
**Universidad de Valladolid**

**ESCUELA DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**
**Mención en Ingeniería del Software**

---

# Aplicación de alerta de proximidad de vehículos con comunicaciones MQTT y REST

---

**Alumno: Óscar Aragón Esteban**

**Tutor: Joaquín Nicolás Adiego Rodríguez**

*A mi madre, mi padre y mi hermana.*

# Agradecimientos

Gracias a toda mi familia, en especial a mis padres y mi hermana, por apoyarme en los momentos más difíciles.

Gracias a todo el grupo *Pystacho*, por todas las tardes de charla en *Discord*, y por todo lo que nos queda por vivir.

Gracias a Luis y a Guaza por acompañarme la mayor parte de los trabajos de la carrera y por convertirse en unos de mis mejores amigos.

Gracias a Berta por confiar más en mi que yo mismo.

Gracias en general a todos mis amigos que me han acompañado durante estos últimos años.

Gracias a todos mis compañeros de GMV por acogerme desde el primer día y por solucionar todas mis dudas.

Gracias a todos lo profesores de la Escuela de Ingeniería Informática que me han trasladado su pasión por esta profesión.

Gracias.

# Resumen

El objetivo de este proyecto es que el personal de mantenimiento de las vías de tren conozcan la posición de los trenes y reciban una alerta cuando el tren está cerca.

Los usuarios, después de iniciar sesión, podrán ver a tiempo real las posiciones de los trenes que actualmente están en marcha en un mapa, teniendo una perspectiva global de aquellos que están más cerca.

Además de ver los trenes dispondrán de información adicional de cada tren, como por ejemplo, el material que carga, en la vista de lista ordenada por cercanía al usuario. También podrán filtrar los trenes por sinóptico y configurar las alertas de tren cercano y de no actualización de posiciones.

Esta aplicación hará la vida más fácil a todo aquel personal de mantenimiento de la vía para optimizar y mantener su seguridad durante su trabajo.

**Palabras clave:** Posición, tren, *Kotlin*, *Jetpack Compose*, *Android*, localización, *MQTT*, topología y alerta.

# Abstract

The goal of this project is that the railway maintenance staff know the train location and receive an alert when a train is close.

Users, after logging in, will be able to see in real time the location of the trains in movement on a map, having a global perspective of those which are closer.

Furthermore, they will have additional information on each train on the train list view ordered by proximity to the user, such as the material they carried. They could also filter the trains by synoptic and configure nearby train and no position updates alerts.

This app would make life easier to the railway maintenance staff to optimize and keep their security during their work

**Keywords:** Position, train, *Kotlin*, *Jetpack Compose*, *Android*, location, *MQTT*, topology and alert.

# Índice general

# Lista de Figuras

# Lista de Tablas

# Capítulo 1

# Introducción

La informatización de la sociedad ha avanzado en muchos aspectos, muchas tareas se desarrollaban de forma manual como transcribir las cuentas de una empresa o bien tocar las vías del tren, comprobando que estas vibran, para así saber cuando un tren se está acercando.

Este proyecto precisamente trata de esto último, hacer la vida más fácil y ahorrar algún dolor que otro de espalda a aquellos encargados del mantenimiento de las vías de tren.

La aplicación está destinada a un operador ferroviario para que sus trabajadores la usen a diario, para que desempeñen su trabajo de una forma más óptima y sobre todo, más segura.

## 1.1. Contexto

El ferrocarril forma parte de una amplia gama de transportes terrestres, transportando tanto mercancías como personas en todo el mundo.

Mucha gente elige el ferrocarril como medio de transporte por sus múltiples ventajas respecto al resto de transportes terrestres. Bien es cierto que es importante conocer las necesidades de cada situación, como por ejemplo la urgencia de la entrega, el tipo de mercancías que se van a transportar, la cantidad, el tamaño, el destino de la carga, la distancia a recorrer, etc.

- **Ventajas:** Alguna de las ventajas por las que se opta por este tipo de transporte son la capacidad de carga y la rapidez frente al resto de transportes terrestres que tienen que lidiar con la congestión de tráfico. Además contamina considerablemente menos, y es que como dice la Asociación Ferroviaria Española, MAFEX, el ferrocarril representa sólo el 0,7 % de las emisiones totales de CO2 mientras que alcanza el 9 % de la demanda mundial de movilidad. Además, un tren de ocho vagones equivale al aforo de 15 autobuses y de 250 a 1.000 coches.

- **Desventajas:** España y Portugal tienen un ancho de vía diferente al resto de los países de Europa. Esto obliga a tener que trasbordar las mercancías o utilizar vagones con ejes intercambiables con la pérdida de tiempo y costes que esto conlleva. Además la mercancía solo podrá ser transportada hasta donde lleguen las vías, es decir, no puede llegar hasta almacenes o centros de producción específicos, esto significa que se necesita de otro transporte para trasbordar la carga hasta el sitio de almacenamiento o el lugar de operaciones.

## 1.2. Motivación

Los ferrocarriles además de ser una de las mejores opciones de transportes existentes, sus trabajadores merecen tener también las mejores condiciones. Además esta aplicación se podría adaptar a las necesidades de cualquier operador ferroviario siempre y cuando se adapte la interfaz de intercambio de datos.

En el caso de este proyecto la aplicación se desarrollará según los requisitos de la empresa contratista.

## 1.3. Objetivos

El objetivo principal de este proyecto es el de desarrollar una aplicación, que permita conocer a tiempo real la posición de los trenes que formen parte del operador ferroviario contratista que están en marcha o bien a punto de salir. Dichas posiciones se podrán leer en forma de mapa, con la punta de abajo del icono indicando la ubicación, o bien por medio de sus coordenadas numéricas.

Se busca una interfaz agradable a la vista e intuitiva, ya que no cuenta con demasiadas pantallas, es importante que el usuario sepa qué va a pasar cuando pulsa cada botón incluso la primera vez de uso.

También, si el usuario lo decide podrá recibir una notificación sonora, visual y/o de vibración cuando un tren esté dentro de un radio de metros configurable.

Se mostrarán todos los detalles que se disponen de cada tren, como por ejemplo su número de tren, la hora a la que se ha recibido la posición...

Cada miembro del personal de mantenimiento tendrá unas credenciales con las cuales podrá iniciar sesión dentro de la app.

Por último se podrán filtrar los trenes por sinóptico.

También existen los objetivos impuestos por el desarrollador:

- Conocer una tecnología actual y nueva para el alumno.

- Crear una interfaz agradable e intuitiva al usuario.
- Seguir una estructura de aplicación de acuerdo con los principios de *Clean Architecture* [1].
- Aprender a trabajar en equipo en un entorno de trabajo real.

## 1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

**Capítulo 2 Planificación:** Describe la metodología seguida para la realización del proyecto, asi como el análisis de riesgos y el cálculo del presupuesto.

**Capítulo 3 Requisitos:** En este capítulo se identificarán los requisitos, tanto funcionales, como no funcionales y de información.

**Capítulo 4 Análisis:** En él se incluye el modelo de dominio, descripción de casos de uso y diagramas de actividad.

**Capítulo 5 Tecnologías y herramientas utilizadas:** Descripción de las tecnologías y herramientas utilizadas durante el desarrollo de este proyecto

**Capítulo 6 Diseño:** Se muestra la arquitectura de la aplicación, así como el diseño de la base de datos y bocetos de la interfaz de usuario.

**Capítulo 7 Implementación:** Se explica cómo se han implementado las partes más importantes de la aplicación y que estrategias se han tomado.

**Capítulo 8 Pruebas:** Se recogen los resultados de las pruebas de aceptación realizadas durante el periodo de pruebas.

**Capítulo 9 Seguimiento del proyecto:** Muestra la comparativa entre la fecha planificada de cada fase y la fecha real tanto de comienzo como de final.

**Capítulo 10 Conclusiones y trabajo futuro:** Se realiza una valoración general de lo que ha sido el proyecto y se reflexiona sobre algunas posibles mejoras futuras.

**Bibliografía:** Referencias bibliográficas consultadas durante la realización del proyecto.

**Anexo A** [**Manuales** Incluye el manual de mantenimiento necesario para seguir el desarrollo del proyecto y lograr compilarlo en cualquier ordenador.

# Capítulo 2

# Planificación

En esta sección se describirá la planificación seguida para la realización de este proyecto englobará la metodología utilizada, el análisis de riesgos, se discutirá el presupuesto del desarrollo y el plan de trabajo que se deberá seguir en el periodo comprendido entre el 10 de febrero y el 24 de junio (límite solicitud de defensa de la primera convocatoria)

## 2.1. Metodología

Se utilizará la metodología en cascada también conocido como *one-shot*, es un proceso de desarrollo secuencial. Esta metodología requiere de un orden de realización de fases de comienzo a fin, es decir para realizar la fase de diseño se debe haber finalizado la fase previa de análisis. Como se ve en la Figura 2.1 está formado por 7 fases:

- **Estudio de Factibilidad:** Esta fase es muy importante, ya que te permitirá evaluar si la realización del proyecto será favorable, de modo que se podrá evaluar si se realiza o no. Dentro de este estudio, los tipos de factibilidad que se van a estudiar son los siguientes:
    1. **Factibilidad técnica:** Se descubrirá si se poseen las tecnologías necesarias para llevar a cabo el proyecto, tanto la infraestructura, el *software* a utilizar como los conocimientos de los desarrolladores.
    2. **Factibilidad legal:** Este análisis servirá para evaluar si el proyecto incumple alguna norma o ley estatal o municipal, ya que las consecuencias podrían ser muy graves pudiendo retrasar el inicio del proyecto.
    3. **Factibilidad de tiempo:** Permite conocer si el tiempo que se tiene planificado para llevar a cabo le proyecto coincide con el tiempo real.
- **Requisitos de usuario:** En esta fase se analizan las necesidades finales de la aplicación sin entrar en detalles técnicos de implementación

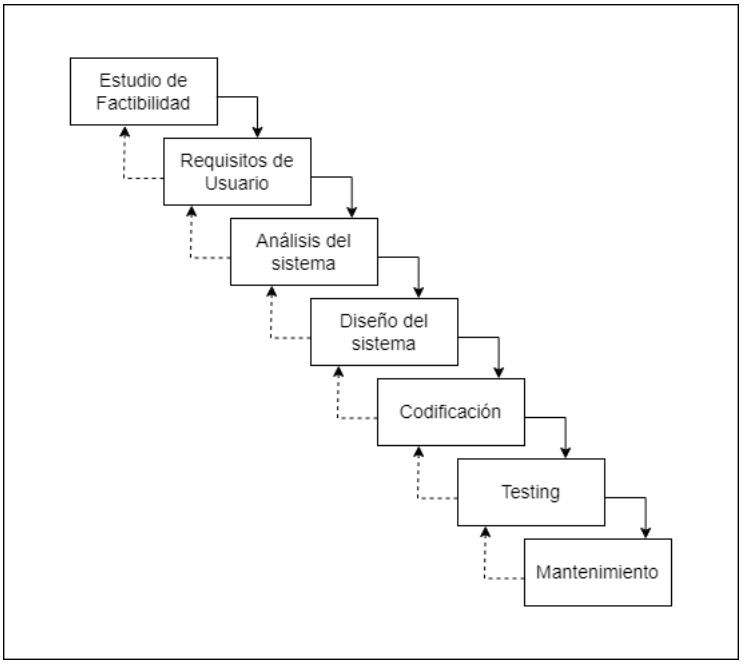



Figura 2.1: Diagrama de cascada

- **Análisis del sistema:** Esta fase se identificarán algunas de las funcionalidades que tendrá la aplicación, es decir, sus casos de uso, y la presentación del modelo del sistema.
- **Diseño del sistema:** Esta fase es una descripción de la estructura interna de la aplicación que sirve de base para su construcción.
- **Codificación:** En esta fase se elegirá la tecnología para desarrollar la programación de la aplicación, usando la información de las fases anteriores como guía.
- ***Testing:*** Esta será la fase de pruebas para verificar el correcto funcionamiento de la aplicación.
- **Mantenimiento:** Esta fase se realizará una vez se despliega la aplicación, incluye corrección de errores, mejoras de rendimiento, es decir, una mejora del *software* en general.

## 2.2. Análisis de riesgos

Según el PM-BOK un riesgo es *un evento o condición incierta que, si ocurre, tendrá un efecto positivo o negativo en los objetivos del proyecto*[2] La finalidad del análisis de riesgos es indentificar, analizar y tratar esos riesgos antes de que ocurran de modo que se pueda actuar según convenga en cada caso. No todos los riesgos afectarán de la misma manera al proyecto, por lo que es preciso definir una cuantía para clasificar dichos riesgos:

- **Probabilidad**: La probabilidad de materialización del riesgo, este valor será aproximado ya que es imposible saber con exactitud la probabilidad de ocurrencia.

| Probabilidad | Rango |
|---|---|
| Alta | Mayor de 50 % de probabilidad de ocurrencia. |
| Significante | 30-50 % de probabilidad de ocurrencia. |
| Moderada | 10-29 % de probabilidad de ocurrencia. |
| Baja | Menos de 10 % de probabilidad de ocurrencia. |

Tabla 2.1: Descriptores cualitativos de la probabilidad de un riesgo y sus valores de rango asociados

- **Impacto**: El impacto es la consecuencia efectiva que supondrá la ocurrencia del riesgo

| Impacto | Rango |
|---|---|
| Alta | Más de 30 % por encima del presupuesto. |
| Significante | 20-29 % por encima del presupuesto. |
| Moderada | 10-19 % por encima del presupuesto. |
| Baja | Menos de 10 % por encima del presupuesto. |

Tabla 2.2: Descriptores cualitativos del impacto de un riesgo y sus valores de rango asociados

- **Riesgos:**

| R001 | Fallo en la estimación |
|---|---|
| Descripción | Debido a que la estimación es una aproximación, es posible que no se sigan las fechas señaladas, si no se sigue la estimación afectará a la duración final del proyecto |
| Probabilidad | Moderada. |
| Impacto | Significante. |
| Acciones de mitigación | • Organizar un horario de trabajo en un entorno agradable |

Tabla 2.3: R001. Fallo en la estimación

| R002 | Desconocimiento de la tecnología |
|---|---|
| Descripción | Debido a que el proyecto se desarrollará en tecnologías nuevas para el desarrollador, es posible que afecte a la duración final del proyecto |
| Probabilidad | Significante. |
| Impacto | Moderada. |
| Acciones de mitigación | • Formación del desarrollador en las áreas nuevas para él |

Tabla 2.4: R002. Desconocimiento de la tecnología

| R003 | Cambio en los requisitos por parte del cliente. |
|---|---|
| Descripción | Debido a que los requisitos del proyecto están definidos por el cliente, la alteración o confusión en alguno de los requisitos puede suponer un cambio en duración final de proyecto |
| Probabilidad | Moderada. |
| Impacto | Significante. |
| Acciones de mitigación | • Planificar reuniones para discutir los posibles requisitos que presenten algún rasgo de ambigüedad. |

Tabla 2.5: R003. Cambio en los requisitos por parte del cliente

| R004 | Fallo en la red del entorno de trabajo. |
|---|---|
| Descripción | Debido a que el proyecto depende de conexión un fallo en la red del entorno de trabajo afectaría a la duración final del proyecto. |
| Probabilidad | Moderada. |
| Impacto | Moderado. |
| Acciones de mitigación | • Preparar un punto de conexión alternativo. |

Tabla 2.6: R004. Fallo en la red del entorno de trabajo.

| R005 | Disponibilidad del desarrollador. |
|---|---|
| Descripción | El desarrollador puede ver afectada su disponibilidad por motivos personales o de salud por lo tanto afectaría a la duración final del proyecto. |
| Probabilidad | Moderada |
| Impacto | Significante. |
| Acciones de mitigación | • Planificar las tareas segun su importancia y dejar tiempo de colchón entre ellas por posibles retrasos. |

Tabla 2.7: R005. Disponibilidad del desarrollador.

| R006 | Pérdida de ficheros. |
|---|---|
| Descripción | La pérdida de cualquier documento o fichero del proyecto puede llegar a afectar a la duración final del proyecto. |
| Probabilidad | Baja |
| Impacto | Alto. |
| Acciones de mitigación | • Guardar el proyecto en varios repositorios, tanto en la nube como locales. |

Tabla 2.8: R006. Pérdida de ficheros

| R007 | Vulnerabilidades Android |
|---|---|
| Descripción | El descubrimiento de una vulnerabilidad afectaría en la duración final del proyecto ya que habría que mejorar la seguridad de la aplicación. |
| Probabilidad | Baja |
| Impacto | Significante. |
| Acciones de mitigación | • Conocer las noticias del *mundo* de Android |

Tabla 2.9: R007. Vulnerabilidades Android

## 2.3. Coste

Para el presente proyecto se estimará un calendario cuya fecha de inicio se corresponde con el 10 de febrero de 2022 y fecha fin de proyecto prevista será el 10 de junio. Por tanto la duración estimada del proyecto será de 3,67 meses, siendo la jornada laboral de 8:00 a 14:00 de Lunes a Viernes, lo que hace un total de 84 días laborales, en total 498 horas de trabajo.

### 2.3.1. Coste del personal

El equipo que desarrollará el proyecto está formado por 4 personas: Jefe de proyecto, Analista, Desarrollador Back-end, Desarrollador Android.

Estimacion de costes[3]

- **Jefe de proyecto** Su trabajo no es exclusivamente a este proyecto, por lo tanto se estimará que durante el periodo de desarrollo del proyecto ha participado el 20 % de su horario.
  - **Sueldo promedio:** 48.445€/año.
  - **Sueldo durante el periodo del proyecto:** 12.111,25€
  - **Porcentaje de tiempo dedicado al proyecto:** 20 %
  - **Coste total:** 2.422,25€
- **Analista** Su trabajo tampoco es exclusivamente a este proyecto, por lo tanto, se fijará un porcentaje de 30 % de su horario dedicado al proyecto
  - **Sueldo promedio:** 32.000€/año.
  - **Sueldo durante el periodo del proyecto:** 8.000€
  - **Porcentaje de tiempo dedicado al proyecto:** 20 %
  - **Coste total:** 1600€
- **Desarrollador Back-end junior:**
  - **Sueldo promedio:** 18.964€/año.
  - **Sueldo durante el periodo del proyecto:** 4.741€
  - **Porcentaje de tiempo dedicado al proyecto:** 75 %
  - **Coste total:** 3.555,75€
- **Desarrollador Android:**
  - **Sueldo promedio:** 18.964€/año.
  - **Sueldo durante el periodo del proyecto:** 4.741€
  - **Porcentaje de tiempo dedicado al proyecto:** 100 %

- **Coste total:** 4.741€

**Costes de personal totales:**

| Rol | Coste estimado |
|---|---|
| Jefe de proyecto | 2.422,25€ |
| Analista | 1600€ |
| Desarrollador Back-End | 3.555,75€. |
| Desarrollador Android | 4.741€ |
| Total | 12.319€ |

Tabla 2.10: Tabla de costes de personal

### 2.3.2. Coste hardware

En esta sección se calculará una estimación de los costes hardware que supondrá el desarrollo del proyecto.

