UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Waltrip: tu cartera de viaje. Desarrollo del Back-end

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Morata Gomila, Alberto

Tutor/a: Penadés Gramage, María Carmen

CURSO ACADÉMICO: 2022/2023

A mi familia y amigos, por apoyarme y acompañarme siempre a lo largo del camino.

A mi tutora, Maria Carmen Penades Gramage por su dedicación y compromiso con este trabajo.

A Paula, por el tiempo y las experiencias compartidas en esta importante etapa.

Gracias

Resum

L'objectiu del següent document és la presentació de l'estudi i el procés de desenvolupament de Waltrip, una eina per a la gestió de rutes turístiques per a viatgers. Basada en una aplicació mòbil, els usuaris poden cercar entre una àmplia varietat de rutes disponibles, guardar-les, i navegar-hi o consultar-les en qualsevol moment.

Aquest treball s'ha realitzat com a Treball Final de Grau, en la modalitat d'emprenedoria, per al grau d'Enginyeria Informatica de la Universitat Politècnica de València, on es contemplen els diferents aspectes que envolten el negoci i el desenvolupament del producte per a un projecte emprenedor.

Waltrip és un projecte dut a terme en conjunt per Paula Romero Salas, encarregada de l'apartat del frontend, i Alberto Morata Gomila, encarregat del backend, tots dos estudiants del mateix grau. Així, cadascun dels membres se centra en un dels aspectes fonamentals per a la creació del producte. Sota aquesta premissa, a la memòria següent es presentarà el desenvolupament de l'apartat del Backend de l'aplicació, exposant les metodologies, arquitectures, tecnologies utilitzades i processos seguits, entrant al detall en la seva relació amb el treball realitzat per la resta de membres.

A més de l'exposició de l'apartat tècnic del desenvolupament del producte, i per la condició de projecte emprenedor, l'aspecte del negoci cobrarà una gran importància al llarg del document, sent condicionant principal en la presa de decisions durant tot el procés, sent un de els aspectes diferenciadors del següent treball. A diferència de la distribució per parts realitzada per al desenvolupament, els aspectes de negocis són compartits per tots els membres de l'equip i així queden il·lustrats al següent document.

Paraules clau: aplicació mòbil, rutes, turisme, navegació, comerç

Resumen

El objetivo del siguiente documento es la presentación del estudio y proceso de desarrollo de Waltrip, una herramienta para la gestión de rutas turísticas para viajeros. Basada en una aplicación móvil, los usuarios pueden busca entre una amplia variedad de rutas disponibles, guardarlas, y navegar por ellas o consultarlas en cualquier momento.

Este trabajo se ha realizado como Trabajo Final de Grado, en la modalidad de emprendimiento, para el grado de Ingenieria Informatica de la Universidad Politécnica de Valencia, en el que se contemplan los diferentes aspectos que rodean al negocio y el desarrollo del producto para un proyecto emprendedor.

Waltrip es un proyecto llevado a cabo en conjunto por Paula Romero Salas, encargada del apartado del frontend, y Alberto Morata Gomila, encargado del backend, ambos estudiantes del mismo grado. De esta forma, cada uno de los miembros se centra en uno de los aspectos fundamentales para la creación del producto. Bajo esta premisa, en la siguiente memoria, se presentará el desarrollo del apartado del Backend de la aplicación, exponiendo las metodologías, arquitecturas, tecnologías utilizadas y procesos seguidos, entrando al detalle en su relación con el trabajo realizado por el resto de miembros.

Además de la exposición del apartado técnico del desarrollo del producto, y por la condición de proyecto emprendedor, el aspecto del negocio cobrará una gran importancia a lo largo del documento, siendo condicionante principal en la toma de decisiones durante todo el proceso, siendo uno de los aspectos diferenciadores del siguiente trabajo. A diferencia de la distribución por partes realizada para el desarrollo, los aspectos de negocios son compartidos por todos los miembros del equipo y así quedan ilustrados en el siguiente documento.

Palabras clave: aplicación móvil, rutas, turismo, navegación, comercio

Abstract

The aim of the following document is to present the study and development process of Waltrip, a tourist route management tool for travellers. Based on a mobile application, users can search among a wide variety of available routes, save them, and browse or consult them at any time.

This work has been carried out as a Final Degree Project, in the modality of entrepreneurship, for the degree of Computer Engineering of the Polytechnic University of Valencia, in which the different aspects that surround the business and the development of the product for an entrepreneurial project are contemplated.

Waltrip is a project carried out jointly by Paula Romero Salas, in charge of the frontend section, and Alberto Morata Gomila, in charge of the backend, both students of the same degree. In this way, each of the members focuses on one of the fundamental aspects for the creation of the product. Under this premise, the following report will present the development of the Backend section of the application, explaining the methodologies, architectures, technologies used and processes followed, going into detail in relation to the work done by the other members.

In addition to the presentation of the technical section of the product development, and due to the condition of the entrepreneurial project, the business aspect will be of great importance throughout the document, being the main conditioning factor in the decision making process throughout the whole process, being one of the differentiating aspects of the following work. Unlike the distribution by parts carried out for the development, the business aspects are shared by all the members of the team and are thus illustrated in the following document.

Key words: mobile application, routes, tourism, navigation, commerce

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este capitulo se abarca el origen del proyecto emprendedor y se define la idea que será desarrollada a lo largo de todo el documento. Se describen los objetivos y la estructura del documento.

1.1 Motivación

En la época actual, viajar se ha convertido en una de las actividades más comunes dentro de nuestra sociedad. La gran oferta de transportes y los bajos precios han permitido que visitar otros lugares sea más fácil y accesible que nunca para la gran mayoría de la población.

A pesar de ello, planificar qué ver o hacer en los lugares que visitas sigue siendo una de las labores más arduas de viajar. Nos encontramos con rutas turísticas y guías estandarizadas que no contemplan las preferencias y condiciones del viajero; tours con horarios, aforos e itinerarios preestablecidos que te impide disponer de esa libertad que buscas cuando viajas; y el exceso o carencia de información que podrías encontrar que puede hacer que planificar tu propio trayecto se convierta en una odisea. Es en este contexto, en el que observamos la necesidad de una herramienta que ofrezca al viajero la posibilidad de encontrar, planificar y realizar, de manera flexible, actividades turísticas que se ajusten a sus necesidades y preferencias.

A raíz de ello, surge la idea de desarrollar una aplicación donde los usuarios puedan crear sus propias rutas turísticas, guías e itinerarios, y compartirlas con el resto. De esta forma, cualquier persona, que acceda a ella, podrá disponer de una gran variedad de oferta turística e información con la que planificar y conocer, a su manera, los lugares que visita. Basada en una app móvil, los usuarios puedrán disponer del servicio en cualquier momento y realizar turismo por su propia cuenta a la vez que se apoyan en el contenido generado por la comunidad. Sin limitación de aforos, horarios ni dependencia de ningún tipo. Un modelo diferente que no contempla ningún competidor dentro del mercado de turismo actual y que harán de cada visita una experiencia única adaptada al viajero y a sus necesidades.

1.1.1. Idea de negocio.

Tras las necesidades identificadas, ideamos desarrollar Waltrip, un *marketplace* de rutas y guías turísticas que dispone al viajero de una oferta de contenido completa, flexible y siempre disponible en su bolsillo gracias a su gestión a través de una aplicación móvil.

En concreto, la idea es ofrecer al usuario contenido turístico que se adapte a sus necesidades y permitirle realizar toda la gestión a través de una interfaz sencilla, que además, disponga de funcionalidades útiles para la planificación de sus viajes.

1.1.2. Equipo promotor.

En conjunto, el proyecto emprendedor busca desarrollar la idea y mejorar la visión del producto mediante el constante análisis del mismo. Para ello el equipo de desarrollo contará con Paula Romero Salas, que se encargará, principalmente, de todo los aspectos relacionados con el frontend de la aplicación y Alberto Morata Gomila, autor de este trabajo, encargado, principlamente, del desarrollo del backend.

A continuación, cada uno de los miembros expone sus conocimientos, actitudes, experiencias y motivaciones que le han llevado a trabajar en el proyecto:

Alberto Morata Gomila

Estudiante de Ingeniería informática y Administración y Dirección de la Empresa en la Universidad Politécnica de Valencia. Actualmente trabajando como desarrollador de backend para la compañía Meliá Hotels International. Apasionado por el mundo del emprendimiento y los mercados, busco estar al día de los últimos proyectos y tecnologías que podrían revolucionar el mundo y adoptar modelos de negocio innovadores. Autodidacta y con facilidad para el manejo de nuevas tecnologías, busco adaptarme a las últimas herramientas utilizadas en los diferentes mercados para después poder llevar las mejores prácticas a mis proyectos. Me considero una persona estoica, ambiciosa y con gran interés por generar valor para mi comunidad. Con este proyecto busco revolucionar un mercado que cada vez está menos adaptado a las necesidades cambiantes del turista actual y desarrollar una herramienta potente para el sector.

Paula Romero Salas

Estudiante de la Universidad Politécnica de Valencia en Ingeniería informática y Administración y Dirección de Empresas. En la actualidad trabajo como ingeniera de base de datos en Capgemini S.A. Siempre he sido una persona muy inquieta, y gracias a eso, mi primer pensamiento siempre es actuar y lograr mis objetivos. Autodidacta y apasionada por la innovación y la creatividad, en los últimos meses he descubierto la magia del marketing y diseño digital. Me considero una persona diligente, enérgica y con gran capacidad de adaptación. Me encanta disfrutar de los detalles y las pequeñas cosas, viajar y conocer nuevas culturas. Con este proyecto emprendedor busco adoptar un nuevo modelo de negocio que revolucione la industria turística. Con ayuda de una investigación de mercado el objetivo es conocer al nuevo turista digital que ayude a desarrollar una herramienta que mejore su experiencia de viaje.

1.2 Objetivos

El principal propósito del siguiente Trabajo Final de Grado o TFG es mostrar las primeras etapas de desarrollo, centrandonos en el Backend, de la aplicación móvil Waltrip, con la capacidad

de gestionar rutas y proporcionar información en referencia a los usuarios. Para alcanzarlo, se plantean los siguientes subobjetivos:

- Realizar un análisis financiero para proyectar la viabilidad económica del proyecto donde, además, se evalúe y analice la competencia directa para identificar las características y requisitos principales del servicio.
- Desarrollar un Backlog que permita identificar las preferencias, necesidades y expectativas de los usuarios. Validar el diseño de la aplicación a través de pruebas con usuarios reales y retroalimentarla según su feedback.
- Utilizar el framework móvil Flutter, creado por Google, para ofrecer una solución de software escalable.
- El objetivo final y más desafiante es conseguir una versión funcional que pueda ser lanzada al mercado.

Este proyecto emprendedor pone en práctica los conocimientos y habilidades adquiridas a lo largo de la carrera.

1.3 Estructura de la memoria

La memoria se compone de 7 capítulos y de 4 apéndices con información relevante para la definición del proceso de desarrollo llevado a cabo:

Capitulos

Capítulo 2, Estudio de mercado: Descripción del modelo de negocio, análisis y estudio de los competidores y de la proyección economica.

Capítulo 3, Solución propuesta: Waltrip: Metodologías, técnicas y análisis para la implementación.

Capítulo 4, Tecnologías: Herramientas utilizadas y servicios integrados para el desarrollo.

Capítulo 5, Desarrollo del primer producto mínimo viable: Descripción del proceso de desarrollo llevado a cabo para la primera versión de Waltrip.

Capítulo 6, Desarrollo del segundo producto mínimo viable: Descripción del proceso de desarrollo se la segunda versión de Waltrip.

Capítulo 7, Conclusiones: Presentación de las ideas finales como resultado del trabajo realizado.

Apéndices

Apéndice A, Manual de uso: Manuales para el uso de las diferentes versiones de la aplicación.

Apéndice B, Formulario primer MVP: Encuesta y resalados obtenidos de la opinión de los usuarios para la primera versión.

Apéndice C, Formulario segundo MVP: Encuesta y resalados obtenidos de la opinión de los usuarios para la segunda versión.

Apéndice D, Objetivos de desarrollo sostenible: Relación del proyecto con los Objetivos de Desarrollo Sostenible.

CAPÍTULO 2
Estudio de mercado.

El siguiente capitulo comprende una versión reducida del profundo estudio y análisis de mercado realizado como modelo para la ejecución de la idea emprendedora. Se realiza una revisión del modelo de negocio inicial junto la presentación de las diversas lineas de negocio planteadas, un estudio de las oportunidades, mediante el uso del diagrama DAFO y el estudio de los competidores, y la proyección de las expectativas económicas del proyecto.

2.1 Modelo de negocio inicial.

A continuación, en la figura 2.1 se expone el modelo de negocio inicial mediante la herramienta de visualización Lean Canvas [16].

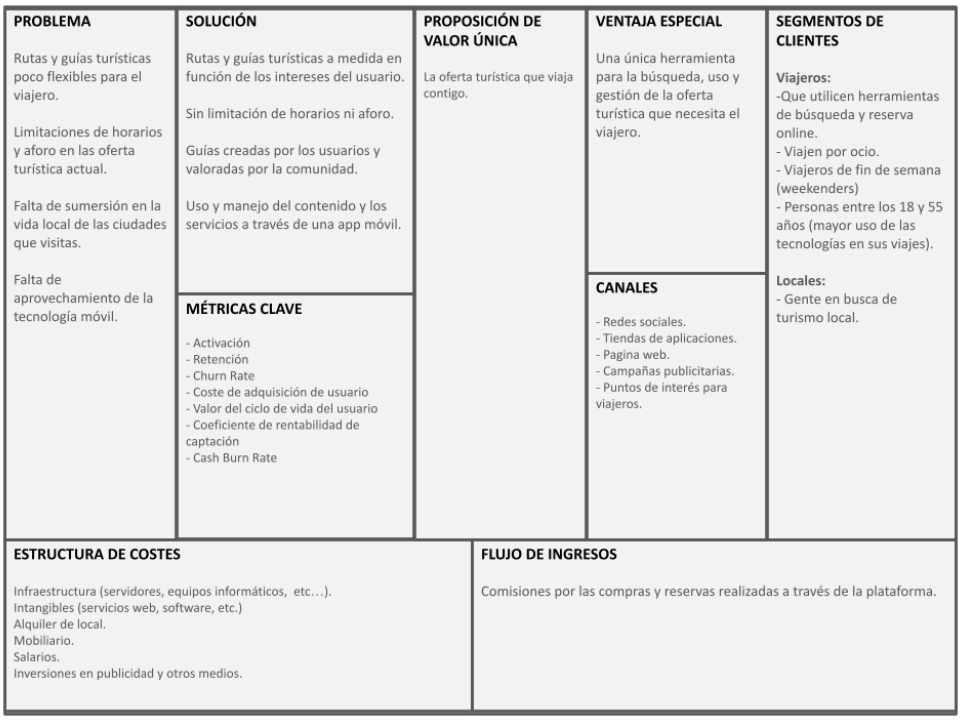


Figura 2.1: Modelo de negocio, Lean Canvas.

Como podemos observar, se exponen los problemas identificados y se detallan los diferentes aspectos que rodean a la solución propuesta Waltrip, como las diferentes soluciones que ofrece, su diferenciación frente a los competidores, canales de distribución y publico objetivo, además de su viabilidad presentando sus posibles costes y fuentes de ingreso. De esta forma, se establecen las bases para el desarrollo del negocio y el enfoque a seguir.

2.2 Lineas de negocio.

Las líneas de negocio que planteamos en nuestro modelo se basan en ofrecer diferentes servicios a través de nuestra plataforma cobrando una comisión por su uso:

En primer lugar, y como línea de negocio principal, tenemos la venta de contenido a través de la aplicación. Con este servicio, buscamos ofrecer un entorno donde viajeros y conocedores puedan compartir y obtener rentabilidad de sus experiencias y conocimientos, elaborando rutas, guías e itinerarios turísticos, conectándolos con los clientes a través de una interfaz intuitiva, cómoda y sencilla. Mediante el uso de nuestra plataforma, dispones de las herramientas para difundir tu contenido y que este sea fácilmente utilizado y accesible para el usuario, ofreciendo un sistema adecuado para ello y aprovechando las funcionalidades de la tecnología móvil para hacerlo más práctico y cómodo para el cliente. Adecuamos el contenido pensando en su empleabilidad, creando un modelo estándar de presentación y uso que aporta valor para el usuario y que no es contemplado por ningún otro competidor del mercado.

En segundo lugar, tenemos las reservas en restaurantes. La plataforma ofrecerá a los usuarios la posibilidad de realizar reserva directamente a través de la aplicación y los restaurantes podrán incentivarlas proponiendo ofertas para su local. Los usuarios podrán acceder a la oferta de restauración del lugar que deseen y se les mostrarán sugerencias de locales durante la realización de rutas. De esta forma, mediante el uso de los datos aportados por la situación del usuario, podemos encontrar al cliente potencial para cada restaurante y ofrecer así un servicio diferente y de valor añadido frente al resto de competidores.

Finalmente, contamos con la compra de entradas. Este servicio funciona de la misma forma que las reservas en restaurantes, pero con la compra de entradas para eventos y otros locales. También se tendrá en cuenta la situación y preferencias de cada usuario para las recomendaciones como punto diferencial frente al mercado actual.

Cada línea de negocio cobrará un porcentaje por transacción por la utilización del servicio como fuente principal de ingresos para la empresa.

2.3 Competidores.

Aunque ninguna aplicación turística llegue a ofrecer el mismo servicio, podemos destacar un conjunto de empresas que por su afinidad se convierten en competidores directos de Waltrip. Estas empresas son maps.me, nativoo, alltrails y wikiloc.

Tras descargar y probar las diferentes aplicaciones de estudio, elaboramos la tabla de comparación con el fin de analizar las características de la competencia que resulten interesantes incluir

en nuestro servicio, además de detectar ventajas competitivas no implementadas por ninguna de éstas. Se finaliza con una breve conclusión recopilatoria de cada una de ellas.

2.3.1. Ventaja sobre los competidores: tabla comparativa

A continuación, realizaremos un estudio en detalle de las funcionalidades contempladas por los 4 principales competidores de Waltrip, con el fin de analizar las características de la competencia que resulten interesantes incluir en nuestro servicio, además de detectar ventajas competitivas no implementadas por ninguna de éstas.

Este análisis nos servirá para definir con mayor precisión el orden de prioridad que debemos dar a las diferentes funcionalidades que serán presentadas más adelante en las distintas versiones de la aplicación, MVP. En la tabla 2.1, *Tabla comparativa competidores directos*, podemos ver las diferentes funcionalidades contempladas y su presencia en los distintos competidores.

Tabla 2.1: Tabla comparativa competidores directos

	Alltrails	Nativoo	Maps.me	Wikiloc	Waltrip
Visualizar/búsqueda rutas	✓	✓	✗	✓	✓
Visualizar contenido de la ruta (puntos de interés, valoración, fotos, descripción, mapa…)	✓	✓	✗	✓	✓
Visualizar/búsqueda rutas por ubicación	✓	✓	✗	✓	✓
Filtro de búsqueda	✓	✓	✗	✓	✓
Navegación GPS	✓	✗	✓	✓ PREMIUM	✓
Sugerencias en ruta	✓ PREMIUM	✗	✗	✗	✓
Pausar/retomar navegación	✓	✗	✗	✓	✓
Guía por pantalla 3D y por voz	✗	✗	✓	✗	✗
Mapas offline (descargar mapa)	✓ PREMIUM	✗	✓	✓	✓
Herramienta lifeline (compartir mi ubicación)	✓ PREMIUM	✗	✓ PREMIUM	✓ PREMIUM	✗

	Alltrails	Nativoo	Maps.me	Wikiloc	Waltrip
Reservar restaurantes	✗	✓	✓	✗	✓
Comprar rutas	✓	✗	✗	✓	✓
Establecer favoritos (rutas, lugares, restaurantes..)	✓	✓	✓	✓	✓
Personalizar rutas (replanear rutas guardadas)	✗	✓	✗	✗	✓
Crear rutas/itinerarios	✓	✓	✓	✓	✓
Añadir contenido ruta (informacion, precio, imagenes)	✓	✓	✓	✓	✓
Añadir información en cada localización de la ruta	✓	✓	✓	✓	✓
Etiquetar según modalidad los puntos de interés (punto gastronómico, natural, histórico, cultural)	✗	✗	✓	✗	✓
Linkear (p.ej: blogs)	✓	✓	✓	✓	✓
Compartir (lugares, perfiles, rutas, restaurantes...)	✓	✓	✓	✓	✓
Guardar mis rutas	✓	✓	✓	✓	✓
Dejar reseñas (lugares, perfiles, rutas, restaurantes...)	✗	✓	✗	✓	✓
Identificarse con usuario y contraseña	✓	✓	✓	✓	✓
Identificarse con Google o Facebook	✓	✓	✓	✓	✓
Identificación en dos pasos	✗	✗	✓	✗	✗
Perfil de usuario con seguidos y seguidores	✓	✗	✗	✓	✓
Recibir comentarios	✓	✗	✗	✓	✓
Recibir sugerencias	✗	✗	✗	✗	✓
Recibir “Me gusta”	✓	✗	✗	✓	✓
Aplicación Web	✓	✗	✗	✓	✗
Landing page	✓	✓	✓	✓	✓
Tiempo	✓	✗	✗	✓	✗
Multiplataforma	✓	✓	✓	✓	✓
Multilenguaje	✓	✓	✓	✓	✓
Publicidad (anuncios)	✓	✓	✓	✓	✗
Precio Premium **(Modelo de negocio)**	*29,99 €/año* **Freemium**	*0 €/año* **Gratuito**	*0,59-23 €/element* **Freemium**	*9,99 €/año* **Freemium**	*0 €/año* **Gratuito**

2.4 Análisis DAFO.

Con el propósito de analizar la situación a largo plazo de la empresa y tener una visión más estratégica, a continuación, en la tabla 2.2, *Matriz DAFO* [23], identificamos las posibles oportunidades y amenazas externas y las fortalezas y debilidades internas con las que puede encontrarse el proyecto.

Tabla 2.2: Matriz DAFO.

EXTERNOS	INTERNOS
Oportunidades	**Fortalezas**
❖ Mercado relacionado con las aplicaciones en expansión. ❖ Dependencia hacia el móvil. ❖ Público objetivo no limitado. Gran cantidad de clientes potenciales. ❖ Probabilidad de añadir nuevas funcionalidades y crecer internacionalmente. ❖ Colaboraciones con otras empresas del sector, como National Geographic. ❖ Englobar distintas tipologías turísticas	❖ Producto útil e innovador. ❖ Coste mínimo en la creación de la empresa. ❖ Gratuita. ❖ Posible rápida expansión por el rápido y fácil acceso. ❖ Diseño intuitivo. ❖ No es necesaria una gran infraestructura. ❖ Conocimientos técnicos.
Amenazas	**Debilidades**
❖ Covid-19. ❖ Crisis económica. ❖ Fácil aparición de sustitutivos. ❖ Desconfianza clientes ❖ El coste de las aplicaciones baja. ❖ Productos competidores fuertes. ❖ Tecnología cambiante.	❖ Falta de experiencia. ❖ Miedo a emprender. ❖ Búsqueda de clientes. ❖ Suscripción gratuita con baja rentabilidad al principio. ❖ No hay base de datos de rutas

En cuanto al entorno externo:

- **Oportunidades:**

  Resaltar un mercado tecnológico en crecimiento. Hoy en día, la enorme dependencia de los teléfonos móviles hace que el desarrollo de aplicaciones sea un negocio rentable y a explotar. Además, el segmento potencial al que va dirigido nuestro servicio son los *Millenials*, quienes están totalmente concienciados con las nuevas tecnologías y las consideran casi una necesidad en sus viajes.

Por otro lado, no tener barreras geográficas y contar con una presencia online será un gran aliado para llegar a más clientes y vender más, incluso a nivel internacional. La flexibilidad que otorga Internet permite que tu negocio se reinvente, que con el mismo esfuerzo puedas añadir nuevas funcionalidades a tu servicio, escalar horizontal o verticalmente o colaborar con otras empresas del sector como National Geographic.

- **Amenazas:** La principal amenaza externa que enfrentan nuestro servicio es la crisis económica mundial que comenzó en 2020 debido a la pandemia. El factor del COVID-19 ha cambiado las prioridades políticas, trastocado la economía, ha propiciado a una transformación digital y cambiado las tendencias o los hábitos de consumo turísticos. Además, las medidas de contención del COVID-19 han restringido la movilidad de los viajeros y generado desconfianza entre estos.
  A esta desconfianza se le suma la fácil aparición de sustitutivos. El usuario online no siente apego por la marca, simplemente busca un servicio que le facilite, en este caso el viaje. El ratio de abandono en el mercado de las aplicaciones móviles es altísimo y su coste baja. Una buena estrategia de diferenciación del servicio es imprescindible si queremos competir con los nuevos competidores o los ya asentados en el mercado y fidelizar al público.

En lo que respecta al análisis interno:

- **Fortalezas:** Que el producto sea innovador, además de gratuito y tenga un diseño intuitivo son tres de las mayores fortalezas del proyecto.
  Otra fortaleza es que para desarrollar un Producto Mínimo Viable (MVP) no es necesaria una gran infraestructura y esto tiene un coste mínimo. Además gracias a la carrera contamos con los conocimientos técnicos necesarios.

- **Debilidades:** El hecho de tener una base de datos nula es la mayor debilidad. Conseguir clientes y hacer que tu proyecto despegue es uno de los principales retos, es necesaria una buena estrategia de marketing que publicite el servicio.
  Es por esto que emprender y rentabilizar tu negocio en menos de un año es casi imposible, además el hecho de que la suscripción sea gratuita no ayuda. Por último, la falta de experiencia también puede jugar en contra y suponer una debilidad.

2.5 Proyección económica.

A continuación, en base al estudio previo realizado en *Waltrip: tu cartera de viaje. Plan de empresa* [25], vamos a mostrar una estimación del beneficio esperado para los 3 primeros años de vida del proyecto una vez haya sido consolidado como empresa, atendiendo al desglose de los ingresos y gastos en los que se espera incurrir.

2.5.1. Clientes.

Para poder llevar a cabo un calculo aproximado de las ventas que podrían llegar a realizarse en la aplicación durante estos primeros años de vida y así poder realizar la estimación de los beneficios, será necesario saber cual será el numero de usuarios que esperamos tener para dichos años.

Para ello, nos basaremos en los datos mostrados por la compañia Wikilok en sus primeros años de vida a través del Sabi[28]. De esta forma, atendiendo a su volumen de negocio, estimamos el siguiente volumen de usuarios para cada año como se muestra en la figura 2.2.

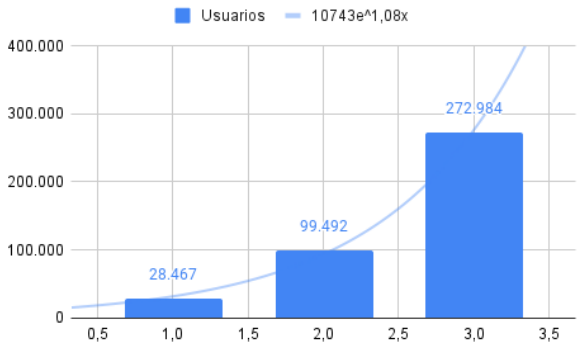


Figura 2.2: Usuarios esperados por año.

2.5.2. Beneficio esperado.

En resumen de la proyección económica esperada, en la tabla 2.3 podemos contemplar el conjunto de ingresos y gastos esperados para cada año y el beneficio obtenido en cada uno de ellos.

Tabla 2.3: Facturación esperada en los 3 primeros años.

INGRESOS			
Linea de negocio	**Año 1**	**Año 2**	**Año 3**
Comisiones por ventas usuarios	58.254,24 €	171.543,12 €	505.141,92 €
Comisiones por reservas restaurantes	0,00 €	0,00 €	841.903,20 €
Venta de entradas	0,00 €	107.214,45 €	315.713,70 €
Total Ingresos	**58.254,24 €**	**278.757,57 €**	**1.662.758,82 €**

GASTOS			
Concepto	**Año 1**	**Año 2**	**Año 3**
Gastos de personal	0,00 €	61.655,54 €	136.395,00 €
Costes variables			
Distribución	105,80 €	84,47 €	84,47 €
Subcontrataciones	25.631,87 €	104.887,59 €	312.210,86 €
Gastos generales			
Alquileres	0,00 €	1.200,00 €	2.400,00 €
Servicios web	240,00 €	240,00 €	240,00 €
Publicidad	3.500,00 €	12.000,00 €	24.000,00 €
Otros Gastos	18.000,00 €	21.000,00 €	24.000,00 €
Gestoria laboral	0,00 €	0,00 €	1.500,00 €
Gestoria fiscal	0,00 €	0,00 €	3.000,00 €
Seguros	1.800,00 €	1.980,00 €	2.160,00 €
Tributos	0,00 €	1.166,49 €	11.356,50 €
Constitución	600,00 €	0,00 €	0,00 €
Total Gastos	**49.877,67 €**	**204.214,08 €**	**517.346,83 €**
Beneficio	**8.376,57 €**	**74.543,49 €**	**1.145.411,99 €**

CAPÍTULO 3
Solución propuesta: Waltrip.

