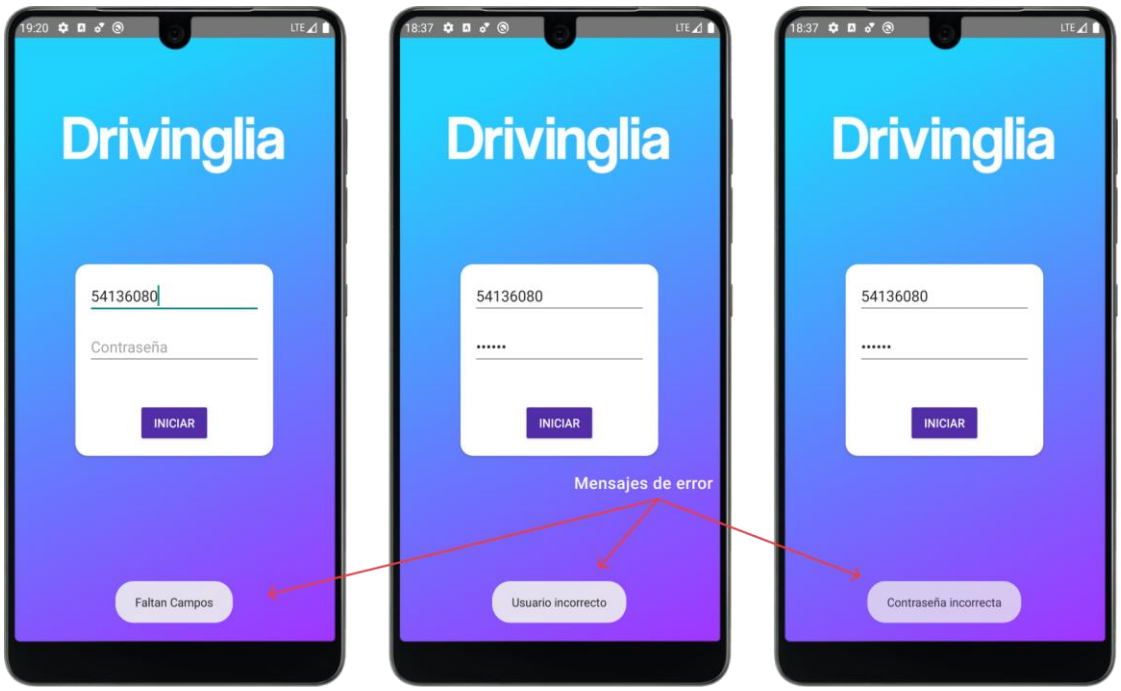


Figura 61. Inicio de Sesión: Mensajes de error

6.1.2. Administrador

El perfil de administrador es el encargado de gestionar las cuentas de los alumnos y profesores. Hay que destacar que la creación de las cuentas de administrador está restringida debido a las funciones críticas a las que puede acceder este perfil de usuario.

Pantalla de Perfil de Administrador

Una vez inicias sesión con una cuenta de administrador, entrarás en la pantalla de **perfil** que aparece en la Figura 62. En la parte superior de la pantalla vas a encontrar tu **imagen de usuario,** si la hubiera, o una imagen por defecto junto con tu **nombre** y **apellidos**. Las pantallas del perfil de administrador tienen como temática el color **rojo**.

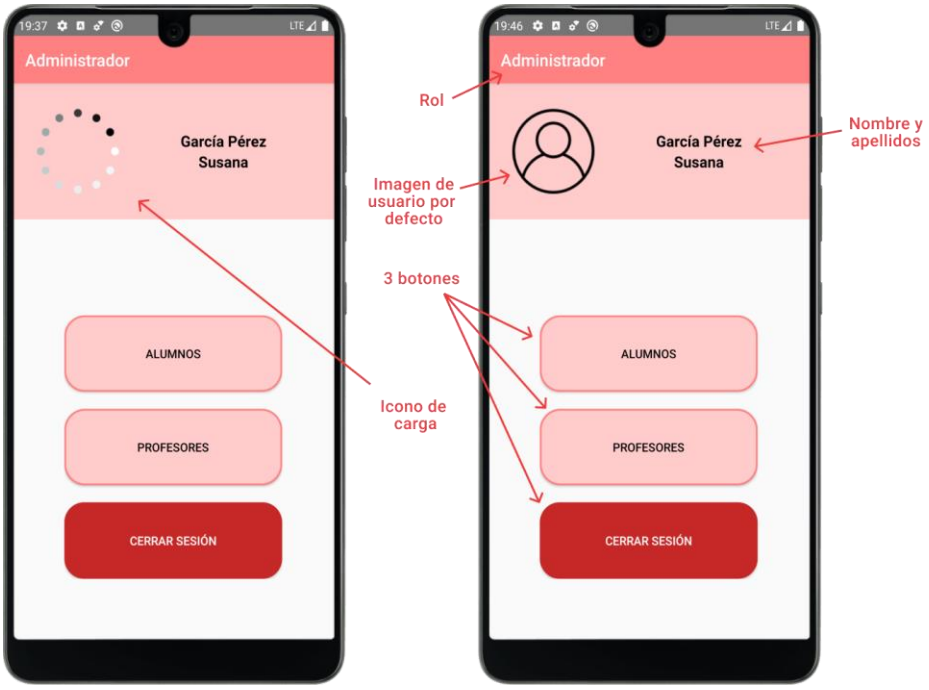


Figura 62. Perfil de Administrador: Elementos

También encontrarás 3 botones con funciones diferenciadas:

- **Alumnos.** Te llevará a la pantalla del listado de alumnos registrados.
- **Profesores.** Te llevará a la pantalla del listado de profesores registrados. *(Similar a las pantallas de alumnos, solo se mencionarán las diferencias).*

- **Cerrar sesión.** Inicia el proceso de cierre de sesión para volver a la pantalla de inicio.

En el caso en el que desees cerrar sesión, aparecerá una ventana en la que deberás confirmar la acción como puedes observar en la Figura 63.

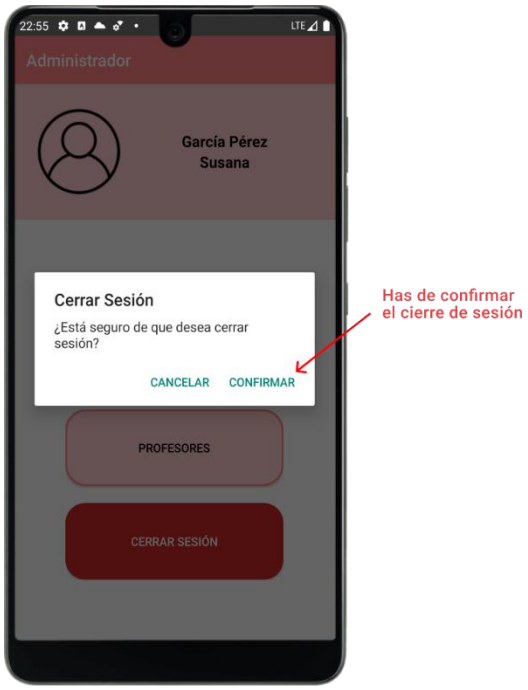


Figura 63. Perfil de Administrador: Cerrar Sesión

Por último, si giras la pantalla, la disposición de los elementos cambiará ligeramente, permitiéndote realizar *scrolling*, como puedes observar en la Figura 64.

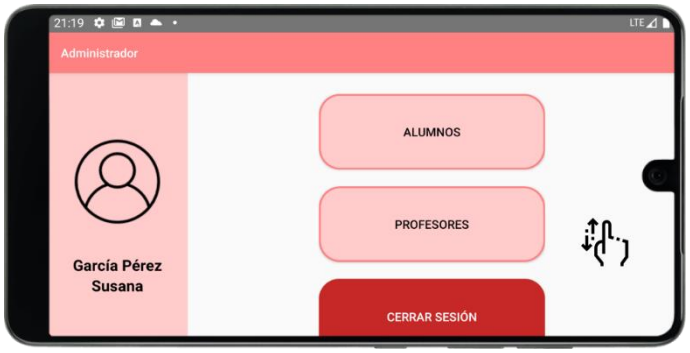


Figura 64. Perfil de Administrador: Giro de pantalla

Pantalla de Alumnos

En la pantalla de alumnos, podrás acceder al listado de **alumnos totales**, registrados una vez se complete el acceso a la base de datos. Los elementos de esta pantalla los puedes ver en la Figura 65**.**

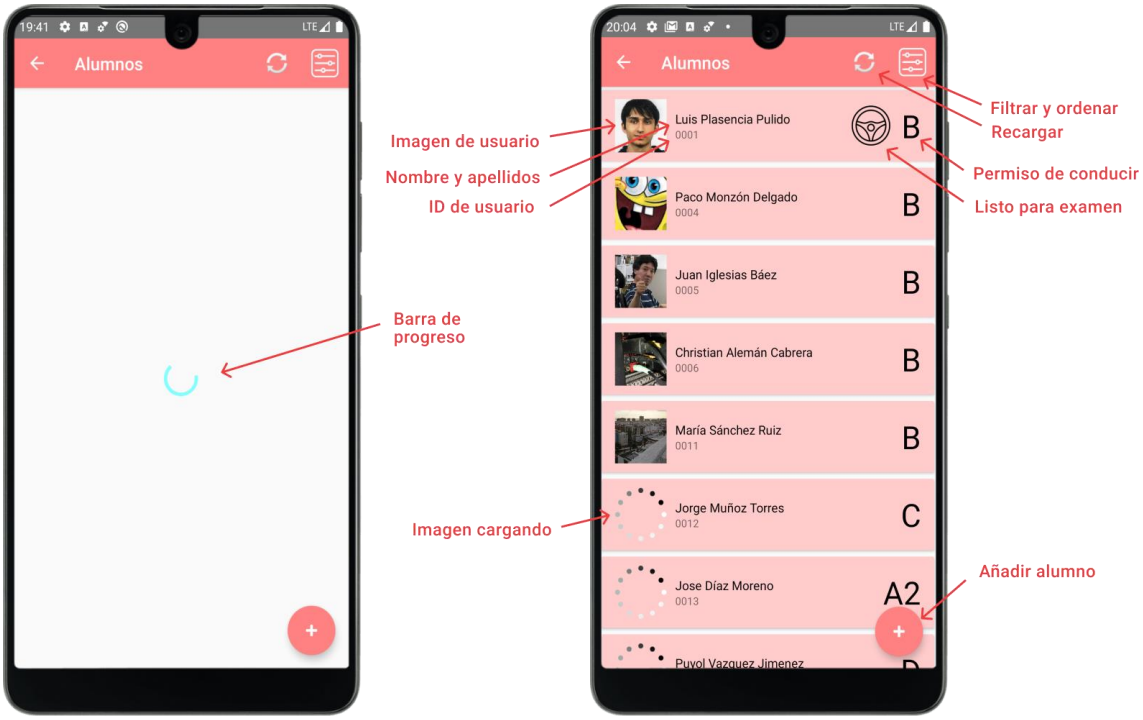


Figura 65. Alumnos: Elementos

Para cada alumno se muestra:

- **Imagen de usuario.** Aparece un icono de progreso hasta que la imagen se termine de descargar.
- **Nombre y apellidos.** Como título de la casilla.
- **Id de usuario**. Como subtítulo de la casilla.
- **Permiso de conducir.** En letras grandes en el borde derecho de cada elemento.
- **Listo o no para examen.** Si al lado del nombre de un alumno aparece el dibujo de un volante de conducir significa que dicho alumno está listo para examen.

En la parte inferior derecha de la pantalla se encuentra un botón flotante con el que podrás **añadir** un nuevo alumno.

Además, en la barra superior tendrás un botón para **actualizar** el listado y otro botón para desplegar una ventana de **filtrar y ordenar,** como puedes ver en la Figura 66.

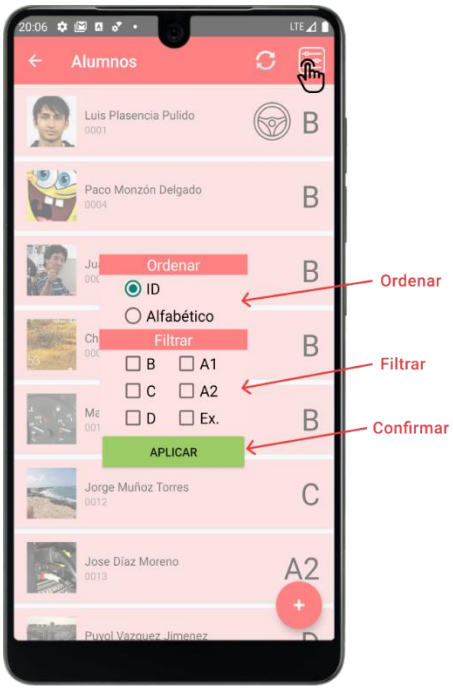


Figura 66. Alumnos: Elementos al Filtrar y Ordenar

Puedes **ordenar** los alumnos **alfabéticamente** o por **id** y **filtrarlos** en función del **permiso de conducir** o si están o no **listos para examen**. En el caso de que no hubiera coincidencias a la hora de filtrar, te aparecerá un mensaje en pantalla. En la Figura 67, puedes ver un ejemplo de la ejecución del filtrado y ordenado.

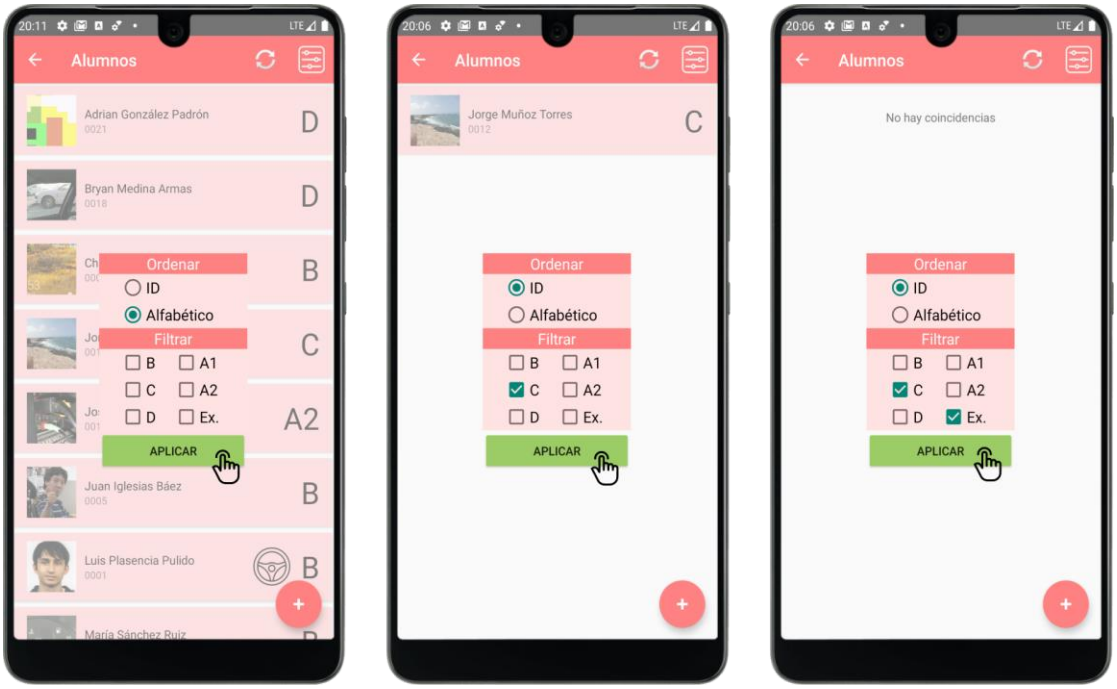


Figura 67. Alumnos: Combinaciones al Filtrar y Ordenar

Si pulsas sobre alguno de los alumnos de la lista te dirigirás a la pantalla de **detalle del alumno.** Donde podrás ver más información acerca del usuario.

Pantalla de Añadir Alumno

En la pantalla de añadir alumno, cuyos elementos puedes ver en la Figura 68, podrás crear un nuevo alumno en la base de datos introduciendo la información de usuario a través de un formulario. Siendo la selección de imagen de usuario y profesor de prácticas asignado los únicos **campos opcionales.**

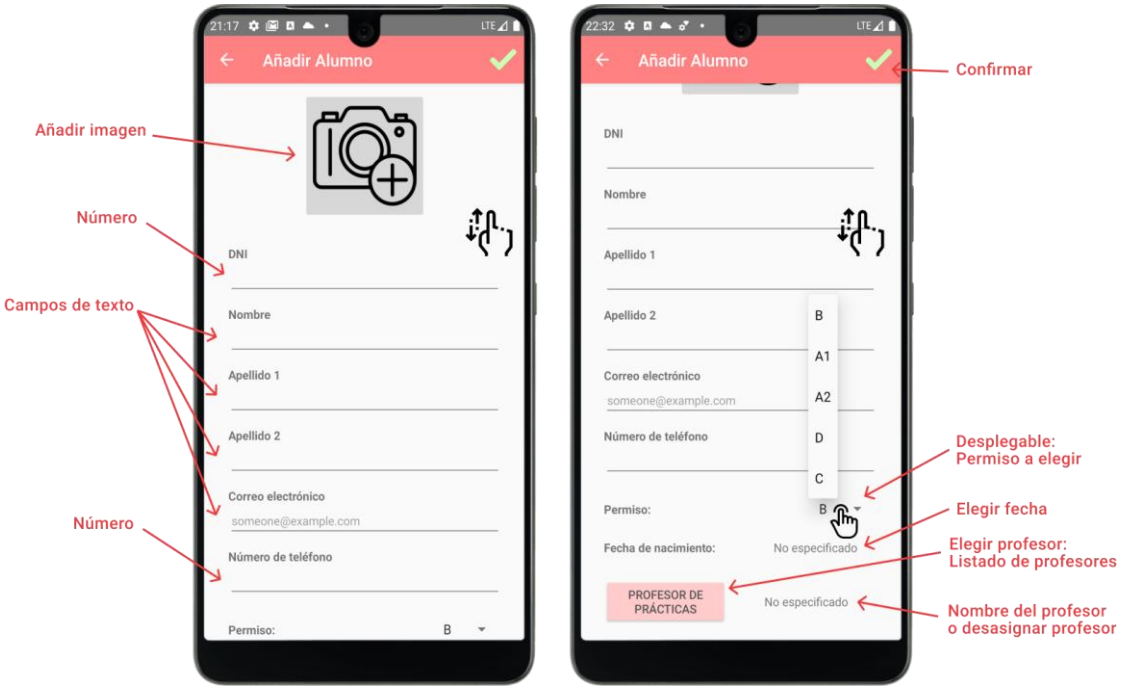


Figura 68. Añadir Alumno: Elementos

Entre los distintos campos que deberás introducir en esta pantalla, encontrarás los siguientes:

- **Añadir imagen.** Botón que te dará a elegir distintas opciones para añadir la imagen del alumno. Como pueden ser: tomar una foto directamente o elegirla de la galería. Una vez elegida la imagen, se te facilitará una herramienta para recortarla a **escala 1:1**, con opciones de **rotar** y **reflejar**. (Opcional)

En la Figura 69, puedes ver un ejemplo de ejecución de la opción de añadir imagen.

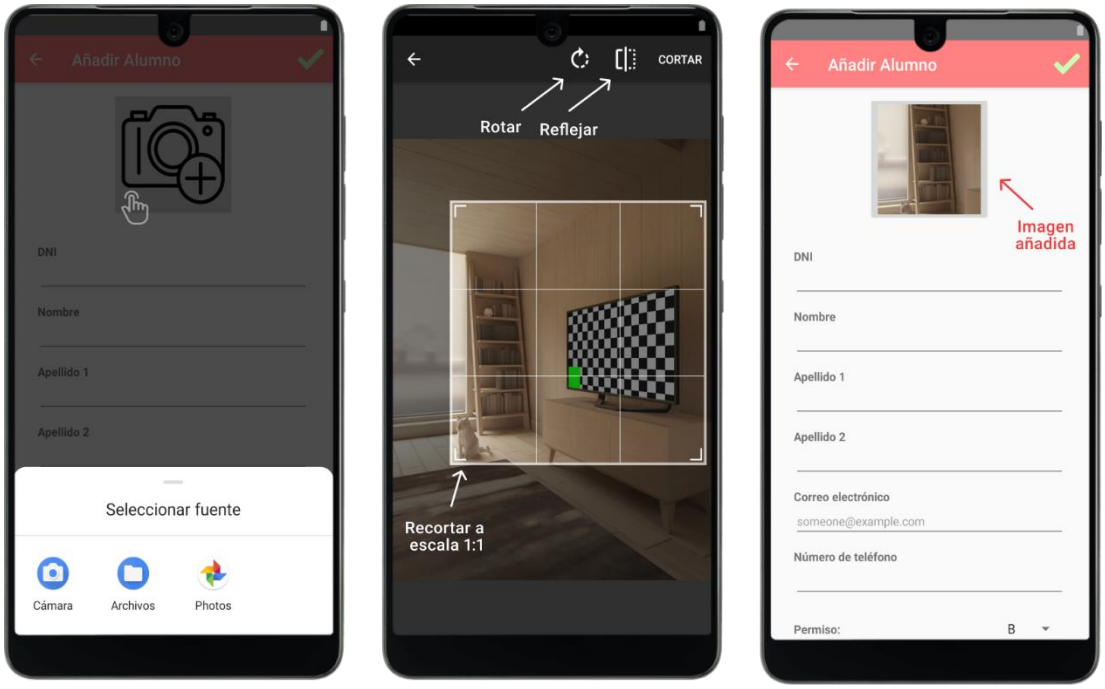


Figura 69. Añadir Alumno: Seleccionar Imagen

- **DNI.** Desplegará un teclado numérico con 8 caracteres de límite.
- **Nombre.** Campo de texto con límite de 11 caracteres.
- **Apellido 1.** Campo de texto con límite de 11 caracteres.
- **Apellido 2.** Campo de texto con límite de 11 caracteres.
- **Correo electrónico.** Campo de texto que ha de tener un formato de correo electrónico válido. Ej: alguien@ejemplo.com
- **Número de teléfono.** Desplegará un teclado numérico con 9 caracteres de límite.
- **Permiso.** Desplegable con los 5 permisos disponibles: B, A1, A2, D y C.
- **Fecha de nacimiento.** Para poder elegir la fecha de nacimiento, debes pulsar sobre el texto a la derecha. En ese momento aparecerá una ventana donde podrás elegir la fecha que desees. Al confirmar, verás la fecha que has elegido en el formulario.

En la Figura 70, puedes ver un ejemplo en el que se elige la fecha de nacimiento.

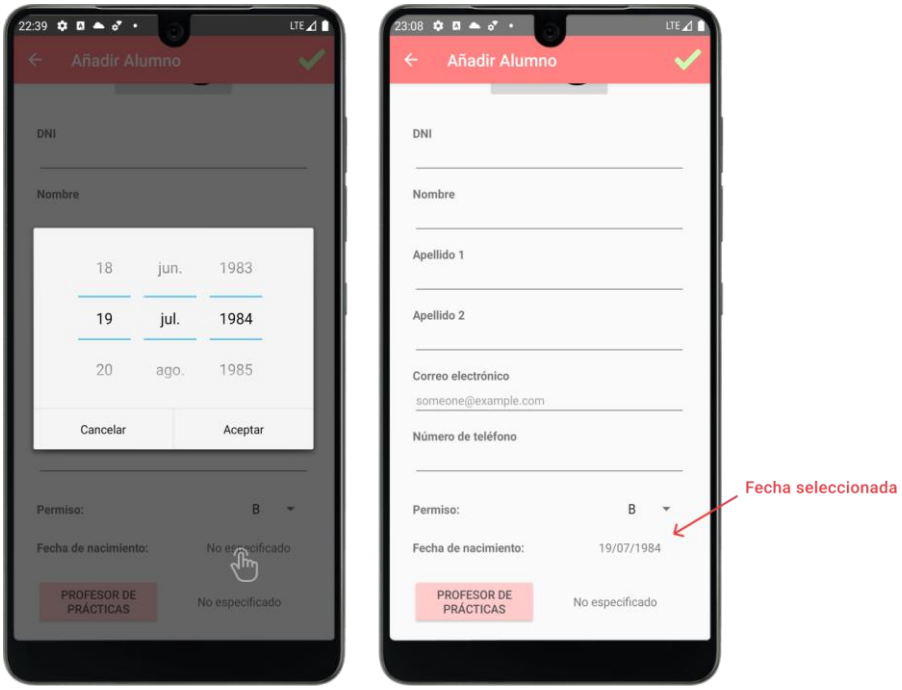


Figura 70. Añadir Alumno: Seleccionar Fecha de Nacimiento

- **Profesor de prácticas.** Botón que te dirige a la pantalla de elección del profesor de prácticas que queremos asignar a nuestro alumno. Al seleccionarlo, verás su nombre en el formulario. Si quisieras dejar de asignarlo o no especificar profesor, puedes pulsar sobre el nombre y se borrará la elección. (Opcional)

En la Figura 71, puedes ver un ejemplo en el que se selecciona el profesor de prácticas.

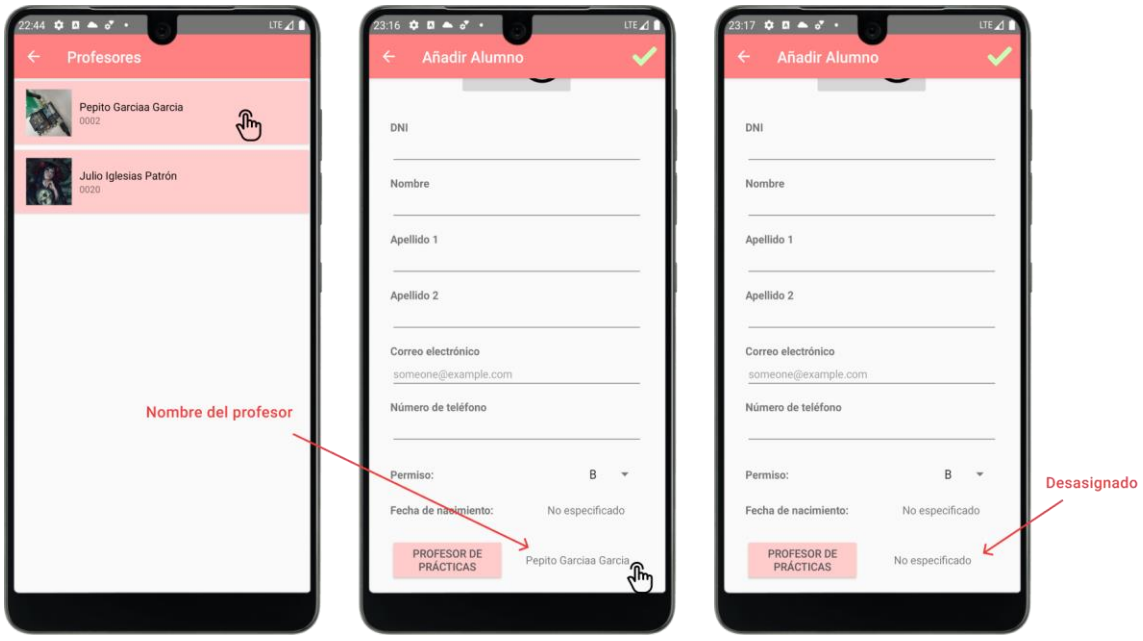


Figura 71. Añadir Alumno: Elegir Profesor de Práctica y Desasignarlo

En el momento de confirmar la creación del alumno, se bloqueará el formulario y aparecerá una **barra de progreso** que te indicará que se está añadiendo el usuario, como puedes ver en la Figura 72.