Debido a que la duración del proyecto se estima que será de 4 meses se calculará el coste porcentual correspondiente de los dispositivos según la vida útil de los mismos Los dispositivos hardware utilizados son:

| Dispositivo | Nombre | Vida útil y % de uso | Coste | Coste proyecto |
|---|---|---|---|---|
| Portátil 1 | Lenovo Thinkpad P14s | 7 años, 4.7 % de uso | 1100€ | 52,38€ |
| Portátil 2 | Hp Notebook | 7 años, 4.7 % de uso | 600€ | 28.2€ |
| Smartphone | Samsung Galaxy A12 | 4 años, 8.3 % de uso | 168€ | 14€ |
| Monitor | Lg Led 24” | 10 años, 3.3 % de uso | 120€ | 4€ |
| Teclado | Logitech K120 | 5 años, 6.67 % de uso | 10.5€ | 0,7€ |
| Ratón | Yocktec RGB | 5 años, 6.67 % de uso | 18€ | 1.2€ |
| Total | | | | 100,48€ |

Tabla 2.11: Tabla de costes de hardware

### 2.3.3. Coste software

Los costes de software serán de 0€ debido a que el software que se utilizará será totalmente gratuito

### 2.3.4. Costes fijos

En esta sección se describirán los gastos necesarios derivados del entorno de trabajo, como por ejemplo la electricidad, el agua, etc.

Los costes se han calculado teniendo en cuenta que se dedicarán 6 horas al día (1/4 de día) y que el periodo del proyecto son 4 meses, es decir el total será la suma del coste de cada servicio en un mes

**Costes fijos:**

| Servicio | Coste |
|---|---|
| Electricidad | 35€ |
| Agua | 30€ |
| Internet | 40€ |
| Calefacción | 70€ |
| Total | 175€ |

Tabla 2.12: Tabla de costes fijos

### 2.3.5. Coste total

El coste total será el resultado de sumar los costes calculados anteriormente. El coste total estimado es de 12.594,48€ o lo que es lo mismo 3.148,62€/mes.

| Tipo | Coste |
|---|---|
| Personal | 12.319€ |
| Hardware | 100,48€ |
| Software | 0€ |
| Costes fijos | 175€ |
| Total | 12.594,48€ |

Tabla 2.13: Tabla de costes totales

### 2.3.6. Presupuesto

Aunque los dos términos puedan sonar similares, existe una gran diferencia entre coste y presupuesto. En esta sección se ha calculado un coste estimado de todo el proyecto, pero esa

cantidad es muy diferente al presupuesto. El presupuesto se basa en recuperar el dinero de los costes, y aplicar un margen teórico para financiar algunos gastos derivados. En este caso se aplicará un porcentaje de 50 % sobre los costes para calcular el presupuesto.

El presupuesto final calculado es 18.891,72€

## 2.4. Plan de trabajo

El plan de trabajo describe las tareas y su estimación de tiempo que se deberán seguir para completar con éxito el proyecto, de esta forma se tendrá una visión global de todo el proyecto y decidir así la factibilidad de éste.

Los tiempos son una estimación, es decir, los tiempos reales del proyecto pueden variar por la manifestación de algún riesgo a pesar de los planes de mitigación escogidos.

El periodo de trabajo definido será del día 10 de febrero hasta el día 10 de junio de 2022, pero habrá unos días más por evitar cualquier tipo de retraso en la realización de tareas.

La fecha límite de finalización es el día 24 de junio de 2022. Si contamos desde el inicio hasta el día límite existen 94 días laborales.

Como ya se ha comentado anteriormente, el plan se divide en 7 fases tal y como se muestra en la Figura 2.2

| | | | | | |
|---|---|---|---|---|---|
| Aplicación móvil | 97 días | jue 10/02/22 | vie 24/06/22 | Óscar Aragón | 1 |
| Estudio de Factibilidad | 4 días | jue 10/02/22 | mar 15/02/22 | Óscar Aragón | 1.1 |
| Análisis de requisitos | 2 días | mié 16/02/22 | jue 17/02/22 | Óscar Aragón | 1.2 |
| Análisis del sistema | 6 días | vie 18/02/22 | vie 25/02/22 | Óscar Aragón | 1.3 |
| Diseño de Sistema | 13 días | lun 28/02/22 | mié 16/03/22 | Óscar Aragón | 1.4 |
| Codificación | 49 días | jue 17/03/22 | mar 24/05/22 | Óscar Aragón | 1.5 |
| Testing | 7 días | mié 25/05/22 | jue 02/06/22 | Óscar Aragón | 1.6 |
| Mantenimiento | 16 días | vie 03/06/22 | vie 24/06/22 | Óscar Aragón | 1.7 |

Figura 2.2: Plan de trabajo

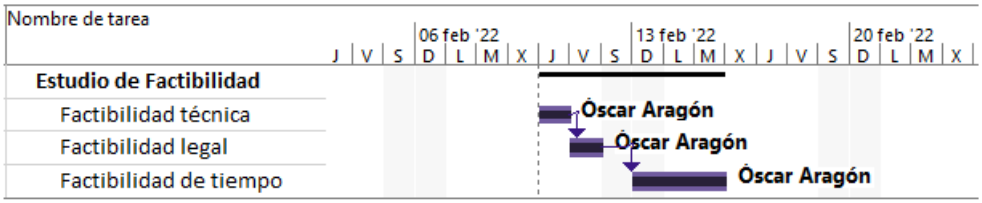



Figura 2.3: Diagrama de Gantt de la fase de Estudio de factibilidad


Figura 2.4: Diagrama de Gantt de la fase de análisis de requisitos

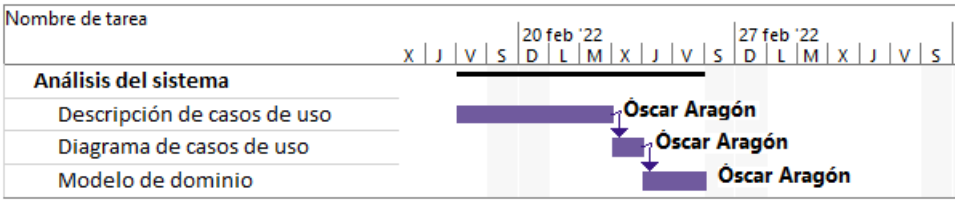



Figura 2.5: Diagrama de Gantt de la fase de análisis de sistema

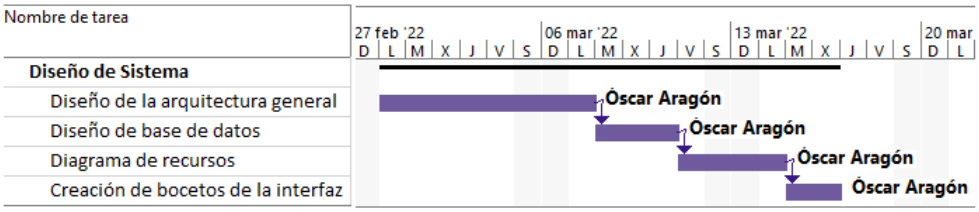



Figura 2.6: Diagrama de Gantt de la fase diseño de sistema

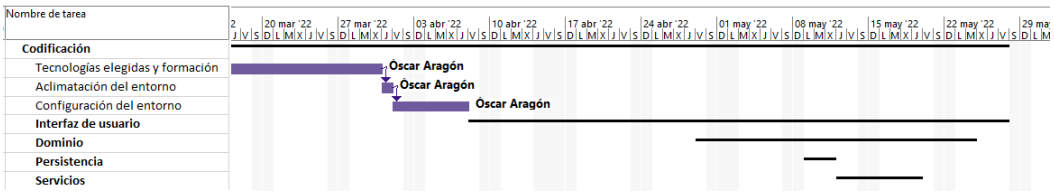



Figura 2.7: Diagrama de Gantt de la fase de implementación

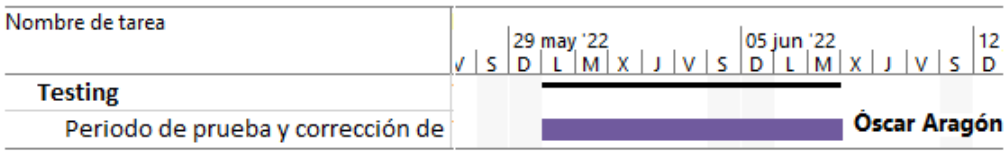



Figura 2.8: Diagrama de Gantt de la fase de pruebas

La fase de mantenimiento está pensada para que se desarrolle a lo largo del ciclo de vida de la aplicación, servirá para corregir errores aun no detectados y añadir mejoras futuras.

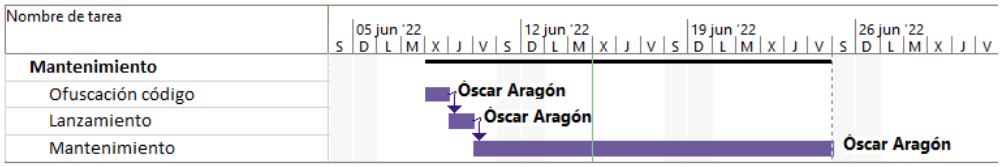



Figura 2.9: Diagrama de Gantt de la fase de mantenimiento

Los diagramas anteriores muestran la planificación de las tareas a realizar durante el proyecto. Al final de este documento se verá el tiempo real dedicado a cada una de ellas de forma que se compruebe la eficacia de la planificación.

# Capítulo 3

# Requisitos

## 3.1. Requisitos

Los requisitos son las características, las expectativas, los aspectos esperados o las capacidades que debe cumplir la aplicación, es decir es una descripción completa del sistema.

### 3.1.1. Requisitos funcionales

Los requisitos funcionales son aquéllos que describen qué debe hacer la aplicación

| Código | Requisito | Descripción |
|---|---|---|
| RF-001 | Inicio de sesión | La aplicación permitirá iniciar sesión con las credenciales propias del trabajador, rellenando los campos de usuario y contraseña. |
| RF-002 | Mapa | La aplicación deberá ser capaz de mostrar un mapa en la pantalla principal. |
| RF-003 | Trenes en mapa I | La aplicación deberá mostrar los trenes en trayecto o a punto de salir sobre el mapa. |
| RF-004 | Trenes en mapa II | La aplicación deberá mostrar información de los trenes cuando se pulse en el icono sobre el mapa. |
| RF-005 | Tren más cercano | La aplicación deberá mostrar información sobre el tren más cercano en la pantalla principal. |

| | | |
|---|---|---|
| RF-006 | Filtrado por sinóptico | La aplicación deberá ser capaz de filtrar por sinóptico y mostrar solo los trenes que pertenezcan al sinóptico seleccionado. |
| RF-007 | Ajustes de tren cercano | La aplicación deberá activar una alarma si un tren entra dentro del radio de cercanía configurable. |
| RF-008 | Ajustes de datos no actualizados | La aplicación deberá activar una alarma si no se reciben actualizaciones de las posiciones de los trenes en un tiempo configurable. |
| RF-009 | Opciones de ajustes | La aplicación deberá permitir elegir de qué forma será la alerta, sonora, visual, de vibración o por lo contrario, no mostrar alertas. |
| RF-010 | Listado de trenes | La aplicación deberá mostrar los trenes en trayecto o a punto de salir sobre una lista, ordenados por cercanía con el usuario. |
| RF-011 | Filtrado por sinóptico, listado de trenes | La aplicación deberá poder filtrar por sinóptico en la vista del listado de trenes. |
| RF-012 | Información de trenes, listado de trenes | La aplicación deberá mostrar información de cada tren en la vista del listado de trenes. |
| RF-013 | Política de privacidad | La aplicación deberá mostrar la política de privacidad que sigue la aplicación. |
| RF-014 | Atribuciones | La aplicación deberá mostrar las atribuciones. |
| RF-015 | Cerrar Sesión | La aplicación deberá ser capaz de cerrar sesión. |

Tabla 3.1: Requisitos funcionales

### 3.1.2. Requisitos no funcionales

Los requisitos no funcionales definen propiedades del sistema, es decir, son aquéllos que describen cómo se debe a hacer el sistema.

| Código | Requisito | Descripción |
|---|---|---|
| RNF-001 | Sistema Operativo | La aplicación se implementará en Android. |
| RNF-002 | Versión | La aplicación funcionará para versiones de Android 9.0 o superior. |

| | | |
|---|---|---|
| RNF-003 | Seguridad | Las comunicaciones que realice la aplicación se realizarán sobre protocolos seguros TLS[1]. |
| RNF-004 | Tiempo de respuesta II | La aplicación deberá garantizar que el tiempo de respuesta en el 95 % de los casos será inferior a 5 segundos. |
| RNF-005 | Mapa | El mapa mostrado en la aplicación deberá usar cartografía OpenStreetMaps[2]. |
| RNF-006 | Modo vertical | La aplicación solo se podrá utilizar en modo vertical. |
| RNF-007 | Intuitivo | La aplicación deberá tener una interfaz intuitiva para el usuario. |
| RNF-008 | Gandalf | La aplicación deberá implementar Gandalf[3] para el manejo de actualizaciones. |
| RNF-009 | Comunicación I | La aplicación deberá comunicarse con diferentes servicios alojados en una máquina DMZ de la empresa contratista para autenticar al usuario. |
| RNF-010 | Comunicación II | La aplicación deberá comunicarse con diferentes servicios alojados en una máquina DMZ[4] de la empresa contratista para obtener información sobre las últimas posiciones de los trenes, mediante protocolo MQTT[5] y sondeo HTTPS[6]. |
| RNF-011 | Comunicación III | La aplicación deberá comunicarse con diferentes servicios alojados en una máquina DMZ de la empresa contratista para obtener información sobre la topología de los trenes, mediante sondeo HTTPS. |
| RNF-012 | Comunicación IV | La aplicación deberá comunicarse con diferentes servicios alojados en una máquina DMZ de la empresa contratista para obtener parámetros de configuración, mediant sondeo HTTPS. |
| RNF-013 | Autenticación | La aplicación deberá autenticar con tokens las llamadas a servicios. |

Tabla 3.2: Requisitos no funcionales

[1]Protocolos criptográficos que proporcionan comunicaciones seguras por Internet
[2]Mapa editable y libre elaborado por voluntarios y publicado con una licencia de contenido abierto
[3]Librería para el manejo de actualización de versiones
[4]Zona desmilitarizada que se ubica entre la red interna de una organización y una red externa
[5]Protocolo de red de publicación-suscripción que transporte mensajes entre dispositivos
[6]Versión segura de HTTP

### 3.1.3. Requisitos de información

Los requisitos de información son aquellos que describen qué debe almacenar la aplicación.

| Código | Requisito | Descripción |
|---|---|---|
| RI-001 | Posiciones | La aplicación deberá almacenar las últimas posiciones de los trenes recibidas. |
| RI-002 | Topología | La aplicación deberá almacenar la topología. |
| RI-003 | Ajustes | La aplicación deberá almacenar los ajustes de la aplicación. |
| RI-004 | Ajustes de servidor | La aplicación deberá almacenar los parámetros de configuración obtenidos. |
| RI-005 | Token de acceso y token de refresco | La aplicación deberá almacenar los tokens de acceso y de refresco |

Tabla 3.3: Requisitos de información

# Capítulo 4

# Análisis

## 4.1. Modelo de Dominio

El modelo de dominio es un diseño conceptual en el que se describen las entidades, atributos y relaciones que albergará la aplicación. El modelo propuesto cuenta con:

- **Posiciones de tren**, reflejarán la ubicación de un tren en el instante que marca el atributo timestamp, además tendrá información extra para mostrar al usuario, como el número de tren y el material que transporta. También contiene el id de la última estación visitada, de esta forma se sabrá a qué sinópticos pertenece dicho tren.
- **Sinópticos**, cómo su nombre indica, ayudan a dividir de forma clara, rápida y en zonas geográficas las estaciones. Cada sinóptico contará con un grupo de estaciones que a su vez pueden pertenecer a más de un sinóptico. Además se verán identificados con un color.
- **Estaciones**, reflejarán todas y cada una de las estaciones donde los trenes que pertenezcan a su hoja de ruta harán sus paradas. Como atributos, cuentan con código y nombre, para ayudar a identificar al usuario de qué parada se trata. También se tendrá información de la ubicación de cada parada para que se puedan dibujar sobre el mapa.
- **Topología**, lo conforman la lista de sinópticos y estaciones que recogerá la aplicación.
- **Ajustes**, está formado por seis notificaciones, cada una podrá estar activada o desactivada. Habrá tres formas posibles de notificaciones, visual, de sonido o de vibración, y a su vez habrá dos tipos de notificación, alerta por proximidad de tren y alerta por datos no actualizados
- **Ajustes de servidor**, reflejarán ajustes generales de la aplicación, entre ellos está, la tasa de refresco con la que se hará una petición HTTPs de las posiciones y la de la toplogía, el tiempo de expiración de las posiciones, la distancia límite de proximidad a

la que un tren se tiene que acercar a la ubicación del dispotivio para que salte la notificación previamente comentada, el tiempo límite de alerta sin recibir actualizaciones de posiciones y el instante en el que los ajustes de servidor son recibidos.