En el siguiente capitulo se expondrá la solución Waltrip, las decisiones previas al proceso de desarrollo exponiendo la metodología y arquitectura de software aplicada junto a una periodización detalla de los diferentes aspectos a implementar mediante el uso del Backlog.

3.1 Metodología.

Para la gestión del proyecto y llevar acabo el desarrollo de la aplicación, decidimos optar por el uso de la metodología ágil conocida como Kanban [24].

Kanban es una metodología ágil de gestión visual que se utiliza para mejorar el flujo de trabajo en equipos y proyectos. La metodología Kanban se basa en el uso de tableros visuales que representan el flujo de trabajo y las tareas en diferentes etapas. Estos tableros suelen estar divididos en columnas que representan el estado de las tareas, desde el inicio hasta la finalización. Cada columna puede tener un límite de trabajo en progreso (Work In Progress, WIP), lo que evita la sobrecarga del equipo y fomenta la colaboración.

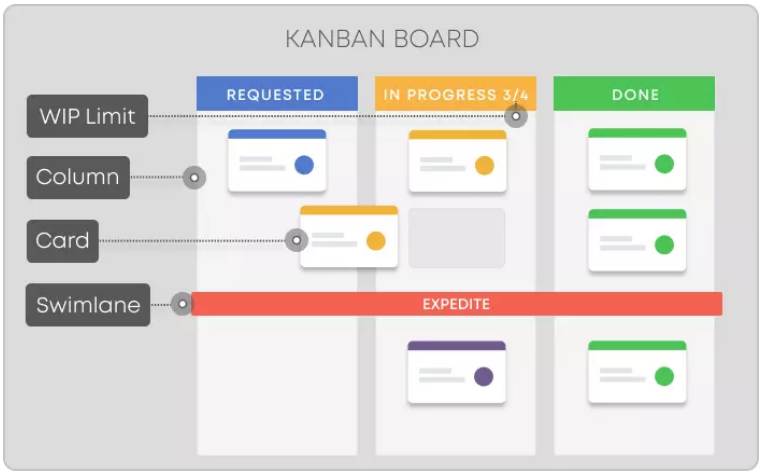


Figura 3.1: Metodología Kanban y sus elementos.

Entre los motivos por los que decidimos utilizar esta metodología destacamos:

- **Visibilidad y transparencia:** Kanban proporciona una visualización clara y en tiempo real del flujo de trabajo y el estado de las tareas. Esto permite a todos los miembros del equipo y a los interesados comprender fácilmente el progreso del proyecto y las tareas que están en curso.
- **Flexibilidad y adaptabilidad:** Kanban se adapta bien a proyectos y equipos de diferentes tamaños y características.
- **Entrega continua de valor:** El trabajo se distribuye en pequeñas tareas permitiendo el incremento de de funcionalidades de manera regular, sin tener que esperar al resultado final de la versión. De esta forma, permite obtener retroalimentación temprana y realizar ajustes según las necesidades.
- **Facilidad de implementación:** Kanban es relativamente sencillo de implementar, especialmente en comparación con otras metodologías ágiles más complejas como Scrum. Esto nos permite comenzar a desarrollar rápidamente y obtener resultados desde el principio.

En general, nos decidimos en su utilización por su enfoque en la visualización, la flexibilidad y la mejora continua, lo que lo convierten, dada las incertidumbres que rodean al proyecto, en una de las mejores opción para el desarrollo de Waltrip.

3.2 Arquitectura.

El patrón de diseño escogido para la arquitectura de la aplicación ha sido el de Controlador - Servicio - Repositorio [5] propio de la aplicaciones desarrolladas en Spring [29].

En nuestro caso, hemos decido utilizar este modelo por ya contar con experiencia previa en su uso y facilitarnos la implementación de nuevas funciones para la interfaz abstrayendo la lógica de negocio de la de la propia aplicación.

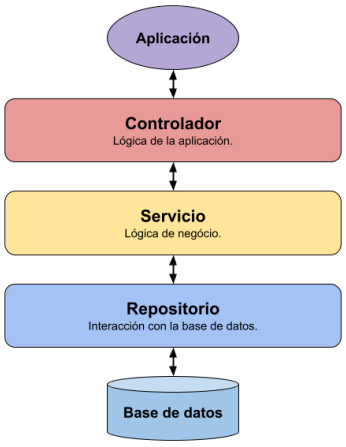


Figura 3.2: Arquitectura Controlador - Servicio - Repositorio.

En este modelo, los objetos de negocio (que contienen la información a gestionar por la base de datos) son definidos como entidades, y son las utilizadas por la capa de servicio para gestionar la información que se intercambia entre la de repositorio y controlador.

De forma detallada, cada una de las capas realiza las siguientes funciones:

- **Repositorio:** Donde se definen las funciones que llaman directamente a la base de datos devolviendo, insertando, eliminando o actualizando la información.
- **Servicio:** Donde se gestionan las entidades utilizadas por la aplicación para que estos puedan ser convertidos en información valida de la base de datos y viceversa.
- **Controlador:** Donde se definen las funciones con las que trabajará la interfaz de la aplicación, abstrayendo la lógica de la aplicación de la de negocio.

De esta forma, la organización del apartado del backend del proyecto queda definida como se refleja a continuación, en la figura 3.3:

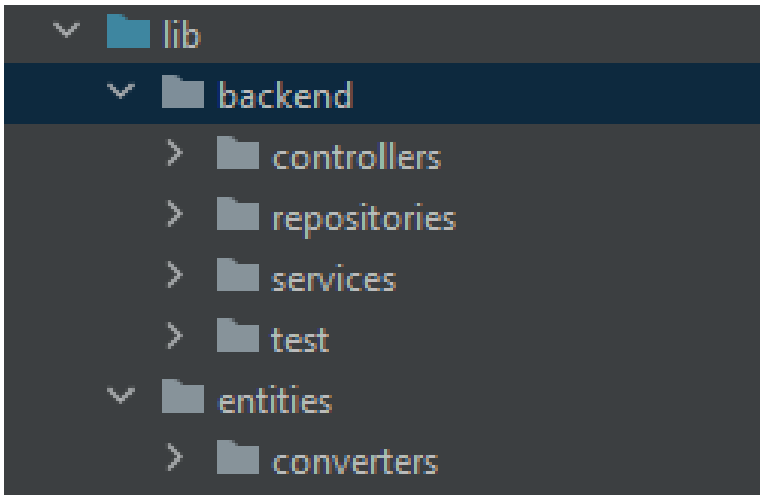


Figura 3.3: Organización del backend.

En esta configuración, las clases implementadas en la carpeta *services* (capa servicio) llaman a las de *repositories* (capa repositorio) para la interacción con la base de datos y, de forma similar, las clases en la carpeta *controllers* (capa controlador) llaman a las de*services* para la gestión de la lógica de negocio.
En la carpeta *entities* se implementan las entidades con las que la capa de servicio trabajará para el manejo de datos entre la capa repositorio y controlador. Además, se implementan los conversores en la carpeta *converters* para la conversión de las entidades a objetos validos entre ambas capas.

3.3 Funcionalidad

Antes de comenzar con el desarrollo de las primeras versiones de Waltrip o MVP, que incluirá las características principales necesarias para el lanzamiento de la aplicación al mercado, debemos priorizar el Backlog teniendo en cuenta los diferentes factores que afectarán en su implementación.

En el Capítulo 5 del TFG *Waltrip: test de concepto* [27], donde se realiza un estudio de mercado más detallado del producto, se presentan las necesidades e intereses de nuestro posible público objetivo, análisis en el que debemos enfocarnos a la hora de definir las funciones y su prioridad, con el fin de obtener retroalimentación, en forma de encuestas, por parte de los usuarios sobre las principales funcionalidades y, de esta forma, poder ajustar el enfoque del desarrollo en etapas tempranas. Es por ello, que para las primeras versiones se implementarán las funciones consideradas de más interés entre los usuarios o que son imprescindibles para la puesta en marcha de la aplicación. Entre estas desatacamos el diseño de rutas, su visualización y la posibilidad de navegar por ellas.

Por otro lado, el análisis de la competencia nos da una idea de qué características funcionales son interesantes a la hora de diferenciar nuestro producto o generar valor añadido. Una característica interesante a incluir y que comparten nuestros competidores es la descarga offline de mapas o "mapas sin conexión". Otra interesante, y menos compartida, es la personalización de rutas. Es decir, a partir de las rutas de otros usuarios añadir puntos de paso, o modificar la salida y llegada de la ruta a nuestro gusto, y añadirla a nuestra cartera. Además, para diferenciar el producto, una buena alternativa que no ofrecen nuestros competidores radica en la comunidad, buscando generar una red social con perfiles de usuario que reciban puntuaciones, sugerencias, comentarios, etc.

Finalmente, para obtener rentabilidad financiera debemos añadir la capacidad de realizar compras a través de la aplicación. Que los usuarios puedan, además de adquirir y crear rutas gratuitamente, comercializarlas por un precio, permitiéndonos monetizar la aplicación y generar unos primeros ingresos gracias al cobro de una comisión por transacción.

Teniendo en cuenta el análisis realizado, las principales características funcionales previstas a implementar y conformarán el Backlog para el desarrollo son:

- Crear, visualizar y navegar por las rutas.
- Poder identificarse y disponer de un perfil de usuario en la aplicación.
- Añadir rutas a favoritos y guardarlas para disponer de ellas en otro momento.
- Personalizar rutas.
- Compra de rutas.

3.3.1. Mapa de características

Con el fin de organizar las funciones y definir un orden por prioridad en su desarrollo, a continuación, en la figura 3.4, se presenta un mapa de características con el conjunto de funcionalidades ordenado:

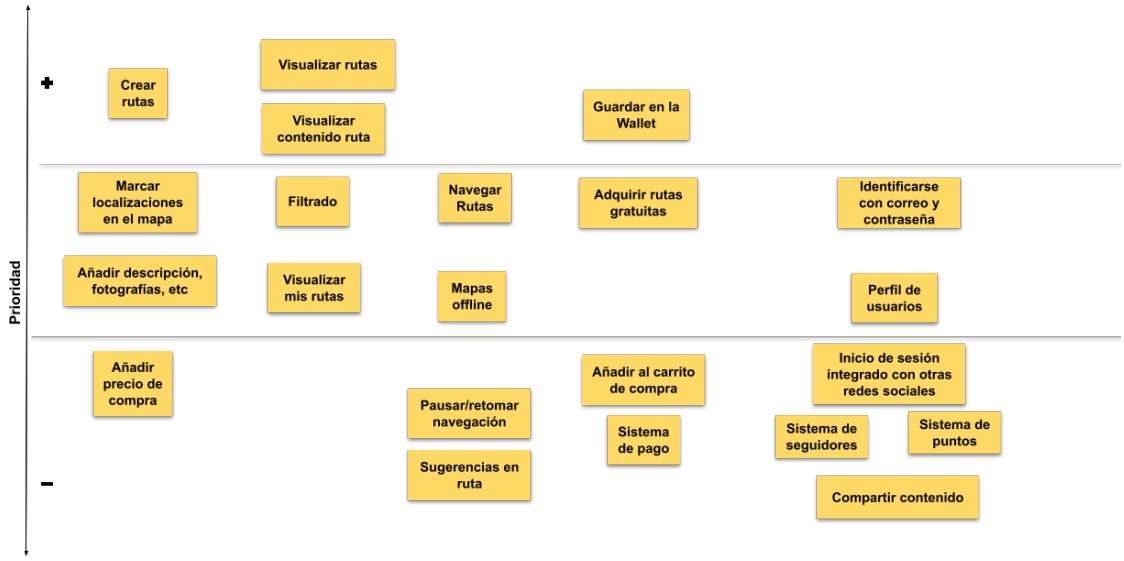


Figura 3.4: Mapa de características

Como podemos observar, las funciones se clasifican por prioridad de arriba hacia abajo, siendo de mayor prioridad cuanto mas arriba se posicionen. De esta forma, gracias a esta clasificación, podremos definir el alcance de cada una de las versiones presentadas, haciendo hincapié en las funciones de más prioridad y valorando su viabilidad en su implementación.

Entre las funciones que se plantearán encontramos:

- **Visualización del contenido:** Permitir la visualización del contenido disponible en la aplicación.
- **Contenido de las rutas:** Permitir acceder a las rutas y ver su información.
- **Guardado de rutas:** Permitir a los usuarios guardar rutas para disponer de ellas posteriormente.
- **Creación de rutas:** Permitir a los usuarios crear sus propias rutas y que esta pasen a estar disponibles dentro del contenido ofrecido por la aplicación.
- **Perfil de usuario:** Crear un espacio para el usuario donde pueda ver y gestionar toda la información referente a su perfil.
- **Registro y acceso de usuarios:** Controlar el acceso de usuarios mediante el registro y necesidad de credenciales para entrar en la aplicación.
- **Filtrado de contenido;** Poder filtrar el contenido por distintos atributos para facilitar la busqueda de rutas.
- **Mapas y navegación:** Permitir el tratamiento de mapas y la navegación en ruta a través de la aplicación.

- **Edición de rutas:** Permitir al usuario modificar rutas y personalizarlas para ajustarlas a sus necesidades.
- **Compras:** Permitir transacciones de pago en la aplicación para poder aplicar precio a las rutas y que estas puedan se adquiridas pagando un importe por los usuarios.
- **Valoraciones:** Permitir a los usuarios puntuar y comentar las rutas, valorando el contenido en la aplicación.

CAPÍTULO 4
Tecnologías.

En el siguiente capitulo se presentan las herramientas utilizadas y servicios que se integrarán para la gestión del backend de Waltrip.

4.1 Herramientas para el desarrollo.

4.1.1. Android Studio.

Android Studio[3] es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) diseñado específicamente para el desarrollo de aplicaciones para dispositivos Android, aunque también permite el desarrollo para otras plataformas. A parte de las que podemos encontrar en cualquier IDE, proporciona un conjunto de herramientas y características que simplifican el proceso de desarrollo de aplicaciones y entre las que desatacamos:

- **Emulador y dispositivos virtuales:** Incluye un emulador de Android que permite ejecutar y probar aplicaciones en diferentes dispositivos virtuales con diversas configuraciones de hardware y versiones. Esto nos facilita el proceso de ejecución de pruebas y depuración de la aplicación.

- **Asistente de diseño y generador de interfaces:** Incluye herramientas que facilitan y agilizan la creación de interfaces, además de contar con componentes de terceros que optimizan esta labor y, de otras funcionalidades, como el diseño multidispositivo, lo que nos ayudará en la disponibilidad de la aplicación en los diferentes terminales.

- **Integración con herramientas de desarrollo de terceros:** Fácil gestión de la integración de herramientas que nos permitirá trabajar de forma adecuada con el resto de tecnologías planteadas para el proyecto.

Nos proporciona un conjunto completo de herramientas que facilitan el desarrollo, la depuración y la optimización de aplicaciones, lo que permite crear aplicaciones móviles de alto rendimiento y calidad.

4.1.2. Flutter.

Flutter[12] es un framework de código abierto desarrollado por Google que permite crear aplicaciones multiplataforma de manera rápida y eficiente. Utiliza el lenguaje de programación Dart[6], también desarrollado por Google, y se destaca por su enfoque en la creación de interfaces de usuario atractivas y fluidas.

Una de las características más destacadas de Flutter es su capacidad para compilar aplicaciones nativas tanto para iOS como para Android a partir de un solo código base, lo que significa que no necesitaremos escribir y mantener dos conjuntos de código separados para cada plataforma, lo que permite un desarrollo más rápido y menos propenso a errores.

En cuanto al front, Flutter utiliza un enfoque de "pintura"para construir la interfaz de usuario, lo que permite personalización y control precisos, y ofrece widgets adaptables para crear interfaces modernas y consistentes en diferentes plataformas y tamaños de pantalla.

Otra ventaja de Flutter es la amplia disponibilidad de documentación, tutoriales y paquetes facilitan la implementación de funcionalidades en las aplicaciones. Además, cuenta con herramientas de desarrollo eficientes, como Hot Reload, que permiten ver los cambios en tiempo real, lo que agiliza el proceso de desarrollo y facilita la depuración.

Flutter nos permite crear aplicaciones multiplataforma de manera rápida y eficiente, con una apariencia atractiva y un rendimiento sólido. Es por ello que lo elegimos como basé para el desarrollo de nuestra aplicación.

4.1.3. Git y GitHub

Git[31] es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Proporciona una forma eficiente de rastrear y gestionar los cambios realizados en el código fuente a lo largo del tiempo. De esta forma, podemos realizar un seguimiento de las modificaciones, revertir cambios no deseados y fusionar fácilmente las contribuciones de los diferentes miembros del equipo.

Por otro lado, GitHub[15] es un servicio de alojamiento de repositorios de código que permite almacenar, gestionar y colaborar en proyectos utilizando Git. Proporciona una interfaz amigable que facilita la gestión de proyectos, permitiendo la revisión de cambios y la colaboración en equipo.

En conjunto, estas herramientas nos permitirán trabajar de forma más eficiente al equipo, disponiendo del código siempre en linea, permitiéndonos trabajar a cada uno en diferentes implementaciones y evitando perder el control sobre el código.

4.2 Configuración de las herramientas

A continuación, realizaremos la configuración del IDE Android Studio y le integraremos las diferentes herramientas necesarias para el desarrollo de la aplicación.

4.2.1. Configuración del IDE

El entorno de desarrollo escogido para llevar acabo el proyecto ha sido Android Studio[3]:

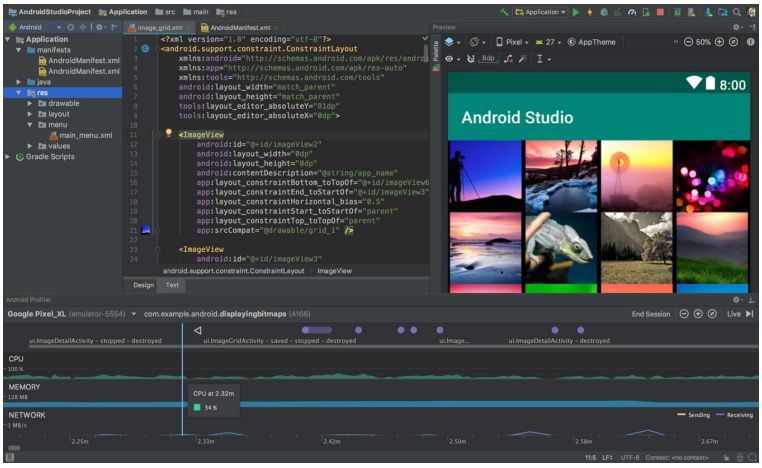


Figura 4.1: Interfaz de Android Studio.

Un IDE de desarrollo basado en IntelliJ IDEA[18] para aplicaciones Android gratuito con Licencia Apache 2.0[4] creado por Google LLC como remplazo a otras opciones como Eclipse[7] o Netbeans[26].

Para poder comenzar a desarrollar nuestras aplicaciones en el entorno, primero necesitaremos instalar el Kit de desarrollo de Java (JDK)[19] que nos permitirá la creación, construcción y ejecución de código java. En nuestro caso, utilizaremos la versión 11 del JDK para Windows, en concreto la 11.0.12.

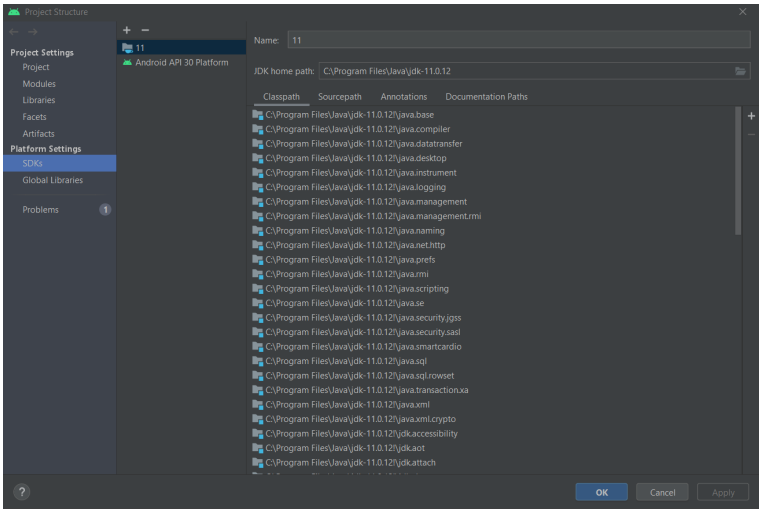


Figura 4.2: JDK del proyecto.

Por otro lado, también necesitaremos disponer del SDK de Android que nos ofrecerá un emulador, herramientas de desarrollo y las librerías necesarias para la creación de aplicaciones para sistemas Android. En nuestro caso, trabajaremos con el SDK para Android 11.0 que trabaja con la versión 30 de la API de Android, la cual permite a las aplicaciones interactuar con dicho sistema.

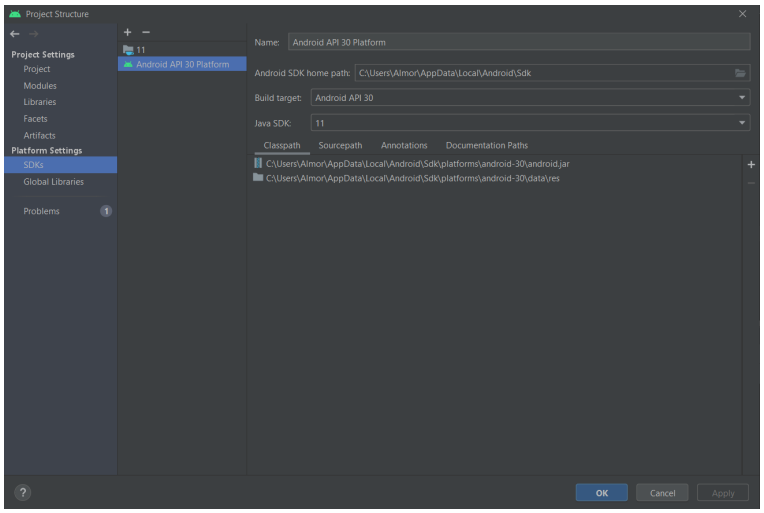


Figura 4.3: Android API del proyecto para Android 11.0.

Añadir también que el entorno cuenta con todo lo necesario para trabajar con Kotlin[21], un lenguaje de programación desarrollado por JetBrains[20] que facilita el desarrollo de aplicaciones en Java y que también utilizaremos para el desarrollo de nuestra aplicación.

De esta forma, ya tenemos preparado el entorno de desarrollo con la configurado básica para poder empezar a trabajar en la creación de nuestras aplicaciones.

4.2.2. Integración con Flutter

Para el desarrollo de la interfaz, nuestra aplicación utilizará el SDK Flutter[12]. Por ello, para poder trabajar en el desarrollo de nuestra aplicación, primero debemos contar con dicho kit de desarrollo en nuestro entorno.

Para ello, debemos descargar e instalar el SDK a través de la documentación de Flutter[13], en nuestro caso la versión 2.2.3 para Windows, y el plugin para Android Studio que nos permitirá su integración en el entorno de desarrollo.

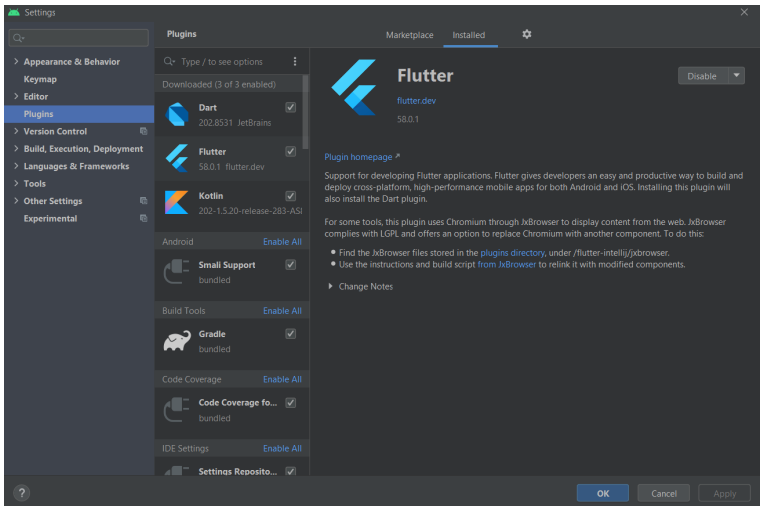


Figura 4.4: Plugin de Flutter para Android Studio.

Una vez instalado el SDK definimos la dirección en los ajustes del IDE para su integración y, poder así, empezar a utilizarlo en nuestros proyectos.

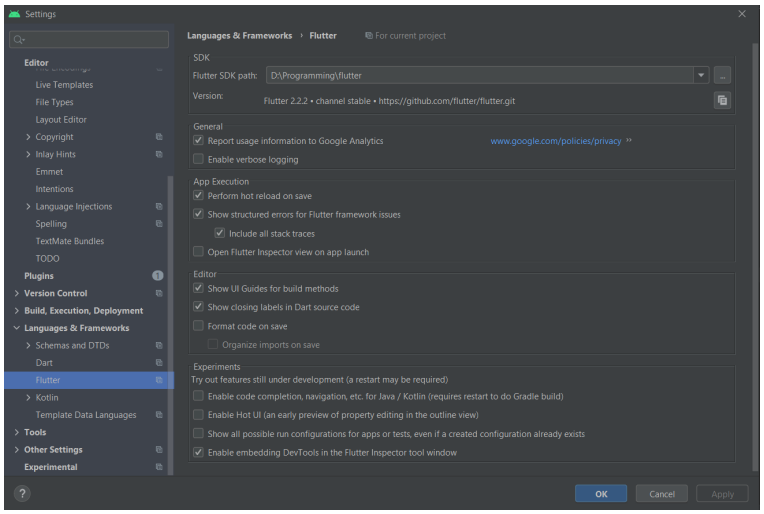


Figura 4.5: Configuración Android Studio del SDK Flutter.

4.2.3. Integración con Git y GitHub

Para llevar a cabo el control de versiones, integraremos la herramienta Git[31] y utilizaremos la plataforma GitHub[15] para tener almacenado el proyecto y toda la información relativa a los cambios en la nube, además de permitirnos compartir el proyecto entre los participantes del grupo y tenerlo disponible siempre a través de la Web.

Para ello, descargaremos la versión de Git 2.32.0 para Windows y definiremos su ruta en Android Studio.

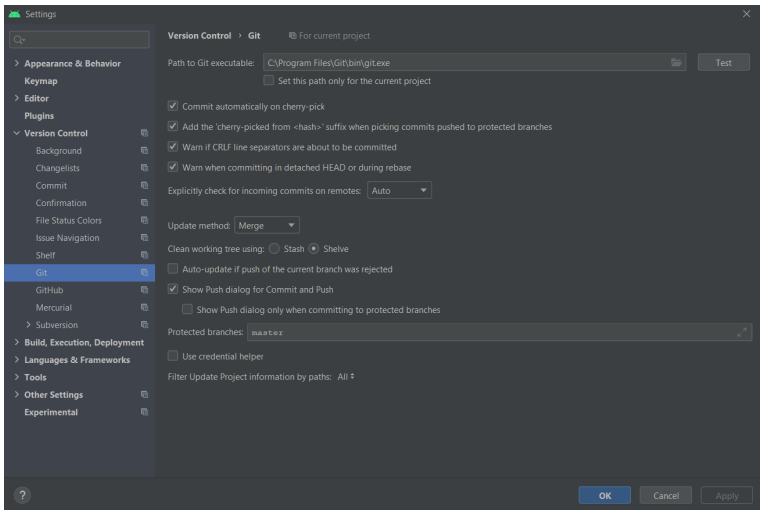


Figura 4.6: Configuración Android Studio de Git.

Tras esto, solo nos quedará acceder a nuestra cuenta de GitHub a través de Android Studio para poder sincronizar el proyecto con nuestro repositorio en la plataforma. En mi caso, el usuario de mi cuenta en GitHub es *Almorat*[1].

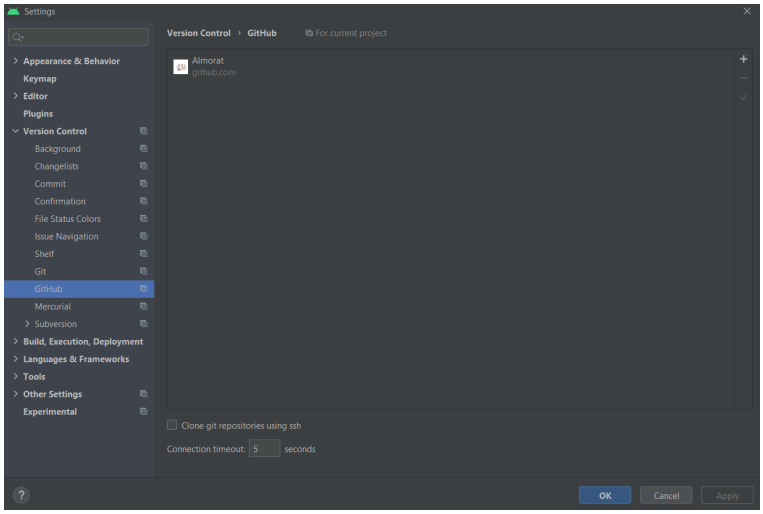


Figura 4.7: Conectar cuenta de GitHub en Android Studio.

