UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación móvil para la gestión de puntos kilométricos de una obra civil lineal. Extensión de funcionalidades

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Singh Kaur, Harkirat

Tutor/a: Busquets Mataix, José Vicente

Cotutor/a: Gómez Mataix, Gonzalo Vicente

CURSO ACADÉMICO: 2023/2024

RESUMEN

En el presente trabajo se detalla el desarrollo e implementación de una aplicación móvil para Android, diseñada para asistir a los usuarios en la localización de puntos específicos dentro de una obra lineal. Esta herramienta es el resultado de la colaboración y la visión original del Dr. Ing. De Caminos Gonzalo Gómez Mataix, quien ha desempeñado un papel fundamental en la dirección de este proyecto.

La principal funcionalidad de esta aplicación es permitir la captura de puntos de interés a lo largo de una obra lineal utilizando el GPS del dispositivo móvil. Esto proporciona a los usuarios la capacidad de crear y gestionar sus propias obras lineales, almacenando cada punto capturado en una base de datos para futuras referencias. Adicionalmente, la aplicación calcula la distancia más corta entre la ubicación actual del usuario y cualquier punto registrado en la obra lineal, facilitando la comunicación de coordenadas y datos relevantes entre usuarios.

Las posibles aplicaciones de esta herramienta son variadas, incluyendo la identificación de incidencias durante la explotación de recursos o la localización de elementos específicos como compuertas o mecanismos asociados a la obra lineal.

Para la realización de este proyecto, se ha partido de una aplicación móvil estable, y se ha mejorado y ampliado sus funcionalidades para ofrecer una herramienta más completa y eficiente. Finalmente, se llevaron a cabo una serie de pruebas exhaustivas para identificar y corregir errores potenciales. Además, se documenta el proceso completo de despliegue de la aplicación en la Google Play Store.

Palabras clave: Android, obra lineal, aplicación, punto kilométrico, base de datos, GPS, despliegue.

RESUM

En aquest treball es detalla el desenvolupament i implementació d'una aplicació mòbil per a Android, dissenyada per assistir els usuaris en la localització de punts específics dins d'una obra lineal. Aquesta eina és el resultat de la col·laboració i la visió original del Dr. Ing. De Camins Gonzalo Gómez Mataix, qui ha jugat un paper fonamental en la direcció d'aquest projecte.

La principal funcionalitat d'aquesta aplicació és permetre la captura de punts d'interès al llarg d'una obra lineal utilitzant el GPS del dispositiu mòbil. Això proporciona als usuaris la capacitat de crear i gestionar les seues pròpies obres lineals, emmagatzemant cada punt capturat en una base de dades per a futures referències. Addicionalment, l'aplicació calcula la distància més curta entre la ubicació actual de l'usuari i qualsevol punt registrat en l'obra lineal, facilitant la comunicació de coordenades i dades rellevants entre usuaris.

Les possibles aplicacions d'aquesta eina són variades, incloent la identificació d'incidències durant l'explotació de recursos o la localització d'elements específics com comportes o mecanismes associats a l'obra lineal.

Per a la realització d'aquest projecte, es va partir d'una aplicació mòbil estable, i es van millorar i ampliar les seues funcionalitats per oferir una eina més completa i eficient. Finalment, es van dur a terme una sèrie de proves exhaustives per identificar i corregir errors

potencials. A més, es documenta el procés complet de desplegament de l'aplicació en la Google Play Store.

Paraules clau: Android, obra lineal, aplicació, punt quilomètric, base de dades, GPS, desplegament

ABSTRACT

This paper describes the development and implementation of a mobile application for Android, designed to assist users in locating specific points within a linear work. This tool is the result of the collaboration and original vision of Dr. Ing. De Caminos Gonzalo Gómez Mataix, who has played a crucial role in directing this project.

The main functionality of this application is to allow the capture of points of interest along a linear work using the mobile device's GPS. This enables users to create and manage their own linear works, storing each captured point in a database for future reference. Additionally, the application calculates the shortest distance between the user's current location and any registered point in the linear work, facilitating the communication of coordinates and relevant data between users.

The potential applications of this tool are varied, including the identification of incidents during resource exploitation or the location of specific elements such as gates or mechanisms associated with the linear work.

To carry out this project, a stable mobile application was used as a starting point, and its functionalities were improved and expanded to offer a completer and more efficient tool. Finally, a series of exhaustive tests were conducted to identify and correct potential errors. Additionally, the complete deployment process of the application in the Google Play Store is documented.

Keywords: Android, linear work, application, kilometer point, database, GPS, deployment.

Índice

Índice de ilustraciones

1 INTRODUCCIÓN

Este Trabajo de Fin de Grado (TFG) continúa y mejora un proyecto presentado en el curso anterior. Se basa en una aplicación móvil estable diseñada para la gestión de puntos kilométricos (PK) en obras civiles lineales, y tiene como objetivo mejorar y ampliar sus funcionalidades para ofrecer una herramienta más completa y eficiente.

La aplicación original fue concebida y desarrollada bajo la supervisión de mi tutor, Jose Vicente Busquets, junto con el Dr. Ing. de Caminos Gonzalo Gómez Mataix. El Dr. Gómez continúa como codirector en esta fase del proyecto, aportando su experiencia y conocimientos para guiar las mejoras y extensiones propuestas.

1.1 CONTEXTO Y MOTIVACIÓN

Para comprender el funcionamiento de esta aplicación, es fundamental explicar qué es una obra lineal y qué es un punto kilométrico (en adelante PK), ya que estos son los conceptos clave del proyecto.

Una obra lineal es una infraestructura cuya geometría se puede aproximar a una línea. Este tipo de obras son cruciales en el ámbito de la ingeniería civil y se dividen en varias categorías:

- Líneas de telecomunicaciones o transporte de energía: Incluyen canalizaciones de cables y líneas de alta tensión.
- Conducciones: Comprenden tuberías de agua, canales, y acueductos.
- Obras de defensa marítimas o hidráulicas: Engloban encauzamientos fluviales y diques marítimos.
- Vías de comunicación: Abarcan caminos, carreteras, autovías, puentes y líneas de ferrocarril.

La característica distintiva de estas obras es su extensión significativa en una dimensión, lo que hace esencial la determinación precisa de cualquier punto a lo largo de su longitud.

El método más efectivo para establecer un sistema de referencia en estas obras es mediante la medición del "camino recorrido" desde el inicio de la obra, conocido como PK (Punto Kilométrico). El PK se expresa mediante una cantidad entera de kilómetros seguida de un número de metros adicionales, por ejemplo, PK 1+250 indica un punto situado a 1 kilómetros y 250 metros del inicio de la obra.

El uso de PK es fundamental para la planificación, construcción y mantenimiento de las obras lineales, ya que permite una localización precisa y facilita la comunicación entre los equipos de trabajo. Sin un sistema de referencia claro, la gestión de estas infraestructuras sería mucho más compleja y propensa a errores.

Este proyecto surge de la necesidad de mejorar y ampliar las funcionalidades de una aplicación existente que ha demostrado ser efectiva en la gestión de PK gracias a funcionalidades como la captura de PKs y el cálculo del PK más cercano al usuario. Con el objetivo de aumentar la precisión, usabilidad y capacidad de respuesta de la aplicación, se propone implementar nuevas características y migrar a tecnologías más modernas y mantenibles.

1.2 OBJETIVOS

Los objetivos de este proyecto son mejorar la interfaz de usuario para proporcionar una experiencia más intuitiva, corregir las deficiencias identificadas en la versión anterior y ampliar las capacidades de la aplicación. Asimismo, se buscará mejorar la calidad del código mediante la migración a un lenguaje más moderno y la implementación de mejores prácticas de desarrollo. Finalmente, se planea llevar a cabo el despliegue exitoso de la aplicación en Google Play Store para aumentar su accesibilidad y su alcance entre los usuarios.

1.3 ESTRUCTURA DE LA MEMORIA

La memoria se estructura en ocho capítulos, cada uno detallando una fase específica del proyecto:

Introducción Presenta el contexto, la motivación y los objetivos del proyecto, proporcionando una visión general de la aplicación original y la necesidad de su mejora.

Contexto Tecnológico Describe las tecnologías utilizadas en el desarrollo del proyecto, incluyendo las plataformas, lenguajes de programación, herramientas y librerías. Además, explica la razón detrás de la selección de estas tecnologías y cómo se alinean con los objetivos del proyecto.

Análisis Primero se analiza la aplicación existente para identificar sus puntos fuertes y áreas de mejora. Posteriormente, se establecen los requisitos para la mejora, detallando las funcionalidades que deben ser desarrolladas y los aspectos no funcionales a considerar, como la migración a Kotlin y la mejora de la estructura del código.

Diseño Expone el diseño de la aplicación, incluyendo diagramas de arquitectura, la estructura de las capas de presentación, dominio y datos, y la planificación de la interfaz de usuario.

Desarrollo e Implementación Documenta el proceso de desarrollo de las nuevas funcionalidades, la migración del código de Java a Kotlin, y la implementación de la arquitectura recomendada por Google. Se incluye también el uso de la librería Room y Hilt para la gestión de la base de datos y la inyección de dependencias.

Pruebas Realizadas Detalla las pruebas realizadas para verificar el funcionamiento correcto de las nuevas funcionalidades, describiendo los escenarios de prueba, los resultados obtenidos y los problemas solucionados durante el proceso.

Despliegue Este capítulo detalla el proceso de preparación y publicación de la aplicación en Google Play Store. Se describen los pasos necesarios para cumplir con los requisitos de la plataforma y asegurar su aceptación y distribución. Además, se incluyen imágenes explicativas para facilitar la comprensión del proceso de despliegue.

Conclusiones Resume los logros alcanzados, las mejoras implementadas y los beneficios obtenidos con la nueva versión de la aplicación. Además, se discuten las limitaciones encontradas y las posibles futuras líneas de desarrollo.

2 CONTEXTO TECNOLÓGICO

En este capítulo se describen las tecnologías y herramientas utilizadas en el desarrollo de la aplicación, así como las razones detrás de su selección. Se exploran los lenguajes de programación, frameworks, librerías y otras herramientas esenciales que permitieron llevar a cabo este proyecto con éxito.

2.1 LENGUAJES DE PROGRAMACIÓN

Java

Originalmente, la aplicación fue desarrollada en Java, un lenguaje de programación ampliamente utilizado en el desarrollo de aplicaciones Android. Java es conocido por su portabilidad, robustez y extensa comunidad de desarrolladores. Sin embargo, con la evolución de las tecnologías y las recomendaciones de Google, se ha decidido migrar el código de Java a Kotlin para aprovechar las ventajas que este último ofrece.

Kotlin

Kotlin es un lenguaje de programación moderno y conciso que se ha convertido en el lenguaje preferido para el desarrollo de aplicaciones Android. Desarrollado por JetBrains y soportado oficialmente por Google, Kotlin ofrece varias ventajas sobre Java, como una sintaxis más clara y segura, compatibilidad completa con Java, y un soporte mejorado para la programación funcional. Estas características hacen que el código sea más mantenible y menos propenso a errores, resultando en una aplicación más robusta y eficiente.

2.2 ANDROID

Android es un sistema operativo basado en Linux, desarrollado por Google, que se utiliza principalmente en dispositivos móviles como smartphones y tabletas. Android proporciona una plataforma abierta que permite a los desarrolladores crear aplicaciones innovadoras y de alto rendimiento para una amplia gama de dispositivos.

Android es el sistema operativo móvil más utilizado en el mundo, con una cuota de mercado superior al 70%, lo que lo convierte en una plataforma crucial para el desarrollo de aplicaciones móviles. Esta popularidad se debe a su flexibilidad, amplia gama de dispositivos compatibles y fuerte respaldo de Google.

2.2.1 Arquitectura

Aplicaciones: La capa más alta que interactúa directamente con los usuarios finales. Incluye aplicaciones básicas como teléfono, contactos, y navegador, así como aplicaciones desarrolladas por terceros.

Framework de Aplicaciones: Proporciona bibliotecas y servicios que permiten a los desarrolladores crear aplicaciones innovadoras y eficientes. Incluye componentes clave como Activity Manager, Content Providers, y Resource Manager.

Bibliotecas: Esta capa incluye un conjunto de bibliotecas C/C++ utilizadas por varios componentes del sistema operativo Android. Incluye bibliotecas como libc, media framework y Surface Manager.

Android Runtime: Incluye la Máquina Virtual de Android (Dalvik) y el entorno de tiempo de ejecución ART (Android Runtime) que ejecutan las aplicaciones en dispositivos Android.

Hardware Abstraction Layer (HAL): Proporciona una interfaz estándar para interactuar con el hardware del dispositivo, permitiendo que los componentes de hardware se puedan utilizar de manera uniforme por las aplicaciones.

Kernel de Linux: La base del sistema operativo Android que proporciona servicios fundamentales como seguridad, gestión de memoria y redes.

Se puede observar de manera clara las distintas capas explicadas previamente en la Imagen 1.

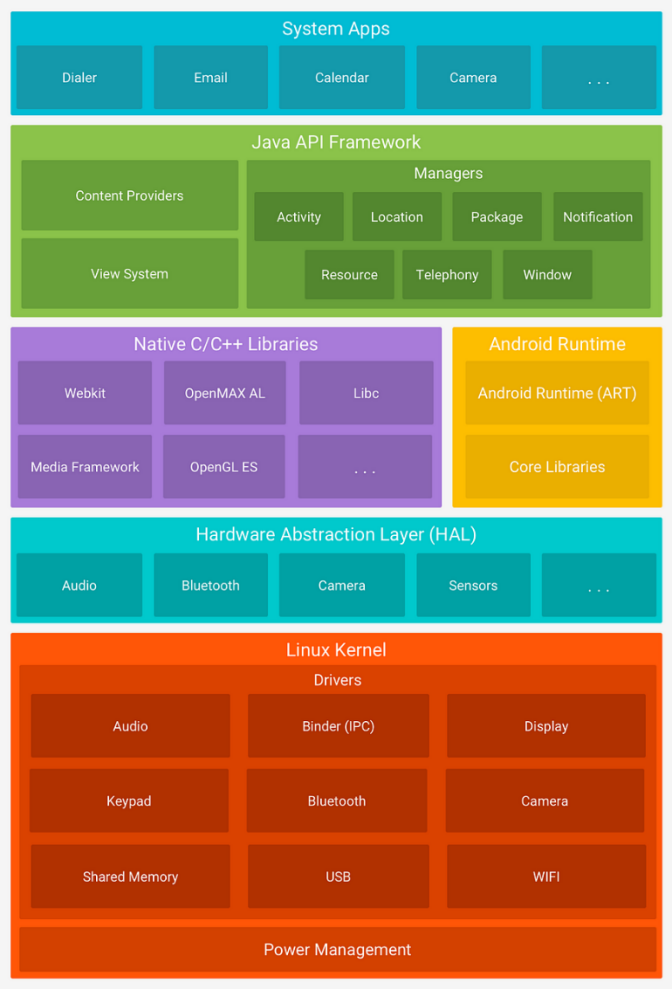


Imagen 1. Arquitectura de Android. Fuente: Artículo de web[8]

2.2.2 Versiones

Android ha lanzado múltiples versiones a lo largo de los años. La aplicación desarrollada en este proyecto es compatible hasta la API 21 (Android Lollipop), asegurando que pueda ser utilizada en una amplia gama de dispositivos como podemos observar en la Imagen 2.

Android Lollipop (API 21): Introdujo Material Design y mejoras significativas en el rendimiento y la eficiencia energética.

Android Marshmallow (API 23): Aportó características como permisos de aplicaciones granulares y Doze mode.

Android Nougat (API 24-25): Introdujo la capacidad de multiventana y mejoras en las notificaciones.

Android Oreo (API 26-27): Mejoró la gestión de fondo y las notificaciones.

Android Pie (API 28): Aportó un nuevo sistema de navegación por gestos y mejoras en el bienestar digital.

Android 10 (API 29) y superiores: Introdujeron una navegación por gestos completamente nueva, un tema oscuro a nivel de sistema y mejoras de privacidad.

Version	SDK / API level	Version code	Codename	Cumulative usage [1]	Year [4]
Android 15 DEV	Level 35	VANILLA_ICE_CREAM	Vanilla Ice Cream [2]	—	*TBD*
Android 14	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake [2]	16.3%	2023
	▪ `targetSdk` will need to be 34+ for new apps and app updates by August 31, 2024.				
Android 13	Level 33	TIRAMISU	Tiramisu [2]	42.5%	2022
	▪ `targetSdk` must be 33+ for new apps and app updates since August 31, 2023.				
Android 12	Level 32 Android 12L	S_V2	Snow Cone [2]	59.5%	
	Level 31 Android 12	S			2021
Android 11	Level 30	R	Red Velvet Cake [2]	75.7%	2020
Android 10	Level 29	Q	Quince Tart [2]	84.5%	2019
Android 9	Level 28	P	Pie	90.2%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	92.1%	2017
	Level 26 Android 8.0	O		95.1%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	95.6%	2016
	Level 24 Android 7.0	N		97.0%	
Android 6	Level 23	M	Marshmallow	98.4%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	99.2%	
	Level 21 Android 5.0	LOLLIPOP, L		99.5%	2014
	▪ Jetpack/AndroidX libraries require a `minSdk` of 21 or higher since April 2024. ▪ Jetpack Compose requires a `minSdk` of 21 or higher. ▪ Google Play services v23.30.99+ (August 2023) drops support for API levels below 21.				

