UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación móvil para facilitar la consulta de información de transporte público utilizando datos abiertos del Ayuntamiento de València

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: López Álvarez, Álvaro

Tutor/a: Andrés Martínez, David de

CURSO ACADÉMICO: 2023/2024

Resumen

Algo muy importante en una ciudad sostenible y adaptada a las necesidades de sus ciudadanos es la comunicación de la ciudad con el transporte público. Una ciudad que dispone de transporte público busca la productividad y eficiencia de sus ciudadanos aportando una ayuda para acceder al trabajo, las escuelas y, en definitiva, cualquier zona de la ciudad. La ciudad de Valencia tiene varias formas de transporte público como son: los autobuses de EMT, el tranvía y el metro de metrovalencia y las bicis de valenbisi. *ValenMove* permite obtener información de cada transporte en una misma aplicación utilizando técnicas de integración de datos a través del portal de datos abiertos del Ayuntamiento de Valencia. Con esta solución se busca que el usuario de cualquier transporte de Valencia pueda encontrar fácilmente la información necesaria para moverse por la ciudad.

Palabras clave: Android, Kotlin, móvil, transporte, ciudad, Valencia, EMT, metrovalencia, valenbisi, aplicación móvil, integración, extracción, datos en abierto, servicio web.

Una cosa molt important en una ciutat sostenible i adaptada a les necessitats dels seus ciutadans és la comunicació de la ciutat amb el transport públic. Una ciutat que disposa de transport públic busca la productivitat i eficiència dels seus ciutadans aportant una ajuda per a accedir al treball, les escoles i, en definitiva, qualsevol zona de la ciutat. La ciutat de València té diverses formes de transport públic com són: els autobusos d'EMT, el tramvia i el metre de metrovalencia i les bicis de valenbisi. *ValenMove* permet obtindre informació de cada transport en una mateixa aplicació utilitzant tècniques d'integració de dades a través del portal de dades obertes de l'Ajuntament de València. Amb esta solució es busca que l'usuari de qualsevol transport de València puga trobar fàcilment la informació necessària per a moure's per la ciutat.

Paraules clau: Android, Kotlin, móbil, transport, ciutat, València, EMT, metrovalencia, valenbisi, aplicació móbil, integració, extracció, dades obertes, servici web.

A very important aspect of a sustainable city that is adapted to the needs of its citizens is the city's communication with public transport. A city that has public transport seeks the productivity and efficiency of its citizens by helping them to get to work, schools and, in short, any area of the city. The city of Valencia has several forms of public transport, including such as: EMT buses, the tram and subway of metrovalencia and the bycicles of valenbisi. *ValenMove*allows to obtain information about each transport in the same app using data integration techniques through the open data portal of the Valencia City Hall. The aim of this solution is to help the users to easily find the information they need to move around the city.

Key words: Android, Kotlin, phone, transport, city, Valencia, EMT, metrovalencia, valenbisi, phone app, integration, extraction, open data, web service.

Índice

1 Introducción **1**
1.1 Motivación del proyecto 1
1.1.1 Transporte público de Valencia 1
1.1.2 Datos en abierto 2
1.2 Objetivos del proyecto 3
1.3 Metodología 3
1.3.1 Metodología tradicional 4
1.3.2 Metodología ágil 4
1.3.3 Propuesta de metodología final 4
1.4 Estructura del trabajo 5

2 Contexto actual de la tecnología **7**
2.1 Aplicaciones oficiales del transporte público en Valencia 7
2.1.1 Aplicación de la EMT 7
2.1.2 Aplicación de Metrovalencia 8
2.1.3 Aplicación de valenbisi 9
2.2 Análisis de las aplicaciones estudiadas 9
2.3 Propuesta 11

3 Especificación de *ValenMove* **12**
3.1 Especificación de casos de uso 12
3.2 Diseño de la interfaz 14
3.2.1 Pantalla de búsqueda 14
3.2.2 Pantalla de información detallada 15
3.2.3 Pantalla de favoritos 16
3.2.4 Pantalla de ajustes 17
3.2.5 Diagrama de flujo entre pantallas 18
3.3 Modelo de dominio de *ValenMove* 19

4 Diseño y arquitectura de *ValenMove* **21**
4.1 Arquitectura MVVM 21
4.2 Principios SOLID 22
4.3 Estructura de carpetas de la solución 23
4.3.1 Carpeta *data* 23
4.3.2 Carpeta *di* 24
4.3.3 Carpeta *model* 25
4.3.4 Carpeta *ui* 25
4.3.5 Carpeta *utils* 26
4.3.6 Carpeta *res* 26
4.4 Tecnologías utilizadas para el desarrollo de *ValenMove* 27
4.4.1 Android Studio 27
4.4.2 Kotlin 28
4.4.3 Retrofit 29
4.4.4 Moshi 29
4.4.5 Room 29
4.4.6 Google Cloud 30
4.4.7 Google Maps 31

4.4.8 Street View . . . 31
4.4.9 JSoup . . . 32
4.4.10 Paging v3 . . . 32
4.4.11 Figma . . . 33
4.4.12 Material Theme Builder . . . 33
4.5 Diseño de la interfaz de usuario . . . 33

5 Implementación y desarrollo de la solución . . . 36
5.1 API del Ayuntamiento de Valencia . . . 36
5.1.1 Servicio web de EMT . . . 36
5.1.2 Servicio web de metrovalencia . . . 38
5.1.3 Servicio web de valenbisi . . . 40
5.1.4 Consultas a los servicios del Ayuntamiento de Valencia . . . 41
5.2 Implementación de la capa de aplicación y datos . . . 43
5.2.1 Lógica de la conectividad . . . 43
5.2.2 Lógica de la geolocalización . . . 44
5.2.3 Lógica detrás de la configuración . . . 44
5.2.4 Lógica para el almacenamiento de paradas en favoritos . . . 45
5.3 Lógica detrás de la información detallada de las paradas . . . 46
5.4 Implementación de la capa de presentación . . . 47
5.4.1 Implementación de la vista de forma general . . . 47
5.4.2 Implementación de los *ViewModel* de forma general . . . 47
5.4.3 Implementación de la pantalla de búsqueda . . . 48
5.4.4 Implementación de la pantalla de favoritos . . . 50
5.4.5 Implementación de la pantalla del mapa . . . 51
5.4.6 Implementación de la pantalla de ajustes . . . 51
5.4.7 Implementación de la pantalla de información detallada . . . 52

6 Resultados del desarrollo de *ValenMove* . . . 53
6.1 Resultado final de la pantalla de búsqueda . . . 53
6.2 Resultado final de la pantalla de información detallada . . . 54
6.3 Resultado final de la pantalla de configuración . . . 54
6.4 Resultado final de la pantalla de favoritos . . . 55
6.5 Resultado final de la pantalla del mapa . . . 56
6.6 Disponibilidad en horizontal de las pantallas . . . 56
6.7 Resultado final de la internacionalización . . . 57

7 Implantación y pruebas . . . 58
7.1 Implantación . . . 58
7.2 Guías de calidad de Android . . . 58
7.2.1 Experiencia visual . . . 59
7.2.2 Rendimiento y estabilidad . . . 61
7.2.3 Privacidad y seguridad . . . 63
7.3 Pruebas de usabilidad con usuarios reales . . . 65

8 Conclusiones del trabajo . . . 72
8.1 Relación del trabajo con los estudios cursados . . . 73
8.2 Trabajo futuro . . . 73

Anexo A. Detalle de los casos de uso . . . 78

Anexo B. Pruebas de calidad de Android **84**

Anexo C. Guía paso a paso **89**

Anexo D. Objetivos de desarrollo sostenible **100**

Índice de tablas

1 Desglose de características comunes entre aplicaciones 10
2 CU-01 - Obtener paradas de transporte público. 78
3 CU-02 - Filtrar paradas por tipo de transporte. 78
4 CU-03: Obtener información detallada de una parada. 79
5 CU-04: Guardar parada en favoritos. 79
6 CU-05: Mostrar información de transporte en un mapa interactivo. . . . 80
7 CU-06: Buscar paradas por nombre. 80
8 CU-07: Mostrar parada en el mapa. 81
9 CU-08: Buscar paradas por cercanía. 81
10 CU-09: Cambiar el radio de cercanía. 82
11 CU-10: Cambiar lenguaje de la aplicación. 82
12 CU-11: Mostrar lista de paradas favoritas. 82
13 CU-12: Filtrar favoritos por tipo de transporte. 83
14 CU-13: Eliminar paradas de favoritos. 83
15 Desglose de la prueba de experiencia visual 85
16 Segunda tabla desglose de la prueba de experiencia visual 86
17 Desglose de la prueba de Rendimiento y estabilidad 87
18 Desglose de la prueba de Privacidad y seguridad 88
19 Implicación del proyecto sobre los objetivos de desarrollo sostenible. . . 100

Índice de figuras

1 Tablero Kanban durante el desarrollo del proyecto 5
2 Pantalla principal de la aplicación de EMT 7
3 Pantalla de información de una parada de EMT 7
4 Pantalla principal de la aplicación de metrovalencia 8
5 Pantalla de información de una parada de metrovalencia 8
6 Pantalla principal de la aplicación de Valenbisi 9
7 Pantalla de información de un estacionamiento de Valenbisi 9
8 Arquitectura en 3 niveles 12
9 Diagrama de casos de uso de *ValenMove* 13
10 Boceto de la pantalla de búsqueda 15
11 Boceto de la pantalla de información 15
12 Boceto de la pantalla de favoritos 16
13 Boceto de la pantalla de ajustes 17
14 Diagrama de flujo de navegación entre pantallas 18
15 Diagrama de Clases de *ValenMove* 19
16 Arquitectura MVVM 21
17 Principios SOLID 22
18 Estructura de carpetas de la solución 23
19 Estructura de carpetas de la solución 24
20 Estructura de la carpeta de la interfaz de usuario 25
21 IDE *Android Stuido* 27
22 Lenguaje de programación Kotlin 28
23 Diferencias entre Java y Kotlin 29
24 Icono de Google Cloud. 30
25 Icono de Google Maps. 31
26 Icono de Street View 31
27 Logo de Figma 33
28 Material Theme Builder aplicado a mi solución 33
29 Logo de *ValenMove* 34
30 Iconos de marcador en el mapa. 34
31 Resultado obtenido del API de EMT 37
32 Vista de la página web que almacena las llegadas próximas. 38
33 Resultado obtenido del API de EMT 39
34 Próximas llegadas de metrovalencia. 39
35 Resultado obtenido con una consulta al API de valenbisi. 40
36 Arquitectura utilizada para la extracción de datos. 42
37 DAO utilizado en la solución 46
38 Boceto de las funcionalidades de la pantalla buscar 48
39 Estructura de la pantalla buscar 49
40 Resultado final de la pantalla buscar 53
41 Resultado final de la pantalla información detallada 54
42 Resultado final de la pantalla de configuración 55
43 Resultado final de la pantalla de favoritos. 55
44 Resultado final de la pantalla del mapa. 56
45 Resultado final de todas las pantallas dispuestas de manera horizontal. . 57
46 Diferencias entre idiomas de la aplicación en la pantalla de buscar. . . . 57
47 Diferencias entre un contraste alto y uno bajo de contenedor y contenido. 61

48 Tiempos capturados en milisegundos para la prueba de carga de la aplicación 62
49 Gráficas representativas de la renderización de la pantalla en tiempo real 62
50 Diálogo para dar permisos de localización. 64
51 Diálogo explicativo del porqué se piden permisos de ubicación. 64
52 Cuestionario de pruebas de usabilidad primera pregunta 65
53 Cuestionario de pruebas de usabilidad segunda pregunta 66
54 Cuestionario de pruebas de usabilidad tercera pregunta 66
55 Cuestionario de pruebas de usabilidad cuarta pregunta 67
56 Cuestionario de pruebas de usabilidad quinta pregunta 67
57 Cuestionario de pruebas de usabilidad sexta pregunta 68
58 Cuestionario de pruebas de usabilidad séptima pregunta 68
59 Cuestionario de pruebas de usabilidad octava pregunta 69
60 Cuestionario de pruebas de usabilidad novena pregunta 69
61 Cuestionario de pruebas de usabilidad décima pregunta 70
62 Cuestionario de pruebas de usabilidad onceava pregunta 70
63 Pantalla buscar 89
64 Pantalla buscar - pulsar en buscar 90
65 Pantalla buscar - pulsar en filtrar 91
66 Configuración 92
67 Ir a configuración 92
68 Pantalla Información 93
69 Ir a favoritos 94
70 Pantalla Favoritos 95
71 Pantalla Favoritos - pulsar en filtrar 96
72 Pantalla Favoritos - pulsar en eliminar todos 96
73 Pantalla Mapa 97
74 Pantalla Mapa - pulsar sobre marcador 98
75 Pantalla Mapa - pulsar sobre esconder marcadores 98

1. Introducción

En este capítulo se expone la motivación que me llevó a realizar este trabajo, seguidamente de los objetivos a conseguir con la finalización del mismo y, por último, la metodología utilizada para el desarrollo de esta solución. Además, se expone en un último apartado, la estructura general de la memoria. He analizado una metodología tradicional frente a una ágil.

1.1. Motivación del proyecto

Soy un estudiante de la Universidad Politécnica de Valencia (UPV) que se desplazó desde Mallorca a Valencia para estudiar en esta universidad. Después de años de utilizar el transporte público de Valencia, di con la página de datos abiertos del Ayuntamiento, donde se puede obtener información del transporte público de aquí y mucho más. Así es como empieza mi motivación, tanto por la extracción de datos en abierto como por el transporte público de esta interesante ciudad.

1.1.1. Transporte público de Valencia

En una ciudad como Valencia, podernos desplazar por la ciudad de forma eficiente y económica es muy importante para que la ciudad pueda evolucionar económica, laboral y educativamente. El transporte público es muy necesario para garantizar que todos los ciudadanos, sin tener en cuenta su nivel económico, puedan moverse libremente por la ciudad. Esta forma de desplazarse ayuda a miles y miles de personas a llegar a su trabajo, a las escuelas, institutos, hospitales, y, en definitiva, a moverse por toda la ciudad. Consigue conectar a la gente de una punta de la ciudad a la otra, del barrio más pequeño y escondido a la plaza más importante de la metrópoli.

El transporte público ayuda mucho también al medioambiente, reduciendo la congestión del tráfico, pues menos coches en las calles equivale a menos atascos y menos emisión de gases a la atmósfera. Además, los autobuses, tranvías y trenes suelen ser más eficientes energéticamente que los vehículos privados, contribuyendo así a una reducción significativa de la huella de carbono en la ciudad.

No nos podemos olvidar de las ventajas en cuanto a accesibilidad para personas con discapacidades y adultos mayores que nos brinda el transporte público. Personas que no pueden utilizar un vehículo o que directamente no pueden caminar, se ven capaces de moverse gracias a los autobuses, tranvías y trenes, que, además, tienen muchas facilidades para este tipo de personas, como pueden ser los asientos reservados para ellos.

Además, una de las principales causas de muerte en España es debido a los accidentes de tráfico. Según el Instituto Nacional de Estadística (INE), han habido desde el 2018 al 2022, 8615 muertos por causas externas de accidentes de tráfico, según se comenta en el artículo [1]. El transporte público ayuda en gran medida a disminuir estos accidentes, al disminuir la cantidad de coches en las calles o al tener una ruta concreta por la cual sólo puede pasar el vehículo de transporte público como pueden ser las vías de tren o

de tranvía.

Está claro que el transporte público es crucial para el correcto desarrollo de una ciudad. Esto nos lleva a Valencia, donde conocemos dos grandes empresas que se encargan de esta tarea: la Empresa Municipal de Transporte de Valencia(EMT) y metrovalencia.

Para conocer más sobre la empresa EMT Valencia[1] nos podemos ir a su propia página de la wikipédia [2] donde podemos encontrar que se trata de una empresa que aporta un servicio de transporte público mediante una red extensa de autobuses, creada el 1986 pasando de ser propiedad de la empresa SALTUV a en su totalidad del Ayuntamiento de Valencia. Sabemos también que aporta sus servicios a toda la ciudad de Valencia y, además a las poblaciones de Alboraya, Alfara del Patriarca, Burjasot, Moncada, Tabernes Blanques, Sueca, Sedaví y Vinalesa.

La otra empresa de transporte público es Metrovalencia[2], que según su página de wikipédia [3], es el producto que proporciona la empresa pública Ferrocarriles de la Generalidad Valenciana (FGV). Inaugurada bajo el nombre de metrovalencia el 5 de mayo de 1995, que anteriormente se conocía como FGV. Cuenta con 6 líneas de metro y 4 de tranvía dando un total de 10 líneas de ferrocarriles tanto subterráneos como exteriores.

Además de metrovalencia y EMT, en Valencia existe otra forma de transporte que no tiene que ver con vehículos motorizados, se trata de Valenbisi[3], un servicio de alquiler de bicicletas con estaciones de bicicletas por toda la ciudad. Este servicio aparece en 2010 a manos de la empresa JCDecaux España S.L.U, aunque el propietario es el Ayuntamiento de Valencia, como podemos ver en su página de la wikipédia [4]

1.1.2. Datos en abierto

El otro punto de motivación del proyecto, es la explotación de datos en abierto. En términos de datos, es bien sabido que existen una cantidad exorbitada de datos en todo el mundo. Información que utilizan las empresas, a veces datos privados propios de estas y, otras veces, estos datos se muestran para el uso de cualquiera.

En el caso de España, por ejemplo, tenemos una gran cantidad de datos gratis para el uso de cualquiera, sólo en la base de datos en abierto aportados por el gobierno español hay casi 82700 conjuntos de datos, como podemos ver en su sitio web [5]. Además, como podemos ver en un blog, centrado en el uso de datos abiertos en la educación [6], de la propia pagina web de datos abiertos del gobierno, es complicado estimar la presencia actual de los datos abiertos en los recursos educativos abiertos, pero si es fácil encontrar algunos ejemplos interesantes del uso de estos datos abiertos para la enseñanza. Por ejemplo, citan la página web de procomun [7], que utiliza la plataforma ArcGIS Online de la Universidad Complutense de Madrid para hacer un mapa web para el aprendizaje de paisajes agrarios en España.

Podemos llegar a pensar que existen muchísimos datos útiles a las manos de cual-

[1] Página web de EMT Valencia: `https://www.emtvalencia.es/ciudadano/index.php`
[2] Página web de Metrovalencia: `https://www.metrovalencia.es/`
[3] Página web de Valenbisi: `https://www.valenbisi.es/es/home`

quiera, que no se están reutilizando y se están desperdiciando. No tan solo son útiles en educación, también en estadística, comercio, transporte (el que nos concierne para el ámbito de este trabajo) y muchos ámbitos más.

Con todo esto llegamos a la página web de datos en abierto del Ayuntamiento de Valencia [8]. En esta gran base de datos en abierto contamos con 277 conjuntos de datos llenos de información relacionada con la ciudad de Valencia. De esta página web que almacena tal cantidad de datos en abierto es de donde sacaremos información sobre las paradas de autobús de la Empresa Municipal de Transporte de Valencia(EMT) [9], información sobre paradas de metrovalencia [10] y, finalmente, sobre datos de valenbisi [11].

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es el de crear una aplicación sencilla para obtener información detallada de los distintos tipos de transporte público que se pueden encontrar en la ciudad de Valencia. A esta aplicación la llamaremos "*ValenMove*". La aplicación será diseñada para ser utilizada por dispositivos Android.

Los objetivos específicos del trabajo son los siguientes:

1. Facilitar la búsqueda de datos sobre transporte público de cualquier tipo (emt, metrovalencia o valenbisi) unificándolos en una sola aplicación
2. Permitir el guardar las paradas o estacionamientos favoritos para los usuarios para agilizar la obtención de la información importante para el consumidor.
3. Desarrollar una aplicación móvil con internacionalización, para que pueda ser utilizada por gente que hable otros idiomas (Valenciano, Castellano o Inglés).
4. Diseñar una interfaz agradable para el usuario y sencilla de utilizar gracias a los componentes de "Material Design"[12]
5. Utilizar técnicas de integración de datos para obtener la información de las "Interfaces de Programación de Aplicaciones"(API) del Ayuntamiento de Valencia.

1.3. Metodología

Existen muchas metodologías distintas para el desarrollo software y ninguna es la ideal para todos los casos de desarrollo. Por eso mismo, he decidido observar varias metodologías y analizar la viabilidad de las mismas, para escoger en consecuencia la más apta para este tipo de proyecto. En el libro "Scrum - Un método ágil para sus proyectos" [13] se habla de estas metodologías.

1.3.1. Metodología tradicional

La metodología tradicional trata de ser lo más eficiente posible cuando se conoce el problema a solucionar. Se basa en una planificación inicial total del desarrollo para luego continuar con el ciclo de trabajo. Este tipo de metodologías sufre mucho cuando el problema a resolver es de carácter cambiante, es decir, cuando nos encontramos en un contexto donde los requisitos pueden variar, el diseño puede cambiar, etc. Todo debe estar bien planificado y bien definido para poder comenzar con el desarrollo.

1.3.2. Metodología ágil

Las metodologías ágiles nacen del problema que podía suponer la metodología tradicional. La planificación es mucho menos exhaustiva y, más importante, es adaptativa. Se basa en repetidas entregas con ciclos de desarrollo rápidos, conocidos como incrementos o *sprints*, que suelen durar de 2 a 4 semanas. Busca la máxima conexión entre el cliente y el desarrollador, para cumplir al máximo las necesidades de los interesados. Es un método fácil de aprender y muy flexible a la hora de realizar cambios de planificación, pues depende de los recursos actuales y del tiempo restante de desarrollo.Además esta metodología genera mucha menos documentación que la tradicional.

1.3.3. Propuesta de metodología final

Debido al poco conocimiento de la tecnología a utilizar en este proyecto de primeras y de cómo llegar a la solución. Planteo utilizar una metodología ágil debido a su flexibilidad y a no dar tanta importancia a la documentación exhaustiva.

Seguiré el método *kanban* [14] que sigue un método visual y gráfico donde se pueden ver por medio de tarjetas las tareas que faltan por realizar, las que se encuentran en ejecución y, finalmente, las tareas terminadas con éxito. Con este enfoque, las tareas pueden ser asignadas a los integrantes del equipo de desarrollo y se pueden observar cuellos de botella cuando existen demasiadas tareas en ejecución y poder cambiar así, la planificación del *sprint* en consecuencia.

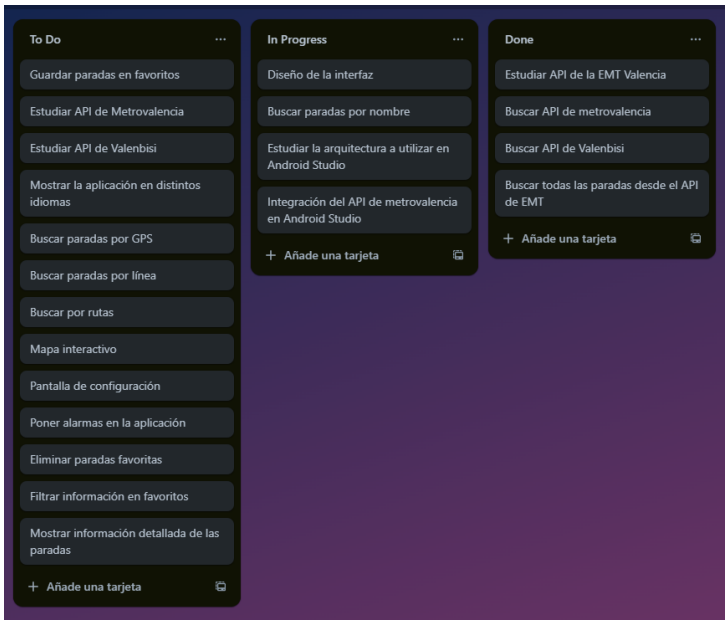

Figura 1: Tablero Kanban durante el desarrollo del proyecto

En la figura 1 podemos observar cómo tenemos una sección de *To Do*, donde almacenamos las funcionalidades y tareas a realizar en algún momento del desarrollo. En esta sección se añaden todas las ideas de funcionalidad que queremos implementar, aunque al final del proyecto exista alguna que no se haya realizado. A continuación, una sección de *in Progress* para poder observar las tareas que están siendo desarrolladas. Finalmente, la sección de *Done* donde podemos observar todas aquellas tareas que ya han sido realizadas con éxito.

1.4. Estructura del trabajo

A continuación, expondré la estructura que va a seguir la memoria, explicando los puntos que se van a tratar posteriormente:

- Capítulo 2 - Contexto actual de la tecnología: Esta sección es una investigación de las soluciones existentes en el mercado para el contexto del transporte público en valencia, junto con un análisis conjunto de las mismas, para finalizar con la propuesta de solución que aporto yo con este proyecto.
- Capítulo 3 - Especificación de *ValenMove*: En este capítulo se va a exponer la especificación de todas las capas de la solución. Se especifican los casos de uso de la aplicación, se muestran los bocetos que se han seguido para hacer la interfaz y, finalmente, el modelo de datos que sigue la capa de datos.
- Capítulo 4 - Diseño y arquitectura de *ValenMove*: Esta zona del proyecto explica la arquitectura que se ha utilizado para la solución, expone las distintas carpetas en que se divide el desarrollo y muestra todas las tecnologías distintas aplicadas en el proyecto.

- Capítulo 5 - Implementación y desarrollo de la solución: Este capítulo muestra a grandes rasgos las soluciones que se han ido encontrando a las tareas que se han especificado para lograr los objetivos del proyecto. Se expone el desarrollo realizado a nivel de capas de la arquitectura detallada en el capítulo anterior.
- Capítulo 6 - Resultados del desarrollo de *ValenMove*: Tras mostrar a grandes rasgos cómo se desarrolló la solución, este capítulo muestra los resultados obtenidos con el primer MVP (mínimo producto viable) del proyecto. Se muestran las pantallas de la aplicación, la relación de las funcionalidades con sus casos de uso y las pantallas y mucho más.
- Capítulo 7 - Implantación y pruebas: Este capítulo trata dos importantes pasos en el desarrollo software, la implantación y las pruebas. Se muestra los pasos necesarios para desplegar la aplicación así como las pruebas realizadas para comprobar el correcto funcionamiento de la solución.
- Capítulo 8 - Conclusiones del trabajo: Este capítulo trata las conclusiones finales tras haber realizado el proyecto para mirar hacia atrás y ver todo el trabajo realizado una vez terminado el proyecto. También se habla del trabajo futuro que se podría realizar para mejorar la aplicación.
- Anexo A: En esta área de la memoria se muestra la especificación formal de los casos de uso del capítulo 3.
- Anexo B: En este anexo se encuentran las pruebas realizadas de la guía de calidad de Android en formato tabluar.
- Anexo C: En esta sección se encuentra la guía paso a paso que ha realizado cada uno de los usuarios que han probado la aplicación para las pruebas de usabilidadad.