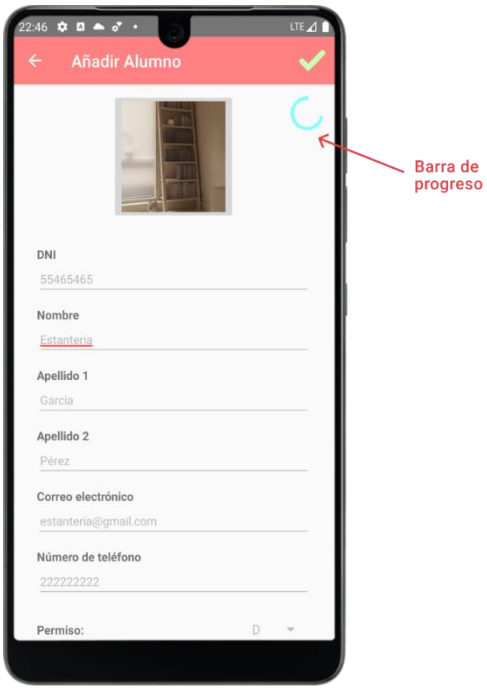


Figura 72. Añadir Alumno: Confirmar

Si faltara introducir algún campo obligatorio, los valores coinciden con alguno de los alumnos ya creados o no se cumpla el formato requerido en los registros, se te comunicaría con un **mensaje en pantalla** como los que puedes ver en la Figura 73.

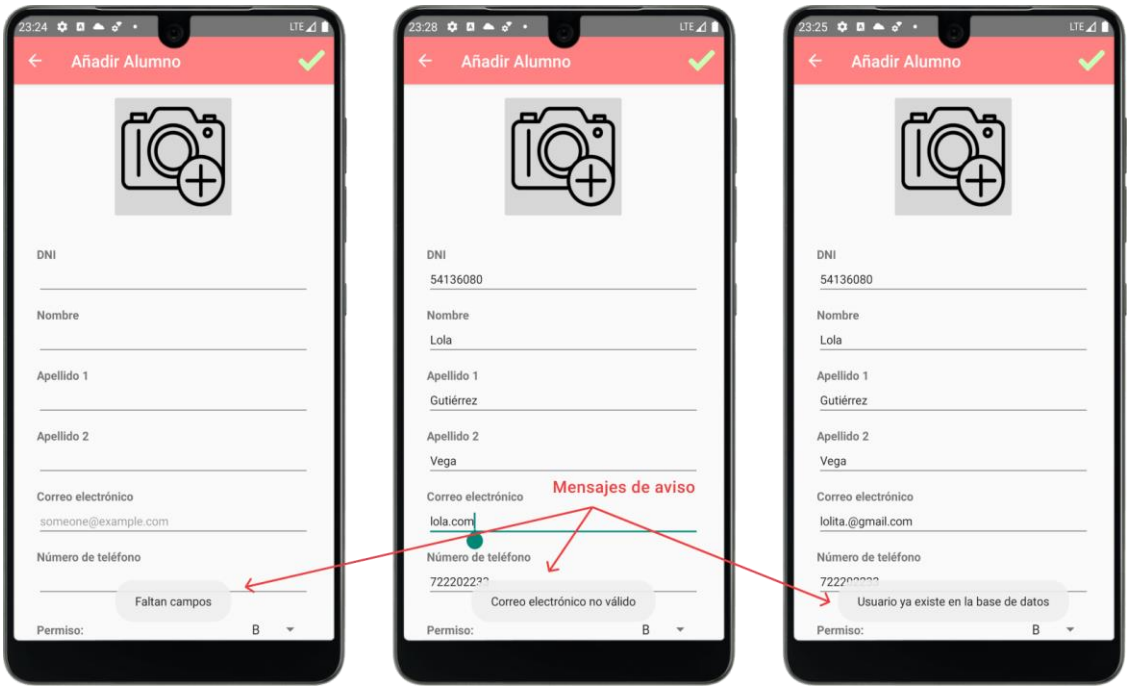


Figura 73. Añadir Alumno: Mensajes de Aviso

Si todo ha ido bien, una vez añadido el alumno se te abrirá la **aplicación de correo electrónico** que tengas instalada en el dispositivo con un mensaje personalizado con el que puedes informar al mismo de sus credenciales de inicio de sesión, como puedes ver en la Figura 74.

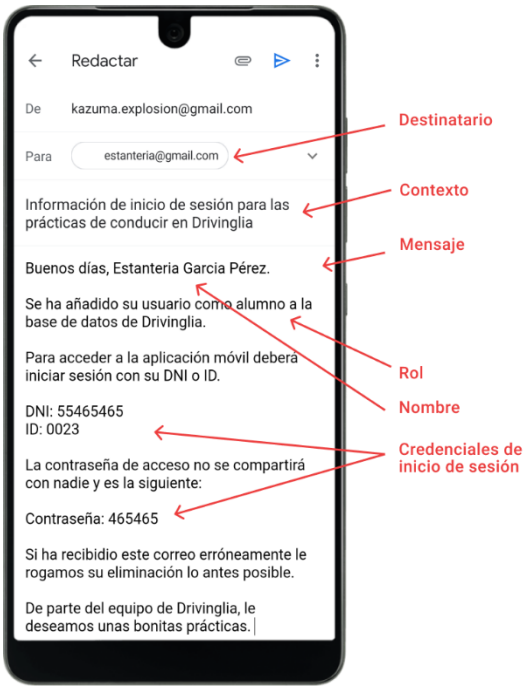


Figura 74. Añadir Alumno: Enviar Credenciales por Correo

Una vez enviado el mensaje, la aplicación volverá a la pantalla del **listado de alumnos** y actualizará la información en pantalla.

*(El funcionamiento de los formularios de las pantallas de **editar alumno** y **añadir** y **editar profesor** es análogo al de **añadir alumno** así que solo se comentarán los cambios a tener en cuenta en estas pantallas).*

Pantalla de Detalle del Alumno

En la pantalla del detalle, que puedes ver en la Figura 75, del alumno podrás ver la información de usuario de dicho alumno.

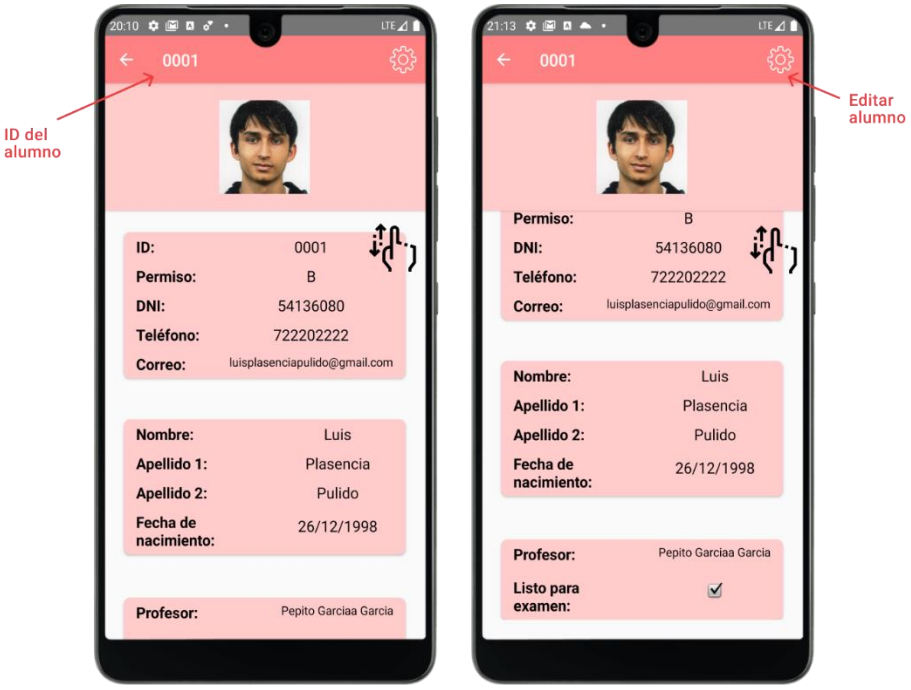


Figura 75. Detalle del Alumno: Elementos

Los campos que se mostrarán son los siguientes:

- **ID, permiso** de conducción del que está matriculado, **DNI, teléfono móvil** y **correo electrónico.**
- **Nombre, apellidos** y **fecha de nacimiento.**
- **Nombre del profesor** de prácticas asignado e imagen que nos indica si el alumno está preparado o no para realizar el **examen de conducir.**

Si giras la pantalla la disposición de los elementos cambiará levemente, como puedes ver en la Figura 76.

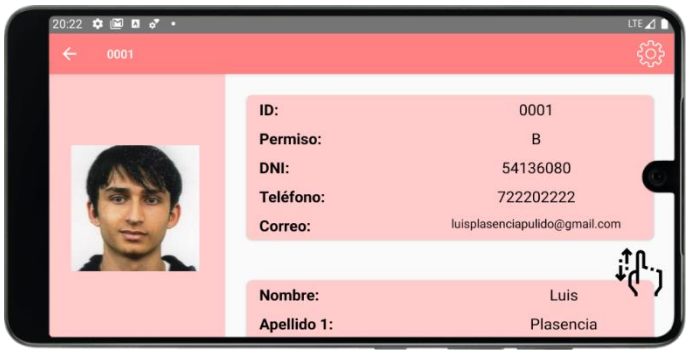


Figura 76. Detalle del Alumno: Giro de Pantalla

En la barra de herramientas superior se te facilitará un botón con forma de engranaje que te dirigirá a la pantalla de **editar alumno**. Donde podrás modificar los datos de dicho usuario.

Pantalla de Editar Alumno

En esta pantalla podrás modificar la información del alumno. Sus elementos se muestran en la Figura 77.

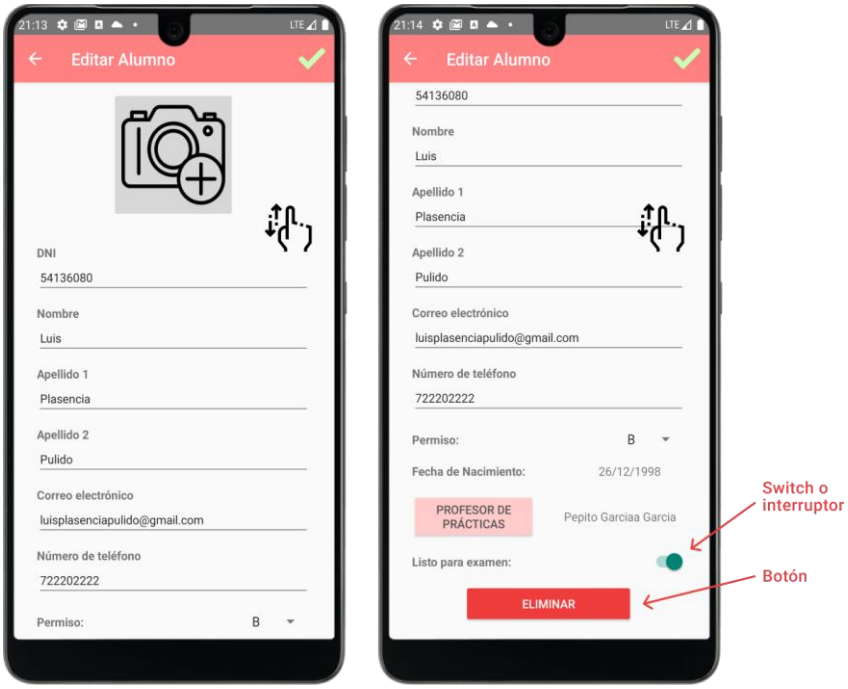


Figura 77. Editar Alumno: Elementos

Cuenta un formulario muy parecido al de la pantalla de añadir alumno, pero cuenta con algunos campos extra:

- **Listo para examen.** Podrás indicar si el alumno está listo o no para realizar el examen de conducir a través de un *switch* o interruptor que se activa y desactiva al pulsarlo.
- **Eliminar alumno.** Botón con el que podrás borrar el alumno de la base de datos junto con toda su información. Al ser una acción delicada, te aparecerá una ventana de confirmación como la que podrás ver en la Figura 78.

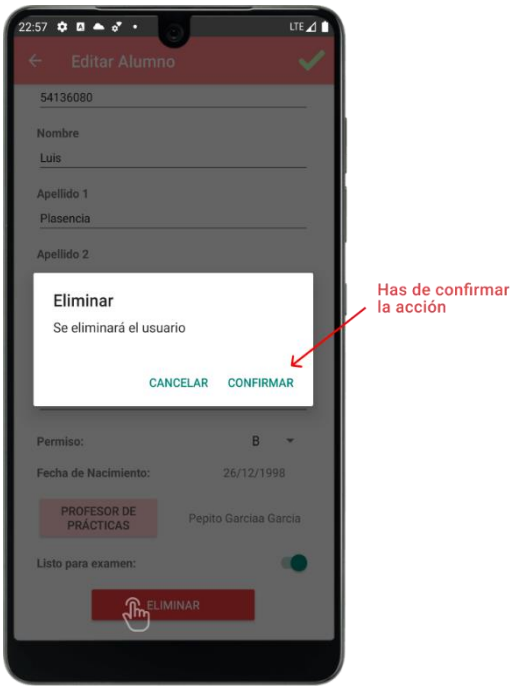


Figura 78. Editar Alumno: Eliminar

Al confirmar los cambios, la aplicación volverá a la pantalla del **detalle de alumno** tras un tiempo en el que accederá a la base de datos y podrás apreciar los cambios. Esta información también se actualizará en el **listado de alumnos**.

Pantalla de Profesores

En esta pantalla, podrás apreciar el listado de profesores registrados. A diferencia con la pantalla de **Alumnos**, esta vez no tendrás la opción de **filtrar y ordenar** y solo aparecerá la **imagen de usuario**, **nombre y apellidos** e **ID** de cada profesor, como puedes en la Figura 79.

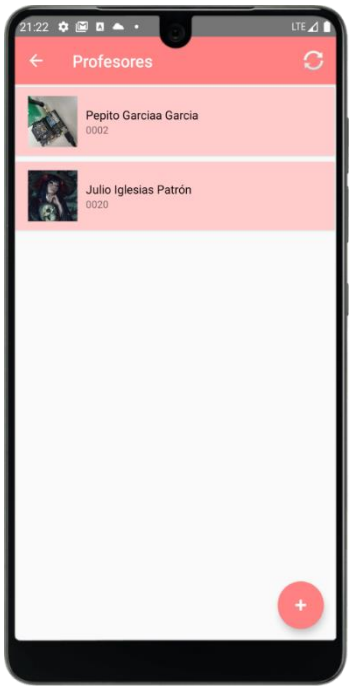


Figura 79. Profesores: Elementos

Pantalla de Añadir Profesor

Pantalla en la que podrás crear un nuevo profesor. Tiene menos campos que el formulario de la pantalla de **añadir alumno** ya que la información del profesor es reducida en comparación, como puedes ver en la Figura 80.

Figura 80. Añadir Profesor: Elementos

Pantalla de Detalle del Profesor

En la pantalla del detalle del profesor, que puedes ver en la Figura 81, podrás observar la información de usuario de dicho profesor.

Figura 81. Detalle del Profesor: Elementos

En este caso hay un botón extra para acceder al **listado de alumnos de prácticas de dicho profesor** y su detalle. Hay que comentar que en esta ocasión **no** estarán presentes los botones de **actualizar** listado y **editar alumno** en el detalle, como puedes ver en la Figura 82.

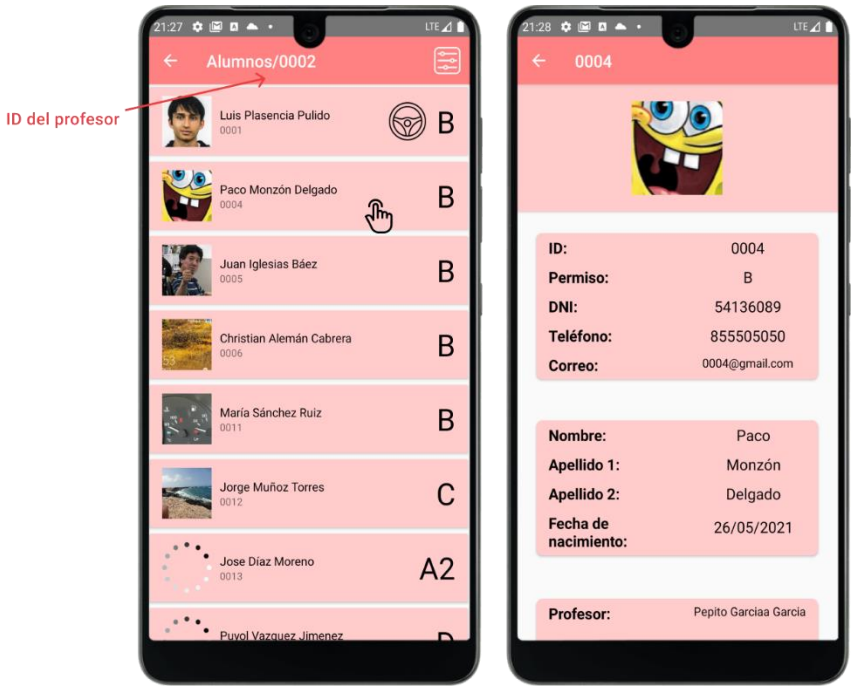


Figura 82. Detalle del Profesor: Alumnos Asignados y su detalle

Si el profesor no tuviera alumnos asignados, aparecería un aviso en pantalla como el que puedes ver en la Figura 83.

Figura 83. Detalle del Profesor: Ningún Alumno Asignado

Pantalla de Editar Profesor

En esta pantalla podrás modificar la información del profesor. El formulario tiene menos campos que en la pantalla de **editar alumno** ya que la información del profesor es reducida en comparación, como puedes ver en la Figura 84.

Figura 84. Editar Profesor: Elementos

6.1.3. Profesor

El perfil del profesor podrá acceder a la información de los alumnos que tenga asignados y generar nuevas prácticas de conducir para cada uno de ellos.

Pantalla de Perfil de Profesor

Una vez inicias sesión con una cuenta de profesor, entrarás en la pantalla de **perfil** que puedes ver en la Figura 85. Las pantallas del perfil de profesor tienen como temática el color **azul**.

(Las pantallas de Perfil de administrador, profesor y alumno son muy similares por lo que solo se comentarán las diferencias).

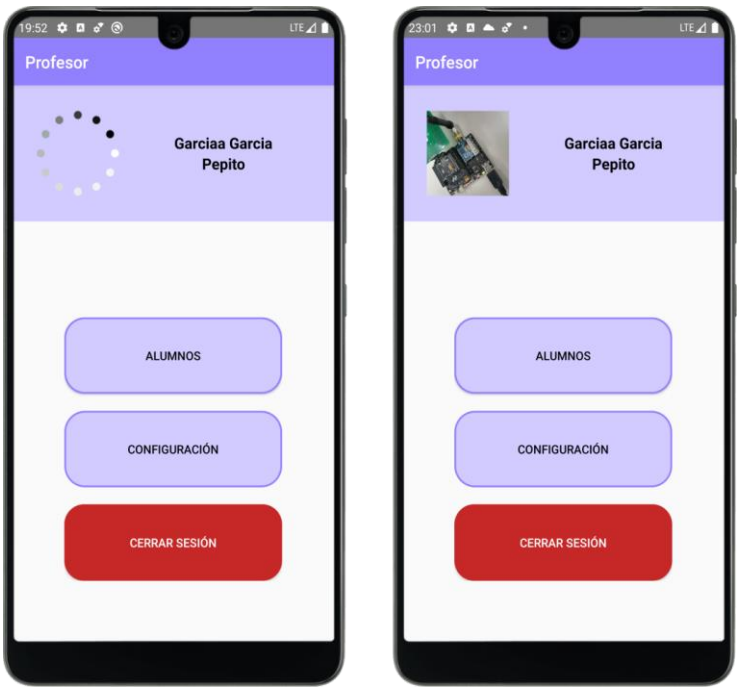


Figura 85. Perfil de Profesor: Elementos

Esta vez, el **primer botón** te llevará a la pantalla del **listado de alumnos de dicho profesor** y el segundo botón, en futuras actualizaciones, te permitirá acceder a la configuración de la cuenta.

Pantalla de Alumnos

En la pantalla de alumnos de la Figura 86, podrás acceder al listado de **alumnos de dicho profesor** una vez se complete el acceso a la base de datos.

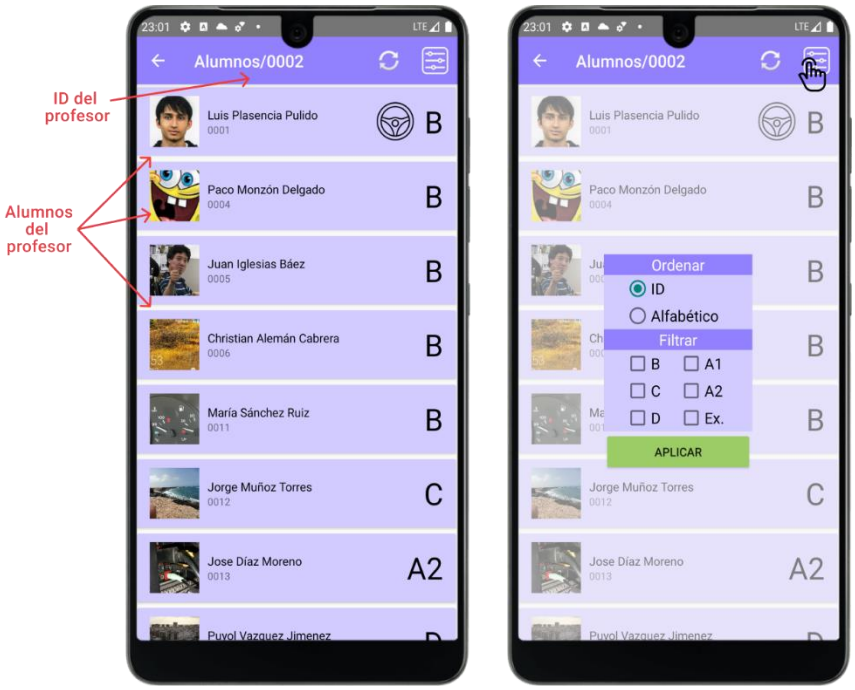


Figura 86. Alumnos del Profesor: Elementos

A diferencia del listado de alumnos del perfil de administrador, esta vez no tendrás la opción de añadir alumnos y en la barra superior aparecerá el id de la cuenta del profesor. La función de filtrado y ordenado funciona de la misma manera.

Si presionas sobre alguno de los alumnos de la lista, la aplicación navegará al **detalle del alumno**, donde podrás comenzar la práctica.

Pantalla de Detalle del Alumno y Configuración

En la pantalla del detalle del alumno podrás ver la información de usuario de dicho alumno. A diferencia de la pantalla de editar alumno del perfil de administrador, esta vez tendrás dos botones en la barra de herramientas superior.

El primer botón te dirigirá a la pantalla de **configuración del alumno**, donde podrás editar ciertos registros del alumno. En un principio, solo tendrás la opción de editar si el alumno está **listo o no listo para realizar examen.** Al confirmar, la aplicación volverá a la pantalla anterior y se reflejarán los cambios en el detalle del alumno.

En la Figura 87, puedes ver un ejemplo de navegación entre las pantallas de detalle de alumno y configuración del alumno.

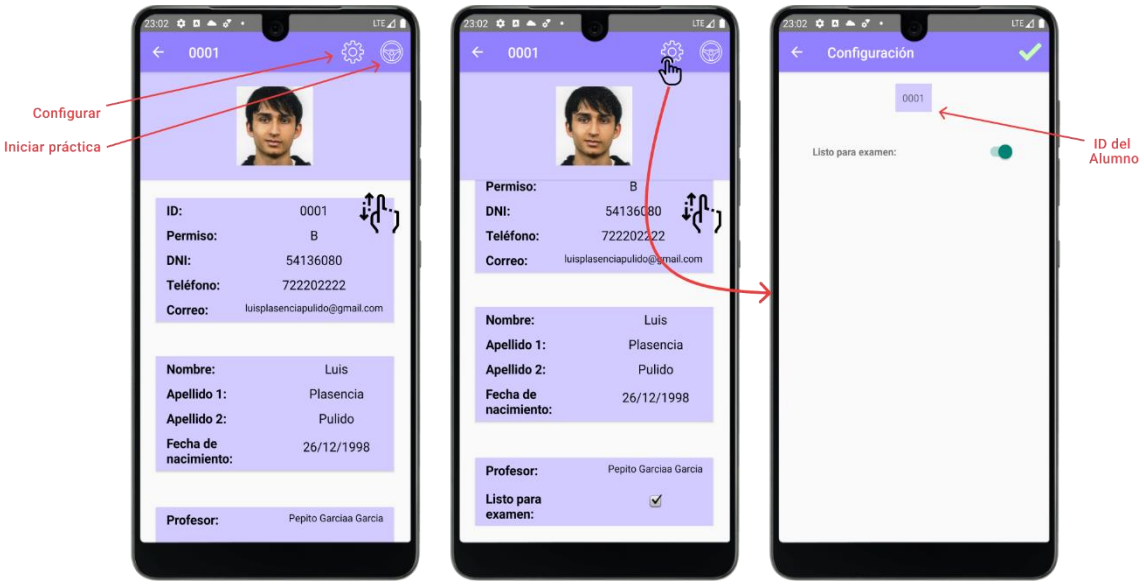


Figura 87. Detalle de Alumno del Profesor: Elementos y navegación

El segundo botón tiene forma de **volante** y te dirigirá al **inicio de la práctica** siempre y cuando aceptes los **permisos para acceder a la ubicación** del dispositivo. Te aparecerá un aviso en pantalla si no aceptas los permisos de ubicación, como puedes ver en la Figura 88.

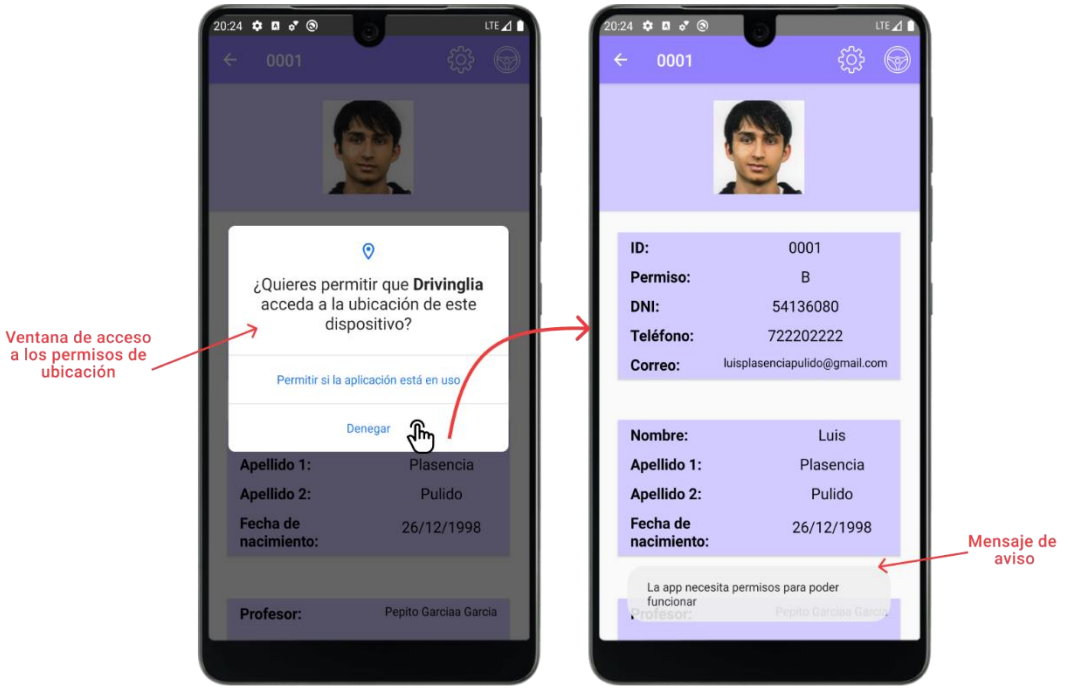


Figura 88. Detalle de Alumno del Profesor: Permisos de Ubicación

Pantalla de Práctica

En esta pantalla, comenzarás una nueva práctica de conducir para el alumno que hayas seleccionado. Con el móvil apoyado siempre en un dock de forma segura, el mapa se centrará inicialmente en la ubicación de **inicio de práctica**, que se indicará con un marcador azul. Cada 100 metros el mapa se volverá a centrar en la posición del vehículo y trazará el recorrido seguido en la práctica en el mapa. Pulsando sobre el recorrido puedes cambiar el estilo del trazado.