Imagen 2. Versiones de Android. Fuente: Página web[7]

2.2.3 Componentes de una aplicación

Activity: Las actividades constituyen el componente principal de la interfaz gráfica de cualquier aplicación. Podemos equiparar una actividad a una ventana o pantalla. Cada actividad incluye una clase, donde se define el código que se ejecutará en esa pantalla, y un layout, que determina el diseño visual de la actividad. Además, el archivo que define estas actividades se implementa como una subclase de la clase Activity.

View: Las vistas son los bloques básicos utilizados para construir la interfaz gráfica de una aplicación. Android ofrece una amplia gama de controles básicos, tales como botones, cuadros de texto, imágenes y listas desplegables. Estas vistas pueden ser creadas o modificadas mediante archivos XML o directamente en el código.

Layout: Los layouts especifican la estructura visual de la aplicación. Se tratan de archivos XML que definen la interfaz gráfica de las actividades.

Fragment: Un fragmento representa una parte de la interfaz de usuario o de la funcionalidad dentro de una Activity. Este componente está estrechamente relacionado con la actividad, ya que esta actúa como contenedor de uno o varios fragmentos.

Service: Los servicios son componentes de la aplicación que no poseen interfaz gráfica y se ejecutan en segundo plano. Pueden llevar a cabo tareas como actualizar datos, enviar notificaciones o interactuar con otros componentes.

Intent: Los intents son esenciales para la comunicación y la navegación entre los distintos componentes de la aplicación. Facilitan la transición de una actividad a otra y permiten el intercambio de datos entre ellas.

2.2.4 Ciclo de vida

En el desarrollo de aplicaciones Android, el ciclo de vida de una actividad es una de las áreas más importantes y distintivas. A diferencia de otros sistemas operativos, en Android, el ciclo de vida de una actividad es gestionado principalmente por el sistema, no directamente por el usuario. Esto se facilita mediante una serie de métodos de devolución de llamada que permiten a una actividad gestionar y responder a los cambios en su estado. A continuación, se describen los distintos estados:

- **Activa (Running):** En este estado, la actividad está en primer plano y es visible para el usuario, con el foco de atención. Es decir, el usuario puede interactuar directamente con ella. La actividad se encuentra encima de la pila de actividades.
- **Visible (Paused):** La actividad está visible, pero no tiene el foco. Esto ocurre cuando otra actividad está activa y puede estar mostrando una parte transparente o cubriendo parcialmente la pantalla. Si la actividad está completamente oculta, pasa al estado de parada.
- **Parada (Stopped):** En este estado, la actividad no es visible para el usuario y no se puede interactuar con ella. Además, puede ser destruida por el sistema si se requiere liberar memoria, aunque esto no ocurre necesariamente cuando la actividad entra en este estado.
- **Destruida (Destroyed):** Una actividad entra en este estado cuando se invoca el método finish() o cuando es eliminada por el sistema debido a restricciones de memoria. Una vez en este estado, la actividad ha sido completamente destruida y no puede ser reanudada.

Para manejar estos estados, Android proporciona una serie de métodos de devolución de llamada que se ejecutan en diferentes momentos del ciclo de vida de una actividad (Imagen 3):

- **onCreate():**Se llama cuando se crea la actividad por primera vez. Es el lugar ideal para inicializar la actividad, configurar la interfaz de usuario y recuperar datos guardados previamente.
- **onStart():** Este método se invoca cuando la actividad va a ser mostrada al usuario. Aquí, la actividad está iniciando su visualización, pero aún no es interactiva.
- **onResume():** Se llama cuando la actividad está a punto de comenzar a interactuar con el usuario. Es el momento adecuado para iniciar animaciones, música o cualquier otro recurso que requiera estar disponible mientras la actividad está en primer plano.
- **onPause():** Este método se ejecuta cuando la actividad va a ser enviada al segundo plano, ya que otra actividad ha sido lanzada. Se utiliza para pausar la reproducción de medios, detener servicios o guardar datos importantes para no perder el progreso.

- **onStop():** Se llama cuando la actividad ya no es visible para el usuario. Si hay una escasez de memoria, es posible que la actividad sea destruida sin ejecutar este método. Por esta razón, es esencial que onPause() se llame antes de onStop().
- **onRestart():** Este método se invoca cuando la actividad va a ser mostrada nuevamente después de haber sido detenida. Se ejecuta después de onStop() y antes de onStart().
- **onDestroy():** Se llama antes de que la actividad sea destruida. Este método es útil para liberar recursos y realizar cualquier limpieza necesaria antes de que la actividad sea eliminada completamente.

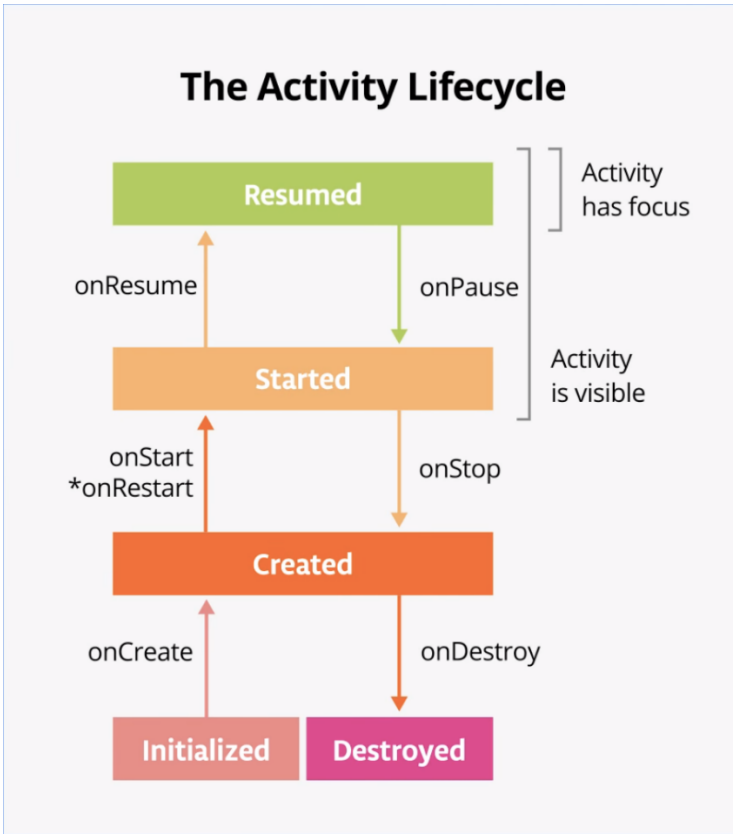


Imagen 3. Ciclo de vida. Fuente: Android Developer[2]

2.3 Entorno de Desarrollo

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android (Imagen 4). Desarrollado por Google, ofrece un conjunto completo de herramientas que facilitan la creación, prueba y depuración de aplicaciones. Entre sus características destacan:

- Un editor de código avanzado con soporte para refactorización y navegación de código.
- Un emulador de Android rápido y con diversas configuraciones de dispositivos.
- Herramientas de análisis y depuración de rendimiento.
- Integración con sistemas de control de versiones como Git.

La migración a Kotlin se ha realizado utilizando las herramientas proporcionadas por Android Studio, que permiten convertir el código Java a Kotlin de manera eficiente.

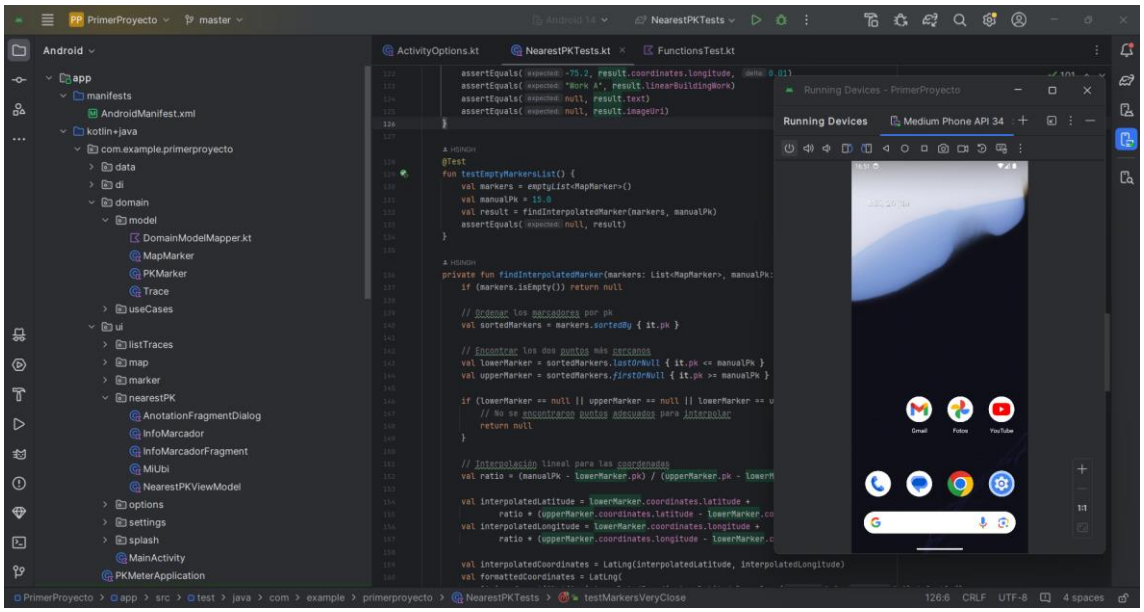


Imagen 4. Android Studio

2.4 CONTROL DE VERSIONES

Para el control de versiones durante el desarrollo se ha utilizado Git. Git es un sistema de control de versiones distribuido ampliamente adoptado en la industria del desarrollo de software por varias razones clave:

- **Gestión de Versiones:** Git permite llevar un registro detallado de todos los cambios realizados en el código fuente a lo largo del tiempo. Cada modificación es capturada como un "commit", lo que facilita la reversión a versiones anteriores en caso de necesidad.
- **Trabajo Colaborativo:** Es especialmente útil cuando se trabaja en equipo, permitiendo a múltiples desarrolladores trabajar simultáneamente en el mismo proyecto. Git gestiona de manera efectiva las fusiones de código, lo que ayuda a mantener la integridad y consistencia del proyecto.
- **Ramas (Branches):** Permite crear ramas independientes del código principal para trabajar en nuevas características o correcciones sin afectar la versión estable del software. Esto facilita el desarrollo paralelo y la experimentación.
- **Historial Completo:** Proporciona un historial detallado de cada cambio, quién lo realizó y cuándo, lo que es invaluable para el seguimiento de errores, auditorías de código y la comprensión de la evolución del proyecto a lo largo del tiempo.
- **Integración con Herramientas de Desarrollo:** Se integra perfectamente con herramientas como Android Studio, facilitando la gestión de versiones directamente desde el entorno de desarrollo.

2.5 LIBRERÍAS Y HERRAMIENTAS

2.5.1 Room

Room es una librería de persistencia proporcionada por Android que facilita la gestión de bases de datos SQLite. Ofrece una abstracción de alto nivel que permite realizar operaciones de base de datos de manera segura y eficiente. Utilizando anotaciones y generando código en tiempo de compilación, Room mejora la integridad del esquema de la base de datos y simplifica las migraciones.

2.5.2 Hilt

Hilt es una librería de inyección de dependencias para Android que simplifica la configuración y el manejo de dependencias en la aplicación. Basado en Dagger, Hilt ofrece una integración más sencilla con Android, facilitando el cumplimiento de los principios SOLID y mejorando la modularidad y mantenibilidad del código.

2.5.3 Material Design

Para el diseño de la interfaz de usuario, se ha seguido la guía de diseño de Material Design de Google. Esta guía proporciona un conjunto de principios y componentes de diseño que ayudan a crear interfaces atractivas, intuitivas y consistentes. Utilizar Material Design garantiza que la aplicación tenga una apariencia moderna y un buen comportamiento en una amplia gama de dispositivos Android.

2.5.4 DataStore

DataStore es una tecnología avanzada en Android diseñada para el almacenamiento seguro y eficiente de preferencias y datos clave-valor. Ha sido introducida como una alternativa moderna a SharedPreferences, mejorando significativamente la forma en que las aplicaciones gestionan sus datos persistentes.

DataStore es seguro, eficiente y funciona asincrónicamente, proporcionando un manejo de datos más robusto y moderno.

2.6 GOOGLE PLAY STORE

Google Play Store es la tienda oficial de aplicaciones para dispositivos Android, administrada por Google. Es la plataforma más grande para la distribución de aplicaciones Android, permitiendo a los desarrolladores llegar a una audiencia global. La tienda ofrece varias funcionalidades clave:

- **Distribución y Monetización:** Los desarrolladores pueden publicar sus aplicaciones y optar por monetizarlas a través de ventas directas, suscripciones o anuncios.
- **Actualizaciones Automáticas:** Los usuarios reciben automáticamente las actualizaciones de las aplicaciones, asegurando que siempre tengan la última versión instalada.
- **Revisión y Seguridad:** Google Play Store realiza revisiones de las aplicaciones para asegurar que cumplan con las políticas de seguridad y privacidad, reduciendo el riesgo de malware y aplicaciones maliciosas.
- **Análisis y Reportes:** Proporciona herramientas de análisis y reportes que permiten a los desarrolladores rastrear el rendimiento de sus aplicaciones y comprender mejor a su audiencia.

2.7 HARDWARE

Para llevar a cabo el desarrollo y las pruebas de la aplicación Android descrita en este trabajo de fin de grado, se utilizó un conjunto de hardware específico que garantizara un entorno eficiente y adecuado tanto para el desarrollo como para la evaluación en diferentes dispositivos. A continuación, se detallan las especificaciones del hardware empleado:

Para todas las etapas del proceso de desarrollo de la aplicación, así como para la ejecución de dispositivos virtuales y pruebas, se utilizó un ordenador portátil con las siguientes características:

- **Procesador:** Intel Core i5 de 10ª generación (por ejemplo, i5-1035G1 @ 1.00GHz, 1.19 GHz)
- **Memoria RAM:** 16.0 GB (DDR4, 2666 MHz)
- **Almacenamiento:** SSD (Solid State Drive) de 512 GB
- **Sistema operativo:** Windows 11 Home 64 bits, versión 23H2, compilación 22631.3737

Este equipo proporciona el poder de procesamiento y la capacidad de memoria suficiente para manejar el entorno de desarrollo Android Studio, realizar compilaciones de aplicaciones de manera eficiente y ejecutar dispositivos virtuales Android con fluidez.

Para validar la aplicación en diferentes configuraciones de hardware y versiones de sistema operativo Android, se utilizaron los siguientes dispositivos:

Smartphone Físico - Xiaomi Redmi Note 9s

- **Procesador:** Qualcomm Snapdragon 720G (2x2.3 GHz Kryo 465 Gold & 6x1.8 GHz Kryo 465 Silver)
- **Memoria RAM:** 4 GB
- **Almacenamiento:** 64 GB
- **Sistema operativo:** Android 13
- **Pantalla:** 6.67”, Resolución 1080x2400 FHD+

Dispositivo Virtual Android (AVD) - Medium Size API 34

Configuración simulada similar a un dispositivo medio estándar asegurando que la aplicación se comporte correctamente en un entorno emulado que refleje las características y limitaciones de dispositivos de tamaño y rendimiento moderados.

- **Procesador y RAM virtual:** Asignados según las especificaciones de Android Studio para un AVD de tamaño medio
- **Sistema operativo emulado:** Android 14
- **Pantalla:** 6.4”, Resolución 1080 x 2400 FHD+

3 ANÁLISIS

En esta fase del proyecto se ha realizado un análisis exhaustivo de la aplicación existente y se han identificado tanto los requisitos funcionales como los no funcionales que deben abordarse para mejorarla significativamente.

3.1 ANÁLISIS DE LA APLICACIÓN EXISTENTE

La aplicación actual ha sido diseñada para la gestión de puntos kilométricos (PK) en obras lineales, ofreciendo funcionalidades básicas como la captura, visualización y gestión de estos puntos.

Está desarrollada en el lenguaje de programación Java y no se implementa ninguna arquitectura de software y el código está poco estructurado.

A continuación, se detalla el funcionamiento de la aplicación a través de las principales pantallas con las que el usuario interactúa:

3.1.1 Pantalla de Inicio (Splash Screen)

La aplicación se inicia con un Splash Screen (Imagen 5) de 3 segundos que muestra el nombre de la aplicación, proporcionando una introducción visual mientras se carga la interfaz principal.