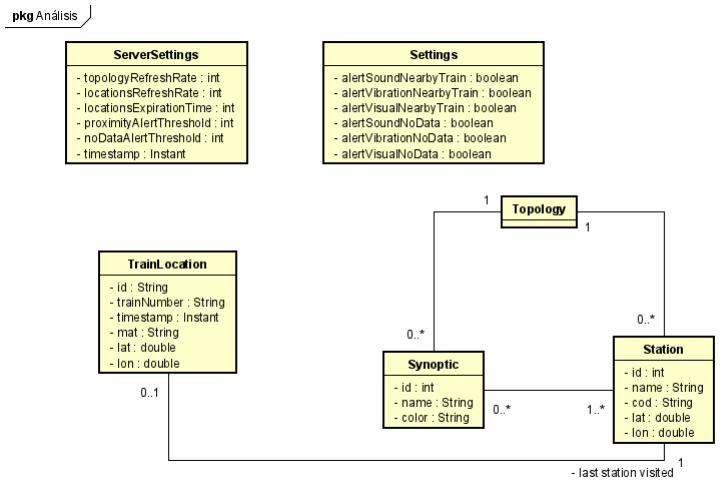



Figura 4.1: Modelo de Dominio

## 4.2. Casos de uso

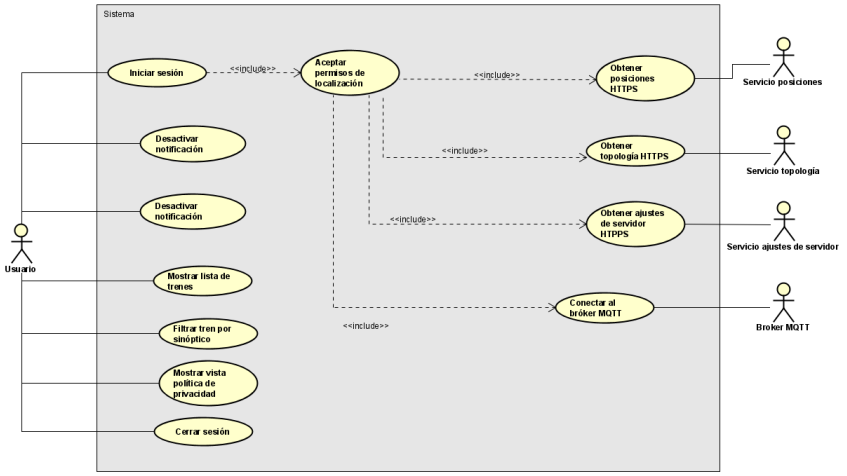



Figura 4.2: Casos de uso

### 4.2.1. Descripción casos de uso

A continuación se describirán los casos de uso mostrados en la Figura 4.2.

| Identificador | CU-01 Iniciar sesión |
|---|---|
| Actor principal | Usuario No autenticado. |
| Actor implicado | Servicio autenticación. |
| Descripción | El usuario se autentifica en el sistema a través de su usuario y contraseña. |
| Precondiciones | 1. El usuario debe estar registrado en el sistema de la empresa contratista.<br>2. El usuario debe disponer de conexión a Internet. |
| Secuencia Normal | 1. El usuario abre la aplicación.<br>2. El sistema muestra los datos que debe rellenar el usuario.<br>3. El usuario rellena los datos y envía el formulario.<br>4. El sistema manda una petición HTTPS al servicio de autenticación con los datos introducidos por el usuario.<br>5. El servicio de autenticación comprueba que los datos son de un usuario registrado.<br>6. Se ejecuta caso de uso 4.2 Aceptar permisos de localización. |
| Secuencia Alternativa | 5.1 Los datos de acceso son incorrectos, el sistema muestra un mensaje de información al usuario y el caso de uso vuelve al paso 2.<br>5.2 El servicio no está operativo, el sistema muestra un mensaje de información y el caso de uso vuelve al paso 2. |
| Postcondiciones | 1. El usuario se autentifica en la aplicación. |

Tabla 4.1: CU-01. Iniciar Sesión.

| Identificador | CU-02 Aceptar permisos de localización. |
|---|---|
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | El usuario acepta los permisos de localización. |
| Precondiciones | 1. El usuario debe estar autenticado. |
| Secuencia Normal | 1. El sistema muestra una pantalla de información con los permisos que debe aceptar el usuario.<br>2. El usuario acepta siempre los permisos de ubicación.<br>3. Se ejecutan las tareas en background. |

| | |
|---|---|
| Secuencia Alternativa | 2.1 El usuario acepta solo los permisos de localización sólo esa vez.<br>3.1 Se ejecutan las tareas en background.<br>2.2 El usuario rechaza los permisos de localización.<br>3.2 El sistema muestra un mensaje de información y termina el caso de uso. |
| Postcondiciones | 1. Los permisos de localización están aceptados.<br>2. Se muestra la pantalla de mapa con la ubicación del usuario. |

Tabla 4.2: CU-02. Aceptar permisos de localización.

En el caso de uso 4.3 nos encontramos con 6 notificaciones distintas que se tratan de igual manera:

- Alerta por proximidad de tren.
  - Notificación Sonora.
  - Notificación Visual.
  - Notificación Vibración.
- Alerta por datos no actualizados.
  - Notificación Sonora.
  - Notificación Visual.
  - Notificación Vibración.

| Identificador | CU-03 Activar notificación. |
|---|---|
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | El usuario activa la notificación de alerta. |
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona Ajustes en la pantalla de mapa.<br>2. El sistema navega hasta la pantalla de ajustes y muestra todos ellos.<br>3. El usuario activa la notificación.<br>4. El sistema guarda el estado de la notificación. |
| Secuencia Alternativa | 4.1 El sistema falla al guardar la notificación y muestra el error. |

| | |
|---|---|
| Postcondiciones | 1. La notificación seleccionada está activada. |

Tabla 4.3: CU-03. Activar notificación.

| | |
|---|---|
| Identificador | CU-04 Desactivar notificación. |
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | El usuario desactiva la notificación de alerta. |
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona Ajustes en la pantalla de mapa.<br>2. El sistema navega hasta la pantalla de ajustes y muestra todos ellos.<br>3. El usuario desactiva la notificación.<br>4. El sistema guarda el estado de la notificación. |
| Secuencia Alternativa | 4.1 El sistema falla al guardar la notificación y muestra el error. |
| Postcondiciones | 1. La notificación seleccionada está desactivada. |

Tabla 4.4: CU-04. Desactivar notificación.

| | |
|---|---|
| Identificador | CU-05 Mostrar lista de trenes. |
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | El usuario activa/desactiva la notificación de alerta. |
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona mostrar lista de trenes.<br>2. El sistema navega hasta la pantalla de lista de trenes y muestra todos los trenes. |
| Secuencia Alternativa | 2.1 No existen trenes, luego el sistema muestra información al usuario. |
| Postcondiciones | 1. El sistema se encuentra en la vista de lista de trenes. |

Tabla 4.5: CU-05. Mostrar lista de trenes.

| Identificador | CU-06 Filtrar trenes por sinóptico. |
|---|---|
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | Se aplica un filtrado en los trenes mostrados en pantalla, ya sea la pantalla de mapa o en la pantalla de lista. |
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona el botón de filtrado.<br>2. El sistema muestra una lista de sinópticos con sus con colores.<br>3. El usuario selecciona un sinóptico.<br>4. El sistema muestra los trenes que están en trayecto o a punto de salir que pertenecen al sinóptico. |
| Postcondiciones | 1. Los trenes mostrados serán aquellos de acuerdo al filtro aplicado. |

Tabla 4.6: CU-06. Filtrar trenes por sinóptico.

| Identificador | CU-07 Mostrar pantalla de política de privacidad. |
|---|---|
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | Se muestra la pantalla de política de privacidad de la aplicación. |
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona Política de privacidad en la pantalla de mapa.<br>2. El sistema muestra la política de privacidad. |
| Postcondiciones | 1. El usuario se encuentra en la pantalla de política de privacidad. |

Tabla 4.7: CU-07. Mostrar pantalla de política de privacidad.

| Identificador | CU-08 Cerrar sesión. |
|---|---|
| Actor principal | Usuario autenticado. |
| Actor implicado | |
| Descripción | Se muestra la pantalla de política de privacidad de la aplicación. |

| | |
|---|---|
| Precondiciones | 1. El usuario debe estar autenticado.<br>2. El usuario debe haber aceptado los permisos de localización. |
| Secuencia Normal | 1. El usuario selecciona cerrar sesión en la pantalla de mapa.<br>2. El sistema borra la información del usuario y navega hasta la pantalla de inicio de sesión. |
| Postcondiciones | 1. El usuario se encuentra sin autenticar en la pantalla de inicio. |

Tabla 4.8: CU-08. Cerrar sesión.

## 4.3. Tareas en *background*

La aplicación, dados los requisitos recogidos en la sección 3.1, exige que se muestren los datos actualizados, tanto de posiciones como de topología, para ello, el usuario al iniciar sesión y aceptar los permisos de localización correspondientes lanzará unas tareas en *background* para que todo esto se cumpla.

### 4.3.1. Obtener posiciones vía HTTPS

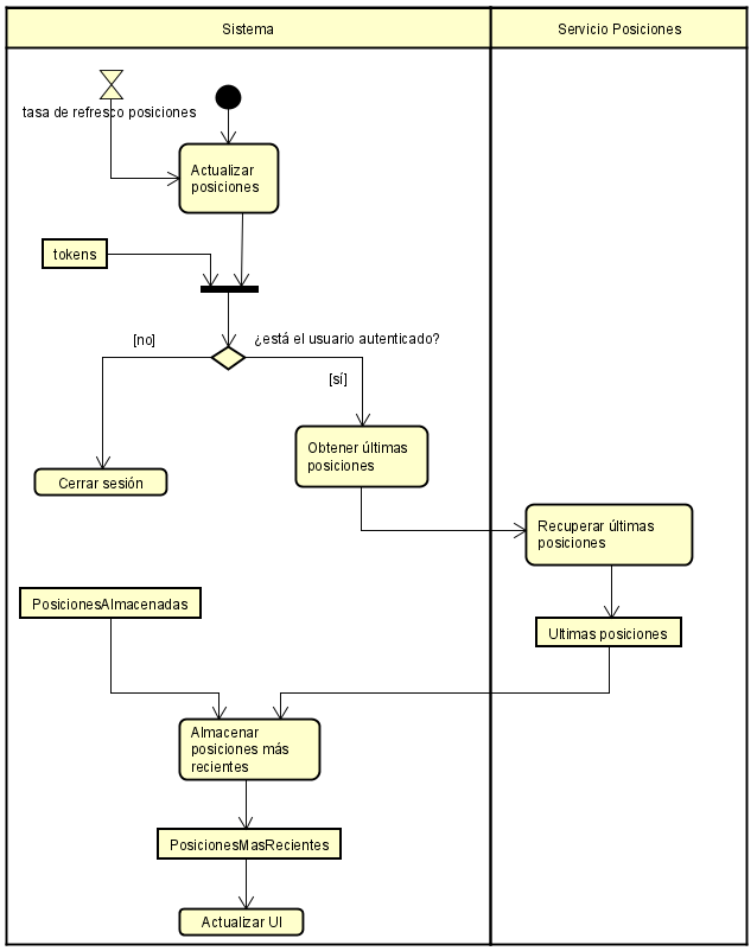



Figura 4.3: Diagrama de actividad Obtener posiciones vía HTTPS

Las posiciones de los trenes es lo más importante de la aplicación, sin posiciones actualizadas podemos decir que la app se queda inutilizable, por ello es necesario abrir dos canales de comunicación para conseguirlas. El primer canal de comunicación será vía HTTPS de manera periódica según indique el atributo tasa de refresco de los ajustes de servidor. La tarea no solo consiste en realizar la petición, si no que también tiene que comprobar que las posiciones conseguidas son más nuevas que las que se tienen guardadas en almacenamiento, por lo tanto se filtrarán aquellas obsoletas y quedarán almacenadas sólo las más recientes. Inmediatamente después se realizan dos tareas paralelas, actualizar la interfaz de manera que el usuario tenga la información de la posición lo antes posible y comprobar si es necesario lanzar una notificación y en caso afirmativo lanzarla, teniendo en cuenta los ajustes de notificaciones actuales de la aplicación.

### 4.3.2. Obtener topología vía HTTPS

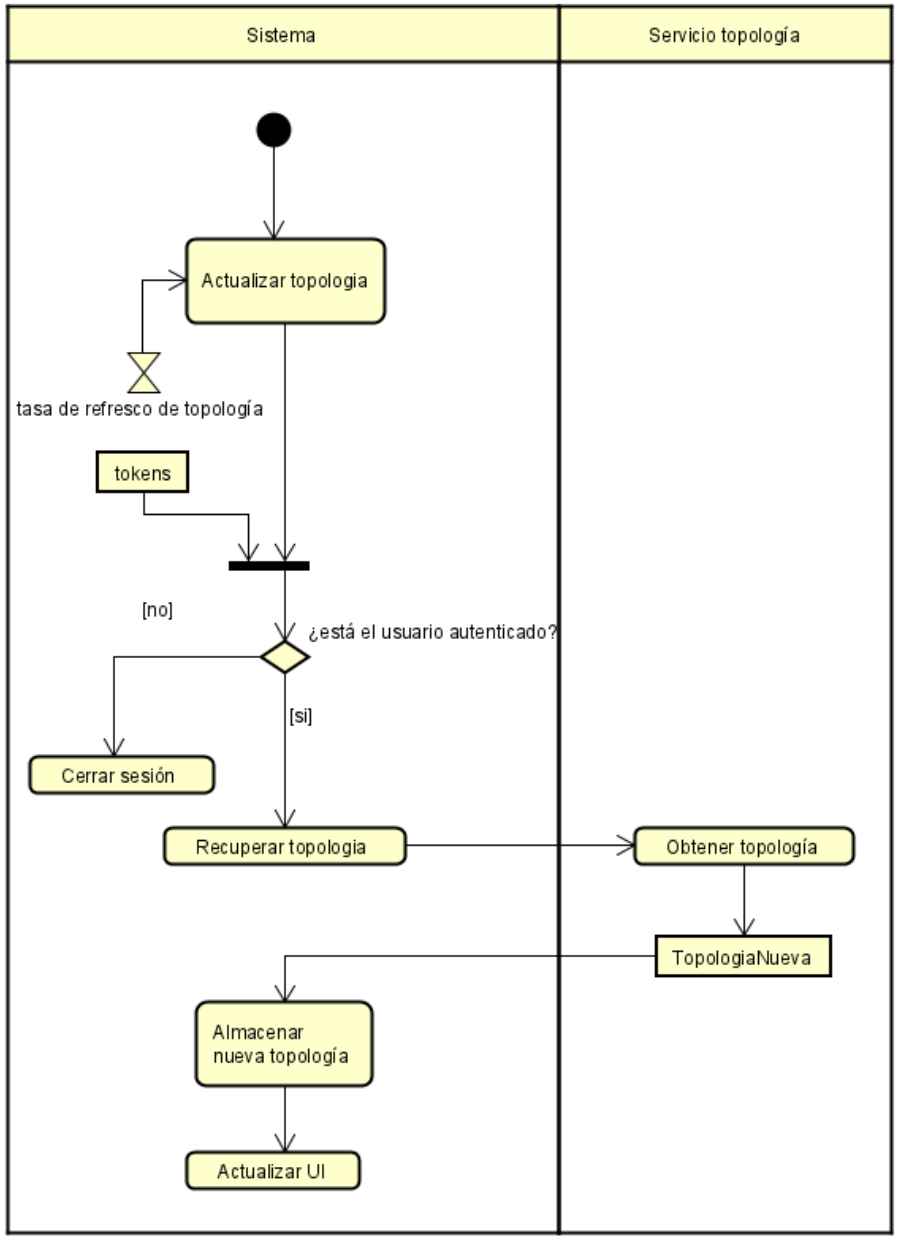



Figura 4.4: Diagrama de actividad Obtener topología vía HTTPS

La topología también es importante en la aplicación, pues sin ella la opción de filtrado carecería de funcionalidad, aunque en este caso solo habrá un canal de comunicación y será

vía HTTPS de forma periódica. La periodicidad lo marcará el atributo tasa de refresco de topología de los ajustes de servidor actuales. Una vez se posea la nueva topología, se borrará del almacenamiento la topología anterior para dar paso a la nueva, de esta forma se evitarán proba forma se evitarán problemas cuando se borre una estación, por ejemplo. Por último se actualizará el estado de la topología para que tanto las estaciones como los sinópticos más recientes aparezcan en la interfaz.

### 4.3.3. Obtener ajustes de servidor vía HTTPS

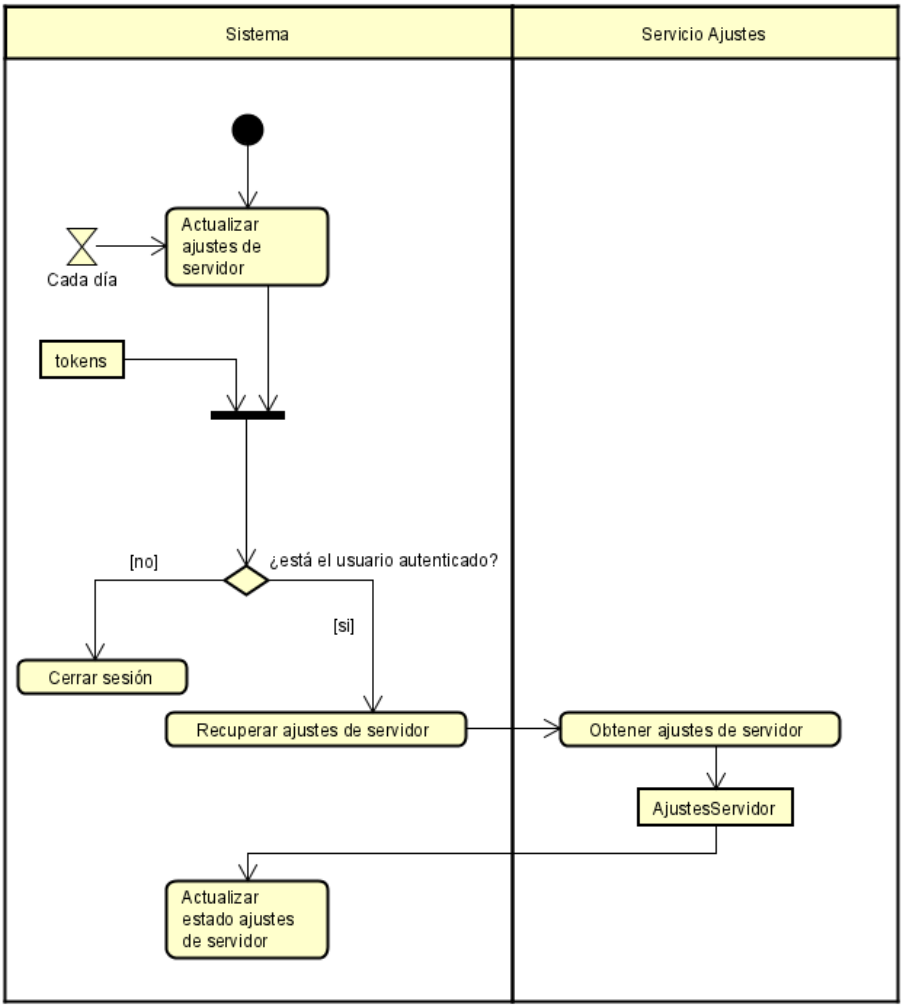



Figura 4.5: Diagrama de actividad Obtener ajustes del servidor vía HTTPS

Para los ajustes de servidor se reserva también un solo canal de comunicación para obtenerlos, vía HTTPS. Dado a que tienen menor tendencia a cambiar, se pedirán con una periodicidad de un día. Una vez recuperados los ajustes de servidor más recientes se remplazarán por los existentes en almacenamiento. Por último se actualizará el estado de los ajustes para que todas las tareas utilicen los más recientes.