En la Figura 89, podrás ver los distintos elementos de la pantalla de práctica y cómo se genera el trazado en el mapa una vez el vehículo se desplaza.

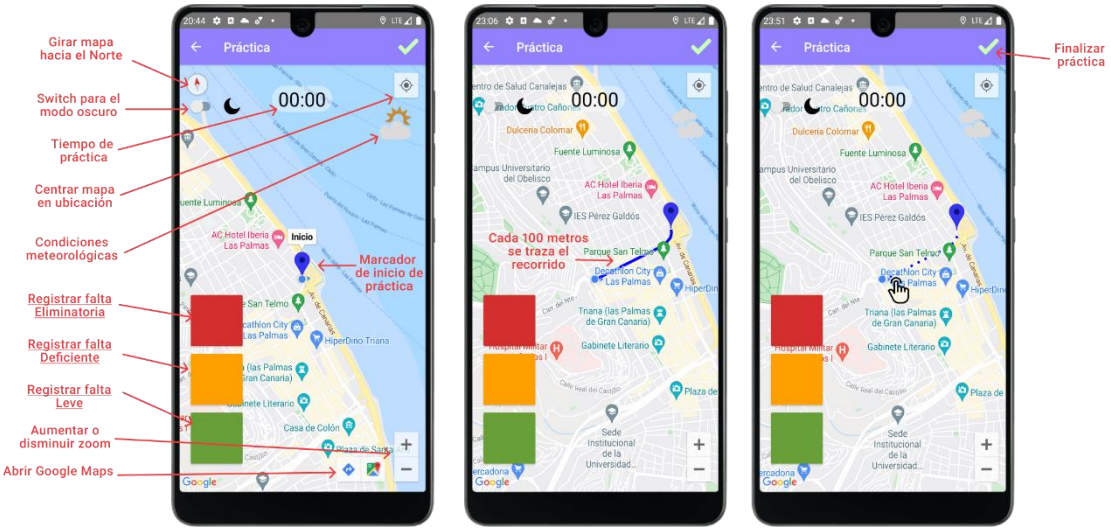


Figura 89. Práctica: Elementos y desplazamiento

A continuación, se mencionan las distintas funciones de la pantalla de práctica:

- **Orientar el mapa.** Pulsando la brújula superior izquierda podrás orientar el mapa hacia el norte cuando esté descentrado.
- **Modo noche.** Podrás activar y desactivar el ***switch*** de la parte superior izquierda para cambiar los colores del mapa y sus elementos, como puedes ver en la Figura 90.

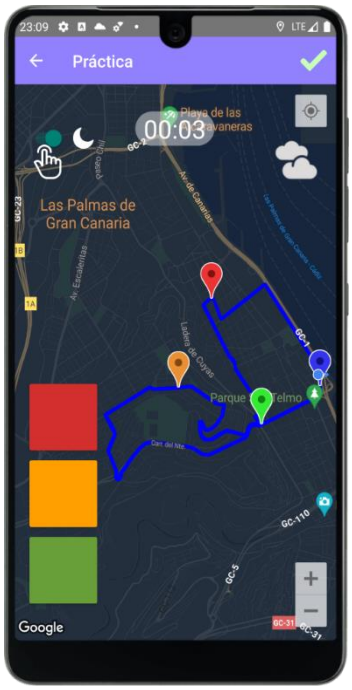


Figura 90. Práctica: Modo noche

- **Tiempo de práctica.** Aparece un contador de las horas y minutos de práctica en la parte superior central.
- **Centrar mapa.** Con el botón de la parte superior derecha podrás centrar el mapa en la ubicación actual.
- **Condiciones meteorológicas.** Un icono te mostrará el clima actual tras unos segundos de comenzar la práctica. Puede tomar los valores que vemos en la Tabla 9.

Condiciones meteorológicas	Icono
Granizo	
Alta nubosidad	

Lluvia pesada	
Ligeramente nublado	
Lluvia ligera	
Lluvia espontánea	
Nieve	
Aguanieve	
Tormenta	

Tabla 9. Posibles Condiciones Meteorológicas

- **Aumentar/disminuir zoom.** Los iconos de menos y más de la parte inferior derecha te permitirán hacer zoom de la manera más cómoda posible sin distracciones.

- **Registrar falta.** Tres botones grandes de color **rojo, naranja y verde** te permitirán **registrar una falta** eliminatoria, deficiente o leve en la ubicación actual. Se abrirá entonces un listado traslúcido de secciones y faltas donde podrás especificar el detalle de la infracción, como puedes ver en el ejemplo de ejecución de la Figura 91.

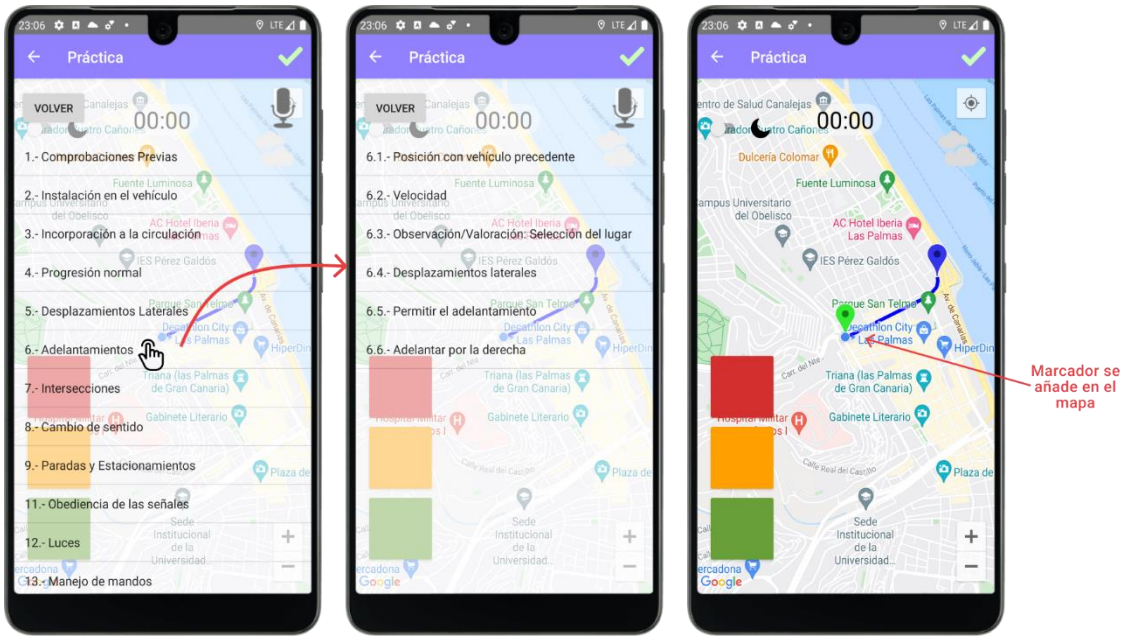


Figura 91. Práctica: Registrar Falta

Con el botón de ***volver*** de la parte superior, puedes cambiar de listado o cancelar la falta. Por otro lado, en la parte superior derecha aparecerá un icono de un **micrófono** con el que podrás registrar el código de la falta de manera oral para **evitar las distracciones,** cuya ejecución puedes ver en la Figura 92.

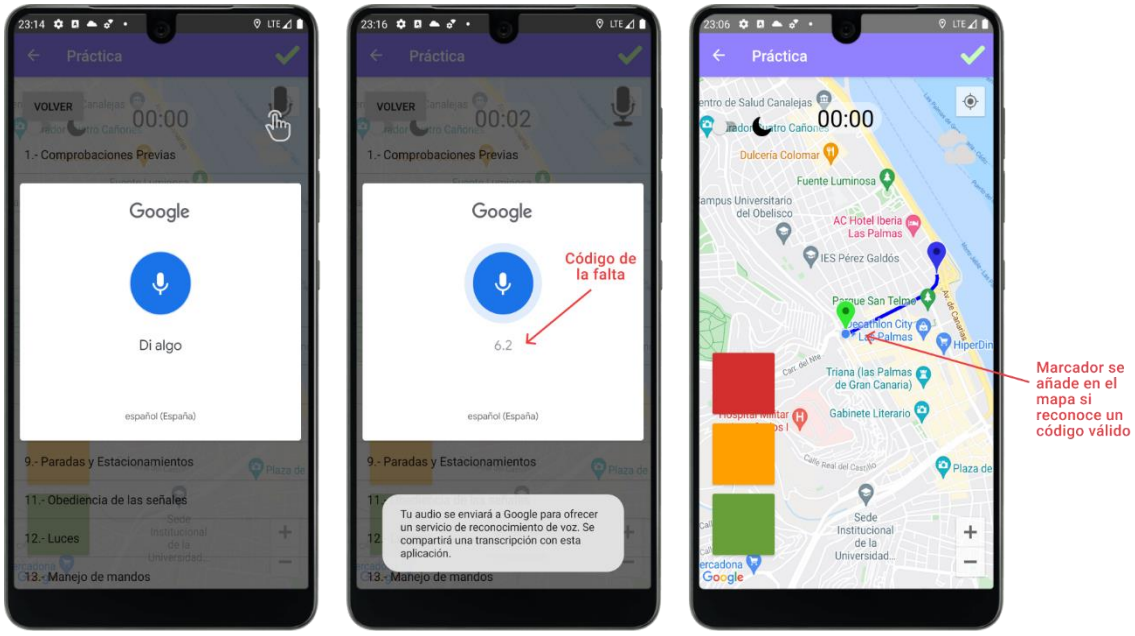


Figura 92. Práctica: Manos Libres

Una vez especificada la falta, aparecerá un **marcador** en el mapa con el color de la gravedad de la falta. Los marcadores **verdes** corresponderán a las faltas leves, los **naranjas** a las faltas deficientes y los rojos a las faltas **eliminatorias**.

Si pulsas sobre el marcador de alguna falta, aparecerá sobre este el **detalle de la incidencia** y a la derecha una pequeña ventana con información de la posición del vehículo en el momento de la falta, como la que puedes ver en la Figura 93.

Figura 93. Práctica: Detalle Marcador

Esta ventana contiene los siguientes campos:

- **Tiempo de práctica** en el que fue registrada la falta en horas y minutos.
- **Velocidad del vehículo** en el momento de la infracción en km/h.
- **Altitud** sobre el nivel del mar en metros.
- **Precisión** de la posición en metros.
- **Botón de modificar falta.** Se te abrirá el listado de faltas de dicha gravedad para poder especificar otro código de falta. En la Figura 94, puedes ver un ejemplo de ejecución en el que el usuario modifica la falta del marcador.

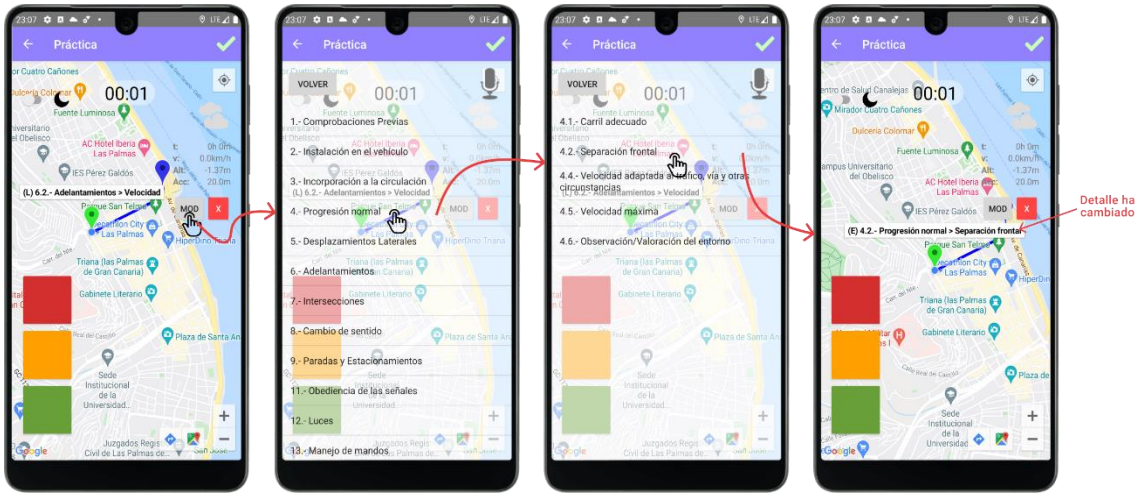


Figura 94. Práctica: Modificar Marcador

- **Botón de eliminar falta.** Nos aparecerá una ventana, como la que puedes ver en la Figura 95, en la que deberemos confirmar la eliminación de la falta y del marcador asociado.

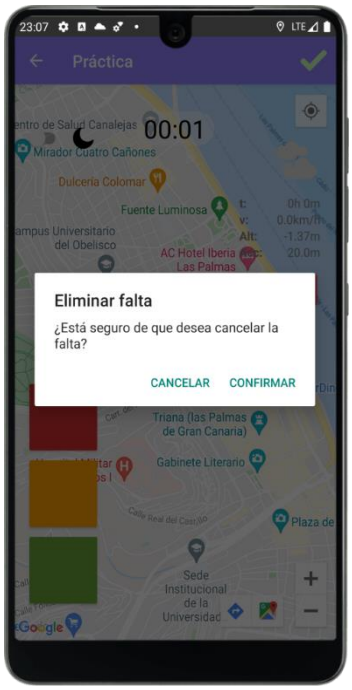


Figura 95. Práctica: Eliminar Falta

Si pulsas sobre el detalle de la falta, la aplicación mostrará el **Street View o vista callejera** de la posición de la incidencia, como puedes ver en la Figura 96.

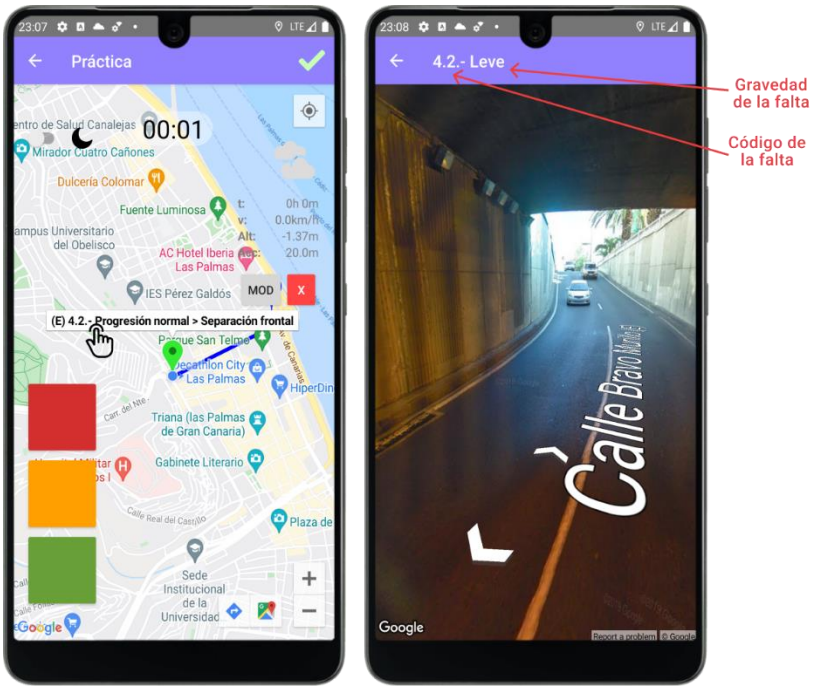


Figura 96. Práctica: Street View

Si en algún momento quieres **cancelar la práctica** basta con pulsar el botón de atrás. Se abrirá una ventana de confirmación, como la que puedes ver en la Figura 97 y volverás al detalle del alumno.

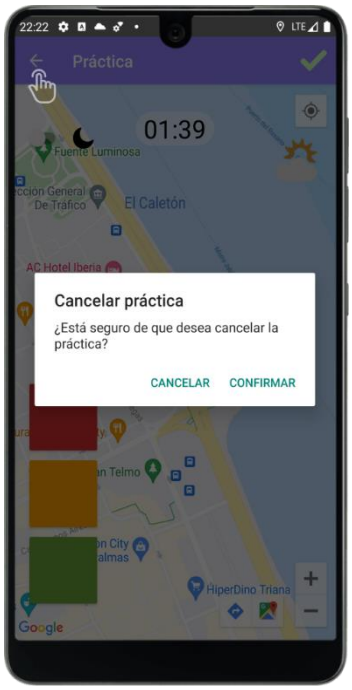


Figura 97. Práctica: Cancelar Práctica

Una vez termines el recorrido de la práctica, podrás presionar sobre el botón de confirmar de la barra de herramientas superior, como puedes ver en la Figura 98, para dirigirte a la pantalla **de anotaciones de la práctica**.

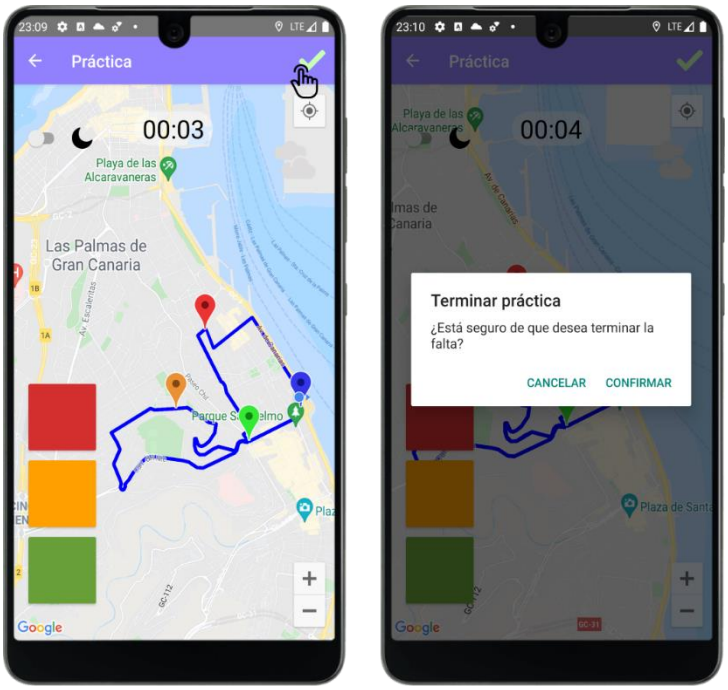


Figura 98. Práctica: Terminar Práctica

Pantalla de Anotaciones

Una vez finalizada la práctica, podrás ver la información recabada de la misma y realizar anotaciones, como puedes ver en la Figura 99. La información de esta pantalla le aparecerá al alumno cuando acceda a la práctica.

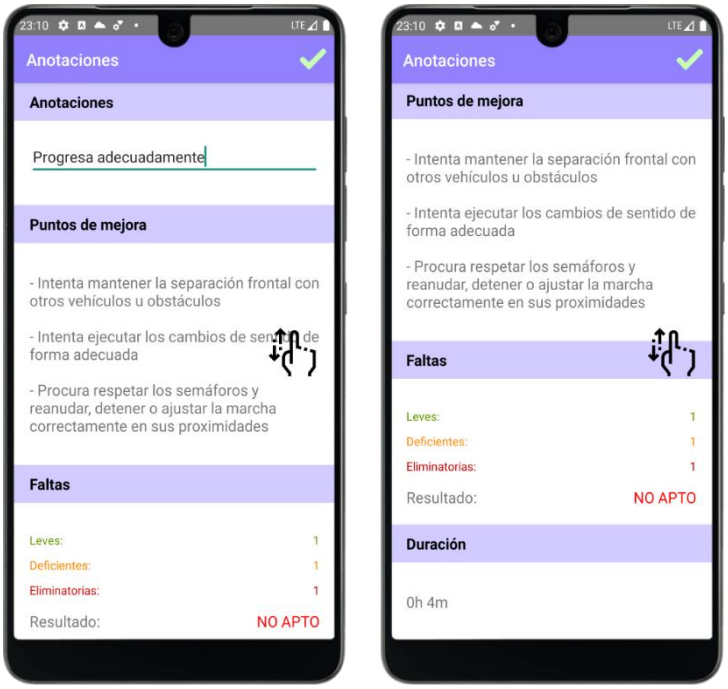


Figura 99. Anotaciones: Elementos

Los campos que aparecerán son los siguientes:

- **Anotaciones.** Campo de texto en el que podrás escribir una nota o apunte acerca de la práctica.
- **Puntos de mejora.** Generados automáticamente por la aplicación en función de las faltas cometidas.
- **Faltas.** Balance de faltas y calificación obtenida si fuera un examen de conducir.
- **Duración.** Tiempo que ha durado la práctica en minutos y horas.

Una vez presiones el botón de confirmar, la aplicación volverá a la pantalla de **detalle del alumno** y la práctica habrá finalizado.

6.1.4. Alumno

El perfil de alumno podrá acceder a la información de prácticas e incidencias cometidas.

Pantalla de Perfil de Alumno

Una vez inicias sesión con una cuenta de alumno, entrarás en la pantalla de **perfil** que puedes ver en la Figura 100. Las pantallas del perfil del alumno tienen como temática el color **verde**.

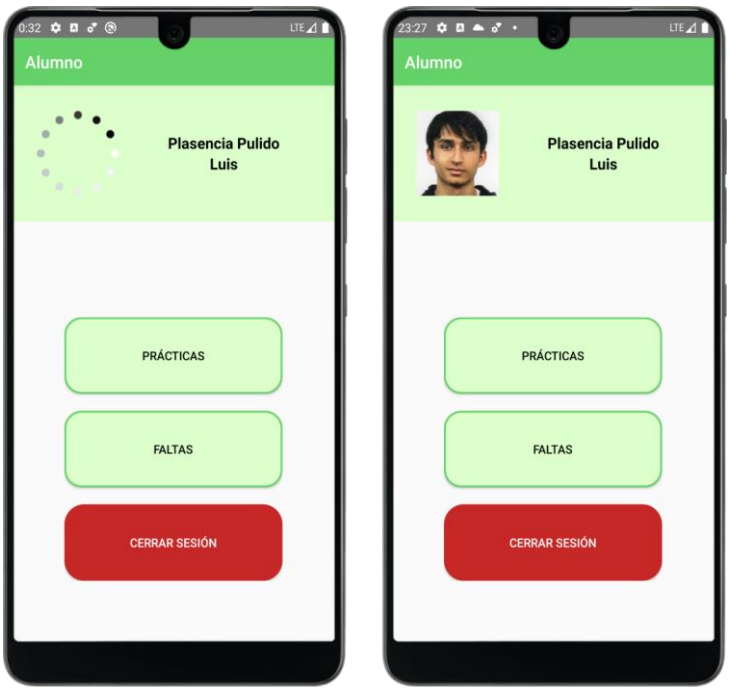


Figura 100. Perfil de Alumno: Elementos

Esta vez, el **primer botón** te llevará a la pantalla del **listado de prácticas del alumno** y el segundo botón a la pantalla de **faltas totales cometidas**.

Pantalla de Faltas Totales

Podrás ver el balance de faltas totales que has cometido a lo largo de las prácticas. Se mostrarán **3 listas desplegables** con las faltas de distinta gravedad y sus códigos. Ya que algunas faltas pueden estar repetidas se indicará entre paréntesis el número de veces que las has cometido al desplegar las listas.

En la Figura 101, puedes ver los elementos en pantalla y las listas una vez estén desplegadas.

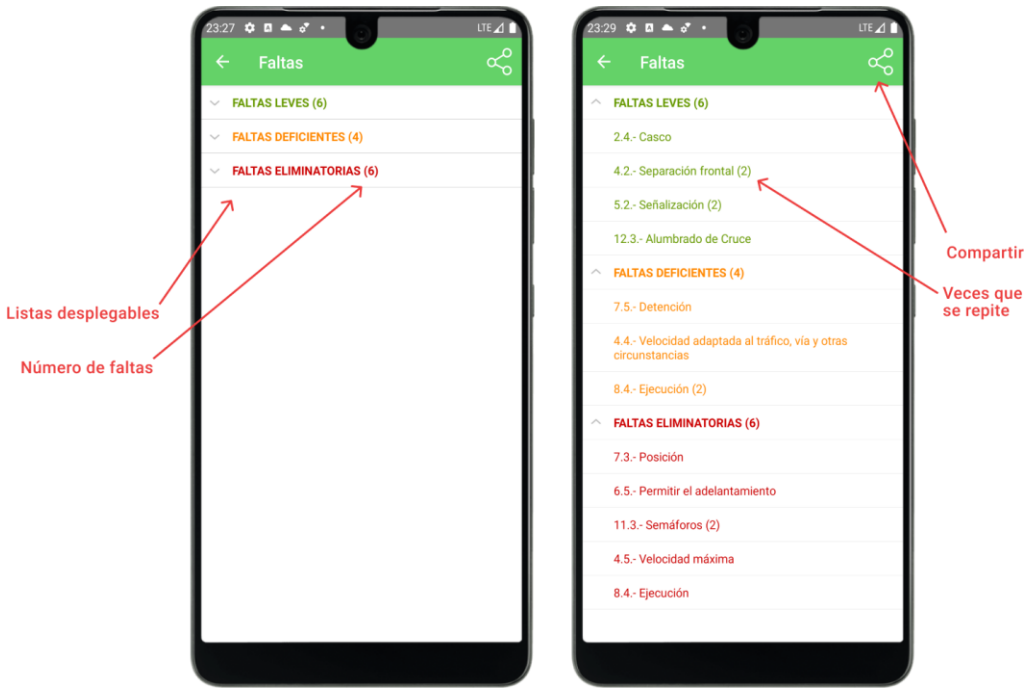


Figura 101. Faltas totales: Elementos

En la barra superior derecha tendrás un botón de **compartir** con el que podrás elegir la aplicación que desees para enviar un mensaje con los porcentajes y número de faltas cometidas dependiendo de la gravedad y el número de prácticas realizadas.

En la Figura 102, puedes ver un ejemplo de uso del botón de compartir y cómo se genera el mensaje con las faltas cometidas.

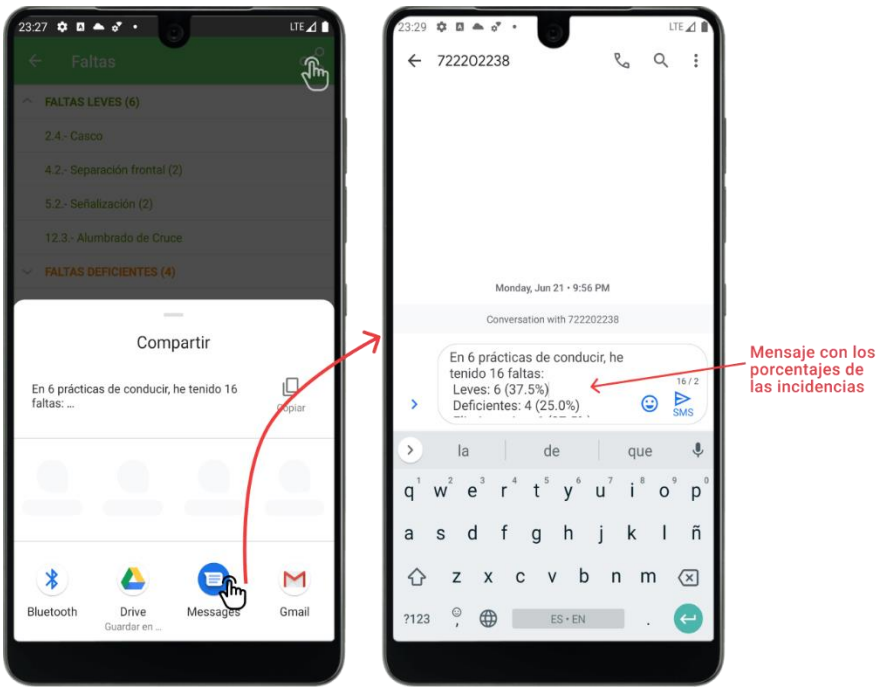


Figura 102. Faltas totales: Compartir

Pulsando alguna de las faltas desplegadas, accederás al **detalle de la falta**.

