DESARROLLO DE UN SERVICIO DE VALOR AÑADIDO PARA CREAR ACTIVIDADES DE OCIO USANDO REPOSITORIOS DE DATOS ABIERTOS EN LA CIUDAD DE MADRID

DEVELOPMENT OF A VALUE-ADDED SERVICE TO CREATE LEISURE ACTIVITIES USING OPEN DATA REPOSITORIES IN THE CITY OF MADRID

Juan Antonio Ávila Catalán

Alberto Fernández-Baíllo Rodríguez

Íñigo García-Conde Trueba

Cristo Fernando López Cabañas

GRADO EN INGENIERÍA DE SOFTWARE

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

CURSO 2019-2020

DIRECTOR

Antonio Sarasa Cabezuelo

AGRADECIMIENTOS

A nuestras familias y nuestras novias por hacer de estos meses de confinamiento algo inolvidable.

RESUMEN

En este trabajo se recoge la especificación, el diseño y la implementación de una aplicación web de valor añadido. Se recogen todos los componentes del sistema, desde los clientes y servicios hasta los sistemas que se acoplan a la aplicación. También se incluyen gráficos, tablas e imágenes que ayudan a la comprensión de este.

La aplicación proporciona al usuario una plataforma donde podrá escoger entre distintos planes que se encuentran disponibles en la ciudad de Madrid, crear y combinar conjuntos de planes para realizarlos de una manera y en una fecha concreta y obtener recomendaciones de la aplicación en función de distintos factores tales como el tiempo meteorológico o la afluencia de personas entre otros. Estos podrán ser eventuales o continuados y contarán con comentarios y valoraciones de los usuarios.

Los datos utilizados en la aplicación se extraen del banco de datos abiertos del Ayuntamiento de Madrid y de distintas fuentes que se irán describiendo en el trabajo. Entre los datos se encuentra información como los horarios, localización, imágenes y descripción sobre los distintos eventos.

PALABRAS CLAVE

Madrid, planes, itinerarios, RabbitMQ, API, Android, Spring

ABSTRACT

This work includes the specification, design and implementation of a value-added web application. All system components are described, from clients and services to systems that fit the application. Also graphs, tables and images are included to help to understand it better.

The application provides the user a platform where they can choose between different plans that are available in the city of Madrid, create and combine sets of plans to perform them in a specific way and on a specific date and obtain recommendations of the application based on different factors such as weather or the influx of people among others. The plans can be eventual or continued and will have comments and ratings from users.

The data used in the application is extracted from the open data bank of the town hall of Madrid and from different sources that will be described in the work. Among the data is information such as schedules, location, images and description about the different events.

KEYWORDS

Madrid, plans, itineraries, RabbitMQ, API, Android, Spring

ÍNDICE

ÍNDICE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1. Introducción

En este primer capítulo de la memoria se explicará cual es la motivación que ha llevado a realizar este trabajo, los objetivos que se pretenden alcanzar y como está estructurada la memoria.

1.1. Motivación

El turismo en España es uno de los principales motores de la economía. La tradición, la historia y la cultura, atraen a personas de dentro y fuera de España a visitar las distintas ciudades de este país. Es por eso por lo que el acceso a la información de calidad ha de estar al alcance de todos, de manera sencilla, intuitiva y gratuita.

Existen una gran variedad de servicios que disponen de este tipo de información, sin embargo, estos servicios están especializados en un tipo concreto, como por ejemplo locales de moda, restaurantes de calidad, etc.

Estos servicios ofrecen algunas limitaciones con respecto de este proyecto, pues no se recoge información de todos los lugares turísticos que aparecen, no hay posibilidad de crear un itinerario personalizado englobando varias actividades de las ya existentes, o no cuentan con muchas opciones de filtrado de actividades.

Para resolver estas limitaciones, en este trabajo se ha planteado el desarrollo de una plataforma en la que se pueda disponer de toda esta información, desde museos y restaurantes, hasta monumentos y obras de teatro, toda la información disponible sobre la ciudad de Madrid a nivel turístico unificada en un único servicio. Más adelante, esto podría extenderse a otras ciudades, siempre que sea posible la explotación de fuentes de datos abiertos de la ciudad correspondiente para ofrecer al turista información en forma de servicios de valor añadido.

Mediante la explotación de la base de datos pública del ayuntamiento de Madrid, será posible disponer de toda esta información completa de los lugares de la ciudad y, además, es posible aportar un valor añadido con servicios, que, en función de determinados factores, fuesen capaces de realizar una recomendación sobre qué tipo de actividades turísticas deben realizarse en un momento determinado de tiempo y así poder maximizar la experiencia del turista.

1.2. Objetivos

El objetivo principal del trabajo es la implementación de una aplicación web que permita a sus usuarios organizar planes y experiencias en la ciudad de Madrid. Para ello la aplicación contará con un sistema de recomendación en función de distintos factores.

A continuación, se refina el objetivo principal en los siguientes objetivos específicos:

- Desarrollar una aplicación web que permita a sus usuarios organizar planes en la ciudad de Madrid, organizarlos en conjuntos, categorizarlos y compartirlos con otros usuarios.

- Desarrollar una función que permita recomendar al usuario los planes óptimos en función del tiempo meteorológico, la afluencia y la disponibilidad horaria.

- Implementar una API REST para facilitar el acceso a los servicios que ofrece la aplicación.

- Diseñar e implementar un sistema para la recolección de datos que permita mantener los servicios actualizados.

1.3. Estructura de la memoria

A continuación, se describe de manera breve la estructura de la memoria.

- Capítulo 1: En este capítulo se describe la motivación del trabajo, los objetivos y la estructura de la memoria.

- Capítulo 2: En este capítulo se estudian herramientas similares a la que se ha realizado en el trabajo.

- Capítulo 3: En este capítulo se describe la tecnología utilizada para implementar el proyecto.

- Capítulo 4: En este capítulo se definen los actores y casos de uso que se explicarán mediante tablas junto a sus requisitos.

- Capítulo 5: En este capítulo se explicará el diagrama entidad relación en el que se ha basado el proyecto y la implementación de la base de datos.

- Capítulo 6: En este capítulo se explicará la arquitectura de la aplicación y los patrones utilizados.

- Capítulo 7: En este capítulo se realizará un estudio sobre el diseño e implementación de los casos de uso más relevantes.

- Capítulo 8: En este capítulo se expondrá las estadísticas aportadas por usuarios que han dado su opinión sobre el producto.

- Capítulo 9: En este capítulo se listará casos de usos que se harán en un futuro explicándolos brevemente.

- Capítulo 10: En este capítulo se expone el trabajo realizado por cada uno de los autores.

- Anexo I: Guía instalación.

- Anexo II: Manual de usuario.

Chapter 1. Introduction

In this first chapter of the report, it will be explained which is the motivation that has led to carry out this work, the objectives to be achieved and how the report is structured.

1.1. Motivation

Tourism in Spain is one of the main engines of the economy. Tradition, history and culture, attract people from inside and outside Spain to visit the different cities of this country. That is why access to quality information must be within everyone's reach, in a simple, intuitive and free manner.

There is a great variety of services that have this type of information, however, these services are specialized in a specific type, such as fashionable places, quality restaurants, etc.

These services offer some limitations with respect to this project, since information is not collected from all the tourist sites that appear, there is no possibility of creating a personalized itinerary encompassing several activities of those already existing, or they do not have many options for filtering activities.

In order to solve these limitations, in this work we have proposed the development of a platform where all this information, from museums and restaurants, to monuments and plays, can be made available, all the information available about the city of Madrid at a tourist level unified in a single service. Later on, this could be extended to other cities, whenever it is possible to exploit open data sources of the corresponding city to offer the tourist information in the form of added value services.

By exploiting the public database of the Madrid City Council, it will be possible to have all this complete information on the places in the city and, in addition, it is possible to provide added value with services, which, depending on certain factors, would be able to make a recommendation on what type of tourist activities should be carried out at a given moment in time and thus be able to maximize the tourist's experience.

1.2. Objetives

The main objective of the work is the implementation of a web application that allows its users to organize plans and experiences in the city of Madrid, for which the application will have a recommendation system based on different factors. The main objectives of the work are described below:

- Develop a web application that allows its users to organize plans in the city of Madrid, organize them into sets, categorize them and share them with other users. In addition, the application should recommend the user the most optimal plans according to weather, traffic, travel time and time used to complete each of the plans, if several plans are grouped into one set, the application will recommend the user the best way to do it.

- The implementation of a REST API for easy access to the services offered by the application.
- Design and implement a system for data collection to keep services up to date.
- Design and implement an algorithm for the application's recommendation system.

1.3. Memory structure

The structure of the memory is briefly described below.

- Chapter 1: This chapter describes the motivation for the work, the objectives and the structure of the report.

- Chapter 2: This chapter looks at tools similar to the one in the paper.

- Chapter 3: This chapter describes the technology used to implement the project.

- Chapter 4: This chapter defines the actors and use cases that will be explained by means of tables together with their requirements.

- Chapter 5: This chapter will explain the relationship entity diagram on which the project has been based and the database implementation.

- Chapter 6: This chapter will explain the application architecture and the patterns used.

- Chapter 7: In this chapter a study will be made on the design and implementation of the most relevant use cases.

- Chapter 8: This chapter will present the statistics provided by users who have given their opinion about the product.

- Chapter 9: This chapter will list cases of future uses by briefly explaining them.

- Chapter 10: This chapter presents the work done by each of the authors.

- Annex I: Installation guide.

- Annex II: User’s manual.

Capítulo 2. Estado del arte.

En este capítulo, se describirán las características principales de algunas aplicaciones similares a la desarrollada en este proyecto.