### 4.3.4. Borrar posiciones expiradas

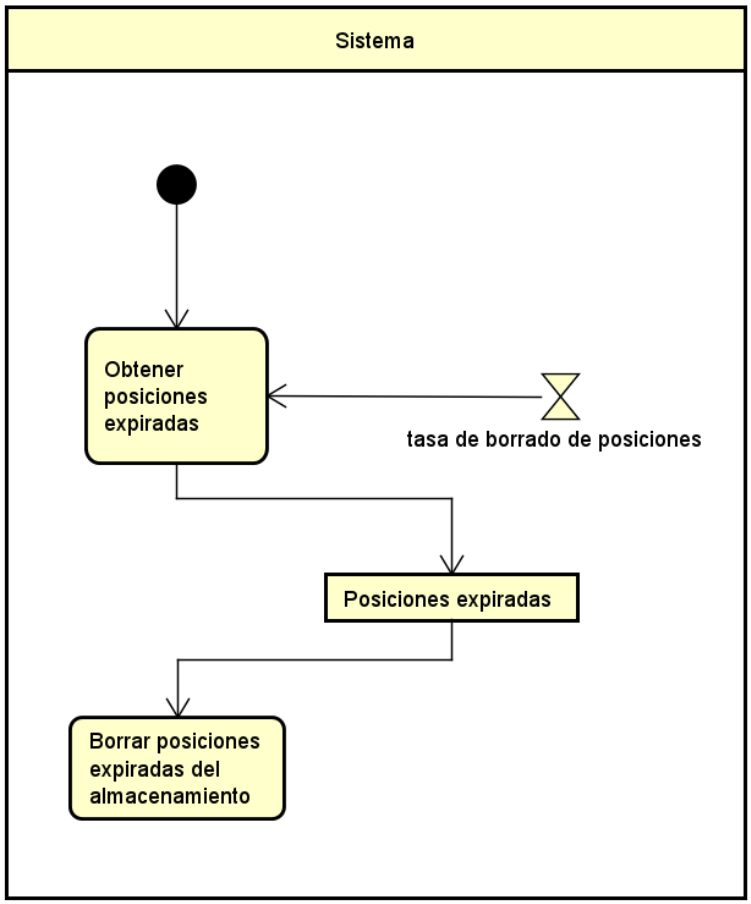



Figura 4.6: Diagrama de actividad Borrado de posiciones expiradas

Como ya sabemos, las posiciones de los trenes son identificadas por el número de tren y el instante de partida del tren. Es decir, un mismo tren, en dos trayectos diferentes, tendrá un identificador diferente. Esto significa que la lista de posiciones crecerán indefinidamente y en un futuro podría dar problemas de almacenamiento. Por ello la tarea de borrado de posiciones es tan importante.

En los ajustes de servidor se encuentra el tiempo de expiración, dicho atributo se utilizará como tasa de borrado de posiciones y como tiempo de expiración desde el instante de la llegada de la posición hasta el instante que convertirá la posición a expirada. Una vez se identifiquen todas las posiciones caducadas, se borrarán, dejando espacio libre para nuevas.

### 4.3.5. Conexión bróker MQTT

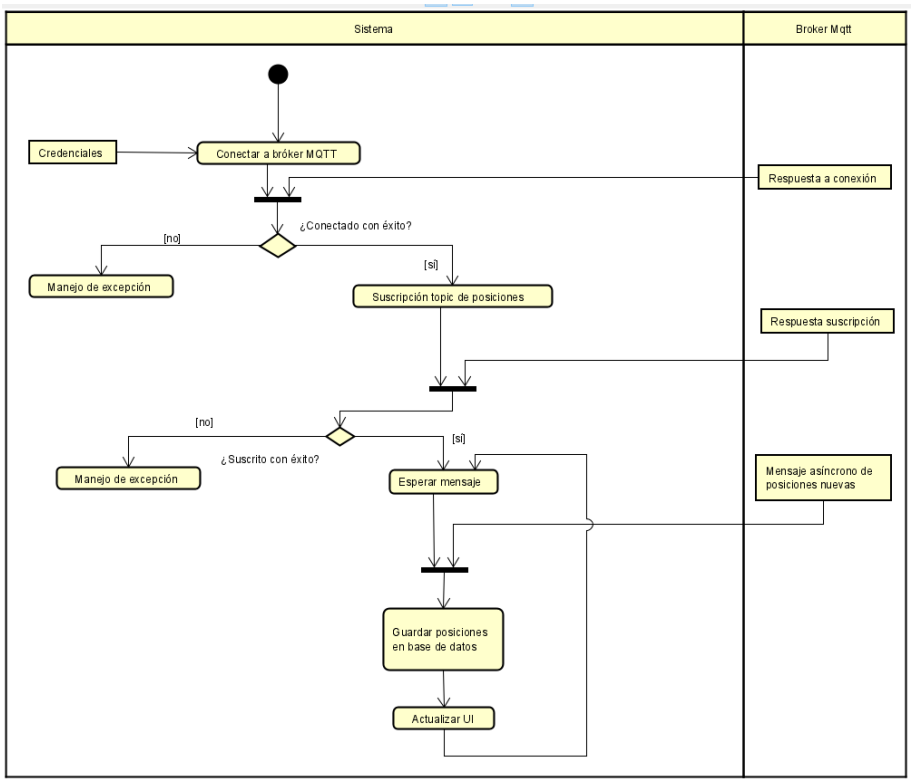



Figura 4.7: Diagrama de actividad Conectar bróker MQTT

MQTT [4] son las siglas de *Message Queuing Telemetry Transport*. Se trata de un protocolo de transporte basado en publicación/suscripción. A diferencia del protocolo tradicional cliente-servidor, en el que el cliente se comunica directamente con el servidor. El modelo de publicación/suscripción separa el cliente, que manda el mensaje (publicador), del cliente que recibe ese mensaje (suscriptor). El publicador y suscriptor nunca interaccionan directamente, de hecho no son conscientes ni siquiera de la existencia del otro. La conexión entre ellos es gestionada por un tercer componente, el *bróker*. El trabajo del *bróker* es el de filtrar todos los mensaje y distribuirlos correctamente a los suscriptores correspondientes.

MQTT es ligero, abierto, simple y diseñado para que sea fácil de implementar. Estas características lo hacen ideal para su uso en en casos de clientes que necesitan una huella de

código pequeña, que están conectados a redes no fiables o con recursos limitados en cuanto al ancho de banda. Se utiliza principalmente para comunicaciones de máquina a máquina (M2M) o conexiones del tipo de Internet de las cosas.

El protocolo se ejcuta sobre TCP/IP o sobre otros protocolos que sean ordenados, sin pérdidas, y permitan conexiones bidireccionales. Sus características incluyen:

- Uso de publicación/suscripción patrón de mensajes que proporciona distribución de mensajes de uno a varios y desacoplamiento de aplicaciones.
- Un transporte de mensajería que no conoce el contenido de la carga útil.
- Tres cualidades de servicio para la entrega de mensajes:
  - **Como mucho una vez**, los mensajes pueden perderse. Este nivel puede usarse, por ejemplo, con datos de sensores ambientales donde no importa si se pierde una lectura individual, ya que la siguiente se publicará poco después.
  - **Como mínimo una vez**, se asegura que los mensajes llegarán, pero puede ocurrir la llegada de mensajes duplicados.
  - **Exactamente una vez**, los mensajes llegarán una sola vez. Este nivel puede utilizarse, por ejemplo, con sistemas de facturación en los que los mensajes duplicados o perdidos podrían dar lugar a la aplicación de cargos incorrectos.
- Una pequeña sobrecarga de transporte e intercambios de protocolos minimizados para reducir el tráfico de red.
- Un mecanismo que notifica a las partes interesadas cuando ocurre una desconexión inesperada.

Los mensajes dentro de MQTT se publican como *topics*. Los *topics* son estructuras en una jerarquía que utilizan el carácter de barra (/) como delimitador.

Como hemos comentado anteriormente, las posiciones son lo más importante de la aplicación, por ello se definen dos canales de información. MQTT será el segundo canal.

Para que este canal funcione, es necesario primero, conectarse al *bróker* donde se publicarán los mensajes bajo el *topic* correspondiente. En el caso de que la conexión sea exitosa se deberá suscribir al *topic*, y una vez el sistema se ha suscrito con éxito, solo le queda esperar a que lleguen publicaciones.

Las nuevas posiciones se guardarán, no sin antes comprobar que son más recientes que las existentes en almacenamiento y finalmente se actualizará la interfaz de usuario.

# Capítulo 5

# Tecnologías y herramientas utilizadas

## 5.1. Entorno de desarrollo

Figura 5.1: Logo de Android Studio

Android Studio [5] es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan tu productividad cuando desarrollas apps para Android, como las siguientes:

- Sistema de compilación basado en Gradle.
- Emulador de smarthpone.
- Integración con Github.
- Consola integrada.
- Inspector de base de datos y llamadas de red.
- Analizador de APKs, que ayuda a detectar el tamaño de la app, inspeccionando el contenido.
- Editor visual de vistas.

## 5.2. Lenguaje

Figura 5.2: Logo de Kotlin

Kotlin [6] [7] es un lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript. Es desarrollado por JetBrains y algunas contribuciones de Google y otros. Entre las principales características de Kotlin se encuentran:

- **Expresivo y conciso**, centra la atención del programador en la expresión de ideas y reducir código duplicado.
- **Código más seguro**, el hecho de incluir los tipados *@Nullable* y *@NonNullable* ayuda a evitar *NullPointerExceptions*. Las aplicaciones que usan Kotlin tienen un 20 % menos de probabilidad de *crashear*.
- **Interoperabilidad**, Kotlin es 100 % interoperable con Java.
- **Concurrencia estructurada**, Las corrutinas de Kotlin agilizan la programación asíncrona, haciendo que tareas comunes como llamadas de red y actualizaciones de bases de datos sean simples y eficientes.

Figura 5.3: Logo de Jetpack Compose

Jetpack Compose [8] es el kit de herramientas moderno de Android para compilar IU nativas. Simplifica y acelera el desarrollo de la IU en Android. Entre las características de Jetpack Compose se encuentran:

- **Menos código**, realiza más tareas con menos código y evita clases internas de errores a fin de que el código sea simple y fácil de mantener.
- **Intuituvo**, a medida que cambie el estado de la app, la interfaz de usuario se actualizará automáticamente.

- **Acelera el desarrollo**, es compatible con todo el código existente para que se puede adoptar donde y cuando se quiera. Itera rápidamente con vistas previas en vivo y compatibilidad completa con Android Studio.
- **Potente**, facilita la creación de apps con acceso directo a las API de la plataforma Android y compatibilidad integrada para Material Design [9], tema oscuro [10], animaciones [11] y mucho más.

Jetpack Compose es un sistema de creación de interfaces declarativas para Android. Las interfaces declarativas son sistemas de vistas que utilizan el paradigma de programación declarativa, es decir, vamos a indicar qué queremos que se vea en la interfaz pero no le vamos a decir cómo lo tiene que hacer *under the hood*[1].

Jetpack Compose se basa en funciones que admiten composición también llamadas *composables*. Estas funciones te permiten definir la interfaz de manera programática, ya que describen su diseño. Además estas funciones se *engancharán* a un estado, y cuando ese estado cambie, el motor se encargará de actualizar ese elemento *composable* y no la vista completa. Esto facilitará mucho la tarea a la hora de mostrar datos que cambien en pantalla ya que no tendrás que preocuparte de casi nada y se hará de manera eficiente.

## 5.3. Planificación

Figura 5.4: Logo de Microsoft Project

Microsoft Project [12] (o MSP) es un software de administración de proyectos y programas de proyectos desarrollado y comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

Figura 5.5: Logo de Microsoft Excel

Microsoft Excel [13] es un software que consiste en una hoja de cálculo de Micorsoft. En este proyecto se ha utilizado para calcular el presupuesto.

[1] *under the hood literalmente es "bajo el capó"donde se encuentra el motor.*

## 5.4. Análisis y Diseño

Figura 5.6: Logo de Astah

Astah [14] es una herramienta de modelado UML creada por la compañía japonesa Change Vision. En este proyecto se ha utilizado para realizar todos los diagramas de los apartados de análisis 4 y diseño.

## 5.5. Control de versiones

Figura 5.7: Logo de Gitlab

Gitlab [15] es un servicio web de forja, control de versiones y DevOps basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una licencia de código abierto, principalmente.

Figura 5.8: Logo de SourceTree

SourceTree [16] es un software que permite visualizar y administrar repositorios a través de una sencilla interfaz de usuario que facilita al desarrollador a centrarse en el código, enseñando los cambios de cada fichero de una forma visual e intuituva.


Figura 5.9: Logo de Gitmoji

Gitmoji [17] es una guía para los mensajes de los *commit*. Pretende ser una ayuda visual estandarizada para los mensajes *commit*, de forma que se tenga una idea rápida y visual de qué se ha hecho en el *commit* sin ni siquiera leer el código.

## 5.6. Depuración

Figura 5.10: Logo de Chucker

Chucker [18] simplifica la inspección de peticiones/respuestas HTTP(S) disparadas en la app Android. Chucker trabaja como un interceptor OkHttp guardando cada evento que ocurre en la aplicación y mostrándo los detalles en una interfaz de usuario. Las apps que usan Chucker recibirán una notificación enseñando un resumen de la actividad HTTP que esté sucendiendo. Clickando en la notificación se lanzará la interfaz de Chucker.

Figura 5.11: Logo de Postman

Postman [19] es una plataforma de API para que los desarrolladores diseñen, construyan, prueben e iteren sus API. Entre sus características, destacan las siguientes:

- Creación de peticiones a APIs internas o de terceros.
- Elaboración de tests para validar el comportamiento de APIs.

- Posibilidad de crear entornos de trabajo diferentes (con variables globales y locales).

En este proyecto se ha utilizado para probar el *Back-End* realizado por el Desarrollador de Back-End mencionado en la sección de Planificación 2.3 y analizar las respuestas del lado del servidor.

## 5.7. Memoria

Figura 5.12: Logo de Overleaf

Overleaf [20] es un servicio de LaTeX colaborativo en línea, que se ha utilizado para la realización de esta memoria.

# Capítulo 6

# Diseño

## 6.1. Arquitectura

### 6.1.1. Arquitectura lógica

La arquitectura lógica define la forma en la que código fuente está estrucutrado, siguiendo patrones y abstracciones. En el caso de esta aplicación la arquitectura sigue el esquema de las 3 capas (presentación, lógica y persistencia) salvo que la capa de persistencia se ha dividido en 3 partes. Uno de los principios que se observan en la Figura 6.1 es la inversión de dependencias.

**¿Qué es la inversión de dependencias?**

La inversión de dependencias es uno de los principios SOLID (Dependency Inversion) [1] que dicta que los módulos de alto nivel no pueden depender de módulos de bajo nivel, deben depender de abstracciones, es decir, interfaces, de esta forma hacemos que los módulos de alto nivel sean independientes de la implementación.

En el caso de nuestra aplicación observamos que se aplica este principio prácticamente en todos los módulos:

- **Dominio**, los casos de uso dependerán de una interfaz, de esta forma, si existe un cambio en la implementación del repositorio, el módulo dominio quedaría intacto, o bien se decide añadir otra implementación distinta en un futuro.
- **Datos**, en este módulo también se realiza inversión de dependencias, tanto de la parte de persistencia local como remota. Esto se hace así para facilitar el cambio de implementación de ambos repositorios, por ejemplo, si queremos cambiar de base de datos local, o bien cambiar el cliente http, el módulo datos quedaría de nuevo intacto.

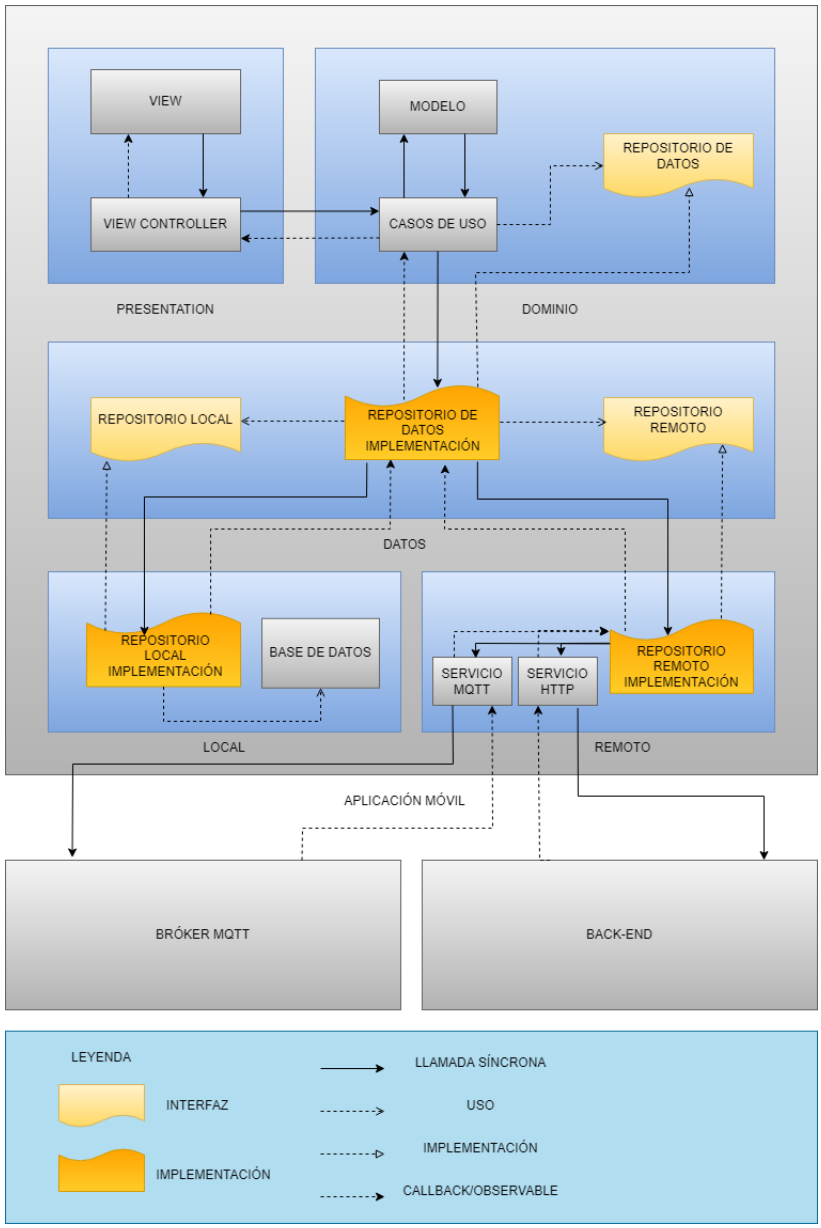



Figura 6.1: Arquitectura lógica de la aplicación.

### 6.1.2. Arquitectura física

La arquitectura física en cambio muestra los elementos físicos en los que se desarrollará todos los componentes lógicos vistos en la arquitectura lógica. Se muestra en la Figura 6.2

La aplicación móvil se desplegará en el teléfono móvil y actuará como cliente. El *smartphone* contará con dos canales de comunicación a partir de los cuales obtendrá información y se autenticará.

- ***Bróker* MQTT**, estará desplegado en un servidor de aplicaciones en una zona **DMZ**, el teléfono móvil, después de conectarse y suscribirse a los *topics* correspondientes, esperará la llegada de mensajes por parte del *bróker*
- **Back-End**, responderá a todas las peticiones realizadas por el cliente móvil, además también estará conectado con un servidor de la empresa contratista que le mandará información, en nuestro caso posiciones, de manera asíncrona. Cuando el Back-End recibe nueva información, la publicará bajo un *topic* en el *bróker*, que a su vez llegará al cliente móvil en el caso de que esté suscrito a ese mismo *topic*.