Pantalla del Detalle de la Falta

En el detalle de la falta podrás ver el motivo y la sección a la que pertenece. El color de las tarjetas depende de la gravedad de la falta.

En la Figura 103, puedes ver cómo cambia el detalle de los distintos tipos de faltas (eliminatorias, deficientes y leves).

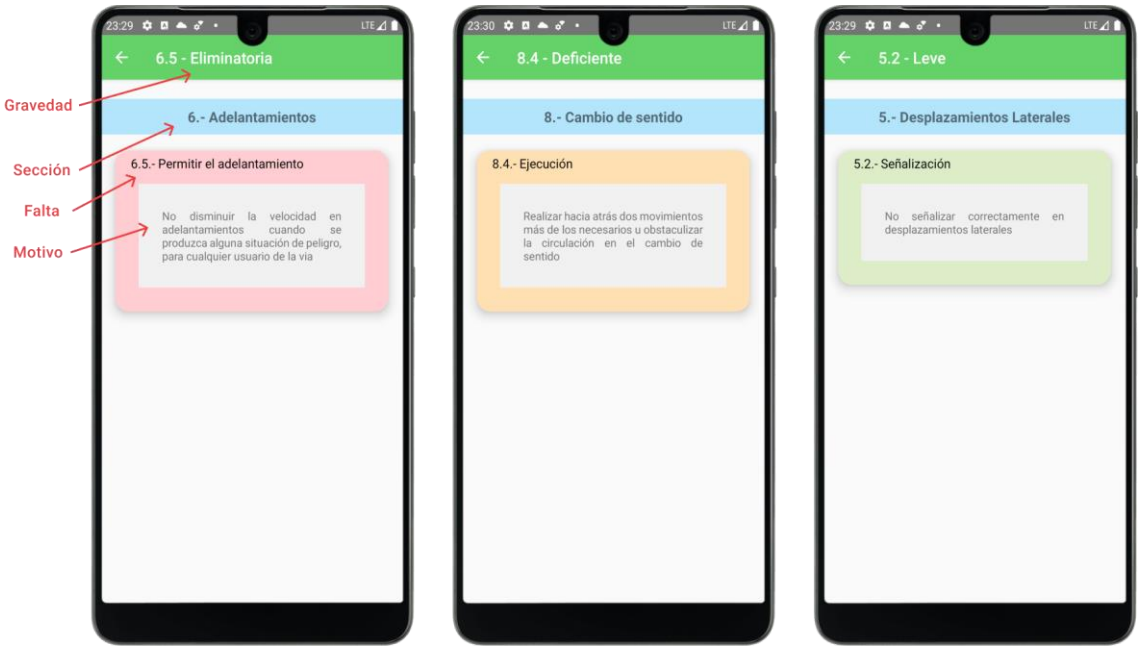


Figura 103. Detalle de la Falta: Elementos

Pantalla de Prácticas

En la pantalla de prácticas podrás ver el listado con las distintas prácticas realizadas, como puedes ver en la Figura 104.

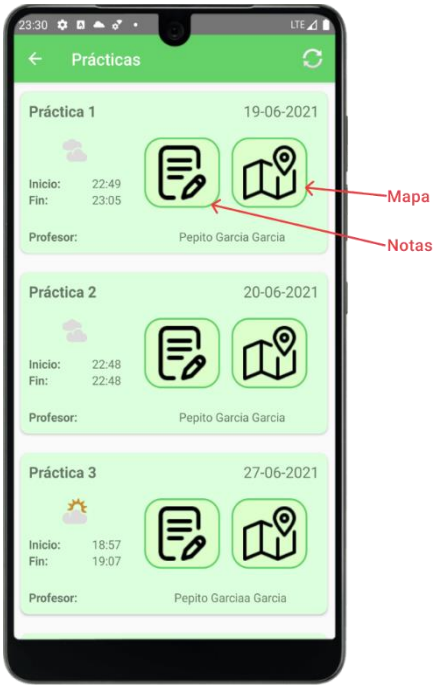


Figura 104. Prácticas: Elementos

Cuenta con un botón superior para **actualizar** el listado y para cada práctica se muestra:

- **Condiciones meteorológicas.** A través de un icono indicativo.
- **Horas de inicio y fin.** En formato 24 horas.
- **Fecha.** En la esquina superior derecha.
- **Nombre del profesor** con el que se realizó la práctica.
- **Botón de Notas.** Accederás a las **notas** de dicha práctica.
- **Botón de Mapa.** Accederás al **mapa de la práctica.**

Pantalla de Notas de la Práctica

En esta pantalla, podrás ver las **anotaciones** que hizo el profesor al final de la práctica, los **puntos** en los que tienes que **mejorar** y el **balance de faltas** con la **calificación** obtenida. La calificación puede ser **APTO** o **NO APTO** en función del número de faltas cometidas y de la gravedad. Sus elementos los puedes ver en la Figura 105.

Figura 105. Notas de la Práctica: Elementos

Pantalla de Mapa de la Práctica

En la pantalla del mapa de la práctica, que puedes observar en la Figura 106, podrás ver el **recorrido** seguido a lo largo de la práctica y las faltas cometidas en forma de **marcadores.**

Esta pantalla cuenta con todas las funciones de la pantalla de realización de prácticas del profesor como el **modo noche** y el **Street View**, pero no se podrán registrar faltas y no se accederá a la ubicación del dispositivo en ningún momento.

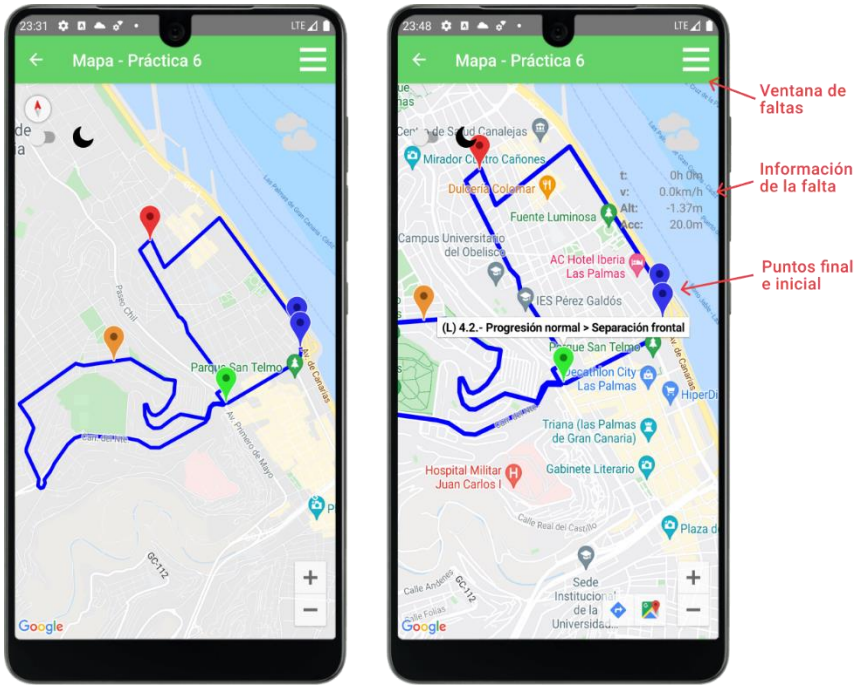


Figura 106. Mapa de la Práctica: Elementos

Esta pantalla cuenta con un botón en la barra de herramientas superior derecha con el que podrás desplegar la **ventana de faltas.**

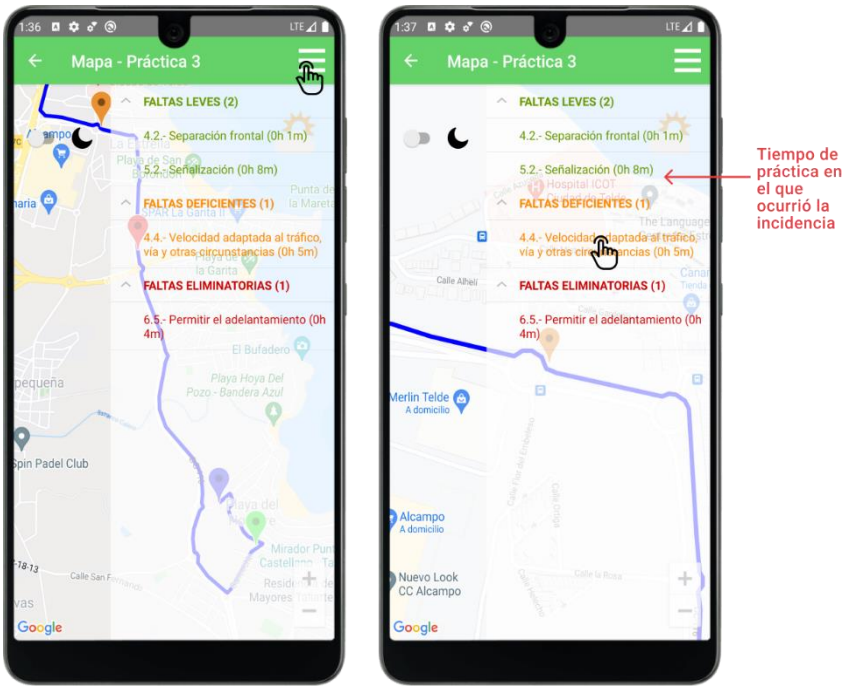


Figura 107. Mapa de la Práctica: Ventana de Faltas

La ventana de faltas te mostrará 3 **listas desplegables** muy parecidas a las de la pantalla de faltas totales pero esta vez, al pulsar sobre alguna de ellas, el mapa se centrará en el marcador correspondiente a dicha falta para **facilitar la navegación,** como puedes ver en el ejemplo de ejecución de la Figura 107. Además, aparecerá el tiempo de práctica en el que ocurrió cada incidencia en el listado.

6.2. Storyboard

En este apartado, mostramos el mapa de navegación de las pantallas según los perfiles.

6.2.1. Perfil de Administrador

El storyboard final de las pantallas del perfil de administrador lo podemos ver en la Figura 108.

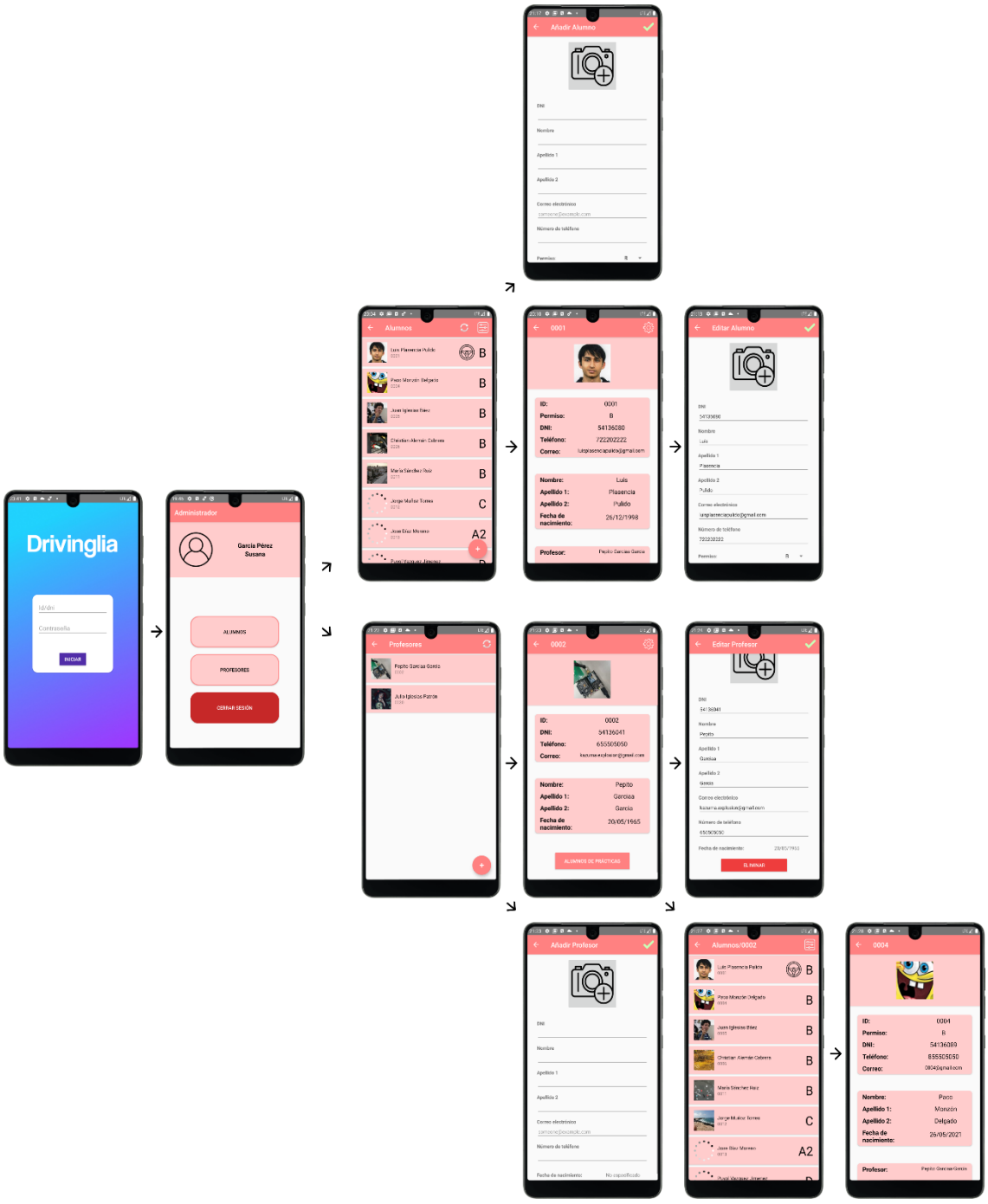


Figura 108. Storyboard: Perfil Administrador

6.2.2. Perfil de Profesor

El storyboard final de las pantallas del perfil de profesor lo podemos ver en la Figura 109.

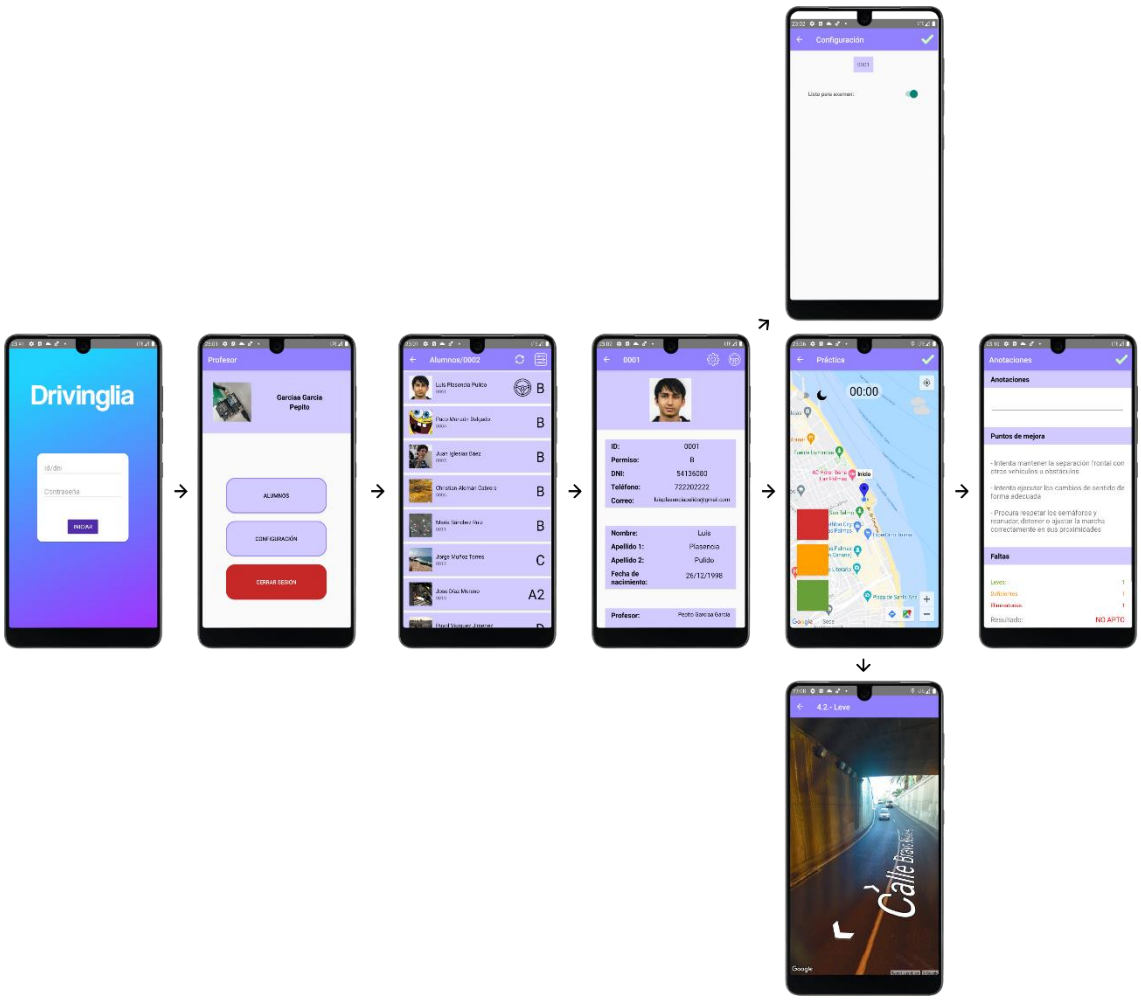


Figura 109. Storyboard: Perfil Profesor

6.2.3. Perfil de Alumno

El storyboard final de las pantallas del perfil de alumno lo podemos ver en la Figura 110.

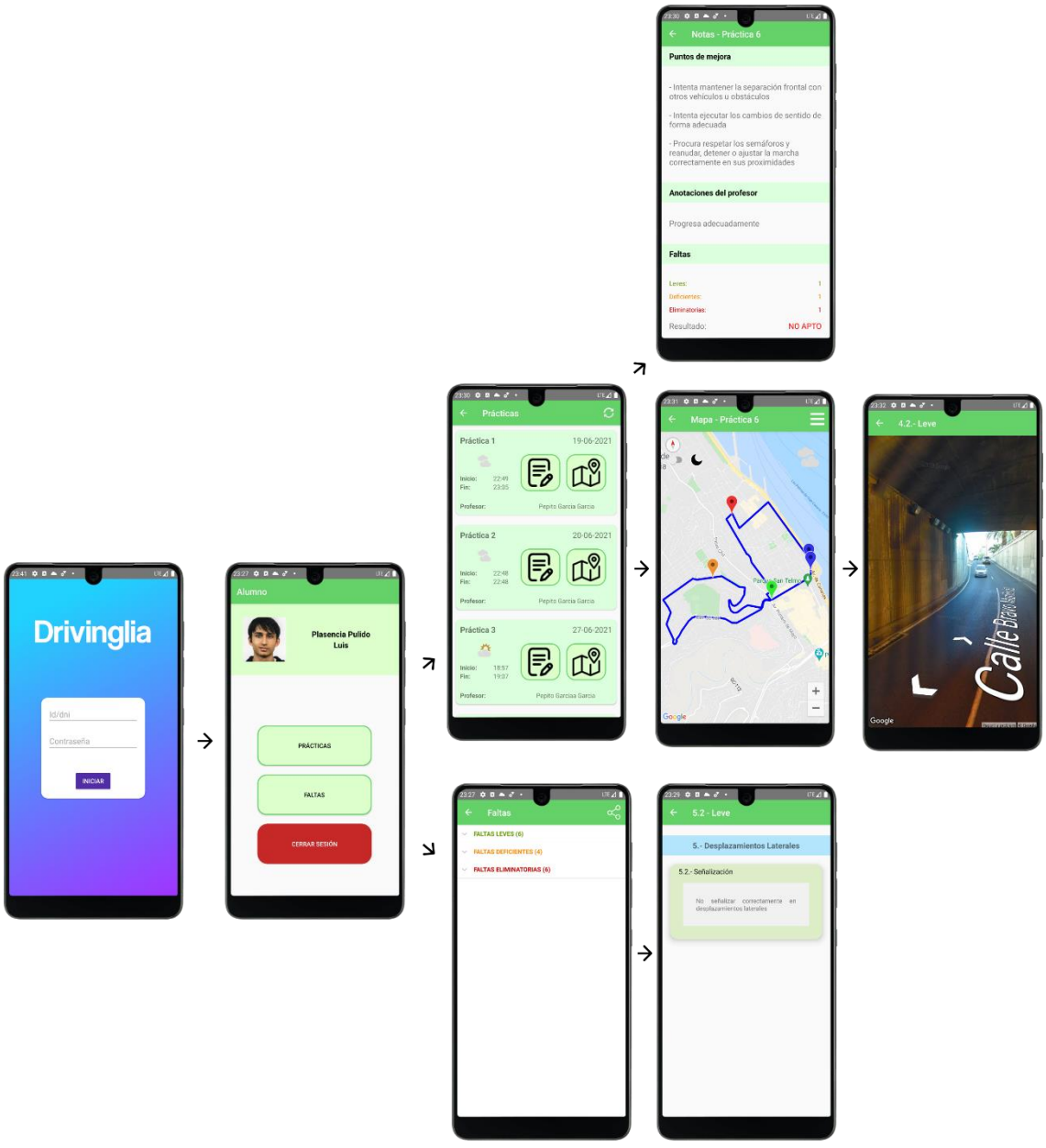


Figura 110. Storyboard: Perfil de Alumno

7. Conclusiones

En este capítulo expondremos las conclusiones alcanzadas tras la elaboración del trabajo de Fin de Grado.

7.1. Introducción

Los índices de suspenso en el primer intento del examen de conducción en España son preocupantemente elevados. Conversando con un profesor de autoescuela con décadas de experiencia acerca de qué se podría hacer para mejorar la enseñanza de las clases prácticas de conducción llegamos a la conclusión de que uno de los apartados que se podría mejorar se encontraba en el **seguimiento de las prácticas de conducir**.

Tras tomar la decisión de que una aplicación móvil podría ayudar al seguimiento de las prácticas de conducir, propusimos desarrollar Drivinglia.

El objetivo principal de la aplicación sería mejorar la enseñanza práctica en vías públicas gracias al tracking de las prácticas y seguimiento del progreso del alumno.

Para cumplir este objetivo generalista, estipulamos una serie de objetivos específicos que nos ayudarían a cumplir nuestras metas:

1. **Análisis del dominio del problema y desarrollo de los mockups de la aplicación para determinar las interfaces de usuario y funcionalidad aproximada que podría tener la aplicación.**
2. **Familiarización con las tecnologías de trackeo en carretera y desarrollo de la parte de la aplicación que realiza el seguimiento de las prácticas.**
3. **Diseño y programación de un sistema de perfiles e inicio de sesión acorde al alcance del proyecto.**
4. **Almacenamiento de los datos en la nube haciendo uso de la base de datos no relacional y en tiempo real Firebase.**
5. **Puesta en marcha de un sistema de seguimiento de voz por manos libres opcional para el establecimiento de las faltas.**

Hemos desarrollado un proyecto software centrado en estos objetivos siguiendo una metodología **ordenada y ágil**.

Ordenada debido a que hemos tenido un desarrollo escalado del proyecto con ciclos de implementación que generaban versiones de la aplicación totalmente funcionales. Estas funcionalidades se iban apuntando según se desarrollaba la app utilizando un tablero de tareas Kanban a través de la aplicación *Trello.* Las funciones de la aplicación que se terminaban de implementar las marcábamos en verde sobre la columna del tablero *done*,

aquellas que estuvieran a medio camino en amarillo sobre la columna *doing* y las que estuvieran por hacer sobre la columna *todo* en rojo.

Ágil porque hemos utilizado metodologías eficientes con los ciclos de trabajo a través de un repositorio online de *GitHub*.

7.2. Conclusiones

Se ha logrado cumplir con todos los objetivos que se tuvieron en mente antes del desarrollo del proyecto. Para ello hubo que reorganizar las prioridades y relegar ciertas funciones a líneas futuras, de este modo desperdiciamos lo mínimo posible las horas planificadas.

A continuación, y teniendo en cuenta los objetivos mencionados en el anterior apartado vamos a justificar cada uno de ellos:

Objetivo 1: A través de la herramienta Figma, hemos conseguido desarrollar unos mockups que han contribuido al diseño de las pantallas finales. Podemos ver las comparaciones entre las pantallas mockups del capítulo de **diseño 4.3.** y las pantallas finales del **capítulo 6.1.** y comprobar su similitud.

Objetivo 2: Hemos desarrollado una pantalla de prácticas en el perfil de profesor que realiza tracking del circuito y dibuja el recorrido en el mapa. El alumno, desde su terminal puede ver las prácticas que ha realizado y llevar un seguimiento de estas.

Objetivo 3: Conseguimos implementar 3 tipos de perfiles y darles un diseño diferente a cada uno de ellos. El perfil de alumno tiene como temática el color verde, el perfil de profesor el azul y el de administrador el rojo. Cada uno de los perfiles tiene funciones determinadas que se han señalado a lo largo de la memoria del trabajo de Fin de Grado.

Objetivo 4: Hemos utilizado *Firebase Real-Time Database* para almacenar la información de los usuarios y de las prácticas. Gran parte de las pantallas de Drivinglia acceden a la base de datos en tiempo real de Firebase de manera reiterada.

Objetivo 5: El profesor tiene la opción de definir la falta cometida oralmente en la pantalla de prácticas. Hemos hecho uso de los servicios que nos proporciona Google para ello.

A parte de los objetivos iniciales, hemos alcanzado también una serie de **objetivos adicionales** que se añadieron en mitad del desarrollo. La justificación de cada uno se encuentra en el capítulo **3.2. Nuevos objetivos alcanzados.**

1. **Almacenamiento y descarga de las imágenes de usuario desde la aplicación haciendo uso de la base de datos *Firebase Database* e implementación de un sistema de gestión de usuarios acorde.**
2. **Aplicación de las distintas herramientas de las que disponen los mapas de Google como pueden ser el Street View o la personalización del estilo del mapa, polilíneas y marcadores.**
3. **Implementar un sistema de acceso a las condiciones meteorológicas.**
4. **Diseño y programación de pantallas de visualización del compendio de faltas del alumno.**
5. **Implementar mensajes de puntos de mejora automáticos en función de las faltas cometidas y realizar un balance de faltas en cada práctica.**

Como nota final, las autoescuelas que apliquen un servicio como **Drivinglia** para el seguimiento de las prácticas de conducir tendrán un valor añadido a tener en cuenta por el alumno a la hora de elegir autoescuela.

7.3. Líneas futuras

Hemos relegado algunas de las funciones que teníamos pensadas para la app a líneas futuras o actualizaciones posteriores.

No hemos implementado estas funciones ya que no han tenido la importancia suficiente frente a otras que sí la tenían. Además, nos empezábamos a quedar cortos de tiempo, lo que fue el agravante para tomar esta decisión.

Primeramente, la **pantalla de configuración** del perfil de profesor que teníamos pensada y diseñamos en el apartado de los mockups no se llegó a implementar. En esta pantalla el profesor podría especificar ciertas funciones por defecto de las prácticas como el modo noche, el color de la traza en el mapa o la manera en la que se registran las prácticas.

Por otro lado, los perfiles de alumno y profesor deberían poder cambiar información de su cuenta como la contraseña por defecto.

Además, se podrían poner todos los motivos de faltas posibles del reglamento en el detalle de la falta en vez de un motivo general.

A su vez, convendría darle tanto al profesor como al administrador **acceso a las prácticas** de los alumnos. Estos perfiles deberían poder generar, para cada alumno, una ficha de anotaciones con la información resumida de todas las prácticas realizadas. De esta forma se podría estudiar el balance conjunto de los estudiantes y extraer información que nos pueda ayudar a mejorar aún más la educación. Con esta metodología podríamos añadirle al perfil de profesor un **historial** con las últimas prácticas realizadas.