2. Contexto actual de la tecnología

En este capítulo, se muestran las diferentes soluciones ya existentes para la consulta de información relacionada con el transporte público en Valencia. El análisis de las características de estas aplicaciones de forma general, me ayudará a perfilar la nueva solución que se propone en este TFG.

2.1. Aplicaciones oficiales del transporte público en Valencia

Para concretar el contexto que plantea el desarrollo de esta aplicación de transporte público, debemos observar otras aplicaciones similares que aborden este tema. Lo lógico es mirar aquellas aplicaciones que ya existen de los servicios que existen en la ciudad de Valencia: la aplicación de la EMT, metrovalencia y valenbisi.

2.1.1. Aplicación de la EMT

En esta aplicación, que podemos encontrar en Play Store como EMT Valencia [15] podemos observar una interfaz amigable, donde la principal funcionalidad es su mapa de paradas de bus. Esta aplicación tiene un sistema para buscar paradas por nombre, por línea y por medio del Sistema de Posicionamiento Global (GPS), además de un sistema para calcular rutas de EMT, metrovalencia y valenbisi. Solo muestra datos de las paradas de la EMT, tanto de metro como de valenbisi indica la ruta que se podría realizar. Es una aplicación bastante completa, con muchas funcionalidades para buscar por todas las paradas de EMT.

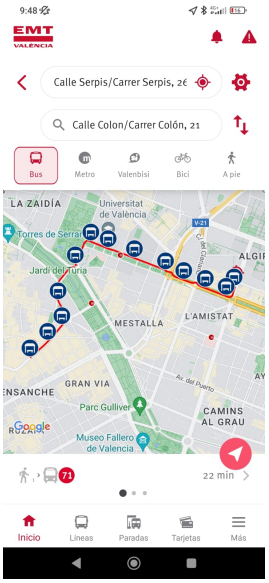

Figura 2: Pantalla principal de la aplicación de EMT

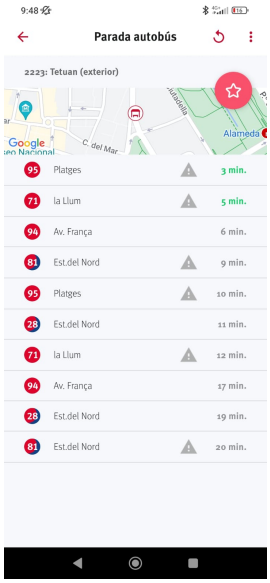

Figura 3: Pantalla de información de una parada de EMT

La aplicación cuenta con muchas funcionalidades relacionadas con las paradas de autobús. En la figura 2 podemos ver la pantalla principal con el mapa y una ruta de autobuses desde la calle Serpis hasta la calle Colón. Aunque nos permite cambiar a metrovalencia o valenbisi, solo nos permite pulsar en las paradas de EMT para obtener la información que se puede ver en la figura 3

2.1.2. Aplicación de Metrovalencia

En la aplicación de metrovalencia [16], podemos observar funcionalidades para buscar paradas concretas y para buscar líneas de ferrocarril (metro y tranvía). Además, tiene un sencillo mapa interactivo para poder ver las líneas y paradas que existen en metrovalencia. La aplicación, permite almacenar en favoritos paradas de metrovalencia, para rápido y fácil acceso. Además, muestra información sobre noticias relacionadas con metrovalencia.

Figura 4: Pantalla principal de la aplicación de metrovalencia

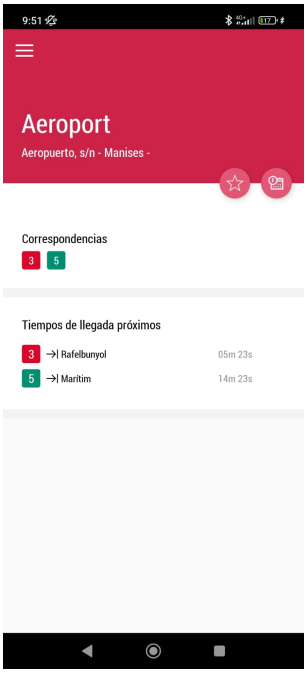

Figura 5: Pantalla de información de una parada de metrovalencia

Se puede observar en la figura 4 que en cuanto a funcionalidad es similar a la aplicación de EMT, aunque en esta no es tan importante el mapa interactivo que se encuentra en la parte superior derecha de la aplicación. Cuando queremos obtener información de una parada, nos envía a la pantalla que se puede ver en la figura 5, donde podemos ver las llegadas próximas, así como también un botón para añadir a favoritos y otro para ver el horario completo de la parada.

2.1.3. Aplicación de valenbisi

La aplicación de valenbisi, es parecida a la de EMT ya que su punto fuerte es el mapa para obtener información de los estacionamientos de bicicletas. Esta aplicación tiene un sistema de pago para los servicios relacionados con el alquiler de bicicletas. Requiere de iniciar sesión para funcionalidades como buscar rutas o alquilar bicicletas (habiendo contratado alguna tarifa anteriormente), la funcionalidad relacionada con la disponibilidad de bicicletas no requiere de iniciar sesión.

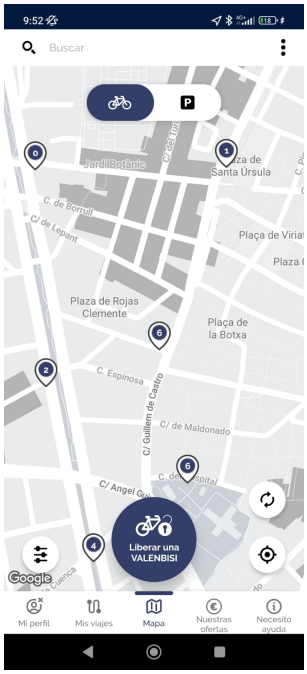

Figura 6: Pantalla principal de la aplicación de Valenbisi

Figura 7: Pantalla de información de un estacionamiento de Valenbisi

Esta aplicación tiene un mapa sencillo donde se ven todos los estacionamientos. Para más rápido acceso a la información podemos seleccionar, como se ve en la figura 6, si queremos ver cuantas bicicletas o cuantos sitios libres hay en cada estacionamiento. Al pulsar en algún estacionamiento, obtendremos información similar a la pantalla que se puede ver en la figura 7 donde se pueden ver las bicicletas disponibles, los parkings restantes y una valoración de las bicicletas por parte de los usuarios.

2.2. Análisis de las aplicaciones estudiadas

En el punto anterior, hemos visto en general un grupo de aplicaciones distintas que tienen que ver con el transporte público en Valencia. Todas estas aplicaciones tienen características comunes y algunas que son más únicas. Para observar con facilidad estas diferencias entre aplicaciones, y poderlas comparar con mi solución que llamaremos "*ValenMove*" se ha creado una tabla que muestra las características que se explican a continuación:

- Buscar paradas por nombre: La aplicación permite filtrar las paradas o estacionamientos por su nombre.
- Buscar paradas por línea: La aplicación permite filtrar las paradas por línea de transporte.
- Buscar paradas cercanas: La aplicación permite obtener las paradas cercanas al usuario sin necesidad de un mapa.
- Añadir pardas a Favoritos: La aplicación permite almacenar en una sección de favoritos las paradas que el usuario desee tener más a mano.
- Mapa interactivo: La aplicación contiene un mapa donde se pueden observar las paradas/estacionamientos del transporte público.
- Internacionalización: La aplicación está disponible en varios idiomas, que se pueden configurar desde un apartado de la aplicación.
- Información detallada de otros transportes: La aplicación permite obtener información de otros transportes públicos además del propio de la aplicación.
- Añadir a favoritos otros transportes: Se pueden guardar en favoritos todo tipo de transportes públicos.
- Visualización gráfica de la parada: Además del mapa interactivo, se puede ver una imagen de la parada de la que se obtiene información.

Tabla 1: Desglose de características comunes entre aplicaciones

Característica	**EMT**	**Metro-valencia**	**Valenbisi**	***ValenMove***
Buscar paradas por nombre	X	X		X
Buscar paradas por línea	X	X	-	
Buscar paradas cercanas				X
Añadir paradas a Favoritos	X	X	X	X
Mapa interactivo	X	X	X	X
Internacionalización	X	X		X
Información detallada de otros transportes				X
Añadir a favoritos otros transportes				X
Visualización gráfica de la parada	X			X

Después de ver todo el desglose de características comunes (Tabla 1), podemos observar que la mayoría de aplicaciones que existen son buenas para su contexto de uso, EMT para paradas de bus de EMT, metrovalencia para paradas de metro y tranvía, pero ninguna es buena en general para todos los transportes. En valenbisi como no hay líneas, no se puede buscar por líneas ni mostrar líneas en el mapa.

La búsqueda de paradas es bastante parecida en metrovalencia y en EMT y todas las aplicaciones tienen una funcionalidad para guardar paradas en favoritos. Aunque EMT sí permite ver la ruta de otros transportes (metrovalencia, valenbisi), no permite mostrar datos detallados de estos como sí hace con las paradas de la EMT.

Visualizar las paradas en forma de imágenes es algo que se puede obtener en la aplicación de EMT por medio de StreetView de Google Maps. Aunque impreciso, puede dar información de valor al usuario que busca una parada y solo tiene un mapa para encontrarla.

Como en Valencia hay muchas aplicaciones distintas y muchas formas distintas de transporte sería conveniente que existiera alguna aplicación que agrupe los detalles de las paradas de todas estas aplicaciones. Aquí es donde entra "*ValenMove*", para obtener fácilmente información tanto de EMT, como de metrovalencia y de valenbisi sin tener que buscar en internet o en cada aplicación distinta que existe.

2.3. Propuesta

El desarrollo de "*ValenMove*" busca el acceso sencillo a la información de paradas de transporte público en Valencia, juntando información tanto de EMT, como de metrovalencia y de valenbisi. Para ello nos centraremos en completar las características funcionales más prácticas de todas estas aplicaciones.

Después de ver la tabla 1, podemos observar que existen características que aparecen en todas las aplicaciones estudiadas, más concretamente, las funcionalidades de "Añadir paradas a Favoritos" y el "Mapa Interactivo". Estas dos funcionalidades son, por tanto, muy importantes para ser añadidas en la solución final.

Otras características interesantes que aparecen en varias aplicaciones son: "Buscar paradas por nombre", "Internacionalización". Estas funcionalidades son importantes para la solución final y serán implementadas con el desarrollo de este trabajo.

Es importante analizar las características que no existen en ninguna de las aplicaciones estudiadas: "Buscar paradas cercanas, "Información detallada de otros transportes" y "Añadir a favoritos otros transportes". Estas funcionalidades son lo que hace diferente mi aplicación de todas las demás. Estas funcionalidades se refieren, por un lado, a otra forma de acceso a la información de transporte público con la búsqueda por cercanía y, por otro lado, las funcionalidades relacionadas con juntar todos los tipos de transporte en una misma aplicación.

Una característica muy interesante que aparece en la aplicación de EMT es la "Visualización gráfica de la parada" por medio de *Street View*. Puede ser una característica muy útil para encontrar la parada que se ha buscado, por ello, se añadirá en mi solución.

Finalmente, la característica de "Buscar paradas por línea", aunque sea muy útil y aparezca tanto en EMT como en metrovalencia, es una funcionalidad compleja debido a un fallo de implementación de las API del ayuntamiento de Valencia y pasa a ser una característica menos prioritaria para ser añadida en mi aplicación.

3. Especificación de *ValenMove*

Después de analizar las aplicaciones que existen en el mercado en relación al transporte público en Valencia, vamos a plasmar una solución funcional y asequible. Aunque la arquitectura que he utilizado no es exactamente una arquitectura en 3 niveles, sino la arquitectura Model-View-ViewModel [17] (se explica en el apartado 4.1), voy a especificar las capas siguiendo el modelo de 3 capas o niveles [18], como se ve en la figura 8

- Capa de Presentación: Donde se compone todo lo relacionado con la interacción de usuario y el sistema a nivel visual en forma de interfaz gráfica de usuario.
- Capa de Dominio: Que hace de puente entre las capas de datos y de presentación, para abstraer información innecesaria de la capa de datos en la capa de presentación.
- Capa de Datos: Encargada de exponer los datos de la aplicación, ya sea en forma de base de datos, obtención de datos de un servicio, API,...

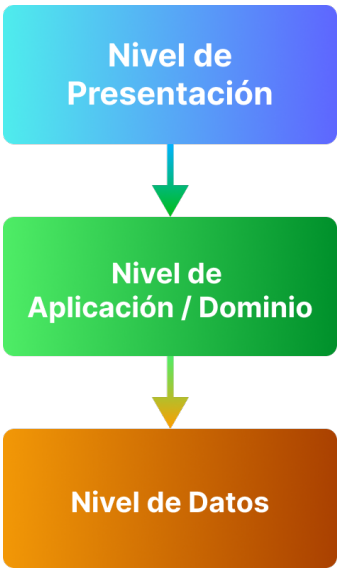

Figura 8: Arquitectura en 3 niveles

Aprovecharé este modelo para especificar cada parte de la misma con: una especificación de casos de uso para la capa de dominio y de presentación, un diseño de la interfaz para la capa de presentación y, finalmente, un modelo de dominio para la capa de datos.

3.1. Especificación de casos de uso

La especificación de Casos de Uso busca explicar la interacción entre los actores (usuarios distintos) y el sistema o sistemas [19]. Estos casos de uso, describen cómo el

usuario actúa con cada funcionalidad de la aplicación, las salidas y entradas esperadas, precondiciones que se deben cumplir para poder utilizar dicha funcionalidad y posibles alternativas/errores que pueden surgir de utilizar la funcionalidad específica. Estos casos de uso se muestran de forma gráfica en el diagrama de casos de uso de la figura 9, donde podemos observar que la aplicación no diferencia entre tipos de usuario, cualquier usuario de la misma tiene acceso a todas las funcionalidades de la aplicación.

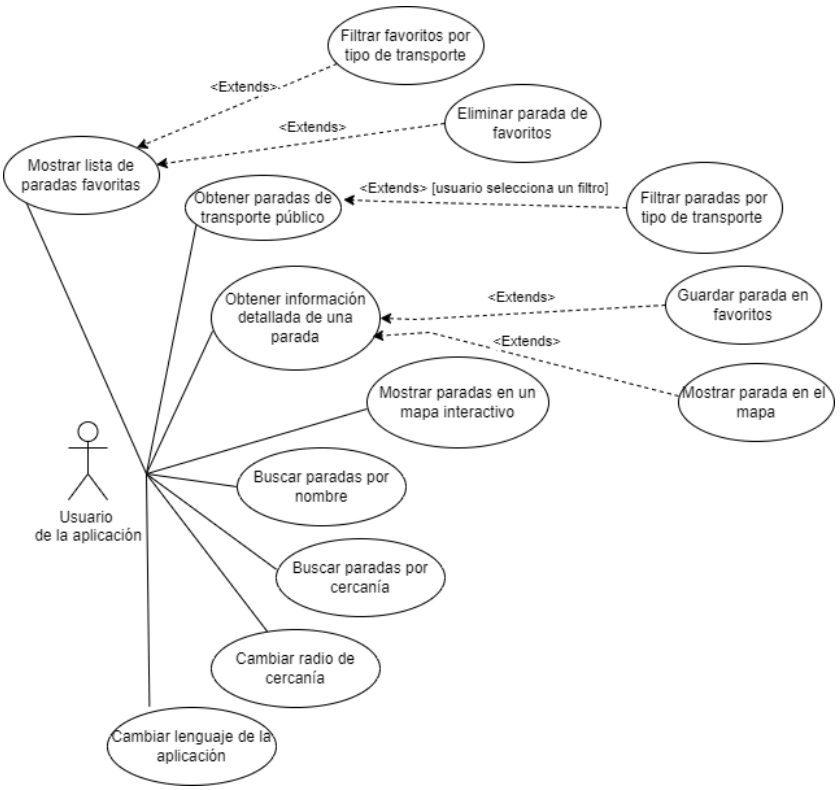

Figura 9: Diagrama de casos de uso de *ValenMove*

Los casos de uso de la aplicación están descritos detalladamente en la sección del Anexo A del documento. A continuación, se muestran los casos de uso descritos de forma corta:

CU-1 Obtener paradas de transporte público (tabla 2): Se muestra una lista de paradas de transporte público.

CU-2 Filtrar paradas por tipo de transporte (tabla 3): Las paradas mostradas se pueden filtrar por tipo de transporte(metrovalencia, EMT, valenbisi).

CU-3 Obtener información detallada de una parada (tabla 4): Al pulsar en una parada se muestra información más detallada sobre la misma.

CU-4 Guardar parada en favoritos (tabla 5): Las paradas se pueden guardar como favoritas para acceder más fácilmente a ellas.

CU-5 Mostrar información de transporte en un mapa interactivo 6): Con un mapa interactivo, se obtiene información de la localización de las paradas.

CU-6 Buscar paradas por nombre (tabla 7): Se puede filtrar las paradas por nombre de parada.

CU-7 Mostrar parada en el mapa (tabla 8): Desde la información detallada de una parada, se puede acceder a la pantalla del mapa interactivo sobre esa misma parada.

CU-8 Buscar paradas por cercanía (tabla 9): Se puede filtrar por paradas cercanas al usuario.

CU-9 Cambiar el radio de cercanía (tabla 10): Se puede ajustar el radio de cercanía utilizado en el filtro de cercanía.

CU-10 Cambiar lenguaje de la aplicación (tabla 11): El idioma de la aplicación se puede cambiar entre español, valenciano e inglés.

CU-11 Mostrar lista de paradas favoritas (tabla 12): Las paradas almacenadas en favoritos se pueden mostrar en forma de lista.

CU-12 Filtrar favoritos por tipo de transporte (tabla 13): Las paradas favoritas se pueden filtrar por tipo de transporte (metrovalencia, EMT, valenbisi).

CU-13 Eliminar paradas de favoritos (tabla 14): Se pueden eliminar una o varias paradas guardadas en favoritos.

3.2. Diseño de la interfaz

Para la capa de presentación, procedo a realizar unos bocetos sencillos a modo de base para el diseño final de la aplicación, donde se pueden ver las distintas funcionalidades que va a tener la solución.

3.2.1. Pantalla de búsqueda

Podemos observar en la figura 10 como se muestran en esta pantalla los casos de uso 1, 2, 6 y 8 (podemos observar el detalle de estos casos en las figuras 2, 3, 7 y 9 respectivamente). Esta pantalla muestra una lista de paradas según los filtros seleccionados tanto de búsqueda (por nombre, gps o todas las paradas) como de tipo de transporte(EMT, Metrovalencia o Valenbisi).

Es importante observar que para cambiar entre las distintas pantallas, tenemos una barra de navegación en la parte de abajo de la pantalla, además de un botón en la parte superior para acceder a la pantalla de configuración.

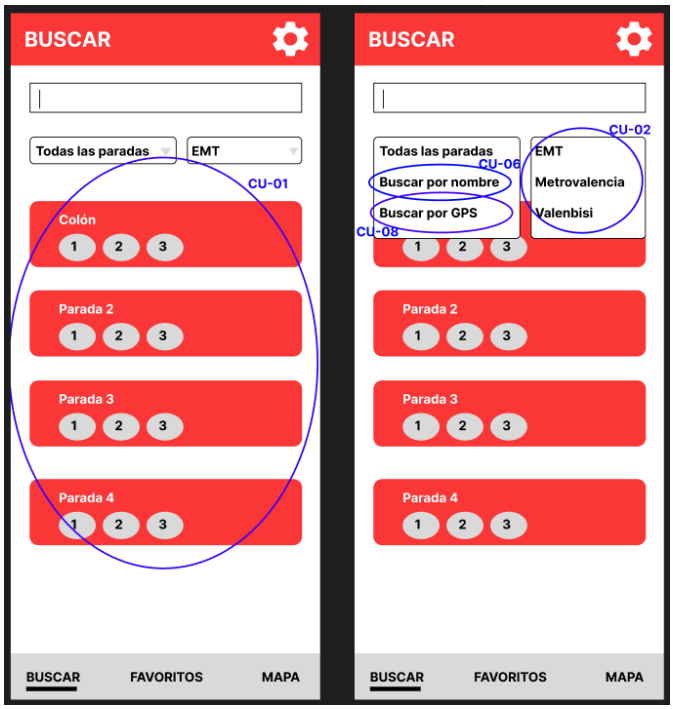

Figura 10: Boceto de la pantalla de búsqueda

3.2.2. Pantalla de información detallada

Figura 11: Boceto de la pantalla de información

Esta pantalla, como podemos ver en la figura 11, contiene la funcionalidad relacionada con los casos de uso 3, 4 y 7 (el detalle de estos casos de uso se encuentra en las tablas 4, 5 y 8 respectivamente). Aquí podemos observar información de próximas llegadas de bus y metro, las líneas que tiene una parada concreta, podremos ver de forma visual la parada que hemos seleccionado, podremos poner la parada en favoritos o ver la parada en el mapa interactivo.

En este caso, hemos accedido a la pantalla a través de la de búsqueda, la idea es que se pueda acceder a esta pantalla desde cualquier otra de la aplicación, simplemente con pulsar una parada.

3.2.3. Pantalla de favoritos

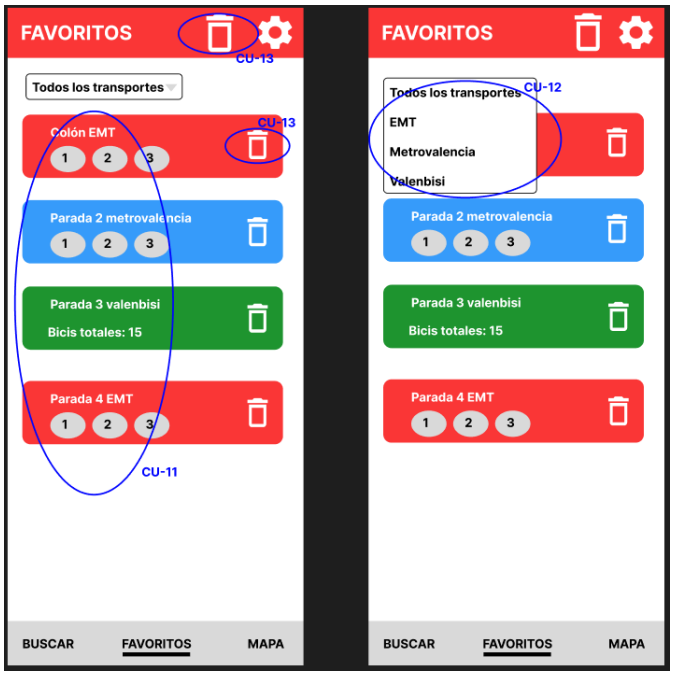

Figura 12: Boceto de la pantalla de favoritos

En la figura 12 podemos ver la pantalla de favoritos. En esta pantalla se encuentran las funcionalidades de los casos de uso 11, 12, 13 (el detalle de estos casos de uso se encuentra en las tablas 12, 13 y14 respectivamente).

En esta pantalla podemos ver cómo se guardan las paradas marcadas como favoritas, los filtros de tipo de transporte para poder ver solo las paradas de un único transporte y, cómo se pueden eliminar de favoritos estas paradas. El botón de arriba, cerca del que nos envía a la pantalla de configuración sirve para eliminar todas las paradas guardadas en favoritos.

3.2.4. Pantalla de ajustes

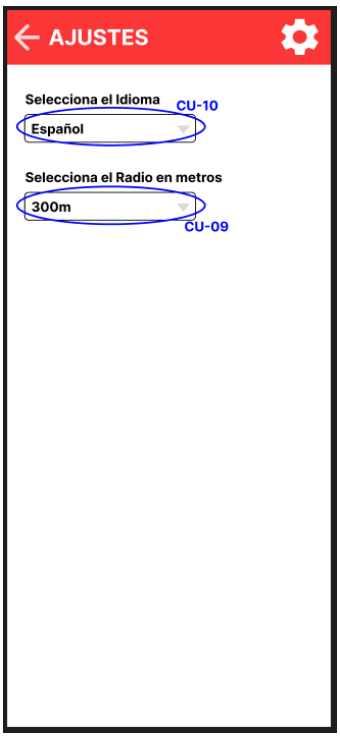

Figura 13: Boceto de la pantalla de ajustes

En esta pantalla, cuyo boceto se muestra en la tabla 13, podemos observar los casos de uso 9 y 10 (el detalle de estos casos de uso se encuentra en las tablas 10, 11 respectivamente).

Esta pantalla es la que se encarga de brindar al usuario las funcionalidades para cambiar el idioma de la aplicación, entre español, inglés o valenciano y, además, de poder ajustar el radio de cercanía en metros utilizado para buscar por cercanía (esta funcionalidad de buscar por cercanía se trata del caso de uso 8, que se detalla en la tabla 9) o para mostrar las paradas del mapa, que también van con este ajuste.

3.2.5. Diagrama de flujo entre pantallas

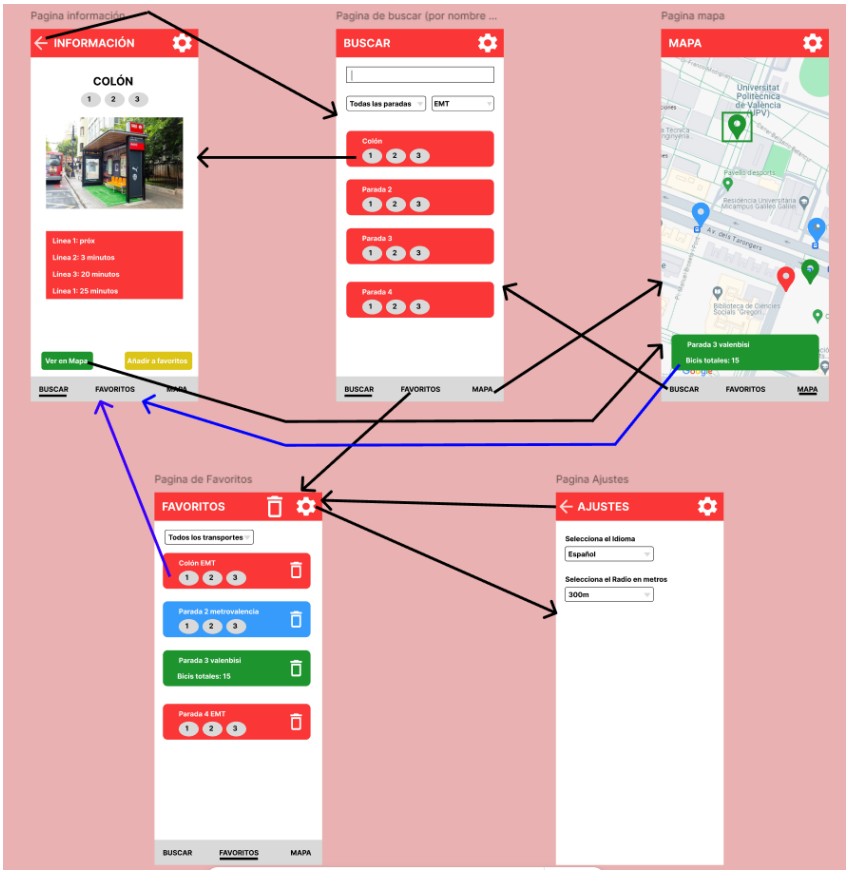

Figura 14: Diagrama de flujo de navegación entre pantallas

Existen 5 pantallas distintas en *ValenMove*, estas pantallas se comunican de diversas formas, que procedo a explicar a continuación analizando el gráfico de navegación que vemos en la figura 14.