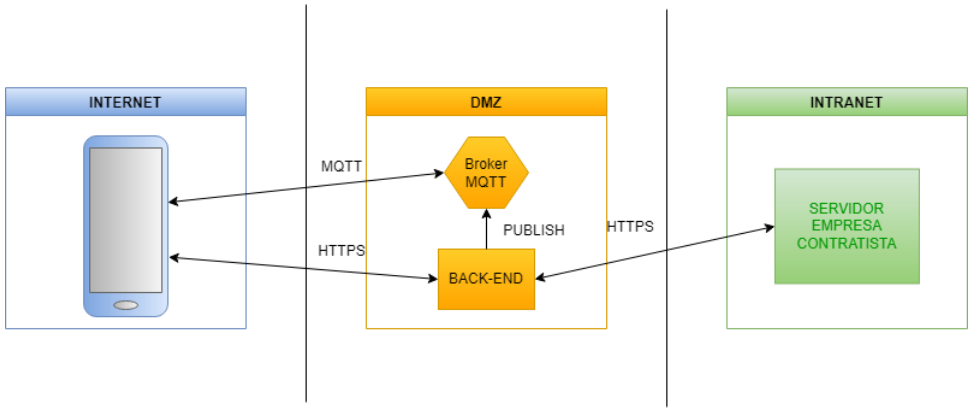



Figura 6.2: Arquitectura física de la aplicación.

## 6.2. Diseño de Base de Datos

Para almacenar los datos de la aplicación se escogerá una base de datos relacional en local. Una base de datos relacional es un tipo de base de datos que proporciona almacenamiento y acceso a datos relacionados entre sí. En una base de datos relacional, cada fila es un registro con un identificador único. Las columnas contienen los atributos. Si un registro guarda relación con un registro de otra tabla, almacenará como atributo el identificador del segundo registro.

En el caso de este proyecto, sólo habrá dos relaciones importantes:

- **Posición-Estación**: cada Posición contendrá como atributo un identificador de un registro de la tabla Estación, dicho identificador será propiedad de la última estación visitada por el tren al que pertenece dicha posición.

- **Sinóptico-Estación**: esta relación es un poco más complicada que la anterior, cada sinóptico está formado por más de una estación y cada estación puede pertenecer a más de un sinóptico, de tal forma que la manera de modelarlo es añadir una nueva tabla que almacenará en cada fila una relación sinóptico-estación.

Las tablas restantes no tienen relación con ninguna otra dado que la aplicación no guardará datos del usuario, solamente los tokens de acceso y de refresco. La idea inicial es que no se usen usuarios distintos en un mismo dispositivo, por lo tanto, los ajustes y los ajustes de servidor serán a nivel de dispositivo y no a nivel de usuario. En el supuesto de que varios usuarios inicien sesión en el mismo dispositivo, se mantendrán los ajustes configurados por el anterior usuario.

El motivo de no guardar datos de usuario es evitar guardar datos sensibles como puede ser la contraseña.

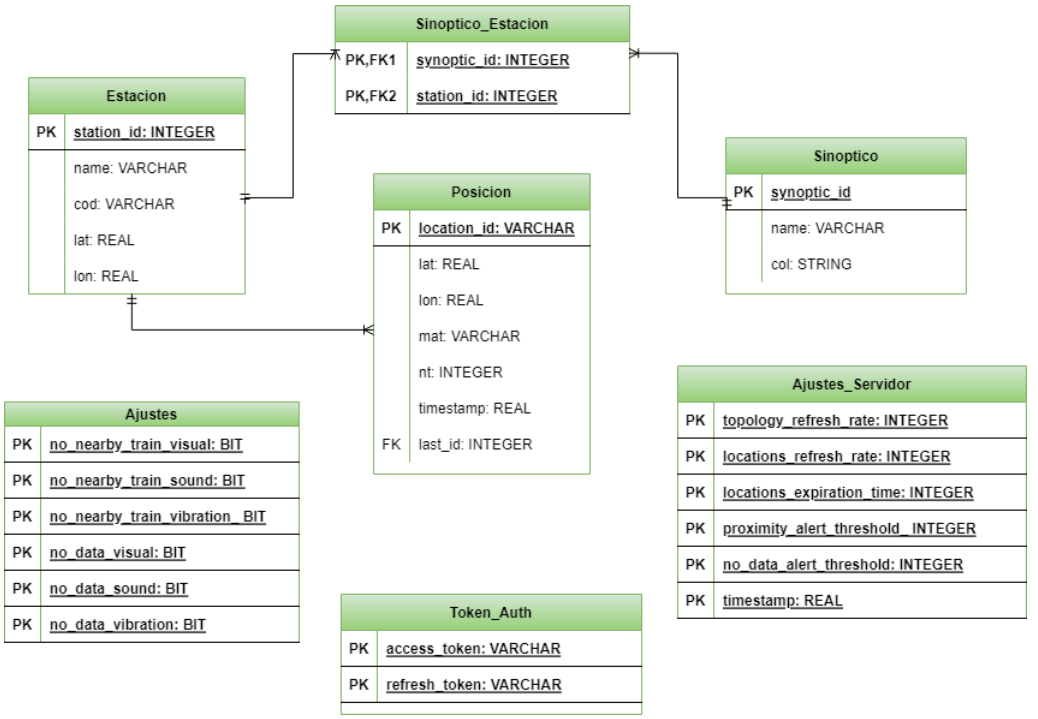



Figura 6.3: Modelo Relacional de la base de datos.

## 6.3. Diseño de interfaz

En esta sección se han realizado unos bocetos con los que se partirá de base para la futura implementación de las vistas de la aplicación. Son ideas iniciales, las vistas pueden cambiar

a lo largo del desarrollo, incluso se pueden añadir nuevas vistas. Los elementos de cada vista se describirán según estén situados de arriba a abajo de la pantalla.

### 6.3.1. Pantalla de Carga


Figura 6.4: Pantalla de carga.

Esta pantalla 6.4 se activará al iniciar la aplicación y servirá para amenizar la espera mientras se prepara todo el sistema y como bienvenida al usuario. La vista cuenta con:

- **Imagen**, Logotipo de la aplicación/empresa.
- **Círculo progresivo de carga.**
- **Texto**, indica al usuario que en breve aparecerá la siguiente pantalla.

### 6.3.2. Pantalla de Inicio de sesión

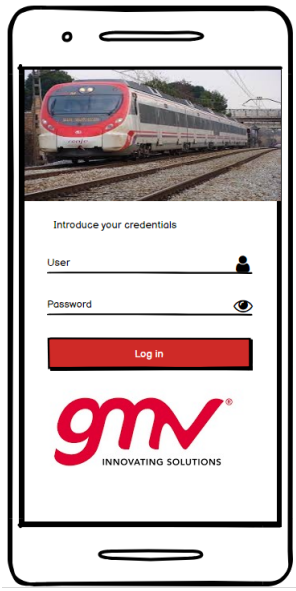
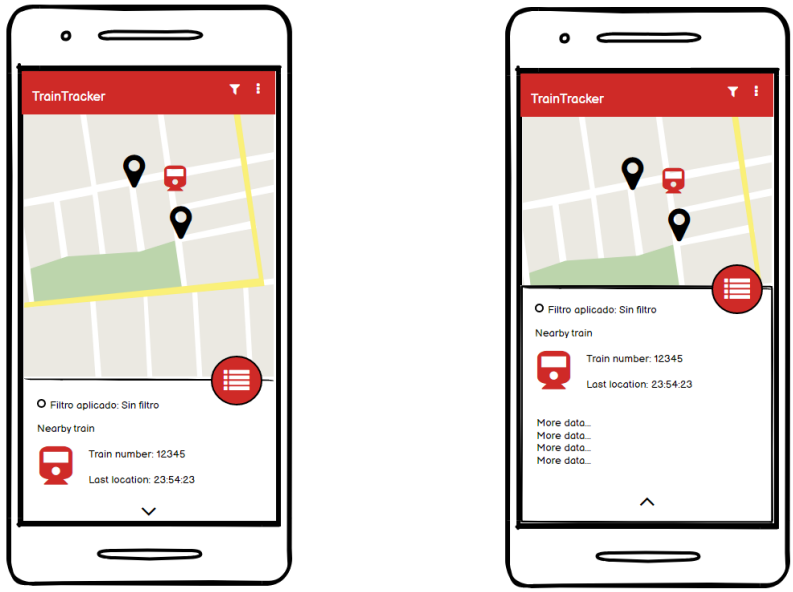



Figura 6.5: Pantalla de Inicio de Sesión.

La pantalla de inicio de sesión 6.5 se activará siempre que el usuario no esté autenticado. La vista cuenta con:

- **Imagen** de un tren en la parte superior.
- **Texto** informativo que indica al usuario que rellene con los campos con sus credenciales.
- **Campo de texto**, aquí el usuario introducirá su nombre de usuario.
- **Campo de texto**, aquí el usuario introducirá su contraseña.
- **Botón**, cuando termine de escribir sus credenciales el usuario deberá pulsar el botón para iniciar sesión.
- **Imagen**, logotipo de la aplicación/empresa.

### 6.3.3. Pantalla principal.




(a) Sin expandir (b) Expandido

Figura 6.6: Pantalla principal.

Esta pantalla se activará después de iniciar sesión y al iniciar la aplicación en el caso de que esté autenticado. La vista contará con:

- **Icono** de un filtro, que activará el caso de uso 4.6
- **Icono** de tres puntos, que desplegará un menú con varias opciones, como acceder a los ajustes, la pantalla de política de privacidad o cerrar sesión.
- **Mapa** centrado en la ubicación del usuario. Tendrá *tiles*[1] de OpenStreetMap según indica el requisito RNF-005 en la tabla 3.2
- **Botón** flotante con icono de lista que navegará al usuario hasta la pantalla de listado de trenes, es decir el caso de uso descrito en 4.5.
- **Cajón de información**, donde se mostrarán los datos más importantes de la posición más cercana a la ubicación actual del usuario, como por ejemplo el instante en el que se ha medido la posición y el número de tren. Se podrá expandir hacia arriba para mostrar más información adicional.
- **Icono** de flecha que expanderá y colapsará el cajón al ser pulsado.

[1]Un *tile* en inglés se denomina a una imagen que muestra una parte de un mapa, todos los *tiles* juntos forman el mapa completo

### 6.3.4. Pantalla lista de trenes.

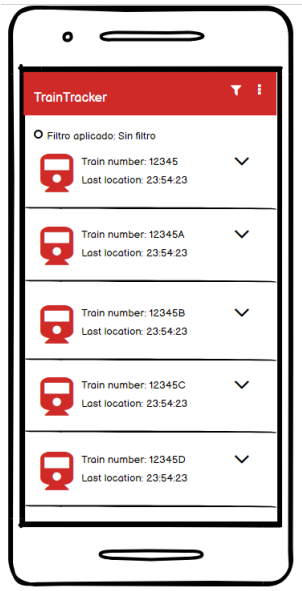



Figura 6.7: Pantalla de lista de trenes.

En esta pantalla se dispondrán en una columna los trenes ordenados por cercanía. La vista contará con:

- **Icono** de un filtro, que activará el caso de uso 4.6
- **Icono** de tres puntos, que desplegará un menú con varias opciones, como acceder a los ajustes, la pantalla de política de privacidad o cerrar sesión.
- **Circulo** que indicará el color del sinóptico seleccionado en el filtro.
- **Texto** que indicará el nombre del sinóptico seleccionado en el filtro.
- **Cajón** de información. Habrá uno por cada posición reciente y mostrará la información más importante. Se podrá expandir para mostrar información adicional.
- **Icono** de flecha que expanderá y colapsará el cajón correspondiente.

### 6.3.5. Pantalla de ajustes.

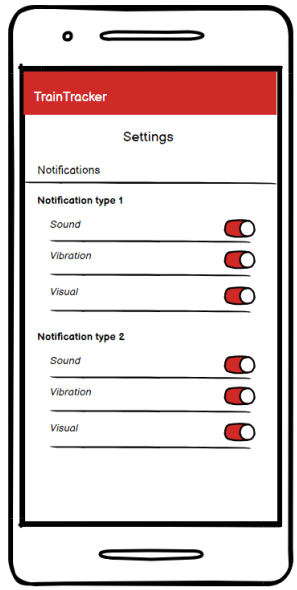



Figura 6.8: Pantalla de Ajustes.

Esta vista facilitará la gestión de notificaciones al usuario, que deberá seleccionar como activas aquéllas que desee. La vista contará con:

- **Texto** a modo de título que informa al usuario que se encuentra en la pantalla de ajustes.
- **Texto** a modo de título de sección que informa el tipo de ajuste que comienza.
- **Separador** que sirve como delimitador de secciones.
- **Texto** a modo de título de subsección que informa el subtipo de ajuste que comienza.
- **Texto** a modo de título de ajuste.
- **Icono** interruptor que el usuario deberá pulsar para activar/desactivar el ajuste.

# Capítulo 7

# Implementación

## 7.1. Formación previa

La formación previa quizás sea una de las partes más importantes de todo proyecto, y más si las tecnologías que se usarán son prácticamente nuevas para el equipo de desarrollo. Por ello se han seleccionado varios cursos gratuitos de Internet para acelerar el aprendizaje y así poder desarrollar el proyecto.

### 7.1.1. Kotlin

A partir de 2017, Kotlin pasó a ser el lenguaje oficial de Android, desbancando así a Java. Para la formación se han escogido unos cursos oficiales del equipo de Android:

- ***Codelabs*** de Conceptos básicos para desarrolladores de Android [21]. Los cursos engloban muy bien los conceptos más importantes para comenzar a programar en Android y además están en Kotlin. Comienzas creando un proyecto de cero para familiarizarte hasta conceptos como *LiveData*, Corrutinas o incluso *WorkManager*. Estos cursos no utilizan Jetpack Compose pero se explican conceptos que se podrán aplicar a cualquier proyecto Android.
- **Kotlin Koans** [22], son una serie de ejercicios que ayudan a familiarizarse con la sintaxis de Kotlin. Se pueden desarrollar en el IntelliJ IDEA o bien realizarlo online.

### 7.1.2. Jetpack Compose

Para Jetpack Compose, al ser una tecnología tan nueva no existen demasiados cursos. Por suerte el equipo de Android ha desarrollado unos *codelabs* que muestra los aspectos más importantes de esta nueva forma de desarrollar interfaces.

- ***Jetpack Compose basics***, está formado por tutoriales con ejemplos de aplicaciones reales, artículos y vídeos acerca de los nuevos conceptos que introduce Compose.

Todos los *Codelabs* incluyen un enlace a un repositorio que incluye la plantilla de la aplicación para realizar el tutorial lo más rápido posible, además también hay otro repositorio con los proyectos completos de cada curso.

## 7.2. Configuración inicial del proyecto

Antes de comenzar la implementación del proyecto, es necesario crear el proyecto y configurar los ficheros de *Gradle*. *Gradle* es una herramienta de compilación avanzada que permitirá automatizar y administrar el proceso de compilación y, al mismo tiempo, definir configuraciones de compilación personalizadas y flexibles. Algunas de las configuraciones que se pueden hacer pueden ser añadir el nombre base de la app, como será los códigos de versión, definir *buildfields* como por ejemplo *endpoints*, establecer la *keystore* para generar las futuras *APKs*, añadir las dependencias de librerías o añadir *build types*.

**¿Qué es un *build type*?**

No es más que un conjunto de configuraciones para una aplicación. Podremos aplicar unas dependencias específicas de librerías o funcionalidad para cada tipo. Por defecto existen dos fundamentales:

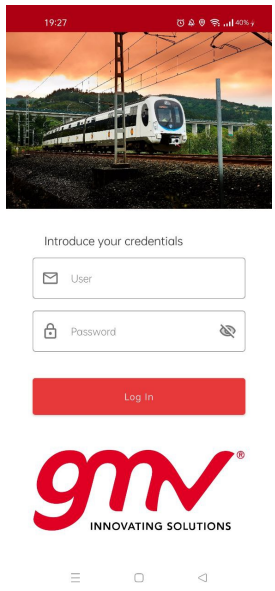
- ***Debug***: es el *build type* que se usará durante el desarrollo del proyecto, que nos permitirá *debuggear* entre otras cosas.

- ***Release***: es el *build type* que aplicaremos para generar la versión final de la aplicación, ésta sera no *debuggeable*, por defecto y se podrá añadir la opción para ofuscar el código para dificultar la tarea a alguien que consiga el código sin nuestro consentimiento. Se explicará más adelante.

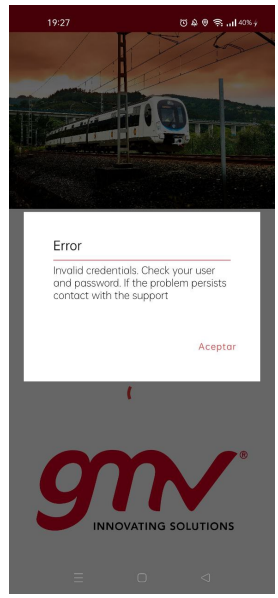
## 7.3. Diseño de interfaz con Jetpack Compose

En la siguiente sección se mostrará el aspecto final de las vistas de la aplicación, desarrollada en Jetpack Compose.

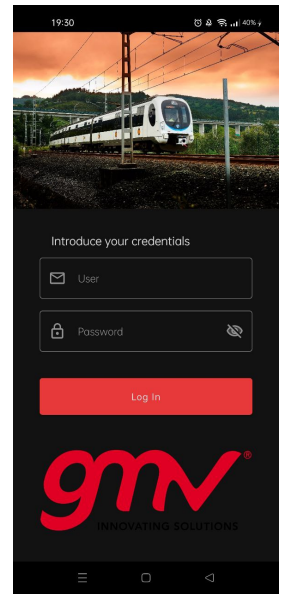
### 7.3.1. Pantalla de Inicio de sesión




(a) Modo claro




(b) Error credenciales erroneas




(c) Modo noche

Figura 7.1: Pantalla final inicio sesión.

La pantalla de inicio sesión se ha realizado con *Constrain Layout*, que es una herramienta ya existente en la creación de layouts con ficheros *XML*. Funciona de manera muy similar, ayuda a crear interfaces de forma muy flexible, dejando al usuario posicionar cada elemento donde desee, y de forma *responsive*[1].

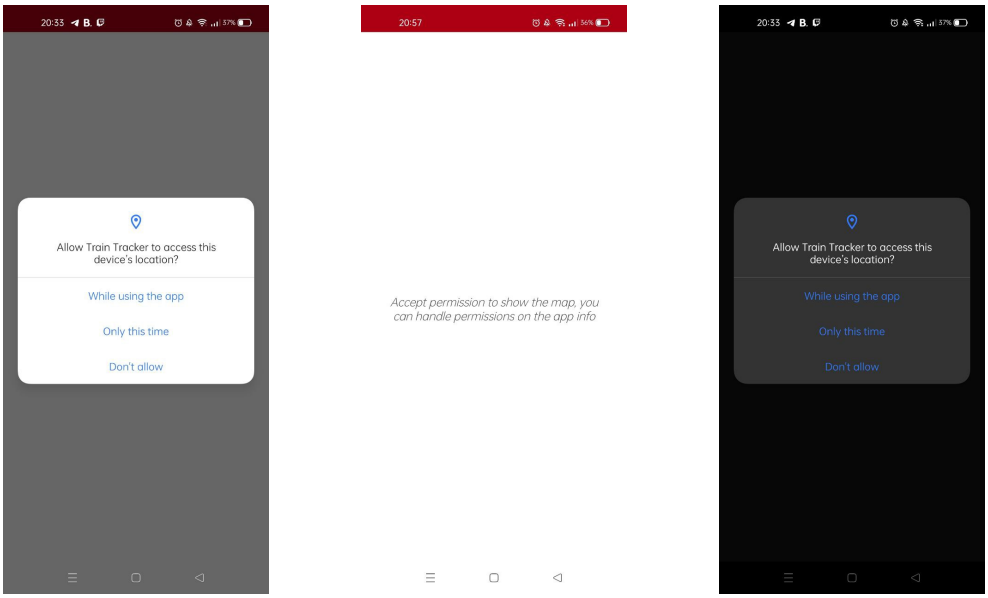
El diseño final sigue el boceto creado en la sección de diseño 6.5. Reservando el espacio central de la pantalla a los campos para rellenar el usuario y contraseña, debido a que es la parte más importante de la vista y donde el usuario debe fijarse en un primer contacto visual.