Por último, la aplicación necesita un icono o **logo** propio y original que la identifique a primera vista que habría que diseñar.

Bibliografía

[1] L. Albor, “Examinarse del carné de conducir: por qué la mayoría no lo aprueba a la primera”, *ABC*, sep. 2017 [En línea]. Disponible en: https://www.abc.es/sociedad/abci-examinarse-carne-conducir-mayoria-no-aprueba-primera-201707272202_noticia.html. [Accedido: 01-feb- 2021]

[2] R. Boyer, *Android 9 Develoment CookBook,* 3a ed., Packt, oct. 2018 [En línea]. Disponible en: https://www.packtpub.com/product/android-9-development-cookbook-third-edition/9781788991216

[3] “Angular”, *Angular.io*, 2021 [En línea]. Disponible en: https://angular.io/guide/what-is-angular

[4] “Free Mobile App Development: Getting started with Ionic Apps”, *Ionic Framework* [En línea]. Disponible en: https://ionicframework.com/getting-started

[5] D. Bernal González, “Principales tipos de aplicaciones móviles: ventajas, desventajas y ejemplos”, *profile*, abr. 2021 [En línea]. Disponible en: https://profile.es/blog/tipos-aplicaciones-moviles-ventajas-ejemplos/

[6] H. Yahiaoui, *Firebase Cookbook*, Packt, nov. 2017 [En línea]. Disponible en: https://www.packtpub.com/product/firebase-cookbook/9781788296335

[7] Subdirección Adjunta de Formación Vial, “Criterios de calificación de la prueba de control de aptitudes y comportamientos en vías abiertas al tráfico general”, *Sede electrónica DGT*, sept. 2019 [En línea]. Disponible en: https://sede.dgt.gob.es/sede-estaticos/Galerias/permisos-de-conducir/notas-de-examen/CRITERIOS-DE-CALIFICACION-VIAS-ABIERTAS-SEPTIEMBRE-2019.pdf

[8] K. Schwaber y J. Sutherland, “The Scrum Guide”, *scrumguides*, nov. 2020 [En línea]. Disponible en: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

[9] J. Kim, "Getting Started with MVP (Model View Presenter) on Android", *raywenderlich*, dic. 2018 [En línea]. Disponible en: https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android. [Accedido: 01-feb-2021]

[10] R. Cecil Martin, *Clean Code: A Handbook of Agile Software Craftsmanship,* Pearson, ago. 2008.

[11] "Autoescuela Clases Prácticas", *Google Play.* [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.cnae.editorial.aeclasespracticas&hl=en&gl=US. [Accedido: 22-jul-2021]

[12] "TodoTest: Test de conducir", *Google Play.* [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.TodoTest. [Accedido: 22-jul-2021]

[13] "Dribo - La autoescuela en tu móvil", *Google Play*. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=es.dribo.user_app&hl=en&gl=US. [Accedido: 22-jul-2021]

[14] "RecognizerIntent", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/reference/android/speech/RecognizerIntent. [Accedido: 13-abr-2021]

[15] "Meet Android Studio", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/studio/intro

[16] "Agrega Firebase al proyecto de Android", *Firebase,* 2021 [En línea]. Disponible en: https://firebase.google.com/docs/android/setup?hl=es. [Accedido: 01-feb-2021]

[17] "Trello", *Trello.com,* 2021. [En línea]. Disponible en: https://trello.com/. [Accedido: 01-feb-2021]

[18] "Figma: the collaborative interface design tool", *Figma*, 2021 [En línea]. Diponible en: https://www.figma.com/

[19] "Espresso", *Android developers*, 2020 [En línea]. Diponible en:

https://developer.android.com/training/testing/espresso

[20] "Github: Where the world builds software", *GitHub,* 2021. [En línea]. Disponible en: https://github.com/. [Accedido: 01-feb-2021]

[21] "API - Wikipedia", *En.wikipedia.org,* 2021 [En línea]. Disponible en: https://en.wikipedia.org/wiki/API. [Accedido: 22-jul-2021]

[22] "Directions API overview", *Google Developers*, 2021 [En línea]. Disponible en: https://developers.google.com/maps/documentation/directions/overview. [Accedido: 22-jul-2021]

[23] "About MetaWeather", *Metaweather.com*, 2021 [En línea]. Disponible en: https://www.metaweather.com/es/about/. [Accedido: 22-jul-2021]

[24] "Volley overview", Android developers, 2021 [En línea]. Disponible en: https://developer.android.com/training/volley. [Accedido: 22-jul-2021]

[25] "Make a standard request", Android developers, 2021 [En línea]. Disponible en: https://developer.android.com/training/volley/request. [Accedido: 22-jul-2021]

[26] "GitHub – google/JSON: A java serialization/deserialization library to convert Java Object into JSON and back", GitHub, 2021. [En línea]. Disponible en: https://github.com/google/gson. [Accedido: 22-jul-2021]

[27] "GitHub – ArthurHub/Android-Image-Cropper: Image Cropping Library for Android, optimized for Camera / Gallery" , *GitHub,* 2021 [En línea]. Disponible en: https://github.com/ArthurHub/Android-Image-Cropper. [Accedido: 22-jul-2021]

[28] "Set up Google Play services", *Google Developers*, 2021. [En línea]. Disponible en: https://developers.google.com/android/guides/setup. [Accedido: 22-jul-2021]

[29] "Request location updates", *Android developers,* 2021 [En línea]. Disponible en: https://developer.android.com/training/location/request-updates. [Accedido: 22-jul-2021]

[30] "StandardCharsets (Java SE 10 & JDK 10)", *Docs.oracle.com* [En línea]. Disponible en: https://docs.oracle.com/javase/10/docs/api/java/nio/charset/StandardCharsets.html#UTF_8. [Accedido: 22-jul-2021]

[31] "Free Vector Icons and Stickers – Thousands of resources to download", *Flaticon*, 2021. [En línea]. Disponible en: https://www.flaticon.com/

[32] "Integer", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/reference/java/lang/Integer. [Accedido: 22-jul-2021]

[33] "Understand the Activity Lifecycle", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/guide/components/activities/activity-lifecycle. [Accedido: 22-jul-2021]

[34] "Assets Folder in Android Studio", *geekforGeeks.org*, Mar. 2021 [En línea]. Disponible en: https://www.geeksforgeeks.org/assets-folder-in-android/. [Accedido: 22-jul-2021]

[35] "Recupera datos | Firebase Realtime Database", *Firebase* https://firebase.google.com/docs/database/admin/retrieve-data. [Accedido: 22-jul-2021]

[36] "Directions Service | Maps JavasCRIPT API", *Google developers*, 2021 [En línea]. Disponible en: https://developers.google.com/maps/documentation/javascript/directions#DirectionsResults. [Accedido: 22-jul-2021]

[37] "Encoded Polyline Algorithm Format | Google Maps Platform", Google Developers, 2021. [En línea]. Disponible en: https://developers.google.com/maps/documentation/utilities/polylinealgorithm. [Accedido: 22-jul-2021]

[38] "Android Google Map – Drawing Route Between two points", *JournalDev*, 2021 [En línea]. Disponible en: https://www.journaldev.com/13373/android-google-map-drawing-route-two-points. [Accedido: 22-jul-2021]

[39] "API – MetaWeather", Metaweather.com, 2021 [En línea]. Disponible en: https://www.metaweather.com/api/. [Accedido: 22-jul-2021]

[40] "Android Weather App using REST api 09 Fetch JSONArray with Volley", Youtube.com, Jun. 2020 [En línea]. Disponible en: https://www.youtube.com/watch?v=M_iDypENBVY&list=PLhPyEFL5u-i2YhBVIRrAP4WIFUZ-IimES&index=13. [Accedido: 22-jul-2021]

[41] "How to enable button in Android Studio", *Stack Overflow*, Jul. 2020 [En línea]. Disponible en: https://stackoverflow.com/questions/63098303/how-to-enable-button-in-android-studio. [Accedido: 22-jul-2021]

[42] "Descarga archivos con Cloud Storage en Android", *Firebase*, 2021 [En línea]. Disponible en: https://firebase.google.com/docs/storage/android/download-files#java. [Accedido: 22-jul-2021]

[43] "Add a Floating Action Button", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/guide/topics/ui/floating-action-button. [Accedido: 22-jul-2021]

[44] "What is the best way to filter a Java Collection?", *Stack Overflow,* 2008 [En línea]. Disponible en: https://stackoverflow.com/questions/122105/what-is-the-best-way-to-filter-a-java-collection. [Accedido: 22-jul-2021]

[45] "Sorting arraylist in alphabetical order (case insensitive), *Stack Overflow*, 2011 [En línea]. Disponible en: https://stackoverflow.com/questions/5815423/sorting-arraylist-in-alphabetical-order-case-insensitive. [Accedido: 22-jul-2021]

[46] "Comparator", *Android developers, 2021* [En línea]. Disponible en: https://developer.android.com/reference/java/util/Comparator. [Accedido: 22-jul-2021]

[47] "DatePickerDialog", *Android developers, 2021* [En línea]. Disponible en: https://developer.android.com/reference/android/app/DatePickerDialog. [Accedido: 22-jul-2021]

[48] "Android Beginner Tutorial #25 – DatePicker Dialog [Choosing a Date from a Dialog Pop-Up]", *Youtube.com,* Abr. 2017, [En línea]. Disponible en: https://www.youtube.com/watch?v=hwe1abDO2Ag. [Accedido: 22-jul-2021]

[49] "How to check edittext's text is email address or not?", *Stack Overflow,* 2011 [En línea]. Disponible en: https://stackoverflow.com/questions/6119722/how-to-check-edittexts-text-is-email-address-or-not. [Accedido: 22-jul-2021]

[50] "How can I get current date in Android", Stack Overflow, 2011 [En línea]. Disponible en: https://stackoverflow.com/questions/8654990/how-can-i-get-current-date-in-android. [Accedido: 22-jul-2021]

[51] "Base 64 encode and decode example code", *Stack Overflow*, 2011 [En línea]. Disponible en: https://stackoverflow.com/questions/7360403/base-64-encode-and-decode-example-code. [Accedido: 22-jul-2021]

[52] "Sending simple data to other apps", *Android developers,* 2021 [En línea]. Disponible en: https://developer.android.com/training/sharing/send. [Accedido: 22-jul-2021]

[53] "ArrayAdapter", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/reference/android/widget/ArrayAdapter. [Accedido: 22-jul-2021]

[54] "FusedLocationProviderClient | Google Play Services", *Google developers,* 2021 [En línea]. Disponible en: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient. [Accedido: 22-jul-2021]

[55] "Adding a Styled Map | Maps SDK for Android", *Google developers*, 2021 [En línea]. Disponible en: https://developers.google.com/maps/documentation/android-sdk/styling. [Accedido: 22-jul-2021]

[56] "Updating the UI from a Timer | Android Developers", *Docs.huihoo.com* [En línea]. Disponible en: https://docs.huihoo.com/android/2.3/resources/articles/timed-ui-updates.html. [Accedido: 22-jul-2021]

[57] "Android Tutorial: Convert Speech To Text | Speech Recognition", *Youtube.com*, Abr. 2017 [En línea]. Disponible en: https://www.youtube.com/watch?v=0bLwXw5aFOs. [Accedido: 22-jul-2021]

[58] "Street View | Maps SDK for Android", *Google developers,* 2021 [En línea]. Disponible en: https://developers.google.com/maps/documentation/android-sdk/streetview. [Accedido: 22-jul-2021]

[59] "Markers | Maps SDK for Android", *Google developers,* 2021 [En línea]. Disponible en: https://developers.google.com/maps/documentation/android-sdk/marker. [Accedido: 22-jul-2021]

[60] "Android ExpandableListView Example Tutorial", *JournalDev,* 2021 [En línea]. Disponible en: https://www.journaldev.com/9942/android-expandablelistview-example-tutorial#:~:text=Android%20ExpandableListView%20is%20a%20view,and%20their%20respective%20children%20items. . [Accedido: 22-jul-2021]

[61] "Test UI for a single app", *Android developers*, 2021 [En línea]. Disponible en: https://developer.android.com/training/testing/ui-testing/espresso-testing#setup. [Accedido: 22-jul-2021]

[62] "Create UI tests with Espresso Test Recorder", *Android developers,* 2020 [En línea]. Disponible en: https://developer.android.com/studio/test/espresso-test-recorder. [Accedido: 22-jul-2021]

[63] "How to access child view of recycler view in android espresso?", *Stack Overflow,* 2016 [En llínea]. Disponible en: https://stackoverflow.com/questions/41376039/how-to-access-child-view-of-recycler-view-in-android-espresso. [Accedido: 22-jul-2021]

[64] "Matcher", *Android develoepers,* 2021 [En línea]. Disponible en: https://developer.android.com/reference/java/util/regex/Matcher. [Accedido: 22-jul-2021]

[65] *Smarkmockups.com,* 2021 [En línea]. Disponible en: https://smartmockups.com/

Presupuesto

En este capítulo se estima el coste de la realización del proyecto software.

1. Desglose del presupuesto

Para la elaboración del presupuesto de este proyecto nos vamos a ceñir a las pautas del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT) de 2008. Para ello, desglosaremos los gastos del proyecto en los siguientes apartados:

- Recursos materiales.
- Trabajo tarificado por tiempo empleado.
- Costes de la redacción del documento.
- Derechos del visado del COITT.
- Gastos de tramitación y envío.
- Aplicación de impuestos.

2. Recursos materiales

Siguiendo las recomendaciones del COITT, la partida referente a los recursos materiales empleados se divide en recursos hardware y software.

En la elaboración del Trabajo de Fin de Grado, hemos utilizado una variedad de programas y herramientas informáticas que vamos a considerar como **recursos software**. A su vez, cualquier dispositivo electrónico del que hayamos hecho uso entrará dentro de los **recursos hardware**.

El coste de amortización lo vamos a estipular para un periodo de cuatro años. Utilizaremos un sistema de amortización lineal en el que el material inmovilizado se deprecia de forma constante a lo largo del tiempo de vida útil. Realizaremos los cálculos con la siguiente expresión:

$$Coste\ de\ amortización = \frac{Valor\ de\ adquisición - Valor\ residual}{Años\ de\ vida\ útil}$$

El valor residual es el valor estimado de los elementos después de su vida útil.

Como el desarrollo del Trabajo de Fin de Grado tiene una duración aproximada de 300 horas en cuatro meses, el coste de amortización se calculará en función de esos cuatro meses de trabajo (dividiremos el coste de amortización anual entre 3 para que equivalga a 4 meses de TFG).

3. Recursos Software

Las herramientas software de las que se ha hecho uso en el desarrollo del Trabajo de Fin de Grado han sido las siguientes:

- **Android Studio.** Open source. IDE en el que programamos la aplicación.
- **Figma.** Open source. Programa de edición.
- **Firebase.** Hemos utilizado el plan Spark, el cual es gratuito y tiene un límite de 100 usuarios simultáneos, 1 GB de almacenamiento y permite un volumen de datos de 10 GB al mes.
- **Google Cloud Platform.** Hemos añadido información de facturación, pero al no superar las mil peticiones en los distintos servicios de pago el coste ha sido nulo.
- **Microsoft Office 365.** Nos basaremos en los precios estándar para uso personal. La licencia tiene un precio de 69 euros anuales.
- **GitHub.** Pagar en GitHub te permite compartir un repositorio privado. Como el desarrollo de la aplicación se realizó de manera individual, no hemos necesitado hacer uso de esta opción.

El coste del conjunto de las herramientas software empleadas ha sumado un total de sesenta y nueva euros **(69,00€)**.

4. Recursos Hardware

Las herramientas hardware de las que se ha hecho uso en el desarrollo del Trabajo de Fin de Grado han sido las siguientes:

- **Ordenador de sobremesa.** (Características en el pliego de condiciones)
- **Periféricos.** (Detalles en el pliego de condiciones)
- **Smartphone Honor 9X.** (Características en el pliego de condiciones)

En la Tabla 10, podemos ver el desglose de los costes en recursos hardware.

Recurso	Valor de adquisición (€)	Valor residual (€)	Coste de amortización (€)
Ordenador sobremesa	1729,8	1215	42,9
Periféricos	591,96	450	11,83
Smartphone	219,16	145	6,18
TOTAL			60,91

Tabla 10. Coste de los Recursos Hardware

Tras estipular los costes de amortización de los distintos elementos tenidos en cuenta, el coste total de los recursos hardware utilizados asciende a los sesenta euros y noventa y un céntimos **(60,91 €)**.

5. Trabajo tarificado por tiempo empleado

Para la elaboración del presente Trabajo de Fin de Grado, se han invertido unas 350 horas de tiempo repartidas en 4 meses. En estas 350 horas se han realizado las labores de familiarización con las tecnologías de las que se han hecho uso, diseño y desarrollo de la aplicación y elaboración de la memoria del trabajo.

Si seguimos las recomendaciones del COITT, obtenemos una aproximación del importe de las horas empleadas en la realización del proyecto en manos de un Ingeniero de Telecomunicaciones como la siguiente expresión:

$$H = C_t * 74{,}88 * H_n + 96{,}72 * H_e$$

Donde:

- C_t indica un factor de corrección función del número de horas trabajadas.
- H_n indica las horas trabajadas dentro de la jornada laboral (horas normales).
- H_e indica las horas trabajadas fuera de la jornada laboral (horas especiales).

Según el COITT, el coeficiente C_t tiene un valor variable en función del número de horas empleadas de acuerdo con la Tabla 11.

Horas empleadas (H_n)	Factor de corrección (C_t)
Hasta 36	1
36-72	0,9
72-108	0,8
108-144	0,7
144-180	0,65
180-360	0,6

360-540	0,55
540-720	0,5
720-1080	0,45
Más de 1080	0,4

Tabla 11. Factor de corrección en función de las horas trabajadas.

Ya que hemos realizado el trabajo en los 4 meses estipulados en el anteproyecto, nos situamos en el grupo de las 180-360 horas con un factor de corrección de 0,6.

Al aplicar dicho factor a la expresión señalada anteriormente y dividir las horas totales en horas dentro y fuera del horario laboral nos saldría la siguiente expresión:

$$H = 0{,}6 * 74{,}88 * 300 + 96{,}72 * 50 = 17.314{,}4€$$

Los honorarios totales por tiempo dedicado ascienden a diecisiete mil trescientos catorce euros y cuarenta céntimos **(17314,4€)**.

6. Costes asociados a la redacción del documento

El importe de la redacción del proyecto se calcula de acuerdo con la siguiente expresión:

$$R = 0{,}07 * P * C_n$$

Donde:

- P es el presupuesto del proyecto.
- C_n es el coeficiente de ponderación en función del presupuesto.

El coeficiente C_n está determinado por el presupuesto del proyecto, en nuestro caso dicha suma asciende a diecisiete mil trescientos setenta y cinco euros y treinta y un céntimos (0 + 60,91 + 17.314,4 = 17.375,31€).

El COITT establece que para un presupuesto menor de 30.050€, el coeficiente de ponderación tiene un valor de 1. Por tanto, incorporando esta información a la expresión anterior obtenemos la siguiente expresión:

$$R = 0{,}07 * 17.375{,}31 * 1 = 1.216{,}27€$$

Por lo tanto, el coste libre de impuestos obtenido de la redacción del documento asciende a mil doscientos dieciséis euros y veintisiete céntimos **(1.216,27€)**.

7. Derechos de visado COITT

Los gastos del visado del COITT se tarifan mediante la siguiente expresión:

$$V = 0{,}006 * P * C_v$$

Donde:

- P es el presupuesto del trabajo.
- C_v es el coeficiente reductor en función del presupuesto del trabajo.

El presupuesto P calculado hasta el momento asciende a la suma de los costes de ejecución material (hardware) y de redacción. No tenemos costes asociados a material fungible ya que no es necesaria la impresión del documento para su evaluación.

Obtenemos que el presupuesto actual es:

$$P = 17.375{,}31 + 1.216{,}27 = 18.591{,}58€$$

El coeficiente de ponderación C_v para presupuestos menores de 30.050€ definido por el COITT tiene un valor de 1, por lo que el coste de los derechos de visado para el presente trabajo asciende a:

$$V = 0{,}006 * 18.591{,}58 * 1 = 111{,}55€$$

Podemos concluir que el coste de los derechos de visado del proyecto asciende a ciento once euros y cincuenta y cinco céntimos **(111,55€)**.

8. Gastos de tramitación y envío

Los gastos de tramitación y envío están estipulados en seis euros y un céntimo **(6,01€)**.

9. Aplicación de impuestos

El coste total del proyecto, antes de aplicarle los correspondientes impuestos asciende a 18.709,14€.

A esta cantidad hay que sumarle el Impuesto General Indirecto Canario (IGIC) del 7%. En la Tabla 12, podemos ver el desglose de los costes totales del proyecto.

Recurso	Coste (€)
Recursos materiales	129,91
Trabajo tarificado por tiempo empleado	17.314,4
Costes asociados a la redacción del documento	1.216,27

Derechos de visado del COITT	111,55
Gastos de tramitación y envío	6,01

Subtotal	18.778,14€
Aplicación de impuestos (IGIC 7%)	1.314,47€
Total	20.092,61€

Tabla 12. Costes totales del proyecto

El presupuesto total del presente Trabajo de Fin de Grado asciende a veinte mil dieciocho euros y setenta y ocho céntimos **(20.092,61€)**.

Las Palmas de Gran Canaria a 23 de Julio de 2020

Firma:

Luis Plasencia Pulido

Anexos

En este apartado vamos a detallar las consideraciones complementarias a la elaboración del Trabajo de Fin de Grado. Destacamos las interfaces de funcionalidades comunes y básicas, secciones que fueron incluidas como anexos debido a su extensión.

Interfaces de funcionalidades comunes

En este apartado vamos a describir las interfaces de funcionalidades comunes, a las que acceden las distintas clases de cada pantalla de manera indistinta.

Repositorio

El repositorio es una clase Java encargada de almacenar ciertos métodos relacionados con el modelo que se repiten entre pantallas, como pueden ser los métodos de acceso a la base de datos Firebase o a la información del fichero *JSON* de configuración. El modelo de cada pantalla será el encargado de acceder a los métodos del repositorio.

REPOSITORYCONTRACT

public interface RepositoryContract

Interfaz utilizada para definir los métodos públicos que formarán parte del repositorio.

REPOSITORY

public class Repository
extends java.lang.Object
implements RepositoryContract

CONSTRUCTOR

Repository

private Repository(Context context)

Al crearse el repositorio, extrae la información del fichero JSON de configuración como listas y almacena el contexto de la vista como variable global.

Parámetros		
Nombre	**Tipo**	**Descripción**
context	Context	Referencia al contexto de la vista. En este caso nos sirve para acceder a los archivos locales del a aplicación.

VARIABLES

TAG

public static String TAG

Etiqueta que identifica la clase. Nos puede servir a la hora de imprimir un mensaje en la consola, por ejemplo.

context

private Context context

Referencia al contexto de la vista que creo el repositorio. Nos sirve para acceder a la carpeta de *assets* o recursos donde está contenido el JSON de configuración.

INSTANCE

private static Repository INSTANCE

Almacena la referencia al repositorio.

reference

com.google.firebase.database.DatabaseReference reference

Referencia a la base de datos Firebase.

userList

private List<User> userList

Lista de usuarios recogidos de Firebase.

practicaList

private java.util.List<Practica> practicaList

Lista de practices de un usuario recogidas de Firebase.

faltasEliminatorias

private List<Falta> faltasEliminatorias

Lista de faltas eliminatorias del fichero JSON de configuración.

faltasDeficientes

private List<Falta> faltasDeficientes

Lista de faltas deficientes del fichero JSON de configuración.

faltasLeves

private List<Falta> faltasLeves

Lista de faltas leves del fichero JSON de configuración.

secciones

private List<Seccion> secciones

Lista de secciones del fichero JSON de configuración.

postListenerPracticas

private com.google.firebase.database.ValueEventListener postListenerPracticas

Listener de eventos de Firebase que utilizaremos para acceder a la lista de practices de un alumno y que cualquier cambio actualice la lista de prácticas automáticamente.

CONSTANTES

JSON_FALTAS_FILE

public static final String JSON_FALTAS_FILE

Almacena el nombre del fichero JSON de configuración.

JSON_FALTAS_ROOT

public static final String JSON_FALTAS_ROOT

Almacena la clave del array JSON de faltas del fichero de configuración.

JSON_FALTAS_SECCIONES

public static final String JSON_FALTAS_SECCIONES

Almacena la clave del array JSON de secciones del fichero de configuración.

JSON_FALTAS_LEVES

public static final String JSON_FALTAS_LEVES

Almacena la clave del array JSON de faltas leves del fichero de configuración.

JSON_FALTAS_DEFICIENTES

public static final String JSON_FALTAS_DEFICIENTES

Almacena la clave del array JSON de faltas deficientes del fichero de configuración.

JSON_FALTAS_ELIMINATORIAS

public static final String JSON_FALTAS_ELIMINATORIAS

Almacena la clave del array JSON de faltas eliminatorias del fichero de configuración.

MÉTODOS

fetchPracticasFromFirebase

public void fetchPracticasFromFirebase(String userId)

De manera asíncrona, inicia un listener de eventos de Firebase para extraer la información de prácticas de un alumno en concreto. El listener es añadido a la referencia de la base de datos para permanecer en escucha, por lo que cualquier cambio en las tablas de Firebase hará que se vuelva a llamar al listener y se modifique la variable global de listado de prácticas.

Parámetros

Nombre	Tipo	Descripción
userId	String	Id del usuario. Lo utilizaremos para obtener la referencia de Firebase a las prácticas de dicho usuario.

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

De manera asíncrona, inicia un listener de eventos de Firebase para extraer la información de usuarios de la raiz de nuestra tabla Firebase. El listener es añadido a la referencia de la base de datos para solo tomar un valor, por lo que si queremos comprobar si ha habido cambios, tendremos que volver a llamar al método.

Parámetros

Nombre	Tipo	Descripción
callback	fetchUsersFromFirebaseCallback	Al ser una operación asíncrona para no interrumpir el hilo principal de ejecución, el callback notificará de vuelta al modelo cuando se hayan recuperado los datos.

generarFaltas

private void generarFaltas()

Llama a los métodos necesarios para iniciar la extracción de los datos del archivo JSON de configuración de manera asíncrona.

getFaltasDeficientes

List<Falta> getFaltasDeficientes()

Getter para la lista de faltas deficientes del archivo JSON de configuración.

Retorno

Tipo	Descripción
List<Falta>	Lista de faltas deficientes del archivo JSON de configuración.

getFaltasEliminatorias

public List<Falta> getFaltasEliminatorias()

Getter para la lista de faltas eliminatorias del archivo JSON de configuración.

Retorno	
Tipo	**Descripción**
List<Falta>	Lista de faltas eliminatorias del archivo JSON de configuración.

getFaltasLeves

public List<Falta> getFaltasLeves()

Getter para la lista de faltas leves del archivo JSON de configuración.

Retorno	
Tipo	**Descripción**
List<Falta>	Lista de faltas leves del archivo JSON de configuración.

getInstance

public static RepositoryContract getInstance(Context context)

Si no se ha creado la clase (repositorio) antes, genera una referencia a la misma, la asigna a la variable global y la retorna. Si ya existía, devuelve la que ya estaba creada. Este método sirve para conservar información del repositorio y así operar como un singleton, una clase de la que solo se genera una referencia o instancia.

Parámetros		
Nombre	**Tipo**	**Descripción**

context	Context	Contexto de la vista (*Activity*). Se utilizará para acceder al archivo JSON local.

getPracticasList

public List<Practica> getPracticasList()

Getter que retorna la última lista de prácticas del alumno accedida de Firebase.

Retorno	
Tipo	**Descripción**
List<Practica>	Lista de prácticas del alumno.

getSecciones

public List<Seccion> getSecciones()

Getter que retorna la lista de secciones del archivo JSON de configuración.

Retorno	
Tipo	**Descripción**
List<Seccion>	Lista de secciones del archivo JSON de configuración.

getUserlist

public List<User> getUserlist()

Getter que retorna la última lista de usuarios accedida de Firebase.