Para simplificar las flechas del gráfico, se han omitido todas las que van a la pantalla de ajustes, se pretende que todas las pantallas principales de la aplicación permitan acceder a esta pantalla de configuración, como se observa en la navegación desde la pantalla de favoritos a esta por medio del botón de ajustes. Es importante observar también que esta pantalla de ajustes tiene un botón para volver a la pantalla anterior a esta.

La aplicación tendrá una barra de navegación en la parte inferior de la pantalla con la que podremos acceder a la pantalla de búsqueda, la pantalla de favoritos y la del mapa interactivo. Para simplificar las flechas, se han omitido de todas las pantallas menos de la de búsqueda.

Al pulsar una parada en concreto, nos enviará a la pantalla de información, para poder ver datos detallados de la parada seleccionada. Podemos ver unas flechas azules que indican alternativamente, desde dónde se puede acceder a esta pantalla que, además

de la pantalla de búsqueda, podemos acceder desde favoritos y desde el mapa interactivo. Esta pantalla tendrá un botón de volver atrás para acceder a la pantalla anterior desde donde se accedió a la información detallada.

3.3. Modelo de dominio de *ValenMove*

Después de ver la parte funcional, con los casos de uso y su diagrama de casos de uso, y de realizar el diseño de la interfaz de la aplicación, con bocetos y un diagrama de navegabilidad, me falta diseñar la parte más interna de la solución, las relaciones de las entidades del proyecto.

En la figura 15 se muestra un modelo de dominio para esta aplicación. Al ser una aplicación móvil que se basa en obtener datos de una API, no existe mucha carga de diseño en esta sección.

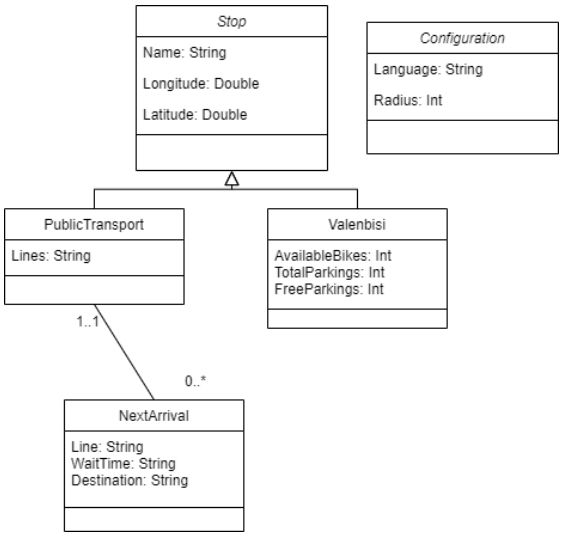

Figura 15: Diagrama de Clases de *ValenMove*

Podemos observar como hay dos tipos de parada en mi solución, una de transporte público, que se refiere a autobús, metro y tranvía y otra de valenbisi, para los estacionamientos de bicicletas.

Todas estas paradas tienen propiedades comunes, que se detallan en la clase *Stop*, que son el nombre y la geolocalización por latitud y longitud.

El estacionamiento de Valenbisi es más sencillo pues todos sus parámetros son enteros y no necesita de ninguna relación, como sí es el caso del transporte público, que para obtener las próximas llegadas tengo la clase *NextArrival*.

Finalmente, la aplicación tiene una pantalla de configuración y, por ello creo la clase *Configuration* para almacenar ahí la selección de configuración que cada usuario tiene en su teléfono.

Como se puede observar, no existe una clase *User*, por ejemplo. Esto se debe a que mi solución no tiene ningún tipo de registro o inicio de sesión de usuarios. De esta forma, la solución es más sencilla pues no necesito de una base de datos y de un servidor como podría ser "*Firebase*" [20].

4. Diseño y arquitectura de *ValenMove*

En este capítulo se va a detallar la arquitectura que se ha utilizado para desarrollar la solución, además de contar a rasgos generales las tecnologías utilizadas para realizar este proyecto.

4.1. Arquitectura MVVM

El trabajo está pensado para ser utilizado únicamente por dispositivos con sistema operativo "Android". Es por esto mismo que se ha utilizado la arquitectura "Model-View-ViewModel"(MVVM) [17] muy popular en este tipo de desarrollo de aplicaciones. Este patrón de diseño contempla una separación muy clara de las partes de la solución, garantizando así un mejor desarrollo, mantenibilidad, escalabilidad y prueba del código. En esta arquitectura los componentes principales son:

- Modelo (Model): Junta la lógica de negocio con la capa de datos. Esta capa se encarga de tener la funcionalidad general de la aplicación independiente de la interfaz de la misma, así como también del acceso a los datos. En mi caso, el acceso a los datos se realiza por medio de llamadas al API del Ayuntamiento de Valencia.
- Vista-Modelo (ViewModel): Se trata del intermediario entre la capa de modelo y la vista de la aplicación o interfaz. Esta capa se encarga de obtener los datos a partir de la capa de modelo y prepararlos para ser mostrados por la interfaz, por tanto mantiene un estado de la información que se encuentra en la interfaz.
- Vista (View): Esta es la capa que se encarga de mostrar los datos al usuario, para ello debe observar los datos que cambian en la capa vista-modelo y actualizar la interfaz acorde a estos cambios.

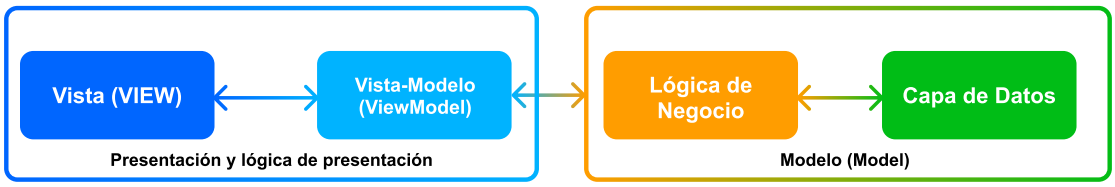

Figura 16: Arquitectura MVVM

Podemos observar gráficamente esta arquitectura en la figura 16.Este modelo intenta que la capa más cercana al usuario, es decir la vista, esté lo más separada posible del modelo, simplemente se encarga de mostrar la información que recibe a través del intermediario conocido como *ViewModel*.

4.2. Principios SOLID

En el desarrollo de esta aplicación, he tratado de seguir al máximo posible los principios SOLID, detallados por Robert C. Martin (Uncle Bob) [21]. Estos principios no son reglas, ni leyes perfectas, son afirmaciones que pueden conducir a una implementación mejor. Es necesario conocer cuando es conveniente aplicarlas y cuando puede entorpecer más que ayudar. Los principios SOLID se pueden ver en la figura 17

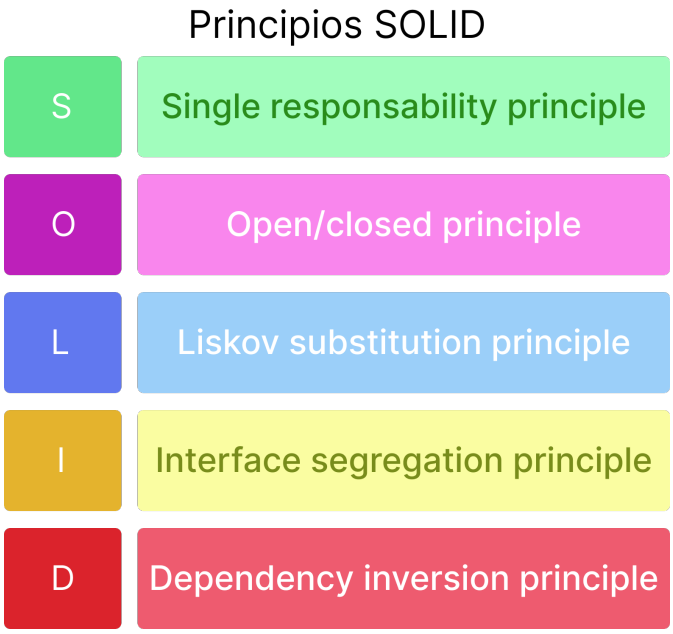

Figura 17: Principios SOLID

- Principio de responsabilidad única (*Single responsibility principle*): Cada clase debe tener una única responsabilidad, es decir, una única razón de cambiar. Se busca que las clases realicen las mínimas tareas posibles, a ser posible, una única funcionalidad por clase.
- Principio de abierto/cerrado (*Open / closed principle*): Las entidades (clases, funciones, módulos, etc) deben estar abiertas para la extensión y cerradas para la modificación.
- Principio de sustitución de Liskov (*Liskov substitution principle*): Las clases derivadas pueden sustituir a los objetos de clases base sin cambiar su funcionamiento. Este principio busca que no existan errores a la hora de utilizar clases derivadas.
- Principio de segregación de interfaces (*Interface segregation principle*): Los clientes no deberían depender de interfaces que no utilizan. Este principio recomienda tener varias interfaces con pocas funcionalidades en vez de una sola interfaz general que englobe todas las funcionalidades.
- Inversión de dependencias (*Dependency inversion principle*): Este principio fomenta el diseño orientado a las interfaces. Los módulos no deben depender de módulos de nivel más bajo, sino de abstracciones (interfaces) y las interfaces no deben depender de sus implementaciones.

Para lograr todos estos principios, la aplicación está separada por interfaces distintas. Por ejemplo, para toda la funcionalidad relacionada con la pantalla de favoritos, tengo varias interfaces de nombres "FavouritesDataSource" (para el acceso a los datos) y "FavouritesRepository", que se encarga de mostrar los datos preparados para ser utilizados por el ViewModel. El ViewModel, depende únicamente de la interfaz del repositorio, no de su implementación.

La inyección de dependencias se realiza gracias a la biblioteca de Android llamada "*Hilt*" que simplifica la funcionalidad de otra biblioteca de nombre "*Dagger*", encargada de llevar toda esta inyección de dependencias. Para ello, necesitamos proporcionarle el punto de entrada de nuestra aplicación a Hilt y definir unas clases módulo para detallar cómo se crean las dependencias. Finalmente, con anotaciones de Hilt, referenciar a la interfaz desde algún punto de la aplicación y Hilt se encarga de todas las dependencias.

De esta forma, se puede desarrollar por módulos sin tener el detalle de los demás módulos acabados. Podemos crear una especie de imitaciones de la implementación final pero más sencilla para ir probando cómo va a quedar el módulo que se desarrolla y este módulo no conoce qué hay por detrás de los módulos de los que depende.

4.3. Estructura de carpetas de la solución

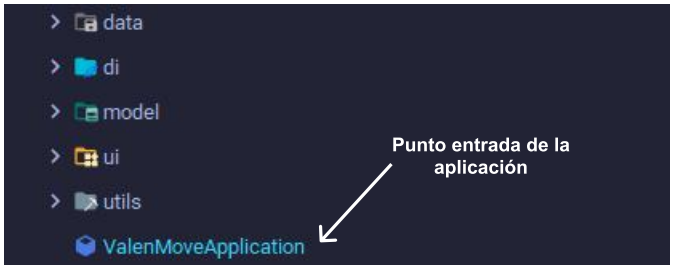

Figura 18: Estructura de carpetas de la solución

Además de las carpetas generadas automáticamente por Android studio (res, manifest, gradle Scripts, ...), he creado varias carpetas para diferenciar las partes lógicas de mi aplicación, como se puede ver en la figura 18.

4.3.1. Carpeta *data*

Es una de las carpetas más importantes, pues almacena toda la lógica de negocio y el acceso a los datos de la aplicación (se trata de la capa Modelo que veíamos en el apartado 4.1). En esta carpeta hay todo tipo de funcionalidades distintas como la obtención de la localización del usuario, la gestión de la conectividad del usuario o la obtención de los datos de la aplicación a través de las API del Ayuntamiento de Valencia. Se puede ver en la figura 19 todas las carpetas que tiene esta área de la solución.

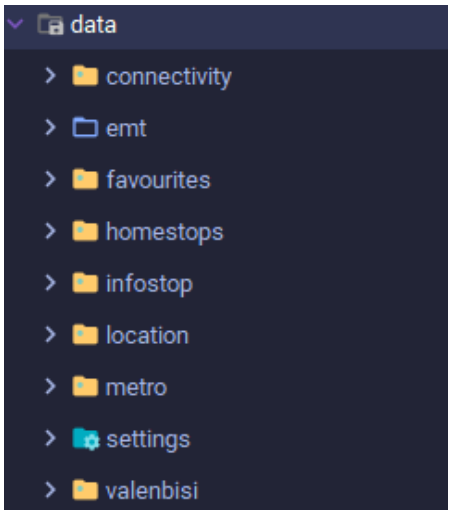

Figura 19: Estructura de carpetas de la solución

Tenemos las funcionalidades de almacenar datos en favoritos en la carpeta "favourites", la funcionalidad necesaria para mostrar información detallada en la carpeta "infostop", con las carpetas "metro", "valenbisi" y "emt" conseguimos obtener los datos de cada API de transporte de la solución y mucho más.

Todas estas carpetas siguen la misma estructura: una carpeta donde almacenar el modelo de datos concreto a utilizar en esa funcionalidad (que luego será transformado al del dominio de la aplicación), una interfaz para el acceso a datos que se llama "*DataSource*" (como en "FavouritesDataSource"), junto con su implementación en otro fichero que exitende esta interfaz, una interfaz para servir de repositorio entre el acceso de los datos y la capa de presentación que se llama "*Repository*" (como "FavouriteRepository") junto con su implementación y algún fichero adicional para funcionalidades concretas.

4.3.2. Carpeta *di*

El nombre de esta carpeta viene de *Dependency Injection* (Inyección de Dependencias). Esta carpeta es donde se gestiona toda la inyección de dependencias, para proveer las implementaciones de las abstracciones a todas partes del código sin depender del detalle de las implementaciones.

Aquí es donde se encuentran todos los módulos que permiten definir el comportamiento de las distintas dependencias del proyecto. Utilizo tanto módulos de enlace ("*bind module*") cómo módulos proveedores ("*provider module*"). Los módulos de enlace son los encargados de entregar la implementación de las interfaces a toda la aplicación, mientras que los módulos proveedores se encargan de proporcionar objetos más complejos, que son creados de una forma concreta y se pueden automatizar, por ejemplo, la creación de una instancia de un objeto que es igual para cualquier caso.

4.3.3. Carpeta *model*

Esta carpeta es dónde se almacenan las entidades del dominio de la aplicación. Estas entidades son las que se obtienen después de transformar los datos conseguidos a través de la capa del Modelo (capa que se describe en el apartado 4.1).

Estos datos serán utilizados por la capa de presentación para aportar la información suficiente y necesaria al usuario.

No se debe confundir con la capa de Modelo, esta capa se encuentra en la carpeta data, detallada en el apartado 4.3.1.

4.3.4. Carpeta *ui*

El nombre de esta carpeta proviene de *User Interface* (Interfaz de usuario). Esta carpeta se encarga de toda la funcionalidad relacionada con la capa de Presentación que, recordemos, contiene tanto la Vista como el mediador Vista-modelo.

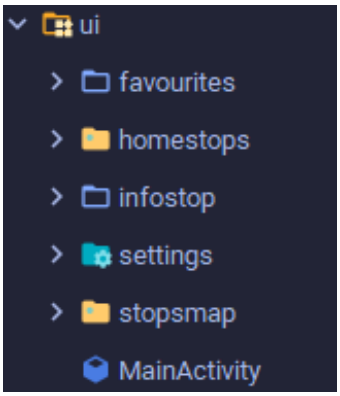

Figura 20: Estructura de la carpeta de la interfaz de usuario

Como se puede ver en la figura 20, esta carpeta está organizada con una carpeta por pantalla de la aplicación. En cada carpeta está todo el código que permite mostrar la pantalla al usuario y se encuentran los siguientes componentes:

- Actividades (*Activity*): Uno de los elementos más importantes de Android pues representa una sola pantalla de la interfaz de usuario, se utiliza como punto de entrada entre el usuario y el sistema y contiene todos los elementos de la interfaz como botones, campos de texto, fragmentos y muchos más componentes distintos. En mi aplicación realmente hay pocas actividades pues todo se maneja con fragmentos, la actividad más importante es la MainActivity (actividad principal) que se encarga de cambiar entre las pantallas de buscar, mapa y favoritos. La pantalla de información y la de configuración son actividades aparte de la principal.
- Fragmentos (*Fragment*): El fragmento en Android studio es un elemento similar a la actividad en cuanto a que también se encarga de almacenar los componentes de

la interfaz para mostrar al usuario y así poder conectar el usuario con el sistema. La diferencia principal entre actividad y fragmento es que en una actividad puede haber varios fragmentos de distintos tamaños. Esto nos permite tener muchísimas funcionalidades en una sola pantalla (actividad). En el caso de mi solución, tengo la pantalla principal que contiene la barra de navegación y luego, un contenedor de Fragmentos que cambia cuando se selecciona la pantalla de Buscar, el Mapa o la pantalla de Favoritos.

- Adaptadores (*Adapter*): Estas clases son las encargadas de manejar toda la funcionalidad interna de las listas que se muestran por pantalla mediante el componente de nombre *RecyclerView*. Una lista de estas características puede llegar a ser muy compleja de implementar, para eso están las clases *Adapter*, que simplifican muchas de las tareas comunes para mostrar un conjunto elevado de objetos en pantalla.
- Vista-Modelo (*ViewModel*): Estas clases, cuya funcionalidad ya ha sido explicada anteriormente en el apartado 4.1, se encargan de pedir datos a los repositorios (capa de lógica de negocio) para después mostrarlos a la interfaz, que es manejada por un fragmento. Además, también se encargan de almacenar el estado de la vista, la vista solo se suscribe a este *ViewModel* y observa los cambios en los datos que provienen del modelo.

4.3.5. Carpeta *utils*

En esta área del proyecto es donde se guardan funcionalidades concretas que no tienen tanto que ver con la arquitectura MVVM. Por ejemplo, tenemos una clase que guarda la funcionalidad para cambiar una imagen vector SVG a Bitmap para poder ser mostrada en el mapa de la aplicación. También almacena las clases de las distintas excepciones que pueden aparecer al utilizar *ValenMove*, como son *NoInternetConnectionException* y *NoGPSPermissionException*.

4.3.6. Carpeta *res*

Esta carpeta es generada automáticamente por el programa utilizado para el desarrollo del proyecto. Se trata de la carpeta donde se almacenan todos los recursos de la aplicación:

- Se encuentran los diseños de interfaces que muestra la solución, definidos mediante XML (Extensible Markup Language).
- Almacena todas las imágenes del proyecto en formato de vector, jpg o png.
- Guarda todas las cadenas de texto que se muestran en la aplicación para poder gestionar la internacionalización y poder tener varios idiomas para el proyecto.
- Guarda otras estructuras concretas para poder realizar, por ejemplo, la pantalla de configuración, de una forma semi-automática

- Se encuentran las fuentes de texto nuevas que se han empleado en el diseño de *ValenMove*.
- Almacena los estilos genéricos utilizados por los componentes de la interfaz, para hacerlos reutilizables y fácilmente modificables.

4.4. Tecnologías utilizadas para el desarrollo de *ValenMove*

Durante el desarrollo del proyecto me he topado con muchos problemas distintos: obtención de datos de API, mostrar datos paginados de API por la interfaz, obtención de datos de páginas web, y mucho más. En informática he aprendido que lo mejor es no reinventar la rueda y, para dar solución a todas estas problemáticas, he utilizado muchas bibliotecas distintas que agilizaron muchísimo el proceso de desarrollo. Voy a comentar a grandes rasgos cada una de estas tecnologías distintas y dónde las he aplicado.

4.4.1. Android Studio

Figura 21: IDE *Android Stuido*

He hablado ya mucho de "Android Studio" [22] pero, ¿qué es Android Studio? Se trata de un *Integrated Development Environment* (IDE) o Entorno Integrado de Desarrollo, como podría ser *Visual Studio Code* o *EclipseIDE*. Desarrollada por "Google", basándose en el IDE de "IntelliJ IDEA" desarrollado por *JetBrains*, se trata de una herramienta cuyo objetivo principal es el desarrollo de programas para el sistema operativo Android. Es creada para la programación en Java o en Kotlin y viene con una cantidad innumerable de facilidades para tratar con los dispositivos distintos de Android, entre ellas destaco algunas que me parecen interesantes:

- Emulador integrado: Este IDE permite probar el código con su emulador Android con un montón de dispositivos y tamaños completamente distintos, desde móviles de pantallas medianas con versiones de Android antiguas hasta tablets enormes y las nuevísimas versiones de Android. Este emulador tiene configuraciones avanzadas muy interesantes que nos permiten cambiar la geolocalización del usuario, el idioma del dispositivo y mucho más.

- Depuración: Permite situar puntos de ruptura en el código para probar el correcto funcionamiento de las distintas características del programa en desarrollo.
- Inteligencia Artificial "Gemini": Contiene una inteligencia artificial que consigue solucionar muchas dudas durante el desarrollo en términos de implementación, además que permite autocompletar código basandose en la estructura de tu programa.
- Prueba con dispositivos reales: Android Studio, además de tener un emulador integrado para ejecutar el programa, podemos conectar un dispositivo móvil al ordenador, ya sea por WiFi o por cable, para instalar la aplicación en este dispositivo.

4.4.2. Kotlin

Figura 22: Lenguaje de programación Kotlin

Kotlin [23] se trata de un lenguaje de programación de alto nivel orientado a objetos, de tipado estático[4] e inferencia de tipos[5]. Este lenguaje, desarrollado por JetBrains, es completamente interoperable con Java y está centrado en su *Java Virtual Machine* (JVM). Es un lenguaje que simplifica muchísimo la sintaxis de Java y aporta muchas funcionalidades que hacen más eficiente el desarrollo de programas.

Kotlin es útil para la gestión de colecciones de datos, la programación funcional, la programación concurrente con sus "corrutinas" y muchas más características que le hacen un lenguaje muy novedoso e interesante.

En la figura 23 se pueden observar las diferencias claras entre ambos. Cuando en Java para crear una clase necesitas un montón de código engorroso, en Kotlin, con una simple línea se crea esa clase de igual forma. Aunque es muy útil para muchos casos, también complica algunos otros casos, como por ejemplo en la herencia entre clases, es mejor seguir la estructura general de la creación de clases.

[4]El tipado estático es cuando el lenguaje de programación no permite la creación de objetos que pueden variar en tipo

[5]La inferencia de tipos es cuando el lenguaje de programación es capaz de detectar el tipo de las expresiones del código

Figura 23: Diferencias entre Java y Kotlin

4.4.3. Retrofit

Esta biblioteca de nombre *Retrofit* [24] ha sido muy importante para el contexto de mi proyecto, puesto que me ha facilitado la difícil tarea de la integración de datos de las API del Ayuntamiento de Valencia. Se trata de una biblioteca de cliente HTTP que simplifica la comunicación con servicios web. Con Retrofit, podemos acceder al servicio web y obtener una respuesta, típicamente en JSON. Además, también permite obtener información de estado de la respuesta, por si ha habido algún error.

4.4.4. Moshi

Moshi [25] se encarga básicamente de transformar y revertir la transformación de las respuestas JSON en objetos de Kotlin con la estructura que queramos. Si sabemos la estructura de las respuestas JSON del servicio web, podemos crear objetos con esa misma estructura para mantener la información en nuestro sistema y facilitar el manejo de la misma. Además, gracias a las anotaciones de *Moshi*, podemos realizar muchos ajustes interesantes a los objetos que genera la consulta al servicio web como, por ejemplo, poder abstraer algunas propiedades de la respuesta.

4.4.5. Room

Room [26] se trata de una biblioteca propia de Android Studio que facilita el almacenamiento de datos en los dispositivos. Con nociones básicas de gestión de bases de datos es una biblioteca sencilla de utilizar para guardar información y acceder a ella con consultas.

Se basa en el sistema de gestión de bases de datos relacionales de nombre SQLite, el motor de bases de datos más utilizado a nivel mundial [27], con una estructura muy sencilla y con la ventaja de no necesitar de un servidor externo para operar.

Room facilita la persistencia de datos en el dispositivo mediante la creación de tablas de datos con la anotación "@Entity", la creación de objetos para el acceso a datos con la anotación "@Dao" y la realización de consultas mediante la anotación "@Query". Además, facilita la migración de las entidades cuando se cambia el esquema de las mismas (cambia algún parámetro, se añade una tabla, se elimina una tabla,...).

Esta biblioteca es la que he utilizado para almacenar las paradas guardadas en favoritos y gestionar las consultas de dichas paradas, además de realizar también el borrado de las mismas.

4.4.6. Google Cloud

Figura 24: Icono de Google Cloud.

Google Cloud [28] es una plataforma de servicios en la nube creados por Google. Sirve para ser utilizados por desarrolladores a través de unas cuotas por interacción con el servicio cuyo precio depende del propio servicio. Se pueden encontrar servicios como su mapa "Google Maps", para crear mapas interactivos, servicios de geolocalización como el "Geocoding", que sirve para transformar posiciones de latitud y longitud en direcciones que el humano entienda, y una cantidad de casi 2000 servicios distintos.

Esta tecnología tiene un periodo de prueba donde te proporcionan 300€ para probar sus servicios durante 3 meses. Además, los servicios de mapas de Google tienen un nivel de prueba donde te dan 200€ al mes gratis para gastar en este tipo de servicios. Además puedes poner alertas que te avisan cuando estás próximo de quedarte sin crédito gratuito.

Es gracias a esta tecnología que he podido crear el mapa interactivo o mostrar la imagen con "Street View" de las paradas de transporte público.

4.4.7. Google Maps

Figura 25: Icono de Google Maps.

Google Maps[29], el mapa online más famoso y utilizado del mundo, con más de 25 millones de descargas solo en 2021 [30]. Google permite la incrustación de algunos de sus servicios, como es el caso de google maps, en el desarrollo de programas, mediante un sistema de cuotas en las que pagas por cada interacción al servicio. Google da 200€ gratis de sus servicios al mes para los desarrolladores.

He utilizado este servicio para crear la pantalla del Mapa interactivo. Este servicio nos permite visualizar un mapa del mundo con el que se puede mostrar información en forma de marcadores dentro del mapa, por ejemplo para mostrar las paradas de transporte público, así como también la localización del usuario, ajustar el zoom del mapa y muchas más funcionalidades muy interesantes de geolocalización.