En la parte superior se ha elegido una imagen de un tren en representación al transporte del que se van a monitorizar las posiciones.

Finalmente en la parte inferior se ha introducido el logo de la empresa contratista, aunque en este caso por motivos de privacidad se ha intercambiado por el logo de GMV Innovating Solutions, empresa donde me encuentro realizando prácticas.

[1]Se adapta al tamaño de la pantalla

### 7.3.2. Pantalla de permisos




(a) Modo claro (b) Permisos denegados (c) Modo noche

Figura 7.2: Pantalla final permisos.

La aplicación deberá acceder, mientres esté activa a la posición del usuario de forma precisa Para ello es necesario aceptar los permisos de acceder a dicha información:

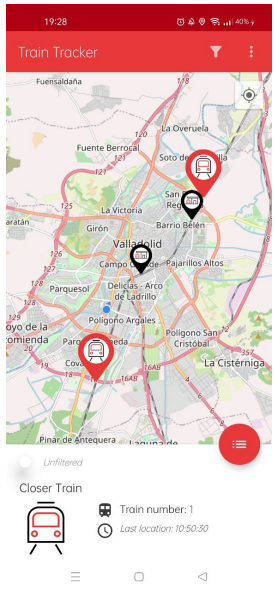
```
Manifest.permission.ACCESS_FINE_LOCATION
Manifest.permission.ACCESS_COARSE_LOCATION
```

La implementación se ha realizado con:

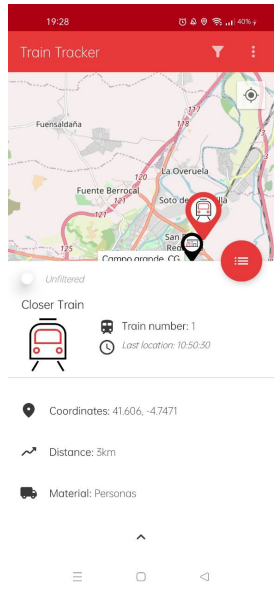
```
implementation("com.google.accompanist:accompanist-permissions:0.23.1")
```

- **Aceptar siempre**: no volverá a preguntar por los permisos a no ser que se borren los datos de la aplicación. Además la aplicación navegará hacia la pantalla de mapa.
- **Aceptar esta vez**: volverá a preguntar cada vez que se inicie la aplicación. Además la aplicación navegará hacia la pantalla de mapa.
- **Denegar**: preguntará solamente una vez más cuando se vuelva a iniciar la aplicación. La aplicación no navegará hacia la pantalla de mapa y se deberán aceptar los permisos de manera manual en la información de la aplicación. Se mostrará un texto informativo al usuario.

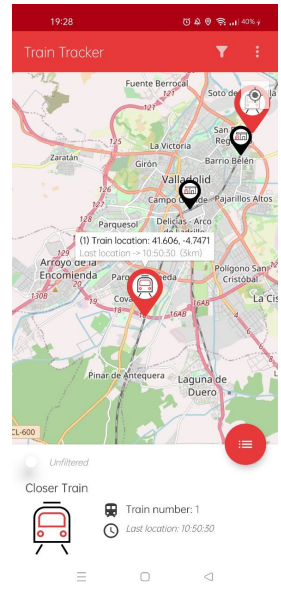
### 7.3.3. Pantalla de mapa




(a) Vista principal

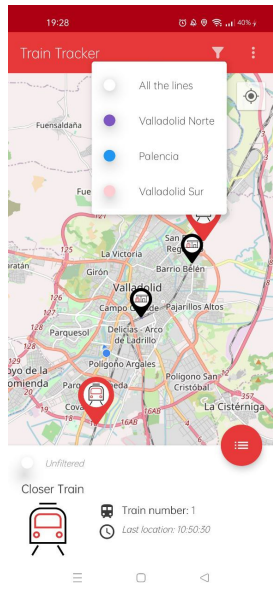



(b) Detalles del tren más cercano

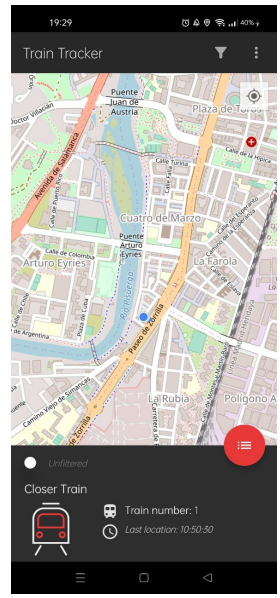



(c) Tren pulsado en mapa


(d) Estación pulsada en mapa




(e) Filtro de sinóptico desplegado




(f) Modo noche

Figura 7.3: Pantalla final mapa.

Esta vista será la pantalla principal de la aplicación, desde la cual podemos navegar hacia el resto de pantallas salvo la pantalla de aceptar permisos. Esta formada por varios compuestos pertenecientes al elemento *BottomSheetScaffold* [23], descritos a continuación de arriba a abajo:

#### Top App bar

No es más que la barra superior de la aplicación, en ella podemos diferenciar el nombre de la aplicación, Train Tracker, y dos iconos:

- **Filtro**, al ser pulsado desplegará un menú con los sinópticos existentes.
- **Tres puntos**, al ser pulsado desplegará un menú donde se encontrarán navegación hacia la pantalla de ajustes, navegación hacia la pantalla de política de privacidad y cerrar sesión.

#### Screen content

Aquí se mostrará el mapa, provisto por la API de Google Maps:

```
implementation("com.google.maps.android:maps-compose:1.2.0")
```

El mapa formará parte de un único *composable* y se pintarán sobre él, las últimas posiciones recibidas de cada tren y las estaciones.

Para mostrar información que vaya cambiando a lo largo del tiempo, como por ejemplo en nuetro caso, las posiciones de los trenes, se creará un estado (*State*) de posiciones el cual estará formado por una lista de posiciones. El estado guardará la información más reciente que se haya obtenido acerca de las posiciones, es decir si llegan nuevas posiciones, se debe actualizar dicho estado. El estado estará *enganchado* al *composable* correspondiente de tal forma que si cambia el estado, se recompondrá la vista descrita por ese *composable* con el nuevo estado, con las nuevas posiciones.

El mapa estará formado por cartografía de *OpenStreetMaps* [24] y además se contará con un botón para centrar el mapa en la ubicación del usuario en la parte superior derecha.

#### Floating Action Button

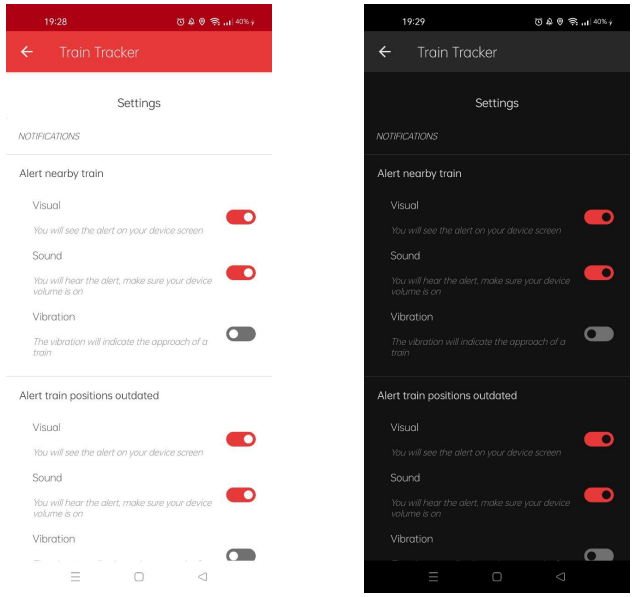
Será un botón con un icono de lista que al ser pulsado navegará hasta la pantalla de listado de trenes. Estará anclado a la hoja inferior.

**Sheet Content**

Coexistirá con la region principal de la interfaz de usuario. El usuario podrá interactuar con ambas regiones de manera simultánea. La utilidad de las hojas inferiores es mostrar información secundaria arrastrada dentro del contenido principal.

La hoja sin expandir mostrará la hora de llegada de la posición y el número de tren, al arrastrar hacia arriba la hoja, se podrán ver las coordenadas exactas en cuatro decimales, la distancia en metros si es menor que un kilómetro o en kilómetros en caso contrario y el material que transporta el tren al que pertenece la posición.

### 7.3.4. Pantalla de ajustes



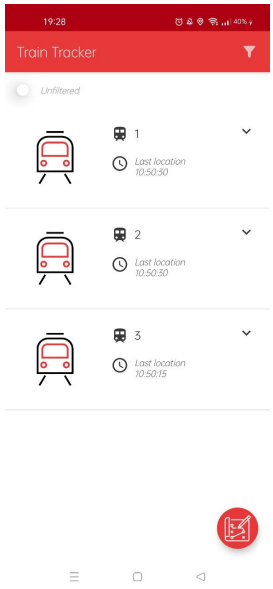

(a) Modo claro (b) Modo noche

Figura 7.4: Pantalla final ajustes.

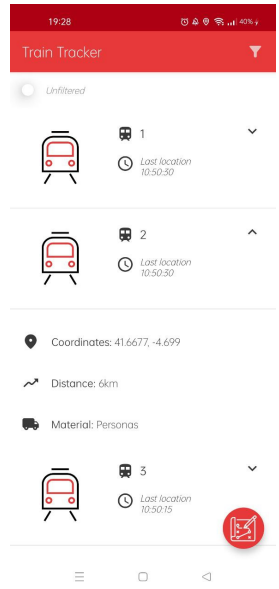
La pantalla de ajustes sigue el modelo habitual de ajustes de cualquier dispositivo, un texto descriptivo en la parte izquierda y un botón interruptor *switch* para activar o desactivar el ajuste. Se ha usado *Constraint Layout* para diseñar la vista. Además contará con una flecha situada en la barra superior que permitirá volver a la pantalla principal.

Al pulsar un interruptor se almacenará en la base de datos el nuevo resultado de forma que al volver a entrar se haya guardado el estado del interruptor.

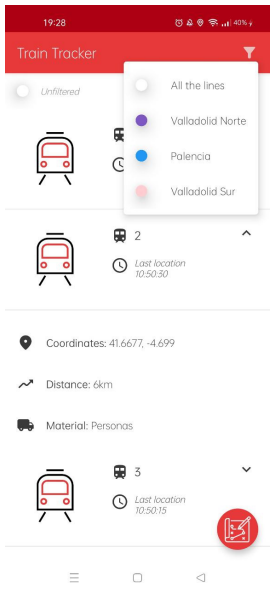
### 7.3.5. Pantalla de lista de trenes



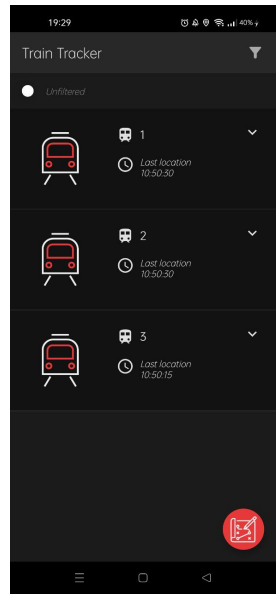

(a) Modo claro




(b) Detalles del tren expandidos




(c) Menú filtro sinóptico desplegado




(d) Modo noche

Figura 7.5: Pantalla final lista de trenes.

La pantalla de lista de tren seguirá la composición del elemento *Scaffold*, que implementa la estructura básica según los principios de *Material Desing* [25]. Los elementos que componen la vista son:

#### Top App Bar

La barra superior será igual que la vista en la pantalla principal salvo el icono de tres puntos. El filtrado funcionará de la misma forma que en la pantalla principal.

#### Screen Content

Se trata de la región principal de la vista, en este caso estará formado por una *LazyColumn*, que es el equivalente al conocido *ReciclerView*. Este *composable* permitirá mostrar en forma de columna solamente los *items* que quepan en la pantalla, de esta forma aunque la lista de *items* sea muy larga el compilador no perderá tiempo en componer la lista completa.

Cada *item* estará diseñado con *ConstraintLayout* y se podrá desplegar pulsando el botón en la parte superior derecha del *item*. Los detalles que se mostrarán de cada posición serán los mismo mostrados en la pantalla principal en la hoja inferior.

#### Floating Action Button

Será un botón con un icono de mapa que al ser pulsado navegará hasta la pantalla principal.

### 7.3.6. Pantalla de política de privacidad


(a) Modo claro (b) Modo noche

Figura 7.6: Pantalla final política de privacidad.

La política de privacidad se encuentra en la carpeta *assets* del proyecto en un fichero *Markdown* [2]. Primero se debe leer el contenido del fichero y pasarlo a una variable de tipo *String*, para mostrarlo en pantalla, se ha utilizado:

```
1 implementation("com.halilibo.compose-richtext-commonmark:0.11.0")
```

Esta librería añadirá varios *composables*, uno de ellos aceptará la cadena leída con anterioridad y lo convertirá según el formato *Markdown* que siga. También contiene funciones para formatear al gusto el estilo del texto.

## 7.4. Colores de la aplicación

El color principal de la aplicación será el rojo y admitirá tanto modo claro o modo noche, según tenga el usuario configurado su dispositivo.

Los colores será los siguientes:

[2]Lenguaje de marcado que facilita la aplicación de formato a un texto plano

**Modo claro**

| | | |
|---|---|---|
| *Background* | Blanco - 0xFFFFFF | |
| *Sobre background* | Gris oscuro - 0x1B1B1B | |
| Color primario | Rojo - 0xE8393A | |
| Sobre primario | Gris claro - 0xEEEEEE | |
| Color secundario | Rojo oscuro - 0xAE0013 | |

Tabla 7.1: Modo claro.

**Modo oscuro**

| | | |
|---|---|---|
| *Background* | Gris oscuro - 0x1B1B1B | |
| *Sobre background* | Gris claro - 0xEEEEEE | |
| Color primario | Rojo - 0xE8393A | |
| Sobre primario | Gris claro - 0xEEEEEE | |
| Color secundario | Negro - 0x000000 | |

Tabla 7.2: Modo oscuro.

## 7.5. Idiomas

La accesibilidad es un punto muy importante a la hora de desarrollar aplicaciones, por eso se ha incluido unos fichero de recursos de *Strings* separados por idiomas.

Por el momento esta aplicación admitirá el español y el inglés.

## 7.6. Inyección de dependencias

Las dependencias entre clases se pueden representar como un diagrama, como muestra la Figura 6.1, por ejemplo, un repositorio del módulo local dependerá de la base de datos, eso quiere decir que al instanciarse el repositorio necesitará la instancia de la base de datos. Hacer todas estas dependencias de forma manual, puede llegar a causar repetición de código, errores y mala legibilidad en el código.

Para solucionar este problema, existe la inyección de dependencias, en este proyecto se ha utilizado *Dagger-Hilt*:

```
plugins {
  id("kotlin-kapt")
  id("dagger.hilt.android.plugin")
}
dependencies {
    implementation("com.google.dagger:hilt-android:2.38.1")
    kapt("com.google.dagger:hilt-compiler:2.38.1")
}
```

Hilt nos ayudará a inyectar todas las dependencias, solamente indicando su instanciación. Se deberá añadir la anotación *@Provides* en la parte superior del nombre de la función.

El ejemplo anterior de un repositorio del módulo local, su función *Hilt* que se usará para inyectar la instancia en las clases que dependan del repositorio sería así:

```
    @Provides
    @Singleton
    fun provideLocalLocationRepository(
        db: RoomDb,
        logger: Logger
    ): LocalLocationsDataStore{
        return LocationsRepositoryImpl(db.locationsDao, logger)
    }
```

Como se puede observar, el repositorio a su vez depende de *Logger* y *RoomDb*, para que *Hilt* pueda inyectar el repositorio necesitará saber también cómo se instancian.

## 7.7. Autenticación

La sesión se gestionará siguiendo el estándar Oauth, de esta manera no será necesario que la app guarde las credenciales del usuario. El usuario al iniciar sesión, se emitirá un *Post* con sus credenciales, que devolverá en caso de que sean correctas, dos datos: *token* de acceso y *token* de refresco.

#### ¿Qué son y cómo funcionan estos *tokens*?

Un *token* es una cadena de caracteres generada automáticamente que guarda información encriptada, como por ejemplo, en este caso, el tiempo de expiración.

El *token* de acceso servirá como llave para seguir en la aplicación autenticado, y recibir información del exterior. Cuando ese tiempo de expiración caduque pueden pasar dos cosas:

- El usuario mantiene la aplicación encendida o que el usuario haya cerrado la aplicación, el móvil estaría continuamente comunicándose con el servidor en busca de nuevas posiciones, y si el servidor observa que el *token* de acceso ha caducado, mirará ahora sí, el

*token* de refresco, que siempre tendrá un tiempo de caducidad mayor, si éste último no está caducado se actualizarán ambos tokens con nuevos tiempos de expiración. De esta forma aseguramos que mientras el usuario use la aplicación no tenga que introducir los credenciales de nuevo.

- El usuario cierre la aplicación durante un tiempo mayor al tiempo de caducidad de ambos *tokens*, se necesitarán de nuevo las credenciales para acceder a información.

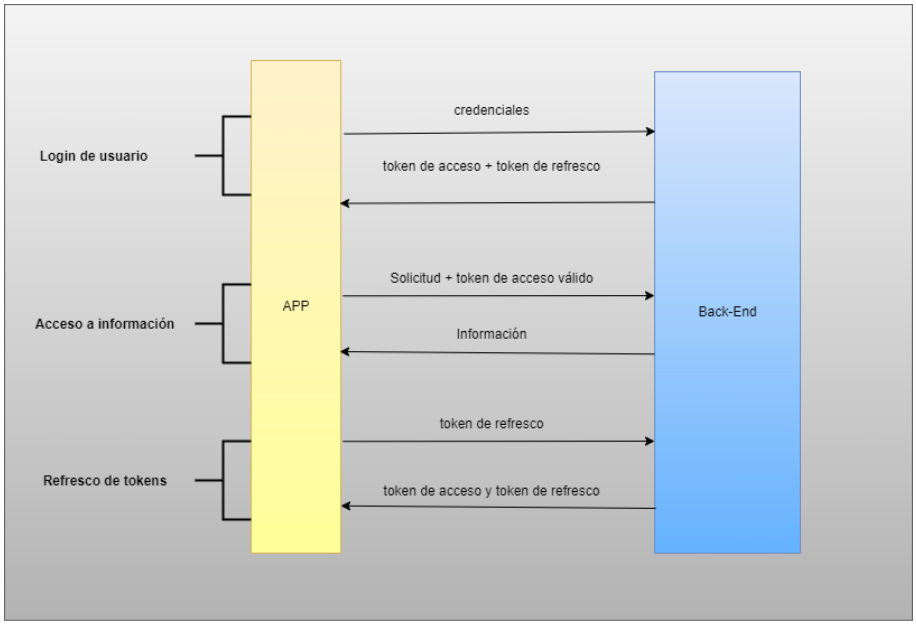



Figura 7.7: Diagrama de autenticación de la app.