La aplicación está orientada a la elaboración de itinerarios personalizados, en base a los lugares turísticos en una ciudad. Estos sitios, son actividades que se pueden hacer o lugares que se pueden visitar en la ciudad de Madrid, que van acompañados de una información acerca de estos, como sus horarios, si son al aire libre o en interior, una descripción, etc.

Los itinerarios serán programados con estos planes disponibles para realizarse en una franja horaria del día, mostrando al usuario aquellos recomendados en base a diversos factores como las condiciones meteorológicas, la afluencia de personas o los horarios de apertura.

En este ámbito, se listan aplicaciones agrupadas en diferentes categorías, las más populares son:

- **Civitatis** [1]: Es una aplicación que se caracteriza por ofrecer una gran cantidad de excursiones y planes por todo el mundo, así como guías turísticas. Cuenta tanto con una aplicación para dispositivos móviles iOS y Android, y con un cliente web para navegadores.

  Sus principales características son:

  - Posibilidad de reserva y pago desde la aplicación.
  - Elección de excursiones ya elaboradas en diferentes ubicaciones. (Figura 1)
  - Enlaces a páginas externas con más información.

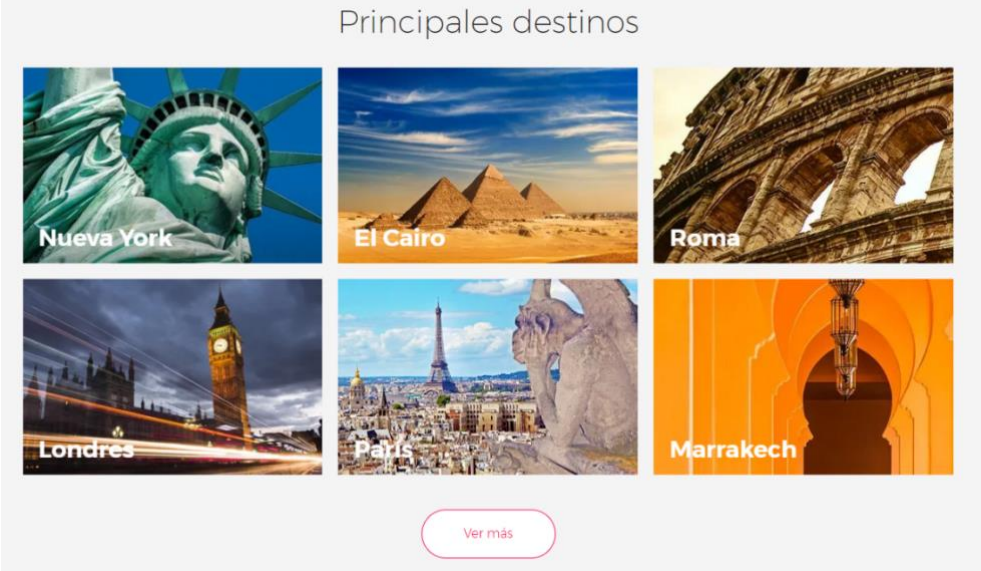


Figura 1 - Diferentes ubicaciones que ofrece la aplicación web de Civitatis donde poder buscar actividades

- **TripAdvisor** [2]: Es un servicio que se centra en ofrecer un servicio de reservas de hoteles o restaurantes para planificar excursiones. Cuenta con guías turísticas y una

gran cantidad de reseñas sobre los hoteles y restaurantes que se ofrecen. Tiene aplicación para dispositivos móviles iOS y Android, además de cliente web.

Sus principales características son:

- Posibilidad de reserva desde la aplicación (pero no todas las actividades).
- Posibilidad de contacto con la entidad responsable de cada lugar turístico.
- Cuenta con una gran cantidad de categorías para añadir precisión a las búsquedas.
- Tiene un catálogo de actividades muy extenso. (Figura 2)

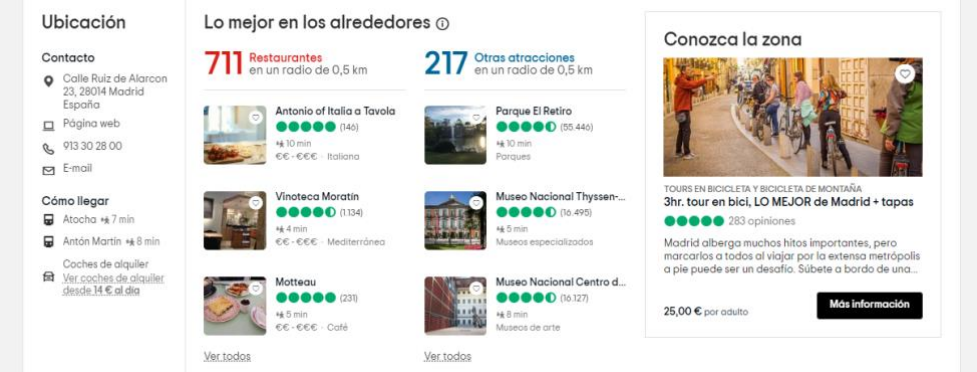


Figura 2 - Gran cantidad de lugares turísticos

- **Booking** [3]: Es una aplicación que ofrece un servicio extenso para realizar reservas en hoteles, facilita opciones de desplazamiento y propone planes en muchos países del mundo. Cuenta con clientes iOS, Android y web.

Sus principales características son:

- Posibilidad de elección de múltiples lugares turísticos
- Permite realizar reservas en la aplicación
- Tiene promociones y descuentos (Figura 3)

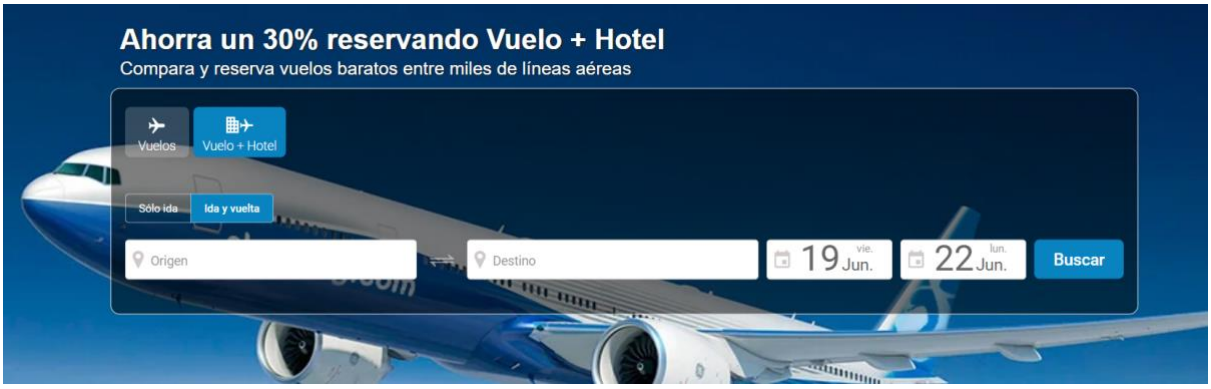


Figura 3 - Promociones

- **Minube** [4]: Es una red social que pone en contacto a usuarios para que estos escriban reseñas acerca de los lugares turísticos que conozcan, y así otros puedan planificar

excursiones. Dispone de clientes tanto web, como para dispositivos móviles (iOS y Android)

Sus principales características son:

Gran cantidad de recomendaciones actuales (Figura 4 - Recomendaciones Minube

-).
- Ofrece muchas categorías
- Posibilidad de programar viajes
- Tiene un buscador de vuelos

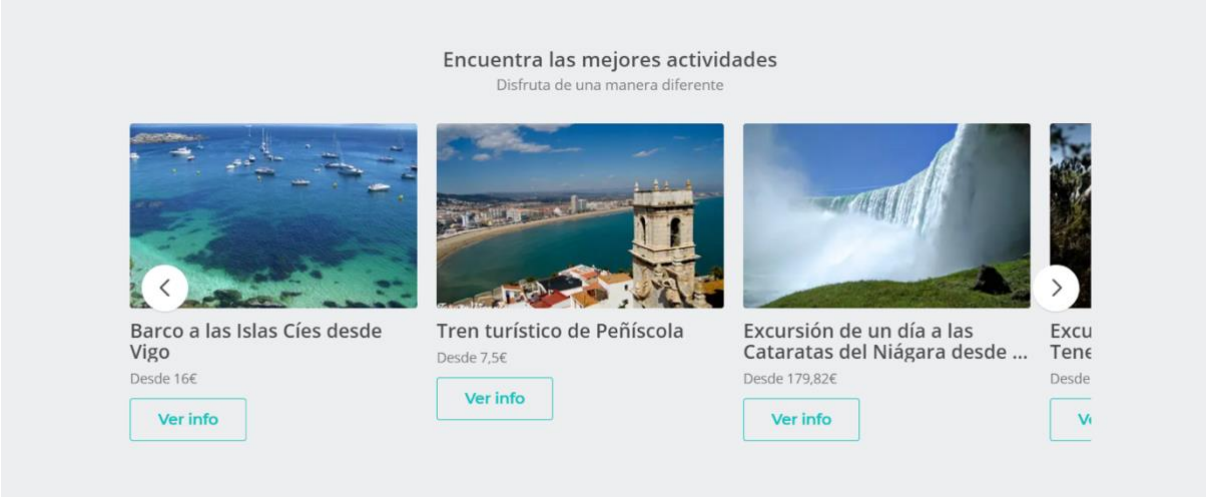


Figura 4 - Recomendaciones Minube

Capítulo 3. Tecnología empleada.

En este capítulo se van a comentar las tecnologías utilizadas para el desarrollo del proyecto.

3.1. Herramientas parte cliente

3.1.1 Android Studio

Android Studio [5] es el entorno de desarrollo integrado oficial establecido por Google para la plataforma Android. Está desarrollado por JetBrains en colaboración con Google y está basado en IntelliJ IDEA [10].