Retorno	
Tipo	**Descripción**
List<User>	Lista de usuarios de Firebase.

loadFaltasFromJSON

public void loadFaltasFromJSON(String JSON)

Extrae la información del JSON de configuración en forma de listas de faltas y secciones haciendo uso de la librería *gson*.

Parámetros

Nombre	Tipo	Descripción
JSON	String	Archivo JSON de configuración decodificado en formato String.

loadJSONFromAsset

public String loadJSONFromAsset()

Accede al JSON de configuración de la carpeta de recursos (*assets*) y decodifica su información en formato String.

Retorno

Tipo	Descripción
String	Archivo JSON de configuración decodificado en formato String.

removePracticasEventListener

public void removePracticasEventListener()

Desvincula el listener de eventos de la referencia Firebase a las prácticas de usuario. Llamamos a este método en el cierre de sesión del alumno.

resetInstance

public void resetInstance()

Hace nula la variable global que almacena la referencia al repositorio (*Repository*), liberando así cualquier información que haya almacenado.

Peticiones http

Para realizar peticiones HTTP a las distintas API de las que hacemos uso en Drivinglia, utilizamos la librería Volley. Recomendada por Android, la librería HTTP Volley nos facilita y acelera las labores de networking para volúmenes de datos comedidos.

VOLLEYREQUESTQUEUESINGLETON

public class VolleyRequestQueueSingleton
extends java.lang.Object

Las peticiones HTTP por Volley hacen uso de colas llamadas *RequestQueue*. Nuestra aplicación va a hacer peticiones *http* de forma frecuente a la hora de acceder a la API Rest de direcciones para almacenar la trayectoria que va siguiendo el vehículo. Utilizaremos esta clase para tener una sola instancia de la *RequestQueue* y así mejorar la eficiencia hacia la red.

Esta manera de trabajar es la recomendada por la documentación oficial de Volley.

CONSTRUCTOR

VolleyRequestQueueSingleton

private VolleyRequestQueueSingleton(Context context)

En su creación, la clase acepta una referencia al contexto de la vista para poder inicializar la *RequestQueue*.

Parámetros

Nombre	Tipo	Descripción

context	Context	Contexto a la vista que utilizaremos para inicializar la *RequestQueue*.

VARIABLES

ctx

private static Context ctx

Almacena el contexto a la vista (*Activity*).

instance

private static VolleyRequestQueueSingleton instance

Referencia a la clase.

requestQueue

private com.android.volley.RequestQueue requestQueue

Solo se creará una cola y la almacenaremos en esta variable.

MÉTODOS

addToRequestQueue

public <T> void addToRequestQueue(com.android.volley.Request<T> req)

Añade una petición a la cola.

Parámetros		
Nombre	**Tipo**	**Descripción**
req	Request<T>	Petición que realizar a través de Volley.

getInstance

public static VolleyRequestQueueSingleton getInstance(Context context)

Si no se ha creado la clase antes, genera una referencia a la misma, la asigna a la variable global y la retorna. Si ya existía, devuelve la que ya estaba creada. Este método sirve para conservar la información de la clase y así operar como un singleton, una clase de la que solo se genera una referencia o instancia.

Parámetros		
Nombre	**Tipo**	**Descripción**
context	Context	Contexto de la vista (Activity).

getRequestQueue

public com.android.volley.RequestQueue getRequestQueue()

Getter para la cola de volley (*RequestQueue*). Si no había ninguna creada, la crea y la asigna a la variable global, si ya estaba asignada retorna la variable global directamente.

Retorno	
Tipo	**Descripción**
RequestQueue	Archivo JSON de configuración decodificado en formato String.

DirectionsDataService

class DirectionsDataService
extends java.lang.Object

Utilizaremos esta clase para gestionar las peticiones a la API de Google Directions y así generar el tracking en forma de polilíneas.

La URL que utilizaremos para acceder a la API través de una petición *http* tiene el siguiente formato:

https://maps.googleapis.com/maps/api/directions/json?origin=ORIGEN&destination=DESTINO&sensor=false&mode=driving&key=API_KEY

Podemos destacar la especificación del modo *driving* ya que queremos calcular la trayectoria sobre la ruta por carretera.

VARIABLES

context

Context context

Almacena el contexto de la vista (*Activity*)

CONSTANTES

API_KEY

public static final String API_KEY

Valor de la llave de Google personal necesaria para acceder a los servicios de desarrollador.

QUERY_FOR_GOOGLE_API

public static final String QUERY_FOR_GOOGLE_API

Comienzo de la Url hacia la API de Google a la que le añadiremos las query necesarias para completar la petición.

CLASES ANIDADAS

public static interface DirectionsDataService.DirectionsResponseListener

void onResponse(com.google.android.gms.maps.model.PolylineOptions polylineOptions)

void onError(String message)

Utilizaremos esta interfaz para declarar el listener de respuesta a la petición http. Devuelve las opciones de polilínea necesarias para trazar la trayectoria en el mapa si no han habido errores y un mensaje de error en caso contrario.

MÉTODOS

getDirectionsFromOriginToDest

public void getDirectionsFromOriginToDest(LatLng origin, LatLng dest, DirectionsDataService.DirectionsResponseListener directionsResponseListener)

Realiza la petición http a la API de direcciones de Google y la mapea para retornar a través de un listener las opciones de trayectoria de la polilínea o un mensaje de error en caso de que no se haya podido completar la petición.

Parámetros

Nombre	Tipo	Descripción
origin	LatLng	Latitud y longitud de la posición de inicio.
dest	LatLng	Latitud y longitud de la posición final.
directionsResponseListener	DirectionsResponseListener	Listener utilizado para retornar el resultado de la petición.

generateURL

private String generateURL(LatLng origin, LatLng dest)

Genera la url para acceder a la API de direcciones de Google en modo conducción (*driving*), con la latitud y longitud de origen y destino y con salida en formato JSON.

Parámetros		
Nombre	**Tipo**	**Descripción**
origin	LatLng	Latitud y longitud de la posición de inicio.
dest	LatLng	Latitud y longitud de la posición final.

Retorno	
Tipo	**Descripción**
String	URL de acceso a la API de Google Directions en modo driving para un origen y destino determinados.

DIRECTIONSJSONPARSER

public class DirectionsJSONParser
extends java.lang.Object

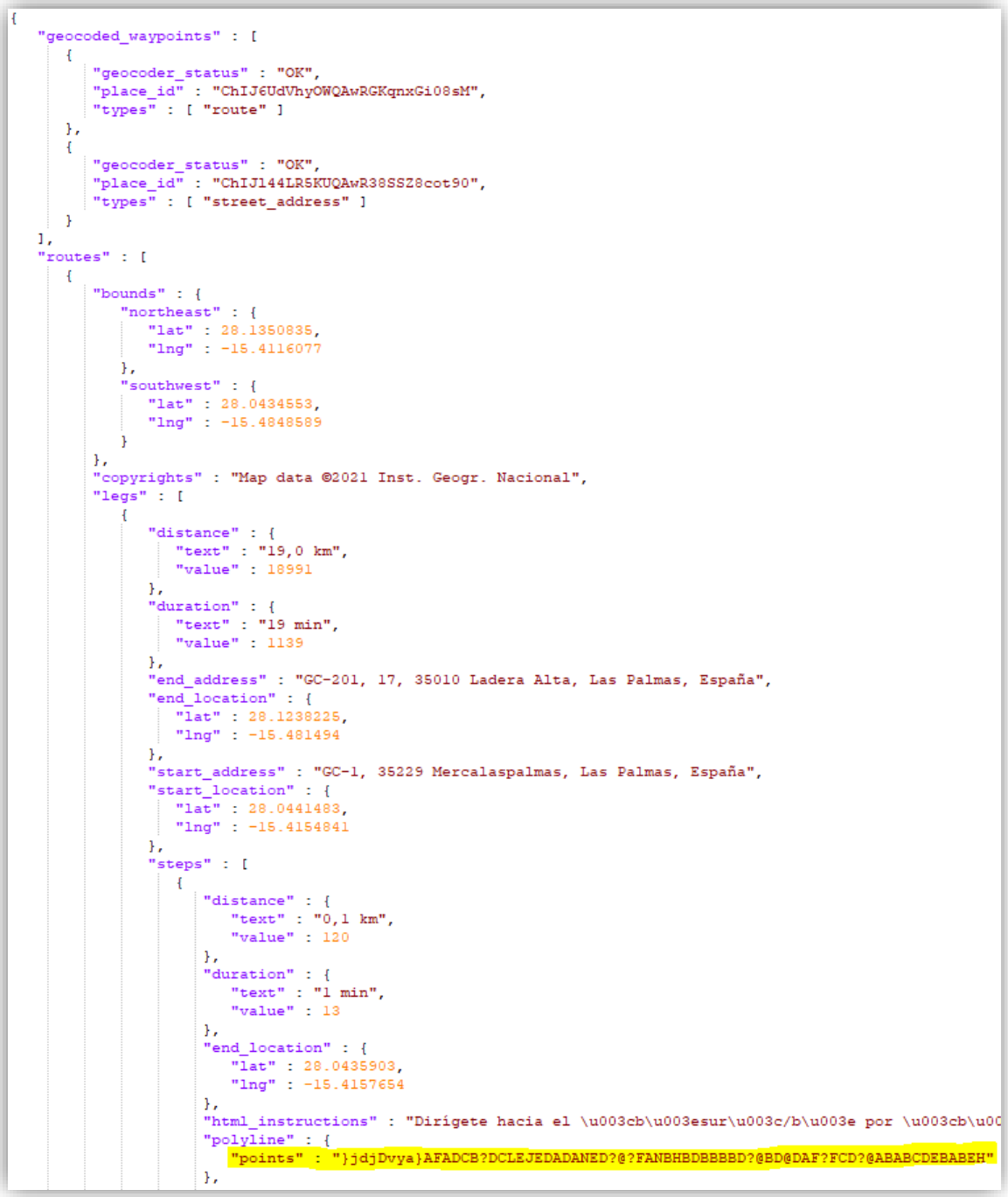
La respuesta de la API de direcciones de Google es un objeto JSON llamado *DirectionsResult* que contiene dos registros:

- ***geocoded_waypoints.*** Contiene un array de objetos de tipo *DirectionsGeocodedWaypoint*, con detalles sobre la geocodificación en destino, origen y puntos de ruta.
- ***routes.*** Contiene un array de objetos de tipo *DirectionsRoute.* Normalmente solo se devuelve una ruta con el recorrido entre el punto de origen y destino.

La información que nos interesa se encuentra en el objeto *DirectionsRoute* del registro *routes*. Dicho objeto contiene varios registros, entre ellos un array denominado *legs* con las etapas de la trayectoria (*DirectionsLeg*) dependiendo de si hubiera múltiples puntos de ruta (geocoded_*waypoints*) o no. Nosotros no especificamos puntos de ruta intermedios así que, en nuestro caso, dicho array solo consistirá en una etapa.

Dicha etapa (*DirectionsLeg*) es un objeto con una serie de registros que especifican las instrucciones de ruta. Entre estos registros, se encuentra un array de pasos denominado *steps*, el cual contiene objetos de tipo *DirectionsStep* con otra serie de registros que especifican la ruta calculada. El registro que buscamos dentro de este objeto se llama *polyline* y contiene una **polilínea codificada** que representa una aproximación suavizada de la ruta del paso en el que se encuentra.

En la Figura 111, podemos ver el JSON devuelto con el registro *polyline* que buscamos resaltado.



```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJ6UdVhyOWQAwRGKqnxGi08sM",
         "types" : [ "route" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJl44LR5KUQAwR38SSZ8cot90",
         "types" : [ "street_address" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 28.1350835,
               "lng" : -15.4116077
            },
            "southwest" : {
               "lat" : 28.0434553,
               "lng" : -15.4848589
            }
         },
         "copyrights" : "Map data ©2021 Inst. Geogr. Nacional",
         "legs" : [
            {
               "distance" : {
                  "text" : "19,0 km",
                  "value" : 18991
               },
               "duration" : {
                  "text" : "19 min",
                  "value" : 1139
               },
               "end_address" : "GC-201, 17, 35010 Ladera Alta, Las Palmas, España",
               "end_location" : {
                  "lat" : 28.1238225,
                  "lng" : -15.481494
               },
               "start_address" : "GC-1, 35229 Mercalaspalmas, Las Palmas, España",
               "start_location" : {
                  "lat" : 28.0441483,
                  "lng" : -15.4154841
               },
               "steps" : [
                  {
                     "distance" : {
                        "text" : "0,1 km",
                        "value" : 120
                     },
                     "duration" : {
                        "text" : "1 min",
                        "value" : 13
                     },
                     "end_location" : {
                        "lat" : 28.0435903,
                        "lng" : -15.4157654
                     },
                     "html_instructions" : "Dirígete hacia el \u003cb\u003esur\u003c/b\u003e por \u003cb\u00
                     "polyline" : {
                        "points" : "}jdjDvya}AFADCB?DCLEJEDADANED?@?FANBHBDBBBBD?@BD@DAF?FCD?@ABABCDEBABEH"
                     },
```


Figura 111: Formato del JSON devuelto por la API Directions de Google

Utilizaremos esta clase para mapear la información del objeto JSON de respuesta de la API de direcciones de Google en forma de listas *HashMap* para decodificar las polilíneas de los distintos pasos o objetos de tipo *DirectionsStep* que conforman la ruta y así generar una polilínea resultante con todos los puntos de ruta a trazar en el mapa.

Esta clase ha sido extraída de la página *JournalDev.*

MÉTODOS

decodePoly

private List decodePoly(String encoded)

Decodifica los puntos de ruta de polilínea de la trayectoria.

Retorno

Tipo	Descripción
List	Lista de puntos de polilínea.

parse

public List<List<HashMap<String, String>>> parse(JSONObject jObject)

Recibe un objeto JSON y retorna una lista de listas de latitud y longitud.

Retorno

Tipo	Descripción
List<List<HashMap<String, String>>>	Lista de listas con los puntos de latitud y longitud de la ruta.

WEATHERDATASERVICE

class WeatherDataService
extends java.lang.Object

Utilizaremos esta clase para gestionar las peticiones a la API REST de MetaWeather y así extraer la información de las condiciones meteorológicas durante la práctica.

VARIABLES

cityId

String cityId

Almacena el id de la ciudad más cercana.

context

Context context

Almacena el contexto de la vista (*Activity*)

CONSTANTES

QUERY_FOR_CITY_ID

public static final String QUERY_FOR_CITY_ID

Comienzo de la URL hacia la API de Metaweather a la que le añadiremos las query necesarias para completar la petición de id de ciudad más cercana.

QUERY_FOR_CITY_WEATHER_BY_ID

public static final String QUERY_FOR_CITY_WEATHER_BY_ID

Comienzo de la Url hacia la API de Metaweather a la que le añadiremos las query necesarias para completar la petición de predicción meteorológica.

CLASES ANIDADAS

public static interface WeatherDataService.CityIDResponseListener

void onResponse(String cityID)

void onError(String message)

Utilizaremos esta interfaz para declarar el listener de respuesta a la petición http de id de ciudad más cercana. Devuelve el id de la ciudad si no han habido errores y un mensaje de error en caso contrario.

public static interface WeatherDataService.CityIDResponseListener

void onResponse(WeatherReportModel weatherReportModel)

void onError(String message)

Utilizaremos esta interfaz para declarar el listener de respuesta a la petición http de predicción meteorológica. Devuelve la predicción del día actual si no han habido errores y un mensaje de error en caso contrario.

MÉTODOS

getCityForecastByID

public void getCityForecastByID(String cityID,
WeatherDataService.ForecastByIDResponseListener forecastByIDResponseListener)

Devuelve la predicción meteorológica del día actual para la ciudad especificada a través del listener una vez se haya completado la petición o un mensaje de error en caso contrario.

Parámetros

Nombre	Tipo	Descripción
cityID	String	Id de la ciudad de la que extraer la predicción meteorológica.
forecastByIDResponseListener	ForecastByIDResponseListener	Listener utilizado para

	retornar el resultado de la petición.

getClosestCityID

public void getClosestCityID(LatLng position, WeatherDataService.CityIDResponseListener volleyResponseListener)

Devuelve el id de la ciudad registrada en MetaWeather más cercana a las coordenadas especificadas a través del listener una vez se haya completado la petición o un mensaje de error en caso contrario.

Parámetros

Nombre	Tipo	Descripción
Position	LatLng	Coordenadas de las que queremos saber la posición de la ciudad más cercana registrada en el servicio.
CityIDResponseListener	CityIDResponseListener	Listener utilizado para retornar el resultado de la petición.

Otras clases de uso

Vamos a describir clases que utilizamos para manejar información. Ya sean alumnos, registros de faltas, prácticas o predicciones meteorológicas.

FALTA

public class Falta
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de las faltas leves, deficientes y eliminatorias del fichero JSON de configuración.

VARIABLES

codigo

String codigo

Código de la falta con la sección y el número de falta.

titulo

String titulo

Título de la falta.

descripcion

String descripcion

Motivo de la falta.

pauta

String pauta

Punto en el que puede mejorar el alumno para que no vuelva a cometer dicha falta.

MÉTODOS

Getters y setters de las variables mencionadas.

FaltaPost

public class FaltaPost
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de las faltas cometidas en las prácticas registradas en la base de datos de Firebase.

Variables

acc

private float acc

Precisión de la toma de posición registrada en el momento de la falta.

alt

private double alt

Altitud de la toma de posición registrada en el momento de la falta.

codigo

private String codigo

Codigo de la falta con la sección y el número de falta.

latitud

private double latitude

Coordenada de latitud de la toma de posición registrada en el momento de la falta.

longitud

private double longitude

Coordenada de longitud de la toma de posición registrada en el momento de la falta.

repeticiones

private Integer repeticiones

Número de veces en las que se repite la falta. La utilizamos a la hora de mostrar las faltas totales del alumno.

tiempo

private String tiempo

Tiempo de práctica en la que ocurrió la incidencia. Tanto las horas como los minutos.

tipo

private String tipo

Gravedad de la falta. Puede ser *Leve*, *Deficiente* o *Eliminatoria*.

titulo

private String titulo

Nombre de la falta.

velocidad

private float velocidad

Velocidad de la toma de posición registrada en el momento de la falta.

MÉTODOS

Getters y setters de las variables mencionadas.

LocationPoint

public class LocationPoint
extends java.lang.Object

Utilizaremos esta clase para recoger las coordenadas de latitud y longitud de las prácticas registradas en la base de datos de Firebase.

Variables

longitude

private double longitude

Coordenada de longitud.

latitude

private double latitude

Coordenada de latitud.

Métodos

Getters y setters de las variables mencionadas.

PRACTICA

public class Practica
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de las prácticas de alumno registradas en la base de datos de Firebase.

VARIABLES

anotacion

private String anotacion

Anotación del profesor acerca de la práctica.

date

private String date

Día, mes y año en el que se realizó la práctica en el formato dd-MM-yyyy.

endPoint

private LocationPoint endpoint

Coordenadas de la posición en la que se terminó la práctica.

endTime

private String endTime

Horas y minutos en los que se terminó la práctica en el formato 24 horas.

faltas

private List<FaltaPost> faltas

Listado de faltas recogidas en el desarrollo de la práctica.

numeroPractica

private int numeroPractica

Número de práctica en el registro del alumno. Lo utilizamos en el detalle de las anotaciones de cada práctica.

profesor

private String profesor

Nombre del profesor encargado de realizar dicha práctica.

startPoint

private LocationPoint startPoint

Coordenadas de la posición en la que se inició la práctica.

startTime

private String startTime

Horas y minutos en los que se inició la práctica en el formato 24 horas.

trayectoria

private List<LocationPoint> trayectoria

Lista de coordenadas de latitud y longitud de la trayectoria seguida por el vehículo. Lo necesitaremos para trazar la práctica en el mapa en forma de polilínea.

weather

private String weather

Condiciones meteorológicas de la práctica según las predicciones de la API Rest de MetaWeather.

MÉTODOS

Getters y setters de las variables mencionadas.

SECCION

public class Seccion
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de las secciones del fichero JSON de configuración.

VARIABLES

codigo

java String codigo

Código de la sección.

titulo

java String titulo

Nombre de la sección.

leves

boolean leves

Nos indica si hay faltas leves de dicha sección.

deficientes

boolean deficientes

Nos indica si hay faltas deficientes de dicha sección.

eliminatorias

boolean eliminatorias

Nos indica si hay faltas eliminatorias de dicha sección.

MÉTODOS

Getters y setters de las variables mencionadas.

USER

public class User
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de los usuarios registrados en la base de datos de Firebase.

VARIABLES

id

private String id

Id del usuario.

nombre

private String nombre

Nombre de usuario.

apellido1

private String apellido1

Primer apellido del usuario.

apellido2

private String apellido2

Segundo apellido del usuario.

dni

private String dni

Dni del usuario sin letra.

fechaCreacion

private String fechaCreacion

Fecha en la que añadió el usuario a la base de datos. Formato dd/MM/yyyy.

fechaNacimiento

private String fechaNacimiento

Fecha de nacimiento del usuario. Formato dd/MM/yyyy.

listoExamen

private boolean listoExamen

Nos indica si el usuario está listo para examen o no.

password

private String password

Contraseña del usuario cifrada en UTF-8.

permiso

private String permiso

Permiso del que está matriculado el usuario. Puede ser *B, D, C, A1* o *A2.*

profesor

private String profesor

Id del profesor asignado al alumno.

rol

private String rol

Perfil de usuario. Puede ser *Alumno*, *Profesor* o *Administrador*.

correo

private String correo

Correo electrónico del usuario. Ha de tener un formato válido (someone@example.com).

numeroTelef

private String numeroTelef

Número de teléfono del usuario.

avatar

private Bitmap avatar

Imagen de Usuario. Esta variable se utiliza una vez tengamos la imagen de usuario de Firebase Storage.

imageDefault

private boolean imageDefault

Nos indica si hay o no registrada una imagen de este usuario en Firebase Storage. En cuyo caso se utilizaría una imagen por defecto, de ahí el nombre.

MÉTODOS

Getters y setters de las variables mencionadas.

WEATHERREPORTMODEL

class WeatherReportModel
extends java.lang.Object

Utilizaremos esta clase para recoger los registros de las predicciones meteorológicas registradas en la API de MetaWeather. En su mayoría haremos uso del nombre del estado del tiempo.

VARIABLES

id

private int id

Identificador interno de la predicción meteorológica.

weather_state_name

private String weather_state_name

Descripción del estado del tiempo.

weather_state_abbr

private String weather_state_abbr

Abreviación de una o dos palabras del estado del tiempo.

wind_direction_compass

private String wind_direction_compass

Punto del compás al que apunta la dirección del viento.

applicable_date

private String applicable_date

Fecha a la que pertenece la predicción del tiempo.

min_temp

private float min_temp

Temperatura minima en centígrados.

max_temp

private float max_temp

Temperatura máxima en centígrados.

the_temp

private float the_temp

Temperatura promedio en centígrados.

wind_speed

private float wind_speed

Velocidad del viento en millas por hora.

wind_direction

private float wind_direction

Dirección del viento en grados.

air_pressure

private int air_pressure

Presión del viento en milibares.

humidity

private int humidity

Humedad del tiempo en porcentaje.

visibility

private float visibility

Visibilidad del tiempo en millas.

predictability

private int predictability

Precisión de la predicción meteorológica en porcentaje.

MÉTODOS

Getters y setters de las variables mencionadas.

Interfaces de funcionalidades básicas

En este apartado vamos a describir las interfaces que desarrollan la lógica y el modelo de cada pantalla.

Pantalla Login

Pantalla de inicio de sesión.

LOGINPRESENTER

public class LoginPresenter
extends java.lang.Object
implements LoginContract.Presenter

MÉTODOS

onLoginButtonClicked

public void onLoginButtonClicked(String id, String password)

Desactiva los elementos de la vista, hace visible la barra de progreso y accede a los usuarios a través del modelo.

Parámetros

Nombre	Tipo	Descripción
id	String	Id o dni introducido por el usuario.
password	String	Contraseña introducida por el usuario.

validateUser

public void validateUser(String id, String password, List<User> users)

Valida las credenciales introducidas por el usuario. Si coincide con las de algún usuario de

la base de datos, navega a la pantalla Perfil, en caso contrario presenta un *Toast* en pantalla.

Parámetros		
Nombre	**Tipo**	**Descripción**
id	String	Id o dni introducido por el usuario.
password	String	Contraseña introducida por el usuario.
users	List<User>	Lista de usuarios de la base de datos.

LOGINMODEL

public class LoginModel
extends java.lang.Object
implements LoginContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

LoginModel

public LoginModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros		
Nombre	**Tipo**	**Descripción**
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al método del repositorio para que inicie la recogida de usuarios de Firebase.

Parámetros		
Nombre	**Tipo**	**Descripción**
callback	String	Para no interrumpir la ejecución del hilo principal de la aplicación, le pasamos al repositorio un *callback* que devolverá los datos cuando estén listos.

fetchPracticasFromFirebase

public void fetchPracticasFromFirebase(java.lang.String id)

Llama al método del repositorio para que inicie la recogida de prácticas de Firebase.

Pantalla Perfil

Pantalla de perfil que se muestra una vez hayamos iniciado sesión con alguno de los 3 perfiles.

PERFILPRESENTER

public class PerfilPresenter
extends java.lang.Object
implements PerfilContract.Presenter

CONSTANTES

ALUMNO_MODE

public static final int ALUMNO_MODE

Dígito que identifica el perfil de alumno. Ya que la pantalla se comparte en los distitnos perfiles lo utilizaremos para que la vista determine en que perfil se encuentra.

PROFESOR_MODE

public static final int PROFESOR_MODE

Dígito que identifica el perfil de profesor. Ya que la pantalla se comparte en los distintos perfiles lo utilizaremos para que la vista determine en que perfil se encuentra.

ADMINISTRATION_MODE

public static final int ADMINISTRATION_MODE

Dígito que identifica el perfil de administrador. Ya que la pantalla se comparte en los distintos perfiles lo utilizaremos para que la vista determine en que perfil se encuentra.

MÉTODOS

distinguirRolYActualizarVista

private void distinguirRolYActualizarVista()

Dependiendo del rol de usuario, actualiza la vista acordemente.

onItem1Clicked

public void onItem1Clicked()

Dependiendo del rol de usuario, navega a la siguiente pantalla correspondiente al primer botón de la interfaz de usuario.

onItem2Clicked

public void onItem2Clicked()

Dependiendo del rol de usuario, navega a la siguiente pantalla correspondiente al segundo botón de la interfaz de usuario.

onLogoutClicked

public void onLogoutClicked()

Llama a la vista para lanzar una ventana emergente de confirmación.

logout

public void logout()

Realiza el cierre de sesión, reiniciando las variables de estado y volviendo a la pantalla de Login.

getAvatarImageFromFirebase

private void getAvatarImageFromFirebase()

Accede a la referencia de Firebase Storage para descargar la imagen de usuario correspondiente y manejarla como un array de bytes.

saveImage

public void saveImage(android.graphics.Bitmap image)

Almacena la imagen de tipo Bitmap como variable de estado en la clase ViewModel.

PERFILMODEL

public class PerfilModel
extends java.lang.Object
implements PerfilContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

PerfilModel

public PerfilModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

removePracticasEventListener

public void removePracticasEventListener()

Llama al método del repositorio que desvincula el listener de eventos de Firebase para dejar de notificar cambios en la misma. Lo llamamos al cerrar sesión de alumno.

Pantalla Alumnos

Utilizamos esta pantalla para mostrar el listado de alumnos. Puede aparecer en múltiples ocasiones ya que se comparte para las listas de alumnos del perfil de administrador o de un profesor en concreto.