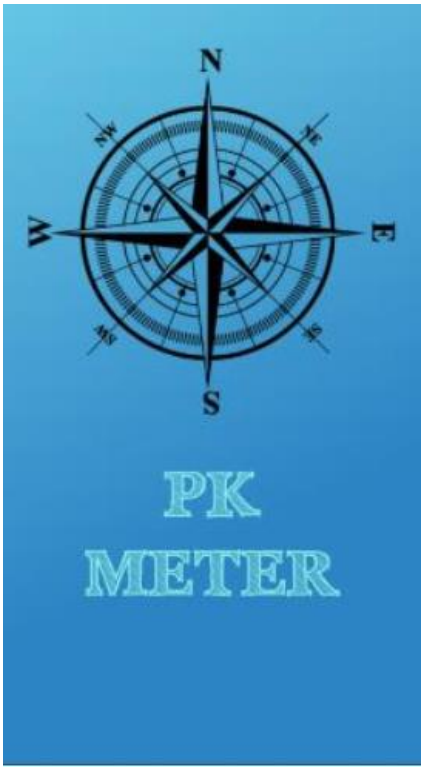


Imagen 5. Splash Screen de la aplicación previa

3.1.2 Pantalla Principal

En la pantalla principal (Imagen 6), el usuario se encuentra con las siguientes opciones:

- **Registrar Marcadores:** Al seleccionar esta opción, se abre el mapa de Google Maps para poder registrar una obra lineal con sus PKs.
- **Encontrar PK más Cercano:** Al elegir esta opción, se muestra un mapa de Google Maps con la ubicación actual del usuario y todos los puntos kilométricos (PK) registrados para poder encontrar el PK más cercano.
- **Consultar Lista de Marcadores:** Al acceder a esta opción, se muestra una lista con todos los marcadores almacenados en la base de datos local.
- **Ajustes:** Desde el icono de engranaje, se puede acceder a la opción de ajustes, donde se pueden configurar aspectos como el relieve del mapa.

Imagen 6. Pantalla Principal de la aplicación previa

3.1.3 Ajustes

La pantalla de ajustes (Imagen 7) permite al usuario configurar varios aspectos de la aplicación para personalizar su funcionamiento. Esta pantalla se divide en tres secciones principales:

Visualización del mapa: El usuario puede seleccionar el modo de visualizar los mapas de Google, específicamente el relieve del propio mapa. Esta configuración afecta cómo se representan las características geográficas y topográficas en el mapa.

Tipo de búsqueda: Se define el tipo de búsqueda que se utilizará para localizar el PK más cercano. El usuario puede elegir entre:

- **Última obra lineal:** Selecciona automáticamente la última obra lineal creada.
- **Manual:** Permite al usuario introducir manualmente el nombre de la obra lineal de referencia para utilizar en la búsqueda.

Precisión de captura: El usuario puede seleccionar el nivel de precisión para la captura de PKs, eligiendo entre las opciones "Baja", "Media" y "Alta".

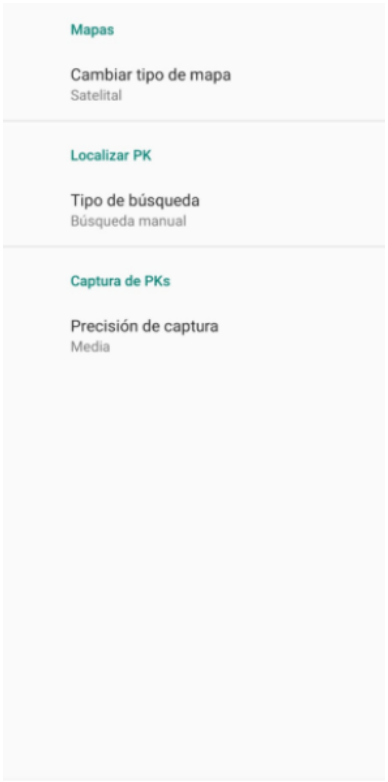


Imagen 7. Pantalla de ajustes de la aplicación previa

3.1.4 Pantalla de Registro de Marcadores

La pantalla de registro (Imagen 8) incluye un mapa de Google Maps con la cámara centrada en España concretamente en Madrid. Además, en la parte superior del mapa hay tres botones con los que el usuario puede iniciar, pausar y guardar la captura de puntos kilométricos (PK).

Cuando se entra en esta pantalla, se solicita al usuario introducir el nombre de la obra lineal antes de iniciar la captura, validando que sea válido (solo caracteres alfanuméricos).

Entonces, se empieza la captura de PKs de una nueva obra lineal registrando el primer marcador en el mapa donde está la posición del usuario. Se van registrando los nuevos PKs cada 20 m.

Si el nombre introducido coincide con uno existente, se ofrece la opción de reanudar la captura previa indicando al usuario situarse a 20m del último marcador y cuando está en la ubicación óptima se empieza la captura de los PKs.

Imagen 8. Pantalla de registro de PKs de la aplicación previa

3.1.5 Pantalla del PK más cercano

La pantalla (Imagen 9) muestra un mapa de Google Maps con la ubicación actual del usuario y todos los puntos kilométricos (PK) registrados si está configurado en el tipo de búsqueda "Última Obra Lineal" o sino en cambio se muestra un diálogo al usuario pidiendo el nombre de la obra lineal.

En la parte superior de la pantalla hay un único botón con el que el usuario puede obtener el PK más cercano llevándolo a una pantalla nueva.

En esta pantalla, se muestra la información acerca de este punto junto con un botón para compartirla.

Imagen 9. Pantalla del PK más cercano de la aplicación previa

3.1.6 Pantalla de lista de PKs

La pantalla (Imagen 10) muestra una lista con todos los PK almacenados en la base de datos local. Cada elemento de la lista ofrece la información del PK y si se hace click en un elemento se abre un diálogo para eliminarlo.

Imagen 10. Pantalla de la lista de PKs de la aplicación previa

3.2 PROBLEMAS O DEFECTOS IDENTIFICADOS

Durante el análisis detallado de la aplicación existente, se han identificado varios problemas y defectos que justifican las mejoras propuestas:

1. **Interfaz de Usuario Obsoleta:** La interfaz actual carece de elementos de diseño modernos, lo que dificulta la navegación y uso eficiente por parte de los usuarios. Además, la estética de la aplicación no está alineada con los estándares actuales de diseño, lo que afecta negativamente la experiencia del usuario.
2. **Limitaciones en la Gestión de PKs:**
    - La aplicación solo permite la eliminación de PKs de uno en uno, lo que resulta poco práctico para usuarios que necesiten realizar una limpieza masiva de datos.
    - No existe una funcionalidad de exportación e importación de PKs, limitando la capacidad de los usuarios para transferir y respaldar datos.
3. **Precisión del GPS:** Actualmente, debido a la inexactitud inherente al GPS, se puede acumular error con cada punto capturado, lo que afecta la exactitud de la obra lineal registrada.
4. **Restricciones en la Continuación de Obras Lineales:** La necesidad de que el usuario se ubique en el siguiente PK después del último registrado para continuar una obra lineal es una limitación operativa que puede optimizarse.
5. **Búsqueda del PK más Cercano Mejorable**: La funcionalidad de buscar el PK más cercano puede mejorarse significativamente. Actualmente, se busca el PK registrado más cercano, lo cual puede no ser la mejor representación de la proximidad real en una obra lineal. Sería mucho más útil calcular el PK más cercano interpolando entre los PK contiguos. Esto implicaría que el punto de cálculo debe ser la interpolación entre las coordenadas de los PKs contiguos, respecto a la proyección de la ubicación del usuario sobre la recta que une los PKs contiguos. Esta mejora proporcionaría una precisión superior y una mejor experiencia de usuario.
6. **Limitaciones en el Guardado de PKs**: Actualmente, la aplicación solo permite guardar los PKs una vez que se ha completado todo el registro. Esto implica que, si ocurre algún fallo durante el proceso de registro, todos los PKs capturados hasta el momento se pierden. Esta limitación puede resultar en una pérdida significativa de datos y tiempo para el usuario, y debería mejorarse permitiendo guardados intermedios para proteger los datos capturados ante cualquier eventualidad.
7. **Falta de Flexibilidad en la Configuración:** Las opciones de configuración actuales no permiten una personalización adecuada de la captura y gestión de PKs según las necesidades específicas de cada usuario. Solamente se pueden capturar los PKs cada 20 m.
8. **Dependencia en Java:** La aplicación está desarrollada en Java, un lenguaje que Google está desaconsejando en favor de Kotlin para el desarrollo de aplicaciones Android, afectando la mantenibilidad y evolución del proyecto.
9. **Código sin Arquitectura Definida:** El código actual no sigue ninguna arquitectura definida, lo que dificulta su mantenimiento y escalabilidad. La falta de una estructura clara puede llevar a problemas de cohesión y acoplamiento, complicando la implementación de nuevas funcionalidades y la corrección de errores.

3.3 Requisitos para la Mejora de la Aplicación

En este apartado, se detallan los requisitos funcionales y no funcionales propuestos para mejorar la aplicación actual, abordando los problemas y defectos identificados en el análisis previo.

3.3.1 Requisitos Funcionales

Exportar los PK:

- **Descripción:** Permitir la exportación de puntos kilométricos (PK) a un formato compatible con otras aplicaciones o sistemas.
- **Problema que Soluciona:** Resuelve la limitación actual de no poder transferir y respaldar datos, mejorando la capacidad de gestión de los PKs.

Importar los PK:

- **Descripción:** Permitir la importación de PK desde archivos externos.
- **Problema que Soluciona:** Facilita la integración de datos externos y la restauración de PKs, abordando la falta de funcionalidad para gestionar datos de forma masiva.

Forzar PK:

- **Descripción:** Implementar una funcionalidad para corregir el error acumulado en la captura de PKs debido a la imprecisión del GPS. Esto permitiría ajustar todos los PKs anteriores desde el último "PK forzado" de manera equidistante respetando la forma de la obra lineal. Si no hay un “PK forzado” anterior se ajustaría desde el primer PK capturado.
- **Problema que Soluciona:** Mitiga el problema de la acumulación de errores en la captura de PKs, mejorando la precisión y exactitud de los datos registrados.

Búsqueda del PK más Cercano Mejorada:

- **Descripción:** Mejorar la funcionalidad de búsqueda del PK más cercano para que se calcule interpolando entre los PK contiguos. El punto de cálculo debe ser la interpolación entre las coordenadas de los PKs contiguos, respecto a la proyección de la ubicación del usuario sobre la recta que une los PKs contiguos.
- **Problema que Soluciona:** Proporciona una precisión superior y una mejor experiencia de usuario al identificar el PK más cercano basado en una interpolación precisa en lugar de simplemente el PK registrado más cercano.

Añadir anotaciones con fotos o textos a PKs:

- **Descripción:** Permitir al usuario añadir una foto o escribir un texto en un PK dado y establecer una alarma que se activaría al pasar por ese PK. El usuario podría añadirlo de manera manual seleccionando el PK o de manera automática en la que lo añadiría al PK más cercano.
- **Mejora:** Introduce una funcionalidad que permite añadir contexto y alertas a los PKs, lo que incrementa la utilidad de la aplicación.

Mejorar la lista de PK:

- **Descripción:** Añadir un filtro por obra lineal y la opción de eliminar todos los PKs a la vez.

- **Problema que Soluciona**: Optimiza la gestión de PKs, permitiendo una limpieza masiva de datos y facilitando la organización y consulta de PKs.

Cambiar la distancia entre PKs:

- **Descripción:** Permitir al usuario modificar la distancia entre cada PK capturado.
- **Problema que Soluciona:** Ofrece mayor flexibilidad en la captura de PKs, adaptándose a las necesidades específicas de diferentes obras lineales.

Continuación de una obra lineal registrada:

- **Descripción:** Permitir que la captura de PKs se inicie cuando el usuario está en el último PK registrado en lugar de tener que ubicarse en el siguiente PK.
- **Problema que Soluciona:** Facilita la continuación de la captura de PKs en una obra lineal ya existente, mejorando la operatividad y eficiencia del proceso.

3.3.2 Requisitos no funcionales

Migración de Java a Kotlin:

- **Descripción:** Migrar la aplicación del lenguaje de programación Java a Kotlin.
- **Problema que Soluciona:** Mejora la mantenibilidad del código y asegura la adherencia a las mejores prácticas recomendadas por Google para el desarrollo de aplicaciones Android.
-

Mejora de la interfaz de usuario de la aplicación:

- **Descripción:** Actualizar el diseño de la interfaz de usuario para hacerlo más moderno e intuitivo.
- **Problema que Soluciona:** Aborda el problema de la interfaz de usuario obsoleta, mejorando la estética y la usabilidad de la aplicación.

Establecer una arquitectura de aplicación Android recomendada:

- **Descripción:** Reestructurar la aplicación utilizando la arquitectura recomendada por Google, dividiendo la aplicación en capas de presentación, dominio y datos.
- **Problema que Soluciona:** Mejora la organización del código, facilitando la modularización y el mantenimiento a largo plazo, y asegurando que la aplicación siga principios de diseño sólidos.

Uso de la librería Room para la gestión de la base de datos:

- **Descripción:** Implementar Room como la biblioteca de persistencia para la gestión de la base de datos.
- **Mejora:** Aumenta la seguridad y mantenibilidad de los datos almacenados, facilitando la interacción con la base de datos y mejorando la integridad de los datos.

Uso de Hilt para la inyección de dependencias:

- **Descripción:** Utilizar Hilt para la inyección de dependencias en la aplicación.
- **Mejora**: Mejora la mantenibilidad y escalabilidad del código, facilitando la gestión de dependencias y asegurando la adherencia a los principios SOLID.

3.4 DIAGRAMA DE CASOS DE USO

Para poder explicar las interacciones del usuario con el sistema con las nuevas mejoras se presenta el diagrama de casos de uso y se detalla cada uno de los casos de uso (Imagen 11).

Imagen 11. Diagrama de Casos de Uso

Caso de Uso 1: Exportar PKs

- **Identificador del Caso de Uso:** CU-01
- **Nombre del Caso de Uso:** Exportar PKs
- **Descripción:** Permitir la exportación de puntos kilométricos (PK) a un archivo JSON compatible con otras aplicaciones o sistemas.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber registrado PKs en la aplicación.
- **Flujo Principal:**
    1. El usuario selecciona la opción "Exportar PKs" en el menú del AppBar.
    2. El sistema genera un archivo JSON con los PKs registrados.
    3. El sistema muestra una opción para guardar el archivo JSON exportado.
- **Postcondiciones:** El archivo JSON con los PKs es generado y está disponible para ser guardado o compartido.

Caso de Uso 2: Importar PKs

- **Identificador del Caso de Uso:** CU-02
- **Nombre del Caso de Uso:** Importar PKs
- **Descripción:** Permitir la importación de PKs desde un archivo JSON.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe tener un archivo JSON de PKs en su dispositivo.
- **Flujo Principal:**
    1. El usuario selecciona la opción "Importar PKs" en el menú del AppBar.
    2. El sistema solicita al usuario que seleccione un archivo JSON desde su dispositivo.
    3. El usuario selecciona el archivo JSON.
    4. El sistema procesa el archivo e importa los PKs a la base de datos.
    5. El sistema confirma que la importación ha sido exitosa actualizando la lista de PKs.
- **Postcondiciones:** Los PKs del archivo JSON importado son añadidos a la base de datos de la aplicación.

Caso de Uso 3: Forzar PK

- **Identificador del Caso de Uso:** CU-03
- **Nombre del Caso de Uso:** Forzar PK
- **Descripción:** Permitir la corrección el error acumulado en la captura de PKs debido a la imprecisión del GPS.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber registrado al menos dos PKs en la aplicación.
- **Flujo Principal:**
    1. El usuario inicia la captura.
    2. Si hay más de dos PKs registrados, el sistema habilita el botón "Forzar PK".
    3. El usuario selecciona la opción "Forzar PK".
    4. El sistema muestra un diálogo solicitando el valor del PK que se quiere forzar.
    5. El usuario introduce el valor del PK y confirma.
    6. El sistema ajusta todos los PKs anteriores desde el último PK forzado de manera equidistante.
    7. El sistema actualiza y muestra los PKs recolocados en el mapa.

- **Flujo Alternativo:**
    1. Si el usuario cancela la operación, los PKs no son ajustados.
- **Postcondiciones:** Los PKs anteriores son ajustados para corregir el error acumulado.

Caso de Uso 4: Añadir Anotaciones con Fotos o Textos a PKs

- **Identificador del Caso de Uso:** CU-04
- **Nombre del Caso de Uso:** Añadir Anotaciones con Fotos o Textos a PKs
- **Descripción:** Permitir al usuario añadir una foto o escribir un texto en un PK dado.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber registrado PKs en la aplicación.
- **Flujo Principal:**
    - **Automatizado:**
        1. El usuario selecciona la opción de "Añadir anotación".
        2. El sistema muestra dos opciones y el usuario elige la opción “Añadir con localización”
        3. El sistema muestra un diálogo solicitando texto y/o imagen para la anotación.
        4. El usuario introduce el texto y/o selecciona la imagen.
        5. El usuario guarda la anotación.
        6. El sistema guarda la anotación y la asocia al PK más cercano.
    - **Manual:**
        1. El usuario selecciona la opción de "Añadir anotación".
        2. El sistema muestra dos opciones y el usuario elige la opción “Introducir PK manualmente”
        3. El sistema muestra un diálogo solicitando texto y/o imagen para la anotación.
        4. El usuario introduce el texto y/o selecciona la imagen.
        5. El usuario guarda la anotación.
        6. El sistema guarda la anotación y la asocia al PK seleccionado.
- **Flujo Alternativo:**
    - Si el usuario no introduce texto ni selecciona imagen, la anotación no se guarda.
- **Postcondiciones:** La anotación con texto y/o imagen es guardada y asociada al PK.