4.4.8. Street View

Figura 26: Icono de Street View

El Street View[31] es otro servicio que proporciona Google para mostrar las áreas geográficas como si estuvieras andando por ellas. A través del coche de Google Maps, se realizan fotos en 360º de la gran mayoría de las calles del mundo y se muestran con este servicio. Con una posición geográfica con longitud y latitud podemos utilizar este servicio para mostrar una imagen de esa misma posición. He utilizado esta tecnología en la pantalla de información detallada de las paradas para aumentar la información de valor que obtiene el usuario interesado en más información sobre alguna parada de transporte público.

4.4.9. JSoup

JSoup[32] es una biblioteca que sirve para obtener información de cualquier página web. Para utilizar esta tecnología se debe tener conocimientos básicos de *Web Scrapping*, es decir, de extracción de datos a través de páginas web conociendo su estructura HTML (HyperText Markup Language).

Las próximas llegadas tanto de EMT, como de metrovalencia, se obtienen a partir de unas páginas que mantienen de forma actualizada esta información y dónde se puede ver la línea de transporte que va a llegar y el tiempo restante para que llegue a la parada solicitada. Como el API del Ayuntamiento no proporciona de forma directa esta información, sino que solo muestra la URL(Uniform Resource Locator) de la página web que mantiene esta información, he tenido que extraer todos estos datos mediante el uso de esta tecnología.

4.4.10. Paging v3

Esta biblioteca de nombre *Paging v3*[33], se encarga de facilitar la obtención de datos de API con paginación.

La paginación es la solución común que siguen las API que mantienen mucha información almacenada. En vez de mostrar miles y miles de datos para una consulta, sólo muestran unos pocos datos y un *link* a la próxima página de pocos datos. Así, las consultas que devuelven muchísimos datos se vuelven más eficientes, con la complejidad de tener que ir de página en página obteniendo los datos.

Paging v3 permite agilizar este proceso y mostrar los datos en pantalla en forma de listas que van haciendo consultas a medida que la lista va llegando al final, dando la sensación de que la aplicación ha recibido todos los datos con la primera consulta.

Las API del Ayuntamiento están paginadas y he requerido de esta tecnología para simplificar y hacer más eficiente la obtención y el mostrado de los datos por pantalla tanto de EMT, como de metrovalencia y valenbisi.

4.4.11. Figma

Figura 27: Logo de Figma

Figma [34], a diferencia de las demás tecnologías, está centrada en el diseño de la aplicación y no tanto en la implementación de la misma. Este programa es con el que he hecho los bocetos y el diagrama de navegabilidad de la aplicación. Su icono es el que se ve en la figura 27.

4.4.12. Material Theme Builder

Material Theme Builder [35] se trata de una tecnología que permite generar los colores de la aplicación proporcionando algunos colores principales. Esta tecnología genera colores para contenedores, contenido, de mensajes de error y mucho más. Facilita mucho el diseño de la interfaz.

4.5. Diseño de la interfaz de usuario

Para diseñar la interfaz de usuario en la aplicación, he utilizado las facilidades de Material Theme Builder [35] para escoger los colores de la aplicación. En la figura 28 se puede observar cómo se ha utilizado esta herramienta para generar los colores.

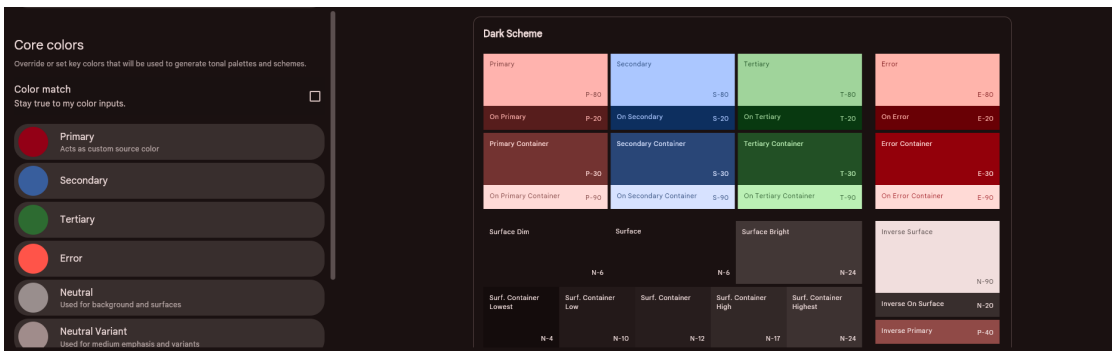

Figura 28: Material Theme Builder aplicado a mi solución

A modo de conseguir diferenciar los 3 tipos de transporte que existen en mi aplicación, he utilizado los colores rojo (para EMT), verde (para valenbisi) y azul (para metrovalencia). Gracias a la herramienta de Material theme builder, tengo un color para los contenedores y otro color para el contenido de estos, de esta forma queda una interfaz limpia y bonita.

Para la mayoría de fuentes de mi aplicación he utilizado la fuente Manrope[6]. A veces en negrita, para mostrar mensajes de error o información importante, por ejemplo, y otras veces en tamaño normal para la información más común.

El logo de la aplicación, que se puede ver en la figura 29, ha sido diseñado utilizando el programa "Figma" [34] y algunas imágenes vectoriales extraídas de la página web Flaticon [36]. Con un color rojo como color principal, pues es también el color principal de la aplicación.

Figura 29: Logo de *ValenMove*

Es un icono muy sencillo que muestra los distintos transportes que la aplicación puede obtener. Las imágenes vectoriales que aparecen en el icono también aparecen en la aplicación para diferenciar las paradas de cada transporte.

Con la ayuda de Figma, he creado también los marcadores que se muestran en el mapa interactivo, siguiendo el esquema de colores de la aplicación de rojo para EMT, azul para metrovalencia y verde para valenbisi. En la figura 30 se puede apreciar el resultado final del diseño de estos marcadores.

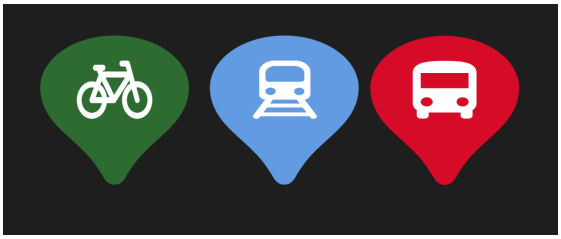

Figura 30: Iconos de marcador en el mapa.

[6]Página web de la fuente de Manrope: `https://fonts.google.com/specimen/Manrope`

He creado la interfaz final de la aplicación con la ayuda del diseño de los bocetos vistos en el apartado 3.2. Aunque estos bocetos son una referencia, he seguido bastante al detalle este diseño. Han habido algunos cambios para hacer la interfaz más simple y bonita al usuario. Estos cambios tienen que ver con la posición de algunas funcionalidades en la pantalla como son la barra de búsqueda para filtrar por nombre o el componente utilizado para filtrar los resultados, entre otros.

5. Implementación y desarrollo de la solución

En esta sección se van a tratar las tareas más importantes que se han realizado para el desarrollo del proyecto. Se explica la implementación detrás de las capas de modelo, que junta capa de acceso a datos y capa de aplicación y la capa de presentación, que junta las vistas y los componentes vista-modelo.

5.1. API del Ayuntamiento de Valencia

La columna vertebral de este proyecto es el API del Ayuntamiento de Valencia, de donde saca *ValenMove* toda la información sobre paradas de transporte público, tanto EMT, como metrovalencia y también la disponibilidad de los estacionamientos de valenbisi.

Para conseguir toda la información que proporciona el API he utilizado la biblioteca "Retrofit" junto con "Moshi" para acceder por una parte al servicio web y, por otra encapsular la respuesta JSON en un objeto de kotlin para facilitar el uso del mismo.

Para entender cómo he podido extraer la información de cada fuente de datos primero hay que ver su estructura.

5.1.1. Servicio web de EMT

La forma más sencilla y rápida de ver la estructura de un API es realizando una consulta al servicio[7] y analizando la respuesta obtenida, como se puede ver en la figura 31.

[7]Página web del servicio de EMT: `https://valencia.opendatasoft.com/explore/dataset/emt/api/`

Figura 31: Resultado obtenido del API de EMT

Analizando esta estructura podemos observar datos más y menos importantes:

- total_count: Se trata del contador de resultados totales a la búsqueda realizada. En este caso, al ser una búsqueda de todas las paradas muestra el número total de paradas.
- results: Esta es una lista de paradas obtenidas por medio de la búsqueda realizada al servicio. Es importante recalcar que, como máximo, sólo puede mostrar 100 paradas a la vez debido a las características de paginación del servicio.
- id_parada: Se trata del identificador numérico que tiene la parada en concreto en el API de EMT. Sirve para diferenciar cada parada.
- codvia: Se trata del código identificativo de la vía en la que se encuentra la parada, como se puede apreciar, puede ser un valor nulo si esta parada no tiene código de vía.
- numportal: Se trata del número de portal de calle en el que se encuentra situada la parada. Es una cadena de texto pues no siempre es un número.
- suprimida: No se explica en ningún lado del portal de datos del Ayuntamiento qué significa esta propiedad pero se puede suponer que es un valor que indica si la parada está o no en funcionamiento. Probando nunca he obtenido una parada con un valor distinto de 0.
- denominacion: Es el nombre que recibe la parada y el que se ve en el letrero del estacionamiento de los autobuses.
- líneas: Es una cadena de texto que almacena todas las líneas de bus que pasan por la parada.

- proximas_llegadas: Se trata de una URL a una página web que muestra información sobre las próximas llegadas en tiempo real, como se puede ver en la figura 32

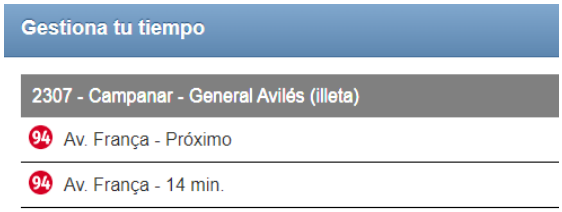

Figura 32: Vista de la página web que almacena las llegadas próximas.

- geo_shape y geo_point_2d: Se tratan de propiedades para obtener la geolocalización exacta de la parada. Se puede utilizar la versión sencilla con geo_point_2d o la versión más compleja que muestra el tipo de forma geométrica utilizada para obtener la zona geográfica de la parada con geo_shape.

Tras ver estas propiedades, he escogido unas cuantas importantes para mi aplicación.

La propiedad id_parada va a ser muy importante para poder diferenciar entre las paradas obtenidas de forma sencilla. En cambio, codvia lo veo menos importante debido a que no es un dato que aporte valor al resultado. El siguiente valor importante es el de denominación, que será la forma que tendrán los usuarios de identificar las paradas en la aplicación. La propiedad de nombre suprimida, como no puedo identificar cómo se utiliza y todas las paradas tienen este valor en 0, no es necesaria. Las líneas si son muy útiles para mostrarlas al usuario y que así sepa qué buses pueden ir a una parada concreta. La propiedad con la página web a próximas llegadas es muy importante para conseguir información en tiempo real de llegadas, aunque hará falta un paso más para sacar esta información de la página web. Finalmente, utilizaré geo_point_2d para obtener la información de geolocalización de la parada.

5.1.2. Servicio web de metrovalencia

Ahora hay que repetir el mismo proceso que con EMT para el API de metrovalencia[8]. En el portal de datos abiertos existen dos servicios para metrovalencia: "FGVBocas", que muestra la localización de las bocas de metro y tranvía y "FGV Estaciones / Estacions" que muestra la información de las paradas de metro. Sabiendo esto, he utilizado únicamente el servicio relacionado con las estaciones, pues el de las bocas es más complicado de utilizar y entender.

[8]Página web del servicio de metrovalencia: `https://valencia.opendatasoft.com/explore/dataset/fgv-estacions-estaciones/api/`

Figura 33: Resultado obtenido del API de EMT

Como se puede ver en la figura 33, existen muchas similitudes en cuanto a las propiedades del servicio web de EMT y el de metrovalencia. El gid en metrovalencia corresponde con la propiedad id_parada de EMT, el código en cambio, es un identificador que se encuentra entre 1 y 142 (el número total de estaciones de tren). El nombre, corresponde con la propiedad "denominación" de EMT y almacena el nombre que recibe la parada en cuestión. Este servicio tiene también una propiedad que almacena las líneas que pasan por la estación en forma de cadena de texto.

Un dato interesante es que las llegadas próximas se pueden obtener tanto en castellano como en valenciano, aunque en la práctica lo único que varía de una propiedad en otra es la palabra "destino" que en valenciano es "destinació". Se puede ver la estructura de esta página web en la figura 34.

Figura 34: Próximas llegadas de metrovalencia.

Como se puede observar, existen diferencias notables con los resultados que proporciona EMT a las próximas llegadas. En este caso, se puede ver la dirección o destino de cada llegada y en vez de ver cuanto queda para que llegue en minutos, se puede ver la hora exacta en la que llega cada metro.

Finalmente, el API proporciona las mismas propiedades para la geolocalización de la parada. Esto simplifica mucho la tarea de ubicar la parada con latitud y longitud.

5.1.3. Servicio web de valenbisi

Siguiendo con los mismos pasos que se han utilizado en los apartados anteriores nos encontramos con 4 API distintas en el portal de datos del Ayuntamiento de Valencia solo para valenbisi.

- Valenbisi 2022 - Alquileres por mes-dia-hora: Se trata de un conjunto de datos que almacena información sobre los alquileres de valenbisi en el año 2022
- Valenbisi 2022 - Tipo de abonos: Otro conjunto de datos del año 2022 en el que se clasifica el tipo de abono seleccionado por usuarios de distintas franjas de edad y sexo.
- Valenbisi 2022 - Alquileres y devoluciones: Un conjunto de datos más para el año 2022 en el que se almacena el número de préstamos y devoluciones realizadas ese mismo año por tramos de horas.
- Valenbisi Disponibilidad: Finalmente, un dataset (conjunto de datos) actualizado sobre la disponibilidad de las bicicletas de valenbisi y sus estacionamientos

Después de analizar los distintos conjuntos de datos, he utilizado el que se refiere a la disponibilidad de las bicicletas de valenbisi[9] pues los otros no son útiles para el contexto del proyecto y además que no están actualizados.

Figura 35: Resultado obtenido con una consulta al API de valenbisi.

[9]Página web del API de valenbisi: `https://valencia.opendatasoft.com/explore/dataset/valenbisi-disponibilitat-valenbisi-dsiponibilidad/api/`

En la figura 35 se puede ver que el API de valenbisi es muy diferente a los anteriormente detallados servicios de EMT y metrovalencia.

- number: Identificador que va del 1 al 276 para diferenciar los estacionamientos de valenbisi.
- address: Propiedad que guarda la dirección del estacionamiento de valenbisi, será la que utilice para nombrar los estacionamientos.
- open: Propiedad que indica si el estacionamiento está abierto al uso con una "T" y cerrado con una "F".
- available: Esta propiedad almacena el número de bicicletas disponibles en el estacionamiento.
- free: Esta propiedad almacena el número de aparcamientos libres que hay en el estacionamiento para dejar bicicletas.
- total: Este atributo guarda el número total de aparcamientos que tiene el estacionamiento de bicicletas.
- ticket: No tiene una explicación en el servicio web y solo se encuentran resultados con la propiedad en "T".
- updated_at: Fecha y hora en la que se ha actualizado la información que proporciona el API. Es importante destacar que este API se actualiza cada 10 minutos aproximadamente.
- geo_shape y geo_point_2d: La forma de obtener la geolocalización de las estaciones de valenbisi es exactamente igual a como se obtienen de las API de metrovalencia y EMT.

Este servicio es más sencillo que los anteriores pues no cuenta de una página extra donde se debe obtener la información detallada de los estacionamientos, como sí pasa con las próximas llegadas tanto de EMT como de metrovalencia. La dificultad del servicio es que la información obtenida por este es totalmente diferente a los anteriores servicios.

Las propiedades que he utilizado en el desarrollo de *ValenMove* en términos del servicio web de valenbisi son: address, para darle nombre a las paradas, number, para diferenciar todas las paradas de valenbisi, available, free y total para mostrar información sobre la disponibilidad de los estacionamientos, updated_at para que el usuario sepa cuan fiable es la información obtenida debido a la última actualización de esta información y finalmente, la propiedad relacionada con la latitud y longitud de los estacionamientos.

5.1.4. Consultas a los servicios del Ayuntamiento de Valencia

Una vez analizados los distintos servicios que tiene el portal de datos abiertos del Ayuntamiento de Valencia, he de realizar las consultas pertinentes para traer los datos a mi aplicación.

Tras probar con muchas consultas, me di cuenta que no existe una consulta a ningún servicio que devuelva correctamente las paradas que pertenecen a una línea concreta, se desestimó esa funcionalidad rápidamente tras el fracaso durante el estudio.

De forma alternativa a esto, tuve que aprender la forma correcta de formular la consulta para obtener paradas cercanas a un punto geográfico dando con un resultado positivo y correcto.

Para obtener los datos de las consultas a los servicios me valgo de dos bibliotecas: "Retrofit" y "Moshi", aunque existe una tercera biblioteca que se llama "Paging v3" que me va a ayudar mucho en la página de búsqueda para mostrar los resultados por pantalla.

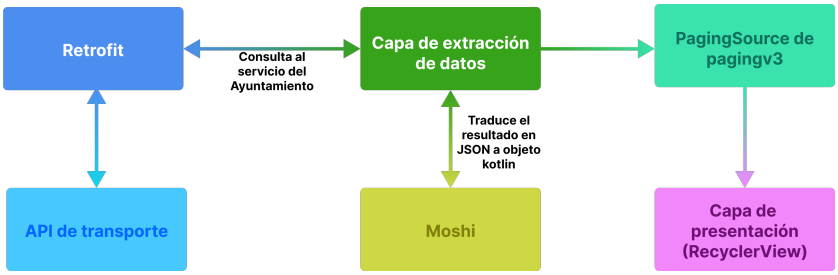

Figura 36: Arquitectura utilizada para la extracción de datos.

En la figura 36 se puede ver a grandes rasgos toda la comunicación que existe dentro del sistema para pasar del API del Ayuntamiento a una lista de paradas en la aplicación.

Lo primero es crear la interfaz que servirá para realizar las llamadas a los distintos servicios con Retrofit. Esta interfaz consta de dos métodos: uno para realizar llamadas generales y obtener cualquier entidad del servicio y otro para realizar consultas más complejas con la propiedad "where" que nos permite afinar a nuestro gusto la consulta. Para las llamadas, cada servicio tiene su propia URL aunque todos comparten una misma base del Ayuntamiento de Valencia.

Moshi ha ayudado a pasar los datos obtenidos en formato JSON a través de Retrofit en objetos kotlin para poder ser manejados por el sistema. Primero, el objeto kotlin resultante consta de la mayoría de propiedades de los servicios para ser traducido luego a una interfaz común que se llama "Stop" que tiene todas las propiedades similares de todos los tipos de transporte. Es importante destacar que valenbisi es algo especial pues tiene otro tipo de propiedades y se necesitará otro objeto distinto. El objeto "Stop" tiene el identificador, las líneas, las próximas llegadas, el nombre, la geolocalización y el tipo de transporte al que se refiere.

Para conseguir que se obtengan las entidades del servicio de una manera elegante y solventar la paginación existente en estos servicios he utilizado PagingV3. Con PagingV3 primero es necesario crear los "PagingSource", se tratan de extractores de datos para servicios con paginación.

Estos "PagingSource" obtienen los datos desde la capa de extracción de datos y calculan las próximas páginas que deben extraer utilizando dos valores de la consulta

llamados limit y offset. Limit (límite) se refiere al número total de resultados que trae la consulta mientras que el offset (desplazamiento) se trata del índice de posición del primer objeto obtenido de la consulta. Si tenemos 1000 objetos y como máximo se pueden devolver 100, utilizando estos dos valores podemos obtener los 100 primeros y con el offset seguir obteniendo los 100 siguientes, así hasta tener los 1000 objetos.

En la capa de presentación se debe crear un objeto de tipo "Pager" que utiliza los "PagingSoruce" y un objeto "PagingConfig" para pasar los datos a un RecyclerView. "PagingConfig" sirve para configurar la extracción de datos, por ejemplo, se puede ajustar el número de entradas por página desde este objeto.

Con todo esto, la lista está conectada a la extracción de datos de la forma en que cuando el usuario se está acercando al final de la lista, los datos se van extrayendo, para dar la apariencia de que no hay tiempos de carga y se ha obtenido toda la lista de paradas a la vez.

Con Paging v3, también podemos manejar errores durante las consultas a los servicios para mostrar mensajes de error o reintentar las consultas cuando se prevé que van a funcionar correctamente. La biblioteca permite capturar errores cuando hay falla al acercarse al final de la lista, cuando tiene que obtener datos al principio de la lista o en la primera consulta de todas. Esto hace muy flexible el manejo de los errores para saber cuándo se ha obtenido un error de conexión, por ejemplo.

5.2. Implementación de la capa de aplicación y datos

Como se veía en el apartado 4.3.1, la carpeta "data" es la que se encarga de almacenar toda la funcionalidad relacionada con la capa tanto de datos como de aplicación o dominio. Voy a explicar a rasgos generales cada parte de esta carpeta.

5.2.1. Lógica de la conectividad

En Android se puede comprobar la conectividad del dispositivo. Para ello, hay que añadir una capa de permisos de internet en el manifiesto de la aplicación. El manifiesto se trata de un fichero que almacena información sobre la aplicación como: pantallas que tiene la aplicación, nombre de la aplicación, logo de la aplicación, api de Android objetivo de la aplicación, permisos que tiene la aplicación y más.

La comprobación del acceso a internet no se considera un permiso con riesgo, es decir, no requiere del permiso del usuario para acceder al estado de la conexión a internet, como sí puede ser el acceso a la cámara o la geolocalización del dispositivo.

Para guardar el estado de la conexión, he creado una interfaz que aporta la funcionalidad para observar los cambios en la conexión y los muestra como un objeto que puede tener 4 estados distintos: "Available", "Losing", "Lost" y "Unavailable" aunque normalmente solo nos interesan dos estados, "Available" y "Unavailable".

Esta interfaz tiene una función observe que devuelve un objeto de tipo "Flow".

Para entender el funcionamiento de esta interfaz, hay que entender qué es un "Flow" de Kotlin. Un Flow es una estructura que ayuda a la asincronía en Kotlin, almacena los valores que toma un objeto desde el momento en el que se suscribe al Flow. Así, suscribiéndose a los cambios en el Flow de conexión, se pueden realizar acciones cuando el valor de la conexión cambia, por ejemplo, cuando pase de "Available" a "Unavailable" debería mostrar un mensaje de error indicando que se ha perdido la conexión.

El Flow debe estar dentro de una corrutina, que es por así decirlo, un hilo separado del hilo principal de Kotlin. Cuando esta corrutina se detiene, se cancela la suscripción al Flow automáticamente para así no gastar recursos. Normalmente se añade el Flow a la corrutina de la pantalla donde se requiere de la comprobación de la conectividad, para que al cambiar de pantalla se cancele el Flow de la conectividad y así se ahorre en recursos.

5.2.2. Lógica de la geolocalización

La geolocalización del dispositivo es un permiso de riesgo, el usuario debe indicar si quiere dar o no permisos para que la aplicación pueda servirse de su geolocalización.

Además de añadir este permiso en el manifiesto, he tenido que manejar los eventos relacionados con realizar la petición al usuario para dar permisos y según el resultado que se obtiene de la interacción del usuario, mostrar unas funcionalidades o no.

La localización del dispositivo funciona de forma similar a la conectividad, aunque toda esta funcionalidad relacionada con el Flow y la suscripción a los cambios en la posición del dispositivo están ya cubiertos con la biblioteca "FusedLocationProviderClient". Con esta tecnología, se puede suscribir a los cambios en geolocalización del dispositivo y definir una serie de acciones a realizar cuando el usuario se mueve, por ejemplo, 50 metros.

Con esta funcionalidad es con la que he podido realizar consultas a los distintos servicios web para obtener paradas desde la posición del dispositivo en un radio de X metros. Cuando la localización se mueve mucho, se vuelve a realizar una consulta para que esté la información en pantalla actualizada a la posición del usuario.

En esta zona de la implementación se encuentran también las funcionalidades relacionadas con el "geocoding". Esta tecnología se encarga de traducir las direcciones en posiciones geográficas con latitud y longitud y viceversa, de posiciones geográficas en direcciones que puede entender el usuario.

5.2.3. Lógica detrás de la configuración

La pantalla de configuración se ha creado de manera semi-automática gracias a las facilidades que aporta Android Studio y su clase "PreferenceDataStore". Extendiendo de esta clase, se puede guardar información en el dispositivo y también se puede extraer esta información.

Las interfaces que se utilizarán en toda la aplicación como repositorio de configuración, permiten actualizar y extraer la información almacenada con el "PreferenceDataStore"

En la configuración se almacenan los datos siguiendo un par clave-valor. Al usuario sólo se muestra el valor almacenado en la clave y el sistema utiliza las claves para modificar o mostrar los valores que almacenan.

En la configuración de *ValenMove* solo hay dos apartados de configuración, uno para ajustar el radio de cercanía que se trata de una lista de valores que va de 100, 200, 500 metros hasta 5000 metros y otro apartado para gestionar el idioma que utiliza la aplicación, entre español, inglés y valenciano.

Cuando se cambia un valor utilizando esta pantalla de configuración, el nuevo valor persiste sin modificar incluso cerrando la aplicación hasta que el usuario vuelve a cambiar el valor. Permitiendo así que el usuario no deba acceder mucho a la pantalla de configuración después de escoger la configuración deseada.

5.2.4. Lógica para el almacenamiento de paradas en favoritos

Esta funcionalidad requiere de la biblioteca de nombre "Room". Las paradas almacenadas se guardan con la misma estructura: un identificador, un nombre, las líneas si tiene, una propiedad de próximas llegadas, y el tipo de transporte que almacena.

En la tabla de paradas donde se almacena toda esta información tengo como clave primaria el identificador (utiliza el mismo que el que se obtiene por medio de las consultas a los servicios) y el tipo de transporte, para que se pueda diferenciar si existen dos objetos con mismo identificador pero distinto tipo de transporte.

La herencia utilizando Room es una tarea muy compleja, se ha optado por esta propiedad que almacena el tipo de transporte para simplificar la complejidad de esta funcionalidad. Así, podemos almacenar dentro de la misma tabla, paradas de EMT, metrovalencia y valenbisi.

Aunque valenbisi no tiene las mismas propiedades que metrovalencia o EMT, tampoco tiene sentido almacenar las propiedades que proporciona el API como las bicis disponibles o los espacios libres, pues son datos que no van a ser actualizados a menos que se realice otra consulta. Mediante el identificador de Valenbisi, se puede realizar una consulta al servicio en busca de ese estacionamiento para así obtener la información actualizada. La única información relevante de ser almacenada por parte del servicio de valenbisi es el identificador, el nombre y la geolocalización, que son valores que sí están en el esquema general de la base de datos.