ALUMNOSPRESENTER

public class AlumnosPresenter
extends java.lang.Object
implements AlumnosContract.Presenter

MÉTODOS

disableClickableElements

private void disableClickableElements()

Desactiva los elementos clickeables de la vista.

enableClickableElements

private void enableClickableElements()

Activa los elementos clickeables de la lista.

filtrar

private void filtrar()

Filtra la lista de alumnos del ViewModel dependiendo de las casillas que haya marcado el usuario antes de aplicar el filtro.

getAlumnosFromProfesor

private List<User> getAlumnosFromProfesor(java.util.List<User> users, User profesor)

Extrae los alumnos asociados a un profesor de prácticas en particular.

Retorno

Tipo	Descripción
List<User>	Lista de alumnos de dicho profesor.

getAlumnosFromUserList

public List<User> getAlumnosFromUserList(List<User> users)

Devuelve los alumnos de la lista de usuarios.

Parámetros

Nombre	Tipo	Descripción
users	List<User>	Lista de usuarios.

Retorno

Tipo	Descripción
List<User>	Lista de alumnos.

getUserActual

public User getUserActual()

Devuelve el usuario que está haciendo uso de la pantalla.

getUserList

public void getUserList(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al modelo para recoger la lista de usuarios con un callback.

Parámetros

Nombre	Tipo	Descripción
callback	boolean	Callback que maneja los datos de usuario cuando estén listos.

onActionButtonClicked

public void onActionButtonClicked()

Realiza la lógica necesaria para iniciar la navegación a la pantalla de añadir alumno debido a que se ha pulsado el *ActionButton*.

isAlumnosFromProfesorSelection

public boolean isAlumnosFromProfesorSelection()

Indica si nos encontramos o no en la pantalla de selección de alumnos de un profesor.

Retorno

Tipo	Descripción
boolean	Si nos encontramos o no en la pantalla de selección de alumno de un profesor.

onFiltrarApplied

public void onFiltrarApplied(boolean id, boolean alfabetico, boolean b, boolean c, boolean d, boolean a1, boolean a2, boolean examen)

Comienza la aplicación del filtrado y ordenado dependiendo de las casillas que haya marcado el usuario.

Parámetros

Nombre	Tipo	Descripción
id	boolean	Indica si el radio botón de ordenado por id está marcada.
alfabetido	boolean	Indica si el radio botón de ordenado alfabéticamente está marcada.
b	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.
c	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.
D	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.
a1	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.
a2	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.
examen	boolean	Indica si la casilla de filtrado por examen de permiso B está marcada.

onFiltrarClosed

public void onFiltrarClosed()

Cierra la Ventana de filtrado y ordenado.

onReloadButtonClicked

public void onReloadButtonClicked()

Vuelve a realizar la consulta de los usuarios a Firebase para actualizar los datos del listado. Mientras esto ocurre, bloquea ciertos elementos de la interfaz de usuario hasta que se complete el proceso.

selectUserData

public void selectUserData(User user)

Realiza la lógica necesaria para iniciar la navegación a la pantalla del detalle de dicho alumno.

sortByAlphabet

private void sortByAlphabet()

Ordena los alumnos de manera alfabética haciendo uso de un comparador (*Comparator*).

sortById

private void sortById()

Ordena los alumnos por su id haciendo uso de un comparador (*Comparator*).

AlumnosModel

public class AlumnosModel
extends java.lang.Object
implements AlumnosContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

AlumnosModel

public AlumnosModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al método del repositorio para que inicie la recogida de usuarios de Firebase.

Parámetros

Nombre	Tipo	Descripción
callback	String	Para no interrumpir la ejecución del hilo principal de la aplicación, le pasamos al repositorio un *callback* que devolverá los datos cuando estén listos.

Pantalla Detail_Alumno

Pantalla del detalle del alumno. Se comparte para el perfil de administrador y profesor

con cambios en la barra de herramientas.

DETAIL_ALUMNOPRESENTER

public class Detail_AlumnoPresenter
extends java.lang.Object
implements Detail_AlumnoContract.Presenter

CONSTANTES

PERMISSIONS_FINE_LOCATION

private static final int PERMISSIONS_FINE_LOCATION

En el perfil de profesor, utilizaremos este identificador para pedir permiso de uso de los servicios de localización en el caso de que no los hubiera y su verificación.

PROFESOR_MODE

public static final int PROFESOR_MODE

Dígito que identifica el perfil de profesor. Ya que la pantalla se comparte en los distintos perfiles lo utilizaremos para que la vista determine en que perfil se encuentra.

ADMINISTRADOR_MODE

public static final int ADMINISTRATION_MODE

Dígito que identifica el perfil de administrador. Ya que la pantalla se comparte en los distintos perfiles lo utilizaremos para que la vista determine en que perfil se encuentra.

MÉTODOS

searchProfesor

private User searchProfesor()

Busca entre la lista de usuarios al profesor asignado al alumno del detalle.

Retorno

Tipo	Descripción
User	Profesor asignado al alumno.

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de usuario como Bitmap.

Parámetros

Nombre	Tipo	Descripción
image	Bitmap	Imagen de usuario.

onCreateOptionsMenu

public int onCreateOptionsMenu()

Le comunica a la vista la barra de herramientas superior que debe utilizar ya que varían respecto a un perfil de usuario u otro.

Retorno

Tipo	Descripción
int	Tipo de usuario a utilizar según las constantes globales.

onConfigActionClicked

public void onConfigActionClicked()

Realiza la lógica necesaria para para navegar a la pantalla de editar alumno del perfil de

administrador.

onConfigActionClicked2

public void onConfigActionClicked2()

Realiza la lógica necesaria para para navegar a la pantalla de configurar alumno del perfil de profesor.

onPracticaActionClicked

public void onPracticaActionClicked()

Si la aplicación tiene permisos para acceder a los servicios de navegación, navega a la pantalla de práctica. En caso contrario, solicita que el usuario atribuya los permisos.

onPermissionsResult

public void onPermissionsResult(int requestCode, int[] grantResults)

Si se han otorgado los permisos de localización, navega a la pantalla de práctica. En caso contrario le comunica a la vista que presente un mensaje en pantalla.

navigateToPractica

private void navigateToPractica()

Recupera la fecha y hora actuales y realiza la lógica necesaria antes de comunicarle a la vista que navegue a la pantalla de práctica.

getAvatarImageFromFirebase

private void getAvatarImageFromFirebase()

Accede a la referencia de Firebase Storage para descargar la imagen de usuario correspondiente y manejarla como un array de bytes.

DETAIL_ALUMNOMODEL

public class Detail_AlumnoModel
extends java.lang.Object
implements Detail_AlumnoContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

Detail_AlumnoModel

public Detail_AlumnoModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getUserList

public List<User> getUserList()

Recupera la lista de usuarios almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<User>	Lista de usuarios del repositorio.

Pantalla Edit_Alumno

Pantalla en la que podemos editar al alumno en el perfil de administrador.

EDIT_ALUMNOPRESENTER

public class Edit_AlumnoPresenter
extends java.lang.Object
implements Edit_AlumnoContract.Presenter

MÉTODOS

onDataSet

public void onDataSet(int year, int month, dayOfMonth)

Recoge la fecha seleccionada por el usuario en el *DatePickerDialog* de la vista.

Parámetros		
Nombre	**Tipo**	**Descripción**
year	int	Año seleccionado por el usuario.
month	int	Mes seleccionado por el usuario.
dayOfMonth	int	Día seleccionado por el usuario.

onOptionsItemSelected

public void onOptionsItemSelected(String nombre, String ap1, String ap2, String dni, String correo, String telefono, String permiso, String fecha, String profesor, boolean listoExamen)

Recupera la información introducida en la vista para, tras comprobar que los datos son válidos, modificar los registros del usuario en la base de datos de Firebase.

Parámetros

Nombre	Tipo	Descripción
nombre	String	Nombre del alumno.
ap1	String	Primero apellido del alumno.
ap2	String	Segundo apellido del alumno.
dni	String	Dni del alumno sin letra.
correo	String	Correo del alumno. Ha de tener un formato válido. Ej: (someone@example.com).
telefono	String	Número de teléfono del alumno.
permiso	String	Permiso del que está matriculado el alumno. Puede ser *B*, *C*, *D A1* o *A2*
fecha	String	Fecha de nacimiento del alumno. Formato dd/MM/yyyy.
profesor	String	Nombre del profesor asignado al alumno. "*No especificado*" en caso de que no se asigne a ninguno.
listoExamen	boolean	Nos indica si el alumno está listo para realizar examen o no.

saveEditTextState

public void saveEditTextState(String nombre, String ap1, String ap2, String dni, String correo, String numero, String permiso, String profesor, boolean listoExamen)

Guarda la información introducida en la vista por el usuario.

Parámetros		
Nombre	**Tipo**	**Descripción**
nombre	String	Nombre del alumno.
ap1	String	Primero apellido del alumno.
ap2	String	Segundo apellido del alumno.
dni	String	Dni del alumno sin letra.
correo	String	Correo del alumno. Ha de tener un formato válido. Ej: (someone@example.com).
telefono	String	Número de teléfono del alumno.
permiso	String	Permiso del que está matriculado el alumno. Puede ser *B*, *C*, *D A1* o *A2*
fecha	String	Fecha de nacimiento del alumno. Formato dd/MM/yyyy.
profesor	String	Nombre del profesor asignado al alumno. "*No especificado*" en caso de que no se asigne a ninguno.
listoExamen	boolean	Nos indica si el alumno está listo para realizar examen o no.

onEliminarClicked

public void onEliminarClicked()

Comunica a la vista que muestre una ventana emergente de confirmación (*AlertDialog*) avisando al usuario de si realmente quiere borrar el alumno.

eliminar

public void eliminar()

Elimina al alumno de la base de datos Firebase junto con toda su información.

onImageButtonClicked

public void onImageButtonClicked()

Comunica a la vista que inicie el servicio que proporciona la librería de selección de imágenes con opciones de escalado 1:1 para que sea un cuadrado.

Esta librería le permitirá al usuario elegir la imagen tanto de la galería como directamente de la cámara de fotos.

En su recuperación, dicha imagen es comprimida y subida a Firebase en formato jpg al 50% de la calidad para que ocupe menos espacio en la base de datos.

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de alumno como Bitmap.

Parámetros

Nombre	Tipo	Descripción
image	Bitmap	Imagen de alumno.

onProfesorButtonClicked

public void onProfesorButtonClicked()

Realiza la lógica necesaria para comunicarle a la vista la navegación hacia la lista de profesores. De forma que el usuario pueda seleccionar el profesor que quiera asignar al alumno.

onProfesorTextViewClicked

public void onProfesorTextViewClicked()

El usuario ha pulsado sobre el nombre del profesor señalando que no quiere especificar profesor asignado de prácticas. Así que lo eliminamos del estado y se lo comunicamos a la vista.

Pantalla Add_Alumno

Pantalla en la que añadimos un nuevo alumno en el perfil de administrador.

ADD_ALUMNOPRESENTER

public class Add_AlumnoPresenter
extends java.lang.Object
implements Add_AlumnoContract.Presenter

MÉTODOS

getUserList

public void getUserList(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al modelo para recoger la lista de usuarios con un callback.

Parámetros

Nombre	Tipo	Descripción
callback	boolean	Callback que maneja los datos de usuario cuando estén listos.

onConfirmButtonClicked

public void onConfirmButtonClicked(String nombre, String ap1, String ap2, String dni, String correo, String numero, String permiso, String fecha, String profesor)

Recupera la información introducida en la vista para, tras comprobar que los datos son válidos, añadir el usuario en la base de datos de Firebase con la contraseña cifrada en UTF-8.

Parámetros

Nombre	Tipo	Descripción
nombre	String	Nombre del alumno.

ap1	String	Primero apellido del alumno.
ap2	String	Segundo apellido del alumno.
dni	String	Dni del alumno sin letra.
correo	String	Correo del alumno. Ha de tener un formato válido. Ej: (someone@example.com).
numero	String	Número de teléfono del alumno.
permiso	String	Permiso del que está matriculado el alumno. Puede ser *B*, *C*, *D A1* o *A2*
fecha	String	Fecha de nacimiento del alumno. Formato dd/MM/yyyy.
profesor	String	Nombre del profesor asignado al alumno. "*No especificado*" en caso de que no se asigne a ninguno.

generateEmail

private void generateEmail(String idString, String password, String[] email, nombre,

String ap1, ap2, String dni)

Llama a la vista para que lance una aplicación de correo electrónico con destinatario, mensaje y asunto correspondientes a la información del alumno añadido. El mensaje le comunica al alumno sus credenciales de inicio de sesión y contraseña, entre otras cosas.

A la vuelta, la aplicación volverá al detalle del alumno porque se considerará completada la operación de añadir alumno.

Parámetros

Nombre	**Tipo**	**Descripción**
idString	String	Id del alumno.
password	String	Contraseña del alumno. Por defecto, comprende las últimas 6 cifras del dni.
ap1	String	Primer apellido del alumno.
ap2	String	Segundo apellido del alumno.
dni	String	Dni del alumno sin letra.

onDataSet

public void onDataSet(int year, int month, int dayOfMonth)

Recoge la fecha seleccionada por el usuario en el *DatePickerDialog* de la vista.

Parámetros

Nombre	Tipo	Descripción
year	int	Año seleccionado por el usuario.
month	int	Mes seleccionado por el usuario.
dayOfMonth	int	Día seleccionado por el usuario.

onImageButtonClicked

public void onImageButtonClicked()

Comunica a la vista que inicie el servicio que proporciona la librería de selección de imágenes con opciones de escalado 1:1 para que sea un cuadrado.

Esta librería le permitirá al usuario elegir la imagen tanto de la galería como directamente de la cámara de fotos.

En su recuperación, dicha imagen es comprimida y subida a Firebase en formato jpg al 50% de la calidad para que ocupe menos espacio en la base de datos.

onProfesorButtonClicked

public void onProfesorButtonClicked()

Realiza la lógica necesaria para comunicarle a la vista la navegación hacia la lista de profesores. De forma que el usuario pueda seleccionar el profesor que quiera asignar al alumno.

onProfesorTextViewClicked

public void onProfesorTextViewClicked()

El usuario ha pulsado sobre el nombre del profesor señalando que no quiere especificar profesor asignado de prácticas. Así que lo eliminamos del estado y se lo comunicamos a la vista.

saveEditTextState

public void saveEditTextState(String nombre, ap1, String ap2, String dni, String correo, String numero, String permiso, String profesor)

Guarda la información introducida en la vista por el usuario.

Parámetros		
Nombre	**Tipo**	**Descripción**
nombre	String	Nombre del alumno.
ap1	String	Primero apellido del alumno.
ap2	String	Segundo apellido del alumno.
dni	String	Dni del alumno sin letra.
correo	String	Correo del alumno. Ha de tener un formato válido. Ej: (someone@example.com).
numero	String	Número de teléfono del alumno.
permiso	String	Permiso del que está matriculado el alumno. Puede ser *B*, *C*, *D A1* o *A2*
profesor	String	Nombre del profesor asignado al alumno. "*No especificado*" en caso de que no se asigne a ninguno.

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de alumno como Bitmap.

Parámetros		
Nombre	**Tipo**	**Descripción**

image	Bitmap	Imagen de alumno.

ADD_ALUMNOMODEL

public class Add_AlumnoModel
extends java.lang.Object
implements Add_AlumnoContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

Add_AlumnoModel

public Add_AlumnoModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al método del repositorio para que inicie la recogida de usuarios de Firebase.

Parámetros

Nombre	Tipo	Descripción
callback	String	Para no interrumpir la ejecución del hilo principal de la aplicación, le pasamos al repositorio un *callback* que devolverá los datos cuando estén listos.

Pantalla Profesores

Utilizamos esta pantalla para mostrar el listado de profesores.

PROFESORESPRESENTER

public class ProfesoresPresenter
extends java.lang.Object
implements ProfesoresContract.Presenter

MÉTODOS

disableClickableElements

private void disableClickableElements()

Desactiva los elementos clickeables de la vista.

enableClickableElements

private void enableClickableElements()

Activa los elementos clickeables de la lista.

getProfesoresFromUserList

public List<User> getProfesoresFromUserList(List<User> users)

Devuelve los profesores de la lista de usuarios.

Parámetros

Nombre	Tipo	Descripción
users	List<User>	Lista de usuarios.

Retorno

Tipo	Descripción
List<User>	Lista de alumnos.

getUserActual

public User getUserActual()

Devuelve el usuario que está haciendo uso de la pantalla.

getUserList

public void getUserList(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al modelo para recoger la lista de usuarios con un callback.

Parámetros

Nombre	Tipo	Descripción
callback	boolean	Callback que maneja los datos de usuario cuando estén listos.

onActionButtonClicked

public void onActionButtonClicked()

Realiza la lógica necesaria para iniciar la navegación a la pantalla de añadir profesor debido a que se ha pulsado el *ActionButton*.

onReloadButtonClicked

public void onReloadButtonClicked()

Vuelve a realizar la consulta de los usuarios a Firebase para actualizar los datos del listado. Mientras esto ocurre, bloquea ciertos elementos de la interfaz de usuario hasta que se complete el proceso.

onCreateOptionsMenu

public boolean onCreateOptionsMenu()

Como la lista de profesores también se puede para asignarle un profesor a un alumno de entre los posibles, este método le comunica a la vista si nos encontramos en esta situación o no. De forma que pueda modificar la barra de herramientas superior de manera acorde.

Retorno	
Tipo	**Descripción**
boolean	Nos indica si nos encontramos en la lista de asignación de profesor o no.

selectUserData

public void selectUserData(User user)

Realiza la lógica necesaria para iniciar la navegación a la pantalla del detalle de dicho profesor.

ProfesoresModel

public class ProfesoresModel
extends java.lang.Object
implements ProfesoresContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

Constructor

ProfesoresModel

public ProfesoresModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al método del repositorio para que inicie la recogida de usuarios de Firebase.

Parámetros

Nombre	Tipo	Descripción
callback	String	Para no interrumpir la ejecución del hilo principal de la aplicación, le pasamos al repositorio un *callback* que devolverá los datos cuando estén listos.

Pantalla Detail_Profesor

Pantalla del detalle del profesor.

DETAIL_PROFESORPRESENTER

public class Detail_ProfesorPresenter
extends java.lang.Object
implements Detail_ProfesorContract.Presenter

MÉTODOS

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de usuario como Bitmap.

Parámetros		
Nombre	**Tipo**	**Descripción**
image	Bitmap	Imagen de usuario.

onConfigActionClicked

public void onConfigActionClicked()

Realiza la lógica necesaria para para navegar a la pantalla de editar profesor del perfil de administrado.

getAvatarImageFromFirebase

private void getAvatarImageFromFirebase()

Accede a la referencia de Firebase Storage para descargar la imagen de usuario correspondiente y manejarla como un array de bytes.

alumnosButtonClicked

public void alumnosButtonClicked()

Realiza la lógica necesaria para comunicarle a la vista la navegación hacia la lista de alumnos de dicho profesor. De forma que el administrador pueda visualizar los alumnos asignados a dicho profesor y su detalle.

DETAIL_PROFESORMODEL

public class Detail_ProfesorModel
extends java.lang.Object
implements Detail_ProfesorContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

Detail_ProfesorModel

public Detail_ProfesorModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getUserList

public List<User> getUserList()

Recupera la lista de usuarios almacenada en el repositorio.

Retorno

Tipo	Descripción
List<User>	Lista de usuarios del repositorio.

Pantalla Edit_Profesor

Pantalla en la que podemos editar al profesor en el perfil de administrador.

EDIT_PROFESORPRESENTER

public class Edit_ProfesorPresenter
extends java.lang.Object
implements Edit_ProfesorContract.Presenter

MÉTODOS

onDataSet

public void onDataSet(int year, int month, dayOfMonth)

Recoge la fecha seleccionada por el usuario en el *DatePickerDialog* de la vista.

Parámetros

Nombre	Tipo	Descripción
year	int	Año seleccionado por el usuario.
month	int	Mes seleccionado por el usuario.
dayOfMonth	int	Día seleccionado por el usuario.

onOptionsItemSelected

public void onOptionsItemSelected(String nombre, String ap1, String ap2, String dni, String correo, String telefono, String fecha)

Recupera la información introducida en la vista para, tras comprobar que los datos son válidos, modificar los registros del usuario en la base de datos de Firebase.

Parámetros		
Nombre	**Tipo**	**Descripción**
nombre	String	Nombre del profesor.
ap1	String	Primero apellido del profesor.
ap2	String	Segundo apellido del profesor.
dni	String	Dni del profesor sin letra.
correo	String	Correo del profesor. Ha de tener un formato válido. Ej: (someone@example.com).
telefono	String	Número de teléfono del alumno.
fecha	String	Fecha de nacimiento del profesor. Formato dd/MM/yyyy.

saveEditTextState

public void saveEditTextState(String nombre, String ap1, String ap2, String dni, String correo, String numero)

Guarda la información introducida en la vista por el usuario.

Parámetros		
Nombre	**Tipo**	**Descripción**
nombre	String	Nombre del profesor.
ap1	String	Primero apellido del profesor.
ap2	String	Segundo apellido del profesor.
dni	String	Dni del profesor sin letra.
correo	String	Correo del profesor. Ha de tener un formato válido. Ej: (someone@example.com).
numero	String	Número de teléfono del profesor.

onEliminarClicked

public void onEliminarClicked()

Comunica a la vista que muestre una ventana emergente de confirmación (*AlertDialog*) avisando al usuario de si realmente quiere borrar el profesor.

eliminar

public void eliminar()

Elimina al profesor de la base de datos Firebase junto con toda su información.

onImageButtonClicked

public void onImageButtonClicked()

Comunica a la vista que inicie el servicio que proporciona la librería de selección de imágenes con opciones de escalado 1:1 para que sea un cuadrado.

Esta librería le permitirá al usuario elegir la imagen tanto de la galería como directamente de la cámara de fotos.

En su recuperación, dicha imagen es comprimida y subida a Firebase en formato jpg al 50% de la calidad para que ocupe menos espacio en la base de datos.

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de profesor como Bitmap.

Parámetros		
Nombre	**Tipo**	**Descripción**
image	Bitmap	Imagen de profesor.

Pantalla Add_Profesor

Pantalla en la que añadimos un nuevo profesor en el perfil de administrador.

ADD_PROFESORPRESENTER

public class Add_ProfesorPresenter
extends java.lang.Object
implements Add_ProfesorContract.Presenter

MÉTODOS

getUserList

public void getUserList(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al modelo para recoger la lista de usuarios con un callback.

Parámetros

Nombre	Tipo	Descripción
callback	boolean	Callback que maneja los datos de usuario cuando estén listos.

onConfirmButtonClicked

public void onConfirmButtonClicked(String nombre, String ap1, String ap2, String dni, String correo, String numero, String fecha)

Recupera la información introducida en la vista para, tras comprobar que los datos son válidos, añadir el usuario en la base de datos de Firebase.

Parámetros

Nombre	Tipo	Descripción
nombre	String	Nombre del profesor.
ap1	String	Primero apellido del profesor.
ap2	String	Segundo apellido del profesor.
dni	String	Dni del profesor sin letra.
correo	String	Correo del profesor. Ha de tener un formato

		válido. Ej: (someone@example.com).
numero	String	Número de teléfono del profesor.
fecha	String	Fecha de nacimiento del profesor. Formato dd/MM/yyyy.

generateEmail

private void generateEmail(String idString, String password, String[] email, nombre, String ap1, ap2, String dni)

Llama a la vista para que lance una aplicación de correo electrónico con destinatario, mensaje y asunto correspondientes a la información del profesor añadido. El mensaje le comunica al profesor sus credenciales de inicio de sesión y contraseña, entre otras cosas.

A la vuelta, la aplicación volverá al detalle del profesor porque se considerará completada la operación de añadir profesor.

Parámetros		
Nombre	**Tipo**	**Descripción**
idString	String	Id del profesor.
password	String	Contraseña del profesor. Por defecto, comprende las últimas 6 cifras del dni.
ap1	String	Primer apellido del profesor.
ap2	String	Segundo apellido del profesor.
dni	String	Dni del profesor sin letra.

onDataSet

public void onDataSet(int year, int month, int dayOfMonth)

Recoge la fecha seleccionada por el usuario en el *DatePickerDialog* de la vista.

Parámetros		
Nombre	**Tipo**	**Descripción**

year	int	Año seleccionado por el usuario.
month	int	Mes seleccionado por el usuario.
dayOfMonth	int	Día seleccionado por el usuario.

onImageButtonClicked

public void onImageButtonClicked()

Comunica a la vista que inicie el servicio que proporciona la librería de selección de imágenes con opciones de escalado 1:1 para que sea un cuadrado.

Esta librería le permitirá al usuario elegir la imagen tanto de la galería como directamente de la cámara de fotos.

En su recuperación, dicha imagen es comprimida y subida a Firebase en formato jpg al 50% de la calidad para que ocupe menos espacio en la base de datos.

saveEditTextState

public void saveEditTextState(String nombre, ap1, String ap2, String dni, String correo, String numero)

Guarda la información introducida en la vista por el usuario.

Parámetros		
Nombre	**Tipo**	**Descripción**
nombre	String	Nombre del profesor.
ap1	String	Primero apellido del profesor.
ap2	String	Segundo apellido del profesor.
dni	String	Dni del profesor sin letra.
correo	String	Correo del profesor. Ha de tener un formato válido. Ej: (someone@example.com).
numero	String	Número de teléfono del profesor.

saveImage

public void saveImage(Bitmap image)

Almacena la imagen de profesor como Bitmap.

Parámetros

Nombre	Tipo	Descripción
image	Bitmap	Imagen de alumno.

ADD_PROFESORMODEL

public class Add_ProfesorModel
extends java.lang.Object
implements Add_ProfesorContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

CONSTRUCTOR

Add_ProfesorModel

public Add_ProfesorModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros		
Nombre	**Tipo**	**Descripción**
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

fetchUsersFromFirebase

public void fetchUsersFromFirebase(RepositoryContract.fetchUsersFromFirebaseCallback callback)

Llama al método del repositorio para que inicie la recogida de usuarios de Firebase.

Parámetros		
Nombre	**Tipo**	**Descripción**
callback	String	Para no interrumpir la ejecución del hilo principal de la aplicación, le pasamos al repositorio un *callback* que devolverá los datos cuando estén listos.

Pantalla Config_Alumno

Pantalla en la que el profesor puede establecer si un alumno está listo para examen o no.

CONFIG_ALUMNOPRESENTER

public class Config_AlumnoPresenter
extends java.lang.Object
implements Config_AlumnoContract.Presenter

MÉTODOS

onConfirmClicked

public void onConfirmClicked()

Si la información introducida en la vista es distinta al valor actual del usuario, modifica el registro de listo para examen del alumno en Firebase y vuelve al detalle del alumno. En caso contrario solo vuelve al detalle del alumno.

onListoExamenPressed

public void onListoExamenPressed(boolean checked)

Guarda el estado del *switch* de la vista que denota si el alumno está listo o no para examen.

Parámetros

Nombre	Tipo	Descripción
checked	boolean	Nos indica si el *switch* de la vista está activado o desactivado. Listo o no listo para examen.

Pantalla Practica

Pantalla en la que se realiza la práctica.

PRACTICAPRESENTER

public class PracticaPresenter
extends java.lang.Object
implements PracticaContract.Presenter

CONSTANTES

FALTA_LEVE_CODE

public static final int FALTA_LEVE_CODE

Dígito que identifica una falta de tipo leve. Lo utilizaremos para que la vista asigne a la lista de faltas un adaptador (*ArrayAdapter*) con las faltas leves.

FALTA_DEFICIENTE_CODE

public static final int FALTA_DEFICIENTE_CODE

Dígito que identifica una falta de tipo deficiente. Lo utilizaremos para que la vista asigne a la lista de faltas un adaptador (*ArrayAdapter*) con las faltas deficientes.

FALTA_ELIMINATORIA_CODE

public static final int FALTA_ELIMINATORIA_CODE

Dígito que identifica una falta de tipo eliminatoria. Lo utilizaremos para que la vista asigne a la lista de faltas un adaptador (*ArrayAdapter*) con las faltas eliminatorias.