Este entorno de desarrollo incorpora características avanzadas para la refactorización de código a través de herramientas Lint [64], refactorizaciones específicas para la plataforma Android y asistencia al desarrollador a través del autocompletado de código.

Entre las herramientas que integra, se puede destacar la emulación de dispositivos móviles, que facilita la realización de pruebas en un entorno lo más parecido al de un dispositivo real. Además, dispone de un editor de interfaces de usuario que permite elaborar los diseños arrastrando los componentes gráficos.

3.2 Herramientas parte servidor

3.2.1. Spring Boot

Spring Boot [6] es un marco de trabajo utilizado para el desarrollo de aplicaciones. Es de código abierto y desarrollado en Java. Además, es compatible con los lenguajes que funcionan a través de la máquina virtual de Java, por ejemplo, Kotlin.

Constituye una alternativa al desarrollo de aplicaciones web en Java mediante JavaEE [11], ya que Spring Boot no requiere de un contenedor de aplicaciones, sino que es posible desplegar la aplicación a través de la ejecución de un archivo JAR.

Está dividido en distintos módulos: Spring Web, Spring Security y Spring Cloud son algunos de ellos. Todos estos integran funcionalidades específicas para el desarrollo de aplicaciones web.

Por último, cabe destacar que, en cuanto a las propiedades y configuraciones, estas pueden ser establecidas a través de anotaciones en el código. En este sentido, no requiere de archivos adicionales de configuración. Además, pueden establecerse propiedades concretas a cada entorno.

3.2.2 RabbitMQ

RabbitMQ [7] es un *broker* de mensajería de código abierto. Implementa el estándar *Advanced Message Queuing Protocol* (AMQP).

Permite establecer colas de mensajes a las que suscribir clientes software escritos en cualquier lenguaje de programación que disponga de una librería sobre el protocolo AMQP.

Además, dispone de herramientas de administración para la gestión de los perfiles de acceso, colas, suscriptores y demás elementos del sistema a través de una interfaz web o de interfaz de línea de comandos.

3.2.3 MariaDB

MariaDB [8] es un sistema de gestión de bases de datos relacionales, derivado de MySQL. Está disponible para múltiples sistemas operativos, como Windows, Linux o Mac.

Soporta todos los niveles de aislamiento, además de procedimientos almacenados y disparadores. Utiliza la sintaxis SQL para la realización de consultas.

Dispone de librerías de conexión para múltiples lenguajes de programación, entre ellos Java.

3.2.4 IntelliJ IDEA

IntelliJ IDEA [10] es un entorno de desarrollo integrado dedicado a la codificación de aplicaciones y programas en Java desarrollado por JetBrains.

Está disponible en dos versiones diferentes, una de ellas de código abierto. Aunque está especializado en Java, dispone de componentes instalables de forma separada para la integración de otros lenguajes.

Entre sus características destaca la asistencia en configuración y autocompletado de código específica para marcos de trabajo como Spring o Hibernate. Además, integra en el propio entorno el control de versiones mediante Git o SVN y el acceso a base de datos.

3.2.5 jOOQ

jOOQ [12] es una librería de mapeo objeto-relacional escrita en Java. Está disponible en forma de licencia de libre uso no comercial y también a través de licencia de pago.

Provee una API para la generación de código SQL a través de código Java y un módulo de ingeniería inversa para la generación de clases a partir de un esquema de base de datos relacional.

Su característica principal es la codificación de sentencias SQL a través de un lenguaje específico de dominio, que provee tipos y advertencia de errores en la sintaxis SQL en tiempo de compilación.

También destaca por la adaptación de funciones no disponibles en el estándar SQL a cada motor de base de datos, proporcionando una capa de compatibilidad entre distintos fabricantes.

3.2.6 JPA

JPA [54] es una API de persistencia propia del lenguaje de programación Java. Permite interactuar con bases de datos a través de objetos, abstrayendo al desarrollador del lenguaje de consulta propio de la base de datos. Existen distintas implementaciones de este estándar, entre las que destacan EclipseLink [65] e Hibernate [66].

Mediante la anotación de las clases Java que representan cada una de las tablas de la base de datos, es posible generar el esquema de tablas sin necesidad de intervención manual.

Además, integra funcionalidades para la gestión de la concurrencia entre distintos procesos, el tratamiento en masa de datos y la gestión de transacciones.

3.3 Otras herramientas

3.3.1. Git

Git [13] es un software que se utiliza para el control de versiones con el fin de facilitar el desarrollo en paralelo y poder hacerlo de manera coordinada, pudiendo así trabajar varias personas sobre un mismo código y desarrollando distintas funcionalidades en paralelo.

Entre las principales ventajas de Git destacan la descentralización, que hace posible el control de versiones en la máquina local del desarrollador y su naturaleza multiplataforma.

3.3.2 Kotlin

Kotlin [14] es un lenguaje de programación fuertemente tipado y orientado a objetos creado por JetBrains y que funciona sobre la máquina virtual de Java.

Se ha convertido en el lenguaje oficial para desarrollo Android, pero se puede utilizar en cualquier plataforma. Provee seguridad ante punteros nulos e incorpora funciones de orden superior, con características propias de un lenguaje funcional.

Es interoperable con Java, de tal forma que pueden escribirse programas que combinen código que interactúe entre los dos lenguajes. Además, es posible transpilar el código escrito en Kotlin a Javascript. Por otro lado, incluye en su API funciones específicas para el desarrollo de interfaces web.

3.3.3 Docker

Docker [44] es una herramienta para el despliegue de aplicaciones mediante contenedores software. Está disponible para Mac, Windows y Linux.

A través de la generación de imágenes software, es posible distribuir distintos sistemas interconectados de tal forma que no sea necesario instalarlos de forma independiente, si no que tan sólo se requiere ejecutar la imagen en un contenedor.

Los contenedores que ejecuta Docker trabajan sobre Linux y cada uno de ellos se encuentra aislado del resto, de tal forma que por defecto forman parte de procesos aislados.

Capítulo 4. Casos de uso

En este capítulo, se describen los casos de uso definidos para el desarrollo de la aplicación, así como los actores del sistema. En la primera sección se presentarán los actores y en la segunda sección se presentarán los casos de uso agrupados por cada uno de los actores del sistema.

4.1. Actores del sistema

A continuación, se describen los distintos actores que podrán utilizar el sistema.

- **Usuario no registrado**

  Representan a los usuarios que podrán acceder a la aplicación sin necesidad de iniciar sesión y tiene asignadas las funciones de ver los planes e itinerarios disponibles, compartirlos a través de aplicaciones externas, consultar sus detalles y realizar búsquedas en función de distintos criterios.

- **Usuarios registrados**

  Representan a los usuarios registraos, los cuales podrán acceder a todas las funciones disponibles para los usuarios sin registrar más la posibilidad de crear itinerarios, guardarlos y valorarlos, así como, valorar los planes. El registro en la aplicación será gratuito.

- **Administrador**

  Representa al encargado de la aplicación y tendrá distintas funciones relacionadas con la actualización y revisión de los datos de la aplicación.

4.2. Modulo usuario no registrado

En esta sección, se muestran los casos de uso asociados al usuario no registrado. En la Figura 5 se puede observar el diagrama de casos de uso.

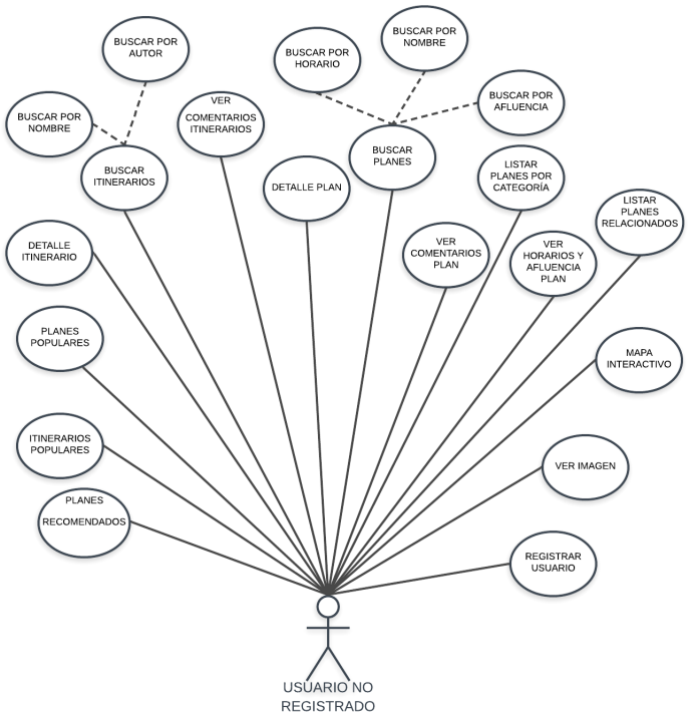


Figura 5 - Diagrama casos de uso módulo usuario no registrado

IT-01	Listar itinerarios populares	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	NA	
Descripción	El usuario podrá visualizar los itinerarios con mejor puntuación para saber qué planes son los que más valoran otros usuarios	
Entrada	Página, tamaño página, idioma	
Salida	Lista de itinerarios	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El sistema muestra una lista reducida de los itinerarios más populares
	3	El usuario pulsa sobre un botón para ver más itinerarios populares
	4	El sistema muestra todos los itinerarios ordenados de mayor a menor popularidad
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	Un itinerario es popular si tiene una valoración mayor a 3	

Tabla 1 - Listar itinerarios populares

PL-01	Listar planes recomendados	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	NA	
Descripción	La aplicación mostrará al usuario los planes que se recomienda (o no) hacer en el momento actual (en función del tiempo, afluencia, etc.)	
Entrada	Página, tamaño página, fecha, hora, idioma	
Salida	Lista de planes recomendados	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El sistema muestra una lista reducida de los planes recomendados por la aplicación
	3	El usuario pulsa sobre un botón para ver más planes recomendados
	4	El sistema muestra las recomendaciones sobre los planes
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	El algoritmo utilizado para recomendar la realización de un plan es el algoritmo ID3, este utiliza parámetros como la afluencia de un lugar, el tiempo meteorológico y la fecha y hora de consulta.	