Con la creación de la tabla y sus propiedades, puedo dar el siguiente paso para crear el DAO (Data Access Object) encargado de proporcionar todas las funcionalidades típicas dentro de una base de datos que son: Insertar, borrar, consultar y actualizar los datos. Room proporciona todas las facilidades posibles para conseguir esta funcionalidad con muy pocas líneas de código, simplemente utilizando anotaciones, como se puede observar en la figura 37.

```kotlin
@Dao
interface FavouritesDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun create(stop: DatabaseStopDto)

    @Delete
    suspend fun delete(stop: DatabaseStopDto)

    @Query("SELECT * FROM $TABLE_NAME")
    fun getAllFavourites(): Flow<List<DatabaseStopDto>>

    @Query("SELECT * FROM $TABLE_NAME WHERE $COLUMN_ID = :id")
    fun getFavById(id: Long): Flow<DatabaseStopDto?>

    @Query("SELECT * FROM $TABLE_NAME WHERE $COLUMN_STOP_TYPE = :source")
    fun getFavBySource(source: String): Flow<List<DatabaseStopDto>>

    @Query("DELETE FROM $TABLE_NAME")
    suspend fun deleteAll()
```

Figura 37: DAO utilizado en la solución

En el DAO utilizado en favoritos, hay métodos para insertar datos en las tablas, para borrar una o todas las entradas de la tabla y las distintas consultas que pueden ser: obtener todos los favoritos, obtener favoritos por identificador y obtener favoritos por tipo de transporte. Con toda esta funcionalidad se consigue la persistencia de la información que el usuario almacena en favoritos.

A través de la implementación del DAO, el repositorio de favoritos se encarga de hacer llegar los datos a la capa de vista-modelo (*ViewModel*) de forma que pueda tratarlos correctamente.

5.3. Lógica detrás de la información detallada de las paradas

En la pantalla donde se puede ver la información detallada de las paradas hay que transformar una URL, que se obtiene a través de la propiedad próximas_llegadas, en una lista de transportes próximos a llegar a la parada, su destino y el tiempo que van a tardar en llegar.

Para obtener información sobre las llegadas próximas a una parada, ya sea de metrovalencia o de EMT, tengo que utilizar la biblioteca “JSoup” [32] para extraer información de la página web que se obtiene a partir de la URL.

De las páginas web sobre llegadas, necesito extraer la línea de cada llegada, el tiempo que se demora en llegar y el destino (en el caso de metrovalencia). Para ello, creé un objeto, que tiene estas propiedades: “line” (línea), “waitTime”(tiempo de espera), “destination”(destino).

Tras analizar el código HTML de las páginas web, he tenido que hacer dos funciones distintas, una para EMT y otra para metrovalencia, pues presentan diferencias en la estructura. Estos métodos consiguen que, al pasarle una URL en cadena de texto, con la

ayuda de JSoup, devuelven una lista de objetos con la estructura descrita anteriormente de línea, tiempo de espera y destino.

A través del repositorio encontrado en esta área del código la capa de presentación puede obtener la lista de llegadas para mostrarla por pantalla por medio de un "RecyclerView" (componente de android para mostrar listas por pantalla)

Para valenbisi es mucho más sencillo pues toda la información sobre disponibilidad se encuentra almacenada en el servicio web de valenbisi, no requiere de este paso después de la consulta para obtener información sobre el estacionamiento de bicicletas.

5.4. Implementación de la capa de presentación

En esta capa existen dos componentes clave, la vista y el componente vista-modelo que hace de mediador entre la vista y el modelo de datos. Voy a explicar por separado cada uno y a exponer cómo se ha utilizado en cada caso de las pantallas de la aplicación.

5.4.1. Implementación de la vista de forma general

La vista se compone de varios componentes distintos. En primer lugar, debe encontrarse dentro de una actividad. Dentro de esta actividad puede o no haber Fragmentos, estos se encuentran contenidos en un "FragmentContainerView". El contenedor de fragmentos más importante de la aplicación es el que se encuentra en la actividad principal y se encarga de mostrar cada pantalla de la barra de navegación (Buscar, Favoritos y Mapa).

A partir de aquí, la implementación de las vistas es controlada directamente por los fragmentos, en vez de las actividades que solo se sirven como contenedores de fragmentos en la práctica.

Cada fragmento se conecta con su "ViewModel" y se suscribe a todas las propiedades públicas del "ViewModel" para realizar acciones en base a los cambios en el estado de los objetos que almacena el "ViewModel".

La vista controla el estado de los componentes de la interfaz como pueden ser los botones, las entradas de texto, los campos con texto, las imágenes por pantalla, crea los adaptadores de listas y los adjunta a las listas de la pantalla ("RecyclerView"), controla los botones de la barra superior de la pantalla y mucho más.

5.4.2. Implementación de los *ViewModel* de forma general

Los componentes vista-modelo juegan un papel muy importante en la presentación de los datos obtenidos por la capa de modelo en la interfaz. Estas clases se encargan de mostrar los datos a la vista y actualizar los datos en función de las interacciones del usuario y los cambios en el modelo, para volver a presentar los datos a la vista si han sido cambiados.

Se caracteriza por tener dos tipos de propiedades, unas privadas y mutables que almacenan los datos obtenidos por los repositorios de la aplicación y otras públicas e inmutables que son los que observa la vista. Así, la vista no puede modificar los datos y solo puede recibirlos y realizar acciones en base a los cambios observados.

He hablado ya de los "Flow", que son *cold streams*, es decir, los datos se generan cuando se suscribe alguien a ellos. En los ViewModel de mi solución, sin embargo, utilizo otra tecnología que es similar, los "StateFlow", que se tratan de *hot streams*, es decir, los datos emitidos por este flujo de datos están disponibles de forma inmediata para cualquier nuevo suscriptor, de esta forma, se comparte un único flujo de datos entre todos los suscriptores. Los suscriptores son los fragmentos que observan los cambios en estas propiedades.

Cuando la vista-modelo requiere cambiar un dato que tiene almacenado, modifica sus propiedades privadas del tipo "MutableStateFlow", esto cambia el valor que se almacena en las propiedades públicas de tipo "StateFlow" que envían el cambio en los datos a los suscriptores, es decir a los fragmentos que estén observando estas propiedades públicas.

5.4.3. Implementación de la pantalla de búsqueda

Esta pantalla es sin duda la más compleja de toda la aplicación. La complejidad de esta pantalla se resume en la mezcla de por un lado el muestreo de los datos obtenidos a través de los servicios web utilizando la biblioteca de "paging v3" y, por otro lado, la gestión de los filtros de búsqueda por nombre, cercanía y el cambio entre tipos de transporte.

Para entender bien esta pantalla, hay que entender primero la estructura de la interfaz para la misma.

Esta pantalla cuenta con una barra de acción en la parte superior, donde se encontrará el botón para buscar por nombre, que despliega un campo para introducir texto y el botón para filtrar los datos obtenidos por cercanía o para obtener todas las paradas.

Como se puede ver en el boceto de las funcionalidades que debe tener esta pantalla en la figura 38, debajo de la barra de acción, se encuentra un componente para cambiar entre tipos de transporte que es un "TabLayout". El "TabLayout" no es más que una vista para cambiar entre pestañas controladas por fragmentos distintos, en este caso, fragmentos que manejan las funcionalidades para mostrar una lista de datos de EMT, metrovalencia o valenbisi.

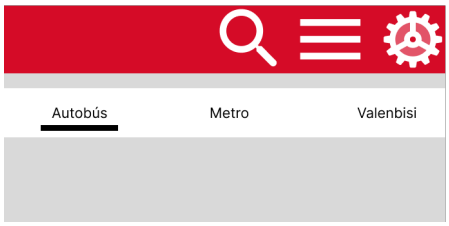

Figura 38: Boceto de las funcionalidades de la pantalla buscar

Se puede observar de forma gráfica la conexión entre los componentes que actúan en esta pantalla en la figura 39.

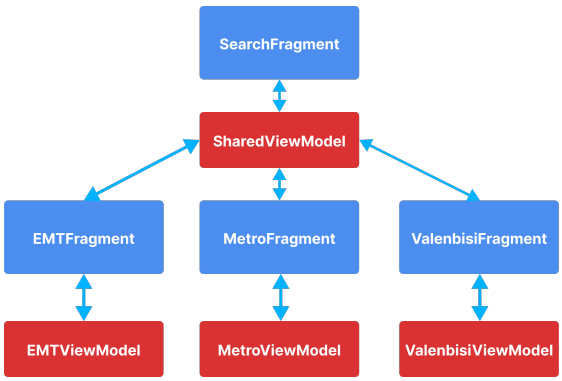

Figura 39: Estructura de la pantalla buscar

En esta funcionalidad de buscar existe una pantalla general, que tiene las funcionalidades para buscar por nombre, buscar todas las paradas y buscar por cercanía. Dentro de la pantalla general tengo fragmentos que muestran las listas de cada tipo de transporte dependiendo del filtro de búsqueda seleccionado en la pantalla general. Estos son los fragmentos "EMTFragment", "MetroFragment" y "ValenbisiFragment".

La pantalla tiene un fragmento que controla la funcionalidad general de la pantalla, que se llama "SearchFragment". Este fragmento tiene un ViewModel al que se suscribirán los demás fragmentos para observar los cambios aplicados a los filtros de búsqueda. Por ejemplo, si se buscan las paradas por nombre de parada, esto lo almacena el componente vista-modelo que es compartido por todos los fragmentos, de esta forma todos los fragmentos observan que se aplicó el mismo filtro de buscar por nombre.

El ViewModel compartido también almacena los posibles errores que aparezcan en la pantalla para que todos los fragmentos de la pantalla puedan observar que apareció un error.

Cada fragmento en la pantalla tiene su propio *ViewModel*, que maneja la funcionalidad concreta de cada transporte, por ejemplo, EMTFragment tiene su "EMTViewModel", el fragmento se conecta con el ViewModel para suscribirse a la lista que va a mostrar por pantalla paradas de autobús. Es el ViewModel el que se encarga de llamar al repositorio de EMT para buscar en el servicio web con los filtros específicos y obtener así las paradas de autobús.

En general, el funcionamiento de cada fragmento concreto que muestra la lista de paradas por tipo de transporte es el siguiente:

- Crea el adaptador que se encarga de gestionar los datos en forma de listas para luego ser mostrados por la interfaz y lo inserta en el componente "RecyclerView" que es el que se encarga de mostrar la lista por la interfaz. El RecyclerView muestra la lista y necesita un adaptador que se encarga de proveer los datos al RecyclerView.
- Se suscribe a los cambios en el *ViewModel* general, para observar cambios en los

estados de los filtros de búsqueda y en la aparición de errores.

- A partir del *ViewModel* compartido avisa al *ViewModel* individual del estado de los filtros de búsqueda
- Se suscribe a su *ViewModel* individual por el cual obtendrá la lista de datos del servicio web a partir del filtro de búsqueda almacenado en el *ViewModel* compartido.

5.4.4. Implementación de la pantalla de favoritos

Esta pantalla tiene dos botones en el menú de acción en la parte superior de la pantalla además del botón para acceder a ajustes. Estos dos botones se encargan de filtrar entre los tipos de transporte y eliminar todas las entradas de favoritos que se encuentren en la pantalla.

Para esta pantalla ha sido mucho más sencilla la estructura de la implementación, requiriendo únicamente un fragmento, un vista-modelo, un adaptador y una pantalla de diálogo para confirmar la eliminación de todas las entradas.

El fragmento (Vista) crea el adaptador que maneja las interacciones realizadas en la lista que se muestra por medio del componente "RecyclerView". Inserta los dos botones que requiere la pantalla de favoritos en la barra de acción y maneja su lógica de eventos (cuando se pulsa un botón, qué acciones deben ocurrir en la pantalla). Maneja la lógica para poder eliminar las paradas guardadas en favoritos deslizando hacia la derecha y así enviar la orden de eliminar la parada concreta al *ViewModel*. Finalmente, se suscribe a la propiedad pública que controla la lista de objetos que se muestran por pantalla para pasar esos objetos al "RecyclerView".

El *ViewModel* se comunica con el repositorio de favoritos para mandar las acciones de realizar consultas por transporte, por identificador, mostrar todos los favoritos y para eliminar una o varias paradas. Cuando se selecciona un tipo de transporte, este componente realiza una consulta a la base de datos por medio de su repositorio para obtener las paradas guardadas en favoritos de ese tipo de transporte. Este resultado cambia la propiedad que observa la vista y así es como se notifica a la interfaz que debe mostrar una lista distinta de datos.

La vista se encarga de mostrar un mensaje por pantalla si no se obtiene ningún dato a través del *ViewModel*.

El botón de eliminar todas las entradas, al tener tanto riesgo de ser pulsado por error y borrar la persistencia de la aplicación, muestra un componente diálogo que permite al usuario confirmar o descartar el borrado de todas las entradas. Además, dependiendo del tipo de transporte que sea seleccionado, se borrarán únicamente las entradas de ese tipo (o todas las entradas si se ha seleccionado el filtro de obtener todas las paradas favoritas).

5.4.5. Implementación de la pantalla del mapa

Para la pantalla del mapa se necesita un enfoque totalmente distinto. Esta pantalla se centra únicamente en utilizar las características de la biblioteca de Google Maps. El funcionamiento general gira en torno a los marcadores que nos proporciona esta tecnología y que son simplemente unos objetos que puedes situar en el mapa conociendo la latitud y longitud donde deben ir posicionados. Estos marcadores no almacenan ningún tipo de información más, lo cual complica el acceso a la información detallada de las paradas.

El ViewModel se encarga de buscar las paradas cercanas a la ubicación que proporciona la Vista. Esta ubicación se trata del centro del mapa o como se conoce en Google Maps, la posición de la cámara en el mapa. Cuando obtiene el resultado, cambia la propiedad pública que almacena esta lista de paradas cercanas y es enviada a la Vista.

Para solventar el problema de los marcadores que no pueden almacenar mucha información, he creado una lista de paradas que se han mostrado en el mapa, cada vez que se crea un marcador se añade la parada que representa el marcador a esta lista. De esta forma, se puede mantener la información de cada marcador con la posición en la lista que almacena el marcador como cadena de texto. Así, cuando se pulsa sobre un marcador se muestra la parada en concreto a la que va dirigido ese marcador en la parte inferior de la pantalla.

Esta pantalla también permite esconder o mostrar los marcadores por tipo de transporte. Para lograr esto he creado 3 listas de marcadores donde se guardan los marcadores que se van creando según el tipo de transporte. Cuando se quieren esconder los marcadores de EMT, simplemente hay que recorrer la lista de marcadores de EMT y marcarlos como escondidos.

Otra funcionalidad que se ha aplicado a esta pantalla es la búsqueda de zonas geográficas en el mapa por medio de su dirección. Para ello necesito del servicio de "geocoding" de Google que permite traducir una posición con latitud y longitud en una dirección que el humano entiende y viceversa. Así, mediante un buscador en la zona superior de la pantalla, el usuario puede escribir una dirección y la cámara del mapa se posicionará en esa misma dirección.

El mapa debe manejar la situación cuando se accede al mismo a través de la pantalla de información detallada. Esta pantalla envía la dirección de la parada seleccionada a la pantalla del mapa y este se encarga de situar la cámara del mapa en la posición obtenida.

5.4.6. Implementación de la pantalla de ajustes

Esta implementación de la capa de presentación ha sido la más sencilla de todas pues se hace de manera casi automática junto con su implementación de la capa del modelo en el apartado 5.2.3.

Lo más complejo de esta pantalla es el cambio de idioma de la aplicación, que utiliza

los ajustes propios del dispositivo que permiten cambiar el lenguaje de manera individual por aplicación. La parte de cambiar el radio se hace de manera automática con la estructura de clases de esta parte del desarrollo. La interfaz de la pantalla es semi-automática siguiendo un archivo de preferencias en XML y creando el fragmento que controla este archivo extendiendo la clase "PreferenceFragmentCompat".

5.4.7. Implementación de la pantalla de información detallada

A diferencia de las pantallas principales de la aplicación, esta tiene una actividad aparte pues se encuentra fuera del flujo de navegación común de la barra de navegación inferior de la pantalla. Para acceder a esta pantalla se debe pulsar sobre cualquier parada mostrada en la aplicación y pasarle la información sobre la parada a esta misma pantalla.

En la pantalla de información detallada, la vista debe controlar el botón que se encuentra en la barra de acción para refrescar la información en pantalla. Este botón, al ser pulsado, envía la acción al *ViewModel* que vuelve a obtener la información sobre llegadas de EMT o metrovalencia. En el caso de valenbisi, funciona de forma distinta pues no existe una página web externa que proporcione información sobre este servicio como sí ocurre con EMT y metrovalencia. Simplemente, se vuelve a hacer una consulta al servicio con el filtro de búsqueda, el identificador de la parada seleccionada para volver a obtener la información sobre la disponibilidad de las bicicletas en el estacionamiento. Cuando se vuelve a obtener la información a través del *ViewModel*, se notifica el cambio a la vista para actualizar la información mostrada por pantalla.

La Vista debe inicializar el componente donde se encuentra la imagen de la parada con "streetView" de Google Maps. Para ello, cuenta con la localización de la parada que ha sido seleccionada y se vuelve una tarea relativamente sencilla gracias a la biblioteca de Google Maps que, simplemente indicando la posición geográfica y adjuntando la imagen al componente donde se debe mostrar en la pantalla se puede mostrar el Street View de la dirección de la parada obtenida por el servicio web.

En esta pantalla se encuentra la funcionalidad para añadir paradas a favoritos. Para lograr este cometido hace falta que el *ViewModel* dependa del repositorio de favoritos y así conseguir la función encargada de insertar paradas en la tabla. A continuación, simplemente se pasa a esta función la parada seleccionada y así se inserta en la tabla. Además, se debe buscar en la tabla para ver si ya existe la parada seleccionada y mostrar o no el botón de añadir a favoritos en consecuencia.

Esta pantalla se debe comunicar con la pantalla del mapa para que al pulsar sobre el botón de ir al mapa, aparezca el mapa situado sobre la posición de la parada. Para conseguir esto, se llama a la pantalla del mapa pasándole por parámetros la geolocalización de la parada.

6. Resultados del desarrollo de *ValenMove*

Después de realizar el desarrollo de la aplicación llego al primer MVP (Mínimo producto viable, del inglés Minimum Viable Product) donde se observan las funcionalidades mínimas necesarias para la primera versión del producto. He realizado un desarrollo que ha cubierto con toda la especificación de funcionalidades que se comentaron en el capítulo 3. En este capítulo se van a mostrar los resultados finales en cuanto a interfaces y funcionalidades conseguidas tras el desarrollo del proyecto.

6.1. Resultado final de la pantalla de búsqueda

Esta pantalla cubre todas las funcionalidades relacionadas con buscar paradas y aplicar filtros de búsqueda para encontrar paradas tanto de autobús, como de metro o de valenbisi.

Figura 40: Resultado final de la pantalla buscar

Como se puede observar en la figura 40, las funcionalidades para cambiar el filtro de búsqueda se encuentran en la barra de acción superior de la pantalla. Se puede buscar por nombre y por cercanía, cumpliendo los casos de uso de las tablas 7 y 9. Cuando se pulsa sobre la lupa de buscar por nombre, se despliega un campo para introducir el nombre de la parada a buscar, quedando en una interfaz limpia, bonita y útil para el usuario.

Además, se pueden diferenciar los tipos de transporte y se puede cambiar entre el tipo de transporte pulsando en la pestaña correspondiente o, incluso deslizando con el dedo entre las ventanas, cumpliendo así con el caso de uso especificado en la tabla 3.

Se puede observar también el radio de cercanía seleccionado en un campo de texto encima de las pestañas de los transportes, para que el usuario no tenga que ir a la pantalla de ajustes a recordar el radio que ha seleccionado.

6.2. Resultado final de la pantalla de información detallada

Al pulsar sobre cualquier parada se accede a la pantalla que muestra información sobre la parada seleccionada. Se puede ver en la figura 41 el resultado final en la aplicación de esta pantalla.

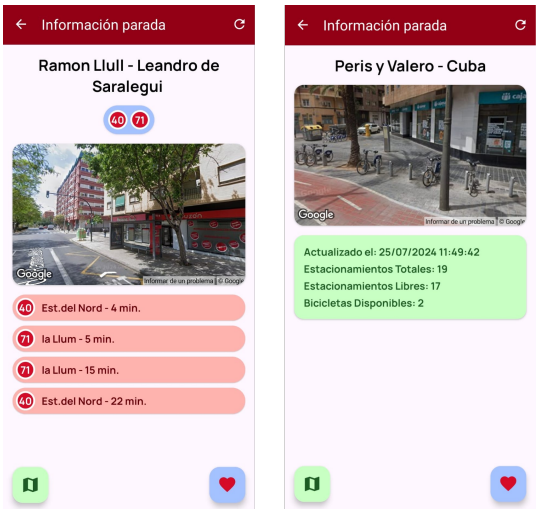

Figura 41: Resultado final de la pantalla información detallada

A la izquierda, la información de una parada de autobús, a la derecha, de un estacionamiento de bicicletas.

En esta pantalla se muestran el nombre de la parada, las líneas que circulan por la parada, las llegadas y la información sobre la disponibilidad de valenbisi en el caso de valenbisi.

Esta pantalla cuenta con un botón para actualizar la información que se muestra por la interfaz en la esquina superior derecha, así como también las funcionalidades para ir al mapa y para guardar la parada en favoritos cumpliendo con los casos de uso de las tablas 8 y 5 respectivamente.

Se observa también, cómo funciona el StreetView de Google Maps que muestra la zona geográfica de la posición de la parada seleccionada. De esta forma, el usuario tiene información de valor que le puede ayudar a encontrar la parada en concreto.

6.3. Resultado final de la pantalla de configuración

Esta pantalla es bastante simple pues solo cuenta con dos apartados que el usuario puede ajustar, estos son la configuración del idioma y del radio de cercanía. Se puede ver en la figura 42 cómo han quedado plasmadas estas dos características en el resultado final de la pantalla.

Figura 42: Resultado final de la pantalla de configuración

Cuando el usuario ajusta algún valor de esta pantalla, el resultado se mantiene invariable incluso cerrando la aplicación.

6.4. Resultado final de la pantalla de favoritos

La pantalla de favoritos ha sufrido algunos cambios en relación al boceto original, para disponer del máximo de información en la pantalla, separando la funcionalidad para filtrar la lista de favoritos por tipo de transporte en la barra de acción superior de la pantalla. El resultado final de esta pantalla se observa en la figura 43.

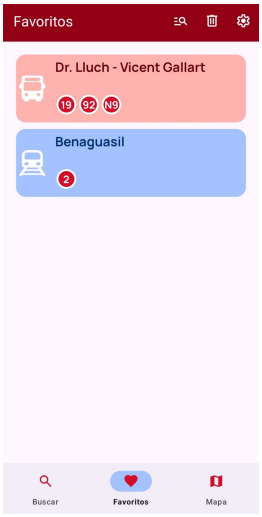

Figura 43: Resultado final de la pantalla de favoritos.

Se puede observar que la pantalla se dispone únicamente para mostrar las paradas guardadas en favoritos, cumpliendo así con el caso de uso 11 detallado en la tabla 12 y

se utiliza el espacio de la barra de acción para las funcionalidades relacionadas con los casos de uso 12 y 13, de las tablas 13 y 14, para filtrar y borrar las entradas de favoritos.

Se pueden eliminar las paradas guardadas en favoritos deslizando la parada concreta hacia la derecha.

6.5. Resultado final de la pantalla del mapa

Esta pantalla ha recibido varias mejoras en relación a la especificación realizada. Se ha añadido una sección con botones para esconder o mostrar las paradas de cualquier tipo de transporte. Así el usuario puede decidir si solo quiere ver las paradas de autobús, por ejemplo.

En adición a la funcionalidad para esconder las paradas, se ha habilitado en la barra de acción un buscador de localizaciones para situar el mapa en una dirección concreta que indique el usuario. Todas estas mejoras y el resultado final de la pantalla se puede ver en la figura 44.

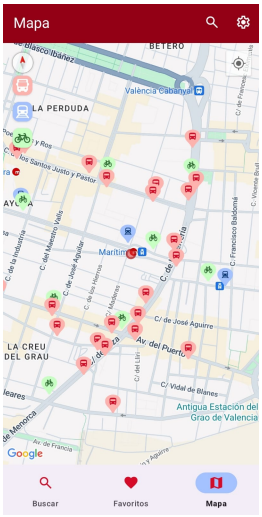

Figura 44: Resultado final de la pantalla del mapa.

6.6. Disponibilidad en horizontal de las pantallas

La aplicación está disponible en ambas orientaciones de pantalla, tanto vertical como horizontal. Para conseguir que las pantallas se vean bien en horizontal se ha necesitado adaptar un poco la pantalla. La barra de navegación que normalmente se encuentra en la zona inferior de la pantalla, ahora está dispuesta como una barra vertical a la izquierda de la pantalla. Además de este cambio, la pantalla de detalles de las paradas ha sufrido un cambio en la disposición de la lista y el StreetView, que ahora se encuentran en la misma línea horizontal ocupando cada uno la mitad de la pantalla, como se puede ver en la figura 45 además de cómo han quedado las demás pantallas en esta disposición.

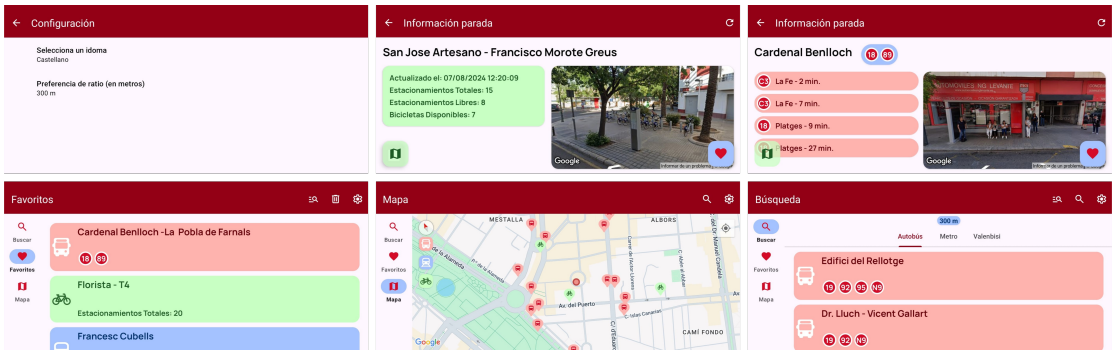

Figura 45: Resultado final de todas las pantallas dispuestas de manera horizontal.