## 7.8. Servicios HTTPS

Para que la aplicación disponga de la información más actualizada posible se realizarán peticiones HTTPS de manera periódica a un servidor preparado para que devuelva los últimos datos que haya recibido.

Para la implementación del cliente HTTP se ha utilizado una librería del *framework* desarrollado por JetBrains, Ktor [26]. La información obtenida al realizar las peticiones vendrá codificada en formato *JSON* lo que hace muy sencillo serializarlo a objetos. Existen varias librerías que cumplen esta función, como *GSON*, *Jackson* o *kotlinx*, para el desarrollo de esta aplicación se ha escogido ésta última. Serán necesarias las siguientes dependencias.

```
implementation("io.ktor:ktor-client-core:1.6.7")
implementation("io.ktor:ktor-client-serialization:1.6.7")
implementation("io.ktor:ktor-client-logging:1.6.7")
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.3.2")
implementation("io.ktor:ktor-client-okhttp:1.6.7")
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.3.2")
```

### 7.8.1. Servicio de posiciones

Las posiciones se obtendrán de manera periódica según indique la tasa de refresco de las posiciones en los ajustes de servidor. Las posiciones llegarán en el formato del siguiente *Json*:

```
[
    {
        "id": "20220322124155-3001",
        "nt": "3001",
        "timeStamp": "20220505124155",
        "mat": "30",
        "lat": 50.5,
        "lon": 20.2,
        "lastId": 5
    }
]
```

Luego para recibir la información, y la serialización lo reconozca como un objeto, se creará el *data class RemoteLocation*, que deberá tener la anotación *@Serializable* al comienzo:

```
@Serializable
data class RemoteLocation(
    val id: String,
    val nt: String,
    val timeStamp: String,
    val mat: String,
    val lat: Double,
    val lon: Double,
    val lastId: Int
)
```

Finalmente se creará el cliente que hará la petición, se deberá añadir el contenido de la respuesta que se espera obtener, en este caso *Json*.

```
val response: HttpResponse = client.get() {
    setUrl(host, LOCATIONS_PATH)
    contentType(ContentType.Application.Json)
}
 response.receive<List<RemoteLocation>>()
```

### 7.8.2. Servicio topología

La topología por otro lado no cambiará tanto de un instante a otro, por lo que la tasa de refresco siempre será mucho mayor a la tasa de las posiciones. La topología que devolverá el servidor tendrá el formato del siguiente *Json* de ejemplo.

```
{
    "estaciones":
    [
        {
            "id": 1,
            "nombre": "Campo Grande",
            "cod": "CG",
            "lat": 50.5,
            "lon": 20.2,
        }
    ]
    "sinopticos":
    [
        {
            "id": 1,
            "nombre": "Valladolid Norte",
            "col": "0xFFFFFF",
            "estaciones":   [
                1,
                2,
                3,
                4
               ]
        }
    ]
}
```

Como se puede observar la estructura es un poco más compleja, para simplificar el desarrollo, se crearán tres *data classes* que será las encargadas de organizar la información recibida.

```
@Serializable
data class RemoteSynoptic(
    val id: Int,
    @SerialName("nombre")
    val name: String,
    val col: String,
    @SerialName("estaciones")
    val stations: List<Int>
)
```

```kotlin
@Serializable
data class RemoteStation(
    val id: Int,
    @SerialName("nombre")
    val name: String,
    val cod: String,
    val lat: Double,
    val lon: Double,
)
```

```kotlin
@Serializable
data class RemoteTopology(
    val estaciones: List<RemoteStation>,
    val sinopticos: List<RemoteSynoptic>
)
```

La petición se realizará de la misma forma que las posiciones salvo por el tipo esperado, que en este caso será *RemoteTopology*.

### 7.8.3. Servicio ajustes de servidor

Igual que la topología, los ajustes de servidor, no cambiarán con tanta facilidad, se actualizarán como mínimo una vez al día. Los ajustes llegarán con el formato del siguiente *Json* de ejemplo.

```json
[
    {
        "topologyRefreshRate": 86400,
        "locationsRefreshRate": 30000,
        "locationsExpirationTime": 30000,
        "proximityAlertThreshold": 5000,
        "noDataAlertThreshold": 120000
    }
]
```

Para almacenar los datos recibidos del servidor se creará la siguiente *data class*

```kotlin
@Serializable
data class GetSettingsResponse(
    val topologyRefreshRate: Int,
    val locationsRefreshRate: Int,
    val locationsExpirationTime: Int,
    val proximityAlertThreshold: Int,
    val noDataAlertThreshold: Int
)
```

La petición HTTPS se realizará de igual forma que las posiciones y la topología salvo por el objeto que espera la respuesta, que en este caso será, *GetSettingsResponse*.

## 7.9. Obtención de posiciones vía MQTT

El objetivo de esta parte es añadir un nuevo canal de comunicación de obtención de posiciones. El cliente, en este caso el dispositivo móvil, se conectará al *bróker* y se sucribirá al *topic* "*/locations*", donde el servidor *Back-End* se encargará de publicar la lista de posiciones en formato *Json* cuando reciba actualizaciones. La implementación se ha realizado con la librería:

```
1 implementation("org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.5")
```

Para el uso se ha creado una clase que actúa de *manager*, que será capaz de conectar con un *bróker*, suscribirse a un *topic*, etc. Al instanciarse este *manager* se creará un objeto *Callback* que ejecutará una función en el momento de llegada de un mensaje a un cualquier *topic* del que se está suscrito. A la llegada del mensaje se deberá deserializar el mensaje a una lista de *RemoteLocation*, esta vez con la librería *GSON*:

```
1 implementation("com.google.code.gson:gson:2.9.0")
```

La lista resultado se emitirá en un *Flow*, la emisión activará una parte del código que actualizará el estado de posiciones en la pantalla.

Previamente las posiciones recibidas, antes de guardarse en base de datos, se comprobará que son más recientes que las que existen en la base de datos, es decir se comprobará si el *timestamp*, milisegundos desde 1 de enero de 1970, de la *nueva* es mayor que el *timestamp* de la cacheada.

La aplicación, inmediatamente después de que el usuario inicie sesión habiendo aceptado los permisos de localización se conectará y suscribirá. Cuando el usuario cierra sesión se desuscribirá y desconectará del *bróker*.

## 7.10. Base de datos local

Todos los datos de la aplicación deben almacenarse en algún lugar, para que, al volver a iniciar la aplicación, no se pierdan. Como se indicó en diseño se eligió una base de datos relacional, y la mejor opción en Android es *Room*, una abstracción para *SQLite* que permite acceder a la base de datos sin problemas y, al mismo tiempo, aprovechar toda la potencia de *SQLite*:

```
1 implementation("androidx.room:room-runtime:2.4.2")
2 implementation("androidx.room:room-ktx:2.4.2")
```

Para ello se prepararán unos *data classes* con la anotación de *@Entity* al comienzo, en ella añadiremos el nombre que tendrá la tabla que registrará en cada fila una *entity* con ese esquema de datos.

Para abreviar, se explicarán solamente las *entities* más complejas, la relación entre estación y sinóptico y aquellas que guardan un *timestamp*.

### Relación estación-sinóptico

Como ya se ha comentado previamente, cada sinóptico tendrá varias estaciones asociadas, mientras que cada estación puede pertenecer a distintos sinópticos. Para cumplir con esta relación es necesario añadir una tabla extra que guardará cada relación en un registro:

```
@Entity(tableName = "stations")
data class StationEntity(
    @PrimaryKey(autoGenerate = false)
    val idStation: Int,
    val name: String,
    val cod: String,
    val lat: Double,
    val lon: Double
)
```

```
@Entity(tableName = "synoptics")
data class SynopticEntity(
    @PrimaryKey
    val idSynoptic: Int,
    val name: String,
    val col: String
)
)
```

En la nueva tabla añadida, la clave primaria será la combinación de los dos *ids*, es decir, la relación.

```
@Entity(primaryKeys = ["idStation","idSynoptic"], tableName = "synoptics_stations")
data class SynopticStationCrossRef(
    val idStation: Int,
    val idSynoptic: Int
)
```

Para realizar peticiones a la base de datos, se debe añadir una interfaz con la anotación @Dao en la parte superior. En ella se encontrarán todos los métodos de consulta que se realizarán:

```kotlin
@Dao
interface TopologyDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    abstract suspend fun insertStations(stations: List<StationEntity>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    abstract suspend fun insertSynoptics(stations: List<SynopticEntity>)

    @Insert
    abstract suspend fun insertSynopticStationCrossRefs(crossRefs:
    ↪  List<SynopticStationCrossRef>)

    @Query("DELETE FROM synoptics_stations")
    abstract suspend fun deleteAllRelationsSynopticStation()

    @Query("DELETE FROM synoptics")
    abstract suspend fun deleteAllSynoptics()

    @Query("DELETE FROM stations")
    abstract suspend fun deleteAllStations()

    @Query("SELECT * FROM stations")
    abstract suspend fun getStations(): List<StationEntity>

    @Query("SELECT * FROM synoptics")
    abstract suspend fun getSynoptics(): List<SynopticEntity>

    @Query("SELECT * FROM synoptics_stations")
    abstract suspend fun getSynopticStationCrossRef(): List<SynopticStationCrossRef>
}
```

### *Type Converter*

Como es el caso de las posiciones, o ajustes de servidor, incorporan un timestamp. Ese timestamp se manejará como un *Instant* [27]. La base de datos no admite valores de tipo *Instant* por ello se ha decidido crear un *Type Converter*, para añadir una capa más de abstracción. La base de datos almacenará y devolverá un valor *Long*, para que sepa como convertir ese valor *Instant* en un valor aceptado se ha creado una clase con dos funciones que actuarán de traductor entre el valor devuelto y el modelo de datos.

```kotlin
class DateTypeConverter {
    @TypeConverter
    fun fromLongToInstant(value: Long?): Instant? {
        return value?.let { Instant.ofEpochMilli(it) }
    }

    @TypeConverter
    fun instantToLong(value: Instant?): Long?{
        return value?.toEpochMilli()
    }
}
```

## 7.11. Uso de *Mappers*

Para cada módulo será necesario disponer de los datos de una forma, es decir lo que se guardará en base de datos no será lo mismo que se muestre por pantalla. Para que se entienda mejor, vamos a centrarnos en el tiempo de llegada de una posición.

La posición llegará como un Json en el que el *timestamp* será una cadena de caracteres siguiendo el siguiente formato: YYYYMMDDHHmmSS, en pantalla se mostrará solamente la hora, minutos y segundos, separados por dos puntos cada número, y en la base de datos se guardará como un número entero que simboliza los milisegundos desde el 1 de Enero de 1970. El paso de estos objetos a cada módulo tendrá que pasar un filtro para que cuando llegue, tenga el formato deseado.

## 7.12. Uso de Either. Manejo de excepciones

El manejo de excepciones es una tarea obligatoria en cada proyecto informático. Ningún código está libre de errores y es mejor tenerlos controlados. Por ello para este proyecto se ha escogido el uso de *Either*, que no es más que un tipo de dato que puede almacenar a su vez dos tipos, es decir, puedes almacenar o una excepción o el tipo que esperas obtener.

Lo proporciona las librerías:

```
implementation("io.arrow-kt:arrow-core:1.0.1")
```

### Ejemplo de uso

```
override suspend fun getLatestLocations(expirationDate: Long): Either<AppDataError,
↪  List<DataLocation>> {
        return Either.catch {
            locationsDao.getLocations(expirationDate)
        }
            .mapLeft { ReadAppDataError }
            .map { locations ->
                locations.map { location -> location.toDataLocation() } }
    }
```

Esta función se encarga de obtener de la base de datos las últimas posiciones disponibles, toda consulta puede causar un error, si es así se almacenará en el lado izquierdo *Either*, es decir, como un *AppDataError*. De esta forma se podrá manejar el error desde un sitio más adecuado. Si la consulta se realiza sin excepciones la función devolverá la lista de posiciones.

#### Ejemplo de recepción de datos

```
locationUseCases.getLocations().fold(
    ifRight = {
        _locationState.value = _locationState.value.copy(locations = it.map { location
        ↪  ->
            location.toTrainDisplay(_currentLocation.value)
            }
    },
    ifLeft = { _locationState.value = _locationState.value.copy(error = Error.DBError)
    ↪  }
        )
```

Al recibir los datos se deberá comprobar si devuelve una excepción o la lista de trenes, es decir si devuelve el valor de la izquierda (Excepción) o derecha (lista de posiciones).

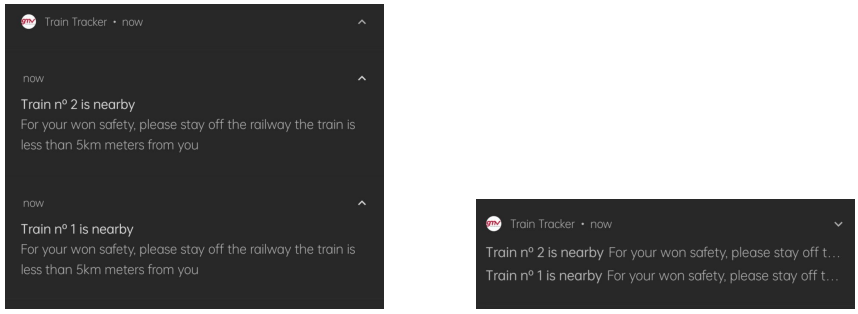
## 7.13. Notificaciones

Uno de los requisitos del proyecto es que se alerte al usuario cuando un tren está dentro de un radio de distancia configurable a nivel de servidor. El radio límite ya lo hemos obtenido al pedirlo al *Back-End*, además la alerta podrá ser, visual, sonora o de vibración.

La alerta visual aparecerá en la barra de notificaciones del dispositivo. En el caso de que lleguen varias, se agruparán, mostrando solamente un resumen del mensaje.

#### Alerta de tren cercano

En el caso de la alerta de tren cercano, cada vez que lleguen nuevas posiciones se comprobará si alguna de ellas entra dentro del límite indicado en los ajustes de servidor.




(a) Expandidas (b) Agrupadas

Figura 7.8: Notificaciones de tren cercano.

#### Alerta de datos no actualizados

La alerta de datos no actualizados saltará cuando la posición más reciente lleve sin actualizarse el tiempo indicado en los ajustes de servidor.

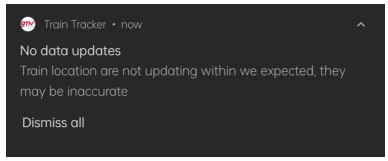



Figura 7.9: Notificaciones de tren cercano.

Ambas notificaciones visuales podrán ser pulsadas, si el usuario está autenticado, le llevará al mapa mientras que si no está autenticado le saldrá la pantalla de inicio de sesión.

La alerta podrá ser también sonora, siempre que el dispositivo tenga el volumen activado, o de vibración.

## 7.14. Ofuscación del código

El objetivo real de la ofuscación es encubrir el código para proteger la propiedad intelectual del fragmento, y que sin embargo siga funcionando la aplicación sin problema. Para ello utilizaremos una herramienta que se activa generalmente en el *build type* de *release*.

#### Funciones de Pro Guard

Cada librería que añadimos como dependencia en el fichero de configuración *gradle*, se añaden al proyecto, es decir, añaden peso innecesario a la aplicación ya que habrá funciones de esas librerías que no utilicemos. *Pro guard* se encargará de eliminar todas las funciones, variables, ficheros de recursos que no se utilicen con el fin de reducir peso.

La ofuscación de código que aplica *Pro guard* se basa en cambiar los nombres de cada variable y función por nombres con las mínimas letras posible, por ejemplo, si existe una instancia de una clase llamada *locationsRepository* que implementa una función llamada *getLocations()* la llamada a la función pasaría a ser: *a.a()*. *Pro guard* además generará unos ficheros de texto:

- **dump.txt**: Describe la estructura interna de todas las clases dentro del fichero apk.
- **mapping.txt**: Lista de mapeo de nombres de las clases, métodos y nombres de variables ofuscadas. Este fichero es necesario paraq que, cuando llega un informe de error, se pueda identificar las clases que provocan ese error.

- **seeds.txt**: Lista de clases que no han sido ofuscadas.
- **usage.txt**: Lista de código eliminado del fichero apk.

Para activar la función se debe añadir en el fichero de configuración este bloque: *gradle*

```
release {
    isMinifyEnabled = true // activa ofuscación de código
    isShrinkResources = true // activa ofuscación de recursos
    signingConfig = signingConfigs.getByName("release")
    proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"),
    ↪  "proguard-rules.pro") // indica el nombre de reglas
}
```

#### Reglas

Puede que al ofuscar todo el código, haya problemas en alguna clase que no reconoce algún nombre nuevo propuesto por *Pro Guard* y eso hace que *crashea* la aplicación. En esta aplicación esto sucede cuando se utilizan herramientas de serialización *Json* como *Kotlinx* o *Gson*. Para evitar errores se debe añadir una serie de reglas para que no cambien el nombre de esas clases y así los encargados de serializar reconozcan los nombres.

## 7.15. Actualización de versiones

Una vez lanzada la aplicación, se realizará un mantenimiento de la misma, dando lugar a futuras versiones. Estas versiones pueden ser de dos tipos:

- **Obligatorias:** Por ejemplo un cambio de *endpoint* del *Back-End* o del *bróker* MQTT. Si permitiesemos utilizar la aplicación con una versión anterior, no sería utilizable.
- **Opcional:** Se eligirá este tipo cuando los cambios no afecten, es decir, que la versión anterior conserve todas sus funcionalidades.