Tabla 2 - Listar planes recomendados

PL-02	Listar planes populares	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	NA	
Descripción	El usuario podrá visualizar los planes con mejor puntuación para saber qué planes son los que más valoran otros usuarios	
Entrada	Página, tamaño página, idioma	
Salida	Lista de planes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El sistema muestra una lista reducida de los planes más populares
	3	El usuario pulsa sobre un botón para ver más planes populares
	4	El sistema muestra todos los planes ordenados de mayor a menor popularidad
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	Un plan es popular si tiene una valoración mayor a 3	

Tabla 3 - Listar planes populares

IT-02	Ver detalle itinerario	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El itinerario debe existir	
Descripción	El usuario podrá visualizar la información del itinerario en detalle para poder ver cuál es la ruta que seguir y los planes a realizar	
Entrada	Identificador itinerario, idioma	
Salida	Identificador Itinerario, nombre itinerario, descripción del itinerario, lista de planes que conforman un itinerario	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación
	2	El usuario accede a una lista de itinerarios
	3	El usuario selecciona un itinerario
	4	El sistema muestra los detalles del itinerario
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	

Tabla 4 - Ver detalle itineario

IT-03	Ver valoraciones de un itinerario	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El itinerario debe existir	
Descripción	Se podrán ver las valoraciones que dejan los usuarios registrados sobre un itinerario	
Entrada	Identificador del itinerario, página, tamaño página, idioma	
Salida	Lista de valoraciones (usuario, rating, comentario), valoración media	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación
	2	El usuario accede a una lista de itinerarios
	3	El usuario selecciona un itinerario
	4	El sistema muestra los detalles del itinerario
	5	En esta sección se encuentran visible la lista de valoraciones de los distintos usuarios de la aplicación
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	Una valoración está compuesta por una puntuación de entre 1 y 5 estrellas además de un comentario	

Tabla 5 - Ver valoraciones de un itinerario

IT-04	Buscar itinerarios por nombre	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	NA	
Descripción	Los usuarios podrán buscar itinerarios por nombre	
Entrada	Nombre de un itinerario, página, tamaño página, idioma	
Salida	Lista de itinerarios coincidentes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de búsqueda
	3	El usuario teclea el nombre del itinerario y pulsa el botón de buscar
	4	El sistema muestra los resultados de la búsqueda
Postcondición	NA	
Excepciones	Paso	Acción
	4	La búsqueda no contiene resultados
	4	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	El sistema agrupará los itinerarios de 10 en 10 mostrándolos por páginas	

Tabla 6 - Buscar itinerarios por nombre

IT-05	Buscar itinerarios por autor	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Alta	
Precondición	NA	
Descripción	Los usuarios podrán buscar itinerarios por el nombre de usuario de la persona que los creó	
Entrada	Nombre del autor, página, tamaño página, idioma	
Salida	Lista de itinerarios coincidentes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de búsqueda
	3	El usuario teclea el nombre del autor
	4	El sistema muestra los resultados de la búsqueda
Postcondición	NA	
Excepciones	Paso	Acción
	4	La búsqueda no contiene resultados
	4	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	El sistema agrupará los itinerarios de 10 en 10 mostrándolos por páginas	

Tabla 7 - Buscar itinerarios por autor

PL-03	Ver detalle plan	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El plan debe existir	
Descripción	Se podrán ver los detalles de un plan, desde la ubicación, la mejor ruta para llegar hasta el, los horarios, la afluencia de personas en función de las horas, etc.	
Entrada	Identificador del plan, idioma	
Salida	Identificador del plan, nombre del plan, descripción del plan, localización (lat, lon), dirección	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación
	2	El usuario accede a una lista de planes
	3	El usuario selecciona un plan
	4	El sistema muestra los detalles del plan
Postcondición	NA	
Excepciones	Paso	Acción
	4	El plan seleccionado ya no existe
	4	Se mostrará una pantalla indicando al usuario que no existen el plan seleccionado
Comentarios	Dentro de la información detallada del plan se encuentran los horarios de apertura, la afluencia en función del momento del día/hora y los comentarios de los usuarios de la aplicación	

Tabla 8 - Ver detalle plan

PL-08	Buscar planes por horario	
Versión	1.0.0	
Prioridad	Madia	
Estabilidad	Media	
Precondición	NA	
Descripción	Los usuarios podrán buscar planes por sus horarios de apertura	
Entrada	Fecha y horas, página, tamaño página, idioma	
Salida	Lista de planes coincidentes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de búsqueda
	3	El usuario introduce una fecha y un rango de horas
	4	El sistema muestra los resultados de la búsqueda
Postcondición	NA	
Excepciones	Paso	Acción
	4	La búsqueda no contiene resultados
	4	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	El sistema agrupará los planes de 10 en 10 mostrándolos por páginas	

Tabla 9 - Buscar planes por horarios

PL-04	Ver valoraciones de un plan	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El plan debe existir	
Descripción	Se podrán ver las valoraciones que dejan los usuarios registrados sobre un plan	
Entrada	Identificador del plan, página, tamaño página, idioma	
Salida	Lista de valoraciones (usuario, valoración, comentario), media valoraciones	
Secuencia normal	Paso	Secuencia normal
	1	El usuario accede a la aplicación
	2	El usuario accede a una lista de planes
	3	El usuario selecciona un plan
	4	El sistema muestra los detalles del plan
	5	Se podrá ver la lista de valoraciones que realizan el resto de los usuarios de la aplicación
Postcondición	NA	
Excepciones	Paso	Acción
	4	El plan seleccionado ya no existe
	4	Se mostrará una pantalla indicando al usuario que no existen el plan seleccionado
Comentarios	Una valoración está compuesta por una puntuación de entre 1 y 5 estrellas además de un comentario	

Tabla 10 - Ver valoraciones de un plan

IMG-01	Visualizar imagen	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	NA	
Descripción	Los usuarios podrán visualizar imágenes que se almacenan en el sistema como por ejemplo la foto de portada de un plan o el avatar de otro usuario	
Entrada	Identificador imagen	
Salida	Fichero imagen	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación
	2	El usuario accede al detalle de cualquier plan
	3	El cliente realiza una petición con el identificador de la imagen del plan
	4	El sistema devuelve la imagen del plan
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	

Tabla 11 - Visualizar imagen

PL-06	Ver horarios y afluencia de un plan	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Baja	
Precondición	El plan debe existir	
Descripción	Se podrán ver los horarios y la afluencia de personas de un plan	
Entrada	Identificador del plan, idioma	
Salida	Lista de horarios y afluencia semanal	
Secuencia normal	Paso	Secuencia normal
	1	El usuario accede a la aplicación
	2	El usuario accede a una lista de planes
	3	El usuario selecciona un plan
	4	El sistema muestra los detalles del plan
	5	Se podrá ver el horario de un plan para los distintos días de la semana, así como los niveles de afluencia en función de un día y una hora
Postcondición	NA	
Excepciones	Paso	Acción
	4	El plan seleccionado ya no existe
	4	Se mostrará una pantalla indicando al usuario que no existen el plan seleccionado
Comentarios	NA	

Tabla 12 - Ver horarios y afluencias de un plan

PL-07	Buscar planes por nombre	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	NA	
Descripción	Los usuarios podrán buscar planes por nombre	
Entrada	Nombre de un plan, página, tamaño página, idioma	
Salida	Lista de planes coincidentes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de búsqueda
	3	El usuario teclea el nombre del plan y pulsa el botón de buscar
	4	El sistema muestra los resultados de la búsqueda
Postcondición	NA	
Excepciones	Paso	Acción
	4	La búsqueda no contiene resultados
	4	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	El sistema agrupará los planes de 10 en 10 mostrándolos por páginas	

Tabla 13 - Buscar planes por nombre

PL-09	Buscar planes por afluencia	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Media	
Precondición	NA	
Descripción	Los usuarios podrán buscar planes por nivel de afluencia	
Entrada	Nivel de afluencia de 0 a 100, página, tamaño página, idioma	
Salida	Lista de planes coincidentes	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de búsqueda
	3	El usuario selecciona un nivel de afluencia y pulsa el botón de buscar
	4	El sistema muestra los resultados de la búsqueda
Postcondición	NA	
Excepciones	Paso	Acción
	4	La búsqueda no contiene resultados
	4	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	Existen 4 niveles de afluencia, bajo de 0 a 25 por ciento de ocupación, medio de 25 a 50, alto de 50 a 75 y lleno de 75 a 100. El sistema agrupará los planes de 10 en 10 mostrándolos por páginas.	