Además de las 5 pantallas bien diferenciadas de la aplicación, en la figura 45, se puede observar la disposición de la pantalla de detalles de valenbisi y de emt o metro, pues esta pantalla presenta diferencias en cuanto a la información mostrada dependiendo del tipo de transporte.

6.7. Resultado final de la internacionalización

Uno de los objetivos de mi proyecto era que la aplicación pudiera ser utilizada por muchos usuarios sin tener en cuenta la barrera que presenta el idioma. Para ello, la aplicación se dispone en 3 idiomas distintos: el castellano, el valenciano y el inglés. Para cambiar de idioma la aplicación, se puede utilizar la funcionalidad situada en la pantalla de configuración que permite al usuario ajustar el lenguaje de la aplicación a su gusto. Para observar las diferencias entre todos los idiomas, se dispone de la figura 46 que muestra las diferencias principales entre estos 3 lenguajes en la pantalla de buscar.

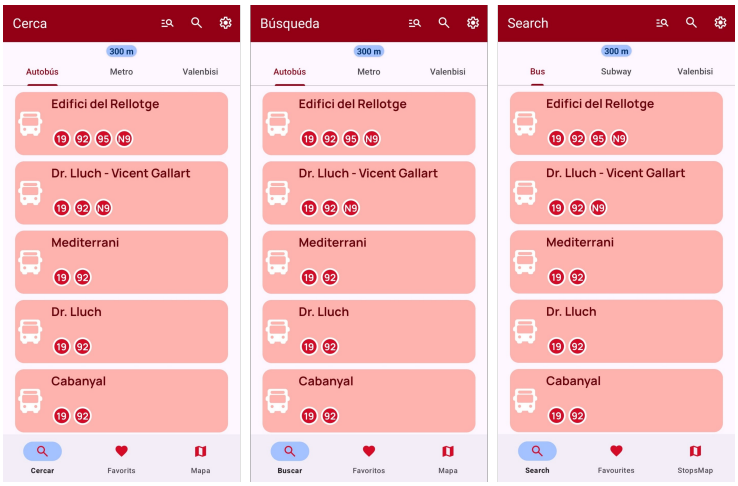

Figura 46: Diferencias entre idiomas de la aplicación en la pantalla de buscar.

Se puede observar que las paradas conservan su nombre original en valenciano para no generar confusiones al usuario a la hora de encontrar la parada en la vida real.

7. Implantación y pruebas

En este capítulo se resumen dos tareas muy importantes del desarrollo de un proyecto software, la implantación y las pruebas. La implantación se refiere a las tareas necesarias para poder desplegar y entregar la solución a los usuarios. Las pruebas son todas aquellas tareas que sirven para medir el correcto funcionamiento de la aplicación. En este caso, utilizaré el estándar de pruebas que deben pasar las aplicaciones antes de subirse a la tienda de "Google Play Store"[10], conocido con el nombre de "*Core App Quality*"[37] y realizaré pruebas con usuarios reales para obtener la opinión de otras personas sobre el funcionamiento de mi solución.

7.1. Implantación

La implantación en Android viene de la mano de los APK (Android Package Kit), se trata del formato utilizado por Android para empaquetar las aplicaciones con un instalador que consigue descargar la aplicación correctamente en el dispositivo.

Para realizar las pruebas de usabilidad con usuarios reales, necesitaré empaquetar el proyecto en un APK y distribuirlo entre los usuarios que realizarán la prueba.

Para poder distribuir un APK es necesario firmar el archivo con una clave privada, para garantizar que el fichero no ha sido alterado desde que se realizó la firma y para verificar la identidad del creador de la aplicación.

Para subir la aplicación a un portal de venta como puede ser "Google Play Store" se necesitará además de la firma, un "Android App Bundle" [38] que se trata de un formato para publicar aplicaciones creado por Google que se encarga de generar el APK de la aplicación más eficiente y correcto dependiendo del dispositivo. Yo no voy a subir la aplicación a ningún portal de ventas así que no necesito un *bundle*, con el apk me basta.

Los usuarios que reciben el APK simplemente deben abrir el archivo desde el dispositivo y confirmar la instalación de la aplicación mediante el administrador de paquetes de su dispositivo.

Utilizando *Android Studio* es muy sencilla la generación de APKs o Bundles. En el IDE existe una funcionalidad de nombre "Generate Signed Bundle / APK" que nos guía en el paso a paso para crear este tipo de ficheros con firma.

7.2. Guías de calidad de Android

La calidad es fundamental para las aplicaciones que se suben al portal de venta de "Google Play Store". Para poder medir la calidad del producto, Google proporciona unas guías con medidas de calidad para facilitar esta tarea. Estas guías son obligatorias

[10]La Google Play Store es la tienda donde se descargan todas las aplicaciones de Android, su página web oficial es: `https://play.google.com/store/`

para poder subir cualquier producto a la "Play Store". La guía que proporciona Google se llama "Core app quality"[37].

Estas guías tienen varios apartados para medir cualquier parte funcional de la aplicación. Estos apartados van desde valorar la interfaz de usuario, el funcionamiento, la eficiencia hasta la seguridad y privacidad, el audio de la aplicación, etc. Además de las guías para dispositivos móviles generales, existen otras guías para otro tipo de dispositivos, como tabletas, relojes, televisores, etc. Yo voy a guiarme únicamente por las pruebas para dispositivos generales.

Para mostrar las medidas de calidad que cumple la aplicación, voy a utilizar tablas que resumen estas métricas además de una pequeña explicación del proceso seguido para verificar el correcto cumplimiento del estándar.

La guía de calidad proporciona unos procedimientos de prueba que explican cómo se deben probar cada una de las medidas de calidad y tienen nombres como CR-01, GP-04, etc. Por ejemplo, la *CR-03* que aparece en muchas medidas relacionadas con el diseño visual, nos dice que se debe pulsar el botón para ir hacia atrás en todas las pantallas y diálogos de la aplicación, para garantizar que no existe ningún funcionamiento extraño al pulsar este botón.

He seguido todas las guías de calidad que aporta Google que creo importantes y aplicables al proyecto, estos son: Experiencia visual, rendimiento y estabilidad y privacidad y seguridad. Aspectos que no he probado son la sección de Google Play debido a que la aplicación no se va a subir a la plataforma y la sección de funcionalidad que se refiere a aspectos relacionados con la reproducción multimedia y de audio, características que no existen en mi aplicación.

7.2.1. Experiencia visual

El resultado de las pruebas realizadas en esta sección se puede observar en las tablas 15 y 16 de forma gráfica. Voy a describir los pasos realizados para seguir estas guías y el porqué de los resultados obtenidos.

Para empezar, en el área de navegación de la guía, se pide a las aplicaciones características relacionadas con el botón hacia atrás común de todos los dispositivos. Se han hecho pruebas desde todas las pantallas de la aplicación para observar que este botón funciona cómo se presupone desde cualquier punto de la aplicación. Lo mismo se ha replicado con la funcionalidad de gestos que existe en Android, concluyendo con un resultado favorable para esta prueba. Otra funcionalidad que se debe probar es la conservación del estado de la pantalla, esta funcionalidad está garantizada gracias al uso de *ViewModel* que permite almacenar el estado de la pantalla y devolverlo a la pantalla en el momento en que se vuelve a poner el foco sobre esta misma, dando con un resultado favorable también en esta prueba. Para probar la última prueba es algo más complejo pues hay que probar todas las funcionalidades que recargan la pantalla como son el volver hacia atrás, volver a la pantalla principal del dispositivo y rotar la pantalla con todas las pantallas de la aplicación.

El área de notificaciones no aplica a mi aplicación por el momento debido a que no

contiene esta funcionalidad.

El área de interfaz de usuario y gráficos se refiere a que la aplicación se debe ver bien en los modos horizontal y vertical y se debe tener una correcta conservación del estado de la pantalla cuando se cambia entre modos. Cuando cambias de vertical a horizontal, la pantalla actual muere y se vuelve a crear la pantalla en modo apaisado, si no se ha almacenado el estado de la pantalla se perderá la información en la misma. El *ViewModel* mantiene el estado de la pantalla permitiendo que el cambio de visualización de la misma parezca sutil y no altere la información que ve el usuario. La aplicación como ya se ha visto en el apartado 6.6, permite cambiar entre estos modos y se adapta para mostrar toda la información posible en pantalla. Para probar esto es necesario ir a todas las pantallas de la aplicación y rotar el dispositivo para observar si existe algún fallo.

El apartado de calidad visual tiene que ver con cómo se ven los elementos de la interfaz, si se ven borrosos, pixelados, poco nítidos, con deformaciones, se cortan, etc. Además, se debe probar para cualquier idioma pues pueden haber errores que no se ven en un idioma y en otro se corta el texto o se ve peor por ejemplo. La aplicación no admite el tema oscuro así que no pasa ese caso de prueba de la guía. Todas las imágenes de la aplicación son vectores o se adaptan al tamaño del dispositivo así que no existen imágenes que se vean mal en ningún dispositivo. Lo mismo ocurre con los textos de la aplicación y los demás componentes visuales.

Las pruebas para el apartado de calidad visual son más complejas pues requieren pasar todos los pasos para probar de la sección de pasos centrales de prueba (CR - Core suites). Existen pruebas como las vistas anteriormente como rotar la pantalla, volver hacia atrás, ir a la pantalla principal del dispositivo y existen otras pruebas como salir de la aplicación y entrar a otra para luego desde ajustes comprobar si se está ejecutando algo de la primera aplicación y muchas otras pruebas más.

El apartado de accesibilidad tiene más que ver con las facilidades que la aplicación aporta al usuario en términos de interfaz para poder interactuar con el sistema. Los botones deben tener un tamaño mínimo de 48 dp[11] que es el estándar para facilitar el pulsado del componente de botones y campos interaccionables en interfaces de usuario para dispositivos móviles. Un elemento interaccionable con menos de 48dp puede ser complicado para el usuario de pulsar y derivar en un usuario poco satisfecho con la interacción de la pantalla.

Los colores en la pantalla también son muy importantes para la interacción con el usuario. Si el contenedor y el contenido se parecen mucho en color puede haber confusiones para el usuario a la hora de diferenciar algunos elementos de la pantalla como se puede ver en la figura 47.

[11] Dp viene de Density-independent pixel y es una medida muy utilizada por Android que se refiere al tamaño de un píxel en una pantalla con densidad de píxeles de 160 píxeles por pulgada. Esto permite traducir cualquier tamaño a cualquier dispositivo independientemente de sus dimensiones o la densidad de píxeles de la pantalla.

Figura 47: Diferencias entre un contraste alto y uno bajo de contenedor y contenido.

Este contraste se respeta en toda la aplicación y no existen elementos cuyo contenedor y contenido tengan una mala relación de contraste. Para conseguir contrastes entre los colores utilicé la tecnología de "Material theme builder" que proporciona muchos colores y contrastes para los colores para diseñar la interfaz.

La siguiente prueba de esta área tiene que ver con que todos los elementos distintos a campos de texto, es decir, botones, imágenes, gráficos, fragmentos, etc, deben tener un texto descriptivo que indique para qué sirven en su propiedad "contentDescription". Esta propiedad permite a los elementos tener una descripción en forma de texto que puede ser leída por lectores de pantalla como TalkBack de Android [39], por ejemplo, para ayudar a personas con discapacidades visuales. Ha sido comprobado que todos los elementos tienen esta propiedad "contentDescription" con alguna descripción del elemento en sí, a excepción de elementos meramente decorativos que no requieren de una descripción.

Para probar el área de accesibilidad se han seguido los mismos pasos que en el área de calidad visual, es decir, realizar todas las pruebas sugeridas por Google de nombre Core suites (CR).

7.2.2. Rendimiento y estabilidad

Para esta sección han sido necesarias herramientas como la creación de perfiles de Android Studio para hacer un seguimiento del rendimiento de la aplicación, así como también herramientas de desarrollador dentro del dispositivo donde se ha probado todo.

En el área de estabilidad nos encontramos con una característica crucial para cualquier aplicación que se despliegue y es el hecho de minimizar al máximo posible los fallos de la aplicación que producen el cierre de la misma. La aplicación durante las pruebas no ha presentado ningún fallo aunque para probar esta característica ha sido necesario realizar todos los procedimientos de prueba centrales así como también descargar la aplicación en una tarjeta SD y probar los mismos procedimientos para comprobar que no falla nada. Finalmente, el resultado es favorable.

El área de rendimiento es el que más problemas me ha dado pues la aplicación, aún siguiendo buenas prácticas de programación, no consigue pasar algunos mínimos muy exigentes que pide esta guía.

La aplicación carga muy rápido, siguiendo las pruebas que propone la guía, mediante el uso del terminal del ordenador y un dispositivo conectado, se puede contabilizar el tiempo que tarda en cargar la aplicación. Como se ve en la figura 48, los tiempos son inferiores a 2 segundos así que no requiere de nigún comentario al usuario que aporte

información sobre la carga de la aplicación.

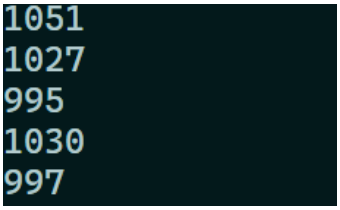

Figura 48: Tiempos capturados en milisegundos para la prueba de carga de la aplicación

Aunque el tiempo de carga de la aplicación sea bajo, existen otros tiempos que sí llegan a ser altos. Estos tiempos son los relacionados a lo que se tarda en obtener los resultados de los diferentes servicios web de transporte del Ayuntamiento de Valencia, que dependen del tiempo de descarga de datos de internet. Para estas cargas, se ha introducido un icono que gira que indica que se está cargando la información. Así el usuario siente que está ocurriendo algo en el sistema y por eso está tardando.

Para el siguiente apartado dentro de rendimiento, se requiere usar una herramienta de desarrollador de los dispositivos de Android. Esta herramienta es el "perfil de renderización HWUI", que se trata de unas gráficas en pantalla que muestran el tiempo de renderizado de los elementos en pantalla con segmentos para diferenciar qué está causando cuellos de botella, si es el dibujado en pantalla, el intercambio de búferes, la emisión de comandos, etc. Se puede ver en la figura 49 que estas gráficas tienen varios colores. Lo importante es respetar que el renderizado no supere los 16 milisegundos, este tiempo se puede ver en la línea horizontal de color verde. Cualquier valor por encima de esta línea no está respetando estos tiempos.

Figura 49: Gráficas representativas de la renderización de la pantalla en tiempo real

Es importante destacar que mi aplicación tiene muchas listas, elementos que son complicados de mostrar a unos fotogramas aceptables por esta guía. El problema llega en el momento que se deslizan las listas, pues es muy complejo que vaya a 60fps fluidos. Aún así, el problema es muy bajo y no es apreciable con la vista humana.

La siguiente característica del área de rendimiento tiene que ver con una funcionalidad que tiene las aplicaciones para observar latencias inadecuadas según los estándares. Esto tiene que ver con un uso muy alto del hilo principal de la aplicación. Con el StrictMode encendido, aparecen destellos de color rojo en la pantalla cuando detecta algún tiempo inadecuado. Esto ocurre generalmente cuando se cambia entre las pantallas, debido al tiempo que se tarda en obtener información de los servicios web para mostrar las listas. Esto quiere decir que la aplicación no es inmediata o tiene un tiempo de respuesta un poco bajo según los estándares.

El área de SDK tiene que ver con la versión del kit de desarrollo software. Cada versión de Android viene con un kit de desarrollo con nuevas funcionalidades o mejoras de las anteriores funcionalidades que ayudan a los desarrolladores a crear aplicaciones para dispositivos Android. Esta área la pasa *ValenMove* de forma totalmente favorable pues está actualizada a la última versión de todas las bibliotecas que utiliza y de los kits de desarrollo de Android. Se han comprobado todas las versiones de las bibliotecas de la aplicación, la versión objetivo de la misma, la versión de compilación del proyecto y se ha probado la aplicación por medio de un emulador en la última versión de la SDK de Android, que al tiempo de este proyecto es el API 34 de Android.

7.2.3. Privacidad y seguridad

En esta guía solo se han podido aplicar las pruebas relacionadas con los permisos, pues mi aplicación no tiene datos sensibles ni servicios en segundo plano ni tampoco uso de redes de servidores ni las demás áreas de esta sección de privacidad y seguridad.

En mi aplicación el usuario sólo tiene que aceptar un permiso si lo desea, que es el uso de la localización precisa o poco precisa. Si el usuario accede a las funcionalidades de buscar paradas cercanas o el mapa, aparece el diálogo para aceptar los permisos como se puede ver en la figura 50.

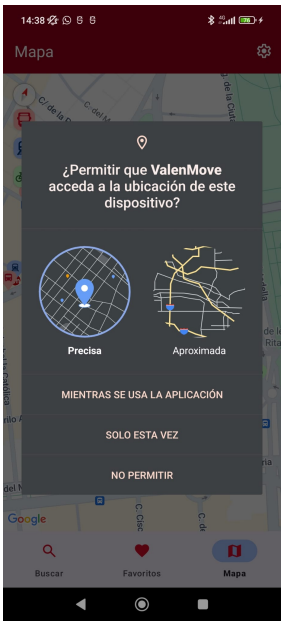

Figura 50: Diálogo para dar permisos de localización.

Si el usuario decide no dar los permisos, no va a poder utilizar la funcionalidad de buscar paradas cercanas y no va a salir ubicado en el mapa. Aún así, puede seguir utilizando las demás funcionalidades a su gusto y el mapa puede seguir siendo utilizado, simplemente el usuario no verá su ubicación en el mapa.

Para que el usuario sepa porqué se necesitan los permisos de ubicación, se muestra un diálogo por pantalla para ayudar al usuario a decidir si dar los permisos o no, como se puede ver en la figura 51.

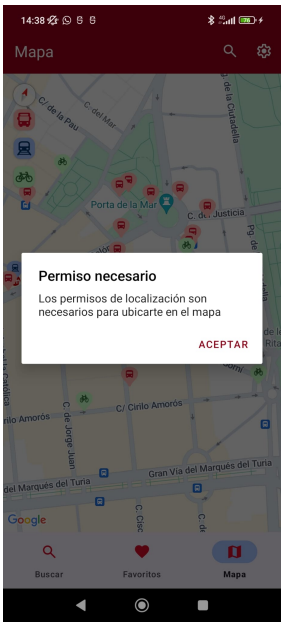

Figura 51: Diálogo explicativo del porqué se piden permisos de ubicación.

Tras haber realizado las pruebas de privacidad, he obtenido un resultado totalmente favorable en esta sección, los permisos en mi aplicación conceden al usuario la decisión para transmitir su ubicación a la aplicación de una forma elegante y nada intrusiva, ayudando al usuario con la decisión de si dar permisos o no a la aplicación.

7.3. Pruebas de usabilidad con usuarios reales

El principal problema para realizar estas pruebas con usuarios ha sido encontrar a usuarios que vivan en Valencia en verano. Yo, al ser de Mallorca, en verano me encuentro en la isla, entonces sólo he podido encontrar algunos compañeros y amigos de estos que puedan probar mi aplicación.

Para realizar las pruebas de usabilidad, cada usuario ha recibido dos archivos y un enlace al cuestionario. Esos dos archivos son el APK para instalar la aplicación y la guía paso a paso que deben seguir para asegurarme de que prueban todas las funcionalidades que tiene mi aplicación. Esta guía paso a paso se puede encontrar al final de la memoria en el Anexo C.

El cuestionario consta de 13 preguntas, la mayoría de ellas tienen resultados con una escala Likert[12] de 1 a 5 desde nada a favor hasta muy a favor. Otras preguntas tienen resultados de sí o no y la última pregunta es a modo de caja de comentarios donde los usuarios pueden plasmar características que durante las pruebas se han dado cuenta que sería interesante mejorar.

Con estas pruebas de usabilidad se puede observar la satisfacción que tienen los usuarios al probar mi aplicación, además de recibir críticas constructivas que pueden ser muy interesantes para aplicar cambios en la aplicación en un futuro.

Ahora voy a comentar en grandes rasgos los resultados obtenidos de las pruebas de usabilidad. El cuestionario ha sido realizado gracias a "Google Forms", una solución que proporciona Google para que cualquier persona pueda realizar cuestionarios con resultados en forma de gráficos.

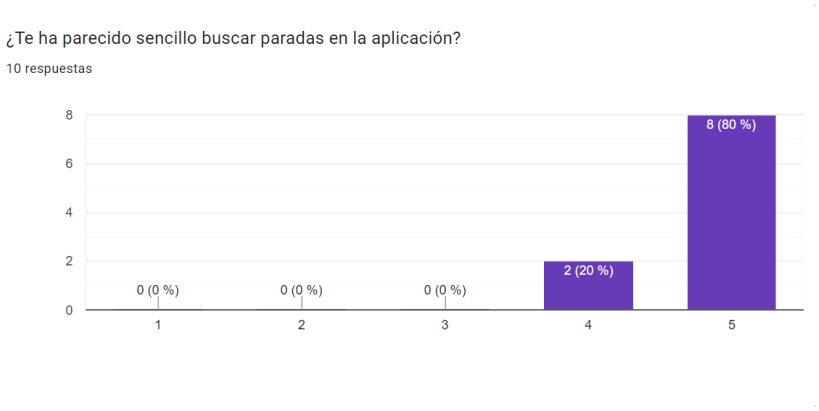

Figura 52: Cuestionario de pruebas de usabilidad primera pregunta

[12]La escala Likert tiene los resultados siguientes: Muy en desacuerdo, en desacuerdo, neutral, de acuerdo y muy de acuerdo.

En la figura 52, se puede observar el resultado de la primera pregunta del cuestionario: "¿Te ha parecido sencillo buscar paradas en la aplicación?". El resultado de esta pregunta es muy positivo pues la mayoría de gente ha puesto un resultado muy favorable de 5 puntos, podemos suponer que en general, la aplicación gestiona de forma sencilla la búsqueda de paradas.

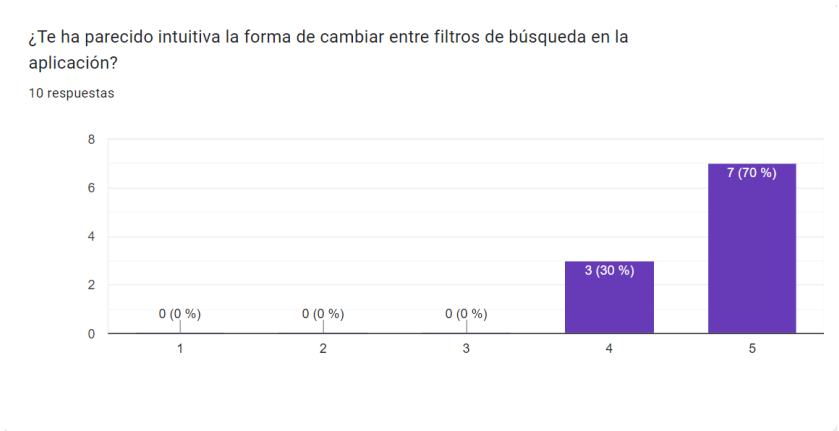

Figura 53: Cuestionario de pruebas de usabilidad segunda pregunta

La segunda pregunta del cuestionario cubre lo que aparece en la figura 53: "¿Te ha parecido intuitiva la forma de cambiar entre filtros de búsqueda en la aplicación?". Volvemos a obtener un resultado favorable con 7 usuarios dando 5 puntos y 3 usuarios dando 4 puntos. El cambio de filtros de búsqueda como GPS o buscar por nombre resulta intuitivo para los usuarios que han probado la solución.

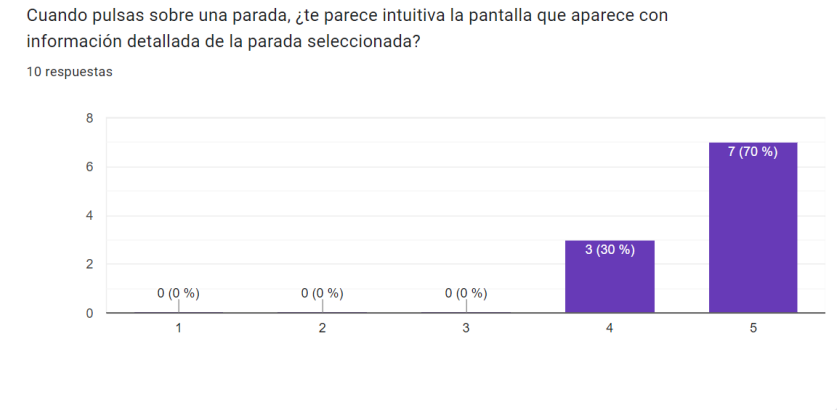

Figura 54: Cuestionario de pruebas de usabilidad tercera pregunta

La sección de la información detallada de las paradas se cubre con la pregunta que se muestra en la figura 54: "Cuando pulsas sobre una parada, ¿te parece intuitiva la pantalla que aparece con información detallada de la parada seleccionada?". El resultado favorable en esta cuestión indica que los usuarios sienten esta pantalla intuitiva de utilizar.

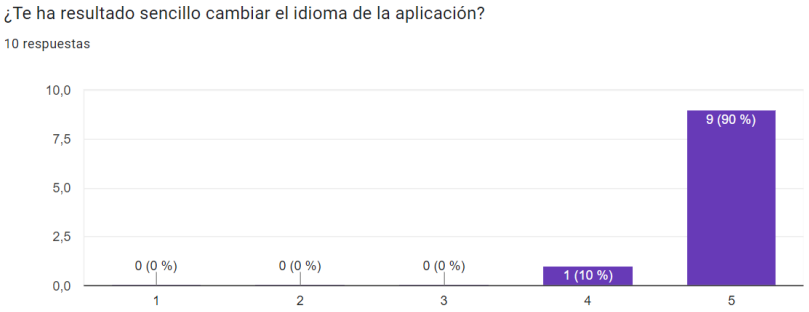

Figura 55: Cuestionario de pruebas de usabilidad cuarta pregunta

La funcionalidad de cambiar el idioma de la aplicación resulta muy sencilla para cualquier usuario según se puede ver en la figura 55, que cubre la pregunta: "¿Te ha resultado sencillo cambiar el idioma de la aplicación?".

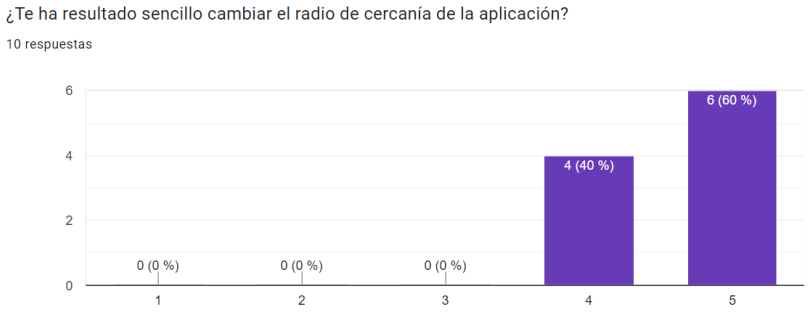

Figura 56: Cuestionario de pruebas de usabilidad quinta pregunta

La quinta pregunta trata la otra funcionalidad que aparece en esta misma pantalla de configuración, el cambio del radio de cercanía. Como se puede ver en la figura 56: "¿Te ha resultado sencillo cambiar el radio de cercanía de la aplicación?", el resultado sigue siendo positivo aunque no perfecto, pues hay usuarios que deben haber encontrado algún problema mínimo con esta funcionalidad o no se sienten tan satisfechos con la misma.