Caso de Uso 5: Configurar Ajustes

- **Identificador del Caso de Uso:** CU-05
- **Nombre del Caso de Uso:** Configurar Ajustes
- **Descripción:** Permitir al usuario modificar los parámetros de la aplicación.
- **Actor(es):** Usuario
- **Flujo Principal:**
    1. El usuario selecciona la opción "Ajustes".
    2. El sistema muestra las opciones de configuración divididas en tres secciones:
        - **Mapas:** Permite cambiar el modo de visualización de los mapas de Google, específicamente el relieve del mapa.
        - **Precisión de Captura de PKs:** Permite al usuario elegir entre cuatro opciones: 10m, 25m, 50m, y 100m.
        - **Precisión del GPS:** Permite al usuario elegir entre "Baja", "Media" y "Alta" precisión.
        - **Alarmas de Anotaciones:** Permite activar o desactivar las alarmas de las anotaciones.
    3. El usuario modifica los parámetros deseados y guarda los cambios.
- **Postcondiciones:** La aplicación guarda los ajustes configurados por el usuario.

Caso de Uso 6: Crear Obra Lineal

- **Identificador del Caso de Uso:** CU-06
- **Nombre del Caso de Uso:** Crear Obra Lineal
- **Descripción:** Permitir al usuario crear sus propias obras lineales para el posterior cálculo del PK más cercano.
- **Actor(es):** Usuario
- **Flujo Principal:**
    1. El usuario selecciona la opción "Registrar PKs".
    2. Se abre el mapa oficial de Google Maps centrado en la ubicación actual del usuario.
    3. En la parte superior se muestran las opciones de "Iniciar", "Pausar" y "Guardar".
    4. El usuario pulsa el botón "Iniciar".
    5. El sistema muestra un diálogo para que el usuario introduzca el nombre de la obra lineal que se va a capturar.
    6. Si el nombre ya existe:
        - Se muestra un Popup preguntando al usuario si desea introducir otro nombre o reanudar la captura de puntos de esa obra lineal.
        - Si se reanuda, se muestran en el mapa todos los puntos guardados respecto a esa obra lineal.
    7. Si el nombre es válido y no existe:
        - Se inicia el algoritmo de captura de PKs.
- **Flujo Alternativo:**
    1. Si el usuario decide continuar con una obra lineal ya creada:
        1. El sistema muestra un aviso pidiendo al usuario que se sitúe en la ubicación del último PK registrado.
        2. El usuario puede aceptar y continuar desde la ubicación del último PK registrado
- **Postcondiciones:** La nueva obra lineal es creada y guardada en la base de datos.

Caso de Uso 7: Consultar Lista de PKs

- **Identificador del Caso de Uso:** CU-07
- **Nombre del Caso de Uso:** Consultar Lista de PKs
- **Descripción:** Permitir al usuario visualizar una lista de todos los PKs guardados en la base de datos local.
- **Actor(es):** Usuario
- **Flujo Principal:**
    1. El usuario selecciona la opción "Lista de PKs "
    2. El sistema muestra una lista de obras lineales con un botón para ver sus PKs y un buscador para filtrar por nombre. El usuario también puede borrar una obra lineal en concreto o borrar toda la lista.
    3. El usuario presiona el botón para ver los PKs de una obra lineal.
    4. El sistema muestra una lista de PKs con la información relevante (título, nombre de la obra lineal, coordenadas).
    5. El usuario puede borrar un PK en concreto.
- **Postcondiciones:** El usuario visualiza la lista de PKs y puede aplicar filtros para una búsqueda más eficiente.

Caso de Uso 8: Calcular PK Más Cercano

- **Identificador del Caso de Uso:** CU-08
- **Nombre del Caso de Uso:** Calcular PK Más Cercano
- **Descripción:** Permitir al usuario ver el mapa con su posición y todos los PKs de la obra lineal asociada, calculando el PK más cercano.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber registrado PKs en la aplicación.
- **Flujo Principal:**
    1. El usuario selecciona la opción "Encontrar PK Más Cercano".
    2. El sistema muestra el mapa con la posición actual del usuario y los PKs de la obra lineal asociada.
    3. El sistema calcula el PK más cercano.
- **Flujo Alternativo:**
    1. Si no hay PKs registrados, el sistema muestra un mensaje indicando que no hay datos para calcular.
- **Postcondiciones:** El PK más cercano es calculado y mostrado.

Caso de Uso 9: Ver PK Más Cercano

- **Identificador del Caso de Uso:** CU-09
- **Nombre del Caso de Uso:** Ver PK Más Cercano
- **Descripción:** Permitir al usuario visualizar la información detallada del PK más cercano calculado.
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber calculado el PK más cercano.
- **Flujo Principal:**
    1. El usuario selecciona la opción "Ver PK Más Cercano".
    2. El sistema muestra la información detallada del PK más cercano (título, nombre de la obra lineal, coordenadas, etc.).
- **Postcondiciones:** El usuario visualiza la información detallada del PK más cercano.

Caso de Uso 10: Compartir Información

- **Identificador del Caso de Uso:** CU-10
- **Nombre del Caso de Uso:** Compartir Información
- **Descripción:** Permitir al usuario compartir la información detallada del PK más cercano a través de diferentes métodos (texto, Bluetooth, WhatsApp, SMS, etc.).
- **Actor(es):** Usuario
- **Precondiciones:** El usuario debe haber calculado el PK más cercano.
- **Flujo Principal:**
    1. El usuario selecciona la opción "Compartir Información" desde la pantalla de detalles del PK más cercano.
    2. El sistema muestra una lista de opciones para compartir la información.
    3. El usuario selecciona el método deseado (por ejemplo, WhatsApp).
    4. El sistema comparte la información del PK según el método seleccionado.
- **Postcondiciones:** La información del PK es compartida según el método seleccionado.

4 DISEÑO

En el contexto del diseño de la aplicación para la gestión de obras lineales, es fundamental asegurar una interfaz de usuario intuitiva, eficiente y estéticamente agradable. El diseño no solo se centra en la apariencia visual, sino también en la usabilidad y la experiencia del usuario durante la navegación por las distintas funcionalidades. A continuación, se detallan los aspectos clave del diseño de la aplicación:

4.1 ARQUITECTURA DE SOFTWARE

Para el desarrollo de la aplicación, se ha seguido una arquitectura en capas moderna, basada en las recomendaciones de Google, con el objetivo de mejorar la calidad, robustez y facilidad de pruebas de la aplicación. La arquitectura se estructura en tres capas principales: la capa de la UI, la capa de datos y la capa de dominio, integrando prácticas recomendadas como la programación reactiva y la inyección de dependencias.

4.1.1 Patrón MVVM

El patrón MVVM (Model-View-ViewModel) es un patrón arquitectónico que se utiliza para separar la lógica de negocio y de presentación de una aplicación, mejorando la mantenibilidad y la capacidad de prueba del código. En el contexto de Android, MVVM se utiliza para estructurar el código de una aplicación de una manera que facilita la separación de responsabilidades.

- **Model (Modelo):** Representa la capa de datos de la aplicación. Puede incluir lógica de negocio, servicios web, bases de datos, etc. Proporciona los datos que la ViewModel necesita.
- **View (Vista):** Es la interfaz de usuario (UI) de la aplicación. En Android, esto incluye las actividades, fragmentos y layouts. La vista observa los cambios en la ViewModel y los refleja en la UI. Debe ser lo más delgada posible, delegando la mayor parte de la lógica a la ViewModel.
- **ViewModel (Modelo de Vista):** Actúa como un intermediario entre el Model y la View. Gestiona la lógica de presentación y procesa los datos del Model para que sean adecuados para la View.

4.1.2 Capa de la UI

La capa de la UI (User Interface) es responsable de mostrar los datos en la pantalla y de interactuar con el usuario. Está compuesta por elementos de la UI y contenedores de estado.

- **Elementos de la UI:** Estos son los componentes visuales que renderizan los datos en la pantalla.
- **Contenedores de Estado (ViewModel):** Los ViewModels en esta capa utilizan MutableStateFlow y StateFlow para mantener y exponer los datos de manera reactiva. Los ViewModels también gestionan la lógica de presentación y se comunican con la capa de dominio o directamente con la capa de datos para obtener los datos necesarios.
- **Interacción del Usuario:** Cuando el usuario interactúa con la UI, como presionar un botón, los eventos son manejados por el ViewModel, el cual actualiza el estado de la UI de manera reactiva.

4.1.3 Capa de Datos

La capa de datos contiene la lógica empresarial de la aplicación y es responsable de manejar los datos que se muestran en la UI.

- **Repositorios:** Los repositorios actúan como una fuente única de verdad para cada tipo de datos en la aplicación. Centralizan los cambios, resuelven conflictos entre múltiples fuentes de datos y abstraen la lógica de acceso a los datos del resto de la aplicación.
- **Fuentes de Datos:** Las clases de fuente de datos trabajan con fuentes específicas como bases de datos locales (Room), servicios de red o archivos. Cada clase de fuente de datos se encarga de interactuar con su respectiva fuente y proporcionar los datos al repositorio.

4.1.4 Capa de Dominio

La capa de dominio se encuentra entre la capa de la UI y la capa de datos, y es utilizada para encapsular la lógica empresarial compleja y mejorar la reutilización del código. En este proyecto, también se ha utilizado para transformar modelos de datos en modelos de dominio, facilitando así la gestión y manipulación de los datos en las capas superiores.

- **Casos de Uso (Use Cases):** Los casos de uso encapsulan funcionalidades específicas y son utilizados por los ViewModels para ejecutar operaciones de negocio. Por ejemplo, un caso de uso podría es “ConvertLatLongToUTMUseCase”, que se encarga de convertir coordenadas geográficas (latitud y longitud) a coordenadas UTM (Universal Transverse Mercator).
- **Transformación de Modelos:** Esta capa se encarga de transformar los datos provenientes de las fuentes de datos (modelos de datos) en modelos de dominio, los cuales son más adecuados para ser utilizados en la lógica de la aplicación y la UI. Esta transformación permite una mayor flexibilidad y control sobre los datos que se manejan en la aplicación.

4.1.5 Gestión de Dependencias

Para la gestión de dependencias, utilizamos la biblioteca Hilt de Dagger. Hilt nos permite inyectar dependencias en nuestras clases de manera automática, lo que facilita la creación de objetos y asegura que todas las dependencias estén disponibles cuando se necesiten.

4.2 INTERFAZ DE USUARIO

En esta sección, se presentan los wireframes de las distintas pantallas de la aplicación, ilustrando cómo se muestra y se navega a través de la interfaz de usuario. Un wireframe es un esquema visual que representa la estructura básica y los elementos de una página o pantalla de una aplicación, permitiendo planificar la disposición y funcionalidad antes de pasar al diseño detallado y al desarrollo.

Al abrir la aplicación, después de que se muestre la pantalla de inicio (*Splash Screen*), se muestra la pantalla principal (Imagen 12) donde el usuario puede elegir las distintas funcionalidades como se muestra a continuación.

Imagen 12. Wireframe de la pantalla principal

Si el usuario aprieta el botón de ajustes, se muestra la siguiente pantalla (Imagen 13) con la opción de configurar los distintos parámetros como el tipo de mapa o la precisión a la hora de la captura de Pks:

Imagen 13. Wireframe de la pantalla de ajustes

Si el usuario selecciona Abrir trazado, se va en la pantalla (Imagen 14) que se muestra a continuación. En esta pantalla se puede visualizar y gestionar las distintas obras lineales. Desde de aquí, se puede exportar e importar y eliminar todas las obras lineales desde el menú que está en la parte superior. Además, hay un buscador en el que se introduce la obra lineal que hace que la lista se filtre. Cada elemento de la lista tiene dos botones, uno para abrir la lista de PKs que contiene la obra lineal y el otro para borrar esa obra lineal en concreto.

Imagen 14. Wireframe de la lista de obras lineales

Si el usuario hace click en el elemento de la lista se llevará a la siguiente pantalla (Imagen 15) donde el usuario podrá buscar el PK más cercano o añadir una anotación.

Imagen 15. Wireframe de la pantalla de la obra lineal

A continuación, se muestra la pantalla de la lista de PKs (Imagen 16) donde en la parte superior hay un menú a través del cual se puede reajustar el PK inicial. Cada elemento de la lista tiene la información acerca del PK con un botón otro para eliminarlo.

Imagen 16. Wireframe de la lista de PKs

A continuación, se muestran las pantallas de Crear Trazado y Hallar PK:

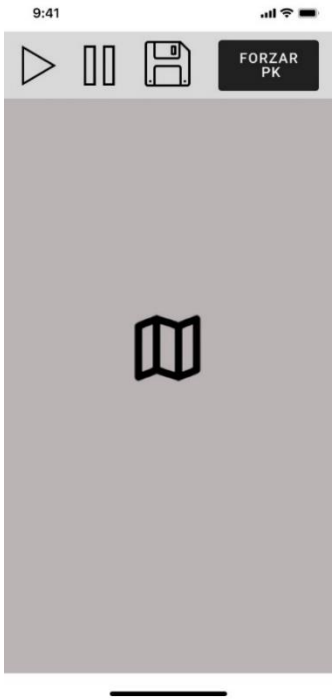


Imagen 17. Wireframe de la pantalla de captura de PKs

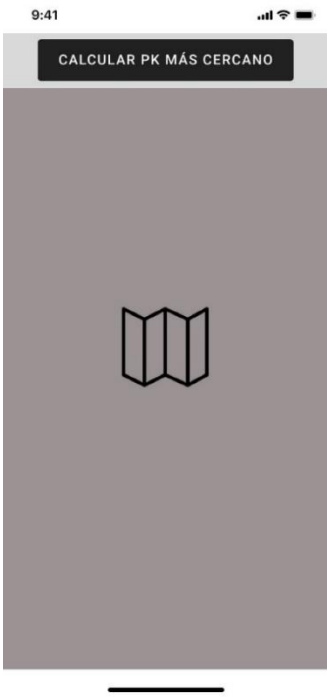


Imagen 18. Wireframe de la pantalla del buscar el PK más cercano

Como se aprecia en la imagen de la izquierda (Imagen 17), la pantalla está dominada por el mapa oficial de Google Maps, y en la parte superior se encuentran los botones de Iniciar, Pausar, Guardar y Forzar PK, que permiten interactuar durante la captura de los PK's. En el wireframe de la derecha (Imagen 18), nuevamente se destaca el mapa de Google Maps, acompañado por el botón Calcular PK más cercano, que nos dirige a la siguiente pantalla y su correspondiente wireframe:

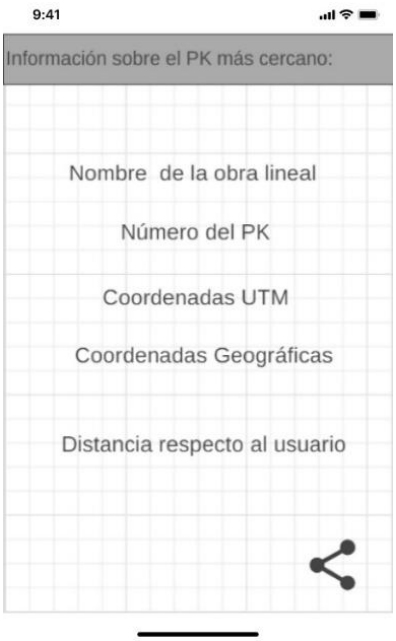


Imagen 19. Wireframe de la pantalla de información del PK

En esta pantalla se presenta la información más importante sobre el PK más cercano al usuario, como sus coordenadas, el número de identificación del PK y la distancia en metros hasta el usuario. Además, con el botón de Compartir ubicado en la esquina inferior derecha, podemos enviar esta información a través de Bluetooth, WhatsApp, SMS, y otras opciones.

Y, por último, si el usuario selecciona la opción de “Añadir anotación” el usuario elige si lo quiere hacer de forma manual o automática y una elegido se muestra la siguiente pantalla (Imagen 20) en la que se muestra el PK, dos iconos para subir la imagen mediante la cámara o la galería. Además, hay un cuadro de texto para escribir el texto de la anotación y en la parte inferior un espacio para mostrar la imagen que se quiere añadir como anotación.