4.3 Servicios.

4.3.1. Firebase

Firebase[9] es una plataforma de desarrollo de aplicaciones móviles y web ofrecida por Google. Proporciona una variedad de servicios y herramientas que simplifican y aceleran el desarrollo de aplicaciones, sin necesidad de configurar y administrar infraestructuras complejas.

Algunas de las herramientas a desatacar y por las cuales decidimos utilizar esta tecnología son:

- **Base de datos en tiempo real:** Firebase proporciona una base de datos en tiempo real y sincronizada en la nube, lo que significa que los datos se actualizan en tiempo real para todos los usuarios de la aplicación.

- **Autenticación de usuarios:** Permite autenticar y autorizar usuarios fácilmente mediante diferentes métodos, como el inicio de sesión con cuentas de Google, Facebook, Twitter, etc.

- **Almacenamiento en la nube:** Firebase ofrece almacenamiento en la nube para almacenar y servir archivos estáticos como imágenes, videos o documentos.

En su conjunto, estas herramientas nos permitirán gestionar la información referente a las rutas y lo usuarios, manteniéndola en un mismo entorno y facilitando la gestión de la lógica de la aplicación. Firebase se caracteriza por su fácil integración con aplicaciones móviles y web, su escalabilidad, buena documentación y ofrecer una solución integral para el desarrollo. Además se encuentra disponibles para Android, IOS y web, convirtiendola en una gran opciíon para las aplicaciones multiplataforma .

4.3.2. Mapbox

Mapbox[22] es una plataforma de mapas y localización en línea que ofrece herramientas y servicios para desarrolladores y empresas. Permite integrar mapas interactivos y funciones de localización en aplicaciones móviles y web, brindando una amplia gama de opciones personalizables y escalables, conviertiéndose en la herramienta ideal para la gestión de rutas en la aplicación. De entra los componenetes que oferece, desatacamos:

- **Mapbox Studio:** Una herramienta de diseño que permite a los usuarios crear y personalizar mapas a medida. De esta forma, podemos generar mapas acorde a la estética que busquemos para la aplicación.

- **Mapbox Maps SDKs:** Ofrece SDKs para varias plataformas, como Android, iOS y web, que permite incorporar mapas interactivos y funciones de localización en las aplicaciones. Ideal para aplicaciones multiplataforma.

- **Mapbox Navigation SDK:** SDK especializado para la creación de aplicaciones de navegación y direcciones paso a paso. Proporciona instrucciones de giro por giro, información de tráfico en tiempo real, cálculo de rutas eficientes y funciones avanzadas de navegación. Un componente que será clave para la implementación de la navegación por las rutas.

- **Mapbox Geocoding API:** roporciona servicios de geocodificación, que permiten convertir direcciones y nombres de lugares en coordenadas geográficas (latitud y longitud), y viceversa. Componente para la gestión de coordenadas.

La personalización, la calidad de los datos geoespaciales, su capacidad en la navegación y escalabilidad son algunos de los aspectos destacados en la decisión de utilizar este servicio.

4.4 Configuración del proyecto y servicios

4.4.1. Estructura inicial del proyecto

Antes de comenzar con el desarrollo de nuestra aplicación, vamos a definir una estructura inicial sobre la que empezar a trabajar y, en nuestro caso, elaborar todo el contenido relacionado con el backend de nuestro proyecto.

A continuación, en la figura 4.8, podemos ver la estructura inicial del proyecto.

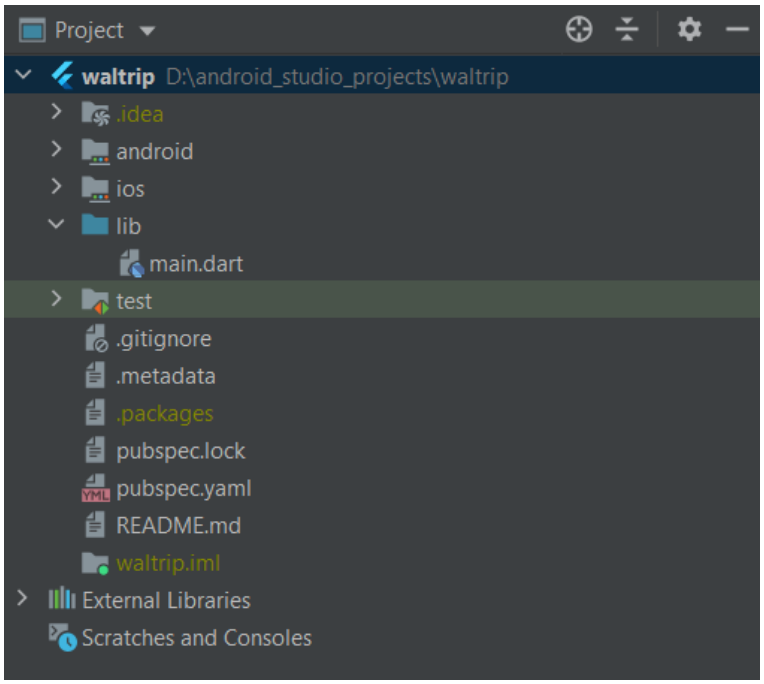


Figura 4.8: Estructura inicial del proyecto.

Dentro de la estructura del proyecto encontramos el siguiente conjunto de directorios:

- **android**: El directorio Android contiene el conjunto de archivos y carpetas necesarias para la ejecución de la aplicación en los sistemas Android.
- **ios**: De la misma forma que el directorio Android, contiene el conjunto de archivos y carpetas necesarias para la ejecución de la aplicación en los sistemas iOS.
- **lib**: Directorio en el que se almacena gran parte del código escrito en Dart. En él encontramos el archivo *main.dart* que es el punto de entrada a la aplicación, es decir, el primer archivo en ejecutarse.
- **.gitignore**: Archivo que especifica otros archivos ocultos del IDE que deben ser ignorados al compartir el proyecto a través de GitHub.
- **.metadata**: Archivo en el que se encuentran los meta datos de un proyecto en Flutter.

- **.packages**: Archivo que contiene la ruta de los paquetes y librerías utilizadas en el proyecto.

- **pubspec.lock**: Archivo que contiene la versión de cada paquete y dependencia utilizada en aplicaciones en Flutter.

- **pubspec.yaml**: Archivo utilizado para añadir meta datos y configuraciones especificas a nuestra aplicación.

- **README.md**: Archivo utilizado para describir la aplicación en el repositorio de GitHub.

- **waltrip.iml**: Modulo creado por el IDE para almacenar información de desarrollo.

Finalmente, para obtener una vista más detallada del proyecto, a continuación, en la fogura 4.9 y 4.10 observamos la estructura dentro del directorio *android* e *ios* respectivamente.

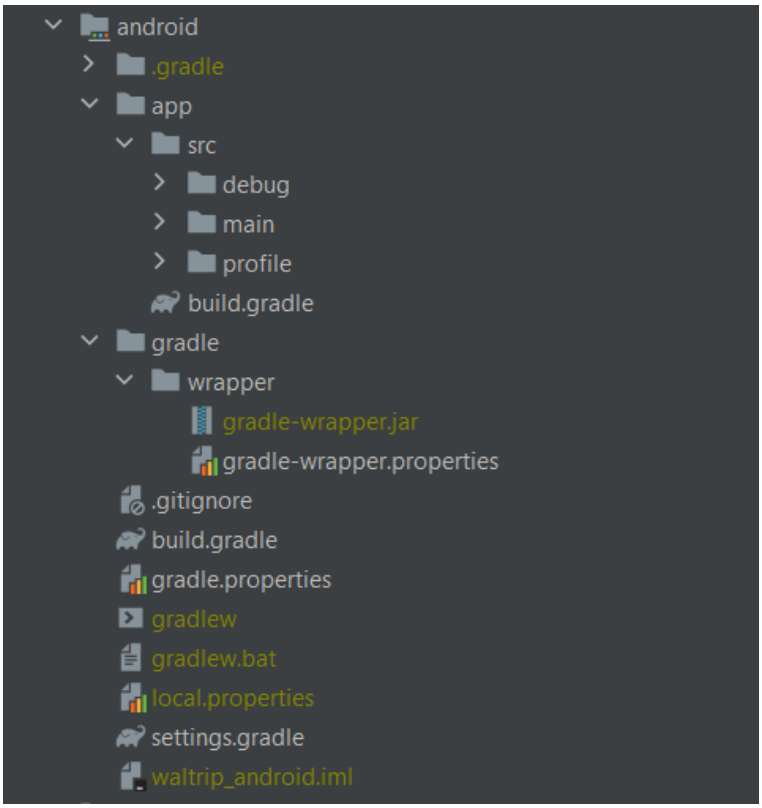


Figura 4.9: Estructura directorio android.

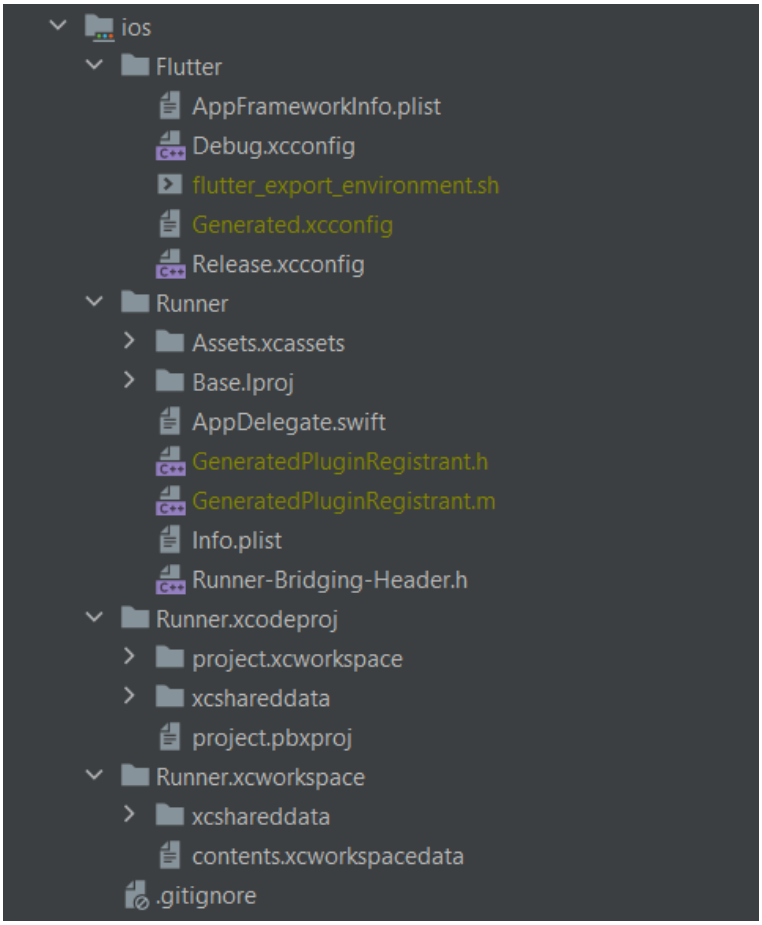


Figura 4.10: Estructura directorio ios.

4.4.2. Control de versiones

Para llevar a cabo el control de versiones del proyecto, dispondremos de un repositorio en la plataforma GitHub donde alojaremos el proyecto y gestionaremos diferentes ramas para llevar a cabo el control de los cambios que vayamos realizando sobre nuestra aplicación.

En la figura 4.11, podemos ver el repositorio *Waltrip*[2] creado para alojar el proyecto en la plataforma y poder así ser compartido con el resto de miembros del equipo de desarrollo.

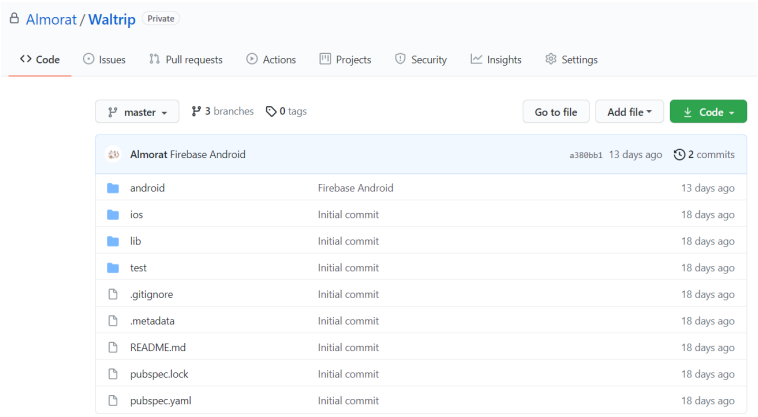


Figura 4.11: GitHub: repositorio Waltrip.

Gracias a la utilización de esta herramientas, los participantes del proyecto pueden subir sus cambios a la plataforma y mantener el proyecto actualizado en su entorno de desarrollo.

Por otro lado, en la figura 4.12, se muestran las ramas creadas para el control de versiones del proyecto.

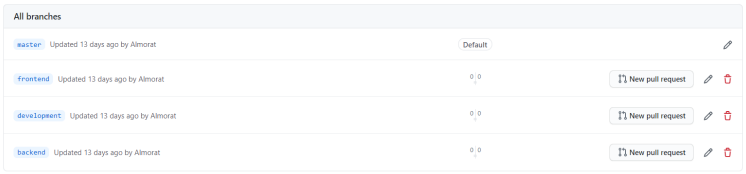


Figura 4.12: GitHub: ramas del repositorio Waltrip.

Como podemos ver, entre las ramas disponibles para el desarrollo del proyecto encontramos:

- **master**: Rama principal del proyecto. En ella se encuentra la versión estable de la aplicación y de la que dispondrá el usuario final.
- **development**: Rama secundaria y previa a la petición de subida a *master*. En ella se encuentra la versión con los últimos cambios propuestos para el proyecto y donde se realizará su validación antes de pasar a la rama *master*.
- **frontend**: Rama terciaria y previa a pasar a formar parte de la rama *development*. En ella se encuentra el código con los últimos cambios aplicados al frontend de la aplicación.

- **backend**: Rama terciaria y previa a pasar a formar parte de la rama *development*. De la misma forma que la de *frontend*, en ella se alojará el código con los últimos cambios aplicados al backend de la aplicación.

En mi caso, para el desarrollo del backend estaremos trabajando sobre la rama *backend* a la que iremos subiendo los cambios para después testearlos junto al frontend en la rama *development* y para que, finalmente, sean aplicados a la versión final en la rama *master*.

4.4.3. Integración con Firebase

Para el desarrollo del backend, integraremos en nuestro proyecto los servicios de Firebase, que nos ofrecerá una base de datos donde almacenar la información de nuestra aplicación, y nos ayudará con la gestión y manejo de los datos.

Para ello, hemos creado un nuevo proyecto llamado *Waltrip* al que le agregaremos la versión para Android e IOS de nuestra aplicación.

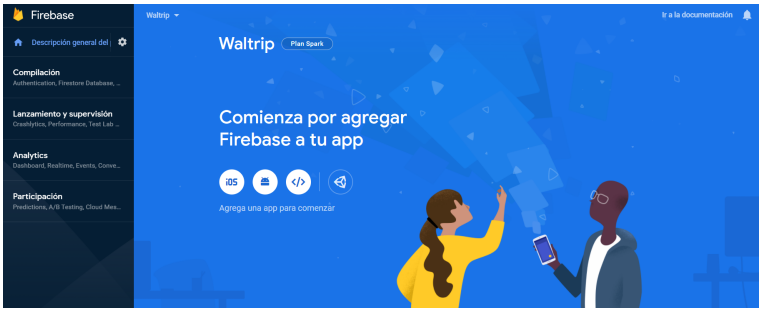


Figura 4.13: Pantalla principal Firebase: Proyecto Waltrip.

Por otro lado, para conectar la API de Firebase con nuestra aplicación en Flutter, añadiremos también la dependencia *firebase_core*[11] al archivo *pubspec.yaml* del proyecto definido para ello:

```
dependencies:
  firebase_core: ^1.4.0
```

De esta forma, ya podemos empezar a utilizar las librerías disponibles en el paquete y empezar a trabajar con los servicios de Firebase.

Firebase para Android

A continuación, añadiremos la versión en Android de nuestra aplicación al proyecto Flutter para poder trabajar con el servicio a través de este sistema.

Para ello, como podemos ver en la figura 4.14 primero definimos el nombre del paquete de Android, en nuestro caso *com.waltrip.app*, y le añadimos el sobrenombre *Waltrip* para su representación en Firebase.

Figura 4.14: Firebase: Registrar aplicación Android.

Tras esto, como se muestra en la figura 4.15 descargamos el archivo de configuración y lo introducimos en nuestro proyecto.

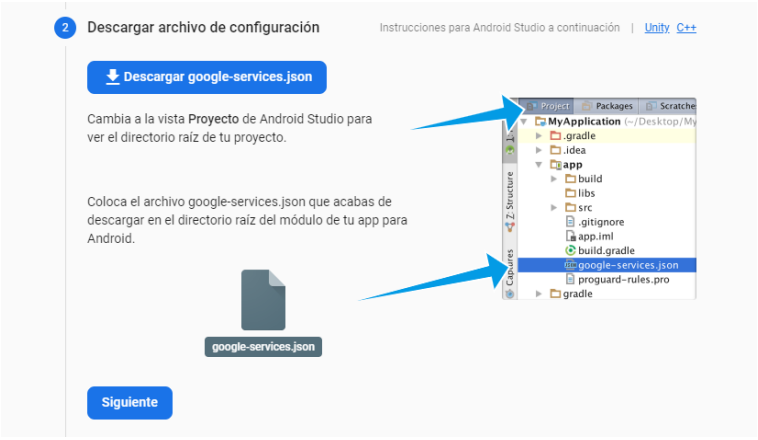


Figura 4.15: Firebase: Archivo de configuración Android.

En el archivo *google-services.json* encontramos el siguiente código de configuración:

```
{
  "project_info": {
    "project_number": "260715873336",
    "project_id": "waltrip-app",
    "storage_bucket": "waltrip-app.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:260715873336:android:5f987c651d75721c31d2cd",
        "android_client_info": {
          "package_name": "com.waltrip.app"
        }
      },
      "oauth_client": [
        {
          "client_id": "260715873336-gnarl8bm2pjh3k1rf7b0h3brf5fiien6.apps.
              googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyC7wM37SVkCO1htf3R8RPekX5DxW7qjpGM"
        }
      ],
      "services": {
        "appinvite_service": {
          "other_platform_oauth_client": [
            {
              "client_id": "260715873336-gnarl8bm2pjh3k1rf7b0h3brf5fiien6.apps.
                  googleusercontent.com",
              "client_type": 3
            }
          ]
        }
      }
    }
  ],
  "configuration_version": "1"
}
```

A continuación, agregamos el SDK de Firebase añadiendo el código necesario al archivo *build.gradle* de nivel de proyecto, figura 4.16, y al de nivel de app, figura 4.17, para el cual elegimos la versión para Kotlin.

Figura 4.16: Firebase: SDK a nivel de proyecto.

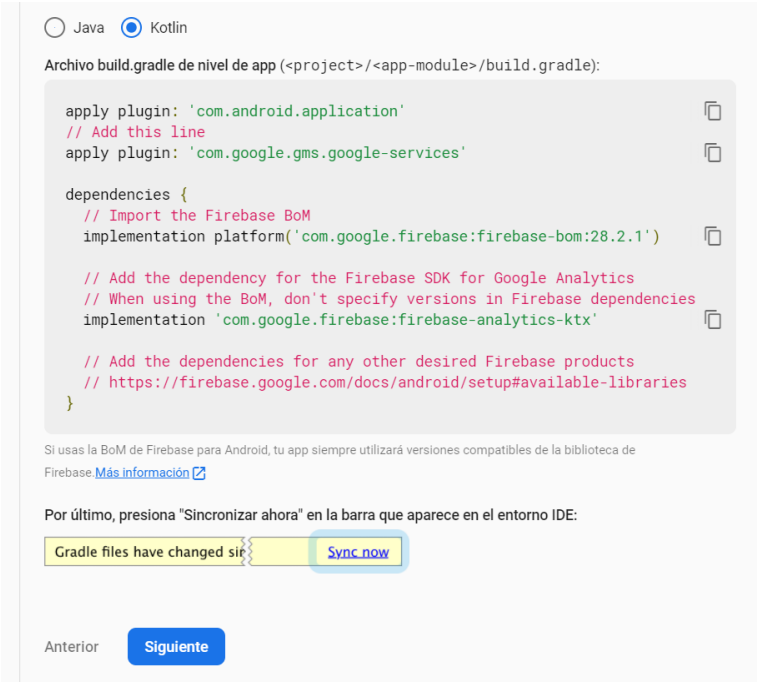


Figura 4.17: Firebase: SDK a nivel de aplicación.

De esta forma, nos queda el siguiente código en el *build.gradle* a nivel de proyecto:

```
buildscript {
    ext.kotlin_version = '1.3.50'
    repositories {
        google()
        jcenter()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:4.1.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
        classpath 'com.google.gms:google-services:4.3.8'
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

...
```

Y a nivel de app:

```
...

apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
apply plugin: 'kotlin-android'
apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"

...

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation platform('com.google.firebase:firebase-bom:28.2.0')
    implementation 'com.google.firebase:firebase-analytics-ktx'
}
```

En la figura 4.18 podemos observar la estructura del directorio *android* tras la integración.

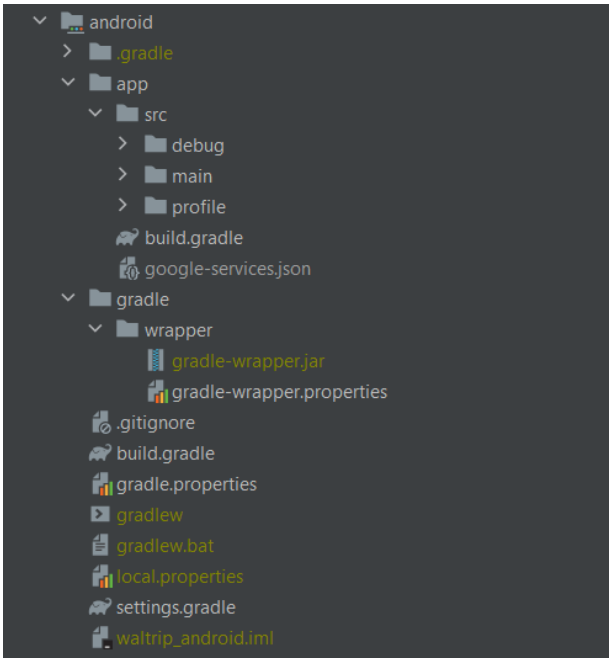


Figura 4.18: Firebase: Estructura directorio android.

Firebase para iOS

A continuación, al igual que hemos hecho para los sistemas Android, añadiremos la versión en iOS de nuestra aplicación al proyecto Flutter para poder trabajar con el servicio también a través de este sistema.

Para ello, como podemos ver en la figura 4.19 primero definimos el nombre del paquete de iOS, en nuestro caso *com.waltrip.app*, y le añadimos el sobrenombre *Waltrip* para su representación en Firebase.

Figura 4.19: Firebase: Registrar aplicación iOS.

Tras esto, como se muestra en la figura 4.20 y de la misma forma que para la versión en Android, descargamos el archivo de configuración y lo incluimos en nuestro proyecto.

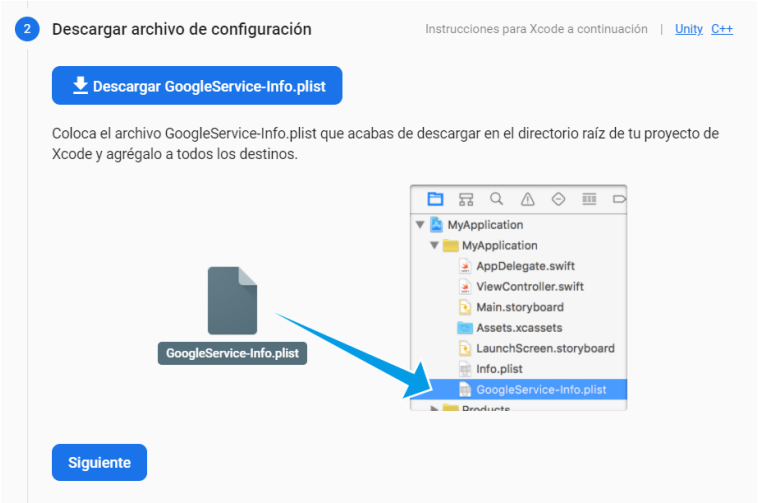


Figura 4.20: Firebase: Archivo de configuración iOS.

En el archivo *GoogleService-Info.plist* encontramos el siguiente código de configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
    PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CLIENT_ID</key>
  <string>260715873336-s2llbtc11qo43igol1jgm513f4gkvt1q.apps.googleusercontent.com</
      string>
  <key>REVERSED_CLIENT_ID</key>
  <string>com.googleusercontent.apps.260715873336-s2llbtc11qo43igol1jgm513f4gkvt1q</
      string>
  <key>API_KEY</key>
  <string>AIzaSyBMoGzC3IrR1tQi4g8obe5uW71FAWbHJqo</string>
  <key>GCM_SENDER_ID</key>
  <string>260715873336</string>
  <key>PLIST_VERSION</key>
  <string>1</string>
  <key>BUNDLE_ID</key>
  <string>com.waltrip.app</string>
  <key>PROJECT_ID</key>
  <string>waltrip-app</string>
  <key>STORAGE_BUCKET</key>
  <string>waltrip-app.appspot.com</string>
  <key>IS_ADS_ENABLED</key>
  <false></false>
  <key>IS_ANALYTICS_ENABLED</key>
  <false></false>
  <key>IS_APPINVITE_ENABLED</key>
  <true></true>
  <key>IS_GCM_ENABLED</key>
  <true></true>
  <key>IS_SIGNIN_ENABLED</key>
  <true></true>
```

```
  <key>GOOGLE_APP_ID</key>
  <string>1:260715873336:ios:75c7530586ed529f31d2cd</string>
</dict>
</plist>
```

Firestore Database: Cloud Firestore

Para la creación de la base de datos de nuestra aplicación vamos a utilizar el servicio de Firebase Cloud Firestore[8], una base de datos NoSQL que mantiene los datos de la plataforma sincronizados para todos los clientes de la aplicación en tiempo real. Se trata de una base de datos flexible y escalable que nos permitirá una fácil gestión de los datos y la realización de un uso intensivo de los mismos, de esta forma, se garantiza la continuidad en el futuro de la utilización de este servicio.

Para poder empezar a utilizar este servicio en nuestro proyecto, añadimos la siguiente dependencia a nuestro proyecto dentro del archivo *pubspec.yaml* definido para ello:

```
dependencies:
  cloud_firestore:
```

De esta forma, ya podemos empezar a utilizar las librerías de Cloud Firestore disponibles en el paquete que nos permitirá interactuar con la base de datos de nuestra aplicación.

4.4.4. Almacenamiento de datos en local

Para poder almacenar datos de forma local en el dispositivo y así disponer de estos durante nuestras consultas a la base de datos para los diferentes servicios, hemos añadido a nuestro proyecto el paquete Shared Preference[14] de Flutter, que nos permite guardar información en el dispositivo y, de esta forma, evitar repetir lecturas de la base de datos.

Para ello, añadimos la siguiente dependencia al archivo *pubspec.yaml* definido para ello:

```
dependencies:
  shared_preferences: ^2.0.6
```

De esta forma, ya podemos empezar a utilizar las librerías disponibles en el paquete y almacenar variables de nuestra aplicación en local.

4.4.5. Integración de Mapbox

Para la integración y uso de mapas en nuestra proyecto, vamos a utilizar el servicios Mapbox[22], un proveedor de mapas en linea que nos permitirá la edición de mapas, y su visualización y navegación a través de nuestra aplicación.

Para ello, utilizaremos el paquete Flutter Mapbox GL[30] que añadiremos a nuestro proyecto mediante la siguiente dependencia, que debe introducirse en el archivo *pubspec.yaml* dispuesto para ello:

```
dependencies:
  mapbox_gl: ^0.12.0
```

De esta forma, nos quedaría el siguiente código en *pubspec.yaml*:

```
...

dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2

  # The following adds Mapbox GL to the application.
  mapbox_gl: ^0.12.0

...
```

Tras esto ya tenemos las dependencias necesarias para la interacción con los servicios de Mapbox a través de nuestro proyecto en Flutter.

Configuración para la localización

Para que nuestros mapas puedan acceder a la localización del dispositivo y así poder interactuar con la posición del usuario, debemos habilitar esta funcionalidad para cada uno de los sistemas en los que se vaya a utilizar la aplicación. En nuestro caso, debemos realizar esta configuración para los dispositivos con sistema Android e iOS.

- **Configurar permisos de localización en Android.**

  En los sistemas Android encontramos dos permisos diferentes de ubicación:

  - `ACCESS_COARSE_LOCATION`: Permite la utilización del Wi-Fi y/o datos móviles para determinar la ubicación del dispositivo.
  - `ACCESS_FINE_LOCATION`: Permite determinar la localización del dispositivo de la manera más precisa posible mediante los proveedores de ubicación disponibles, además del GPS, Wi-Fi y datos móviles.

En nuestro caso, ante la necesidad de contar con una ubicación más precisa, decidimos utilizar los permisos otorgados por la configuración de la opción `ACCESS_FINE_LOCATION`. Para ello, incluiremos la siguiente linea de código en el archivo *AndroidManifest.xml* de la carpeta *android*:

```
<manifest ...
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

De esta forma, así quedaría reflejado en el archivo *AndroidManifest.xml* del directorio Android de nuestro proyecto:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.waltrip.app.waltrip">
   <application
        ...
    </application>

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

</manifest>
```

- **Configurar permisos de localización en iOS**

  Para permitir el acceso a la ubicación del dispositivo en los sistemas iOS, debemos incluir las siguientes lineas de código en el archivo *Info.plist* de la carpeta *ios*:

```
xml ...
    <key>NSLocationWhenInUseUsageDescription</key>
    <string>[Your explanation here]</string>
```

  Definimos la explicación para el uso de este permiso en el apartado *[Your explanation here]* y lo incluimos en el archivo *Info.plist*, quedando reflejado de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
    PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>NSLocationWhenInUseUsageDescription</key>
  <string>Shows your location on the map and helps improve the map</string>
  ...
</dict>
</plist>
```

CAPÍTULO 5
Desarrollo del primer producto mínimo viable.