Tabla 14 - Buscar planes por afluencia

PL-11	Listar planes relacionados	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Alta	
Precondición	NA	
Descripción	Los usuarios podrán ver planes relacionados con uno ya dado	
Entrada	Identificador plan, tamaño página, página, idioma	
Salida	Lista de planes que están relacionados	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario entra al detalle de cualquier plan
	3	El usuario podrá visualizar una lista de planes relacionados con el ya seleccionado en caso de haberlos
Postcondición	NA	
Excepciones	Paso	Acción
	3	La búsqueda no contiene resultados
	3	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	NA	

Tabla 15 - Listar planes relacionados

PL-10	Listar planes por categoría	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Alta	
Precondición	NA	
Descripción	Los usuarios podrán buscar planes por su categoría	
Entrada	Identificador categoría, página, tamaño página, idioma	
Salida	Lista de planes pertenecientes a esa categoría	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario se dirige a la zona de categorías en la pantalla principal
	3	El usuario selecciona una categoría
	4	El sistema muestra una lista reducida de resultados
	5	El usuario pulsa sobre un botón para ver más planes de esa categoría
	6	El sistema muestra todos los planes
Postcondición	NA	
Excepciones	Paso	Acción
	6	La búsqueda no contiene resultados
	6	Se mostrará una pantalla indicando al usuario que no existen coincidencias para su búsqueda
Comentarios	El sistema agrupará los planes de 10 en 10 mostrándolos por páginas	

Tabla 16 - Listar planes por categorías

MI-01	Mapa interactivo	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	NA	
Descripción	Los usuarios podrán buscar en un mapa interactivo los planes disponibles. Podrán hacer uso de la ubicación para mostrar los planes más cercanos a ellos	
Entrada	Localización (lat,lon), página, tamaño página, idioma	
Salida	Lista de planes en un radio de 1,5Km	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario entra la sección del mapa
	3	Pulsa el botón de buscar planes
	4	Los planes aparecerán situados en el mapa y se mostrarán breves descripciones de estos
Postcondición	NA	
Excepciones	Paso	Acción
	3	La búsqueda no contiene resultados
	3	Se mostrará el mapa vacío
Comentarios	NA	

Tabla 17 - Mapa interactivo

US-01	Registrar usuario	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	NA	
Descripción	Los usuarios podrán registrarse para poder disfrutar de las ventajas que esto conlleva	
Entrada	Correo electrónico, nombre, apellido, contraseña, nombre de usuario, avatar(imagen)	
Salida	Código activación de cuenta	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de usuario
	3	El usuario selecciona la opción de registro
	4	El usuario introduce todos los campos necesarios y pulsa el botón de completar registro
	5	El sistema valida la información y el usuario queda pre-registrado
	6	El sistema genera y envía una clave que tendrá que utilizar el usuario para activar su cuenta
Postcondición	Se genera un código de activación único	
Excepciones	Paso	Acción
	4	El registro no se ha podido completar
	4	La contraseña introducida no cumple con los requisitos, mensaje informativo al usuario
	4	El e-mail introducido no cumple con los requisitos, mensaje informativo al usuario
	5	El e-mail introducido ya está activo en otra cuenta, mensaje informativo al usuario
Comentarios	El acceso a la funcionalidad de registro estará presente en distintos lugares de la aplicación además del ya mencionado	

Tabla 18 - Registrar Usuario

4.3. Módulo usuarios registrados

En esta sección, se muestran los casos de uso asociados al usuario registrado. La Figura 6 muestra el diagrama de casos de uso de los usuarios registrados.

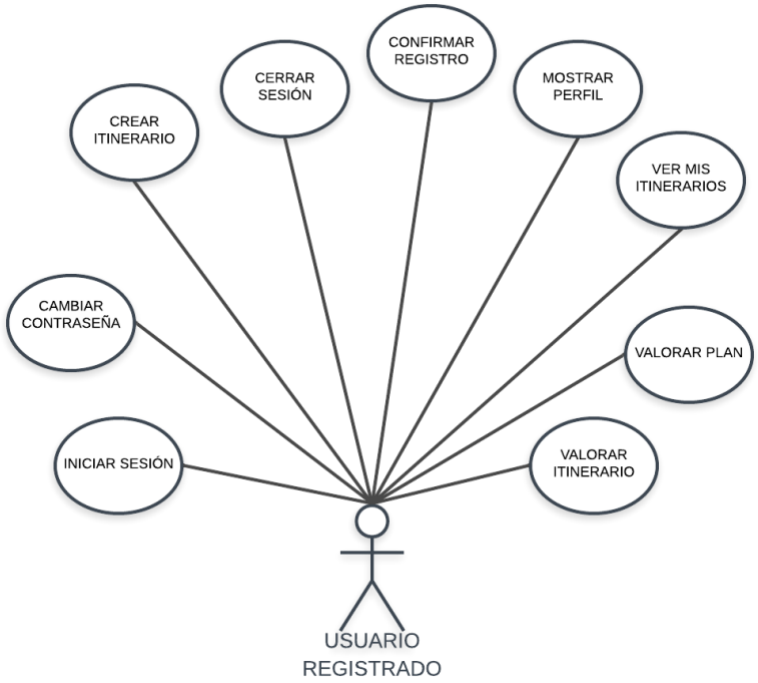


Figura 6 - Diagrama casos de uso módulo usuario registrado

US-02	Confirmar registro	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El usuario debe haber completado el registro	
Descripción	Los usuarios tendrán que confirmar el registro en el sistema	
Entrada	Código único de confirmación de registro	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede al correo enviado por el sistema en el momento del registro
	2	El usuario accede al enlace generado por el sistema
	3	El sistema confirma el código único de confirmación de cuenta
	4	El sistema da de alta al usuario
Postcondición	Se da de alta un usuario	
Excepciones	Paso	Acción
	3	Código de confirmación no corresponde con ninguna cuenta
Comentarios	NA	

Tabla 19 - Confirmar registro

US-04	Restaurar contraseña	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Media	
Precondición	El usuario ha de estar registrado	
Descripción	Un usuario puede restaurar la contraseña en caso de olvidarla	
Entrada	Email de usuario	
Salida	Mensaje del sistema	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se dirige a una de las pantallas de inicio de sesión
	2	Pulsa sobre la opción de restablecer contraseña
	3	Introduce el email y confirma la acción
	4	El sistema genera un identificador único para le cambio de contraseña asociada a una cuenta
	5	El sistema genera un correo electrónico con la clave y la envía a la dirección de correo asociada
	6	El usuario utiliza la clave recibida por correo para cambiar la contraseña
	7	El sistema actualiza la contraseña
Postcondición	El sistema genera una clave única para el cambio de contraseña	
Excepciones	Paso	Acción
	4	Si el correo no se reconoce entonces se informa al usuario
Comentarios	NA	

Tabla 20 - Restaurar contraseña

US-06	Mostrar perfil	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado y con la sesión iniciada	
Descripción	Los usuarios podrán ver la información de su perfil	
Entrada	Token autenticación, identificador de usuario	
Salida	Nombre, apellido, nombre usuario, avatar de usuario	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de usuario
	3	El sistema muestra la información del usuario
Postcondición	NA	
Excepciones	Paso	Acción
	3	El token de autenticación no corresponde con el del usuario, se informa del error
Comentarios	NA	

Tabla 21 - Mostrar perfil

US-05	Cerrar sesión	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado y con la sesión iniciada	
Descripción	Los usuarios podrán cerrar la sesión de su perfil	
Entrada	Token de autenticación, identificador de usuario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de usuario
	3	El usuario selecciona la opción de cerrar la sesión
		El sistema elimina los tokens asociados a esa cuenta y el dispositivo que estaba utilizando el usuario
	4	El sistema confirma el cierre de sesión y muestra la pantalla de inicio de sesión
Postcondición	El token de refresco queda invalidado	
Excepciones	Paso	Acción
	4	El token de autenticación no corresponde con el del usuario, se informa del error
Comentarios	NA	

Tabla 22 - Cerrar sesión

US-07	Ver mis itinerarios	
Versión	1.0.0	
Prioridad	Media	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado y con la sesión iniciada	
Descripción	Los usuarios podrán ver los itinerarios que han creado	
Entrada	Token autenticación, identificador de usuario	
Salida	Lista de itinerarios creados por el usuario	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de usuario
	3	El sistema muestra la lista de itinerarios que ha creado el usuario
Postcondición	NA	
Excepciones	Paso	Acción
	3	El token de autenticación no corresponde con el del usuario, se informa del error
Comentarios	NA	

Tabla 23 - Ver mis itinerarios

PL-12	Valorar plan	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado y con la sesión iniciada	
Descripción	Los usuarios podrán valorar los planes que estén disponibles en la aplicación	
Entrada	Token autenticación, identificador de usuario, valoración de 1-5 estrellas y comentario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación e inicia sesión
	2	El usuario selecciona el plan que quiere valorar
	3	Dentro del detalle del plan podrá rellenar el formulario de valoración
	4	Una vez rellenado pulsará en el botón para enviar la valoración
	5	El sistema confirma que la valoración se ha completado con éxito
Postcondición	NA	
Excepciones	Paso	Acción
	5	La valoración no se ha podido completar porque el usuario ya ha valorado el plan
Comentarios	NA	

Tabla 24 - Valorar un plan

IT-07	Valorar itinerario	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado	
Descripción	Los usuarios podrán valorar los itinerarios que estén disponibles en la aplicación	
Entrada	Token autenticación, identificador de usuario, valoración de 1-5 estrellas y comentario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación e inicia sesión
	2	El usuario selecciona el itinerario que quiere valorar
	3	Dentro del detalle del itinerario podrá rellenar el formulario de valoración
	4	Una vez rellenado pulsará en el botón para enviar la valoración
	5	El sistema confirma que la valoración se ha completado con éxito
Postcondición	NA	
Excepciones	Paso	Acción
	5	La valoración no se ha podido completar porque el usuario ya ha valorado el itinerario
Comentarios	NA	

Tabla 25 - Valorar itinerario

US-03	Iniciar sesión	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	El usuario ha de estar registrado	
Descripción	Los usuarios que ya estén registrados podrán iniciar sesión para acceder a los casos de uso reservados a estos actores.	
Entrada	Correo y contraseña o token de autenticación o token de refresco	
Salida	Toquen de autentificación y token de refresco	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de usuario
	3	El usuario selecciona la opción de iniciar sesión
	4	El usuario introduce sus credenciales y pulsa el botón de inicio
	5	El sistema comprueba las credenciales
	6	El sistema genera un token de autenticación y otro de refresco asociados a esa cuenta y ese dispositivo
	7	El sistema devuelve los tokens
Postcondición	NA	
Excepciones	Paso	Acción
	5	El inicio de sesión no se ha podido completar, las credenciales no son correctas
Comentarios	El token de autenticación se utiliza para iniciar sesión sin tener que utilizar las credenciales. Tiene un tiempo de vida útil y a partir de ese momento el usuario podrá solicitar otro token a través del token de refresco.	