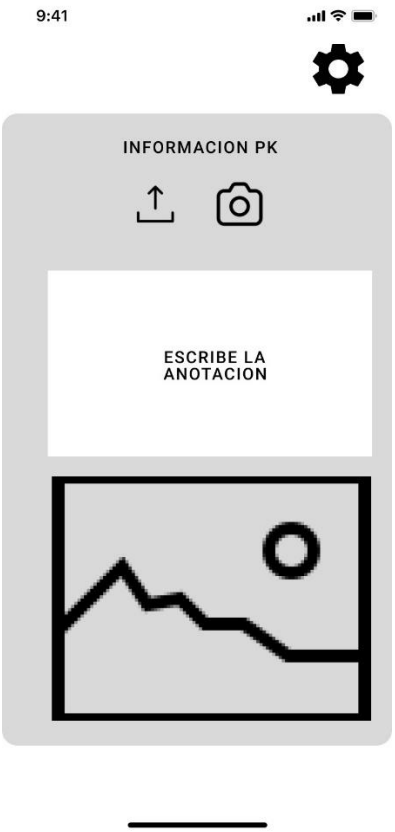


Imagen 20. Wireframe de la pantalla de añadir anotación

5 DESARROLLO

En este capítulo se detalla el proceso seguido para el desarrollo e implementación de las nuevas funcionalidades y mejoras tecnológicas en la aplicación. La transición de Java a Kotlin, la adopción de una arquitectura moderna recomendada por Google y la implementación de componentes avanzados como Room y Hilt, fueron decisiones clave que permitieron optimizar el rendimiento y la mantenibilidad del proyecto. Además, se describen las prácticas empleadas para la migración del código, la estructuración modular de la aplicación, la incorporación de nuevas características.

Primero, se presentará el proceso de migración de Java a Kotlin y la implementación de la nueva arquitectura, explicando los beneficios obtenidos y los retos superados. Posteriormente, se procederá a detallar las mejoras introducidas en cada pantalla de la aplicación, siguiendo el flujo natural del usuario. Esta organización permite explicar de manera cómoda y clara cómo cada cambio contribuye a una experiencia de usuario más intuitiva y eficiente, facilitando la comprensión del desarrollo integral del proyecto.

5.1 MIGRACIÓN DE JAVA A KOTLIN

La migración del código de Java a Kotlin fue una de las primeras y más importantes etapas del proyecto. Kotlin, el lenguaje oficialmente recomendado por Google para el desarrollo de aplicaciones Android, ofrece varias ventajas sobre Java, incluyendo una sintaxis más concisa, mayor seguridad y compatibilidad con la programación funcional. Este apartado describe el proceso de migración, los desafíos enfrentados y los beneficios obtenidos.

La decisión de migrar de Java a Kotlin se basó en varios factores clave:

- **Sintaxis Concisa y Clara:** Kotlin reduce significativamente la cantidad de código repetitivo y extenso, conocido como "boilerplate", lo que facilita la lectura y el mantenimiento del código. El código "boilerplate" se refiere a las secciones de código que se tienen que incluir en muchos lugares con poca o ninguna modificación, como getters y setters, inicializaciones y manejadores de excepciones en Java.
- **Seguridad:** Kotlin ofrece características como la nulabilidad explícita, que ayuda a prevenir errores comunes en tiempo de ejecución.
- **Compatibilidad:** Kotlin es 100% interoperable con Java, lo que permite integrar gradualmente el nuevo código sin necesidad de reescribir completamente la aplicación desde cero.
- **Soporte y Comunidad:** Kotlin cuenta con un fuerte respaldo de Google y una comunidad en crecimiento, lo que asegura un soporte continuo y una amplia disponibilidad de recursos y bibliotecas.

El proceso de migración se realizó en varias fases para asegurar una transición suave y minimizar el impacto en el desarrollo continuo de la aplicación.

Para la migración se ha utilizado la herramienta de conversión integrada en Android Studio para convertir automáticamente el código Java a Kotlin. Esta herramienta facilita la migración inicial al transformar las estructuras de Java en su equivalente en Kotlin.

Después de la conversión automática, se ha realizado una revisión exhaustiva del código convertido. Se optimizaron las partes del código que no se tradujeron de manera eficiente y se aplicaron mejoras para aprovechar las características avanzadas de Kotlin.

5.2 ARQUITECTURA

La implementación de una nueva arquitectura por capas siguiendo el patrón MVVM (Model-View-ViewModel) fue un paso crucial para mejorar la mantenibilidad, escalabilidad y eficiencia de la aplicación. Esta arquitectura, recomendada por Google, incluye el uso de Room para la persistencia de datos y Hilt para la inyección de dependencias. Este apartado describe la motivación para esta nueva estructura y los componentes clave.

La decisión de adoptar una nueva arquitectura se basó principalmente en los siguientes factores:

- **Mantenibilidad:** La arquitectura por capas permite aislar los cambios y facilita la detección y corrección de errores.
- **Escalabilidad:** La separación de responsabilidades facilita la adición de nuevas funcionalidades sin afectar las partes existentes de la aplicación.

La nueva arquitectura se basa en varios componentes clave que trabajan en conjunto para proporcionar una base robusta y flexible:

5.2.1 Room

Room es una biblioteca de persistencia de datos que proporciona una capa de abstracción sobre SQLite. Facilita las operaciones de base de datos y garantiza la seguridad de tipos en las consultas SQL.

Se definieron entidades, DAOs (Data Access Objects) y la base de datos, permitiendo realizar operaciones de inserción, actualización, eliminación y consulta de manera eficiente de los PKs.

Se creó un contrato de la base de datos (Imagen 21) para definir de manera clara y centralizada el esquema de la base de datos, incluyendo el nombre de las tablas y las columnas. Esto asegura la coherencia y facilita las modificaciones futuras.

```
HSINGH
object PKMarkersTable {
    const val TABLE_NAME = "Markers"
    const val COLUMN_ID = "id"
    const val COLUMN_PK = "pk"
    const val COLUMN_LATITUDE = "latitude"
    const val COLUMN_LONGITUDE = "longitude"
    const val COLUMN_LINEAR_BUILDING_WORK = "linearBuildingWork"
    const val COLUMN_TEXT = "text"
    const val COLUMN_IMAGE_URI = "imageUri"
}
```

Imagen 21. Contrato de la base de datos

La estructura de la tabla de los PKs es la siguiente:

- DATABASE_NAME: Nombre de la base de datos SQLite ("marker.db" en este caso).
- PKMarkersTable: Objeto que contiene los nombres de las columnas de la tabla Markers.
- TABLE_NAME: Es el nombre de la tabla en la base de datos cuyo nombre es “Markers”.
- COLUMN_ID: Es la columna principal en la que se guarda el identificador único de cada PK que en este caso es una cadena de texto formada por el PK y su obra lineal.
- COLUMN_PK: Es la columna que contiene el PK.
- COLUMN_LATITUDE/LONGITUDE: Estas columnas contienen la latitud y la longitud respectivamente.
- COLUMN_LINEAR_BUILDING_WORK: Esta columna contiene el nombre de la obra lineal al que pertenece el PK.
- COLUMN_TEXT: Esta columna contiene el texto que añade en la anotación el usuario.
- COLUMN_IMAGE_URI: En esta columna se guarda la dirección de la imagen que el usuario añade en una anotación.

Se creó la entidad PKMarkerDto (Imagen 22) para representar los puntos kilométricos (PK) en la base de datos. Esta entidad define la estructura de la tabla y las columnas que la componen.

```
HSINGH
@Entity(tableName = PKMarkerContract.PKMarkersTable.TABLE_NAME)

data class PKMarkerDto (
    @PrimaryKey @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_ID) val id: String,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_PK) val pk: Double,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_LATITUDE) val latitude: String,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_LONGITUDE) val longitude: String,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_LINEAR_BUILDING_WORK) val linearBuildingWork: String,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_TEXT) val text: String?,
    @ColumnInfo(name = PKMarkerContract.PKMarkersTable.COLUMN_IMAGE_URI) val imageUri: String?,
)
```

Imagen 22. PKMarkerDto

A continuación, se muestra el DAO (Imagen 23) donde se establecen todas las operaciones. El DAO es una interfaz que define los métodos para interactuar con la base de datos. Permite realizar operaciones como inserciones, actualizaciones, eliminaciones y consultas.

```
HSINGH *
@Dao
interface PKMarkerDao {
    HSINGH
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun addPKMarker(pkMarker: PKMarkerDto)
    HSINGH
    @Delete
    suspend fun deletePKMarker(pkMarker: PKMarkerDto)
    HSINGH
    @Query("SELECT * FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME}")
    fun getAllPKMarkers(): Flow<List<PKMarkerDto>>
    HSINGH
    @Query("SELECT * FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME} WHERE ${PKMarkerContract.PKMarkersTable.COLUMN_ID} = :id")
    fun getPKMarkerById(id: Double): Flow<PKMarkerDto?>
    HSINGH
    @Query("SELECT * FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME} WHERE ${PKMarkerContract.PKMarkersTable.COLUMN_LINEAR_BUILDING_WORK} = :linearBuildingWork")
    fun getPKMarkersByLinearBuildingWork(linearBuildingWork: String): Flow<List<PKMarkerDto>>
    HSINGH
    @Query("SELECT DISTINCT ${PKMarkerContract.PKMarkersTable.COLUMN_LINEAR_BUILDING_WORK} FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME}")
    fun getAllLinearBuildingWorks(): Flow<List<String>>
    HSINGH
    @Query("DELETE FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME}")
    suspend fun deleteAllPKMarker()
    HSINGH
    @Query("DELETE FROM ${PKMarkerContract.PKMarkersTable.TABLE_NAME} WHERE ${PKMarkerContract.PKMarkersTable.COLUMN_LINEAR_BUILDING_WORK} = :linearBuildingWork")
    suspend fun deletePKMarkersByLinearBuildingWork(linearBuildingWork: String)
    HSINGH
    @Update
    suspend fun updatePKMarker(pkMarker : PKMarkerDto)
}
```

Imagen 23. PKMarkerDao

5.2.2 Hilt

Hilt es una biblioteca de inyección de dependencias que simplifica la gestión de dependencias en la aplicación. Proporciona un contenedor de dependencias que puede inyectar objetos en los lugares necesarios.

Inyección de ViewModel

Hilt se ha utilizado para inyectar los ViewModel en las actividades y fragmentos (Imagen 24). Esto permite una gestión eficiente y segura de la instancia de los ViewModel, asegurando que se comparte la misma instancia cuando es necesario.

Se utilizó la anotación `@HiltViewModel` para definir los ViewModel, y las anotaciones `@Inject` para las dependencias dentro de los ViewModel. En las actividades y fragmentos, los ViewModel se inyectan utilizando la propiedad `by viewModels()`.

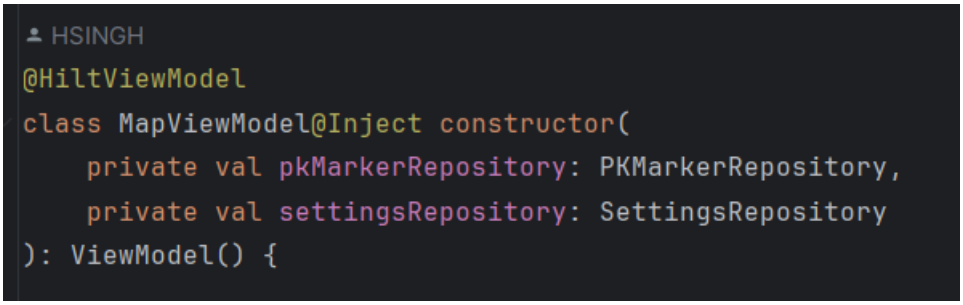


Imagen 24. Uso de Hilt para la inyección de ViewModels

Inyección de Repositorios y DataSource

Los repositorios, que actúan como intermediarios entre la capa de datos y la capa de dominio, también se han inyectado utilizando Hilt. Esto asegura que las dependencias necesarias estén disponibles y configuradas correctamente.

Los datasources representan las diferentes fuentes de datos que la aplicación puede utilizar, como bases de datos locales o servicios remotos. Hilt se encarga de proporcionar estas fuentes de datos a los repositorios.

Los repositorios y datasources se proporcionan a través de módulos de Hilt, utilizando la anotación @Module y @Provides.

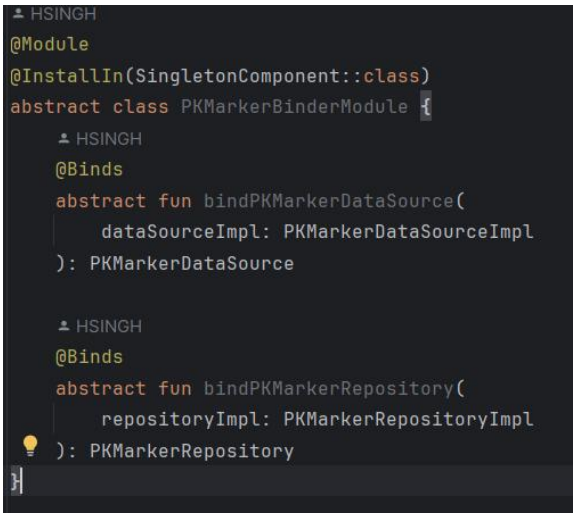


Imagen 25. Uso de Hilt para inyección de repositorios y datasources

Inyección de Room

La base de datos y los DAO se configuran en módulos de Hilt, asegurando que se proporcione una única instancia de la base de datos y sus componentes relacionados a lo largo de la aplicación.

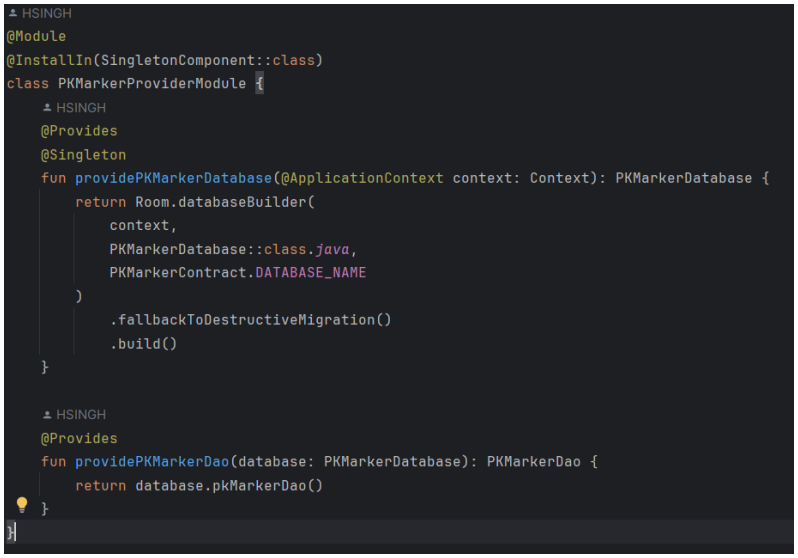


Imagen 26. Uso de Hilt para la inyección de la base de datos

5.2.3 Patrón MVVM (Model-View-ViewModel)

El patrón MVVM (Model-View-ViewModel) es una arquitectura de software que separa la lógica de presentación de la lógica de negocio y los datos, facilitando la prueba, el mantenimiento y la escalabilidad de la aplicación. Este patrón es ampliamente utilizado en el desarrollo de aplicaciones Android debido a sus beneficios.

Componentes de MVVM

1. **Model**: La capa de Model maneja los datos de la aplicación y las reglas de negocio. Interactúa con la capa de datos (por ejemplo, bases de datos locales y servicios remotos) y proporciona datos a la capa de ViewModel.
   En este proyecto, el Model incluye entidades, repositorios y datasources. Los repositorios actúan como intermediarios entre los datasources y la capa de ViewModel, garantizando que los datos se obtengan de manera eficiente y segura.
2. **View:** La capa de View representa la interfaz de usuario y se encarga de mostrar los datos proporcionados por el ViewModel. Observa los cambios en los datos y actualiza la interfaz de usuario en consecuencia.
   En este proyecto, la View está compuesta por actividades y fragmentos que utilizan Data Binding y StateFlow para observar los datos del ViewModel y actualizar la interfaz de usuario automáticamente.
3. **ViewModel**: La capa de ViewModel actúa como intermediario entre la View y el Model. Gestiona la lógica de presentación y proporciona datos a la View. Utiliza MutableStateFlow para observar los cambios en los datos y notificarlos a la View de manera reactiva y eficiente.
   En este proyecto, el ViewModel se implementa utilizando MutableStateFlow se emplea para exponer datos observables a la View, permitiendo actualizaciones fluidas y reactivas en la interfaz de usuario.

5.2.4 Estructura del Proyecto

La estructura del proyecto se organizó en varias capas para asegurar una separación clara de responsabilidades:

Capa de Presentación

Contiene las actividades y fragmentos que conforman la interfaz de usuario. Cada pantalla se asocia con un ViewModel, lo que permite una gestión eficiente de los datos y la lógica de presentación.

En la capa de presentación, se han implementado las interfaces de usuario utilizando actividades y fragmentos que interactúan con los ViewModels correspondientes. Los ViewModels utilizan MutableStateFlow para exponer datos observables y gestionar la lógica de presentación de manera reactiva.