En el siguiente capitulo abarcaremos el desarrollo de la primera versión o primer producto mínimo viable de Waltrip. Se revisará el alcance de esta versión con los requerimientos a implantar, las integraciones realizadas y proceso de desarrollo llevada a cabo.

5.1 Introducción

Para el desarrollo del primer MVP, consideraremos el estudio previo realizado en el backlog para identificar las funcionalidades iniciales a implementar, teniendo en cuenta su prioridad y viabilidad. Siguiendo estos pasos, a continuación presentaremos el mockup conceptual creado como una vista previa de las funciones que se desarrollarán.

Después de esto, procederemos a realizar una evaluación de esta primera versión basada en la opinión obtenida de diferentes usuarios de prueba, a través de la cumplimentación de un formulario. Esta retroalimentación será de vital importancia para identificar áreas de mejora y determinar si las funcionalidades implementadas cumplen con las necesidades y expectativas de los usuarios.

Siguiendo el enfoque del MVP, este proceso nos permitirá validar y ajustar la dirección del desarrollo del producto, centrándonos en las funcionalidades clave y asegurando que la versión final cumpla con los requisitos establecidos. Con la información obtenida de la evaluación del primer MVP, estaremos preparados para seguir iterando y mejorando el producto en versiones futuras.

5.2 Requerimientos y pruebas de validación

Para la elaboración del mockup nos hemos ajustado, principalmente, y como ya hemos mencionado anteriormente, a las prioridades identificadas para cada una de las funcionalidades estudiadas en el backlog. De esta forma, para este primer MVP, decidimos implementar las siguientes funcionalidades:

5.2.1. Pantalla principal:

Primera pantalla que verá el usuario al acceder a la aplicación. Esta debería consistir en una vista previa del contenido disponible en la aplicación y del acceso a los distintos apartados y funciones ofrecidas.

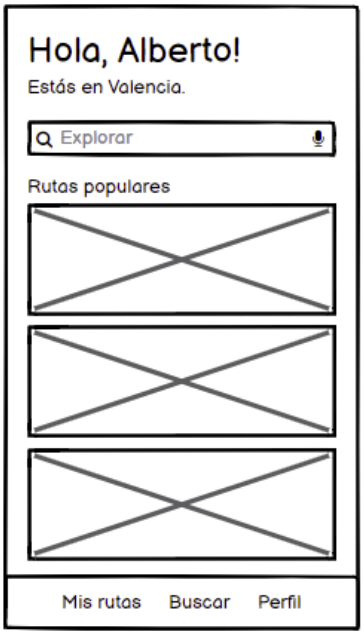


Figura 5.1: Mockup primer MVP: Pantalla principal.

Pruebas de validación (UAT):

Tabla 5.1: UAT pantalla principal.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Visualización de contenido	Acceder a la aplicación	Se carga parte del contenido proporcionado al usuario.
Acceso a las funcionalidades	Acceder a la aplicación	Se muestran y se permite acceder a las diferente funcionalidades disponibles.

5.2.2. Perfil de usuario:

A pesar de no ser una de las funcionalidades ”esenciales” para el usuario, esta es necesaria para la manejar toda la lógica de negocio de la aplicación, por lo que será el punto de partida para la implementación del resto de funciones.

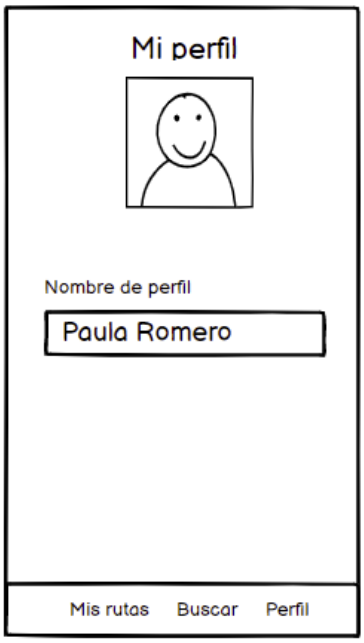


Figura 5.2: Mockup primer MVP: Pantalla perfil.

Pruebas de validación (UAT):

Tabla 5.2: UAT perfil de usuario.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Selección de perfil	Especificar perfil a utilizar	Se carga la información contenida por el perfil de usuario especificado
Información asociada al perfil	Acceder al perfil	Se muestra la información del usuario junto a las rutas guardadas y creadas

5.2.3. Visualizar rutas y su contenido:

Otorga a la aplicación el principal aspecto de usabilidad para el usuario, permitiendo la interacción directa con el contenido ofrecido y la navegación por las principales funcionalidades. De esta forma, el usuario podrá acercarse a una primera experiencia de lo que la aplicación busca ofrecer.

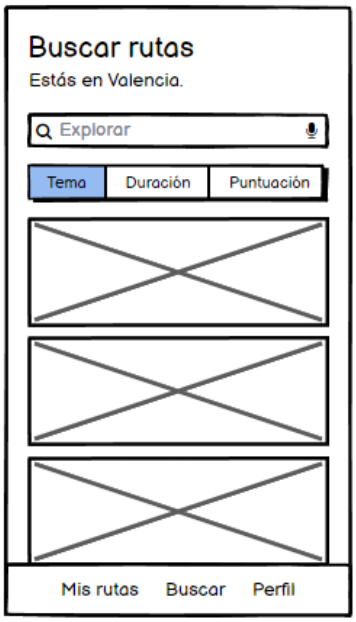


Figura 5.3: Mockup primer MVP: Pantalla buscar.

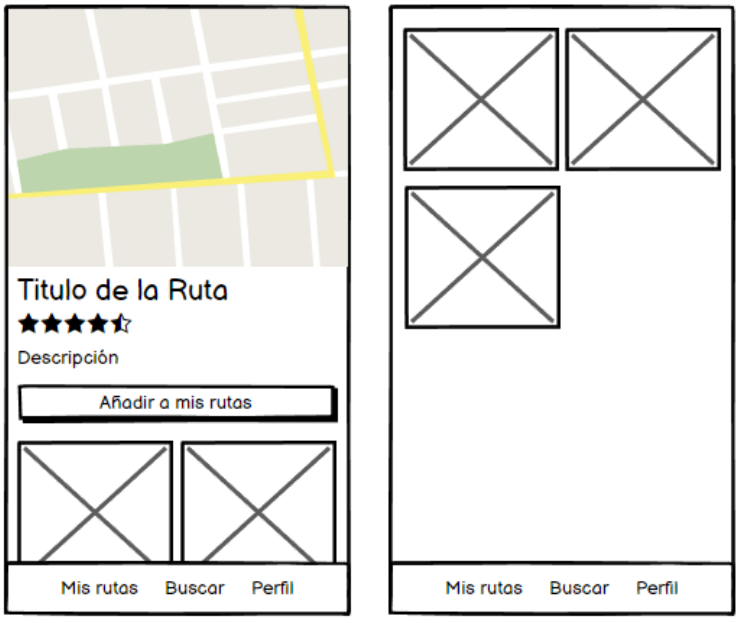


Figura 5.4: Mockup primer MVP: Pantalla información rutas.

Pruebas de validación (UAT):

Tabla 5.3: UAT visualizar rutas y su contenido.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Rutas disponibles	Acceder a la aplicación	Se muestran las rutas disponibles al usuario
Detalles de la ruta	Acceder a una ruta	Se muestran los detalles de la ruta

5.2.4. Guardar rutas:

Su implementación conforma la utilidad principal de la aplicación, siendo esta una de las características ofrecidas más diferenciadoras y, por lo tanto, esenciales para esta primera versión el producto.

El mockup para esta funcionalidad puede observarse en la figura 6.3 anterior.

Pruebas de validación (UAT):

Tabla 5.4: UAT guardar rutas.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Guardar ruta	Acceder a una ruta y guardarla	La ruta se asocia al perfil de usuario actual y queda reflejado

5.2.5. Crear rutas:

Necesario para la generación de contenido en la aplicación. Con ello, ya no solo le permitimos al usuario consumir contenido, sino que también podrá ser creador de este, generando otra visión del producto que atraerá a diferentes perfiles de usuarios y que conformará una de las características principales del producto.

Figura 5.5: Mockup primer MVP: Pantalla nueva ruta.

Pruebas de validación (UAT):

Tabla 5.5: UAT crear rutas.

Funcionalidad involucrada	**Caso de prueba**	**Resultado esperado**
Crear ruta	Crear una nueva ruta	La ruta es creada, mostrada en la aplicación y asociada al perfil de usuario actual

Como se podría intuir, la decisión de implementar estas funcionalidades se basa en aportar al usuario una visión, lo más amplia posible, de lo que va a ser el producto que buscamos ofrecer. Ofreciéndoles, en esta primera versión, las principales funcionalidades que la caracterizan y que permitirán obtener una opinión más amplia por parte de los diferentes usuarios. De esta forma, podremos analizar mejor las necesidades a cubrir para la siguiente versión de la aplicación.

5.3 Integración del Backend

Antes de comenzar con el desarrollo de la primera versión de la aplicación y de las diferentes funcionalidades de las que dispondrá, debemos implementar la integración de esta con los diferentes servicios que darán soporte al backend. En base a la arquitectura escogida, a continuación, detallaremos los diferentes aspectos que permitirán la conexión entre las diferentes capas que conformarán la aplicación.

5.3.1. Repositorio

Para poder empezar con la integración, primero debemos crear la capa de repositorios o *repositories*, encargada de la abstracción de la estructura que conforma el backend, e implementar las conexiones necesarias. Para esta primera version, y atendiendo a las tecnologías seleccionadas, conectaremos los siguientes servicios:

- **Firebase Firestore:** Base de datos NoSQL para el tratamiento de la información. Constituirá el sistema de almacenamiento principal de la aplicación, gestionando la información y sus diferentes relaciones y dependencias.
- **Firebase Storage:** Base de datos para el tratamiento de archivos. Útil para el almacenamiento de contenido como fotos o vídeos.
- **Flutter SharedPreferences[14]:** Sistema de gestión de información persistente. Utilizado para almacenar información en los dispositivos de los usuarios.

De esta forma, y tras realizar las configuraciones previas para la integración de Firebase reflejadas en la sección *Integración con Firebase* 4.4.3, creamos el directorio *repositories*, donde incluiremos los archivos *.dart* con las instancias necesarias para la interacción con los diferentes servicios:

RouteRepository.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';

class RoutesRepository {

  CollectionReference getRoutesReference(){
    return FirebaseFirestore.instance.collection('routes');
  }

  Reference getStorageReference(){
    return FirebaseStorage.instance.refFromURL('gs://waltrip-app.appspot.com').child('
        routes');
  }

}
```

ProfileRepository.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class ProfilesRepository{

  CollectionReference getReference() {
    return FirebaseFirestore.instance.collection('profiles');
  }

}
```

ProfileManager.dart

```dart
import 'package:shared_preferences/shared_preferences.dart';

class ProfileManager{

  void setProfile(String profileId) async {
    final prefs = await SharedPreferences.getInstance();
    prefs.setString('profileId', profileId);
  }

  Future loadProfile() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('profileId');
  }

}
```

Como se puede observar, hemos separado el repositorio en instancias para la gestión de rutas y para el perfil, de esta forma, solo llamaremos a los repositorios que sean necesarios en las siguientes capas.

A diferencia de las base de datos SQL, donde la información esta distribuida en tablas, en *Firestore*, esta distribución se realiza por colecciones, donde, en vez de filas, se incluyen documentos con cada uno de los campos de datos que conforman la entidad. De esta forma, en cada una de las instancias a *Firestore*, recuperamos la colección con la que queremos trabajar, permitiéndonos la lectura, escritura, actualización y eliminación de los documentos que la conforman. En nuestro caso, obtenemos la instancia para la gestión de la información referente a las rutas (*RouteRepository.dart*, función *getRoutesReference()*, linea 6) y los datos de los perfiles (*ProfileRepository.dart*, función *getReference()*, linea 6).

Por otro lado, la gestión de los perfiles se realiza a través del *ProfileManager.dart*, donde se define la función *setProfile()* para indicar el perfil de usuario a utilizar y *loadProfile()* para recuperar el perfil utilizado actualmente. Este utiliza el servicio *SharedPreferences* que permite almacenar el identificativo del perfil actual (*profileId*) en memoria y así saber en todo momento cual esta siendo utilizado por el usuario.

5.3.2. Entidades

Para la gestión de los objetos de negocio, con los que deberá trabajar la aplicación, y su interacción con los diferentes servicios, se han creado las siguientes entidades o *entities* y conversores o *converters* que permitirán la abstracción de los objetos manejados por el frontend del backend, facilitando su manejo futuro en la capa para los *controladores*: *RouteEntity.dart*

```dart
class RouteEntity {

  final String id;
  final String title;
  final String description;
  final String creator;
  double rating;
  double price;
  bool isFavourite;
```

```
  bool isPopular;
  String imagePath;

  RouteEntity({
    required this.id,
    required this.title,
    required this.description,
    required this.creator,
    this.rating = 0.0,
    this.price = 0.0,
    this.isFavourite = false,
    this.isPopular = false,
    this.imagePath = ""
  });

}
```

RouteConverter.dart

```
import 'package:waltrip/entities/RouteEntity.dart';

class RouteConverter {

  RouteEntity mapToRoute(Map map, [String? id]){
    RouteEntity route = new RouteEntity(
      id: id != null ? id : map['id'],
      title: map['title'],
      description: map['description'],
      creator: map['creator'],
      rating: map['rating'],
      price: map['price'],
      isFavourite: map['isFavourite'],
      isPopular: map['isPopular'],
      imagePath: map['imagePath']
    );
    return route;
  }

  //addEmptyId es false por defecto.
  Map<String, dynamic> routeToMap(RouteEntity route, [bool? addEmptyId]){
    Map<String, dynamic> routeMap = new Map();
    if(route.id.isNotEmpty | (addEmptyId != null ? addEmptyId : false)) routeMap['id']
        = route.id;
    routeMap.addAll( {
      'title': route.title,
      'description': route.description,
      'creator': route.creator,
      'rating' : route.rating,
      'price' : route.price,
      'isFavourite' : route.isFavourite,
      'isPopular' : route.isPopular,
      'imagePath' : route.imagePath
    });
    return routeMap;
  }

}
```

ProfileEntity.dart

```dart
class ProfileEntity {

  final String id;
  final String name;
  List<String> createdRoutes;
  List<String> savedRoutes;

  ProfileEntity({
    required this.id,
    required this.name,
    this.createdRoutes = const [],
    this.savedRoutes = const []
  });

}
```

ProfileConverter.dart

```dart
import 'package:waltrip/entities/ProfileEntity.dart';

class ProfileConverter {

  ProfileEntity mapToProfile(Map map, [String? id]){
    ProfileEntity profile = new ProfileEntity(
      id: id != null ? id : map['id'],
      name: map['name'],
      createdRoutes: new List.from(map['createdRoutes']),
      savedRoutes: new List.from(map['savedRoutes'])
    );
    return profile;
  }

  //addEmptyId es false por defecto.
  Map<String, dynamic> profileToMap(ProfileEntity profile, [bool? addEmptyId]){
    Map<String, dynamic> profileMap = new Map();
    if(profile.id.isNotEmpty | (addEmptyId != null ? addEmptyId : false)) profileMap['
        id'] = profile.id;
    profileMap.addAll( {
      'name': profile.name,
      'createdRoutes': profile.createdRoutes,
      'savedRoutes' : profile.savedRoutes
    });
    return profileMap;
  }

}
```

Como podemos observar, respectivamente, contamos con las entidades y conversores para el manejo de las rutas y la información de los perfiles de usuario. Cada una de las entidades conforma un objeto de datos con sus atributos y especificaciones. Por otro lado, los conversores transforman las entidades en *Maps*[17], estructuras de datos comprendidas por los servicios de *Firebase*, y viceversa, para su gestión en las diferentes capas.

5.3.3. Servicios

Con el fin de abstraer la lógica de negocio del de la aplicación, definiremos un conjunto de servicios que conformarán la capa de *services* y contendrán los métodos básicos para la interacción con los elementos del backend desde la capa de controladores o *controllers*.

En nuestro caso, definiremos dos conjuntos de servicios: *RoutesService.dart*

```
import ...

class RoutesService {

  final RoutesRepository _repository = RoutesRepository();

  final RouteConverter _converter = RouteConverter();

  Future addRoute(RouteEntity route, Uint8List image) async{
    return await _repository.getRoutesReference().add(_converter.routeToMap(route)).
        then((documentReference) {
      String id = documentReference.id;
      setRouteImage(id, route.imagePath, image);
      return id;
    });
  }

  Future getRoutes() async{
    List<RouteEntity> routes = [];
    try {
      await _repository.getRoutesReference().get().then((querySnapshot) {
        querySnapshot.docs.forEach((doc) {
          routes.add(_converter.mapToRoute(doc.data() as Map, doc.id));
        });
      });
      return routes;
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  Future getRoute(String id) async{
    try {
      return await _repository.getRoutesReference().doc(id).get().then((
          documentSnapshot) {
        if(documentSnapshot.exists) return _converter.mapToRoute(documentSnapshot.data
            () as Map, id);
        else return null;
      });
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  Future deleteRoute(String id) async{
    try {
      await _repository.getRoutesReference().doc(id).delete();
```

```
47      } catch (e) {
48        print(e.toString());
49      }
50    }
51
52    updateRoute(RouteEntity route) async{
53      Map<String,dynamic> routeMap =  _converter.routeToMap(route);
54      try {
55        await _repository.getRoutesReference().doc(routeMap['id']).update(routeMap);
56      } catch (e) {
57        print(e.toString());
58      }
59    }
60
61    setRouteImage(String id, String imagePath, Uint8List image) async {
62      Reference ref = _repository.getStorageReference().child(id + imagePath);
63      try {
64        await ref.putData(image);
65      } catch (e) {
66        print(e.toString());
67      }
68    }
69
70    Future getRouteImage(String id, String imagePath) async{
71      Reference ref = _repository.getStorageReference().child(id + imagePath);
72      try {
73        const oneMegabyte = 640 * 640;
74        Uint8List? imageUint = await ref.getData(oneMegabyte);
75        if(imageUint != null) return Image.memory(imageUint);
76        else return null;
77      } catch (e) {
78        print(e.toString());
79        return null;
80      }
81    }
82
83  }
```

Donde se implementan las siguientes funciones para la gestión de la información de las rutas:

- **addRoute**, almacenar una nueva ruta.
- **getRoutes**, obtener todas las rutas.
- **getRoute**, obtener una ruta en especifico.
- **deleteRoute**, eliminar una ruta.
- **updateRoute**, actualizar una ruta.
- **setRouteImage**, definir imagen de portada para la presentación de la ruta.
- **getRouteImage**, obtener la imagen de portada de la ruta.

ProfileService.dart

```
import ...

class ProfilesService {

  final ProfileManager _profileManager = ProfileManager();

  final ProfilesRepository _repository = ProfilesRepository();

  final RoutesService _routesService = RoutesService();

  final ProfileConverter _converter = ProfileConverter();

  static const SAVED_ROUTES_PATH = "savedRoutes";
  static const CREATED_ROUTES_PATH = "createdRoutes";

  Future addProfile(ProfileEntity profile) async{
    return await _repository.getReference().add(_converter.profileToMap(profile));
  }

  Future getProfile(String id) async{
    try {
      return await _repository.getReference().doc(id).get().then((documentSnapshot) {
        if(documentSnapshot.exists) return _converter.mapToProfile(documentSnapshot.
            data() as Map, id);
        else return null;
      });
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  Future getProfiles() async{
    List<ProfileEntity> list = [];
    try {
      return await _repository.getReference().get().then((querySnapshot) {
        querySnapshot.docs.forEach((doc) {
          list.add(_converter.mapToProfile(doc.data() as Map, doc.id));
        });
        return list;
      });
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  addSavedRoute(String routeId) async{
    String profile = await _profileManager.loadProfile();
    addRoute(profile, routeId, SAVED_ROUTES_PATH);
  }

  addCreatedRoute(String routeId) async{
    String profile = await _profileManager.loadProfile();
    addRoute(profile, routeId, CREATED_ROUTES_PATH);
  }

  addRoute(String profile, String routeId, String path) async{
```

```
58      await _repository.getReference().doc(profile).update({path : FieldValue.arrayUnion
            ([routeId])});
59    }
60
61    deleteSavedRoute(String routeId) async{
62      String profile = await _profileManager.loadProfile();
63      deleteRoute(profile, routeId, SAVED_ROUTES_PATH);
64    }
65
66    deleteCreatedRoute(String routeId) async{
67      String profile = await _profileManager.loadProfile();
68      deleteRoute(profile, routeId, CREATED_ROUTES_PATH);
69    }
70
71    deleteRoute(String profile, String routeId, String path) async{
72      await _repository.getReference().doc(profile).update({path : FieldValue.
            arrayRemove([routeId])});
73    }
74
75    Future getSavedRoutes() async{
76      String profile = await _profileManager.loadProfile();
77      return await getRoutes(profile, SAVED_ROUTES_PATH);
78    }
79
80    Future getCreatedRoutes() async{
81      String profile = await _profileManager.loadProfile();
82      return await getRoutes(profile, CREATED_ROUTES_PATH);
83    }
84
85    Future getRoutes(String profile, String path) async{
86      List<RouteEntity> routes = [];
87      return await _repository.getReference().doc(profile).get().then((DocumentSnapshot
            documentSnapshot) async {
88        if(documentSnapshot.exists) {
89          List<dynamic> ids = documentSnapshot.get(path);
90          if(ids.length > 0){
91            routes =  await getRouteEntityByIdList(ids, 0, routes);
92          }
93        }
94        return routes;
95      });
96    }
97
98    Future getSavedRoutesId() async{
99      String profile = await _profileManager.loadProfile();
100     return getRoutesId(profile, SAVED_ROUTES_PATH);
101   }
102
103   Future getCurentProfile() async{
104     return await _profileManager.loadProfile();
105   }
106
107   Future getRoutesId(String profile, String path) async{
108     List<String> ids = [];
109     return await _repository.getReference().doc(profile).get().then((DocumentSnapshot
            documentSnapshot) async {
110       if(documentSnapshot.exists) {
111         return documentSnapshot.get(path);
112       } else return ids;
```

```
113     });
114   }
115
116   Future getRouteEntityByIdList(List<dynamic> idList, index, List<RouteEntity> result)
          async{
117     RouteEntity route = await _routesService.getRoute(idList[index]) as RouteEntity;
118     if(route != null) result.add(route);
119     if(index == idList.length - 1) {
120       return result;
121     } else {
122       return getRouteEntityByIdList(idList, ++index, result);
123     }
124   }
125
126 }
```

Donde se implementan las siguientes funciones para la gestión de la información de los perfiles:

- **addProfile**, crear un nuevo perfil.
- **getProfile**, obtener la información de un perfil.
- **getProfiles**, obtener todos los perfiles.
- **addSavedRoute**, vincula al perfil una ruta guardada.
- **addCreatedRoute**, vincula al perfil una ruta creada.
- **addRoute**, función base para vincular rutas al perfil de forma que estas puedan ser tratadas a nivel de usuario.
- **updateName**, actualizar el nombre de un perfil de usuario.
- **deleteSavedRoute**, desvincula al perfil una ruta guardada.
- **deleteCreatedRoute**, desvincula al perfil una ruta creada.
- **deleteRoute**, función base para desvincular una ruta del perfil.
- **getSavedRoutes**, obtener las rutas guardadas del perfil.
- **getSavedRoutesId**, obtener las identificaciones de las rutas guardadas por el perfil.
- **getCreatedRoutes**, obtener las rutas creadas del perfil.
- **getRoutes**, obtener rutas vinculadas al perfil.
- **getCurrentProfile**, obtener la identificación única del usuario.
- **getRoutesId**, obtener la identificación única de rutas vinculadas al usuario.
- **getRouteEntityByIdList**, obtener un conjunto de rutas especificando sus identificativos únicos.

De esta forma, los métodos implementados en la capa de servicios nos permitirá olvidarnos de la gestión de la información en relación a su tratamiento con la base de datos y podremos proceder a centrarnos en la implementación de las funcionalidades.

Más adelante, veremos el uso que se le da a cada uno de estos métodos en la capa de controladores, en las funciones implementadas en la aplicación de cara al usuario final.

5.3.4. Objetos de controlador

Para facilitar el paso de información al frontend desde la capa de controladores, y con el fin de unificar la información en objetos únicos que faciliten su gestión, adicionalmente, hemos diseñado la clase *RouteObject.dart*, un objeto de datos que unifica la información distribuida entre los diferentes servicios para reducir las llamadas al controlador:

RouteObject.dart

```
import ...

class RouteObject {

  final RouteEntity routeInfo;
  final Image routeImage;
  final ProfileEntity routeCreator;

  RouteObject({
    required this.routeInfo,
    required this.routeImage,
    required this.routeCreator,
  });

}
```

De esta forma, cuando se solicite información sobre una ruta, la capa superior podrá disponer, además, de información sobre su autor y otro contenido multimedia enlazada a esta.

5.4 Funciones implementadas.

Tras finalizar el desarrollo de esta primera versión, a continuación, se muestran las funciones finalmente implementadas. Desde la visión del Backend, estas han sido integradas en la capa de controladores y conectadas a la interfaz mediante su implementación en el frontend.

5.4.1. Rutas disponibles.

Muestra las rutas disponibles en la pantalla principal tras la interacción necesaria con la base de datos.

Implementación

RoutesController.dart

```
import ...

class RoutesController {

  RoutesService _service = new RoutesService();

  ProfilesService _profilesService = new ProfilesService();

  Future getAllRoutes() async{
    List<RouteObject> routes = [];
    List<RouteEntity> routeEntityList = await _service.getRoutes();
    if(routeEntityList.isEmpty) {
      return routes;
    } else {
      return getRoutesByEntityList(routeEntityList, 0, routes);
    }
  }

  Future getRouteImage(String id, String storageImagePath) async{
    return await _service.getRouteImage(id, storageImagePath);
  }

  Future getRoutesByEntityList(List<RouteEntity> entityList, index, List<RouteObject>
      result) async{
    Image routeImage = await _service.getRouteImage(entityList[index].id, entityList[
        index].imagePath);
    ProfileEntity routeProfile = await _profilesService.getProfile(entityList[index].
        creator);

    List<dynamic> savedRoutes = await _profilesService.getSavedRoutesId();
    entityList[index].isFavourite = savedRoutes.contains(entityList[index].id);

    RouteObject route = new RouteObject(
        routeInfo: entityList[index],
        routeImage: routeImage,
        routeCreator: routeProfile);

    result.add(route);
    if(index == entityList.length - 1) {
      return result;
    } else {
      return getRoutesByEntityList(entityList, ++index, result);
    }
  }

  ...

}
```

La función *getAllRoutes()* permite obtener todas las rutas en forma de *RouteObjects*. Para ello, primero obtiene las rutas en forma de entidades, *RouteEntity*, conteniéndolas en una lista, para después llamar al método *getRoutesByEntityList()* y devolver una lista de *RouteObjects* con información sobre la ruta, contenido multimedia asociado y datos sobre el creador.

Interfaz

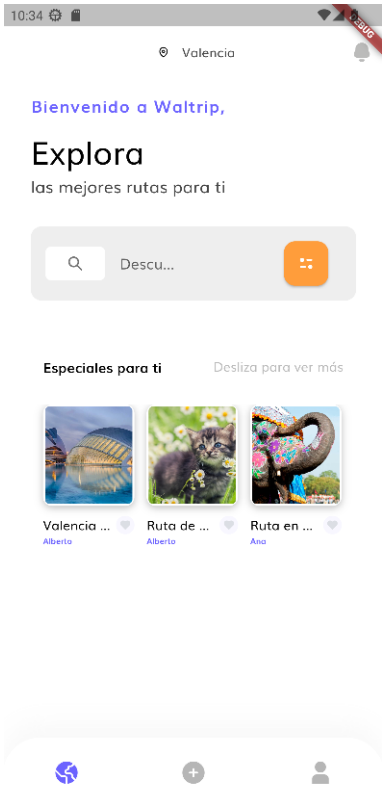


Figura 5.6: Interfaz primer MVP: Inicio.

5.4.2. Gestión del perfil.

Permite definir el perfil utilizado por el usuario. Por el momento, el usuario solo tiene que definir el nombre de perfil, necesario para la lógica de negocio interna, que desea utilizar para cargar e interactuar con toda la información asociada a este.
Una vez definido, el usuario podrá ver la información referente a su perfil, como su nombre, rutas creadas y guardadas, además de poder empezar a interactuar con el resto de funcionalidades de la aplicación.