Tabla 26 - Iniciar sesión

IT-06	Crear Itinerario	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Baja	
Precondición	El usuario ha de estar registrado, con la sesión iniciada y los planes han de existir	
Descripción	El usuario podrá crear itinerarios para después poder disfrutar de ellos. Podrá seleccionar desde el día, la franja horaria y los planes que quiere realizar incluso recibirá una recomendación del sistema con la mejor forma de disfrutar de su itinerario	
Entrada	Token autenticación, identificador de usuario, lista de planes, fecha, horario, categorías, descripción y nombre del itinerario	
Salida	Identificador del itinerario	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación y se le muestra la pantalla principal
	2	El usuario accede al apartado de crear
	3	El usuario escoge hora y fecha para el itinerario
	4	El usuario escoge los planes que conforman el itinerario
	5	El usuario selecciona las categorías del itinerario
	6	El usuario selecciona la hora específica para la realización de cada plan
	7	El usuario rellena un nombre y una descripción para el itinerario
	8	Se pulsa sobre el botón finalizar y el sistema confirma la creación del itinerario
Postcondición	La fecha de realización será superior a la fecha actual	
Excepciones	Paso	Acción
	3	La fecha introducida no cumple con los requisitos, mensaje informativo al usuario
	6	Alguno de los planes se solapa con otro en el horario de realización
Comentarios	Se podrá navegar entre los pasos para la creación de un itinerario	

Tabla 27 - Crear itinerario

4.4 Módulo administrador

En esta sección, se muestran los casos de uso asociados al usuario administrador. La Figura 7 muestra el diagrama de casos de uso para este actor.

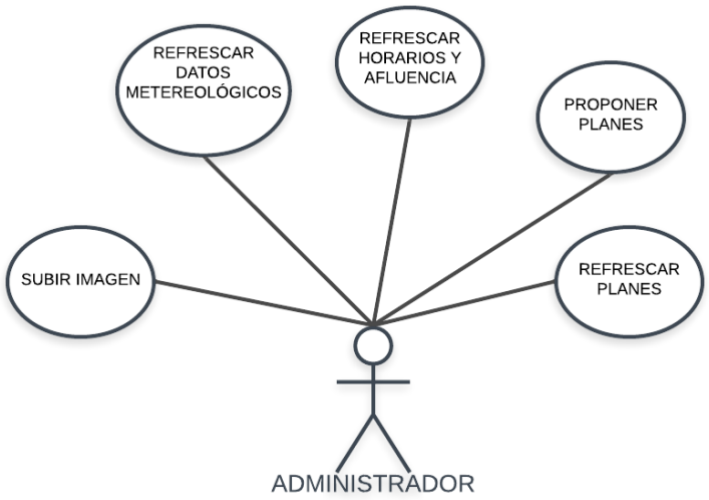


Figura 7 - Diagrama casos de uso módulo administrador

ADM-01	Refrescar planes	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El administrador tiene la sesión iniciada	
Descripción	El administrador puede lanzar una orden para actualizar los datos de los planes del repositorio público	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El administrador solicita que se refresque la información de los planes
	2	Se recibe y encola la orden para su procesamiento
	3	Se solicita al Ayuntamiento de Madrid la información de los planes de la ciudad
	4	El sistema procesa la información de cada plan e inicia el proceso de alta
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	

Tabla 28 - Refrescar plan

ADM-02	Proponer planes	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El administrador tiene la sesión iniciada	
Descripción	El administrador propone un plan para que forme parte del catálogo de la aplicación	
Entrada	Lista de información de plan multilenguaje (descripción, nombre e idioma), dirección del recurso imagen, identificador proveedor, un indicador para ver si es traducible o no, lista de identificadores categorías asociadas, apariencia (interior, exterior, desconocido), si está temporalmente cerrado, dirección (calle, identificador ciudad, código postal), localización (lat, lon)	
Salida	Identificador nuevo plan	
Secuencia normal	Paso	Acción
	1	El administrador solicita el alta de un nuevo plan
	2	El sistema comprueba que no existe otro plan con el mismo nombre en la misma localización
	3	Comprueba que la ciudad forma parte de las ya conocidas
	4	Comprueba que todas las categorías existen
	5	Se almacena la imagen en caso de haberla y queda asignada al plan
	6	Se genera un identificador para el nuevo plan
	7	El sistema da de alta el plan
Postcondición	El sistema ha registrado el plan	
Excepciones	Paso	Acción
Comentarios	NA	

Tabla 29 - Proponer planes

IMG-02	Subir imagen	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Media	
Precondición	El administrador tiene la sesión iniciada	
Descripción	El administrador podrá almacenar imágenes en el sistema	
Entrada	Fichero de imagen o dirección del recurso	
Salida	Identificador imagen	
Secuencia normal	Paso	Acción
	1	El administrador localiza la imagen y proporciona al sistema el fichero/dirección
	2	El sistema inicia la copia de la imagen y esta queda registrada con un identificador
	3	El sistema devuelve el identificador de la imagen
Postcondición	La imagen queda almacenada en el sistema	
Excepciones	Paso	Acción
Comentarios	NA	

Tabla 30 - Subir imagen

ADM-03	Refrescar horarios y afluencias de planes	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Baja	
Precondición	El administrador tiene la sesión iniciada	
Descripción	El administrador puede lanzar una orden para actualizar los datos los horarios y afluencias de los distintos planes del sistema	
Entrada	Identificador de un plan	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El administrador solicita que se refresque la información de los horarios y afluencias de un plan
	2	Se recibe y encola la orden de procesamiento
	3	El sistema comienza a desencolar las peticiones, procesándolas una a una
	4	El sistema actualiza los horarios y afluencias del plan
Postcondición	Se actualizan las afluencias y horarios de un plan	
Excepciones	Paso	Acción
Comentarios		

Tabla 31 - Refrescar horarios y afluencias de planes

ADM-04	Refrescar información meteorológica	
Versión	1.0.0	
Prioridad	Alta	
Estabilidad	Alta	
Precondición	La fecha de fin no puede ser 6 días posterior a la actual y el administrador tiene la sesión iniciada	
Descripción	El administrador puede lanzar una orden para actualizar los datos de meteorológicos a través de la AEMET	
Entrada	Fecha inicio, fecha fin, ciudad, ISO país	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El administrador solicita el pronóstico meteorológico en un rango de fechas
	2	El sistema realiza una petición a la AEMET y comienza a procesar los datos
	3	El sistema actualiza el parte meteorológico
Postcondición	Se actualiza el parte meteorológico	
Excepciones	Paso	Acción
Comentarios	De momento solo se ha integrado el servicio meteorológico de la ciudad de Madrid	

Tabla 32 - Refrescar información meteorológica

Capítulo 5. Modelo de datos

En este capítulo se describe el modelo de datos utilizado para persistir el sistema desarrollado.

5.1. Modelo E-R

En la Figura 8 se muestra el modelo entidad-relación en el que se muestran las entidades que intervienen en el sistema, sus atributos y las relaciones que tienen entre ellas.

El sistema guarda información de planes, su dirección, su nombre, su localización, su descripción y el identificador del proveedor de esta información. Todos los planes están localizados en una sola ciudad, de la cual se guardará el nombre y el país. Cada plan tendrá unas estadísticas diarias, el sistema obtendrá esas estadísticas todos los días del año. Una estadística diaria tiene dos reportes distintos, el horario y la afluencia.

Una afluencia representa el nivel de ocupación de un plan en un día concreto, guarda el porcentaje de ocupación y la hora a la que se da ese porcentaje. Un horario contiene una hora de apertura y una hora de cierre para un día y un plan concreto.

El sistema contiene usuarios registrados. Un usuario tiene un nombre, un apellido, un nombre de usuario, un e-mail, una contraseña y una imagen perfil.

Los usuarios podrán crear itinerarios, estos tienen un nombre, una descripción y una fecha de realización. Cada itinerario se relaciona con una lista de planes que lo componen y una hora de realización por cada plan.

Los itinerarios y los planes pueden ser valorados por los usuarios, estas valoraciones tienen un comentario y una valoración numérica de entre 1 y 5 puntos. Además, cada plan y cada itinerario están relacionados con una o más categorías.

Por último, el sistema guardará información sobre el tiempo meteorológico para una ciudad de un país en un día concreto. Este informe diario contiene pequeños informes de cada hora donde se recogen valores como la temperatura, la velocidad del viento, si hace calor o frio y el estado del cielo.

.