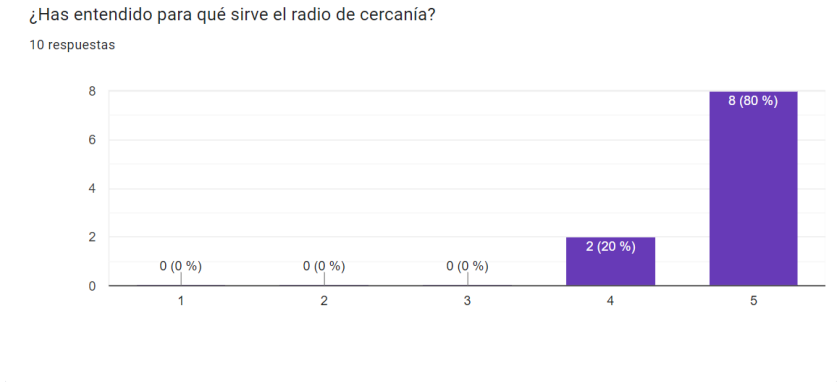

Figura 57: Cuestionario de pruebas de usabilidad sexta pregunta

Debido a mi temor porque los usuarios no entendieran correctamente para qué sirve el radio de cercanía, propuse la pregunta que se muestra en la figura 57: "¿Has entendido para qué sirve el radio de cercanía?". Como se puede observar en el resultado graficado, la gran mayoría de usuarios ha entendido a la perfección este valor que llamo radio de cercanía.

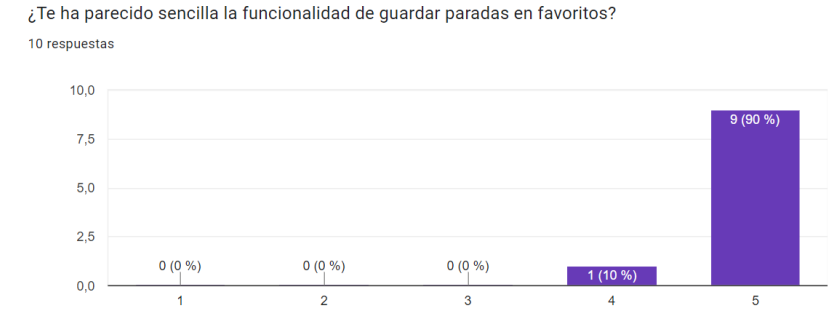

Figura 58: Cuestionario de pruebas de usabilidad séptima pregunta

Pasando a la funcionalidad de favoritos, uno de mis principales objetivos en este proyecto, he formulado la pregunta que se ve en la figura 58: "¿Te ha parecido sencilla la funcionalidad de guardar paradas en favoritos?". Se puede comprobar que casi todos los usuarios están de acuerdo con la sencillez de guardar paradas en favoritos.

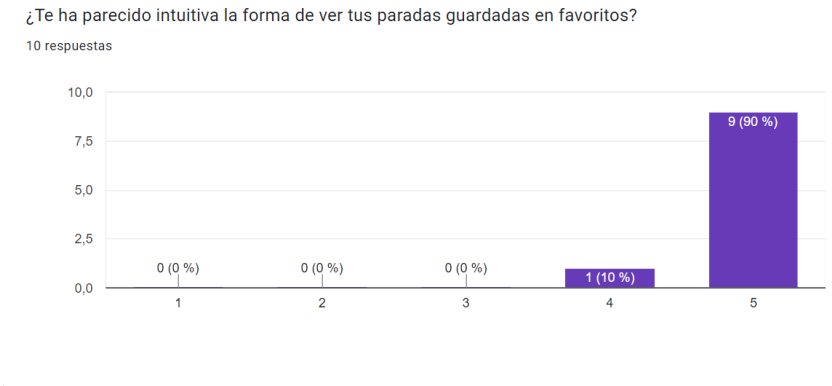

Figura 59: Cuestionario de pruebas de usabilidad octava pregunta

Otra pregunta relacionada con la pantalla de favoritos es la que aparece en la figura 59:"¿Te ha parecido intuitiva la forma de ver tus paradas guardadas en favoritos?". Esta pregunta cubre la funcionalidad relacionada con observar las paradas almacenadas en favoritos y vuelve a ser muy favorable que esta funcionalidad es sencilla para los usuarios.

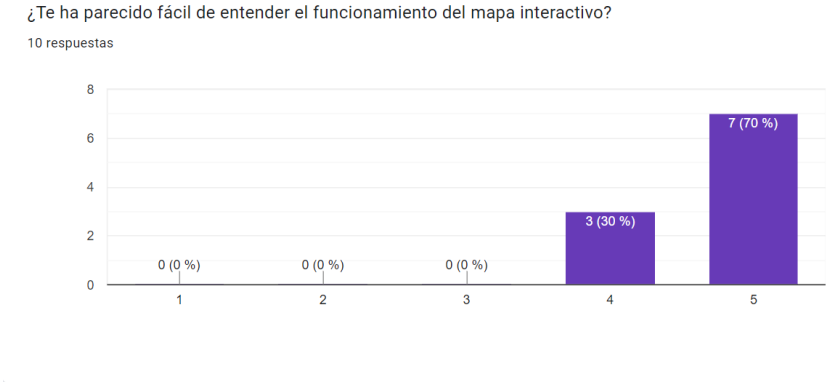

Figura 60: Cuestionario de pruebas de usabilidad novena pregunta

Pasando al mapa interactivo, se lanzó la siguiente pregunta: "¿Te ha parecido fácil de entender el funcionamiento del mapa interactivo?" cuyo resultado se puede observar en la figura 60. Por lo general, el resultado es positivo en esta pregunta aunque no perfecto con 5 puntos. Indica que los usuarios a grandes rasgos están contentos con esta funcionalidad.

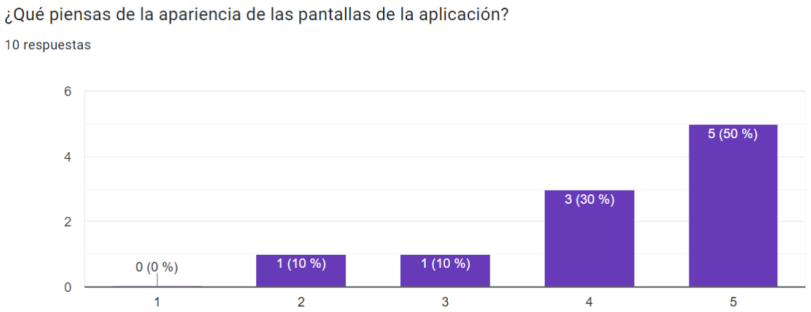

Figura 61: Cuestionario de pruebas de usabilidad décima pregunta

Pasando de lo funcional al diseño de interfaces, he preguntado sobre la apariencia de la aplicación con la pregunta que se ve en la figura 61: "¿Qué piensas de la apariencia de las pantallas de la aplicación¿'. Esta es la pregunta más debatida de todas, con resultados negativos en 2 puntos y muy favorables en 5. Hay gente que piensa que la apariencia es muy bonita y gente que ve muchas carencias en el diseño de la aplicación.

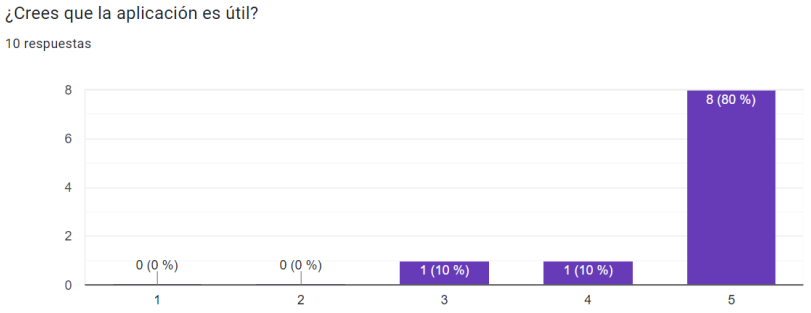

Figura 62: Cuestionario de pruebas de usabilidad onceava pregunta

La última pregunta del cuestionario es la que se observa en la figura 62: "¿Crees que la aplicación es útil? ". Con un resultado bastante favorable sobre si la aplicación es útil o no y únicamente un usuario se siente indiferente con esta cuestión.

Después de estas preguntas que cuentan de 1 a 5 como menos favorable a muy favorable, se proporciona una caja de comentarios para dejar posibles cambios en la aplicación. He recibido unas cuantas críticas a la aplicación:

Existe un usuario que propone un cambio a la forma en que se ve el filtro de búsqueda que ha sido seleccionado pues por ahora no existe tal funcionalidad. Este mismo usuario se queja también con los botones que aparecen en el mapa para esconder las paradas de tipos de transporte, le parecen poco visibles con el mapa detrás.

Otro usuario se ha dado cuenta de un pequeño *bug*[13] cuando se quiere cambiar el idioma de la aplicación. Al parecer, si accedes por primera vez a esta pantalla de configuración, la opción seleccionada es inglés aunque la aplicación está en castellano. Este *bug* ha sido corregido rápidamente después de obtener el comentario.

Otro usuario se queja del diseño y supongo que es el que no le ha parecido nada bien la apariencia de mi aplicación.

Finalmente, un usuario comenta que estaría bien algún tipo de sistema de rutas para llegar a algún lugar como hace Google Maps.

Estos comentarios aportan mucha información sobre los usuarios que quieren nuevas funcionalidades en mi aplicación y así puedo observar qué secciones necesitan mejoras para un futuro.

[13]bug: error o fallo en el sistema que hace que la aplicación funcione de manera incorrecta o inesperada.

8. Conclusiones del trabajo

La idea principal del proyecto fue desarrollar una solución que consiga unificar todos los transportes presentes en la ciudad de Valencia. Los usuarios pueden gracias a mi aplicación, obtener información de metrovalencia, EMT y valenbisi todo en una única aplicación.

Para agilizar el encontrar las paradas más importantes de cada usuario era necesario proporcionarles una sección donde guardar estas paradas importantes. La sección de favoritos ayuda a los usuarios a encontrar rápidamente las paradas que más utilizan.

En la ciudad de Valencia no solo vive gente que habla Español, es por ello que uno de los objetivos principales del proyecto fue la internacionalización de la aplicación para poder llegar a más público. La solución por ahora cubre los idiomas Español, Valenciano e Inglés, aunque en un futuro se podrán añadir más idiomas.

He tenido que analizar el mercado de aplicaciones de transporte en Valencia para encontrar una solución útil y diferente al transporte público en Valencia utilizando las mejores prácticas de desarrollo en Android junto con muchas bibliotecas que me han ayudado a extraer la información de los datos abiertos del Ayuntamiento de Valencia.

Un objetivo principal dentro de la aplicación fue el desarrollo de una interfaz sencilla de entender y agradable al usuario. Tras pasar las guías de calidad de Android y las pruebas de usabilidad con usuarios reales, se puede concretar que la interfaz ha sido un éxito para cumplir con el objetivo. En las pruebas de usabilidad se puede observar como existen usuarios que no están muy contentos con la apariencia de la aplicación, aunque hay muchos otros que sí se encuentran conformes con este diseño.

Durante el desarrollo del proyecto he tenido que aprender muchas tecnologías distintas sobre la extracción de información a través de servicios web como el API del Ayuntamiento de Valencia. Aunque el API presenta algunas problemáticas como el almacenamiento de información sobre llegadas en páginas web externas al API o la gestión de las consultas con cláusulas *where*, han sido solucionadas poco a poco durante el desarrollo. He podido aprender mucho de tecnologías relacionadas con dispositivos Android, extracción de datos, persistencia en dispositivos, Google Maps y mucho más.

La aplicación consta con funcionalidades como filtrar las paradas por transporte, por nombre, por cercanía, proporciona un mapa interactivo interesante para encontrar las paradas de transporte, la sección de favoritos que ya he comentado su importancia en el desarrollo y una pantalla para ajustar algunas preferencias del usuario como son el idioma y el radio de cercanía.

Los objetivos que fueron planteados al inicio del proyecto han sido cubiertos durante el transcurso del desarrollo y plasmados en el resultado final que es la aplicación de *ValenMove*. Una aplicación muy útil de transporte público en Valencia.

8.1. Relación del trabajo con los estudios cursados

Durante el desarrollo del proyecto han habido muchas asignaturas que me han dado facilidades para aplicar en la solución de las distintas problemáticas que me aparecían. La aplicación de técnicas de extracción siguiendo las prácticas realizadas durante la asignatura de "Integración e interoperabilidad", realizada en el último curso de mi carrera universitaria en la rama de ingeniería del software, me ha ayudado bastante para todo el desarrollo del proyecto. El bloque principal de este proyecto es la extracción de información a través de las API del ayuntamiento, es bastante clara la relación entre esta tarea y la asignatura mencionada.

Para almacenar la información de paradas en el dispositivo he requerido de algunos conceptos básicos de gestión de bases de datos, que obtuve en la asignatura de nombre "Bases de datos y sistemas de información", cursada en el segundo año de mi carrera universitaria, donde aprendí conceptos como qué es un DAO (*Data Access Object*), la importancia del sistema de gestión de bases de datos o cómo realizar consultas a una base de datos.

En la asignatura optativa de "Desarrollo de aplicaciones para dispositivos móviles" es que he aprendido todas las nociones del IDE de Android Studio así como el lenguaje de programación Kotlin. Gracias a esta asignatura he podido aplicar las mejores prácticas del desarrollo de aplicaciones Android en mi proyecto. Además de ser el profesor de esta asignatura, David de Andrés Martínez, el profesor dispuesto a tutorizar mi proyecto para presentarlo como trabajo de fin de grado.

La competencia de "análisis y resolución de problemas" que he perfeccionado durante el transcurso de mi carrera universitaria me ha sido de ayuda para el desarrollo de este trabajo, pues ha sido necesario mucho análisis crítico de las problemáticas que iban apareciendo para dar con las soluciones finales, así como también otra competencia importante ha sido "diseño y proyecto" para planificar y especificar en términos formales todo el trabajo realizado. Al aplicar las tecnologías nuevas para mí como son las bibliotecas de extracción utilizadas, el mapa de Google Maps, Google Cloud y mucho más, me ha servido otra competencia aprendida durante el transcurso de la carrera como es "herramientas y conceptos desconocidos".

8.2. Trabajo futuro

Como es común con cualquier proyecto, siempre existe un abanico de mejora, una serie de características nuevas o mejorables que podrían ser aplicadas en el futuro.

Gracias a las opiniones de los usuarios que han probado la aplicación, se pueden extraer muchas mejoras en términos de diseño como son la mejora del icono de la aplicación, de algunas pantallas de la misma para hacerla más "moderna" y más críticas constructivas que se van a poder aplicar en un futuro. Al no ser un diseñador de interfaces, la interfaz de mi aplicación aunque es útil podría mejorar en muchos aspectos, es por eso que se podría hablar con un diseñador para potenciar la vista de la aplicación.

En cuanto a funcionalidades nuevas a aplicar, desde el principio ha sido un problema

el obtener las paradas por línea de transporte, pues los servicios web del Ayuntamiento, aunque cuentan con cláusulas para filtrar los resultados obtenidos, ninguna consulta que se haya estudiado consigue dar un resultado correcto de las paradas pertenecientes a una línea concreta. Esto es una problemática que no ha sido importante solucionar durante el desarrollo del proyecto pero que podría ser útil tener en un futuro.

Si observamos detenidamente las aplicaciones de transporte público, todas cuentan con una característica muy compleja pero útil a la vez y es la obtención de las mejores rutas de transporte para llegar a cualquier punto de la ciudad. Al ser una tarea muy compleja, fue desestimada rápidamente durante las fases iniciales del proyecto, aunque en un futuro podría ser una funcionalidad que daría mucho valor a mi solución. Además, uno de los usuarios que ha probado la aplicación se da cuenta de que falta una funcionalidad como esta.

Otra funcionalidad interesante que podría ser aplicada es la utilización de alarmas para notificar al usuario de la llegada próxima de un transporte a cualquiera de las paradas favoritas del usuario. Una característica que no presenta ninguna aplicación estudiada.

Como comenta uno de los usuarios que ha pasado las pruebas de usabilidad, podría mostrarse de forma más elegante el filtro de búsqueda que ha sido seleccionado pues ahora mismo no existe ningún comentario al usuario de qué filtro está puesto más que el resultado de la búsqueda de paradas con ese filtro.

La aplicación por ahora consta de 3 idiomas únicamente, estaría bien en el futuro poder llegar a mucha más gente con la inclusión de nuevos lenguajes en el proyecto.

Finalmente, al ser una aplicación Android, sería muy interesante lanzar este proyecto en el portal de ventas de Google Play Store, así como también crear una versión para la plataforma de la App Store para dispositivos de la marca Apple.

Referencias

[1] Instituto Nacional de Estadística (INE). *Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia (EDAD) 2022*. 2022. URL: https://www.ine.es/prensa/edcm_2022_d.pdf.

[2] Wikipedia contributors. *Empresa Municipal de Transportes de Valencia — Wikipedia, La enciclopedia libre*. 2024. URL: https://es.wikipedia.org/wiki/Empresa_Municipal_de_Transportes_de_Valencia.

[3] Wikipedia contributors. *Metrovalencia*. 2024. URL: https://es.wikipedia.org/wiki/Metrovalencia (visitado 05-07-2024).

[4] Wikipedia contributors. *Valenbisi Wikipedia info*. 2024. URL: https://en.wikipedia.org/wiki/Valenbisi (visitado 12-05-2024).

[5] Gobierno de España. *Portal de datos abiertos*. 2024. URL: https://datos.gob.es/es/ (visitado 13-06-2024).

[6] Gobierno de España. *Los datos abiertos como herramientas para la educación y la formación*. 2024. URL: https://datos.gob.es/es/blog/los-datos-abiertos-como-herramientas-para-la-educacion-y-la-formacion (visitado 13-06-2024).

[7] Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado. *Procomún*. 2024. URL: https://procomun.intef.es/ (visitado 13-06-2024).

[8] Ayuntamiento de Valencia. *Valencia Open Data*. 2024. URL: https://valencia.opendatasoft.com/pages/home/ (visitado 13-06-2024).

[9] Ayuntamiento de Valencia. *EMT Dataset*. 2024. URL: https://valencia.opendatasoft.com/explore/dataset/emt/table/ (visitado 13-06-2024).

[10] Portal de l'Ajuntament de la ciutat de València. *FGV Estacions / Estaciones*. 2024. URL: https://valencia.opendatasoft.com/explore/dataset/fgv-estacions-estaciones/table/ (visitado 05-07-2024).

[11] Portal de l'Ajuntament de la ciutat de València. *ValenBisi Disponibilidad*. 2024. URL: https://valencia.opendatasoft.com/explore/dataset/valenbisi-disponibilitat-valenbisi-dsiponibilidad/table/ (visitado 05-07-2024).

[12] Google. *Material Design*. 2024. URL: https://m3.material.io/ (visitado 05-07-2024).

[13] Jean-Paul Subra y Aurélien Vannieuwenhuyze. *Scrum [Recurso electrónico-En línea] : un método ágil para sus proyectos*. spa. ExpertIT. Cornellà de Llobregat: ENI, 2018. ISBN: 9782409012921.

[14] Portal de asana - Julia Martins. *¿Qué es la metodología Kanban y cómo funciona?* 2024. URL: https://asana.com/es/resources/what-is-kanban (visitado 12-05-2024).

[15] EMT Valencia. *EMT Valencia*. 2024. URL: https://play.google.com/store/apps/details?id=es.emtvalencia.emt&hl=es (visitado 05-07-2024).

[16] Ferrocarrils de la Generalitat Valenciana. *Metrovalencia*. 2024. URL: https://play.google.com/store/search?q=Metrovalencia&c=apps&hl=es (visitado 05-07-2024).

[17] Wikipédia. *página de la arquitectura Model-View-ViewModel en Wikipédia*. 2024. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel (visitado 05-06-2024).

[18] IBM. *¿Qué es la arquitectura en 3 niveles?* 2024. URL: https://www.ibm.com/es-es/topics/three-tier-architecture (visitado 05-06-2024).

[19] Wikipédia. *página de la wikipedia sobre casos de uso*. 2024. URL: https://es.wikipedia.org/wiki/Caso_de_uso (visitado 05-06-2024).

[20] Google. *Firebase*. URL: https://firebase.google.com/?hl=es-419 (visitado 15-06-2024).

[21] Robert C. Martin. *Getting a Solid Start*. 2009. URL: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start?pli=1 (visitado 25-05-2024).

[22] Android Developers. *Android Studio*. 2024. URL: https://developer.android.com/studio (visitado 26-07-2024).

[23] JetBrains. *Kotlin*. URL: https://kotlinlang.org/ (visitado 20-06-2024).

[24] Squareup. *Documentación de la librería Retrofit*. 2024. URL: https://square.github.io/retrofit/ (visitado 05-06-2024).

[25] Squareup. *Documentación de la librería Moshi*. 2024. URL: https://square.github.io/moshi/1.x/moshi/ (visitado 05-06-2024).

[26] Android Developers. *Documentación de la librería Room*. 2024. URL: https://developer.android.com/jetpack/androidx/releases/room (visitado 05-06-2024).

[27] SQlite. *Well-Known Users of SQLite*. 2024. URL: https://www.sqlite.org/famous.html (visitado 05-06-2024).

[28] Google Cloud. *Página oficial de Google Cloud*. URL: https://cloud.google.com/ (visitado 30-06-2024).

[29] Google. *Google Maps*. 2024. URL: https://www.google.com/maps (visitado 26-07-2024).

[30] OnTheMap. *Google Maps Statistics*. URL: https://www.onthemap.com/blog/google-maps-statistics/ (visitado 28-06-2024).

[31] Google. *Google Street View*. 2024. URL: https://www.google.com/streetview/ (visitado 26-07-2024).

[32] Jonathan Hedley. *Jsoup: Java HTML Parser*. 2024. URL: https://jsoup.org/ (visitado 26-07-2024).

[33] Android Developers. *Paging 3 Library Overview*. 2024. URL: https://developer.android.com/topic/libraries/architecture/paging/v3-overview?hl=es-419 (visitado 26-07-2024).

[34] Figma. *Página web oficial de Figma*. URL: https://www.figma.com/ (visitado 22-06-2024).

[35] Material Foundation. *Material Theme Builder*. URL: https://material-foundation.github.io/material-theme-builder/ (visitado 25-06-2024).

[36] Flaticon. *Página web oficial de Flaticon*. URL: https://www.flaticon.com/ (visitado 18-06-2024).

[37] Android Developers. *Core App Quality Guidelines*. 2024. URL: https://developer.android.com/docs/quality-guidelines/core-app-quality (visitado 02-07-2024).

[38] Android Developers. *Android App Bundle*. 2024. URL: https://developer.android.com/guide/app-bundle (visitado 02-07-2024).

[39] Google Support. *Android Accessibility Help*. 2024. URL: https://support.google.com/accessibility/android/answer/6283677?hl=en (visitado 02-07-2024).

Anexo A. Detalle de los casos de uso

Tabla 2: CU-01 - Obtener paradas de transporte público.

Id	CU-01
Nombre	Obtener paradas de transporte público.
Descripción	Se mostrará una lista de paradas de transporte enseñando algo de información sobre cada una.
Precondición	Ninguna.
Proceso	1. El usuario se sitúa sobre la pantalla de búsqueda 2. El sistema muestra una lista de paradas.
Postcondición	Ninguna.
Alternativas / Errores	Si el usuario no tiene acceso a internet, se muestra un mensaje de error.

Tabla 3: CU-02 - Filtrar paradas por tipo de transporte.

Id	CU-02
Nombre	Filtrar paradas por tipo de transporte.
Descripción	Se mostrará una lista de paradas / estacionamientos dependiendo de la selección de tipo de transporte del usuario (EMT, Metrovalencia, Valenbisi).
Precondición	El usuario selecciona entre EMT, Metrovalencia y Valenbisi.
Proceso	1. El usuario se sitúa sobre la pantalla de búsqueda 2. El usuario selecciona el tipo de transporte 3. El sistema muestra una lista acorde con el transporte seleccionado.
Postcondición	Ninguna.
Alternativas / Errores	1. Si el usuario no tiene acceso a internet, se muestra un mensaje de error. 2. El transporte por defecto es EMT.

Tabla 4: CU-03: Obtener información detallada de una parada.

Id	CU-03
Nombre	Obtener información detallada de una parada.
Descripción	Se mostrará información visual y detallada de las paradas (nombre de la parada, líneas, próximas llegadas,...).
Precondición	El usuario pulsa sobre una parada.
Proceso	1. El usuario pulsa sobre una parada en cualquier punto de la aplicación. 2. El sistema muestra una pantalla con información detallada de las paradas.
Postcondición	Ninguna.
Alternativas / Errores	1. Si el usuario no tiene acceso a internet, se muestra un mensaje de error. 2. Si la parada es de Valenbisi, se muestra otro tipo de información de esta fuente de datos.

Tabla 5: CU-04: Guardar parada en favoritos.

Id	CU-04
Nombre	Guardar parada en favoritos.
Descripción	Se podrá guardar una parada como favorita para acceder más rápido a ella en un futuro.
Precondición	El usuario pulsa sobre una parada.
Proceso	1. El usuario pulsa sobre una parada en cualquier punto de la aplicación. 2. El sistema muestra una pantalla con información detallada de las paradas 3. El usuario pulsa en el botón de guardar en favoritos.
Postcondición	Si la parada ya está guardada, no aparece el botón de favoritos.
Alternativas / Errores	Ninguna.

Tabla 6: CU-05: Mostrar información de transporte en un mapa interactivo.

Id	CU-05
Nombre	Mostrar información de transporte en un mapa interactivo
Descripción	Se mostrarán las paradas de todos los tipos de transporte en un mapa interactivo, que permite obtener información detallada de las paradas.
Precondición	Ninguno.
Proceso	1. El usuario selecciona la pantalla del mapa 2. El sistema muestra las paradas cercanas al usuario según el radio de cercanía.
Postcondición	Ninguna.
Alternativas / Errores	1. Si el usuario ingresa por primera vez en esta funcionalidad, se muestra un diálogo para que el usuario seleccione el permiso de localización. 2. Si el usuario no permite la localización por GPS, el sistema muestra un error

Tabla 7: CU-06: Buscar paradas por nombre.