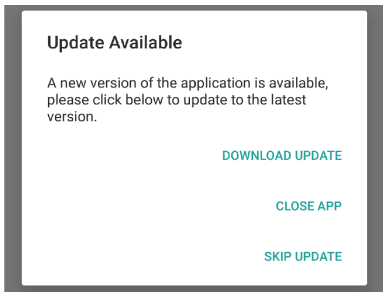
Para gestionar la actualización de versiones se usará una modificación de la siguiente librería:

```
implementation("com.btkelly:gandalf:1.4.1")
```

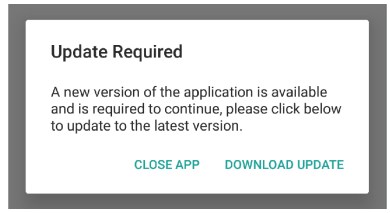
Se proporcionará un archivo Json a la instancia *Gandalf* de este formato:

```
{
  "android": {
    "alert": {
      "message": "We are currently performing server maintenance. Please try again later.",
      "blocking": true
    },
    "optionalUpdate": {
      "optionalVersion": "6",
      "message": "A new version of the application is available, please click below to update to the latest version."
    },
    "requiredUpdate": {
      "minimumVersion": "7",
      "message": "A new version of the application is available and is required to continue, please click below to update to the latest version."
    }
  }
}
```

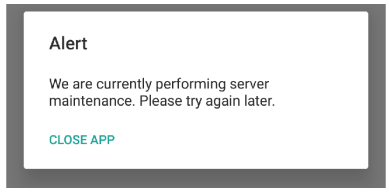
Esto le dará información a la librería para lanzar un *Alert Dialog* correspondiente, en caso de alerta, bloqueará el acceso a la aplicación, en caso de actualización opcional se dará la opción de descargar o pasar a la aplicación directamente y la actualización obligatoria que sólo permitirá descargar la aplicación.




(a) Actualización opcional




(b) Actualización requerida




(c) Alerta por mantenimiento

Figura 7.10: *Alert Dialogs* que proporciona la librería btkelly/gandalf.

Por defecto la librería mencionada al pulsar el botón de descargar, navegará al usuario hasta la página de *Google play* con el mismo nombre de paquete. Sin embargo en este proyecto las nuevas versiones estarán alojadas en un servidor y se podrá acceder a través de una url[3].

Para ello, se ha realizado un *fork*[4] [28]. del proyecto para modificar la parte de la descarga de la nueva versión.

Además de la ubicación del archivo *Json* habrá que suministrar la url donde se ubicará el archivo con la nueva versión.

[3]Cada aplicación debe tener un nombre de paquete único, por ejemplo el paquete de *Whatsapp* es *com.whatsap*.

[4]Copia idéntica

# Capítulo 8

# Pruebas

En este capítulo se mostrarán las pruebas realizadas llevadas a cabo para comprobar el correcto funcionamiento de la aplicación.

## 8.1. Pruebas de aceptación

Estas pruebas se han realizado una vez se ha completado la implementación de la aplicación. Se comprobarán las funcionalidades más importantes y cómo se comporta el sistema ante su uso en el mundo real.

| | |
|---|---|
| Identificador | PA-01. |
| Nombre | Inicio de sesión con datos correctos. |
| Prueba | El usuario introduce los credenciales de prueba correctos. |
| Resultado | El servidor devuelve los tokens de acceso y la aplicación navega hasta la pantalla de permisos. |
| Valoración | Fallo. |

Tabla 8.1: PA-01. Inicio de sesión con datos correctos.

#### Corrección PA-01

El servidor devuelve error, porque no reconoce el formato recibido. El error estaba en el campo de usuario, que incorrectamente estaba marcado como *email*, mientras que el correcto sería *user*.

| | |
|---|---|
| Identificador | PA-02. |
| Nombre | Inicio de sesión con datos incorrectos. |
| Prueba | El usuario introduce los credenciales de prueba incorrectos. |
| Resultado | El servidor devuelve error y se muestra un cuadro de diálogo informativo. |
| Valoración | Correcta. |

Tabla 8.2: PA-02. Inicio de sesión con datos incorrectos.

| | |
|---|---|
| Identificador | PA-03. |
| Nombre | Inicio sesión campos vacíos. |
| Prueba | El usuario pulsa el botón de inicio sesión sin completar algún campo. |
| Resultado | Se muestra un cuadro de diálogo informativo. |
| Valoración | Correcta. |

Tabla 8.3: PA-03. Iniciar sesión campos vacíos.

| | |
|---|---|
| Identificador | PA-04. |
| Nombre | Aceptar permisos. |
| Prueba | Aceptar permisos de localización. |
| Resultado | Se navega a la pantalla de mapa. |
| Valoración | Correcta. |

Tabla 8.4: PA-04. Aceptar permisos.

| | |
|---|---|
| Identificador | PA-05. |
| Nombre | Rechazar permisos. |
| Prueba | Rechazar permisos de localización. |
| Resultado | Se muestra un texto informativo acerca de la necesidad de aceptar permisos y como hacerlo. |
| Valoración | Correcta. |

Tabla 8.5: PA-05. Rechazar permisos.

| | |
|---|---|
| Identificador | PA-06. |
| Nombre | Obtener posiciones HTTPS. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones no caducadas en la base de datos del servidor. |
| Resultado | Se muestran las posiciones sobre el mapa. |
| Valoración | Correcta. |

Tabla 8.6: PA-06. Obtener posiciones HTTPS.

| | |
|---|---|
| Identificador | PA-07. |
| Nombre | Obtener posiciones HTTPS caducadas. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones caducadas en la base de datos del servidor. |
| Resultado | No se muestran las posiciones sobre el mapa. |
| Valoración | Correcta. |

Tabla 8.7: PA-07. Obtener posiciones HTTPS.

| | |
|---|---|
| Identificador | PA-08. |
| Nombre | Obtener topología HTTPS. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar topología en la base de datos del servidor. |
| Resultado | Se muestran las estaciones sobre el mapa y los sinópticos al pulsar en el filtro. |
| Valoración | Correcta. |

Tabla 8.8: PA-08. Obtener topología HTTPS.

| | |
|---|---|
| Identificador | PA-09. |
| Nombre | Notificación visual tren cercano. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones cercanas a la ubicación del usuario en la base de datos del servidor. |
| Resultado | Se muestra la notificación en la barra de notificaciones. |
| Valoración | Fallo. |

Tabla 8.9: PA-09. Notificación visual tren cercano.

**Correción PA-09**

Las notificaciones se mostraban correctamente, pero no se agrupaban correctamente, si no que salían de forma independiente. Para solucionar este problema se añade una notificación de grupo cada vez que se activa la alerta y el usuario tiene activada la notificación visual, agrupando todas estas notificaciones con un mismo identificador de grupo, se añaden correctamente.

| | |
|---|---|
| Identificador | PA-10. |
| Nombre | Notificación sonora tren cercano. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones cercanas a la ubicación del usuario en la base de datos del servidor. |
| Resultado | Suena. |
| Valoración | Correcta. |

Tabla 8.10: PA-10. Notificación visual tren cercano.

| | |
|---|---|
| Identificador | PA-11. |
| Nombre | Notificación de vibración de tren cercano. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones cercanas a la ubicación del usuario en la base de datos del servidor. |
| Resultado | Vibra. |
| Valoración | Correcta. |

Tabla 8.11: PA-11. Notificación visual tren cercano.

| | |
|---|---|
| Identificador | PA-12. |
| Nombre | Notificación visual de datos no actualizados. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones no actualizadas en la base de datos del servidor. |
| Resultado | La notificación se muestra en la barra de notificaciones. |
| Valoración | Correcta. |

Tabla 8.12: PA-12. Notificación visual de datos no actualizados.

| | |
|---|---|
| Identificador | PA-13. |
| Nombre | Notificación sonora de datos no actualizados. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones no actualizadas en la base de datos del servidor. |
| Resultado | Suena. |
| Valoración | Correcta. |

Tabla 8.13: PA-13. Notificación sonora datos no actualizados.

| | |
|---|---|
| Identificador | PA-14. |
| Nombre | Notificación de vibración de datos no actualizados. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y almacenar posiciones no actualizadas en la base de datos del servidor. |
| Resultado | Vibra. |
| Valoración | Correcta. |

Tabla 8.14: PA-14 Notificación de vibración datos no actualizados.

| | |
|---|---|
| Identificador | PA-15. |
| Nombre | Obtención posiciones vía MQTT. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y publicar posiciones en el broker MQTT. |
| Resultado | Se muestran las posiciones en el mapa. |
| Valoración | Correcta. |

Tabla 8.15: PA-15 Obtención posiciones vía MQTT.

| | |
|---|---|
| Identificador | PA-16. |
| Nombre | Borrado de posiciones expiradas. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y modificar el tiempo de expiración de posición a un periodo corto de tiempo. |
| Resultado | Desaparecen del mapa las posiciones. |
| Valoración | Fallo. |

Tabla 8.16: PA-16 Borrado de posiciones expiradas.

**Corrección PA-16**

Los trenes a punto de salir no se borraban correctamente porque su Instant en la aplicación figura como null. Para corregirlo, se añade como al atributo *timestamp* el valor de *Instant.now()* al recibir la posición del tren. Para diferenciar de un tren a punto de salir de un tren en trayecto se ha añadido un nuevo atributo.

| | |
|---|---|
| Identificador | PA-17. |
| Nombre | Cerrar sesión. |
| Prueba | Inicio de sesión con datos correctos, aceptación de permisos y posteriormente cierre de sesión. |
| Resultado | Todos los servicios paran, comprobado con Chucker, desconexión del *bróker* mqtt y navegación hacia la pantalla de login. |
| Valoración | Correcta. |

Tabla 8.17: PA-17 Cerrar sesión.

# Capítulo 9

# Seguimiento del proyecto

Durante la planificación se realizó una estimación de las fases en las que se dividía el proyecto, según se puede observar en la Figura 2.2. Una vez finalizado el proyecto podemos observar los tiempos que se han cumplido y los que, por el contrario, no han sido precisos debido a la ejecución de algunos de los riesgos recogidos en la sección 2.2

En la siguiente tabla observaremos los tiempos reales de cada fase del proyecto

| Fase | Inicio estimado | Inicio real | Fin estimado | Fin real |
|---|---|---|---|---|
| Estudio de Factibilidad | 10/02/2022 | 15/02/2022 | 16/02/2022 | 16/02/2022 |
| Análisis de requisitos | 16/02/2022 | 16/02/2022 | 17/02/2022 | 22/02/2022 |
| Análisis del sistema | 18/02/2022 | 23/02/2022 | 25/02/2022 | 28/02/2022 |
| Diseño de sistema | 28/02/2022 | 29/02/2022 | 16/03/2022 | 10/03/2022 |
| Codificación | 17/03/2022 | 11/03/2022 | 24/05/2022 | 03/06/2022 |
| Pruebas | 25/05/2022 | 06/05/2022 | 02/06/2022 | 08/06/2022 |
| Mantenimiento | 03/06/2022 | 09/02/2022 | - | - |

Tabla 9.1: Comparación de tiempo de la completitud de las fases.

Como se puede observar las fechas estimadas no se han completado con exactitud debido a la ejecución de los riesgos 2.4 y 2.7. A pesar de ello gracias a que se planificó para días antes de lo necesario, se ha conseguido realizar el proyecto en una fecha cercana a la fecha fin estimada.

# Capítulo 10

# Conclusiones y trabajo futuro

Una vez finalizado el proyecto, se puede hacer una valoración general de lo que ha sido el proyecto para mí, las dificultades y sobre todo lo que he aprendido.

En primer lugar, cuando me ofrecieron el proyecto, con una tecnología que era nueva para mí, me asustó, pensé que no iba a ser capaz de ni siquiera empezarlo. La fase de formación fue fundamental, ya que me dio los conocimientos necesarios para comenzar la aplicación.

Poco a poco, según iba completando tareas, iba tomando forma y cada vez me sentía más esperanzado. Finalmente puedo decir que estoy muy orgulloso de mi trabajo y de lo que he aprendido.

He comprendido la importancia de realizar todos los pasos previos a empezar a programar, como el análisis, el diseño o la planificación, he mejorado mi capacidad de buscar información útil en Internet, lo cual ha sido vital para el desarrollo del proyecto, he conocido lo que es estar en un entorno real de trabajo, he fortalecido mis *soft skills* en cada reunión de proyecto.

En resumen, la balanza general del proyecto ha resultado ser muy positiva, a pesar de todo el trabajo dedicado, se han cumplido todos los objetivos marcados en la sección 1.3 al comienzo de este trabajo.

La aplicación carece de demasiada funcionalidad, pero es cierto que, con la poca información que recibe no puede hacer mucho más. Como mejoras futuras se podría añadir:

- **Tiempo restante de los trenes a la siguiente parada**. Esto actualmente no se puede hacer porque la información recibida del tren no contiene ningún dato que indique la velocidad actual del tren. Se podría hacer una estimación pero no llegaría a ser tan precisa si se tuviesen el histórico de velocidades en cada trayecto.
- **Filtro de búsqueda de tren por número de tren**. Esta funcionalidad se podría incluir sin problema, serviría para identificar un tren específico, aunque el propósito de la aplicación es alertar de los trenes cercanos.
- **Integración con *iOS***. Al estar escrita en *Kotlin*, habría mucho trabajo ya realizado para empezar a integrarla con el sistema operativo de *Apple*.
- **Mostrar incidencias en el mapa.** Para ello, sería necesario que se recibiese información acerca de posibles tramos cortados, accidentes, o alguna incidencia que pueda surgir en las vías de tren.
- **Mostrar información de las vías que se encuentran en mantenimiento.** Igual que los trenes avisarán de su posición al personal de las vías, se podría incluir que el personal de mantenimiento indique qué vía se encuentra en mantenimiento, a los trenes. Esto ampliaría el número de usuarios de la aplicación y aumentaría la seguridad, ya que se sumaría un nuevo canal de comunicación.

# Bibliografía

[1] Robert C. Martin (Uncle Bob). Clean architecture: A craftsman's guide to software structure and design. (2017).

[2] Bob Hughes, Mike Cotterell, Rajib Mall. Software project management, fifth edition. (2011).

[3] Glassdoor team. Glassdoor. https://www.glassdoor.es/member/home/index.htm:. (2022).

[4] OASIS MQTT Technical Committee. Mqtt specifications. https://mqtt.org/mqtt-specification/. (2022).

[5] Android developers. Meet android studio. https://developer.android.com/studio/intro. (2022).

[6] Android developers. Develop android apps with kotlin. https://developer.android.com/kotlin. (2022).

[7] Jetbrains. A modern programming language that makes developers happier. kotlin v1.6.21. https://kotlinlang.org/. (2022).

[8] Android developers. Jetpack compose. https://developer.android.com/jetpack/compose. (2022).

[9] Material Components. Build beautiful, usable products with material components for android, flutter, ios, and the web. https://material.io. (2022).

[10] Material Components. Dark theme. https://material.io/design/color/dark-theme.html. (2022).

[11] Android developers. Introduction to animations. https://developer.android.com/training/animation/overview. (2022).

[12] Wikipedia. Microsoft project. https://es.wikipedia.org/wiki/Microsoft_Project. (2022).

[13] Wikipedia. Microsoft excel. https://es.wikipedia.org/wiki/Microsoft_Excel. (2022).

[14] Wikipedia. Astah. `https://en.wikipedia.org/wiki/Astah*`. (2022).

[15] GitLab. The one devops platform. `https://about.gitlab.com/`. (2022).

[16] Atlassian. Sourcetree. a free git client for windows and mac. `https://www.sourcetreeapp.com/`. (2022).

[17] Carlos Cuesta. Gitmoji. an emoji guide for your commit messages. `https://gitmoji.dev/`. (2022).

[18] Nicola Corti, Olivier Perez. Chucker v3.5.2. `https://github.com/ChuckerTeam/chucker`. (2022).

[19] Abhinav Asthana, Ankit Sobti y Abhijit Kane. Postman. learning center. `https://learning.postman.com/docs/getting-started/introduction/`. (2022).

[20] Overleaf Team. Overleaf documentation. `https://www.overleaf.com/learn`. (2022).

[21] Android developers. Codelabs de conceptos básicos para desarrolladores de android. `https://developer.android.com/courses/kotlin-android-fundamentals/toc`. (2022).

[22] JetBrains. Kotlin koans. `https://kotlinlang.org/docs/koans.html`. (2022).

[23] Android developers. Componentes y diseños de material. hojas inferiores. `https://developer.android.com/jetpack/compose/layouts/material#bottom-sheetsl`. (2022).

[24] UCL, Fastly, Bytemark Hosting. Openstreetmap. `www.openstreetmap.org`. (2022).

[25] Material Components. Material design principles. `https://material.io/design/layout/understanding-layout.html#principles`. (2022).

[26] JetBrains. Ktor.create asynchronous client and server applications. `https://ktor.io`. (2022).

[27] Oracle. Instant (java se 11 & jdk 11 ) - oracle. `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/Instant.html`. (2022).

[28] Bryan Kelly, Albin Poignot, Pierre Duchemin, Jeff Sibbold, Stuart Kent, Kyle Lehman, Frieder Bluemle. Github, btkelly/gandalf. `https://github.com/btkelly/gandalf`. (2022).

[29] Android Developers. Android studio. download. `https://developer.android.com/studio`. (2022).

# Apéndice A

# Manuales

## A.1. Manual de mantenimiento

Un proyecto *Android* por lo general tiene una sencilla instalación, pero en este caso hay que tener varias consideraciones en cuenta.

La primera consideración, es que el proyecto se realiza con *Jetpack Compose*, por lo tanto es necesario disponer de una versión reciente, se recomienda descargar la última disponible [29]. Además, el proyecto utiliza una librerías externas, como la ya comentada modificación de *gandalf*.

Una vez dispongamos de la última versión disponible, se deberán seguir los siguientes pasos:

1. Descargar Maven si no se tiene instalado.

2. Una vez instalado, localizamos el repositorio local, por defecto se encuentra:

```
Windows 10: C:/Users/<username>/.m2/repository
Linux:      /home/<username>/.m2/repository
Mac:        /Users/<username>/.m2/repository
```

3. Extraer el fichero *com.zip* proporcionado en el repositorio tal que siga la siguiente jerarquía de directorios:

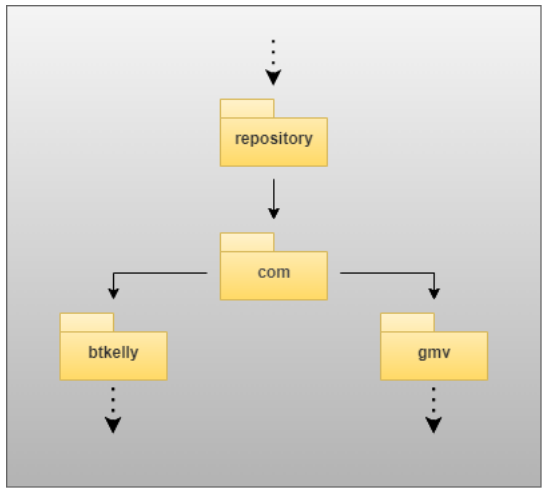



Figura A.1: Jerarquía de directorios

4. Extraer, ahora si el proyecto en la carpeta que desee.
5. Abrir el proyecto en *Android Studio*. *File* → *Open*, → seleccionamos la carpeta *root* de nuestro proyecto.
6. Esperar varios minutos a que *Gradle* haga su trabajo.
7. Ya podrá mantener el código.

# Apéndice B

# Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio donde se encuentra el entregable: `https://gitlab.inf.uva.es/oscarag/tfgoscararagonesteban`

  El entregable es un fichero comprimido que contiene:

  - **TrainTrackerOscarAragonEsteban.zip**: Fichero zip que contiene el código de la aplicación.
  - **com.zip**: Fichero zip que contiene las librerías que hay que guardar según se indica en el manual A