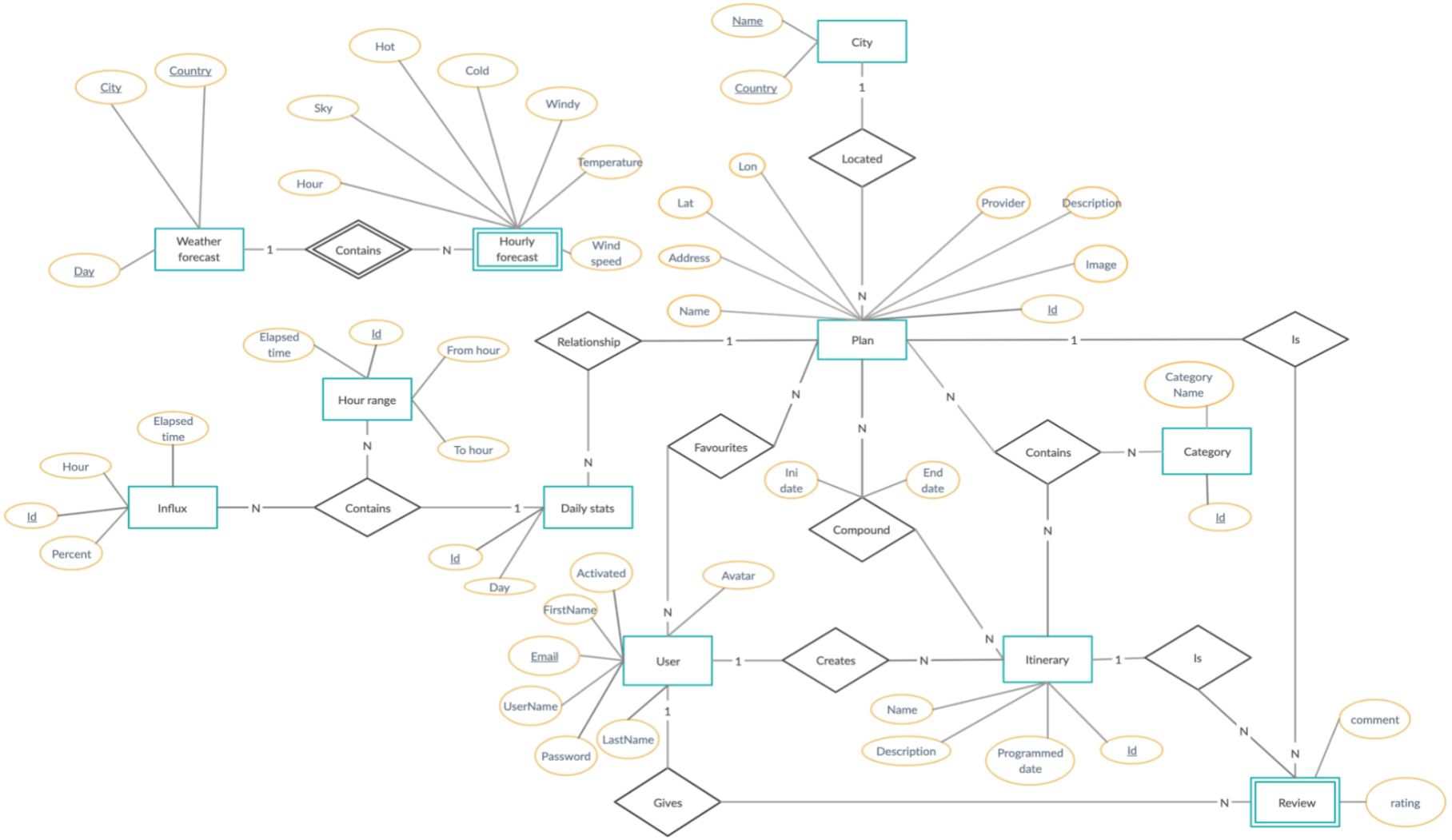


Figura 8 - Modelo entidad relación

5.3. Implementación base de datos

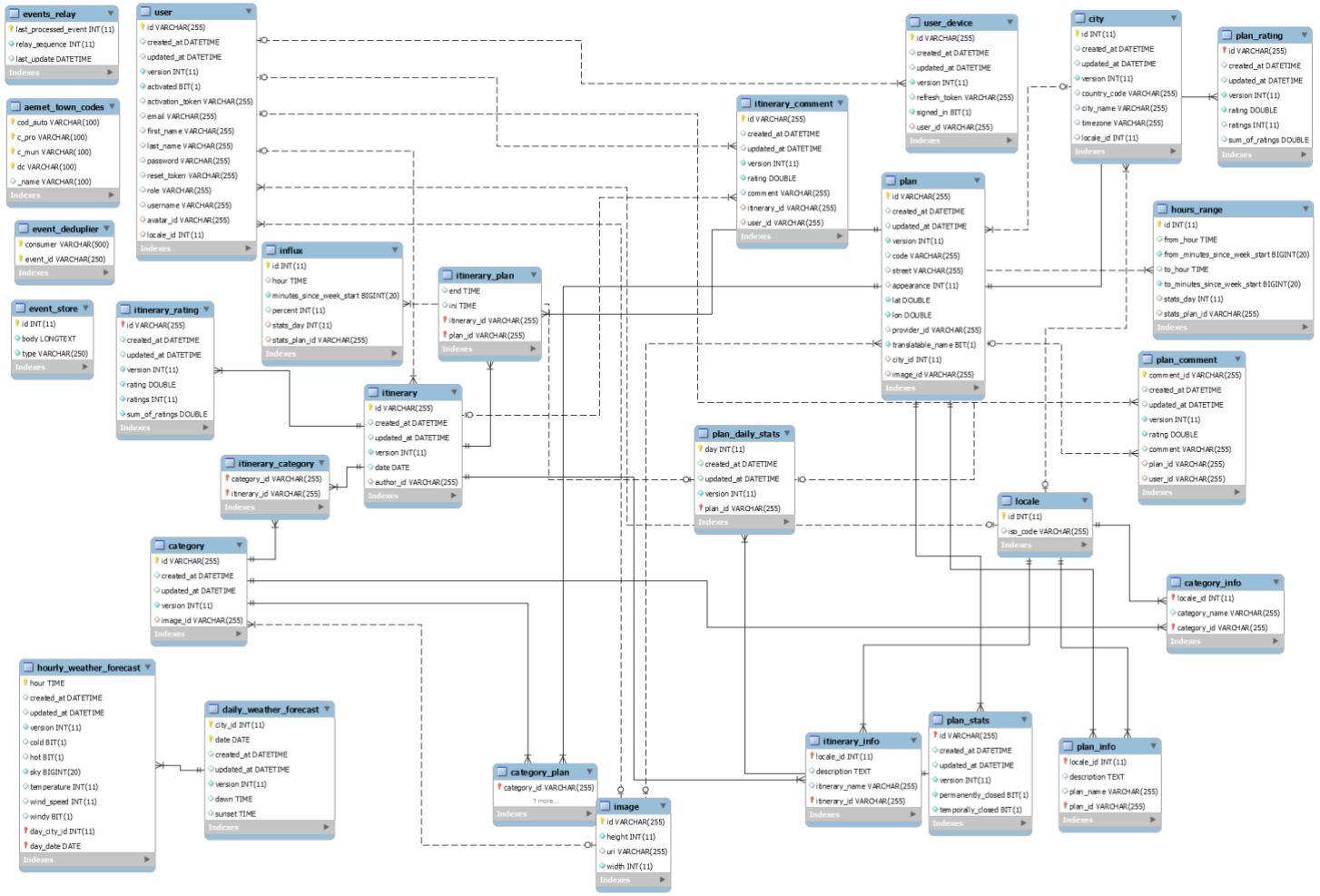


Figura 9 - Implementación modelo de datos

El modelo E-R descrito en el apartado anterior se ha implementado mediante una base de datos relacional de tipo MariaDB. En la Figura 9 se muestra el esquema con las tablas definidas y sus relaciones.

A continuación, se van a describir cada una de las tablas de la base de datos agrupadas por el tipo de información que almacenan.

5.3.1. Tablas sobre planes de ocio

En esta sección se describen las tablas que almacenan la información sobre los planes de ocio.

1) Tabla Plan: Contiene la información básica sobre un plan y está formada por los siguientes campos de información:
   - id: Es un identificador del plan.
   - created_at: Representa la fecha de creación del plan.
   - updated_at: Representa la fecha de actualización del plan.
   - version: Representa la versión del plan.
   - city_id: Es un identificador de la ciudad.
   - postal_code: Representa el código postal del plan.
   - street: Representa la calle del plan.
   - appearance: Representa si el plan es de interior o de exterior.
   - lat: Es la latitud del plan.
   - lon: Es la longitud del plan.
   - image_id: Es un identificador de la imagen.
   - provider_id: Es un identificador del proveedor.
   - translatable_name: Es un indicador para representar si el nombre del plan es traducible o no.

2) Tabla Locale: Guarda la información de los distintos idiomas en los que se puede obtener la información del sistema.
   - id: Es un identificador del idioma.
   - iso_code: Es un código ISO 639-1 [15] del idioma.

3) Tabla Plan Info: Contiene la información sobre un plan.
   - locale_id: Es un identificador del idioma con el que se quiere obtener la información de un plan
   - plan_id: Es un identificador del plan.
   - description: Representa la descripción del plan.
   - plan_name: Representa el nombre del plan.

4) Tabla Plan Comment: Contiene los comentarios y puntuaciones de un plan.
   - comment_id: Es un identificador del comentario.
   - created_at: Representa la fecha de creación del comentario.

- updated_at: Representa la fecha de actualización del comentario.
- version: Es una versión del comentario.
- comment: Es un comentario.
- plan_id: Es un id del plan asociado.
- rating: Representa la valoración numérica.
- user_id: Es un identificador del usuario.

5) Tabla Plan Rating: Contiene la media de puntuación de un plan, así como número de puntuaciones totales, de esta manera se puede recalcular la media fácilmente cada vez que se añade una valoración al plan.
   - plan_id: Es un identificador del plan asociado.
   - created_at: Representa la fecha de creación de la valoración media.
   - updated_at: Representa la fecha de actualización de la valoración media.
   - version: Es una versión de la valoración.
   - rating: Representa la valoración media numérica.
   - ratings: Representa el número de valoraciones
   - user_id: es un identificador del usuario asociado

6) Tabla City: Guarda la información sobre las distintas ciudades que se utilizan en el sistema.
   - id: Es un identificador ciudad.
   - created_at: Representa la fecha de creación de las estadísticas.
   - updated_at: Representa la fecha de actualización de las estadísticas.
   - version: Es una versión de las estadísticas.
   - country_code: Representa el código de país.
   - city_name: Representa el nombre de la ciudad.
   - timezone: Representa la zona horaria.
   - locale_id: Es un identificador del idioma.