Id	CU-06
Nombre	Buscar paradas por nombre
Descripción	El sistema muestra una lista de las paradas con un nombre parecido al que se está intentando encontrar.
Precondición	El usuario introduce el nombre de la parada que quiere encontrar.
Proceso	1. El usuario selecciona buscar por nombre 2. El usuario introduce el nombre de la parada 3. El sistema muestra las paradas de nombre parecido al introducido.
Postcondición	Ninguna.
Alternativas / Errores	Si el usuario no tiene acceso a internet, se muestra un error.

Tabla 8: CU-07: Mostrar parada en el mapa.

Id	CU-07
Nombre	Mostrar parada en el mapa
Descripción	Se podrá mostrar la posición geográfica de una parada concreta en el mapa interactivo.
Precondición	El usuario selecciona una parada concreta y pulsa para buscar en el mapa.
Proceso	1. El usuario selecciona una parada concreta 2. El usuario pulsa el botón de ver en el mapa 3. El sistema muestra la parada seleccionada en el mapa interactivo.
Postcondición	Ninguna.
Alternativas / Errores	1. Si el usuario ingresa por primera vez en esta funcionalidad, salta un diálogo para que el usuario seleccione el permiso de localización. 2. Si el usuario no permite la localización por GPS, el sistema muestra un error.

Tabla 9: CU-08: Buscar paradas por cercanía.

Id	CU-08
Nombre	Buscar paradas por cercanía.
Descripción	Se mostrará una lista de paradas cercanas al usuario.
Precondición	El usuario selecciona filtrar las paradas por cercanía.
Proceso	1.El usuario se sitúa en la pantalla de búsqueda 2.El usuario selecciona buscar por cercanía 3.El sistema muestra una lista de paradas cercanas según el radio de cercanía.
Postcondición	Ninguna.
Alternativas / Errores	1.Si el usuario ingresa por primera vez en esta funcionalidad, salta un diálogo para que el usuario seleccione el permiso de localización. 2.Si el usuario no permite la localización por GPS, el sistema muestra un error.

Tabla 10: CU-09: Cambiar el radio de cercanía.

Id	CU-09
Nombre	Cambiar el radio de cercanía.
Descripción	Se adaptarán los resultados de los filtros de cercanía y el mapa al nuevo radio.
Precondición	El usuario selecciona un radio de cercanía.
Proceso	1.El usuario se sitúa en la pantalla de configuración 2.El usuario selecciona un radio de una lista de preferencias. 3.El sistema guarda esta nueva preferencia de radio.
Postcondición	Ninguna.
Alternativas / Errores	Nada.

Tabla 11: CU-10: Cambiar lenguaje de la aplicación.

Id	CU-10
Nombre	Cambiar lenguaje de la aplicación.
Descripción	Se podrá cambiar el idioma de la aplicación a gusto del usuario entre castellano, valenciano e inglés.
Precondición	El usuario selecciona un idioma,
Proceso	1. El usuario se sitúa en la pantalla de configuración 2. El usuario selecciona un idioma entre los 3 admitidos por la aplicación. 3. El sistema cambia el idioma de la aplicación.
Postcondición	Ninguna.
Alternativas / Errores	Nada.

Tabla 12: CU-11: Mostrar lista de paradas favoritas.

Id	CU-11
Nombre	Mostrar lista de paradas favoritas.
Descripción	Se muestran las paradas que se han guardado como favoritas.
Precondición	Ninguna.
Proceso	1. El usuario se sitúa en la pantalla de favoritos 2. El sistema muestra la lista completa de paradas favoritas.
Postcondición	Ninguna.
Alternativas / Errores	Nada.

Tabla 13: CU-12: Filtrar favoritos por tipo de transporte.

Id	CU-12
Nombre	Filtrar favoritos por tipo de transporte.
Descripción	Se muestran todas las paradas favoritas de un transporte concreto.
Precondición	Se selecciona uno de entre los distintos tipos de transporte.
Proceso	1. El usuario se sitúa en la pantalla de favoritos 2. El usuario selecciona un tipo de transporte. 3. El sistema muestra una lista de paradas favoritas de ese transporte concreto.
Postcondición	Ninguna.
Alternativas / Errores	Nada.

Tabla 14: CU-13: Eliminar paradas de favoritos.

Id	CU-13
Nombre	Eliminar paradas de favoritos.
Descripción	El sistema elimina una o todas las paradas guardadas en favoritos.
Precondición	Se selecciona borrar una parada o borrar varias.
Proceso	1. El usuario se sitúa en la pantalla de favoritos 2. El usuario selecciona una parada a eliminar. 2. El usuario selecciona borrar todas las paradas 3. El sistema elimina todas las paradas seleccionadas.
Postcondición	Ninguna.
Alternativas / Errores	Nada.

Anexo B. Pruebas de calidad de Android

En este anexo se encuentran detalladas las pruebas realizadas sobre las guías de calidad de aplicaciones Android en forma de tablas. Cada tabla se centra en una guía concreta como: experiencia visual, rendimiento y estabilidad o privacidad y seguridad. Dentro de cada campo existen distintas áreas, distintas pruebas con sus descripciones, los pasos necesarios para realizar las pruebas de cada característica y, finalmente, el resultado de la prueba realizada a mi solución.

Tabla 15: Desglose de la prueba de experiencia visual

Experiencia Visual			
Área	**Descripción**	**Pasos para probar**	**¿Pasa la prueba?**
Navegación	La aplicación soporta el botón hacia atrás de forma normal y no utiliza ningún botón visual en la pantalla para esta funcionalidad	CR-3	Sí
	La aplicación soporta los gestos de navegación para ir hacia atrás o ir a la pantalla principal del dispositivo	CR-3	Sí
	La aplicación conserva correctamente el estado de la pantalla o la información del usuario. Esto se refiere a cuando el usuario cambia el foco de la pantalla, se almacena el estado para que cuando vuelva a esta pantalla no se reinicie la información de la pantalla.	CR-1 CR-3 CR-5	Sí
Notificaciones	Las notificaciones siguen el siguiente estándar: 1. No se utilizan con fines de publicitar otros productos, pues está prohibido en Google Play Store. 2. Muchas notificaciones se agrupan en grupos de notificaciones. 3. Poner temporizadores cuando sea apropiado. 4. Las notificaciones son persistentes únicamente si están relacionadas con eventos en curso (llamadas, música, etc.). 5. Los canales de notificaciones están diseñados utilizando las mejores prácticas.	CR-9	No aplica

Tabla 16: Segunda tabla desglose de la prueba de experiencia visual

Experiencia Visual			
Área	**Descripción**	**Pasos para probar**	**¿Pasa la prueba?**
Interfaz de Usuario y gráficos	La aplicación soporta el modo vertical y horizontal.	CR-5	Sí
	La aplicación ocupa todo el ancho y largo de la pantalla tanto en modo apaisado como en modo vertical.	CR-5	Sí
	La app controla de manera eficiente el cambio entre el modo horizontal y vertical sin problemas de renderizado de la pantalla y manteniendo el estado de la interfaz.	CR-5	Sí
Calidad visual	La app muestra elementos de la IU, como texto, imágenes, gráficos y más sin distorsiones ni pixelado notables	CR-todos	Sí
	La aplicación admite el tema oscuro.	CR-todos	No
	La app muestra texto y bloques de texto sin problemas para ser visualizado en todos los idiomas que admite.	CR-todos	Sí
Accesibilidad	El tamaño de los elementos táctiles como botones o campos para introducir texto por ejemplo, tienen un tamaño mínimo de 48dp.	CR-todos	Sí
	El contenido de cualquier elemento de la interfaz debe mantener una relación de contraste lo suficientemente alta con su contenedor.	CR-todos	Sí
	Todos los elementos de la interfaz, excepto los campos de texto, deben estar descritos mediante su propiedad contentDescription.	CR-todos	Sí

Tabla 17: Desglose de la prueba de Rendimiento y estabilidad

Rendimiento y estabilidad			
Área	**Descripción**	**Pasos para probar**	**¿Pasa la prueba?**
Estabilidad	La aplicación no falla ni bloquea el subproceso de la interfaz que provoca errores "Android no responde"	CR-todas SD-1	Sí
Rendimiento	La app se carga rápidamente o le proporciona al usuario información de porqué está tardando tanto en cargar	CR-5	Sí
	Las aplicaciones deben renderizar los fotogramas cada 16 milisegundos para garantizar los 60 fotogramas por segundo.	CR-todas SD-1	Sí, con problemas
	Realizar pruebas con StrictMode activado y garantizar que no se verán destellos rojos durante las pruebas de la aplicación	PM-1	No
SDK	La aplicación se ejecutará en la última versión pública de la plataforma Android sin que se produzca una falla.	CR-0	Sí
	La aplicación se orienta al SDK (Software Development Kit - kit de desarrollo de software) de Android más reciente siguiendo los requisitos de Google Play.	SP-1	Sí
	Se compliará la app con el último SDK de Android.	SP-1	Sí
	Todos los SDK de Google o de terceros están actualizados a su última versión.	SP-2 SP-3	Sí

Tabla 18: Desglose de la prueba de Privacidad y seguridad

Privacidad y seguridad			
Área	**Descripción**	**Pasos para probar**	**¿Pasa la prueba?**
Permisos	La app sólo pide los permisos mínimos necesarios para los casos de uso de la aplicación.	CR-0	Sí
	La aplicación solicita permiso a datos sensibles (registro de llamadas, sms, localización,...) o servicios que cuesten dinero si están directamente relacionados con los casos de uso de la aplicación. Si existen alternativas al uso de estos permisos se deben aplicar.		Sí
	La aplicación solicita los permisos en tiempo de ejecución, solo cuando se debe utilizar ese permiso, en lugar de hacerlo cuando se inicia la aplicación.	CR-0	Sí
	La aplicación explica correctamente para qué necesita cada permiso.	CR-0	Sí
	La aplicación debe seguir funcionando aún si el usuario ha revocado algún permiso	CR-0	Sí

Anexo C. Guía paso a paso

Este documento es una guía paso a paso para utilizar la aplicación *ValenMove*. Se utilizará el documento para unas pruebas de usabilidad con usuarios reales. A continuación se detallarán los pasos que debes seguir para realizar la prueba. Tiene muchos pasos para explicar todo de una forma mucho más entendible. No te abrumes, que se hace en poco tiempo.

C.1 Pantalla de búsqueda

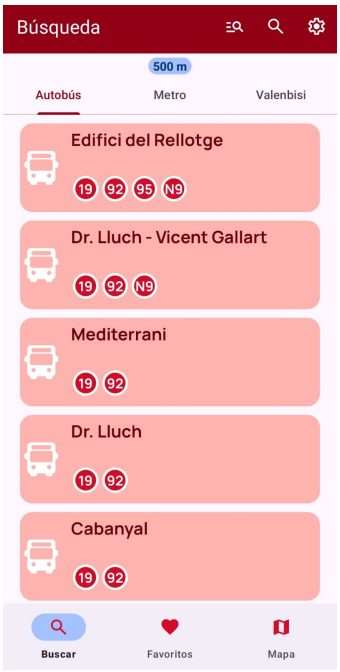

Figura 63: Pantalla buscar

1. Abre la aplicación. Verás la pantalla de búsqueda de la aplicación, como se ve en la figura 63 y verás una lista de paradas con nombre y líneas.

2. Cambia de tipo de transporte pulsando en Metro o en Valenbisi (puedes cambiar también deslizando con el dedo hacia los lados).

3. Vuelve al transporte de autobús pulsando en autobús.

4. Sitúate sobre la barra de acción que se encuentra en la parte de arriba de la pantalla (donde aparece el letrero de Búsqueda) y ahí pulsa sobre el botón de la lupa, como en la figura 64.

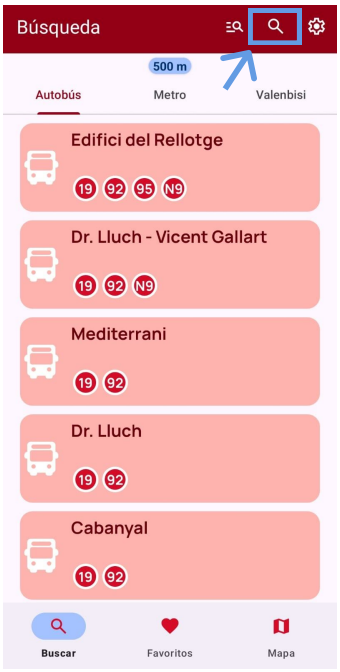

Figura 64: Pantalla buscar - pulsar en buscar

5. Aparecerá el teclado, con el que ahora tendrás que poner algún nombre de parada que conozcas (puedes probar con Blasco, Colón, Amistat,...).

6. Observa que ha cambiado el filtro de búsqueda a la palabra que has puesto en el buscador.

7. Prueba a cambiar de tipo de transporte (como hiciste en el paso 2) para observar que se mantiene el filtro entre los demás tipos de transporte.

8. A continuación, pulsa sobre el botón de al lado del de buscar, como se observa en la figura 65 (igual no se encuentra como en la figura debido a que se ha desplegado el campo para introducir el nombre de la parada).

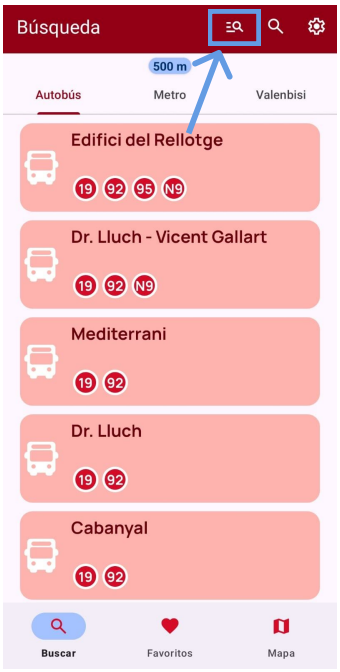

Figura 65: Pantalla buscar - pulsar en filtrar

9. Se desplegará una lista de opciones, Puedes pulsar en todas las paradas y observarás que el filtro de buscar por nombre se cambia al que tenías al iniciar la aplicación.

10. Prueba a pulsar otra vez ese botón que se ve en la figura 65 y selecciona buscar por GPS.

11. Observa si te aparecen paradas de algún tipo con esta funcionalidad. Si no te aparece nada puede ser que estés muy lejos de las paradas, el radio de cercanía se puede ver en el texto encima de los tipos de transporte.

C.2 Pantalla de configuración

Figura 66: Configuración

1. Vamos a cambiar a la pantalla de configuración, pulsando en el botón del engranaje como se observa en la figura 67.

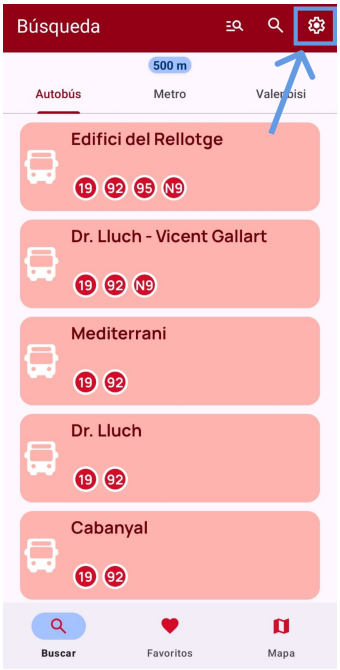

Figura 67: Ir a configuración

2. Se mostrará una pantalla como en la figura 66, con dos campos de configuración. Uno para cambiar el idioma y otro para cambiar el radio de cercanía.

3. Prueba a cambiar el idioma, volver atrás y observar los cambios en la pantalla anterior.

4. Si has cambiado el idioma, te recomiendo volver al Español para seguir mejor la guía.

5. Vuelve a la pantalla de configuración y ahora prueba a cambiar el radio de cercanía.

6. Una vez cambiado el radio de cercanía vuelve a la pantalla de buscar y observa como aparecen más o menos resultados dependiendo del radio en metros seleccionado en la funcionalidad de buscar por GPS(Si no recuerdas cómo, mira el paso 10 del apartado 8.2).

C.3 Pantalla de información detallada

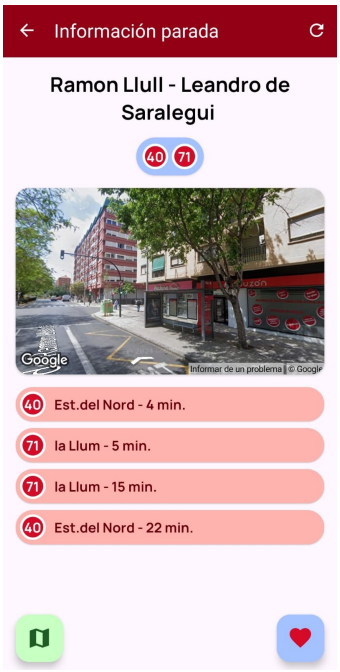

Figura 68: Pantalla Información

1. Ahora debes volver a la pantalla de buscar y pulsar sobre cualquier parada o estacionamiento que encuentres.

2. Se verá una pantalla como la que aparece en la figura 68. Aquí puedes observar información detallada de la parada seleccionada. Puedes jugar con la imagen de la parada que se encuentra en medio de la pantalla si lo deseas.

3. Prueba a pulsar el botón de favoritos que se encuentra en la parte **inferior derecha** con un **icono de un corazón**. Para añadir la parada a favoritos.

4. puedes pulsar sobre el botón de actualizar arriba a la derecha para refrescar la información en pantalla.

5. Si pulsas sobre el botón de la zona **inferior izquierda** con un **icono de un mapa**, te llevará a la pantalla del mapa situado sobre la parada en concreto que has seleccionado.

6. Vuelve hacia atrás hasta volver hasta la pantalla de buscar. Ahí, selecciona la **opción de Favoritos** en la **barra de navegación** situada en la parte **inferior de la pantalla**, como se ve en la figura 69.

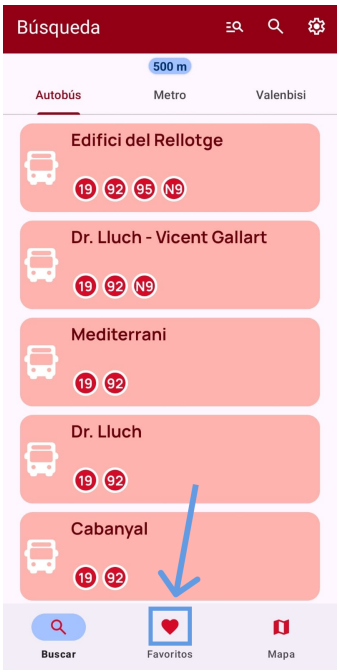

Figura 69: Ir a favoritos

C.4 Pantalla de favoritos

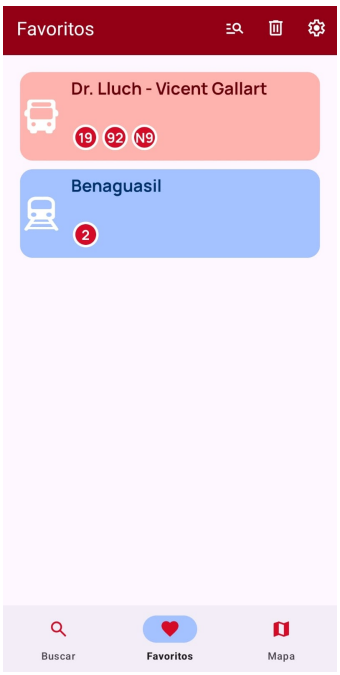

Figura 70: Pantalla Favoritos

En esta pantalla habrá una interfaz similar a la que se ve en la figura 70.

1. Si has guardado alguna parada en favoritos como en el paso 3 del apartado 8.2, la lista de favoritos contendrá alguna parada guardada. Prueba a pulsar sobre la parada para ver información detallada de esta.
2. Vuelve atrás a la pantalla de favoritos y **desliza a la derecha alguna parada** que tengas en favoritos. Debe desaparecer, se ha eliminado de favoritos.
3. Prueba a **volver a la pantalla de buscar** y añadir las paradas que tú quieras en favoritos. Debes **añadir más de 2** y **de tipos distintos** de transporte.
4. Vuelve a la pantalla de Favoritos y prueba a pulsar sobre el **botón de filtrar por tipo** como se ve en la figura 71.

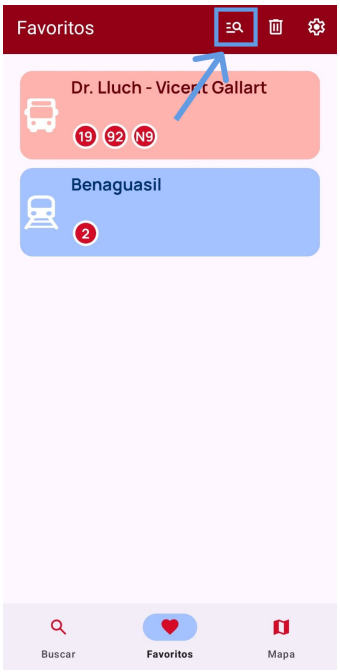

Figura 71: Pantalla Favoritos - pulsar en filtrar

5. Prueba a seleccionar cualquier tipo de transporte y así observar los cambios para ver paradas de un único transporte guardadas en favoritos.

6. Vuelve al **filtro para ver todas las paradas de favoritos** y pulsa en el botón de la basura para eliminar todas tus paradas en favoritos, como se ve en la figura 72.

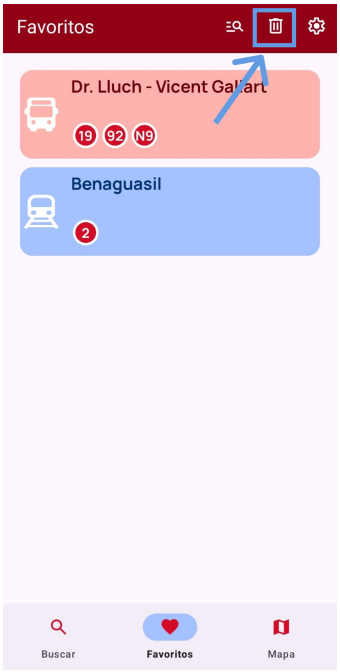

Figura 72: Pantalla Favoritos - pulsar en eliminar todos

7. Pulsa sobre la **opción del mapa** en la **barra de navegación** situada debajo de la pantalla. Aparecerá una pantalla como la de la figura 73.

C.5 Pantalla del mapa

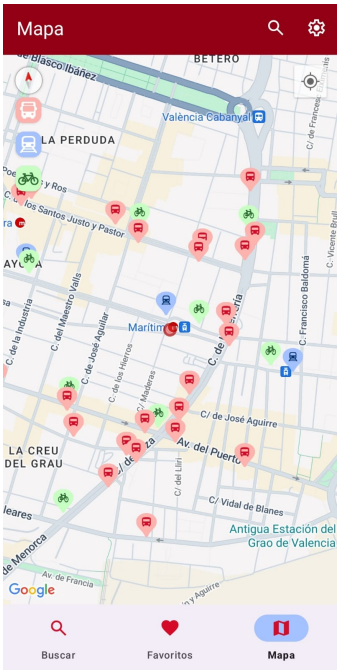

Figura 73: Pantalla Mapa

Finalmente, la pantalla del mapa, que tiene una interfaz como la de la figura 73.

1. Prueba a situarte **sobre Valencia** con el mapa y pulsa sobre algún marcador de autobús, metro o bicicletas. Saldrá una parada como se ve en la figura 74.

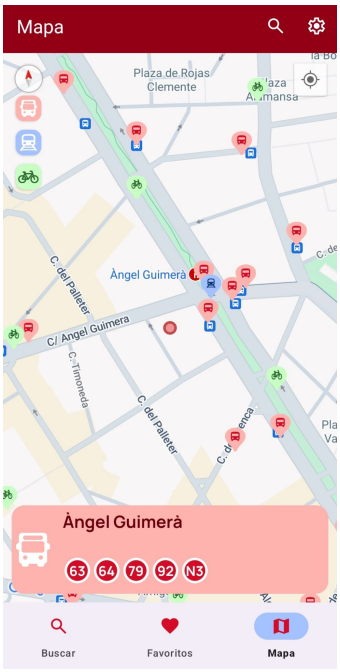

Figura 74: Pantalla Mapa - pulsar sobre marcador

2. Si pulsas sobre esta parada, te enviará a la anteriormente ya visitada pantalla de información detallada.

3. Volviendo al mapa, prueba a pulsar alguno de los **botones de la parte superior izquierda** para **esconder** los marcadores que quieras, como se ve en la figura 75.

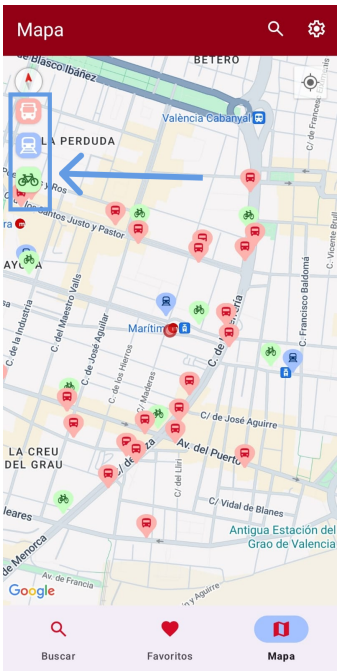

Figura 75: Pantalla Mapa - pulsar sobre esconder marcadores

4. Busca localizaciones con el **botón de buscar** situado en la **parte superior** de la pantalla. Prueba con UPV, por ejemplo.

C.6 Conclusiones de la guía

Después de realizar esta guía paso a paso por la aplicación, se presentará un cuestionario a rellenar con una serie de preguntas sobre las funcionalidades vistas en este paso a paso de mi aplicación *ValenMove*. ¡Muchas gracias por llegar hasta aquí con las pruebas y por continuar con el cuestionario!

Anexo D. Objetivos de desarrollo sostenible

Los Objetivos de Desarrollo Sostenible (ODS) son los objetivos propuestos por las naciones unidas para poner fin a la pobreza, garantizar la paz y la prosperidad y proteger el planeta.

En la tabla 19, he plasmado la implicación de mi proyecto sobre cada uno de los 17 objetivos propuestos por las Naciones Unidas.

Tabla 19: Implicación del proyecto sobre los objetivos de desarrollo sostenible.

Objetivos de desarrollo sostenible	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 infraestructura.				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 los objetivos				X

El proyecto tiene implicación sobre dos objetivos en concreto: Ciudades y comunidades sostenibles (ODS 11) y Acción por el clima (ODS 13).

Al tratarse de una aplicación de transporte que trata de ayudar a los ciudadanos a utilizar esta movilidad, se puede ver la relación directa con el objetivo relacionado con las ciudades y comunidades sostenibles.

El potenciamiento del uso del transporte público ayuda de forma indirecta a reducir las emisiones de gases nocivos a la atmósfera debido a la disminución de coches en la ciudad, relacionando así el proyecto también con el objetivo número 13 de acción por el clima.