DEFAULT_UPDATE_INTERVAL

private static final int DEFAULT_UPDATE_INTERVAL

Intervalo de tiempo por defecto en segundos en el que se realizarán las actualizaciones de posición a través de los servicios de *Google Play Services*.

FAST_UPDATE_INTERVAL

private static final int FAST_UPDATE_INTERVAL

Intervalo de tiempo mínimo en segundos en el que se realizarán las actualizaciones de posición a través de los servicios de *Google Play Services*.

MÉTODOS

setupLocationRequest

private void setupLocationRequest()

Método que inicia la configuración de acceso a la posición del dispositivo móvil a través de los servicios de *Google Play Services*. Concretamente hace uso del *FusedLocationProviderClient*.

Se realiza la configuración del cliente de manera que recibamos la posición del dispositivo a través de un *callback* cada cierto tiempo (definido en las constantes globales) y con una precisión alta.

Asignar una precisión alta a las tomas de posición tiene la desventaja de que el consumo de batería es algo mayor, pero lo hemos compensado reduciendo el tiempo entre actualizaciones de posición en 30 segundos por defecto y 5 segundos como mínimo para las tomas rápidas.

Hemos utilizado este método y no otros como el propio de Android (*LocationManager)* debido a que es la forma recomendada por la documentación oficial de Android para recibir actualizaciones de posición cada cierto tiempo.

El *callback* comprobará si entre la posición actual y la última posición registrada hay más una distancia de 100 metros entre sí a través de las coordenadas. En cuyo caso, llama a la interfaz que realiza la llamada a la API de direcciones de Google para trazar el trayecto en forma de polilínea.

onNightModeClicked

public void onNightModeClicked(boolean checked)

Configura el modo noche del mapa, el cual oscurece el estilo de mapa y adapta los elementos en pantalla para que se sigan apreciando sin dificultad.

Retorno	
Tipo	**Descripción**

boolean	Nos indica si se activado o desactivado el modo noche.

onMapReady

public void onMapReady()

Llama a actualizar la vista debido a que el mapa ya está listo para utilizarse.

onTimerSchedule

public void onTimerSchedule()

Se llama cada vez que el contador (*Timer)* de tiempo de práctica registra que ha pasado un minuto o 60000 milisegundos. Modifica las variables de tiempo (horas y minutos) acordemente y actualiza la vista.

onTimerCreated

public long onTimerCreated(long ms)

Si la pantalla se ha creado y destruido, le comunicamos a al contador (*Timer)* que inicie la cuenta con un offset o diferencia que se iguale a los milisegundos que faltaban para llegar al minuto dependiendo del tiempo en milisegundos actual y el tiempo en milisegundos del momento en el que se creó inició el contador.

En caso contrario, simplemente le comunica que empiece a contar dentro de un minuto o 60000 milisegundos.

Parámetros		
Nombre	**Tipo**	**Descripción**
ms	long	Tiempo actual en milisegundos.

onDataWeatherFetched

public void onDataWeatherFetched(WeatherReportModel weatherReportModel)

Nos indica el resultado de la predicción meteorológica una vez hayamos recibido datos de la consulta a la API REST de *MetaWeather*.

Parámetros		
Nombre	**Tipo**	**Descripción**
weatherReportModel	WeatherReportModel	Predicción meteorológica de la práctica.

onFaltaLeveClicked

public void onFaltaLeveClicked()

Realiza la lógica una vez el usuario haya pulsado el botón de falta leve. De manera que recupera la posición actual del dispositivo y le comunica a la vista que presente un listado de las secciones de las faltas leves.

onFaltaDeficienteClicked

public void onFaltaDeficienteClicked()

Realiza la lógica una vez el usuario haya pulsado el botón de falta deficiente. De manera que recupera la posición actual del dispositivo y le comunica a la vista que presente un listado de las secciones de las faltas deficiente.

onFaltaEliminatoriaClicked

public void onFaltaEliminatoriaClicked()

Realiza la lógica una vez el usuario haya pulsado el botón de falta eliminatorias. De manera que recupera la posición actual del dispositivo y le comunica a la vista que presente un listado de las secciones de las faltas eliminatorias.

onMicrophoneClicked

public void onMicrophoneClicked()

Le comunica a la vista que inicie un *Intent* de reconocimiento de voz en el caso de que

estuviera disponible para que el usuario pueda registrar el código de la falta con la voz.

En caso de que el dispositivo no admitiera esta función, muestra un mensaje en pantalla.

onSpeechRecognized

public void onSpeechRecognized(ArrayList<String> speechArray)

Según la información que ha interpretado el reconocimiento de voz, registra la falta correspondiente y añade un marcador o lo modifica en caso de que el usuario estuviera editando una falta. Si no hay códigos de falta que coincidan, presenta un mensaje en pantalla.

Parámetros

Nombre	Tipo	Descripción
speechArray	ArrayList<String>	Información del reconocimiento de voz.

onMarkerClicked

public void onMarkerClicked(Integer posicion)

Realiza la lógica necesaria para que cuando el usuario pulsa sobre un marcador de falta, aparezca información acerca de la toma de posición asignada al mismo (altitud, velocidad, precisión, tiempo de práctica) y dos botones con los que podrá **modificar** la falta o **eliminar** dicho marcador.

Parámetros

Nombre	Tipo	Descripción
posicion	Integer	Número del marcador pulsado.

onModMarkerClicked

public void onModMarkerClicked()

Cuando un usuario presiona sobre el botón de modificar la falta del marcador, le saldrá un listado de secciones y de faltas de esa sección para que vuelva a elegir la incidencia asignada a dicho marcador.

onDeleteMarkerClicked

public void onDeleteMarkerClicked()

Comunica a la vista que muestre una ventana emergente de confirmación para que el usuario decida si realmente quiere borrar o no ese marcador del mapa.

deleteMarkerConfirmed

public void deleteMarkerConfirmed()

Elimina el marcador de la falta del mapa y del almacenamiento.

onInfoWindowClicked

public void onInfoWindowClicked(com.google.android.gms.maps.model.Marker marker)

Realiza la lógica necesaria para navegar a la pantalla de *StreetView* con las coordenadas de dicho marcador.

Parámetros		
Nombre	**Tipo**	**Descripción**
marker	Marker	Marcador del que se ha pulsado la ventana de información.

onConfirmButtonClicked

public void onConfirmButtonClicked()

Comunica a la vista que muestre una ventana emergente de confirmación para que el usuario decida si realmente quiere finalizar o no ese la práctica.

onPracticaEnd

public void onPracticaEnd()

Realiza la lógica necesaria para finalizar la práctica. Trazando la última polilínea hasta el

punto final y navegando a la pantalla de fin de práctica (Practica_End).

uploadDataToFirebase

private void uploadDataToFirebase()

Añade la práctica al registro del alumno en Firebase con toda la información recabada en el transcurso de esta.

onMapClicked

public void onMapClicked()

Cuando el usuario pulsa sobre el mapa y se está mostrando información de un marcador, oculta dicha información.

onDirectionsDataService

public void onDirectionsDataService(com.google.android.gms.maps.model.PolylineOptions polylineOptions)

Nos indica el resultado del acceso a la API Rest de direcciones de Google Maps una vez se haya completado la consulta. De forma que se pueda añadir la polilínea al mapa actualizando la vista.

Parámetros

Nombre	Tipo	Descripción
polylineOptions	PolylineOptions	Opciones de polilínea con los puntos recogidos de la llamada a la API Rest de direcciones de Google Maps entre dos coordenadas.

onDirectionsDataServiceError

public void onDirectionsDataServiceError()

Recoge que se ha producido un error en el acceso a la API Rest de direcciones de Google Maps para que cuando haya que realizar otra petición se tenga esto en cuenta y se avise al

usuario con un mensaje en pantalla.

onFaltaListItemClicked

public void onFaltaListItemClicked(String itemAtPosition)

Si el usuario pulsa sobre una sección de la lista de secciones, presenta una lista de las faltas de dicha sección. Si, por otro lado, el usuario pulsa sobre el código de la falta en el listado, asocia almacena dicha incidencia y la añade en el mapa en forma de marcador ***(Marker)*** sobre las coordenadas en las que el profesor pulsó uno de los botones de falta (leve, deficiente o eliminatoria).

Si hubiera algún marcador a 5 metros de este último, se posicionará a 0.000025 grados de longitud adicionales (ligeramente al este), así evitamos que se solapen los marcadores.

Parámetros

Nombre	Tipo	Descripción
itemAtPosition	String	Posición del elemento pulsado en la lista.

onVolverClicked

public void onVolverClicked()

El usuario puede pulsar el botón de volver o atrás una vez se le haya mostrado tanto la lista de secciones como de faltas. En el primer caso, oculta el listado, en el segundo vuelve al listado de secciones.

PRACTICAMODEL

public class PracticaModel
extends java.lang.Object
implements PracticaContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

faltasLeves

private List<Falta> faltasLeves

Lista de faltas leves del archivo JSON de configuración.

faltasDeficientes

private List<Falta> faltasDeficientes

Lista de faltas deficientes del archivo JSON de configuración.

faltasEliminatorias

private List<Falta> faltasEliminatorias

Lista de faltas eliminatorias del archivo JSON de configuración.

secciones

private List<Seccion> secciones

Lista de secciones del archivo JSON de configuración.

CONSTRUCTOR

PracticaModel

public PracticaModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getFaltasLeves

public List<Falta> getFaltasLeves()

Recupera la lista de faltas leves almacenada en el repositorio.

Retorno

Tipo	Descripción
List<Falta>	Listado de faltas leves.

getFaltasDeficientes

public List<Falta> getFaltasDeficientes()

Recupera la lista de faltas deficientes almacenada en el repositorio.

Retorno

Tipo	Descripción
List<Falta>	Listado de faltas deficientes.

getFaltasEliminatorias

public List<Falta> getFaltasEliminatorias()

Recupera la lista de faltas eliminatorias almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas eliminatorias.

getSecciones

public List<Seccion> getSecciones()

Recupera la lista de secciones de las incidencias almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Seccion>	Listado de secciones de las faltas.

Pantalla Practica_End

Pantalla de fin de práctica en la que el profesor puede ver el balance de faltas y apuntar anotaciones.

PRACTICA_ENDPRESENTER

public class Practica_EndPresenter
extends java.lang.Object
implements Practica_EndContract.Presenter

CONSTANTES

FALTA_LEVE_CODE

public static final int FALTA_LEVE_CODE

Dígito que identifica una falta de tipo leve.

FALTA_DEFICIENTE_CODE

public static final int FALTA_DEFICIENTE_CODE

Dígito que identifica una falta de tipo deficiente.

FALTA_ELIMINATORIA_CODE

public static final int FALTA_ELIMINATORIA_CODE

Dígito que identifica una falta de tipo eliminatoria.

MÉTODOS

desglosarFaltas

private void desglosarFaltas()

Desglosa las faltas recibidas por la pantalla de prácticas en leves, deficientes y eliminatorias.

calcularResultado

private void calcularResultado()

Calcula si la práctica es apta o no apta dependiendo del número de faltas leves, deficientes y eliminatorias cometidas para preparar y familiarizar al alumno con el sistema de calificación que utilizará el examinador en el momento del examen.

La calificación será no apta si:

- Se comete una falta eliminatoria.
- Se cometen dos faltas deficientes.
- Se comete una falta deficiente y 5 leves.

- Se cometen 10 faltas leves.

calcularPuntosMejora

private void calcularPuntosMejora()

Extrae los puntos de mejora que debe seguir el alumno en función de las faltas cometidas. Si hubiera puntos de mejora que se repiten solo saldrán una vez. Los puntos de mejora tienen relación con la variable ***pauta*** de las faltas (*Falta*).

onConfirmButtonClicked

public void onConfirmButtonClicked(String text)

Sube las anotaciones del profesor a Firebase y vuelve a la pantalla del detalle de alumno.

Parámetros

Nombre	Tipo	Descripción
text	String	Anotaciones del profesor recogidas por la vista.

Pantalla Statistics

Pantalla en la que el alumno puede ver una lista desplegable con sus faltas totales leves, deficientes y eliminatorias.

STATISTICSPRESENTER

public class StatisticsPresenter
extends java.lang.Object
implements StatisticsContract.Presenter

MÉTODOS

setupExpandableListAdapter

private void setupExpandableListAdapter()

Inicia la configuración de la lista expandible, la cual tiene título y detalle para cada grupo.

getExpandableListTitle

public void getExpandableListTitle()

Genera el título de cada grupo de la lista, el cual debe contener el número de faltas de ese tipo entre paréntesis.

getExpandableListDetail

public void getExpandableListDetail()

Genera el detalle de los grupos de la lista expandible dependiendo de la gravedad de la falta. Para cada incidencia aparecerá el código y el nombre. Si hubiera faltas que se repiten, aparecerá en el nombre del detalle el número de ellas entre paréntesis.

onGroupExpand

public void onGroupExpand(int groupPosition)

Almacena el estado de la lista expandible cuando se expande algún grupo de misma. De forma que se mantengan expandidos los grupos que correspondan ante cambios en el ciclo de vida de la aplicación.

Parámetros

Nombre	Tipo	Descripción
groupPosition	int	Posición del grupo de la lista que ha expandido el usuario.

onGroupCollapse

public void onGroupCollapse(int groupPosition)

Almacena el estado de la lista expandible cuando se colapsa algún grupo de la lista. De forma que se mantengan colapsados los grupos que correspondan ante cambios en el ciclo de vida de la aplicación.

Parámetros		
Nombre	**Tipo**	**Descripción**
groupPosition	int	Posición del grupo de la lista que ha colapsado el usuario.

onChildClicked

public void onChildClicked(int groupPosition, int childPosition, long id)

Realiza la lógica necesaria para navegar hacia la pantalla del detalle de la falta seleccionada.

Parámetros		
Nombre	**Tipo**	**Descripción**
groupPosition	int	Posición del grupo de la lista.
childPosition	int	Posición del elemento de la lista.
id	long	Id del elemento.

onShareButtomClicked

public void onShareButtomClicked()

Comparte el balance de faltas totales y el porcentaje de faltas leves, deficientes y eliminatorias tenidas a lo largo de las distintas prácticas a través de un *Intent* a otras aplicaciones.

StatisticsModel

public class StatisticsModel
extends java.lang.Object
implements StatisticsContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

Constructor

StatisticsModel

public StatisticsModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

Métodos

getPracticas

public List<Practica> getPracticas()

Recupera la lista de prácticas almacenada en el repositorio.

Retorno

Tipo	Descripción
List<Practicas>	Lista de prácticas del repositorio.

Pantalla Statistics_Detail

Nos muestra el detalle de la falta cuando el usuario pulsa sobre un elemento del balance total de faltas.

STATISTICS_DETAILPRESENTER

public class Statistics_DetailPresenter
extends java.lang.Object
implements Statistics_DetailContract.Presenter

MÉTODOS

calcularFalta

private void calcularFalta()

Accede a la lista de faltas de la gravedad que corresponda (leve, deficiente o eliminatoria) y extrae la aquella que coincida con el código de la falta pulsada en la lista de la pantalla anterior.

calcularSeccion

private void calcularSeccion()

Accede a la lista de secciones y extrae aquella a la que pertenezca la falta pulsada en la lista de la pantalla anterior en función del código (primer dígito del código de falta coincide con el código de sección).

STATISTICS_DETAILMODEL

public class Statistics_DetailModel
extends java.lang.Object
implements Statistics_DetailContract.Model

VARIABLES

repository

private RepositoryContract repository

Referencia al repositorio.

faltasLeves

private List<Falta> faltasLeves

Lista de faltas leves del archivo JSON de configuración.

faltasDeficientes

private List<Falta> faltasDeficientes

Lista de faltas deficientes del archivo JSON de configuración.

faltasEliminatorias

private List<Falta> faltasEliminatorias

Lista de faltas eliminatorias del archivo JSON de configuración.

secciones

private List<Seccion> secciones

Lista de secciones del archivo JSON de configuración.

CONSTRUCTOR

Statistics_DetailModel

public Statistics_DetailModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getFaltasLeves

public List<Falta> getFaltasLeves()

Recupera la lista de faltas leves almacenada en el repositorio.

Retorno

Tipo	Descripción
List<Falta>	Listado de faltas leves.

getFaltasDeficientes

public List<Falta> getFaltasDeficientes()

Recupera la lista de faltas deficientes almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas deficientes.

getFaltasEliminatorias

public List<Falta> getFaltasEliminatorias()

Recupera la lista de faltas eliminatorias almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas eliminatorias.

getSecciones

public List<Seccion> getSecciones()

Recupera la lista de secciones de las incidencias almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Seccion>	Listado de secciones de las faltas.

Pantalla Practicas

Muestra el listado de prácticas realizadas por el alumno.

PracticasPresenter

public class PracticasPresenter
extends java.lang.Object
implements PracticasContract.Presenter

Métodos

selectPracticaNotas

public void selectPracticaNotas(Practica practica)

Realiza la lógica necesaria para navegar hacia la pantalla de notas de la práctica en cuestión.

Parámetros

Nombre	Tipo	Descripción
practica	Practica	Practica del listado seleccionada

selectPracticaMapa

public void selectPracticaMapa(Practica practica)

Realiza la lógica necesaria para navegar hacia la pantalla de mapa de la práctica en cuestión.

Parámetros

Nombre	Tipo	Descripción
practica	Practica	Practica del listado seleccionada.

onReloadButtonClicked

public void onReloadButtonClicked()

Accede de nuevo a las prácticas de usuario para recargar la información del listado.

PracticasModel

public class PracticasModel
extends java.lang.Object
implements PracticasContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

Constructor

PracticasModel

public PracticasModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

Métodos

getPracticas

public List<Practica> getPracticas()

Recupera la lista de prácticas almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Practicas>	Lista de prácticas del repositorio.

Pantalla Notas

Pantalla en la que se muestra la información de notas del detalle de la práctica en el perfil de alumno.

NotasPresenter

public class NotasPresenter
extends java.lang.Object
implements NotasContract.Presenter

Métodos

calcularNumeroFaltas

private void calcularNumeroFaltas()

Calcula el número de faltas leves, deficientes y eliminatorias de la práctica.

calcularResultado

private void calcularResultado()

Calcula si la práctica es apta o no apta dependiendo del número de faltas leves, deficientes

y eliminatorias cometidas para preparar y familiarizar al alumno con el sistema de calificación que utilizará el examinador en el momento del examen.

La calificación será no apta si:

- Se comete una falta eliminatoria.
- Se cometen dos faltas deficientes.
- Se comete una falta deficiente y 5 leves.
- Se cometen 10 faltas leves.

calcularPuntosMejora

private void calcularPuntosMejora()

Extrae los puntos de mejora que debe seguir el alumno en función de las faltas cometidas en dicha práctica. Si hubiera puntos de mejora que se repiten solo saldrán una vez. Los puntos de mejora tienen relación con la variable ***pauta*** de las faltas (*Falta*).

NotasModel

public class NotasModel
extends java.lang.Object
implements NotasContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

faltasLeves

private List<Falta> faltasLeves

Lista de faltas leves del archivo JSON de configuración.

faltasDeficientes

private List<Falta> faltasDeficientes

Lista de faltas deficientes del archivo JSON de configuración.

faltasEliminatorias

private List<Falta> faltasEliminatorias

Lista de faltas eliminatorias del archivo JSON de configuración.

CONSTRUCTOR

NotasModel

public NotasModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getFaltasLeves

public List<Falta> getFaltasLeves()

Recupera la lista de faltas leves almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas leves.

getFaltasDeficientes

public List<Falta> getFaltasDeficientes()

Recupera la lista de faltas deficientes almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas deficientes.

getFaltasEliminatorias

public List<Falta> getFaltasEliminatorias()

Recupera la lista de faltas eliminatorias almacenada en el repositorio.

Retorno	
Tipo	**Descripción**
List<Falta>	Listado de faltas eliminatorias.

Pantalla Mapa

Pantalla en la que se muestra el mapeado del detalle la práctica en el perfil de alumno.

MapaPresenter

public class MapaPresenter
extends java.lang.Object
implements MapaContract.Presenter

Métodos

extraerTrayectoria

private void extraerTrayectoria()

Genera las opciones de polilínea a presentar en el mapa a partir de los puntos de trayectoria de la práctica.

extraerMarcadoresYFaltas

private void extraerMarcadoresYFaltas()

Adquiere las faltas cometidas por el usuario en la práctica y genera los marcadores a presentar en el mapa.

getExpandableListTitle

public void getExpandableListTitle()

Genera el título de cada grupo de la lista, el cual debe contener el número de faltas de ese tipo entre paréntesis.

getExpandableListDetail

public void getExpandableListDetail()

Genera el detalle de los grupos de la lista expandible dependiendo de la gravedad de la falta. Para cada incidencia aparecerá el código, el nombre y el tiempo de práctica en el que sucedió.

onChildClicked

public void onChildClicked(int groupPosition, int childPosition, long id)

Le comunica a la vista que centre la cámara del mapa sobre la falta pulsada.

Parámetros

Nombre	Tipo	Descripción
groupPosition	int	Posición del grupo de la lista.
childPosition	int	Posición del elemento de la lista.
id	long	Id del elemento.

onGroupExpand

public void onGroupExpand(int groupPosition)

Almacena el estado de la lista expandible cuando se expande algún grupo de misma. De forma que se mantengan expandidos los grupos que correspondan ante cambios en el ciclo de vida de la aplicación.

Parámetros

Nombre	Tipo	Descripción
groupPosition	int	Posición del grupo de la lista que ha expandido el usuario.

onGroupCollapse

public void onGroupCollapse(int groupPosition)

Almacena el estado de la lista expandible cuando se colapsa algún grupo de la lista. De forma que se mantengan colapsados los grupos que correspondan ante cambios en el ciclo de vida de la aplicación.

Parámetros

Nombre	Tipo	Descripción
groupPosition	int	Posición del grupo de la lista que ha colapsado el usuario.

onInfoWindowClicked

public void onInfoWindowClicked(com.google.android.gms.maps.model.Marker marker)

Realiza la lógica necesaria para navegar a la pantalla de *StreetView* con las coordenadas de dicho marcador.

Parámetros

Nombre	Tipo	Descripción
marker	Marker	Marcador del que se ha pulsado la ventana de información.

onMapClicked

public void onMapClicked()

Cuando el usuario pulsa sobre el mapa y se está mostrando información de un marcador, oculta dicha información.

onMapReady

public void onMapReady()

Llama a actualizar la vista debido a que el mapa ya está listo para utilizarse.

onMarkerClicked

public void onMarkerClicked(Integer posicion)

Realiza la lógica necesaria para que cuando el usuario pulsa sobre un marcador de falta, aparezca información acerca de la toma de posición asignada al mismo (altitud, velocidad, precisión, tiempo de práctica).

Parámetros		
Nombre	**Tipo**	**Descripción**
posicion	Integer	Número del marcador pulsado.

onNightModeClicked

public void onNightModeClicked(boolean checked)

Configura el modo noche del mapa, el cual oscurece el estilo de mapa y adapta los elementos en pantalla para que se sigan apreciando sin dificultad.

Retorno	
Tipo	**Descripción**
boolean	Nos indica si se activado o desactivado el modo noche.

MapaModel

public class MapaModel
extends java.lang.Object
implements MapaContract.Model

Variables

repository

private RepositoryContract repository

Referencia al repositorio.

secciones

private List<Seccion> secciones

Lista de secciones del archivo JSON de configuración.

CONSTRUCTOR

MapaModel

public MapaModel(RepositoryContract repository)

En su creación, el modelo va a necesitar una referencia al repositorio para tener comunicación con este y acceder a sus métodos.

Parámetros

Nombre	Tipo	Descripción
repository	RepositoryContract	Referencia al repositorio.

MÉTODOS

getSecciones

public List<Seccion> getSecciones()

Recupera la lista de secciones de las incidencias almacenada en el repositorio.

Retorno

Tipo	Descripción
List<Seccion>	Listado de secciones de las faltas.

Acceso a los recursos de Google

Para acceder a las funciones de Google Maps, hemos tenido que crear una cuenta en el servicio de ***Google Cloud Platform.***

Este servicio nos proporciona una consola con la que podemos acceder a la información de uso de las API y rendimiento de los distintos proyectos que creemos. Para hacer uso de los mapas de Google en la aplicación solamente hemos tenido que crear un proyecto y generar una llave API asociada. Debido a que esta clave no puede ser compartida con nadie, hemos tenido que hacer el repositorio online **privado**.

Para hacer uso de algunos de los servicios a los que accede la aplicación, como el ***StreetView***, hemos tenido que asociar información de facturación al proyecto de Google Cloud Platform.

Cada servicio de Google Maps tiene su coste en función del número de veces que accedamos a ellos. Como simplemente estamos desarrollando la aplicación no hemos llegado al mínimo de peticiones necesarias en ninguno de los servicios para tener que pagar por ellos. Sin embargo, si la aplicación se distribuyera masivamente a clientes reales y empezara a utilizarse de manera considerable, Google nos empezaría a cobrar para los servicios de pago.

Concretamente, para el servicio de Street View, pagaríamos 14 dólares cada 1000 peticiones. A partir de las 100 mil peticiones el precio bajaría a 11 dólares y 20 centavos.

Firebase es propiedad de Google y aplica una filosofía parecida. En este caso, estamos usando el plan ***Spark*** al ser clientes gratuitos. Para la base de datos en tiempo real solo podemos tener 100 conexiones simultáneas, 1 GB de información almacenada y un volumen de datos de 10 GB al mes. Si quisiéramos que la aplicación se utilizase a gran escala, tendríamos que cambiar el plan a ***Blaze,*** donde nos empezarían a cobra en función del volumen de información que mueva nuestra aplicación y número de clientes o conexiones simultáneas.

Pliego de Condiciones

En este apartado vamos a detallar los recursos técnicos de los que se ha hecho uso en la elaboración del Trabajo de Fin de Grado y su memoria.

Recursos Software

Hemos hecho uso de los siguientes recursos software:

- **Microsoft Windows 10 Pro.** Sistema operativo con el que se ha realizado el trabajo. También se ha hecho uso de herramientas de Windows como los recortes de pantalla.
- **Microsoft Office 365.** Paquete de programas Office con el que realizamos la redacción de la memoria del trabajo.
- **Android Studio 4.1.2.** IDE con el que se ha realizado la programación de la aplicación.
- **GitHub.** Nos proporciona las herramientas para acceder de manera cómoda al repositorio online del proyecto software.
- **Figma.** Se ha utilizado para el diseño del aspecto de la aplicación y como herramienta de edición a la hora de realizar la guía de usuario.
- **Navegadores Web (Firefox, Google Chrome y Google Edge).** Cada navegador tiene sus ventajas y desventajas a la hora de acceder a internet.
- **Android 10.** Sistema operativo del dispositivo móvil con el que se ha testeado la aplicación.
- **Firebase.** Tanto la base de datos en tiempo real (Real-Time Database) como el almacenamiento en la nube (Firebase Storage).
- **Google Cloud Platform.** Para acceder a los servicios de Google Maps.

Recursos Hardware

Hemos hecho uso de los siguientes recursos hardware:

- **Ordenador de sobremesa**
  - **Procesador.** Intel Core i7 8700K - 3.7GHz - 6 núcleos
  - **Memoria Ram.** 32 GB RAM DDR4 – Corsair – 3200 MHz
  - **Disco Duro.** Samsung 970 EVO Plus – 1TB (V-NAND SSD)
  - **Tarjeta Gráfica.** Nvidia RTX 2070 Super – modelo MSI Gaming X Trio
- **Periféricos**
  - **Monitor.** LG 27GL850-B – 27 pulgadas.
  - **Teclado.** Logitech G513
  - **Ratón.** Logitech G502 HERO
- **Smartphone**
  - **Marca.** Huawei
  - **Modelo.** Honor 9X
  - **Sistema operativo.** Android 10
  - **Memoria Ram.** 4 GB RAM
  - **Almacenamiento.** 128 GB
  - **Tamaño de la pantalla.** 6.6 pulgadas.
  - **Resolución.** 2340 x 1080p
  - **Procesador.** HiSilicon Kirin 710F (12 nm)