Implementación

ProfileController.dart

```
import ...

class ProfilesController {

  ProfilesService _profilesService = new ProfilesService();
  RoutesService _routesService = new RoutesService();

  Future createProfile(String profileName) async {
    ProfileEntity profile = new ProfileEntity(
```

```
10          id: '',
11          name: profileName
12      );
13      return await _profilesService.addProfile(profile).then((documentReference) {
14        String id = documentReference.id;
15        print('Profile ' + profileName + ' with id ' + id + ' created');
16        return id;
17      });
18    }
19
20    Future getProfile() async{
21      return _profilesService.getCurentProfile();
22    }
23
24    Future getProfileByName(String profileName) async{
25      return await _profilesService.getProfiles().then((list) {
26        for(var profile in list) {
27          if (profile.name == profileName) return profile;
28        }
29        return null;
30      });
31    }
32
33    Future getProfileById(String id) async{
34      return await _profilesService.getProfile(id);
35    }
36
37    Future getSavedRoutes() async {
38      List<RouteObject> routes = [];
39      List<RouteEntity> routeEntityList = await _profilesService.getSavedRoutes();
40      if(routeEntityList.isEmpty) {
41        return routes;
42      } else {
43        return getRoutesByEntityList(routeEntityList, 0, routes);
44      }
45    }
46
47    Future getCreatedRoutes() async {
48      List<RouteObject> routes = [];
49      List<RouteEntity> routeEntityList = await _profilesService.getCreatedRoutes();
50      if(routeEntityList.isEmpty) {
51        return routes;
52      } else {
53        return getRoutesByEntityList(routeEntityList, 0, routes);
54      }
55    }
56
57    Future loadProfile(String profileName) async{
58      ProfileEntity profile = await getProfileByName(profileName);
59      if(profile != null){
60        return profile;
61      } else {
62        return await createProfile(profileName).then((id) async {
63          return await getProfileById(id);
64        });
65      }
66    }
67
```

```
68    Future getRoutesByEntityList(List<RouteEntity> entityList, index, List<RouteObject>
          result) async{
69      Image routeImage = await _routesService.getRouteImage(entityList[index].id,
            entityList[index].imagePath);
70      ProfileEntity routeProfile = await _profilesService.getProfile(entityList[index].
            creator);
71
72      List<dynamic> savedRoutes = await _profilesService.getSavedRoutesId();
73      entityList[index].isFavourite = savedRoutes.contains(entityList[index].id);
74
75      RouteObject route = new RouteObject(
76          routeInfo: entityList[index],
77          routeImage: routeImage,
78          routeCreator: routeProfile);
79
80      result.add(route);
81      if(index == entityList.length - 1) {
82        return result;
83      } else {
84        return getRoutesByEntityList(entityList, ++index, result);
85      }
86    }
87
88  }
```

En este caso, contamos con diferentes funciones para la gestión del perfil:

- **createProfile**, crea un nuevo usuario pasando un *ProfileEntity* al servicio y devuelve la identificación única del nuevo perfil.
- **getProfile**, obtiene el nombre del perfil actual.
- **getProfileByName**, obtiene la información del perfil por nombre.
- **getProfileByID**, obtiene la información del perfil por su identificador.
- **getSavedRoutes**, obtiene las rutas guardadas del usuario actual devolviendo una lista de *RouteObjects*.
- **getCreatedRoutes**, obtiene las rutas creadas por el usuario actual devolviendo una lista de *RouteObjects*.
- **loadProfile**, obtiene la información del perfil actual si existe, en su defecto lo crea y devuelve su información.
- **getRoutesByEntityList**, convierte una lista de *RouteEntity* en una lista de *RouteObject* con información detallada sobre la ruta y su creador.

Interfaz

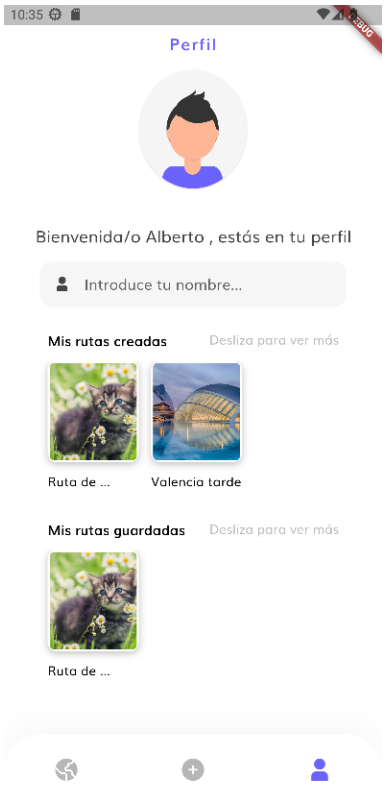


Figura 5.7: Interfaz primer MVP: Perfil.

5.4.3. Crear ruta.

Permite la inclusión de una nueva ruta en la base de datos. Dicha ruta es asociada al perfil del usuario que la ha creado.

Para poder crear una ruta es necesario definir:

- El Título de la nueva ruta.
- La Descripción de la nueva ruta.
- Una portada.

Implementación

RoutesController.dart

```
1 import ...
2
3 class RoutesController {
4
5   RoutesService _service = new RoutesService();
6
```

```
7   ProfilesService _profilesService = new ProfilesService();
8
9   Future createRoute(String title, String description, XFile image) async {
10    String profile = await _profilesService.getSavedRoutesId();
11
12    RouteEntity route = new RouteEntity(
13        id: '',
14        title: title,
15        description: description,
16        creator: profile,
17        imagePath: "/" + image.name
18    );
19
20    return await _service.addRoute(route, await image.readAsBytes()).then((id) {
21        print('Route with id ' + id + ' created');
22        _profilesService.addCreatedRoute(id);
23        return id;
24    });
25  }
26
27  ...
28
29 }
```

La función *createRoute()* crea una nueva ruta pasándosela en forma de *RouteEntitiy* al servicio. Una vez creada, devuelve su identificador.

Interfaz

Interfaz:

Figura 5.8: Interfaz primer MVP: Creación de rutas.

5.4.4. Detalles de la ruta.

Muestra los detalles de una ruta tras la interacción necesaria con la base de datos. Entre los detalles que se ofrecen de cada ruta encontramos:

- Titulo.
- Creador.
- Descripción.
- Portada.

Implementación

RoutesController.dart

```
1 import ...
2
3 class RoutesController {
4
```

```
5   RoutesService _service = new RoutesService();
6
7   ProfilesService _profilesService = new ProfilesService();
8
9   Future getRouteById(String id) async{
10
11    RouteEntity routeInfo = await _service.getRoute(id);
12    Image routeImage = await _service.getRouteImage(id, routeInfo.imagePath);
13    ProfileEntity routeProfile = await _profilesService.getProfile(routeInfo.creator);
14
15    List<dynamic> savedRoutes = await _profilesService.getSavedRoutesId();
16    routeInfo.isFavourite = savedRoutes.contains(routeInfo.id);
17
18    RouteObject route = new RouteObject(
19        routeInfo: routeInfo,
20        routeImage: routeImage,
21        routeCreator: routeProfile);
22    return route;
23  }
24
25  ...
26
27 }
```

La función *getRouteById()* permite obtener toda la información referente a una ruta, mediante su identificador único, en forma de *RouteObjects*.

Interfaz

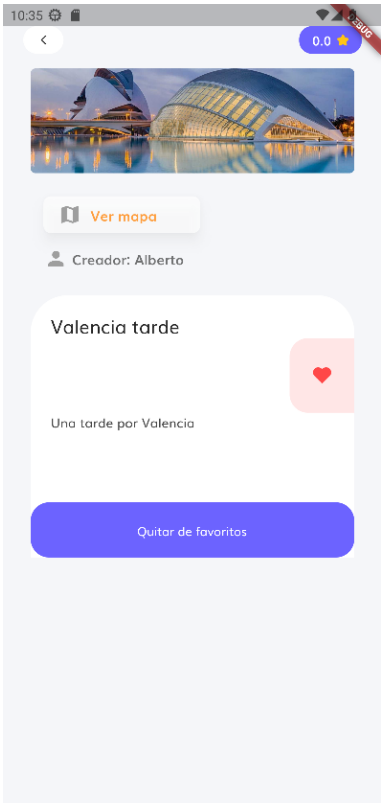


Figura 5.9: Interfaz primer MVP: Información rutas.

5.4.5. Guardar ruta.

Permite la asociación de rutas al perfil de usuario especificado, haciéndolas accesibles desde el perfil del usuario

Implementación

ProfileController.dart

```
import ...

class ProfilesController {

  ProfilesService _profilesService = new ProfilesService();

  saveRoute(String routeId) async {
    _profilesService.addSavedRoute(routeId);
  }

  removeRoute(String routeId) async {
    _profilesService.deleteSavedRoute(routeId);
  }

```

```
15    ...
16
17 }
```

La función *saveRoute()* y *removeRoute()* permite al usuario guardar o eliminar una ruta guardada respectivamente, indicando el id de la ruta.

Interfaz

La interfaz implementada para esta funcionalidad puede observarse en la figura 5.9 anterior.

5.5 Feedback de los usuarios

Para analizar en profundidad el resultado obtenido en la primera versión del producto desarrollado, se procede a la creación de un formulario con el cual recoger la opinión de los usuarios sobre el uso y manejo de la aplicación. Este formulario puede encontrarse en el Anexo B adjunto a esta memoria, bajo el titulo *Formulario primer MVP*, en el apartado *Questiones primer MVP*.

EL objetivo es obtener feedback sobre la idea de negocio, las funcionalidade implementadas y la interfaz mostrada al usuario. Las respuestas obtenidas serán utilizadas para la mejora de la aplicación y estudiar que funcionalidades deberán ser planteadas para el desarrollo del segundo MVP. De la misma forma, los resultados obtenidos puede consultarse en el apéndice adjunto a esta memoria, bajo el titulo *Formulario primer MVP*, en el apartado *Resultados primer MVP*.

En base a los resultados obtenidos, en general, los usuarios encuentran utilidad a la aplicación y estas satisfechos con las funciones implementadas para esta versión. Por otro lado, la interfaz implementada, junto a sus características como tamaños o colores, parece ser adecuada para el usuario y deja en claro las posibilidades de la aplicación.

A pesar de ello, en cuanto a los comentarios obtenidos, podemos resaltar la falta de implementación de utilidades y de algunas mejoras en la interfaz, ya que esta aún se encuentra un poco ”vacia” en la experiencia de uso de la aplicación, es decir, la interfaz construida contempla un mayor alcance de funcionalidades que las que realmente se han llegado a implementar.

Como conclusión, para el segundo MVP, deberíamos centrarnos en la implementación de nuevas utilidades que, aunque puedan parecer básicas, le den algo más de sentido a la interfaz para eliminar ese sentimiento de contenido ”vacio” como se ha contemplado en el análisis. Además, atendiendo a la aprobación de la interfaz recibida por los usuario hasta el momento, buscaremos movernos sobre las mismas lineas de diseño para las futuras versiones.

CAPÍTULO 6
Desarrollo del segundo producto mínimo viable.

En el siguiente capitulo abarcaremos el desarrollo de la segunda versión o segundo producto mínimo viable de Waltrip. De la misma forma que con la primera versión, se revisará el alcance de esta versión con los requerimientos a implantar, las integraciones realizadas y proceso de desarrollo llevada a cabo. Finalmente, se expondrán una revisión de la evolución de Waltrip y sus futuras implementaciones.

6.1 Introducción

Para el desarrollo del segundo MVP (Producto Mínimo Viable), continuaremos basándonos en el estudio previo realizado en el backlog, priorizando las funcionalidades según su viabilidad y relevancia. A continuación, presentaremos un nuevo mockup que servirá como concepto previo para las funciones que serán desarrolladas. Luego, compararemos este mockup con las características finales implementadas en esta versión.

Una vez completada la implementación, llevaremos a cabo una evaluación de este segundo MVP, recopilando opiniones de diversos usuarios de prueba a través de un nuevo formulario, que nos permitirá obtener feedback sobre la utilidad, usabilidad y satisfacción de los usuarios con respecto a las nuevas funcionalidades introducidas en esta versión.

Los comentarios y sugerencias recibidos durante la evaluación de este segundo MVP, se utilizaran para la aplicación de las mejoras pertinentes en futuras iteraciones del producto. El objetivo principal de este segundo MVP es validar y refinar las características clave que se han desarrollado hasta ahora, buscando alcanzar un equilibrio entre la funcionalidad y la experiencia del usuario, antes de continuar con el desarrollo y lanzamiento de versiones más completas y avanzadas.

6.2 Requerimientos y pruebas de validación

Para la elaboración del segundo mockup, ademas de las funcionalidades y prioridades analizadas en el backlog, se ha tenido en cuenta el feedback recibido por parte de los usuarios para dar prioridad a las funciones y mejoras mas valoradas por estos. De esta forma, para este segundo MVP, se decide realizar las siguientes implementaciones:

6.2.1. Autenticación:

Los usuarios deberán estar identificados en la aplicación para poder empezar a utilizarla. De esta forma, a parte de seguridad en los accesos, se realizará una mejor gestión de la información del usuario, garantizando así una mejor experiencia en el uso y manejo de la aplicación.

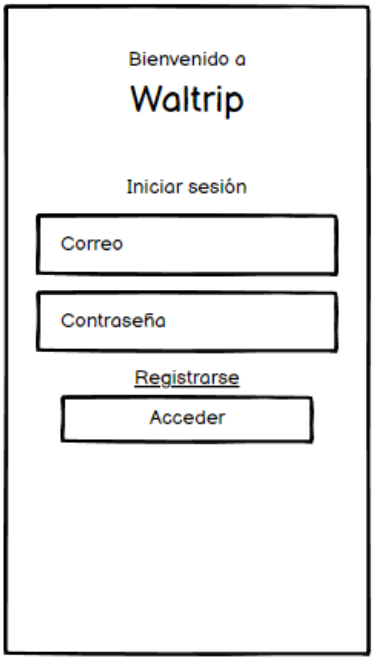


Figura 6.1: Mockup segundo MVP: Inicio de sesión.

Figura 6.2: Mockup segundo MVP: Registro.

Pruebas de validación (UAT):

Tabla 6.1: Pruebas de validación segundo MVP.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Autenticación de usuarios	Registrar usuario	El usuario queda registrado en el sistema
Autenticación de usuarios	Iniciar sesión	El usuario puede acceder usando sus credenciales y el perfil es cargado

6.2.2. Filtrado de rutas:

Con el fin de facilitar la búsqueda de oferta de rutas, estas podrán ser filtradas por diferentes parámetros que las identifiquen.

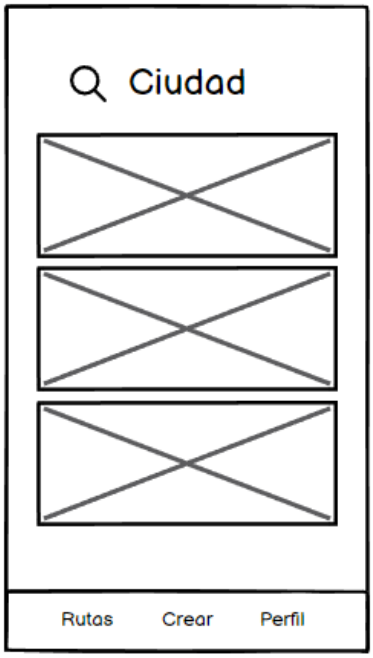


Figura 6.3: Mockup segundo MVP: Filtrado.

Pruebas de validación (UAT):

Tabla 6.2: Pruebas de validación segundo MVP.

Funcionalidad involucrada	Caso de prueba	Resultado esperado
Filtrar rutas	Buscar rutas por parámetro	Se obtienen las rutas con referencia al parámetro especificado

Con la implementación de estas mejoras buscamos acercarnos un poco más al producto que buscamos ofrecer, mostrando las diferentes utilidades disponibles y facilidades dispuestas a los usuarios.

6.3 Integración del Backend

Tras las implementaciones realizadas en la primera versión, a continuación, introduciremos diferentes cambios en las distintas capas con el fin de mejorar la estructura del código de nuestra aplicación y desarrollar las nuevas funcionalidades.

6.3.1. Repositorio

En cuanto a la capa de repositorios, para esta versión, integraremos también **Firebase Auth**, servicio que nos permitirá la autenticación y gestión de usuarios, permitiéndonos controlar los accesos y permisos en la aplicación. Para ello, ajustaremos el repositorio *ProfileRepository.dart* para incluir la nueva instancia:

ProfileRepository.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

class ProfilesRepository{

  FirebaseAuth getInstance() {
    return FirebaseAuth.instance;
  }

  ...

}
```

Por otro lado, incluiremos también una nueva instancia al repositorio *RouteRepository.dart* para la aplicación de atributos a las rutas que nos permitan filtrar sus búsquedas:

RoutesRepository.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';

class RoutesRepository {

  CollectionReference getAttributesReference(){
    return FirebaseFirestore.instance.collection('attributes');
  }

  ...

}
```

6.3.2. Entidades

Tras la implementación de las nuevas instancias en la capa de repositorios, y siguiendo la misma linea que con la primera versión, incluimos también la nueva entidad para el manejo de los atributos y su conversor correspondiente:

AttributeEntity.dart

```dart
class AttributeEntity {

  final String id;
  String value;

  AttributeEntity({
    required this.id,
    required this.value
  });

}
```

RouteConverter.dart

```dart
import 'package:waltrip/entities/AttributeEntity.dart';

class ProfileConverter {

  AttributeEntity mapToAttribute(Map map, String id){
    AttributeEntity attribute = AttributeEntity(
      id: id,
      value: map['value'],
    );
    return attribute;
  }

  Map<String, dynamic> attributeToMap(AttributeEntity attribute){
    Map<String, dynamic> attributeMap = Map();
    attributeMap.addAll( {
      attribute.id: attribute.value
    });
    return attributeMap;
  }

}
```

6.3.3. Servicios

En cuento a la capa de servicios, se han realizado diferentes ajustes para incluir las nuevas funcionalidades integradas, además de algunas mejoras en la presentación del código:

RoutesService.dart

```dart
import ...

```

```
class RoutesService {
  final RoutesRepository _repository = RoutesRepository();

  final RouteConverter _converter = RouteConverter();

  Future addRoute(RouteEntity route, Uint8List image) async {
    return await _repository
        .getRoutesReference()
        .add(_converter.routeToMap(route))
        .then((documentReference) {
      String id = documentReference.id ?? '';
      setRouteImage(id, route.imagePath, image);
      return id;
    });
  }

  Future addAttribute(String routeId, List<AttributeEntity> attributes) async {
    attributes.forEach((attribute) async {
      await _repository.getAttributesReference().doc(attribute.id).update({
        attribute.value: FieldValue.arrayUnion([routeId])
      });
    });
  }

  Future<List<RouteEntity>> getRoutesByAttribute(
      String path, String attribute) async {
    List<RouteEntity> routes = [];
    var documentSnapshot =
        await _repository.getAttributesReference().doc(path).get();
    if (documentSnapshot.exists) {
      List<dynamic> routeIds = documentSnapshot.get(attribute);
      for (var routeId in routeIds) {
        RouteEntity? route = await getRoute(routeId) as RouteEntity?;
        if (route != null) routes.add(route);
      }
    }
    return routes;
  }

  Future getRoutes() async {
    List<RouteEntity> routes = [];
    await _repository.getRoutesReference().get().then((querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        routes.add(_converter.mapToRoute(doc.data() as Map, doc.id));
      });
    });
    return routes;
  }

  Future getRoute(String id) async {
    return await _repository
        .getRoutesReference()
        .doc(id)
        .get()
        .then((documentSnapshot) {
      if (documentSnapshot.exists) {
        return _converter.mapToRoute(documentSnapshot.data() as Map, id);
      } else {
        return null;
```

```
62        }
63      });
64    }
65
66    Future deleteRoute(String id) async {
67      await _repository.getRoutesReference().doc(id).delete();
68    }
69
70    updateRoute(RouteEntity route) async {
71      Map<String, dynamic> routeMap = _converter.routeToMap(route);
72      await _repository.getRoutesReference().doc(routeMap['id']).update(routeMap);
73    }
74
75    setRouteImage(String id, String imagePath, Uint8List image) async {
76      Reference ref = _repository.getStorageReference().child(id + imagePath);
77      await ref.putData(image);
78    }
79
80    Future getRouteImage(String id, String imagePath) async {
81      Reference ref = _repository.getStorageReference().child(id + imagePath);
82      const oneMegabyte = 1024 * 1024 * 5;
83      Uint8List? imageUint = await ref.getData(oneMegabyte);
84      if (imageUint != null) {
85        return Image.memory(imageUint);
86      } else {
87        return null;
88      }
89    }
90  }
```

ProfileService.dart

```
1  import ...
2
3  class ProfilesService {
4    final ProfilesRepository _repository = ProfilesRepository();
5
6    final RoutesService _routesService = RoutesService();
7
8    final ProfileConverter _converter = ProfileConverter();
9
10   Future<String?> loginProfile(String email, String password) async {
11     final FirebaseAuth auth = FirebaseAuth.instance;
12
13     final UserCredential userCredential =
14         await auth.signInWithEmailAndPassword(
15       email: email,
16       password: password,
17     );
18
19     return userCredential.user?.uid;
20   }
21
22   Future<String?> addProfile(String email, String password) async {
23     final UserCredential userCredential =
24         await _repository.getInstance().createUserWithEmailAndPassword(
25               email: email,
26               password: password,
```

```
27            );
28
29      final User? user = userCredential.user;
30      if (user != null) {
31        final String userId = user.uid;
32        final String userName = userId.substring(0, 8);
33        await user.updateDisplayName(userName);
34        ProfileEntity profile = ProfileEntity(id: '', name: userName);
35        await _repository
36            .getReference()
37            .doc(userId)
38            .set(_converter.profileToMap(profile));
39      }
40
41      return userCredential.user?.uid;
42    }
43
44    Future<void> updateUProfileName(String newName) async {
45      User? user = _repository.getInstance().currentUser;
46      if (user != null) {
47        await user.updateDisplayName(newName);
48        updateName(user.uid, newName);
49      }
50    }
51
52    Future getProfile(String uid) async {
53      return await _repository
54          .getReference()
55          .doc(uid)
56          .get()
57          .then((documentSnapshot) {
58        if (documentSnapshot.exists) {
59          return _converter.mapToProfile(documentSnapshot.data() as Map, uid);
60        } else {
61          return null;
62        }
63      });
64    }
65
66    Future getProfiles() async {
67      List<ProfileEntity> list = [];
68      return await _repository.getReference().get().then((querySnapshot) {
69        querySnapshot.docs.forEach((doc) {
70          list.add(_converter.mapToProfile(doc.data() as Map, doc.id));
71        });
72        return list;
73      });
74    }
75
76    addRoute(String routeId, String path) async {
77      await _repository.getReference().doc(getCurrentUserUid()).update({
78        path: FieldValue.arrayUnion([routeId])
79      });
80    }
81
82    updateName(String uid, String newName) async {
83      await _repository.getReference().doc(uid).update({"name": newName});
84    }
85
```

```
86    deleteRoute(String routeId, String path) async {
87      await _repository.getReference().doc(getCurrentUserUid()).update({
88        path: FieldValue.arrayRemove([routeId])
89      });
90    }
91
92    Future getRoutes(String path) async {
93      List<RouteEntity> routes = [];
94
95      return await _repository
96          .getReference()
97          .doc(getCurrentUserUid())
98          .get()
99          .then((DocumentSnapshot documentSnapshot) async {
100       if (documentSnapshot.exists) {
101         List<dynamic> ids = documentSnapshot.get(path);
102         if (ids.length > 0) {
103           routes = await getRouteEntityByIdList(ids, 0, routes);
104         }
105       }
106       return routes;
107     });
108   }
109
110   String getCurrentUserUid() {
111     User? user = _repository.getInstance().currentUser;
112     String uid = user?.uid ?? '';
113     return uid;
114   }
115
116   Future getRoutesId(String path) async {
117     List<String> ids = [];
118
119     return await _repository
120         .getReference()
121         .doc(getCurrentUserUid())
122         .get()
123         .then((DocumentSnapshot documentSnapshot) async {
124       if (documentSnapshot.exists) {
125         return documentSnapshot.get(path);
126       } else {
127         return ids;
128       }
129     });
130   }
131
132   Future getRouteEntityByIdList(
133       List<dynamic> idList, index, List<RouteEntity> result) async {
134     RouteEntity route =
135         await _routesService.getRoute(idList[index]) as RouteEntity;
136     if (route != null) result.add(route);
137     if (index == idList.length - 1) {
138       return result;
139     } else {
140       return getRouteEntityByIdList(idList, ++index, result);
141     }
142   }
143 }
```

En el caso de *RoutesService.dart*, se han incluido los métodos *addAttribute()* y *getRoutesByAttribute()* que permiten añadir atributos a las rutas y obtenerlas en base a estos respectivamente, además de diferentes ajustes en la estructura de las funciones para eliminar controles de excepción innecesarios que serán controlados en capas superiores y el tratamiento de nulos.

Por otro lado, en *ProfileService.dart*, además de haber realizado también ajustes en el control de excepciones y nulos, tras la integración con **Firebase Auth**, se han añadido los métodos *loginProfile()* y *updateProfileName()*, para poder acceder a la aplciación mediante claves de acceso y poder actualizar el nombre del perfil respectivamente, y ajustado *addProfile()* para que trabaje con dicho servicio. De esta forma, dejamos de utilizar *Flutter SharedPreferences*, implementado en *ProfileManager.dart*, para la gestión de los usuarios. Por último, cabe también recalcar la reducción en el numero de funciones en *ProfileService.dart* dado que el parámetro *path*, de funciones como *addRoute()*, pasara a ser definido en la capa de controladores, en vez de en la de servicios. De esta forma, reducimos la longitud del código y facilitamos el manejo del mismo.

6.4 Funciones implementadas.

Tras finalizar el desarrollo del segundo MVP, a continuación, exponemos las funciones y mejoras implementadas, con las que contará esta versión. Como podremos apreciar, algunas de las pantallas ya presentadas anteriormente han sido modificadas para poder integrar las nuevas características desarrolladas:

6.4.1. Autenticación de usuarios.

Permite el registro de un usuario mediante la especificación de un correo y contraseña, los cuales, serán las claves de acceso al perfil en la aplicación. Esta funcionalidad ha sido implementando utilizando los servicios de Firebase Auth[10], sistema de gestión de usuarios integrado con Firebase.

Implementación

ProfileController.dart

```
import ...

class ProfilesController {

  final ProfilesService _profilesService = ProfilesService();

  String? register(String email, String password) {
    _profilesService.addProfile(email, password);
  }

  String? login(String email, String password) {
    _profilesService.loginProfile(email, password);
  }

  ...

}
```

La función *register()* y *login()* permiten registrarse o acceder en la aplicación al usuario respectivamente, mediante el paso de sus credenciales.

Interfaz

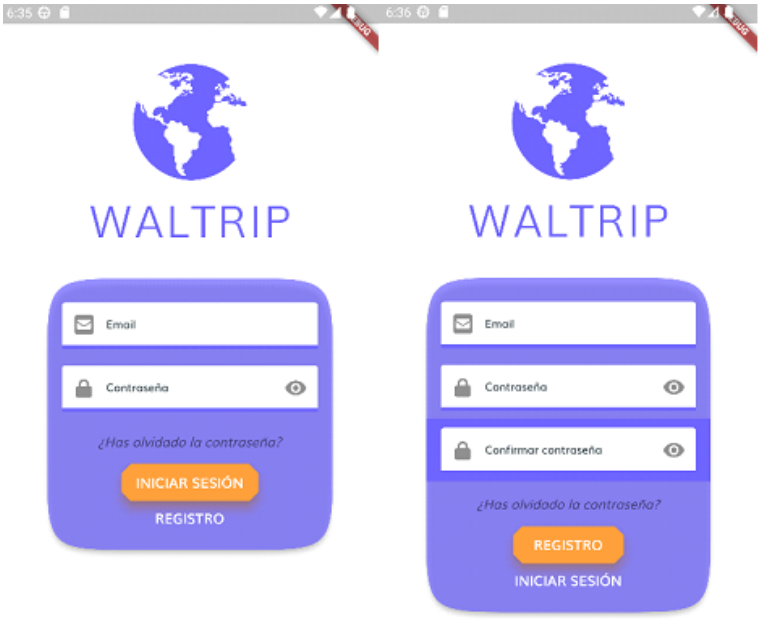


Figura 6.4: Interfaz segundo MVP: Inicio de sesión y registro.

6.4.2. Nombre de usuario.

Permite a los usuario definir su nombre de perfil para la aplicación.

Implementación

ProfileController.dart

```
import ...

class ProfilesController {

  final ProfilesService _profilesService = ProfilesService();

  updateUserName(String name) async {
    _profilesService.updateUProfileName(name);
  }

  ...

```

```
}
```

La función *updateUserName()* permite al usuario cambiar su apodo especificando el nuevo valor.