Capa de Dominio

Incluye las entidades y casos de uso que encapsulan las reglas de negocio. Las entidades representan los objetos principales de dominio y los casos de uso definen las operaciones permitidas en esos objetos.

Además, se ha implementado un caso de uso centralizado que encapsula la lógica de negocio compartida entre diferentes partes de la aplicación que se trata de convertir coordenadas del

sistema LatLong a UTM. Este caso de uso se ha diseñado para ser reutilizado por múltiples ViewModels, promoviendo la coherencia y reusabilidad del código.

Capa de Datos

La capa de datos se encarga de la persistencia y la gestión de datos, utilizando tecnologías como Room y DataStore para interactuar con la base de datos local y manejar preferencias de la aplicación, respectivamente.

- **Room (DataSource para SQLite):**
    - Se ha utilizado Room como biblioteca de persistencia de datos que facilita la interacción con una base de datos SQLite local. En esta capa, se definen y configuran los DAOs (Data Access Objects) que proporcionan métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos.
    - Se implementan Entities como PKMarkerDto que representan las tablas de la base de datos, modelando los datos de acuerdo con las necesidades de la aplicación.
    - Los Repositories actúan como abstracciones sobre los DAOs, proporcionando métodos para recuperar y manipular datos desde y hacia la base de datos.
- **DataStore (DataSource para preferencias):**
    - DataStore se utiliza para manejar preferencias de la aplicación de manera segura y eficiente, reemplazando la antigua API de SharedPreferences.
    - Se definen y configuran Preferences que representan las claves y tipos de datos utilizados para almacenar preferencias específicas de la aplicación.
    - Los repositorios asociados a DataStore proporcionan métodos para leer y escribir datos de preferencias, asegurando la persistencia y la recuperación de configuraciones y ajustes de usuario de manera confiable.

5.2.5 Beneficios de la nueva Arquitectura

La adopción de esta nueva arquitectura trajo múltiples beneficios al proyecto:

- **Modularidad:** La clara separación de responsabilidades facilita la localización y resolución de problemas, así como la adición de nuevas funcionalidades.
- **Reutilización de Código**: Componentes bien definidos pueden ser reutilizados en diferentes partes de la aplicación, reduciendo la duplicación de código.
- **Facilidad de Pruebas:** La arquitectura modular permite probar componentes individuales de manera aislada, asegurando su correcto funcionamiento antes de integrarlos con otros módulos.
- **Mantenibilidad y Escalabilidad:** La estructura clara y modular facilita el mantenimiento del código y la implementación de nuevas características sin afectar el sistema global.

5.3 MEJORAS APLICADAS

Para abordar las mejoras necesarias en la aplicación, se explorará cada una de las pantallas principales y funcionalidades críticas. En algunas funcionalidades clave, se proporcionará el código para facilitar un mejor entendimiento de las propuestas de mejora.

5.3.1 Pantalla Principal

En la pantalla principal (Imagen 27), podemos observar que se ha cambiado la paleta de colores a un morado y blanco para que quede mejor estéticamente. Además, se ha establecido un diseño más minimalista.

Desde esta pantalla, el usuario puede ir a los ajustes o elegir entre iniciar una captura de PKs o gestionar las capturas previas.

Imagen 27. Pantalla principal

5.3.2 Ajustes

En la pantalla de ajustes (Imagen 28) tenemos la gestión de las preferencias de la aplicación. Como se puede observar ha añadido la gestión de la precisión de la captura de PKs donde el usuario puede seleccionar entre: 10, 25, 50 y 100 m. Además, se ha añadido una opción para activar o desactivar la alarma de las anotaciones.

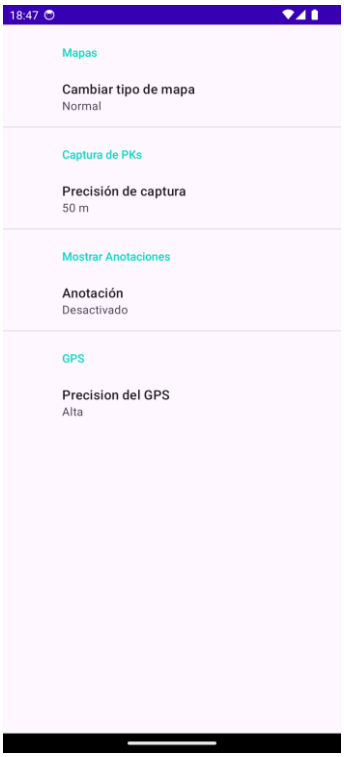


Imagen 28. Pantalla de ajustes

5.3.3 Registro de PKs

En la pantalla del registro de PKs (Imagen 29) se ha añadido un AppBar donde aparece un menú en el que el usuario puede reajustar el PK inicial adecuando así el resto de PK y forzar PK.

Además, se ha hecho que cuando se haga click en el botón de guardar no desaparezcan los tres botones para que el usuario pueda guardar las veces que quiera los PKs capturados.

Y, por último, se mejorado la forma de continuar una captura. El usuario en lugar de situarse a 20 m del último PK ahora se tiene que situarse cerca de 5 m ya que se tiene en cuenta el error que tiene el GPS por lo que se deja ese margen.

Imagen 29. Pantalla de registro de PKs

Forzar PK

La función de "forzar PK" en la aplicación tiene como objetivo corregir el error acumulado en la captura de puntos kilométricos (PK) debido a la imprecisión del GPS. Esto se logra obligando a que el punto en el que se encuentra el usuario tenga un PK específico fijo, conocido por algún motivo (como información previamente registrada o un punto de referencia exacto). Al hacer esto, se recalculan y ajustan todos los PKs anteriores hasta el último "PK forzado", asegurando que la obra lineal registrada mantenga una distribución equidistante y precisa.

```
282     private fun calculateDistance(
283         start: LatLng,
284         end: LatLng
285     ): Double {
286         val earthRadius = 6371000.0 //EN METROS
287         val dLat = Math.toRadians(end.latitude - start.latitude)
288         val dLng = Math.toRadians(end.longitude - start.longitude)
289         val a = sin( x: dLat / 2) * sin( x: dLat / 2) + cos(Math.toRadians(start.latitude)) * cos(
290             Math.toRadians(
291                 end.latitude
292             )
293         ) * sin( x: dLng / 2) * sin( x: dLng / 2)
294         val c =
295             2 * atan2(sqrt(a), sqrt( x: 1 - a))
296         return earthRadius * c
297     }
```

Imagen 30. Función calculateDistance

calculateDistance(start: LatLng, end: LatLng): Esta función (Imagen 30) calcula la distancia entre dos puntos geográficos (dadas sus coordenadas de latitud y longitud) utilizando la

fórmula del Haversine[5]. Esta fórmula es útil para calcular distancias en la superficie de la Tierra.

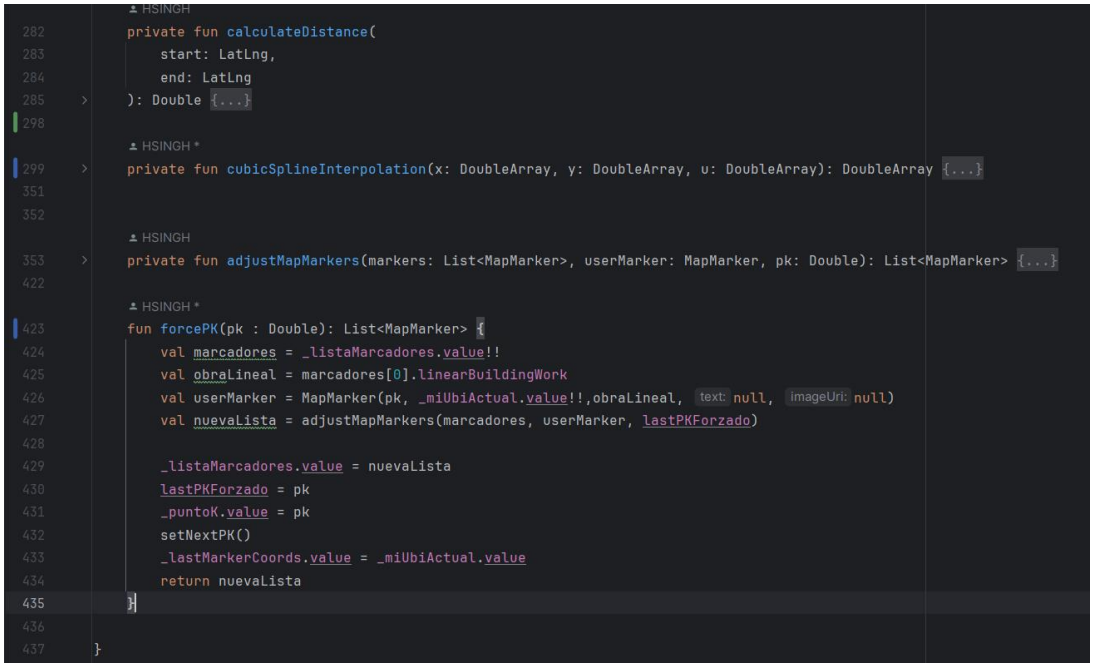
```
private fun cubicSplineInterpolation(x: DoubleArray, y: DoubleArray, u: DoubleArray): DoubleArray {
    val n = x.size
    val a = y.copyOf()
    val b = DoubleArray(n)
    val d = DoubleArray(n)
    val h = DoubleArray( size: n - 1)
    val alpha = DoubleArray( size: n - 1)
    for (i in 0 ≤ until < n - 1) {
        h[i] = x[i + 1] - x[i]
    }
    for (i in 1 ≤ until < n - 1) {
        alpha[i] = (3 / h[i] * (a[i + 1] - a[i])) - (3 / h[i - 1] * (a[i] - a[i - 1]))
    }
    val l = DoubleArray(n)
    val mu = DoubleArray(n)
    val z = DoubleArray(n)
    l[0] = 1.0
    mu[0] = 0.0
    z[0] = 0.0
    for (i in 1 ≤ until < n - 1) {
        l[i] = 2 * (x[i + 1] - x[i - 1]) - h[i - 1] * mu[i - 1]
        mu[i] = h[i] / l[i]
        z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]
    }
    l[n - 1] = 1.0
    z[n - 1] = 0.0
    val c = DoubleArray(n)
    c[n - 1] = 0.0
    for (j in n - 2 ≥ downTo ≥ 0) {
        c[j] = z[j] - mu[j] * c[j + 1]
        b[j] = (a[j + 1] - a[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
        d[j] = (c[j + 1] - c[j]) / (3 * h[j])
    }
    val interpolatedY = DoubleArray(u.size)
    for (i in u.indices) {
        val xVal = u[i]
        var idx = n - 2
        for (j in 0 ≤ until < n - 1) {
            if (xVal <= x[j + 1]) {
                idx = j
                break
            }
        }
        val deltaX = xVal - x[idx]
        interpolatedY[i] = a[idx] + b[idx] * deltaX + c[idx] * deltaX * deltaX + d[idx] * deltaX * deltaX * deltaX
    }
    return interpolatedY
```

Imagen 31. Función cubicSplineInterpolation

cubicSplineInterpolation(x: DoubleArray, y: DoubleArray, u: DoubleArray): Esta función (Imagen 31) realiza una interpolación cúbica de spline. En este caso, se utiliza para interpolar nuevas coordenadas de latitud y longitud para los marcadores en la ruta.
La interpolación cúbica spline[9] es una técnica matemática utilizada para construir una función suave que pase a través de un conjunto de puntos de datos, en este caso, los puntos kilométricos (PK) registrados en una obra lineal. Este método es especialmente útil cuando se desea obtener una función que no solo se ajuste a los puntos dados, sino que también sea suave y continua, evitando fluctuaciones bruscas.

adjustMapMarkers(markers: List<MapMarker>, userMarker: MapMarker, pk: Double): Esta función (Imagen 32) ajusta los marcadores en la ruta. Primero, verifica si el marcador del

usuario ya está en la lista de marcadores. Si no es así, lo añade. Luego, ordena los marcadores por PK y filtra los marcadores que están entre el marcador inicial y el marcador del usuario. Calcula las distancias acumuladas desde el marcador inicial y ajusta estas distancias para que sean equidistantes. Luego, interpola las nuevas coordenadas de latitud y longitud para los marcadores y crea nuevos marcadores ajustados.



```
    HSINGH
    private fun calculateDistance(
        start: LatLng,
        end: LatLng
    ): Double {...}

    HSINGH *
    private fun cubicSplineInterpolation(x: DoubleArray, y: DoubleArray, u: DoubleArray): DoubleArray {...}


    HSINGH
    private fun adjustMapMarkers(markers: List<MapMarker>, userMarker: MapMarker, pk: Double): List<MapMarker> {...}

    HSINGH *
    fun forcePK(pk : Double): List<MapMarker> {
        val marcadores = _listaMarcadores.value!!
        val obraLineal = marcadores[0].linearBuildingWork
        val userMarker = MapMarker(pk, _miUbiActual.value!!,obraLineal, text: null, imageUri: null)
        val nuevaLista = adjustMapMarkers(marcadores, userMarker, lastPKForzado)

        _listaMarcadores.value = nuevaLista
        lastPKForzado = pk
        _puntoK.value = pk
        setNextPK()
        _lastMarkerCoords.value = _miUbiActual.value
        return nuevaLista
    }

}
```

Imagen 32. Código de la funcionalidad de forzar PK

forcePK(pk : Double): Esta es la función principal (Imagen 32) que utiliza las funciones anteriores para forzar un PK. Toma un PK como entrada y ajusta los marcadores en la ruta para que el PK del marcador del usuario sea el PK dado. Luego, actualiza la lista de marcadores y establece el último PK forzado al PK dado.

5.3.4 Lista de obras lineales

Se accede a esta pantalla (Imagen 33) al dar click en el botón “Abrir Trazado”. Aquí el usuario puede exportar e importar todas las obras lineales y eliminar todos PKs accediendo al menú del AppBar. Además, puede eliminar una obra lineal concreta al darle al icono de borrar y acceder a la lista de PKs de esta con el icono de editar. Si hay muchas obras lineales se puede buscar una en concreto por el nombre así mejorando la usabilidad para el usuario.

Imagen 33. Pantalla de lista de obras lineales

5.3.5 Lista de PKs

En esta pantalla (Imagen 34), se puede eliminar un determinado PK y además en el menú tiene la opción de reajustar el PK inicial adecuando así el resto de PK.

Imagen 34. Pantalla de lista de PKs

5.3.6 Opciones en la obra lineal

Si el usuario hace click a un elemento de la lista de obras lineal se navega a la siguiente pantalla (Imagen 35) en la que se puede añadir una anotación o hallar el PK cercano.

Imagen 35. Pantalla de opciones de la obra lineal

5.3.7 Añadir anotación

Con esta opción, el usuario puede elegir dos formas de añadir una anotación que puede incluir tanto una imagen como un texto. La anotación puede ser asociada de manera automática al PK más cercano a la ubicación del usuario o de forma manual a un PK específico.

Forma Automática

Si el usuario selecciona esta opción, se crea un nuevo marcador PK en la ubicación actual del usuario, al cual se va a asociar la anotación.

El siguiente código (Imagen 36) describe el proceso para añadir una anotación automáticamente a un Punto Kilométrico (PK) más cercano a la ubicación actual del usuario. Se calcula interpolando entre los PK contiguos. El PK calculado es la interpolación entre las coordenadas de los PKs contiguos, respecto a la proyección de la ubicación del usuario sobre la recta que une los PKs contiguos.

La función findInterpolatedMarkerByLocation se utiliza para encontrar el nuevo marcador basado en la ubicación actual del usuario. Primero, ordena los marcadores existentes por su distancia a la ubicación del usuario. Luego, encuentra los dos marcadores más cercanos y proyecta la ubicación del usuario sobre la línea que los une con la función projectPointOnLine. Usando esta proyección, se interpola el PK correspondiente con la función interpolatePk y se crea un nuevo marcador con estas coordenadas.