Interfaz

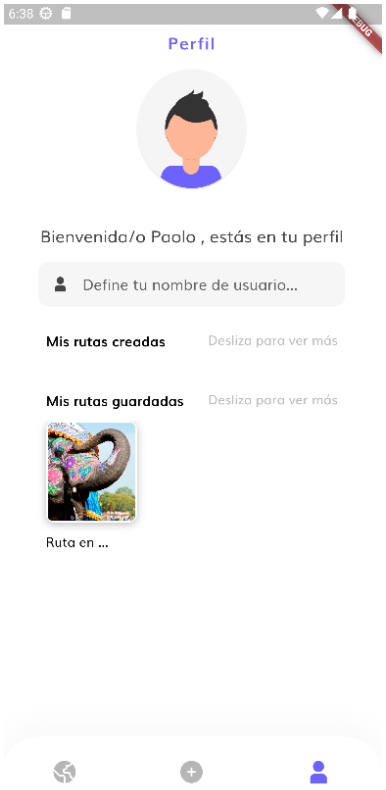


Figura 6.5: Interfaz segundo MVP: Perfil de usuario.

6.4.3. Ciudad de la ruta.

Permite especificar la ciudad donde se localizará la ruta generada.

Implementación

RoutesController.dart

```dart
import ...

class RoutesController {
  final RoutesService _service = RoutesService();

  final ProfilesService _profilesService = ProfilesService();

  final ProfileManager _profileManager = ProfileManager();

  static const CITIES_ATTRIBUTE_ID = "cities";
```

```
11    static const CREATED_ROUTES_PATH = "createdRoutes";
12
13    Future createRoute(
14        String title, String description, XFile image, String city) async {
15      String profile = _profilesService.getCurrentUserUid();
16
17      RouteEntity route = RouteEntity(
18          id: '',
19          title: title,
20          description: description,
21          creator: profile,
22          imagePath: "/" + image.name);
23      AttributeEntity cities = AttributeEntity(id: CITIES_ATTRIBUTE_ID, value: city);
24      List<AttributeEntity> attributes = [cities];
25      return await _service
26          .addRoute(route, await image.readAsBytes())
27          .then((id) {
28        print('Route with id ' + id + ' created');
29        _service.addAttribute(id, attributes);
30        _profilesService.addRoute(id, CREATED_ROUTES_PATH);
31        return id;
32      });
33    }
34
35    ...
36
37  }
```

La función *createRoute()* a sido ajustada para incluir la asignación del atributo ciudad a la ruta, mediante una llamada al servicio correspondiente tras su creación.

Interfaz

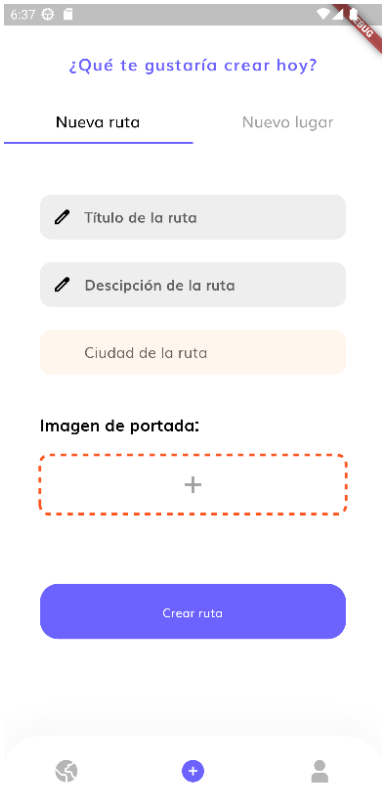


Figura 6.6: Interfaz segundo MVP: Creación.

6.4.4. Filtrar rutas.

Permite realizar búsquedas de rutas filtrando los resultados. Actualmente, el filtrado puede realizarse por ciudad.

Implementación

RoutesController.dart

```
import ...

class RoutesController {
  final RoutesService _service = RoutesService();

  final ProfilesService _profilesService = ProfilesService();

  final ProfileManager _profileManager = ProfileManager();

  static const CITIES_ATTRIBUTE_ID = "cities";
  static const SAVED_ROUTES_PATH = "savedRoutes";
  static const CREATED_ROUTES_PATH = "createdRoutes";

  Future getRoutesByCity(String city) async {
```

```
15      List<RouteObject> routes = [];
16      List<RouteEntity> routeEntityList =
17          await _service.getRoutesByAttribute(CITIES_ATTRIBUTE_ID, city);
18      if (routeEntityList.isEmpty) {
19        return routes;
20      } else {
21        return getRoutesByEntityList(routeEntityList, 0, routes);
22      }
23    }
24
25    Future getRoutesByEntityList(
26        List<RouteEntity> entityList, index, List<RouteObject> result) async {
27      Image routeImage = await _service.getRouteImage(
28          entityList[index].id, entityList[index].imagePath);
29      ProfileEntity routeProfile =
30          await _profilesService.getProfile(entityList[index].creator);
31
32      List<dynamic> savedRoutes =
33          await _profilesService.getRoutesId(SAVED_ROUTES_PATH);
34      entityList[index].isFavourite = savedRoutes.contains(entityList[index].id);
35
36      RouteObject route = RouteObject(
37          routeInfo: entityList[index],
38          routeImage: routeImage,
39          routeCreator: routeProfile);
40
41      result.add(route);
42      if (index == entityList.length - 1) {
43        return result;
44      } else {
45        return getRoutesByEntityList(entityList, ++index, result);
46      }
47    }
48
49    ...
50
51  }
```

La función *getRoutesByCity()* permite obtener una lista de rutas *RouteObject* en base al atributo ciudad que tengan asignado.

Interfaz

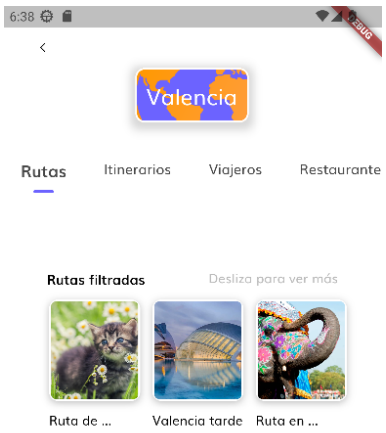


Figura 6.7: Interfaz segundo MVP: Filtrado.

6.5 Feedback de los usuarios.

De la misma forma que con la primera versión, con la elaboración de un segundo formulario analizamos la opinión de los usuarios tras las implementaciones y mejoras realizadas en este segundo Mínimo Producto Viable o MVP. Este formulario junto a las respuestas obtenidas, puede encontrarse en el Anexo C, *Formulario segundo MVP*. Partiendo de las respuestas proporcionada por los encuestados, en primer lugar, resaltaremos la buena aceptación de algunas de las nuevas funcionalidades implementadas. La posibilidad de poder acceder con tus propias credenciales a la aplicación y poder empezar a buscar rutas mediante la utilización de filtros, han ayudado al usuario a entender mejor la aplicación y comenzar a ver el potencial del producto. Esto podemos denotarlo, sobre todo, en el mismo formulario, por los comentarios recibidos sobre las funcionalidades, en general, de la aplicación. A pesar de la mejora, los usuarios han echado en falta una mayor capacidad de categorización y filtración de rutas en sus búsquedas, además de la inclusión de más detalles sobre la ruta. En concreto, algunos usuarios han sentido la necesidad de poder empezar a utilizar los mapas y la navegación en ruta, además de la posibilidad de incluir puntos de interés.

Por otro lado, lo usuarios se ven agradados por la interfaz de usuario implementada y las lineas de diseño escogida, además, también, de su capacidad para adaptarse a los diferentes tamaños de dispositivos. En este sentido, se intentará mantener el camino seguido hasta ahora en la

implementación de la interfaz, permitiendo centrarnos en potenciar las funcionalidades y características de la aplicación.

En general la experiencia de los usuarios, pese a sus limitaciones, esta siendo positiva, pero existe un gran interés en potenciar los diferentes aspectos distintivos de Waltrip. La capacidad para guardar rutas y llevarlas siempre contigo, el poder organizarlas y disponer de ellas en cualquier momento, y la posibilidad de descubrir nuevos lugares tanto en tus viajes como en los lugares que frecuentas, son algunas de las características más valoradas y que deben ser el foco de atención del desarrollo en posteriores versiones.

Entrando en detalle con algunos de los comentarios recibidos, algunas de las sugerencias de mejora que podemos encontrar son:

- **Clasificación de rutas:** Dar la posibilidad de tener tus rutas organizadas por categoría, temática, filtros, etc.
- **Marcar puntos de interés en los mapas:** Poder añadir puntos de interés en las rutas y mapas.
- **Compartir, importar y exportar rutas:** Poder disponer de rutas a través de diferentes vías, pudiendo compartirlas, exportarlas o integrarlas en la aplicación.
- **Reseñas, puntuaciones y comentarios:** Potenciar el apartado social, mediante valoraciones de los usuarios y permitiendo conectar entre ellos.
- **Planes de rutas:** Permitir crear experiencias conjuntas para varios días.

A modo de conclusión, y con el fin de influir en los objetivos de desarrollo para las futuras versiones de la aplicación, a pesar de haber conseguido transmitir la esencia del producto que buscamos ofrecer, en el futuro, deberemos enfocarnos en la mejora e incremento de las funcionalidades de la aplicación con el fin de alcanzar el atractivo buscado por nuestro cliente, además, de enfocarnos en aquellas que atraigan a la oferta y nos permitan crear un activo atractivo tanto para el usuario como para el proveedor.

6.6 Evolución y futuras implementaciones.

Tras la implementación de los diferentes MVP, y enfocándonos en la hoja de ruta marcada por el backlog junto a los objetivos marcados antes del inicio del desarrollo, podemos observar como algunas funcionalidades consideradas de alta prioridad para la aplicación no han sido contemplada en ninguna de las versiones presentadas anteriormente.

Durante el desarrollo de la primera y segunda versión nos hemos encontrado con diferentes dificultades y situaciones que nos han llevado a considerar diversos cambios en el enfoque y futura evolución de la aplicación. Algunos de estos aspectos afectan tanto al modelo de negocio como a la experiencia de los usuarios en el uso de la aplicación. Por ello, y como resultado del proceso de desarrollo llevado acabo hasta el momento, planteamos, a continuación, los siguientes planteamientos y acciones a realizar en el futuro para la mejora de la viabilidad y funcionalidad de negocio de Waltrip.

6.6.1. Creación y oferta de rutas

Hasta el momento planteábamos la opción de que los usuarios creasen sus propias rutas y así se ha evidenciado en las versiones presentadas. Aunque consideramos que, al menos inicialmente, esto ha sido una buena forma de acercar más al usuario a la idea desarrollada, mostrándoles la capacidad de la aplicación a registrar nuevas rutas en el sistema y que estas estas puedan ser accesible ágilmente, y, sobre todo, para obtener un primer feedback sobre la aplicación en general, existen ciertas barreras en cuanto a viabilidad y esfuerzo que consideramos que pueden tomar un rol clave en el éxito o fracaso de Waltrip.

Entre los diferentes aspectos considerados encontramos:

- **Complejidad:**
  El desarrollo de un sistema de creación y edición de rutas presenta una alta complejidad a nivel técnico que puede repercutir en un elevado coste de producción, servicios y de tiempo que difícilmente puede llegar a ser asumido en una temprana etapa de desarrollo.

- **Oferta:**
  La implementación de un sistema de creación de rutas no asegura, en ningún caso, un nivel adecuado de oferta en la aplicación. A pesar de ofrecer las herramientas necesarias, crear una ruta seguirá siendo algo costoso para los usuarios. Necesitas disponer de ciertos conocimientos y estar dispuesto a invertir tu tiempo en un ejercicio que, en ocasiones, puede ser considerado poco rentable. Esto puede conllevar a una situación crítica en etapas tempranas en las que el nivel de usuarios pueda no ser el deseable, afectando a la generación de oferta y llevando la aplicación a una situación de ”hibernación” en la que no se crea oferta y tampoco se unen usuarios por la falta de esta.

- **Calidad:**
  Al ser los usuarios quienes crean las rutas, resulta complicado asegurar la calidad de las mismas. En una etapa temprana es importante mantener un nivel de calidad que atraiga al público objetivo e incentive el uso de la aplicación, por lo que la falta de filtros en este aspecto puede afectar negativamente a la viabilidad de la aplicación.

- **Gestión:**
  Los usuarios deberían poder generar un ingreso por la venta de sus rutas creadas, de esta forma, se define el motivo e incentivo principal de la participación en la oferta de la aplicación. A pesar de que esto es un motor interesante para el aumento de la oferta, gestionar todo este sistema puede ser complejo y llevarnos a unos costes que difícilmente podamos soportar en los inicios.

En conclusión, entendemos que desarrollar y gestionar un sistema de creación y edición de rutas puede ser una buena idea para el futuro, cuando la aplicación se encuentre asentada y este preparada para dar el salto a hacia nuevas propuestas y modelos de negocio pero, en ningún caso, comprendido dentro de los horizontes de las primeras iniciativas abordadas para las primeras versiones de la aplicación.

Sin embargo, lejos de abandonar la idea, a continuación, se plantea un nuevo modelo para la oferta de rutas:

Mapeo e integración de rutas de proveedores.

Como nuevo modelo para la generación de oferta de rutas en Waltrip, planteamos la generación de un algoritmo de mapeo de rutas junto a la integración de proveedores a través de la disposición de una API u otro canal de distribución. El sistema funcionaria de la siguiente forma:

Una entidad dispone de un conjunto de rutas en su catálogo, este nos dispone de la información necesaria para generar dichas rutas de nuestro lado a través de un canal de distribución definido. La información del proveedor puede venir en forma de un esquema XML, JSON o cualquier otro formato con el que nosotros podamos tratar. La información trasmitida por el proveedor es procesada a través de un algoritmo y convertida en las diferentes rutas dispuestas por este en un formato entendible de nuestro lado. A este paso lo denominamos 'mapeo de rutas' y se realiza a través de un 'algoritmo de mapeo'. De esta forma, traducimos las rutas del proveedor en entidades con las que nosotros podamos trabajar y hacer llegar la oferta para su consumo al cliente final.

Este sistema de integración soluciona parte de los problemas con los que podíamos encontrarnos en la idea inicial, dado que eliminamos la dependencia de los usuarios y limitamos la actividad de la oferta a la proporcionada por proveedores con los que podemos tratar directamente, asegurando la calidad, disponibilidad y control de las rutas ofertadas.

Este nuevo planteamiento genera la necesidad de desarrollar el algoritmo y sistemas implicados en esta tarea pero, al no depender de interfaces y otros elementos complejos, su implementación es más viable que el modelo anterior, además de plantear un modelo de negocio más versátil por la posibilidad de incluir imposiciones en las transacciones y la posibilidad de abrir lineas de negocio enfocadas a los proveedores en el futuro.

6.6.2. Navegación

La implementación de los mapas dará lugar a una de las funcionalidades fundamentales de la aplicación, la navegación por las rutas. Esta utilidad no ha sido implementada dada su dependencia con la generación de rutas que, como se ha comentado en el punto anterior, su complejidad hacia que esta estuviese fuera del alcance de las primeras versiones.

El servicio elegido para llevar acabo esta tarea, y que fue planteado en el plan de negocio, es el del proveedor Mapbox. Hasta el momento no se ha decidido dar ningún cambio en esta dirección y se continuara con la expectativa de utilizar esta herramienta para dicha funcionalidad.

Su desarrollo será determinante para la viabilidad de la aplicación y formará parte de la definición final del producto, siendo este el motor principal para la atracción de usuarios y proveedores. Dada su importancia, se espera invertir un gran esfuerzo en este aspecto para que la experiencia de uso por parte del usuario sea lo más gratificante posible, ya que de este dependen, en gran parte, el resto de funcionalidades, aparte de ser uno de los factores de éxito claves.

6.6.3. Sistema de pagos

Otra de las funcionalidades claves para ejecutar el modelo de negocio planteado es el de la implementación de un sistema de pagos para la venta de rutas. En dependencia estricta de la oferta funcional de rutas, este sistema quede delegado para una futura implementación, una vez el resto de sistemas lleguen a un punto de desarrollo estable que permitan la adquisición de la oferta con ciertas garantías para el comprador.

Por el momento la plataforma escogida para la gestión de los pagos ha sido el servicio proporcionado por Stripe. En el futuro se podría plantear la implementación de otros sistemas de pago, como sistemas multi tarjeta, sistemas de puntos o mediante proveedores de tarjetas virtuales.

6.6.4. Plataformas disponibles

Desde los inicios del desarrollo se buscaba hacer disponible la aplicación a los dispositivos con sistema operativo Android e IOS, uno de los motivos principales del uso de framework Flutter. A pesar de ello, y aunque el código se encuentra preparado para ser ejecutado en IOS, actualmente, la aplicación solo se encuentra disponible para Android. Esto se debe principalmente a la falta de los equipos necesarios para el desarrollo en IOS, ya que es necesario disponer de un dispositivo Mac para ello, condición que no ha podido ser cubierta en la actualidad.

En el futuro se espera disponer de ambas versiones ya que esto lo consideramos dentro del proyecto como requisito necesario para poder acceder al mercado objetivo y garantizar la viabilidad del producto.

CAPÍTULO 7
Conclusiones y trabajos futuros

En el siguiente capitulo se exponen las conclusiones finales tras los trabajos realizados. Se realiza una reflexión sobre las tareas futuros y aplicativos de conocimientos durante la elaboración de este trabajo.

7.1 Conclusión

Tras todo el trabajo realizado en el análisis y desarrollo de la idea de negocio planteada en este documento, existen diferentes puntos a valorar en cuento al presente y futuro de la aplicación para la gestión de rutas Waltrip.

Por un lado, resaltar la evolución de la idea de negocio inicial desde su estudio en el plan de negocio, hasta el desarrollo de las dos primeras versiones de la aplicación. Muchas de las expectativas marcadas en los inicios se han visto replanteadas. Partiendo de una base más teórica y de análisis, con la búsqueda de actores similares en el mercado, comparativas de los productos ofrecidos y estimaciones en la viabilidad, progresión y evolución la aplicación, hemos visto como las diferentes barreras que hemos ido identificando durante el desarrollo, teniendo que combatir los problemas técnicos con los que nos encontrábamos y teniendo que ajustarnos, muchas veces, a algunas realidades del desarrollo de aplicaciones, hasta el momento desconocidas por parte del equipo, nos han llevado a cambios de enfoque en la ejecución de la idea que han variado parte de la utilidad de Waltrip e incluso nos han llevado a valorar otras lineas de negocio para el sustento y viabilidad del producto a futuro.

Con todo ello, durante el proceso hemos logrado el desarrollo de modelos para el producto en la dirección marcada por los objetivos y requerimientos buscados para Waltrip, implementando algunas de las funcionalidades claves para la presentación de la aplicación. Enfocándonos en el desarrollo del backend, resaltar el éxito en las diferentes integraciones de herramientas y servicios, que han permitido el desarrollo del conjunto de funcionalidades, y de las arquitecturas y procesos implementados.

Por otro lado, puede parecer que el ser humano siempre se pone en lo peor, pero en nuestro caso, hemos pecado de optimismo en el desarrollo de nuestra idea de negocio. Durante la creación de la aplicación, hemos sido conscientes de numerosas complicaciones, ya no solo a nivel de habilidades de programación, sino también a nivel de configuraciones y compatibilidad, que nos han llevado a replantearnos en numerosas ocasiones la viabilidad de en lo que estábamos trabajando dado los recursos de los que disponíamos. A pesar de ello, hemos intentado superar todas

las barreras adaptándonos a lo que realmente era posible ofrecer en esta etapa para la aplicación, buscando desarrollar los pilares de una idea que necesitaba ser pulida pero que, gracias al trabajo realizado, se han acabada asentando las bases para su futuro.

En conclusión, el trabajo realizado a resultado en un gran ejercicio de análisis y de puesta en practica de habilidades que marcarán la dirección en las siguientes etapas de desarrollo de Waltrip.

7.2 Relación académica

Como trabajo académico realizado para el Grado en Ingeniería Informática de la Universidad Politécnica de Valencia, este ha sido resultado de los diferentes conocimientos a nivel técnico y de la capacidad analítica y de ejecución adquiridos a lo largo de la titulación. Esto puede observarse a lo largo del documento, a través de las diferentes secciones trabajadas.

Por un lado, en el apartado del estudio de mercado, se trabaja la parte más analítica y de estudio de un proyecto de desarrollo de software, presentando la idea junto a las diferentes lineas de negocio que conformarán la base para el análisis y resultará en el conjunto de funcionalidades que definirán los diferentes aspectos y procesos llevados a cabo en el desarrollo de la aplicación. Asignaturas como *Análisis de requisitos de negocio* o *Estrategia y Diseño de la Organización* han servido de apoyo para el enfoque y elaboración de estas secciones.

Por otro lado, en los apartados de presentación de la solución propuesta, tecnologías y desarrollo, entramos en la parte más técnica y de implementación de procesos adquiridas en asignaturas como *Lenguajes, tecnologías y paradigmas de la programación*, por su conocimientos aportados para la generación del código de Waltrip,*Ingeniería del software*, apoyándonos en la definición de metodologías, arquitecturas, herramientas y procedimientos para la generación de software, o *Gestión de proyectos*, para el estudio y toma de decisiones rigurosas durante el proyecto.

De esta forma, este trabajo pone de manifiesto los conocimientos y capacidades adquiridas a lo largo del grado, haciendo posible el desarrollo efectivo del proyecto presentado.

7.3 Trabajos futuros

Tras el desarrollo de las dos primeras versiones de Waltrip y las conclusiones extraídas del trabajo realizado, a continuación, mencionamos algunas de las mejoras e implementaciones futuras a realizar:

- **Oferta de rutas:**
  Implementar un sistema para la oferta de rutas que permita conectar proveedores y ofrecer productos generado directamente desde la misma aplicación.

- **Navegación:**
  Implementar un sistema de navegación para las rutas, permitiendo al usuario hacer un uso más completo de su contenido.

- **Geolocalización:**
  Reconocer la localización del usuario y posicionamiento de las rutas para ofrecer al usuario contenido más preciso.

- **Sistema de pagos:**
  Integrar un sistema de pagos para permitir la compra de rutas a través de la aplicación.
- **Funcionalidades para el usuario:**
  Permitir a los usuarios compartir rutas, puntuarlas y comentar sobre ellas.
- **Plataformas:**
  Generar una versión multiplataforma de la aplicación para que esta pueda ser utilizada en Android, iOS e incluso en web.

Bibliografía

[1] Almorat. *Almorat en GitHub*. 2021. URL: https://github.com/almorat.

[2] Almorat. *Repositorio de Waltrip en GitHub*. 2021. URL: https://github.com/Almorat/Waltrip.

[3] *Android Studio*. Google LLC. 2021. URL: https://developer.android.com/studio.

[4] Apache. *Licencia Apache 2.0*. 2004. URL: https://www.apache.org/licenses/LICENSE-2.0.

[5] Tom Collings. «Controller-Service-Repository pattern». En: *Medium* (2021). URL: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5.

[6] *Dart*. Google LLC. 2021. URL: https://dart.dev/.

[7] *Eclipse*. Eclipse Companies LLC. 2021. URL: https://www.eclipse.org/.

[8] Firebase. *Cloud Firestore*. 2021. URL: https://firebase.google.com/docs/firestore?hl=es.

[9] firebase.google.com. *Firebase*. 2021. URL: https://firebase.google.com.

[10] firebase.google.com. *Firebase Auth for Flutter*. 2021. URL: https://pub.dev/packages/firebase_auth.

[11] firebase.google.com. *Firebase Core for Flutter*. 2021. URL: https://pub.dev/packages/firebase_core.

[12] *Flutter*. Google LLC. 2021. URL: https://flutter.dev/.

[13] Flutter-dev. *Flutter Documentation*. 2020. URL: https://flutter.dev/docs/get-started/install.

[14] Flutter.dev. *Shrared Preferences*. 2021. URL: https://pub.dev/packages/shared_preferences.

[15] *GitHub*. GitHub Inc. 2021. URL: https://github.com/.

[16] LEANSTACK INC. *Lean Canvas, business model*. 2023. URL: https://www.leancanvas.com/.

[17] Google Inc. *Dart, Map<K, V>class*. 2022. URL: https://api.dart.dev/stable/3.1.0/dart-core/Map-class.html.

[18] *IntelliJ IDEA*. JetBrains. 2021. URL: https://www.jetbrains.com/es-es/idea/.

[19] *Java Development Kit*. Oracle Corporation. 2021. URL: https://www.oracle.com/java/technologies/java-se-glance.html.

[20] *JetBrains*. Oracle Corporation. 2021. URL: https://www.jetbrains.com/.

[21] *Kotlin*. JetBrains. 2021. URL: https://kotlinlang.org/.

[22] *Mapbox*. Mapbox. 2021. URL: https://www.mapbox.com/.

[23] *Matriz DAFO*. 2023. URL: https://dafo.ipyme.org/Home.

[24] «Metodología Kanban». En: *Kanban Tool* (2021). URL: https://kanbantool.com/es/metodologia-kanban.

[25] A. Morata Gomila. *Waltrip: tu cartera de viaje. Plan de empresa*. Universitat Politècnica de València. 2021. URL: https://riunet.upv.es/handle/10251/171540.

[26] *NetBeans*. Oracle Corporation. 2021. URL: https://netbeans.apache.org/.

[27] P. Romero Salas. *Waltrip: tu cartera de viaje. Desarrollo de un test de concepto*. Universitat Politècnica de València. 2021. URL: http://hdl.handle.net/10251/174988.

[28] *Sabi*. 2021. URL: https://sabi.bvdinfo.com/.

[29] *Spring*. VMware INC. 2022. URL: https://spring.io/.

[30] Tobrun. *Flutter Mapbox GL*. 2021. URL: https://pub.dev/packages/mapbox_gl.

[31] Linus Torvalds. *Git*. 2021. URL: https://git-scm.com/.

APÉNDICE A
Manual de uso

A continuación, se presenta la guía de uso para las diferentes versiones disponible de Waltrip.

A.1 Requerimientos

La aplicación se encuentra disponibles para dispositivos con sistemas Android, por lo que es indispensable contar con este sistema para poder empezar a utilizar la aplicación.

A.2 Instalación de la aplicación

Para instalar la aplicación es necesario disponer del APK, disponible a través de los siguientes enlaces siguiente enlace:

Waltrip, tu cartera de viaje. Primer MVP

Waltrip, tu cartera de viaje. Segundo MVP

A.3 Manual de usuario primer MVP

La primera vez que accedamos a la aplicación, como se muestra en la figura A.1 nos encontraremos con una pequeña presentación que nos mostrará un resumen de que podemos hacer con Waltrip.

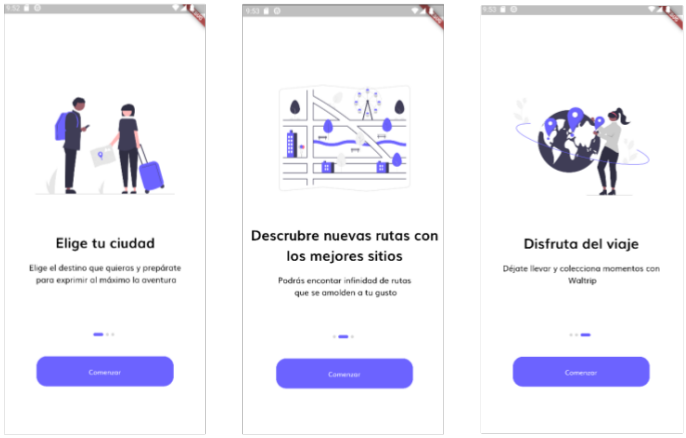


Figura A.1: Onboarding primer MVP

Una vez recorramos las diferentes pantallas de presentación, podremos acceder a la pantalla principal de la aplicación haciendo click sobre el botón ”Comenzar”.

En la figura A.2 podemos ver dicha pantalla donde se mostrará parte del contenido disponible en la aplicación y accesos a las diferentes utilidades. Abajo de esta pantalla encontraremos el menú de navegación. Para volver en cualquier momento a esta pantalla, solo es necesario hacer click sobre el icono del ”globo terraquio”, situado a la izquierda del menú.

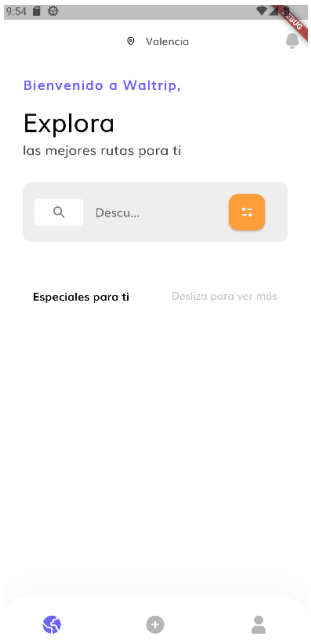


Figura A.2: Pantalla principal primer MVP

A través del icono de la ”persona”, situado a la derecha en el menú de navegación, podemos acceder a la pantalla de perfil de usuario, donde podemos definir el perfil a utilizar introduciendo

el nombre respectivo en la casilla "Introduce tu nombre". Una vez cargado el perfil, podremos ver las rutas creadas y guardadas. Esto puede verse en la figura A.3 a continuación.

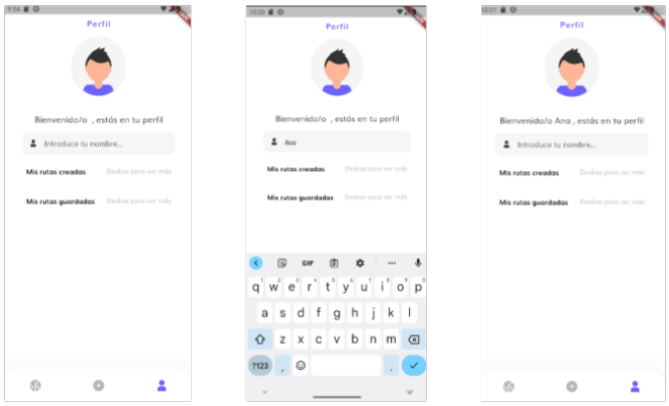


Figura A.3: Identificación primer MVP

También, una vez esté el perfil seleccionado, podremos empezar a interactuar con las diferentes funcionalidades de la aplicación. En la figura A.4 vemos como en la pantalla principal ya aparecen las rutas disponibles.

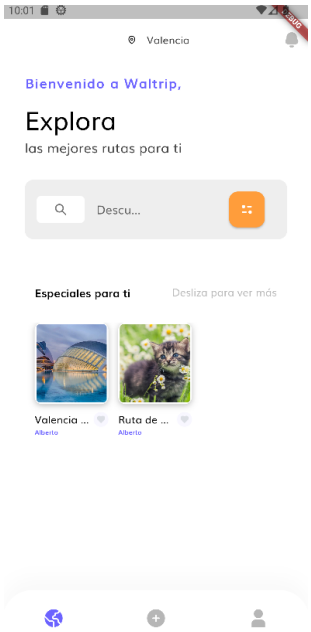


Figura A.4: Pantalla principal primer MVP

Si pulsamos sobre el icono "+", en el centro del menú de navegación, accederemos a la pantalla de creación de rutas, figura A.5.

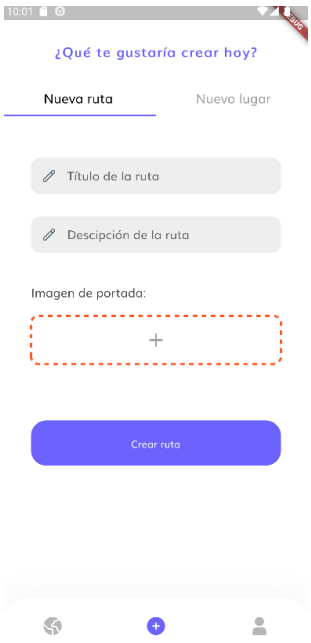


Figura A.5: Pantalla crear rutas primer MVP

En la pantalla de creación, podremos generar una nueva ruta definiendo un nombre, descripción y adjuntando una imagen como portada de presentación de la ruta. Una vez hayamos introducido esta información, podremos pulsar sobre el botón "Crear ruta" y está será almacenada en el sistema, figura A.6.

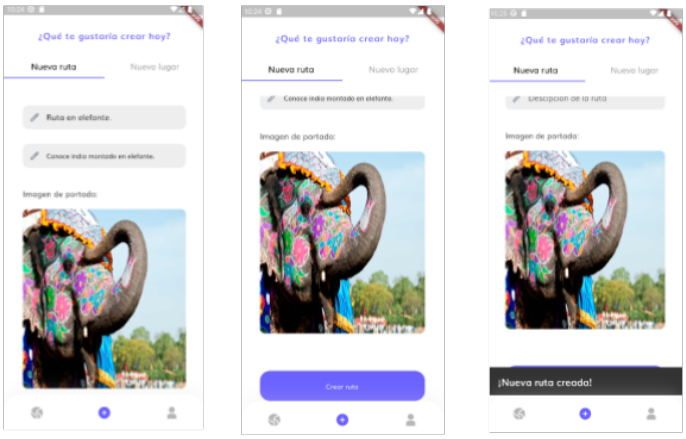
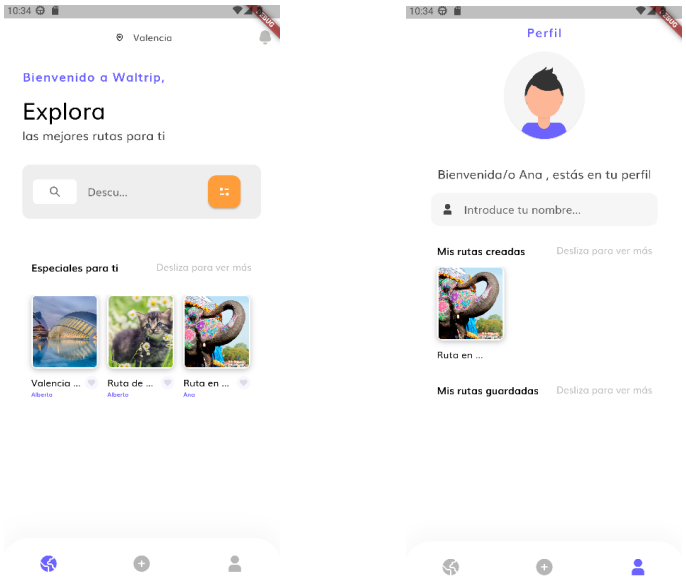


Figura A.6: Crear rutas primer MVP

De esta forma, la ruta se hará accesible a través de la pantalla principal, figura A.15a, y quedará guardada en tu perfil como una de tus rutas creadas, figura A.15b.



(a) Pantalla principal **(b)** Pantalla perfil

Figura A.7: Ruta creada primer MVP

Aparte de crear rutas, también podemos guardarlas. Como se muestra en la figura A.16, para ello, accedemos a una ruta haciendo click sobre ella y la añadimos a favoritos pulsando sobre el botón "Añadir a favoritos", veremos como el corazón que se muestra en la pantalla se vuelve rojo, indicando que hemos guardado la ruta. De la misma forma, si volvemos a pulsar sobre el (ahora indicando "Quitar de favoritos"), la eliminaremos de favoritos y el corazón volverá a su estado inicial. Además, como habremos podido observar, accediendo a la ruta podemos ver toda su información al detalle.

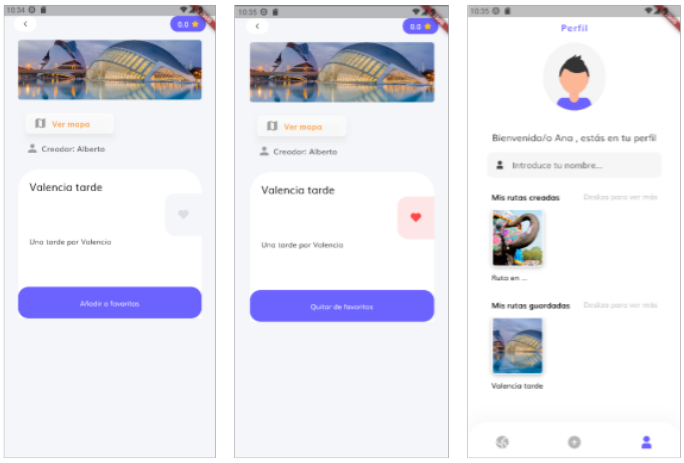


Figura A.8: Añadir ruta a favoritos primer MVP

A.4 Manual de usuario segundo MVP

La primera vez que accedamos a la aplicación, como se muestra en la figura A.9 nos encontraremos con una pequeña presentación que nos mostrará un resumen de que podemos hacer con Waltrip.

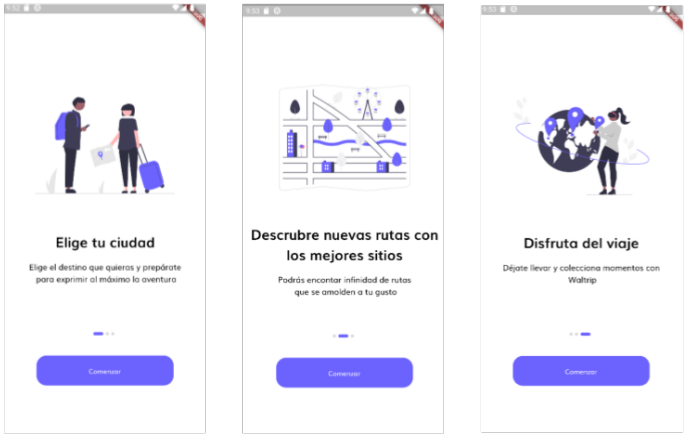


Figura A.9: Onboarding segundo MVP

Una vez recorramos las diferentes pantallas de presentación, podremos acceder a la pantalla de acceso, figura A.10, de la aplicación haciendo click sobre el botón ”Comenzar”.

Una vez en esta pantalla, necesitaremos disponer de nuestras credenciales (correo electrónico y contraseña) para poder iniciar sesión con nuestro perfil en la aplicación. En caso de no disponer de estas, podemos obtenerlas pulsando sobre la opción ”registro”.

Figura A.10: Autenticación segundo MVP

Una vez nos hayamos registrado o iniciado sesión, accederemos a la pantalla principal, figura A.11, donde se mostrará parte del contenido disponible en la aplicación y accesos a las diferentes utilidades. Abajo de esta pantalla encontraremos el menú de navegación. Para volver en cualquier momento a esta pantalla, solo es necesario hacer click sobre el icono del "globo terraquio", situado a la izquierda del menú.

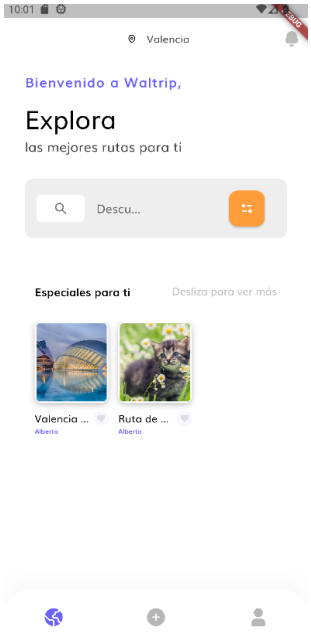


Figura A.11: Pantalla principal segundo MVP

En la pantalla principal, a través de la barra de búsqueda ”Descubre”, podremos buscar rutas por ciudad. Para ello, introducimos el nombre de la ciudad y pulsamos sobre el icono de la lupa. Tras esto, accederemos a la pantalla de búsqueda, figura A.12, donde podremos ver las distintas rutas en referencia a la ciudad indicada.

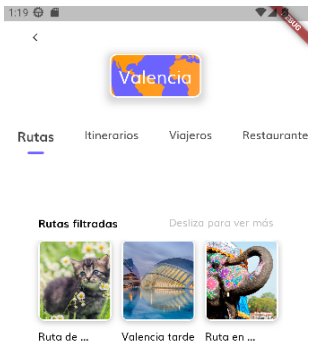


Figura A.12: Filtrado de rutas segundo MVP

Volviendo a la pantalla principal, retrocediendo en la aplicación. A través del icono de la ”persona”, situado a la derecha en el menú de navegación, podemos acceder a la pantalla de perfil de usuario, donde podemos definir un nombre para nuestro perfil introduciéndolo en la casilla ”Define tu nombre de ususario”. En esta pantalla, podremos ver nuestras rutas creadas y guardadas. Esto puede verse en la figura A.13 a continuación.

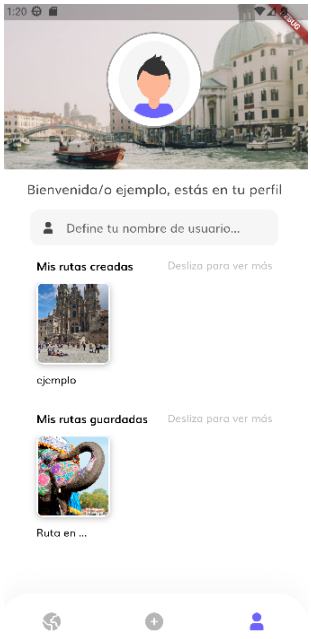


Figura A.13: Perfil segundo MVP

Pulsando sobre el icono "+", en el centro del menú de navegación, accederemos a la pantalla de creación de rutas, figura A.14.

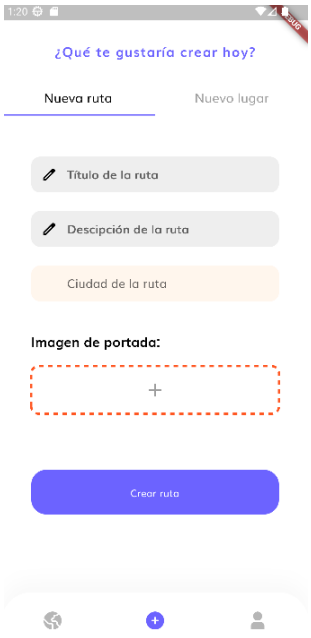
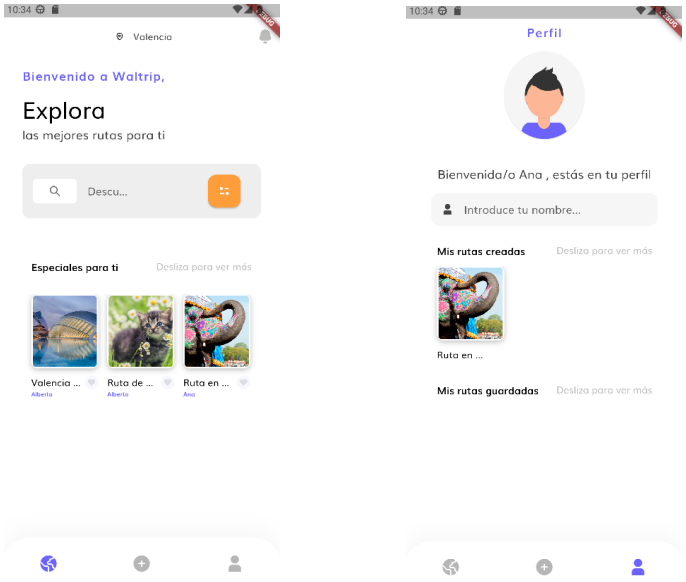


Figura A.14: Pantalla crear rutas segundo MVP

En la pantalla de creación, podremos generar una nueva ruta definiendo un nombre, descripción, ciudad y adjuntando una imagen como portada de presentación de la ruta. Una vez hayamos

introducido esta información, podremos pulsar sobre el botón ”Crear ruta” y está será almacenada en el sistema, figura A.6.

De esta forma, la ruta se hará accesible a través de la pantalla principal, figura A.15a, y quedará guardada en tu perfil como una de tus rutas creadas, figura A.15b.



(a) Pantalla principal **(b)** Pantalla perfil

Figura A.15: Ruta creada primer MVP

Aparte de crear rutas, también podemos guardarlas. Como se muestra en la figura A.16, para ello, accedemos a una ruta haciendo click sobre ella y la añadimos a favoritos pulsando sobre el botón ”Añadir a favoritos”, veremos como el corazón que se muestra en la pantalla se vuelve rojo, indicando que hemos guardado la ruta. De la misma forma, si volvemos a pulsar sobre el (ahora indicando ”Quitar de favoritos”), la eliminaremos de favoritos y el corazón volverá a su estado inicial. Además, como habremos podido observar, accediendo a la ruta podemos ver toda su información al detalle.

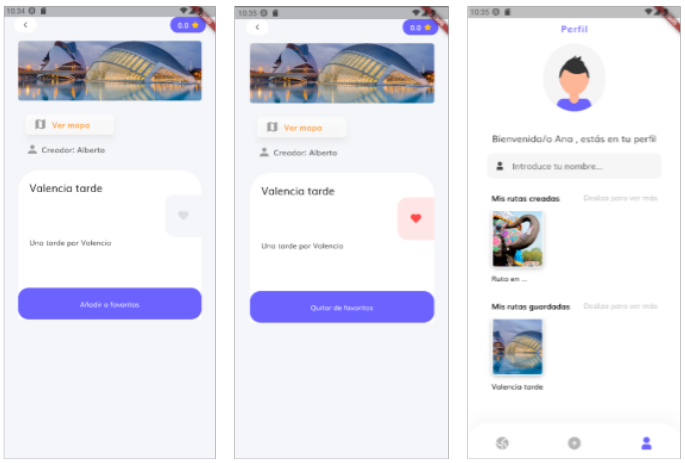


Figura A.16: Añadir ruta a favoritos primer MVP

APÉNDICE B
Formulario primer MVP

B.1 Cuestiones primer MVP

A continuación, en la siguiente página, se exponen las cuestiones presentadas a los usuarios para obtener feedback de la primera versión presentada:

Formulario TFG, primer MVP.

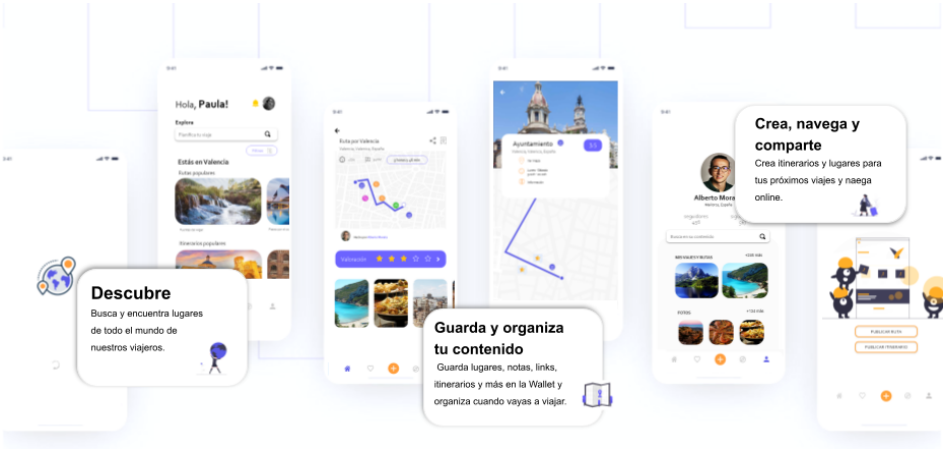
Somos Paula y Alberto, dos alumnos de la UPV que estamos desarrollando una aplicación de turismo para nuestro TFG de carácter emprendedor y hoy necesitamos tu ayuda!

A modo de de resumen, la aplicación te permite realizar rutas turísticas directamente desde tu dispositivo móvil. Podrás buscar entre multitud de rutas para diferentes ciudades Españolas, guardarlas en tu perfil, compartirlas, puntuarlas e incluso crear las tuyas propias.

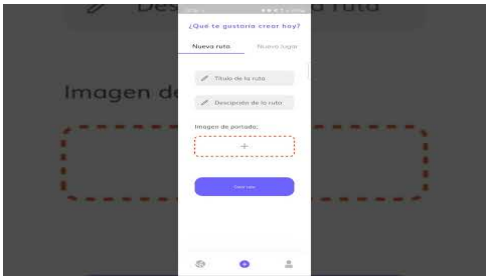
El objetivo principal de esta encuesta es valorar las funcionalidades integradas en esta primera versión de la aplicación e indicarnos que mejoras realizarías o que funciones echas en falta. (Te encontraras ante una prueba de concepto, seguro que algo le puedes sacar 😊)

La encuesta te llevará muy poquito tiempo y nos permitirá mejorar tu futura experiencia con la aplicación! Quién sabe si estarás participando a mejorar una app que el tu del futuro podría estar utilizando... 👀

* Indica que la pregunta es obligatoria



Video Introducción primer MVP



http://youtube.com/watch?v=djTk5KEuPak

Diseño de la aplicación

A continuación, se le realizarán unas cuestiones sobre lo que ve en pantalla.

1. ¿Cuál fue su impresión inicial del diseño? *

Marca solo un óvalo.

- ◯ Insatisfactorio
- ◯ Pobre
- ◯ Justo
- ◯ Bien
- ◯ Excelente

2. En una escala del 1 al 5, ¿En qué grado considera atractivo lo siguiente? *

Marca solo un óvalo por fila.

	Nada atractivo	Poco atractivo	Indiferente	Atractivo	Muy atractivo
¿Qué le parecen los colores?					
¿Qué le parece el tipo de fuente?					
¿Qué le parece el tamaño de fuente?					
¿Qué le parece el tamaño de los botones?					

3. ¿Cuál es tu opinión sobre la organización de la información en pantalla? *

Marca solo un óvalo.

- Muy confuso
- Un poco confuso
- Claro
- Muy claro

4. ¿Consideras que los iconos usados para definir el menú representan el contenido que incluyen? *

Marca solo un óvalo.

- ◯ Muy confuso
- ◯ Un poco confuso
- ◯ Claro
- ◯ Muy claro

5. ¿Tienes algún comentario o surgerencia que pueda ayudarnos a mejorar la interfaz de usuario para nuestros clientes?

_____

Funciones de la aplicación

A continuación, se le realizarán unas cuestiones sobre lo que puede realizar en la aplicación.

6. ¿Cuál fue su impresión inicial sobre las funciones implementadas en la primera versión de la aplicación? *

Marca solo un óvalo.

- ◯ Insatisfactorio
- ◯ Pobre
- ◯ Justo
- ◯ Bien
- ◯ Excelente

7. En una escala del 1 al 5, ¿En qué grado considera útil lo siguiente? *

Marca solo un óvalo por fila.

	Nada útil	Poco útil	Indiferente	Útil	Muy útil
Buscar rutas por filtros					
Guardar rutas					
Crear rutas					
Navegar por las rutas					
Compartir las rutas					

8. ¿Qué opina sobre las funciones ofrecidas, en general, por la aplicación? *

Marca solo un óvalo.

Insuficientes

1

2

3

4

5

Suficientes

9. ¿Se le ocurre alguna mejoras para las funciones ya implementadas?¿Cuáles echas en falta? Exponga su respuesta a continuación:

Cuestiones generales.

A continuación, se le realizarán unas cuestiones sobre el uso, en general, de la aplicación.

10. ¿Intuitivamente, tuviste algún inconveniente con la navegación entre pantallas? *

Marca solo un óvalo.

- Sí
- No

11. Si es así, realice los comentarios que desee para mejorar la interacción con la aplicación.

12. ¿Durante la emulación, tuviste algún inconveniente con el rendimiento de la aplicación? *

Marca solo un óvalo.

- Sí
- No

13. Si es así, describa el problema a continuación:

14. Por último ¿Qué es lo que encuentras mejor de nuestro servicio? *

Marca solo un óvalo.

- Facilidad de uso
- Diseño de la interfaz
- Características proporcionadas
- Calidad de lanzamiento
- Otro

Tu perfil como viajero.

A continuación le realizaremos varias preguntas de carácter personal que nos ayudará a definir su perfil como viajero para después diferenciar entre las necesidades e intereses de cada una de ellos.

15. ¿Con qué frecuencia viaja al año? *

Marca solo un óvalo.

- 1 vez
- 2 veces
- 3 veces
- 4 o más veces
- Nunca

16. ¿Suele utilizar aplicaciones móviles a la hora de organizar sus viajes? *

Marca solo un óvalo.

- Si
- No

17. ¿Cuántas horas de uso le da a su móvil al día? *

Marca solo un óvalo.

- Menos de 1 hora
- Entre 1 y 3 horas
- Entre 3 y 6 horas
- 6 horas o más

Datos sociodemográficos

18. Sexo *

Marca solo un óvalo.

- Hombre
- Mujer
- Otro

19. Edad *

Marca solo un óvalo.

- Menor de 25 años
- De 25 a 34 años
- De 35 a 44 años
- De 45 a 54 años
- De 55 a 64 años
- Mayor de 65 años

20. ¿Cuál es su situación laboral? *

Marca solo un óvalo.

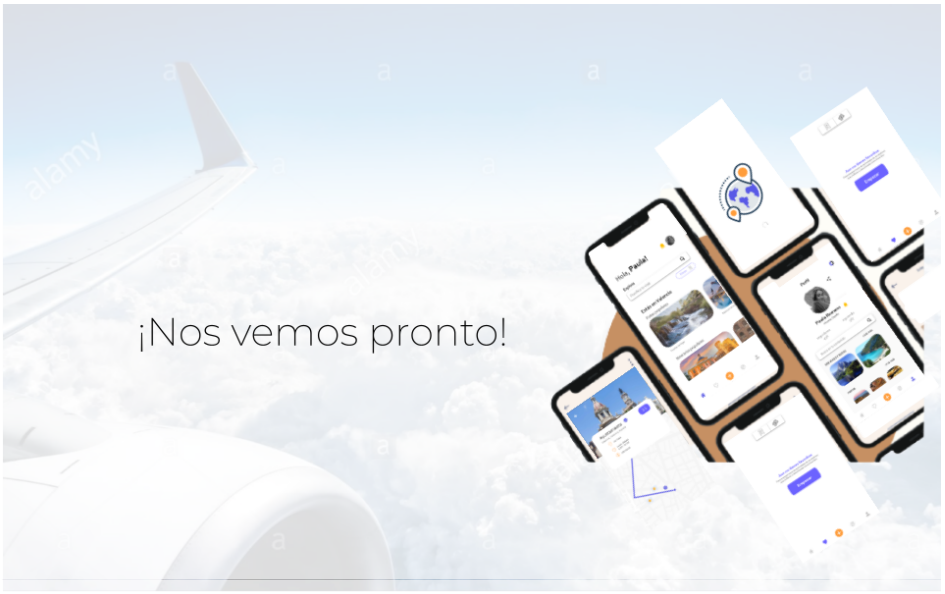
- ◯ Trabajo a tiempo completo
- ◯ Trabajo a media jornada
- ◯ Trabajador autónomo/Empresario
- ◯ Estudiante
- ◯ Estudiante y trabajador
- ◯ Jubilado/a
- ◯ Desempleado/a
- ◯ Otro: _____

21. Indica tu rango de ingresos anuales *

Marca solo un óvalo.

- ◯ Menos de 20.000 euros
- ◯ Entre 21.000 y 30.000 euros
- ◯ Entre 31.000 y 40.000 euros
- ◯ Entre 41.000 y 50.000 euros
- ◯ Más de 50.000

Gracias por tu tiempo y colaboración.

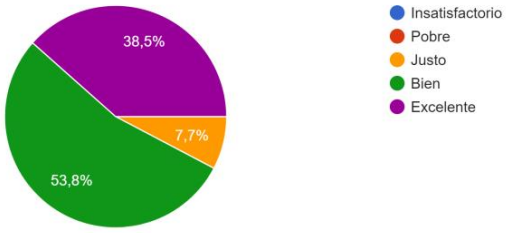


B.2 Resultados primer MVP

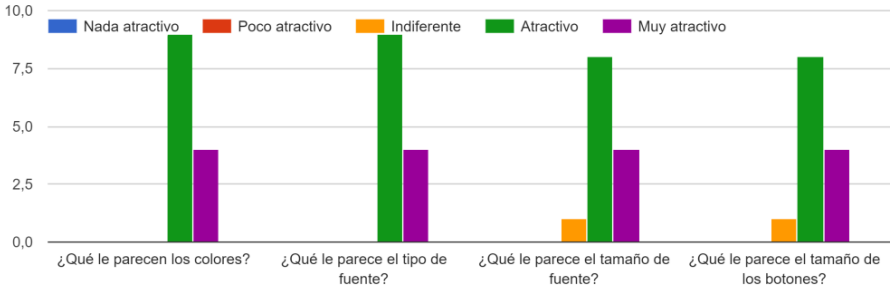
A continuación, en la siguiente página, se expone el feedback obtenido para de la primera versión presentada como resultado de las cuestiones presentadas anteriormente:

¿Cuál fue su impresión inicial del diseño?

13 respuestas

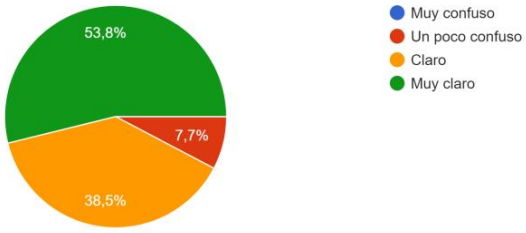


En una escala del 1 al 5, ¿En qué grado considera atractivo lo siguiente?



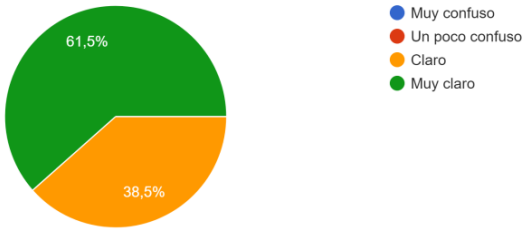
¿Cuál es tu opinión sobre la organización de la información en pantalla?

13 respuestas



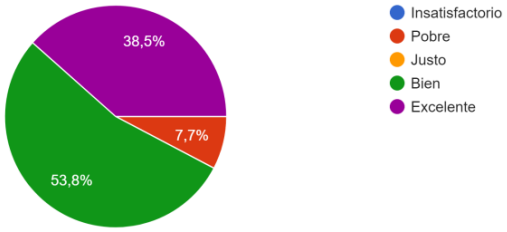
¿Consideras que los iconos usados para definir el menú representan el contenido que incluyen?

13 respuestas

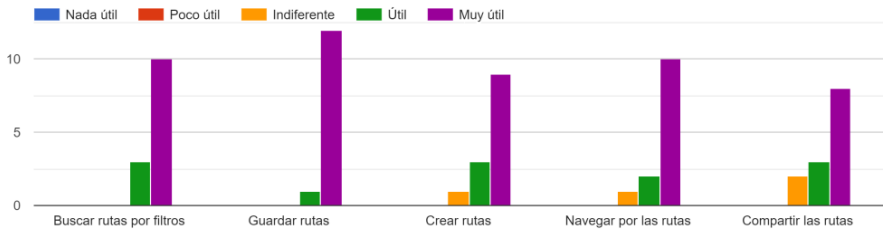


¿Cuál fue su impresión inicial sobre las funciones implementadas en la primera versión de la aplicación?