La función getAutomaticPK es el punto de entrada para añadir una anotación automática. Primero, convierte los marcadores existentes a un formato adecuado y verifica si ya existe un PK cercano a la ubicación actual del usuario (dentro de 1.5 metros). Si es así, actualiza _pkMarker con este marcador existente. Si no, utiliza findInterpolatedMarkerByLocation para encontrar o crear un nuevo marcador interpolado y actualiza _pkMarker con este nuevo marcador.

```
HSINGH
private fun projectPointOnLine(p: LatLng, p1: LatLng, p2: LatLng): LatLng {
    val x1 = p1.latitude
    val y1 = p1.longitude
    val x2 = p2.latitude
    val y2 = p2.longitude
    val x3 = p.latitude
    val y3 = p.longitude

    val k = ((y2 - y1) * (x3 - x1) - (x2 - x1) * (y3 - y1)) / ((y2 - y1).pow( n: 2) + (x2 - x1).pow( n: 2))
    val x4 = x3 - k * (y2 - y1)
    val y4 = y3 + k * (x2 - x1)

    return LatLng(x4, y4)
}

// Función para interpolar el pk
HSINGH
private fun interpolatePk(p1: MapMarker, p2: MapMarker, projectedPoint: LatLng): Double {
    val totalDistance = calculateDistance(p1.coordinates, p2.coordinates)
    val projectedDistance = calculateDistance(p1.coordinates, projectedPoint)
    return p1.pk + (projectedDistance / totalDistance) * (p2.pk - p1.pk)
}

// Función principal para encontrar el nuevo marcador
HSINGH
private fun findInterpolatedMarkerByLocation(markers: List<MapMarker>, location: LatLng): MapMarker? {...}

HSINGH
fun getAutomaticPK(){
    val mapMarkers = markers.value?.map { it.toMapMarker() }
    val pkIsSaved = mapMarkers?.any { calculateDistance(it.coordinates, _miUbicacion.value) < 1.5}
    if(pkIsSaved == true){
        _pkMarker.value =
            mapMarkers.first { calculateDistance(it.coordinates, _miUbicacion.value) < 1.5 }
                .toPKMarker()
        new = false
    }else{
        val pkFounded = mapMarkers?.let { findInterpolatedMarkerByLocation(it, _miUbicacion.value) }
        if(pkFounded != null){
            _pkMarker.value = pkFounded.toPKMarker()
            new = true
        }
    }
}
```

Imagen 36. Código de la forma automática

Forma Manual

Si el usuario selecciona esta opción, puede añadir una anotación manualmente a un PK específico. Si el PK no existe, se interpola entre los dos PKs más cercanos para determinar las coordenadas precisas del nuevo marcador, asegurando que la anotación se coloque en la ubicación correcta en la obra lineal. Esto proporciona al usuario un control preciso sobre dónde se añaden las anotaciones en función de PKs específicos.

El siguiente código describe el proceso para añadir una anotación manualmente a un Punto Kilométrico (PK) específico. Este proceso incluye la interpolación de coordenadas geográficas entre dos PKs contiguos para encontrar las coordenadas precisas del nuevo marcador.

```
private fun findInterpolatedMarker(markers: List<MapMarker>, manualPk: Double): MapMarker? {
    if (markers.isEmpty()) return null
    // Ordenar los marcadores por pk
    val sortedMarkers = markers.sortedBy { it.pk }

    val lowerMarker = sortedMarkers.lastOrNull { it.pk <= manualPk }
    val upperMarker = sortedMarkers.firstOrNull { it.pk >= manualPk }

    if (lowerMarker == null || upperMarker == null || lowerMarker == upperMarker) {
        // No se encontraron puntos adecuados para interpolar
        return null
    }
    val ratio = (manualPk - lowerMarker.pk) / (upperMarker.pk - lowerMarker.pk)
    val interpolatedLatitude = lowerMarker.coordinates.latitude +
            ratio * (upperMarker.coordinates.latitude - lowerMarker.coordinates.latitude)
    val interpolatedLongitude = lowerMarker.coordinates.longitude +
            ratio * (upperMarker.coordinates.longitude - lowerMarker.coordinates.longitude)

    val interpolatedCoordinates = LatLng(interpolatedLatitude, interpolatedLongitude)
    val formattedCoordinates = LatLng(
        String.format("%.6f", interpolatedCoordinates.latitude).replace( oldValue: ",", newValue: ".").toDouble(),
        String.format("%.6f", interpolatedCoordinates.longitude).replace( oldValue: ",", newValue: ".").toDouble()
    )
    return MapMarker(
        pk = manualPk,
        coordinates = formattedCoordinates,
        linearBuildingWork = lowerMarker.linearBuildingWork,
        text = null,
        imageUri = null
    )
}
 HSINGH
fun getManualPK(pk:Double){
    val pkIsSaved = markers.value?.any { pk == it.pk }
    if(pkIsSaved == true){
        _pkMarker.value = markers.value?.first { pk == it.pk }
        new = false
    }else{
        val mapMarkers = markers.value?.map { it.toMapMarker() }
        val pkFounded = mapMarkers?.let { findInterpolatedMarker(it, pk) }
        if(pkFounded != null){
            _pkMarker.value = pkFounded.toPKMarker()
            new = true
        }
    }
}
```

Imagen 37. Código de la forma manual

En la función findInterpolatedMarker se siguen los siguientes pasos:

- **Ordenar los marcadores por PK**: Primero, los marcadores se ordenan por sus PKs en orden ascendente.
- **Encontrar los marcadores más cercanos**: Se busca el marcador con el PK más cercano que sea menor o igual al PK manual (lowerMarker) y el marcador con el PK más cercano que sea mayor o igual al PK manual (upperMarker).
- **Comprobar la validez de los marcadores**: Si no se encuentran marcadores adecuados o si ambos marcadores son el mismo, la función retorna null.
- **Interpolación lineal**: Se calcula la razón entre la diferencia del PK manual y el lowerMarker respecto a la diferencia entre upperMarker y lowerMarker. Luego, se utiliza esta razón para calcular las coordenadas interpoladas (latitud y longitud) entre los dos marcadores más cercanos.
- **Formato de coordenadas**: Las coordenadas interpoladas se formatean para tener una precisión de seis decimales.
- **Crear y retornar el nuevo marcador**: Finalmente, se crea un nuevo marcador con el PK manual y las coordenadas interpoladas, y se retorna.

Y en la función getManualPK verifica si el PK manual ya existe en la lista de marcadores. Si existe, actualiza _pkMarker con el marcador existente. Si el PK manual no existe, convierte los marcadores a un formato adecuado y utiliza findInterpolatedMarker para encontrar o crear un nuevo marcador interpolado. Si se encuentra o crea un nuevo marcador, actualiza _pkMarker con este nuevo marcador.

Una vez obtenido el marcador al que se tiene que asociar la anotación, se abre una pantalla para añadir la anotación (Imagen 38).

Imagen 38. Pantalla de añadir anotación

5.3.8 Hallar PK

En esta pantalla (Imagen 39), ha habido pequeños cambios en cuanto se refiere a la estética como que los marcadores que tienen una anotación son de distinto color.

Donde se ha hecho un gran cambio es en el cálculo de hallar el PK más cercano.

Como hemos visto antes a la hora de añadir una anotación con la ubicación del usuario el PK calculado es la interpolación entre las coordenadas de los PKs contiguos, respecto a la proyección de la ubicación del usuario sobre la recta que une los PKs contiguos.

En el caso de forma manual se interpola entre los dos PKs más cercanos para determinar las coordenadas precisas del nuevo marcador, asegurando que la anotación se coloque en la ubicación correcta en la obra lineal.

Para hallar el PK más cercano se ha utilizado ambos enfoques. Para hallar el PK más cercano hace falta primero calcular el PK con la forma automática y después con la forma manual calcular las coordenadas precisas de ese PK calculado. Así obtendríamos el PK más cercano y sus coordenadas.

Imagen 39. Pantalla de hallar PK más cercano

5.3.9 Pantalla de Información

En la pantalla de información acerca del PK hallado (Imagen 40) solamente ha habido una mejora estética para que sea atractivo visualmente a la hora de leer la información.

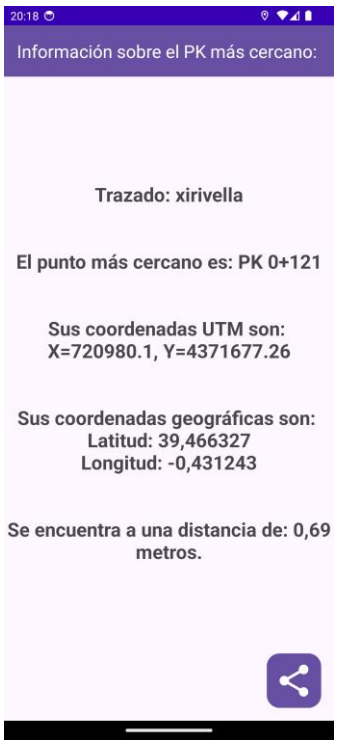


Imagen 40. Pantalla de información del PK

5.4 PROBLEMAS ENCONTRADOS DURANTE EL DESARROLLO

Durante la implementación de la aplicación, se identificaron varios desafíos significativos que requirieron soluciones específicas para garantizar su resolución efectiva:

5.4.1 Nuevas funcionalidades y algoritmos matemáticos

La introducción de nuevas funcionalidades en la aplicación necesitó la integración de conceptos y algoritmos matemáticos con los cuales no estaba familiarizado previamente. Esto representó un desafío considerable, ya que era crucial comprender estos conceptos para implementar correctamente las funcionalidades requeridas.

Solución: Para superar esta barrera, se llevó a cabo una investigación exhaustiva sobre los conceptos matemáticos relevantes. Esto incluyó el estudio de algoritmos específicos como la interpolación lineal y la proyección de puntos. La comprensión mínima necesaria se adquirió a través de la revisión de recursos académicos. Esta preparación permitió integrar estos algoritmos de manera efectiva en el desarrollo de la aplicación, asegurando su funcionamiento correcto y preciso.

5.4.2 Problemas con la base de datos Room y estructura cambiante

Se experimentaron dificultades significativas con la persistencia de datos utilizando Room, especialmente cuando la estructura de la base de datos cambió respecto a los datos ya almacenados. Este problema surgía al detectar discrepancias entre la estructura actualizada y la versión anterior de los datos guardados.

Solución: Para abordar este problema, se evaluaron dos enfoques potenciales:

- **Migraciones de base de datos:** Implementar migraciones de datos para actualizar la estructura de la base de datos de una versión a otra compatible.
- **Eliminación de la estructura anterior:** Optar por eliminar completamente la estructura anterior y sus datos en caso de que se detectara un cambio en la estructura de la base de datos. Esta opción fue seleccionada debido a que los datos con la estructura anterior no representaban la versión final y definitiva que la aplicación utilizaría.

Esta solución no solo aseguró la integridad y consistencia de los datos, sino que también simplificó el proceso de desarrollo y mantenimiento al evitar conflictos persistentes con la estructura de la base de datos.

A continuación, se muestra (Imagen 41) que en la creación de la base de datos de Room se añade la función fallbackToDestructiveMigration() en el builder para este cometido.

```
HSINGH
@Module
@InstallIn(SingletonComponent::class)
class PKMarkerProviderModule {
    HSINGH
    @Provides
    @Singleton
    fun providePKMarkerDatabase(@ApplicationContext context: Context): PKMarkerDatabase {
        return Room.databaseBuilder(
            context,
            PKMarkerDatabase::class.java,
            PKMarkerContract.DATABASE_NAME
        )
            .fallbackToDestructiveMigration()
            .build()
    }
```

Imagen 41. Código de la creación de la base de datos

6 PRUEBAS REALIZADAS

Durante el desarrollo de la aplicación, se realizaron pruebas exhaustivas para validar la funcionalidad y precisión de las siguientes características clave:

6.1.1 Forzar un PK

Se implementó una funcionalidad que permite forzar la creación de PKs en ubicaciones específicas, asegurando que estos puntos sean agregados de manera precisa y acorde con las necesidades del usuario y del sistema.

Se diseñaron casos de prueba (Imagen 42) que simulaban diferentes escenarios de inserción de PKs, incluyendo ubicaciones cercanas y distantes. Se verificó la consistencia de los PKs generados con las coordenadas proporcionadas.

Además, también se comprobó el funcionamiento de los algoritmos matemáticos como el de la interpolación cúbica spline.

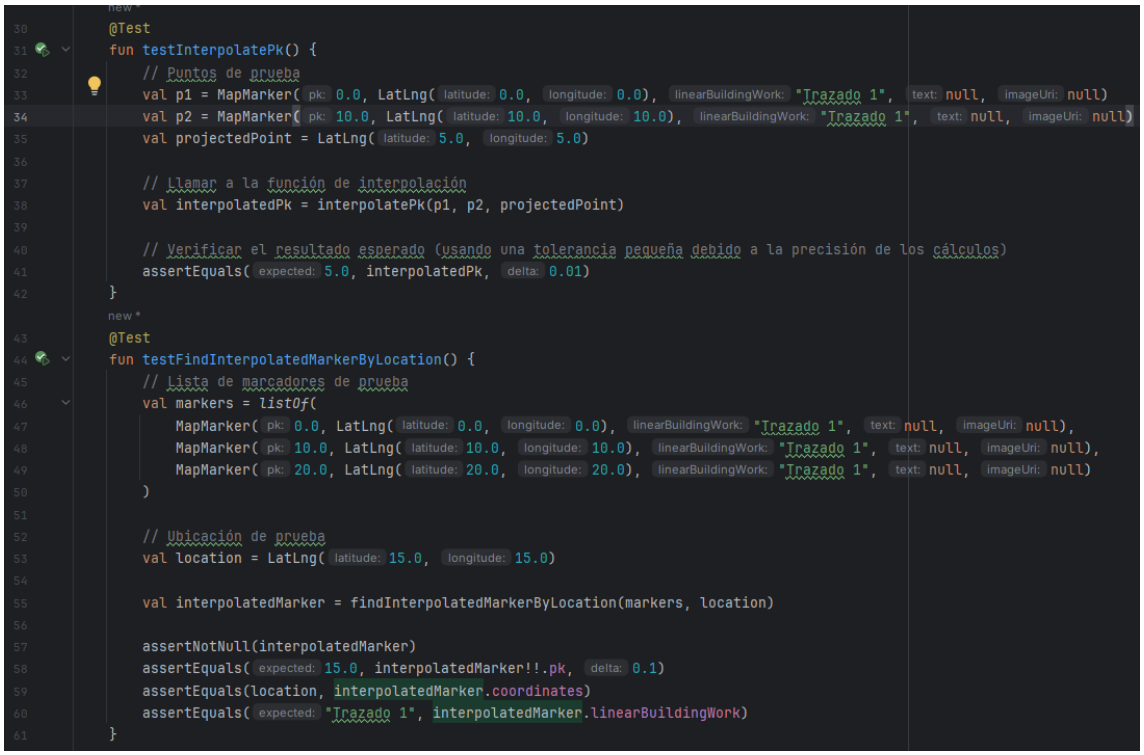
```
HSINGH *
class FunctionsTest {
    HSINGH
    @Test
    fun testCalculateDistance() {
        val start = LatLng( latitude: 40.748817,  longitude: -73.985428) // Ejemplo: Coordenadas de Nueva York
        val end = LatLng( latitude: 34.052235,  longitude: -118.243683) // Ejemplo: Coordenadas de Los Ángeles

        val distance = calculateDistance(start, end)

        // Distancia esperada en metros (aproximadamente)
        val expectedDistance = 3940000.0 // Puede variar ligeramente

        assertEquals(expectedDistance, distance,  delta: 10000.0) // Tolerancia de 10 km
    }

    HSINGH
    @Test
    fun testCubicSplineInterpolation() {
        val x = doubleArrayOf(0.0, 1.0, 2.0, 3.0, 4.0)
        val y = doubleArrayOf(0.0, 1.0, 4.0, 9.0, 16.0)
        val u = doubleArrayOf(0.5, 1.5, 2.5, 3.5)

        val result = cubicSplineInterpolation(x, y, u)

        val expected = doubleArrayOf(0.25, 2.25, 6.25, 12.25)

        assertArrayEquals(expected, result,  delta: 0.1) // Tolerancia ajustada a 0.1
    }

    HSINGH
    @Test
    fun testAdjustMapMarkers() {
        val markers = listOf(
            MapMarker( id: 1.0, LatLng( latitude: 0.0,  longitude: 0.0)),
            MapMarker( id: 2.0, LatLng( latitude: 1.0,  longitude: 1.0)),
            MapMarker( id: 3.0, LatLng( latitude: 2.0,  longitude: 2.0))
        )
        val userMarker = MapMarker( id: 2.5, LatLng( latitude: 1.5,  longitude: 1.5))

        val result = adjustMapMarkers(markers, userMarker,  pk: 1.0)

        assertEquals( expected: 4, result.size)
        assertTrue(result.contains(userMarker))
    }
```

Imagen 42. Pruebas de la funcionalidad forzar PK

6.1.2 Interpolación del PK

Otra funcionalidad crítica fue el cálculo de PKs interpolados entre puntos de referencia conocidos, asegurando una interpolación precisa y coherente a lo largo de una obra lineal.