13 respuestas

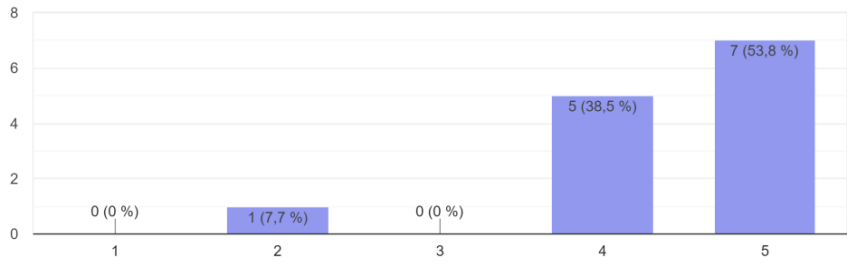


En una escala del 1 al 5, ¿En qué grado considera útil lo siguiente?



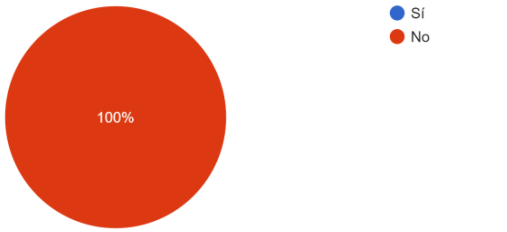
¿Qué opina sobre las funciones ofrecidas, en general, por la aplicación?

13 respuestas



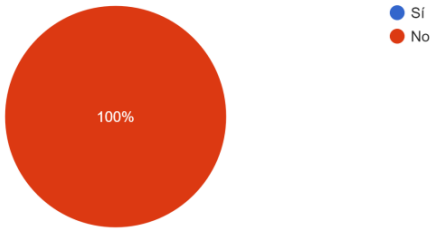
¿Intuitivamente, tuviste algún inconveniente con la navegación entre pantallas?

13 respuestas



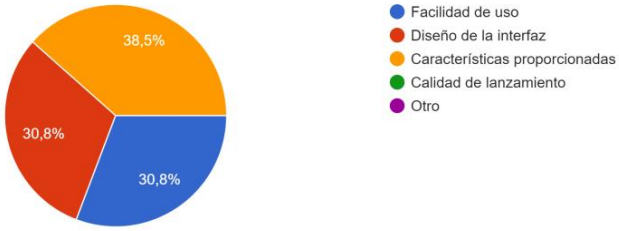
¿Durante la emulación, tuviste algún inconveniente con el rendimiento de la aplicación?

13 respuestas



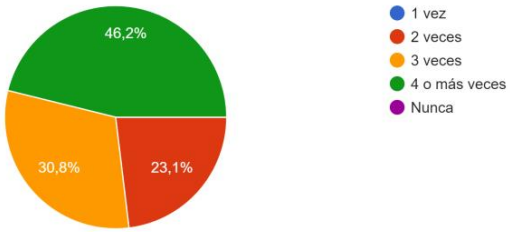
Por último ¿Qué es lo que encuentras mejor de nuestro servicio?

13 respuestas



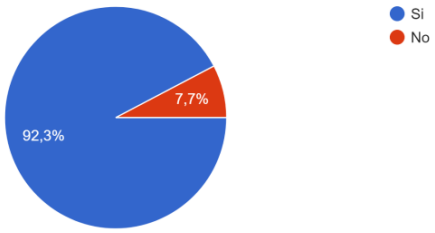
¿Con qué frecuencia viaja al año?

13 respuestas



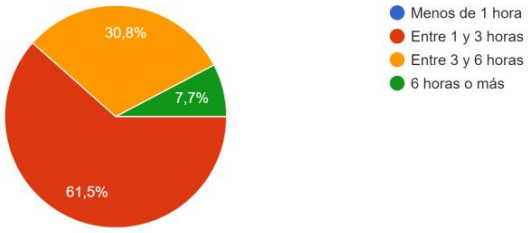
¿Suele utilizar aplicaciones móviles a la hora de organizar sus viajes?

13 respuestas



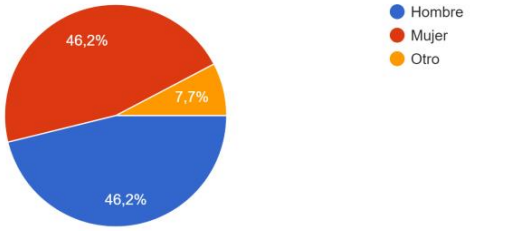
¿Cuántas horas de uso le da a su móvil al día?

13 respuestas



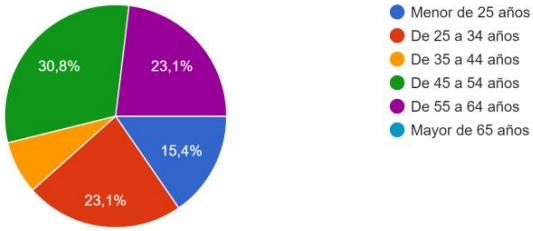
Sexo

13 respuestas



Edad

13 respuestas

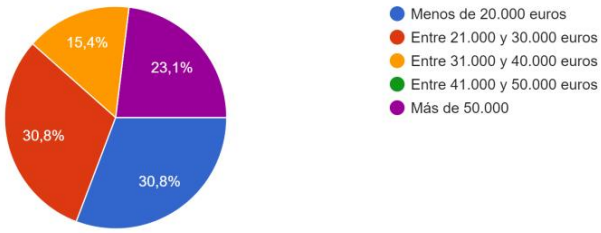


¿Cuál es su situación laboral?

13 respuestas

Indica tu rango de ingresos anuales

13 respuestas



APÉNDICE C
Formulario segundo MVP

C.1 Cuestiones segundo MVP

A continuación, en la siguiente página, se exponen las cuestiones presentadas a los usuarios para obtener feedback de la segunda versión presentada:

Formulario TFG, segundo MVP.

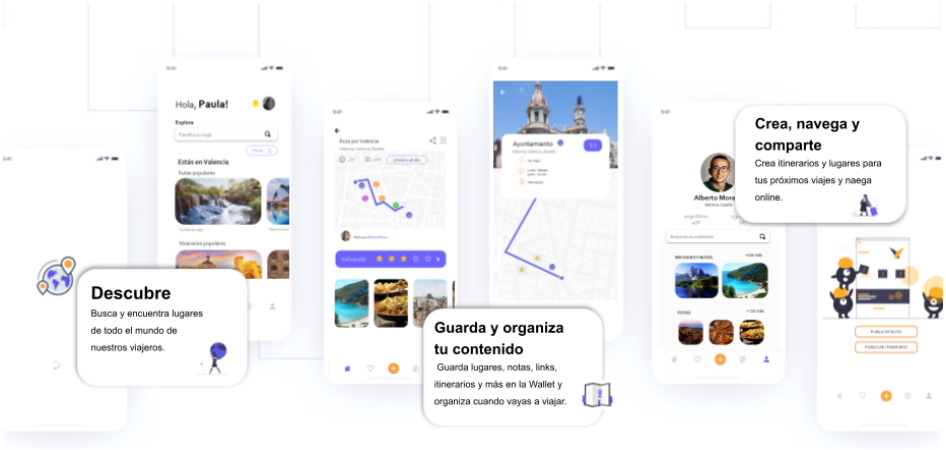
Somos Paula y Alberto, dos alumnos de la UPV que estamos desarrollando una aplicación de turismo para nuestro TFG de carácter emprendedor y hoy necesitamos tu ayuda!

A modo de de resumen, la aplicación te permite realizar rutas turísticas directamente desde tu dispositivo móvil. Podrás buscar entre multitud de rutas para diferentes ciudades Españolas, guardarlas en tu perfil, compartirlas, puntuarlas e incluso crear las tuyas propias.

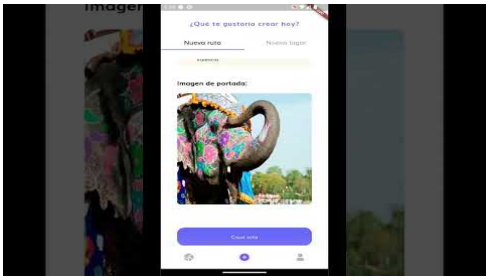
El objetivo principal de esta encuesta es valorar las funcionalidades integradas en esta primera versión de la aplicación e indicarnos que mejoras realizarías o que funciones echas en falta. (Te encontrarás ante una prueba de concepto, seguro que algo le puedes sacar 😊)

La encuesta te llevará muy poquito tiempo y nos permitirá mejorar tu futura experiencia con la aplicación! Quién sabe si estarás participando a mejorar una app que el tu del futuro podría estar utilizando... 👀

* Indica que la pregunta es obligatoria



Video Introducción segundo MVP



http://youtube.com/watch?v=NUunaAdGWJc

Diseño de la aplicación

A continuación, se le realizarán unas cuestiones sobre lo que ve en pantalla.

1. ¿Cómo calificarías el diseño visual de la aplicación en términos de atractivo y usabilidad en la segunda versión de la aplicación? *

Marca solo un óvalo.

- Excelente
- Bueno
- Aceptable
- Pobre
- Muy Pobre

2. ¿La interfaz de usuario es intuitiva y fácil de navegar? *

Marca solo un óvalo.

- Sí, muy intuitiva
- Sí, pero podría mejorarse
- Aceptable
- No muy intuitiva
- Es costosa

3. ¿El diseño de la aplicación es coherente en todas sus secciones y páginas? *

Marca solo un óvalo.

- Totalmente coherente
- Mayormente coherente
- A veces coherente
- Rara vez coherente
- No coherente en absoluto

4. ¿Encuentras que los colores y las imágenes utilizados son adecuados para una aplicación de viajes? *

Marca solo un óvalo.

- Sí, son apropiados
- Aceptables
- No son apropiados

5. ¿El diseño se adapta a diferentes tamaños de pantalla y dispositivos? *

Marca solo un óvalo.

- Sí, se adapta perfectamente
- Sí, pero podría mejorar
- No, tiene problemas de adaptación

6. ¿Tienes algún comentario o surgerencia que pueda ayudarnos a mejorar la interfaz de usuario para nuestros clientes?

Funciones de la aplicación

A continuación, se le realizarán unas cuestiones sobre lo que puede realizar en la aplicación.

7. ¿Cuál fue su impresión inicial sobre las funciones implementadas en la segunda versión de la aplicación? *

Marca solo un óvalo.

- Insatisfactorio
- Pobre
- Justo
- Bien
- Excelente

8. ¿Todas las características y funciones de la aplicación son útiles para tus necesidades al viajar? *

Marca solo un óvalo.

- Sí, todas son útiles
- La mayoría son útiles
- Algunas son útiles
- Pocas son útiles
- Ninguna es útil

9. ¿Encontraste alguna característica que no funcionara como esperabas? *

Marca solo un óvalo.

- Sí, varias características
- Sí, algunas características
- No, todas funcionaron correctamente

10. Con respecto a la pregunta anterior, ¿qué funciones no funcionan como esperabas?

11. En una escala del 1 al 5, ¿En qué grado considera útil lo siguiente? *

Marca solo un óvalo por fila.

	Nada útil	Poco útil	Indiferente	Útil	Muy útil
Buscar rutas por filtros					
Guardar rutas					
Crear rutas					
Navegar por las rutas					
Compartir las rutas					

12. ¿Qué características adicionales te gustaría ver en la aplicación que no estén disponibles actualmente?

Cuestiones generales.

A continuación, se le realizarán unas cuestiones sobre el uso, en general, de la aplicación.

13. ¿Has experimentado bloqueos o cierres inesperados de la aplicación durante su uso? *

Marca solo un óvalo.

- ◯ No, nunca
- ◯ Sí, raramente
- ◯ Sí, con frecuencia

14. Si es así, realice los comentarios que desee para mejorar la interacción con la aplicación.

_____

15. ¿La aplicación se ejecuta sin problemas y sin retrasos notables? *

Marca solo un óvalo.

- ◯ Sí, sin problemas
- ◯ Sí, pero con pequeños retrasos ocasionales
- ◯ Aceptable, con algunos retrasos notables
- ◯ No, con retrasos significativos

16. Si es así, describa el problema a continuación:

_____

17. ¿Cómo calificarías la velocidad de carga de la aplicación? *

Marca solo un óvalo.

- Muy rápida
- Rápida
- Aceptable
- Lenta
- Muy lenta

18. Por último ¿Qué es lo que encuentras mejor de nuestro servicio? *

Marca solo un óvalo.

- Facilidad de uso
- Diseño de la interfaz
- Características proporcionadas
- Calidad de lanzamiento
- Otro

Tu perfil como viajero.

A continuación le realizaremos varias preguntas de carácter personal que nos ayudará a definir su perfil como viajero para después diferenciar entre las necesidades e intereses de cada una de ellos.

19. ¿Con qué frecuencia viaja al año? *

Marca solo un óvalo.

- 1 vez
- 2 veces
- 3 veces
- 4 o más veces
- Nunca

20. ¿Suele utilizar aplicaciones móviles a la hora de organizar sus viajes? *

Marca solo un óvalo.

- Si
- No

21. ¿Cuántas horas de uso le da a su móvil al día? *

Marca solo un óvalo.

- Menos de 1 hora
- Entre 1 y 3 horas
- Entre 3 y 6 horas
- 6 horas o más

Datos sociodemográficos

22. Sexo *

Marca solo un óvalo.

- Hombre
- Mujer
- Otro

23. Edad *

Marca solo un óvalo.

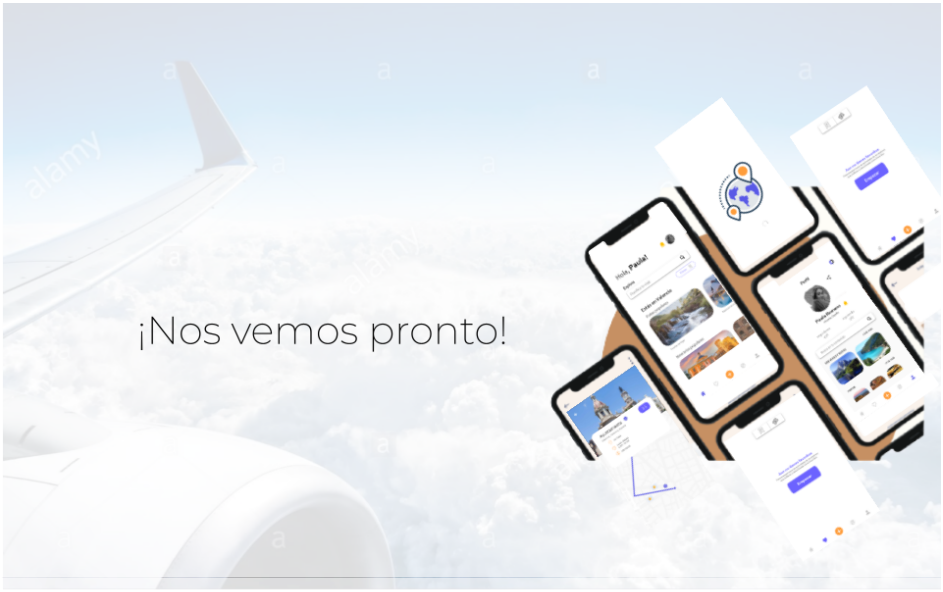
- Menor de 25 años
- De 25 a 34 años
- De 35 a 44 años
- De 45 a 54 años
- De 55 a 64 años
- Mayor de 65 años

24. ¿Cuál es su situación laboral? *

Marca solo un óvalo.

- ◯ Trabajo a tiempo completo
- ◯ Trabajo a media jornada
- ◯ Trabajador autónomo/Empresario
- ◯ Estudiante
- ◯ Estudiante y trabajador
- ◯ Jubilado/a
- ◯ Desempleado/a
- ◯ Otro: _____

Gracias por tu tiempo y colaboración.

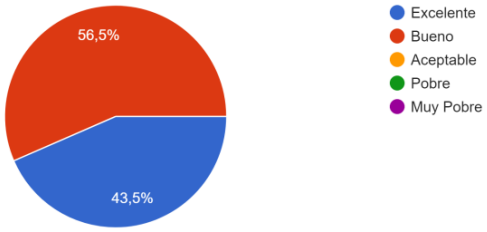


C.2 Resultados segundo MVP

A continuación, en la siguiente página, se expone el feedback obtenido para de la primera versión presentada como resultado de las cuestiones presentadas anteriormente:

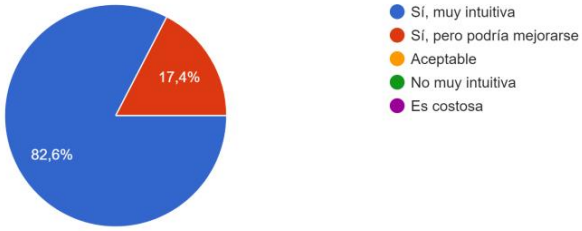
¿Cómo calificarías el diseño visual de la aplicación en términos de atractivo y usabilidad en la segunda versión de la aplicación?

23 respuestas



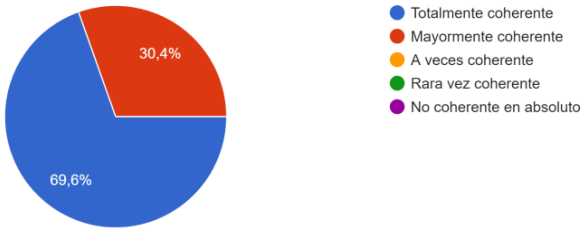
¿La interfaz de usuario es intuitiva y fácil de navegar?

23 respuestas



¿El diseño de la aplicación es coherente en todas sus secciones y páginas?

23 respuestas

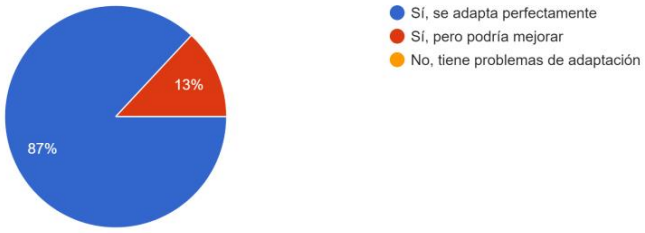


¿Encuentras que los colores y las imágenes utilizados son adecuados para una aplicación de viajes?

23 respuestas

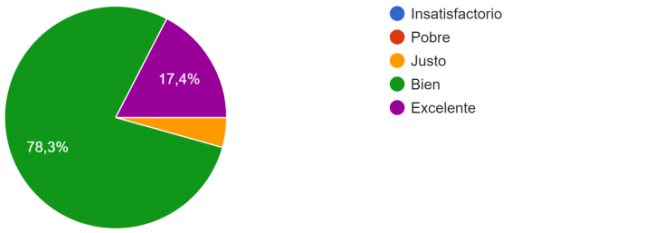
¿El diseño se adapta a diferentes tamaños de pantalla y dispositivos?

23 respuestas



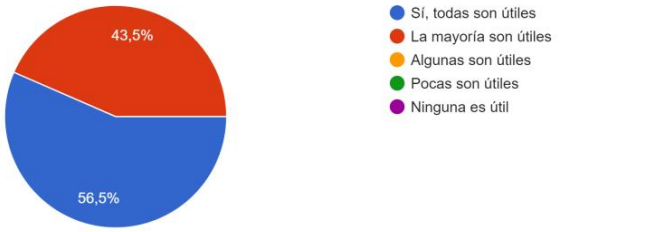
¿Cuál fue su impresión inicial sobre las funciones implementadas en la segunda versión de la aplicación?

23 respuestas



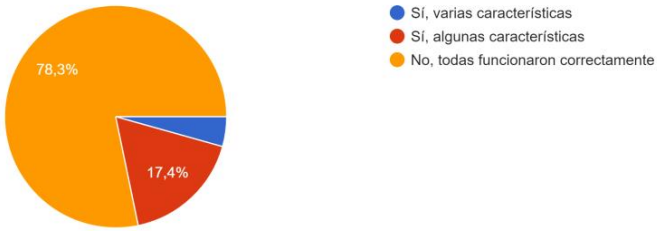
¿Todas las características y funciones de la aplicación son útiles para tus necesidades al viajar?

23 respuestas

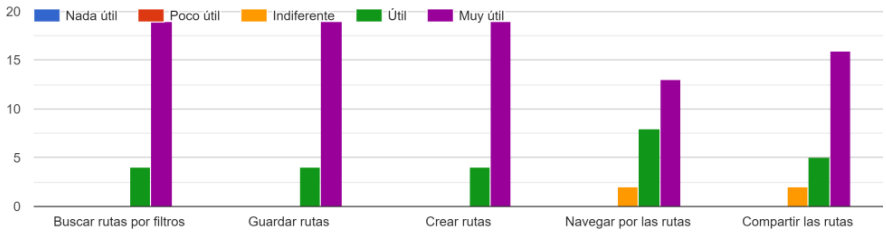


¿Encontraste alguna característica que no funcionara como esperabas?

23 respuestas

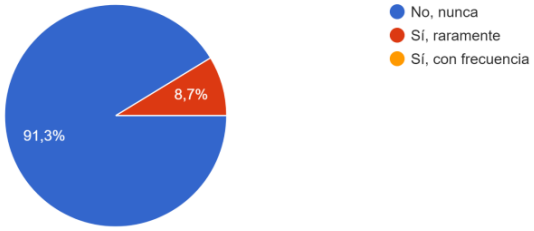


En una escala del 1 al 5, ¿En qué grado considera útil lo siguiente?



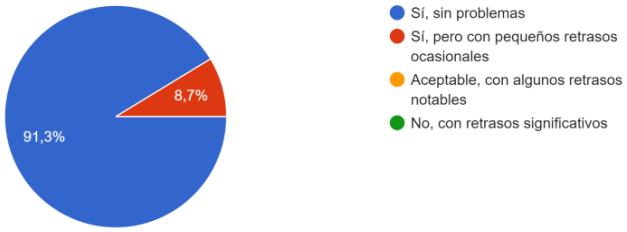
¿Has experimentado bloqueos o cierres inesperados de la aplicación durante su uso?

23 respuestas



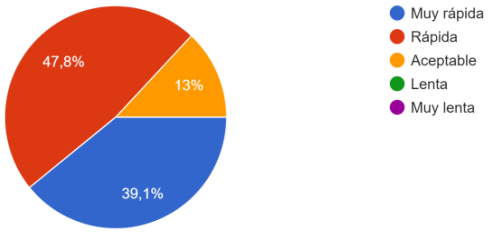
¿La aplicación se ejecuta sin problemas y sin retrasos notables?

23 respuestas



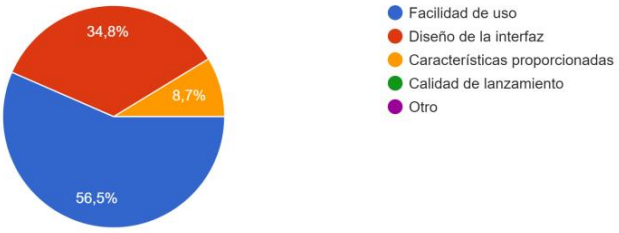
¿Cómo calificarías la velocidad de carga de la aplicación?

23 respuestas



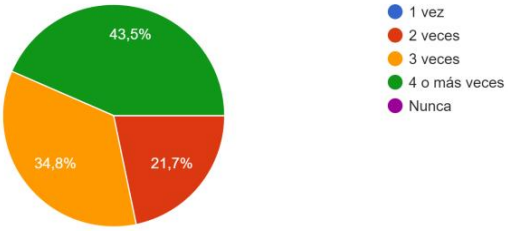
Por último ¿Qué es lo que encuentras mejor de nuestro servicio?

23 respuestas



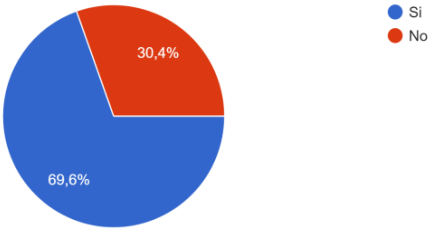
¿Con qué frecuencia viaja al año?

23 respuestas



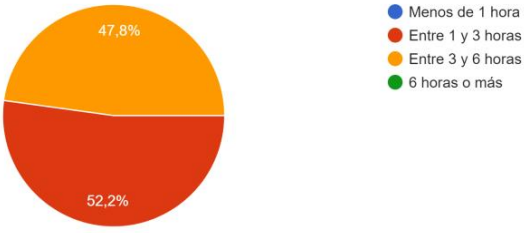
¿Suele utilizar aplicaciones móviles a la hora de organizar sus viajes?

23 respuestas



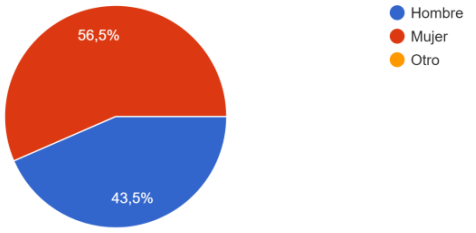
¿Cuántas horas de uso le da a su móvil al día?

23 respuestas



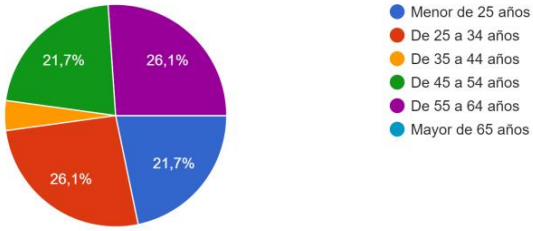
Sexo

23 respuestas



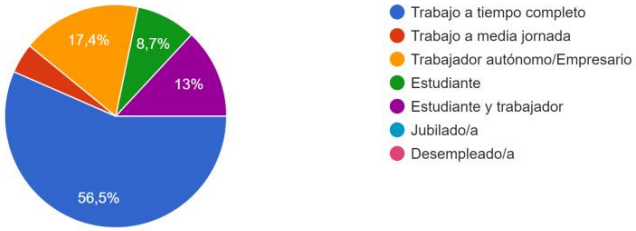
Edad

23 respuestas



¿Cuál es su situación laboral?

23 respuestas



APÉNDICE D
Objetivos de desarrollo sostenible

D.1 Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

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 infraestructuras.**		x		
ODS 10. **Reducción de las desigualdades.**			x	
ODS 11. **Ciudades y comunidades sostenibles.**			x	
ODS 12. **Producción y consumo responsables.**		x		
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**		x		

D.2 Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados

La herramienta que se propone desarrollar para este TFG supondrá una mejora económica para el sector del turismo, al proveer un servicio propicio para el sector, una mayor difusión e integración de los conocimientos y la cultura Europea, al difundir información mediante el turismo, una reducción de las desigualdades, al otorgar un entorno de fácil acceso al sector para la competencia, que se traducirá en la reducción de costes y la mejora de la calidad de los productos para el consumidor, y una mejora para los ecosistemas al reducir el consumo de materias al usar el formato digital. La innovación en cualquier sector económico mejora las posibilidades del mercado y conlleva una mejora de la calidad de vida de las personas donde esta se desarrolla. La libertad de difusión de contenido educacional, informativo y didáctico sobre el territorio europeo ayuda a las personas a tener una visión mas amplia de la sociedad, la economía, la política y la vida, creando personas con capacidad crítica y capaces de generar nuevas ideas y retos que acabarán elevando la calidad de vida dentro y, en algún momento, fuera de nuestro planeta. Gracias a la tecnología, los procesos se especializan, se crean nuevas oportunidades y se consigue vivir en un mundo globalizado en el que algún día, gracias al libre mercado y la capitalización del ahorro, conseguiremos esa igualdad de oportunidades mundial que tanto ansiamos. Es por ello, que cualquier innovación y modelo de negocio, como el que se desarrolla en este proyecto, es una gran herramienta que ayudará a muchas personas a conseguir sus metas y objetivos, y, en algún momento, otorgar ellos las mismas posibilidades a otras. No existe ningún mejor modelo de crecimiento económico y de calidad de vida de las personas que aquel que les otorga a estas la capacidad de emprender sus propios proyectos y desarrollar sus propias ideas, y, en este caso, con esta aplicación es lo que buscamos, dar la oportunidad a todos esos guías, viajeros y conocedores de su entorno la posibilidad de capitalizar sus conocimientos a la vez que enseñan a otros todos lo que el mundo que tienen a su alrededor les pueda ofrecer. Aprender la historia y acontecimientos de la era de la humanidad son los que originan una sociedad educada encaminada en la búsqueda de la mejora de su civilización. Es importante conocer el pasado para crear un futuro mejor en el que no se repitan aquellos errores que no nos permiten avanzar y que perjudicarían a las ODS que aquí buscamos. Avanzar hacia un modelo en el uso de la lógica, la racionalidad y, sobre todo, basado en los pilares de la realidad que nos precede. En este TFG no se desarrolla solo una idea, se presenta Waltrip, un modelo de negocio con el que se difundirá infinidad de conocimientos gracias al propio funcionamiento del mercado y la economía, los mejores inventos de nuestra sociedad, y gracias a los cuales la humanidad a conseguido evolucionar hasta llegar al nivel de vida en el que nos encontramos.