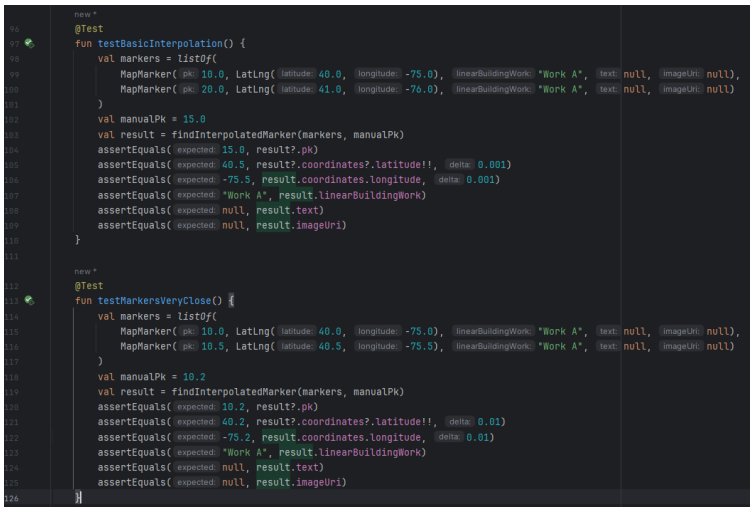
Se crearon escenarios de prueba que involucraban diferentes distancias entre puntos de referencia y condiciones de terreno (Imagen 43). Se compararon los PKs interpolados con los valores esperados para verificar la precisión del algoritmo de interpolación implementado.



```
@Test
fun testInterpolatePk() {
    // Puntos de prueba
    val p1 = MapMarker( pk: 0.0, LatLng( latitude: 0.0,  longitude: 0.0),  linearBuildingWork: "Trazado 1",  text: null,  imageUri: null)
    val p2 = MapMarker( pk: 10.0, LatLng( latitude: 10.0,  longitude: 10.0),  linearBuildingWork: "Trazado 1",  text: null,  imageUri: null)
    val projectedPoint = LatLng( latitude: 5.0,  longitude: 5.0)

    // Llamar a la función de interpolación
    val interpolatedPk = interpolatePk(p1, p2, projectedPoint)

    // Verificar el resultado esperado (usando una tolerancia pequeña debido a la precisión de los cálculos)
    assertEquals( expected: 5.0, interpolatedPk,  delta: 0.01)
}
new *
@Test
fun testFindInterpolatedMarkerByLocation() {
    // Lista de marcadores de prueba
    val markers = listOf(
        MapMarker( pk: 0.0, LatLng( latitude: 0.0,  longitude: 0.0),  linearBuildingWork: "Trazado 1",  text: null,  imageUri: null),
        MapMarker( pk: 10.0, LatLng( latitude: 10.0,  longitude: 10.0),  linearBuildingWork: "Trazado 1",  text: null,  imageUri: null),
        MapMarker( pk: 20.0, LatLng( latitude: 20.0,  longitude: 20.0),  linearBuildingWork: "Trazado 1",  text: null,  imageUri: null)
    )

    // Ubicación de prueba
    val location = LatLng( latitude: 15.0,  longitude: 15.0)

    val interpolatedMarker = findInterpolatedMarkerByLocation(markers, location)

    assertNotNull(interpolatedMarker)
    assertEquals( expected: 15.0, interpolatedMarker!!.pk,  delta: 0.1)
    assertEquals(location, interpolatedMarker.coordinates)
    assertEquals( expected: "Trazado 1", interpolatedMarker.linearBuildingWork)
}
```

Imagen 43. Pruebas de la interpolación del PK por localización

Además, para la interpolación de ubicación cuando se da un PK específico, se llevaron a cabo pruebas exhaustivas para verificar la precisión y fiabilidad de la interpolación de ubicaciones basada en un PK específico (Imagen 44).



```
new *
@Test
fun testBasicInterpolation() {
    val markers = listOf(
        MapMarker( pk: 10.0, LatLng( latitude: 40.0,  longitude: -75.0),  linearBuildingWork: "Work A",  text: null,  imageUri: null),
        MapMarker( pk: 20.0, LatLng( latitude: 41.0,  longitude: -76.0),  linearBuildingWork: "Work A",  text: null,  imageUri: null)
    )
    val manualPk = 15.0
    val result = findInterpolatedMarker(markers, manualPk)
    assertEquals( expected: 15.0, result?.pk)
    assertEquals( expected: 40.5, result?.coordinates?.latitude!!,  delta: 0.001)
    assertEquals( expected: -75.5, result.coordinates.longitude,  delta: 0.001)
    assertEquals( expected: "Work A", result.linearBuildingWork)
    assertEquals( expected: null, result.text)
    assertEquals( expected: null, result.imageUri)
}

new *
@Test
fun testMarkersVeryClose() {
    val markers = listOf(
        MapMarker( pk: 10.0, LatLng( latitude: 40.0,  longitude: -75.0),  linearBuildingWork: "Work A",  text: null,  imageUri: null),
        MapMarker( pk: 10.5, LatLng( latitude: 40.5,  longitude: -75.5),  linearBuildingWork: "Work A",  text: null,  imageUri: null)
    )
    val manualPk = 10.2
    val result = findInterpolatedMarker(markers, manualPk)
    assertEquals( expected: 10.2, result?.pk)
    assertEquals( expected: 40.2, result?.coordinates?.latitude!!,  delta: 0.01)
    assertEquals( expected: -75.2, result.coordinates.longitude,  delta: 0.01)
    assertEquals( expected: "Work A", result.linearBuildingWork)
    assertEquals( expected: null, result.text)
    assertEquals( expected: null, result.imageUri)
}
```

Imagen 44. Pruebas de la interpolación de la ubicación

7 DESPLIEGUE

El despliegue de nuestra aplicación en Google Play se realizó utilizando la cuenta de desarrollador de uno de mis tutores, lo cual simplificó el acceso a las herramientas necesarias para gestionar y mantener nuestra presencia en la tienda de aplicaciones oficial de Android. A continuación, detallo los pasos clave del proceso:

Creación de la Cuenta de Desarrollador:

- Para este proyecto, se utilizó la cuenta de desarrollador de uno de los tutores del trabajo de fin de grado. Esta cuenta permite el acceso completo a las herramientas de Google Play Console[3] después de aceptar los términos y condiciones y realizar el pago único de registro, actualmente de 25€.
- Se completaron todos los detalles personales y de contacto necesarios para configurar correctamente la cuenta de desarrollador.

Preparación de Recursos y Configuración Inicial:

- Acceder al menú principal de Google Play Console utilizando la cuenta del tutor y seleccionar "Crear aplicación".
- Proporcionar información detallada sobre la aplicación, incluyendo el nombre, la categoría, una descripción breve y completa, así como recursos gráficos como capturas de pantalla y vídeos para destacar las funcionalidades principales.
- Definir el idioma predeterminado y establecer un contacto a través de un correo electrónico o sitio web para los usuarios interesados.

Subida del APK y Configuración de Versiones:

- Generar y firmar digitalmente el archivo APK de la aplicación utilizando Android Studio.
    - **Generación del Archivo AAB:** Se utiliza Android Studio para compilar y generar un archivo Android Application Bundle (AAB), que es el formato requerido por Google para distribuir aplicaciones en Google Play.
    - **Firma Digital:** La aplicación es firmada digitalmente utilizando la clave de firma del desarrollador, un paso crucial para garantizar la autenticidad y la seguridad del archivo durante el proceso de subida.
- Cargar el archivo APK en Google Play Console en la sección correspondiente de "Lanzar > Producción".
- Crear una nueva versión de la aplicación e ingresar toda la información relevante, incluyendo los cambios recientes y mejoras implementadas.

Revisión y Aprobación:

- Someter la versión para revisión por parte del equipo de Google Play. Durante este proceso, se verifica que la aplicación cumpla con las políticas y estándares de la plataforma.

- Realizar ajustes o correcciones según sea necesario, basándose en las recomendaciones proporcionadas por el equipo de revisión de Google Play.

Lanzamiento y Monitoreo:

- Una vez aprobada, programar el lanzamiento de la aplicación o hacerlo de manera inmediata.

8 CONCLUSIONES

Para concluir este trabajo, es importante recordar los objetivos planteados al inicio y analizar si se han cumplido. Además, se explorarán posibles mejoras para una versión futura de la aplicación, buscando una herramienta aún más completa y eficiente.

8.1 LOGROS ALCANZADOS

El proyecto se inició con la intención de mejorar y ampliar una aplicación existente para la gestión de puntos kilométricos (PK) en obras lineales. Los siguientes objetivos han sido cumplidos de manera significativa:

1. **Desarrollo de Nuevas Funcionalidades**: Se han implementado nuevas características que mejoran la usabilidad y precisión de la aplicación, como la funcionalidad de forzar PKs y la interpolación precisa entre puntos de referencia conocidos. Estas funcionalidades han demostrado ser robustas y eficientes en pruebas exhaustivas, permitiendo a los usuarios obtener datos más precisos y confiables sobre las obras lineales.
2. **Mejora de la Interfaz de Usuario**: La interfaz ha sido rediseñada para ser más intuitiva y atractiva. La navegación se ha simplificado y se han añadido nuevas funcionalidades, como la lista de PKs con opciones de reajuste y eliminación. Esto facilita un mayor control sobre los datos para el usuario y mejora la experiencia de uso, haciendo que la aplicación sea más accesible y agradable de utilizar.
3. **Migración Tecnológica**: Se ha migrado el código de Java a Kotlin, aprovechando las ventajas de este lenguaje moderno y conciso. Esta migración ha mejorado la mantenibilidad y la robustez de la aplicación, permitiendo un código más limpio y menos propenso a errores. Además, ha facilitado la incorporación de nuevas funcionalidades y mejoras en el futuro.
4. **Despliegue Exitoso**: La aplicación ha sido desplegada exitosamente en Google Play Store, cumpliendo con todos los requisitos y estándares de la plataforma. Esto asegura una mayor accesibilidad y alcance entre los usuarios, permitiendo que más personas puedan beneficiarse de las mejoras y funcionalidades implementadas.

8.2 LIMITACIONES

A pesar de los logros alcanzados, se han identificado algunas limitaciones que deben ser abordadas para mejorar la aplicación en futuras versiones:

Colocación de Marcadores Basada en Distancia Recta: La aplicación actualmente coloca los marcadores basándose en la distancia recta entre el último marcador y la ubicación actual del usuario. Una mejora significativa sería calcular la ubicación a intervalos más cortos y sumar estas distancias hasta alcanzar el intervalo deseado (por ejemplo, cada 50 metros), lo que permitiría una colocación de marcadores más precisa y realista en terrenos irregulares o caminos curvos.

Anotaciones durante la Captura de PKs: Otra limitación importante es que no se pueden añadir anotaciones mientras se están capturando los PKs. Esta funcionalidad es esencial para que los usuarios puedan registrar observaciones o notas importantes sin interrumpir el proceso de captura de PKs.

8.3 FUTURAS LÍNEAS DE DESARROLLO

Para perfeccionar aún más la aplicación, se proponen las siguientes mejoras:

1. **Búsqueda y Modificación de PKs**: Incluir un buscador de PKs en la lista de PKs y permitir la modificación de sus atributos para facilitar la gestión y personalización de los datos. Esta funcionalidad incrementaría significativamente la practicidad y el manejo de los datos por parte del usuario.
2. **Mejoras en la Pantalla de Ajustes**: Añadir opciones como cambiar el color y la forma de los marcadores. Esto ofrecería una experiencia de usuario más personalizada y flexible.
3. **Optimización del Algoritmo de Colocación de Marcadores**: Mejorar el algoritmo para calcular la colocación de marcadores. En lugar de basarse únicamente en la distancia recta entre el último marcador y la ubicación actual del usuario, se propone calcular la ubicación a intervalos más cortos (por ejemplo, cada 5 metros) y sumar estas distancias hasta alcanzar el intervalo deseado (por ejemplo, cada 50 metros). Esto permitiría una colocación de marcadores más precisa y realista en terrenos irregulares o caminos curvos, asegurando que la aplicación pueda usarse en una variedad más amplia de escenarios y necesidades.
4. **Añadir Anotaciones en Tiempo Real**: Permitir a los usuarios añadir anotaciones mientras están capturando PKs. Esta mejora es esencial para que los usuarios puedan registrar observaciones y notas importantes sin interrumpir el proceso de captura, incrementando la utilidad y la eficiencia de la aplicación en el campo.

8.4 RELACIÓN CON LOS ESTUDIOS

Este proyecto ha sido esencial para aplicar y consolidar los conocimientos adquiridos durante el grado, especialmente en la asignatura "Desarrollo de Aplicaciones Para Dispositivos Móviles".

La creación de interfaces de usuario intuitivas, el desarrollo en Android utilizando Android Studio y lenguajes como Kotlin, y la gestión de bases de datos han sido aspectos clave del proyecto. Estos elementos, aprendidos y practicados en la asignatura, han sido fundamentales para el éxito de la aplicación.

Además, la asignatura proporcionó el marco teórico y práctico necesario para abordar estos desafíos, demostrando la importancia y relevancia de los estudios en el desarrollo de soluciones tecnológicas innovadoras.

8.5 REFLEXIÓN PERSONAL

Este proyecto ha sido una experiencia enriquecedora, permitiéndome adquirir nuevos conocimientos sobre la ingeniería civil, específicamente en el sector de la construcción e infraestructuras. Además, el trabajo realizado ha sido crucial para desarrollar mis habilidades como programador en el entorno Android, una tecnología relevante y demandada en la actualidad. Agradezco profundamente esta oportunidad de aprendizaje y crecimiento profesional.

En conclusión, los objetivos planteados al inicio del proyecto se han cumplido en gran medida. Las mejoras propuestas aseguran un camino claro para futuras versiones más avanzadas y

eficientes de la aplicación, abordando las limitaciones actuales y ampliando su funcionalidad para satisfacer mejor las necesidades de los usuarios.

BIBLIOGRAFÍA

1. DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES (2023). *Diapositivas de la asignatura desarrollo de aplicaciones para dispositivos móviles.* Valencia: Universidad Politécnica de Valencia.
2. Portal oficial de Android para desarrolladores. <https://developer.android.com/> [Consulta: abril de 2024]
3. Console, G.P. – Google Play Console. <https://play.google.com/console/about> [Consulta: junio de 2024]
4. Página oficial de Figma editor de gráficos vectorial y herramienta de generación de prototipos. <https://www.figma.com/> [Consulta: mayo de 2024]
5. Fórmula del semiverseno – Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Fórmula_del_semiverseno> [Consulta: abril de 2024]
6. PÉREZ MERCADER, DANIEL (2022). Aplicación móvil en Android para la gestión de obras lineales. Proyecto Final de Carrera. Valencia: Universidad Politécnica de Valencia.
7. Versiones de API de Android. <https://apilevels.com/> [Consulta: mayo de 2024]
8. Arquitectura de Android. <https://crhystamil.medium.com/android-arquitectura-de-android-f6618e6ae969> [Consulta: mayo de 2024]
9. Fórmula de interpolación cúbica spline <https://barcelonageeks.com/interpolacion-de-splines-cubicos/> [Consulta: abril de 2024]
10. SPÄTH, P., 2022. Pro Android with Kotlin: developing modern mobile apps with Kotlin and Jetpack. 2nd ed. New York, NY: Apress.

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**		**X**		
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**		**X**		
ODS 7. **Energía asequible y no contaminante.**			**X**	
ODS 8. **Trabajo decente y crecimiento económico.**		**X**		
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.			**X**	
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**			**X**	
ODS 15. **Vida de ecosistemas terrestres.**			**X**	
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

De la lista de Objetivos de Desarrollo Sostenible (ODS) presentados, mi proyecto está directamente vinculado con los siguientes:

Industria, innovación e infraestructuras: Este objetivo es especialmente relevante para mi proyecto debido a su aplicación en el campo de la Ingeniería Civil. La creación y el mantenimiento de obras lineales son esenciales para la sociedad moderna, y mi aplicación está diseñada para facilitar el trabajo de los técnicos y profesionales en este sector. Al mejorar la eficiencia y precisión en la localización y gestión de puntos específicos en infraestructuras lineales, mi aplicación contribuye de manera significativa a la conservación y desarrollo de la industria, fomentando la innovación y el fortalecimiento de las infraestructuras.

Además, considero que mi proyecto también tiene relevancia en:

Agua limpia y saneamiento: La aplicación desarrollada puede ser una herramienta crucial para los profesionales que trabajan en la gestión de recursos hídricos. Por ejemplo, en caso de fugas o roturas en tuberías situadas a lo largo de obras lineales como acueductos, la aplicación permite a los usuarios localizar rápidamente el punto exacto del incidente y comunicarlo eficazmente al técnico responsable. Esto no solo agiliza la solución del problema, sino que también contribuye a mantener la calidad del agua y el saneamiento en estas infraestructuras.

Trabajo decente y crecimiento económico: El uso de mi aplicación también facilita el trabajo de muchos profesionales en el sector de la ingeniería civil, reduciendo significativamente el tiempo y los costos asociados con el diseño y la construcción de obras lineales. Al prevenir daños y averías, se promueve una mayor eficiencia económica y se mejoran las condiciones laborales, lo que contribuye al crecimiento económico y a la promoción de un trabajo decente.

Salud y bienestar: De manera indirecta, mi proyecto también está relacionado con este ODS. La aplicación ayuda a los trabajadores y técnicos del sector a gestionar de manera más eficiente las incidencias en obras lineales, reduciendo el tiempo necesario para identificar y resolver problemas. Esto no solo mejora la seguridad y el bienestar de los trabajadores, sino también de la población en general, al minimizar los inconvenientes causados por interrupciones en servicios críticos como las telecomunicaciones.