7) Tabla Plan Daily Stats: Es la tabla que relaciona un plan con características como su afluencia o sus estados en función de un día concreto.
   - id: Es un identificador de las estadísticas.
   - created_at: Representa la fecha de creación de las estadísticas.
   - updated_at: Representa la fecha de actualización de las estadísticas.
   - version: Es una versión de las estadísticas.
   - day: Representa el día.
   - plan_id: Es un identificador del plan asociado.

8) Tabla Plan Stats: Guarda el estado de un plan, que puede ser cerrado permanentemente o cerrado temporalmente.
   - id: Es un identificador de las estadísticas.
   - created_at: Representa la fecha de creación de las estadísticas.

- updated_at: Representa la fecha de actualización de las estadísticas.
- permanently_closed: Representa si el plan está cerrado definitivamente.
- temporally_closed: Representa si el plan está cerrado temporalmente.

9) Tabla Influx: Contiene la información de la afluencia de un lugar (en este caso un plan) en una hora en concreto. Un porcentaje representará el nivel de ocupación del lugar y podrá ir acompañado de un mensaje. La afluencia se captura de lunes a domingo
   - id: Es un identificador de la afluencia.
   - hour: Representa la hora.
   - minutes_since_week_start: Representa los minutos que han pasado desde la medianoche del lunes hasta la hora actual.
   - percent: Representa el porcentaje de ocupación.
   - stats_day: Es un identificador del día.
   - daily_stats_id: Es un identificador de las estadísticas.

10) Tabla Hours Range: Contiene la información del horario de un lugar (en este caso un plan). Esta tabla se relaciona con daily_stats ya que queremos saber los horarios para los distintos días.
   - id: Es un identificador del horario.
   - created_at: Representa la fecha de creación del horario.
   - updated_at: Representa la fecha de actualización del horario.
   - version: Representa la versión del horario.
   - from_hour: Representa la hora de apertura.
   - to_hour: Representa la hora de cierre.
   - stats_plan_id: Es un identificador estadístico de un plan.

11) Tabla Category: Contiene la información de una categoría. Las categorías servirán para agrupar los planes en función de su naturaleza (restaurantes, monumentos, museos, etc.).
   - category_id: Es un identificador de la categoría.
   - created_at: Representa la fecha de creación de la categoría.
   - updated_at: Representa la fecha de actualización de la categoría.
   - version: Representa la versión de la categoría.
   - image_id: Es un identificador de la imagen de la categoría.

12) Tabla Categroy Info: Contiene la información de una categoría para cada uno de los idiomas disponibles.
   - category_id: Es un identificador de la categoría.
   - locale_id: Es un identificador idioma.
   - category_name: Representa el nombre de la categoría.

13) Tabla Plan Categories: Es la tabla relación entre plan y categoría.

- plan_id: Es un identificador del plan.
- category_id: Es un identificador de la categoría.

5.3.2. Tablas sobre itinerarios

En esta sección se describen las tablas que guardan información sobre los itinerarios.

14) Tabla Itinerary: Contiene la información básica sobre un itinerario.

- id: Es un identificador del itinerario.
- created_at: Representa la fecha de creación del itinerario.
- updated_at: Representa la fecha de actualización del itinerario.
- version: Representa la versión del itinerario.
- date: fecha de realización.
- autor_id: Es un identificador usuario creador.

15) Tabla Itinerary Stats: Contiene la información detallada sobre un itinerario en los distintos idiomas que maneja el sistema.

- itinerary_id: Es un identificador del itinerario.
- locale_id: Es un identificador idioma.
- description: Representa la descripción del itinerario.
- name: Representa el nombre del itinerario.

16) Tabla Itinerary Plan: Contiene la relación entre un itinerario y los planes que lo componen.

- id: Es un identificador de la relación.
- ini_hour: Representa la hora de inicio del plan.
- end_hour: Representa la hora finalización del plan.
- plan_id: Es un identificador del plan asociado.
- itinerary_id: Es un identificador del itinerario.

17) Tabla Itinerary Comments: Contiene los comentarios y puntuaciones de un itinerario.

- id: Es un identificador del comentario.
- created_at: Representa la fecha de creación del comentario.
- updated_at: Representa la fecha de actualización del comentario.
- version: Representa la versión del comentario.
- comment: Es un comentario.
- itinerary_id: Es un identificador itinerario.
- rating: Representa la valoración numérica.
- user_id: Es un identificador usuario.

18) Tabla Itinerary Rating: Contiene la media de puntuación de un itinerario, así como número de puntuaciones totales, de esta manera se puede recalcular la media fácilmente cada vez que se añade una valoración al itinerario.

- itinerary_id: Es un identificador del itinerario asociado.
- created_at: Representa la fecha de creación de la valoración media.
- updated_at: Representa la fecha de actualización de la valoración media.
- version: Representa la versión de la valoración.
- rating: Representa la valoración media numérica.
- user_id: Representa el número de valoraciones.

19) Tabla Itinerary Categories: Es la tabla relación entre itinerario y categoría.

- itinerary_id: Es un identificador del itinerario.
- category_id: Es un identificador de la categoría.

5.3.3. Tablas sobre meteorología

En esta sección se describen las tablas que guardan la información sobre los partes meteorológicos.

20) Tabla Daily Weather Forecast: Contiene la información meteorológica de una ciudad y día concretos además de datos adicionales.

- city_id: Es un identificador ciudad.
- date: Representa la fecha actual
- created_at: Representa la fecha de creación del parte meteorológico.
- updated_at: Representa la fecha de actualización del parte meteorológico.
- version: Representa la versión del parte meteorológico.
- dawn: Representa la hora salida del sol
- sunset: Representa la hora puesta del sol

21) Tabla Hourly Weather Forecast: Contiene la información meteorológica por horas de un día concreto, está tabla se relaciona con daily_weather_forecast ya que esta última hace referencia al lugar y la fecha.

- hour: Representa la hora del parte.
- created_at: Representa la fecha de creación del parte meteorológico de la hora.
- updated_at: Representa la fecha de actualización del parte meteorológico de la hora.
- version: Representa la versión del parte meteorológico de la hora.
- cold: Representa si hace frio.
- hot: Representa si hace calor.
- sky: Representa el estado del cielo.
- temperature: Representa la temperatura en la escala de Celsius.
- windy: Representa si hay viento o no.
- wind_speed: Representa la velocidad del viento.

- day_city_id: Es un identificador ciudad de parte diario.
- day_date: Representa la fecha del parte diario.

22) Tabla Aemet Town Codes: Guarda la información de las ciudades a las que se puede llamar a través de la API de AEMET [16] .

- cod_auto: Representa el código de la comunidad autonómica.
- c_prod: Representa el código provincia.
- c_mun: Representa el código municipio.
- dc: Representa el digito de control.
- name: Representa el nombre de la ciudad.

5.3.4. Tablas sobre usuarios

En esta sección se van a describir las tablas que contienen la información de los usuarios.

23) Tabla User: Contiene la información del perfil de un usuario.

- id: Es un el identificador del usuario.
- created_at: Representa la fecha de creación del usuario.
- updated_at: Representa la fecha de actualización del usuario.
- version: Representa la versión del usuario.
- activated: Representa si la cuenta de usuario ha sido activada.
- activation_token: Representa el token de activación de cuenta.
- username: Representa el *nick* del usuario.
- email: Representa el email del usuario.
- first_name: Representa el primer nombre del usuario.
- last_name: Representa el apellido del usuario.
- password: Representa la contraseña del usuario.
- reset_token: Es un token para cambiar la contraseña.
- avatar_id: Es un identificador de la imagen de avatar del usuario.
- locale_id: Es un identificador del idioma.
- role: Representa el rol de un usuario.

24) Tabla User Devices: Relaciona el token de refresco de sesión de un usuario con un dispositivo concreto, de esta manera se permite al usuario poder acceder de manera sencilla al sistema desde todos sus dispositivos.

- id: Es un identificador del dispositivo.
- created_at: Representa la fecha de creación del dispositivo.
- updated_at: Representa la fecha de actualización del dispositivo.
- version: Representa la versión del dispositivo.
- refresh_token: Representa el token de refresco, se le proporciona este toquen al usuario para poder solicitar uno nuevo cuando este haya caducado.
- logged_in: Representa si el usuario está usando ese dispositivo.

- user_id: Es un identificador del usuario.

5.3.5. Tablas sobre sistemas de integración

En esta sección se describen las tablas que guardan la información necesaria para el correcto funcionamiento del sistema al integrarse con aplicaciones de terceros (sistema de colas).

25) Tabla Event Store: Almacena los eventos que se envían al sistema externo en un primer momento.

    - id: Es un identificador del evento.
    - body: Representa toda la información necesaria para el procesamiento del evento en el sistema externo (mensaje).
    - type: Representa el tipo de evento.

26) Tabla Event Deduplier: Almacena el identificador del evento y el identificador del sistema que se encarga en un primer momento de procesarlo.

    - consumer: Es un identificador del consumidor del evento.
    - event_id: Es un identificador del evento.

27) Tabla Events Relay: Almacena datos sobre los eventos que se han enviado.

    - last_processed_event: Es un identificador del último evento enviado.
    - relay_secuence: Representa el número de secuencia que indica cual es el siguiente evento por enviar.
    - last_update: Representa la última fecha en la que se enviaron eventos.

Capítulo 6. Arquitectura.

En este capítulo se describe la arquitectura del sistema desarrollado, explicando de forma detallada cada uno de los elementos que la componen, además de los patrones de diseño aplicados para el desarrollo de los componentes software.

6.1. Arquitectura del sistema

La arquitectura del sistema se basa en un modelo tradicional cliente(s) – servidor [17].

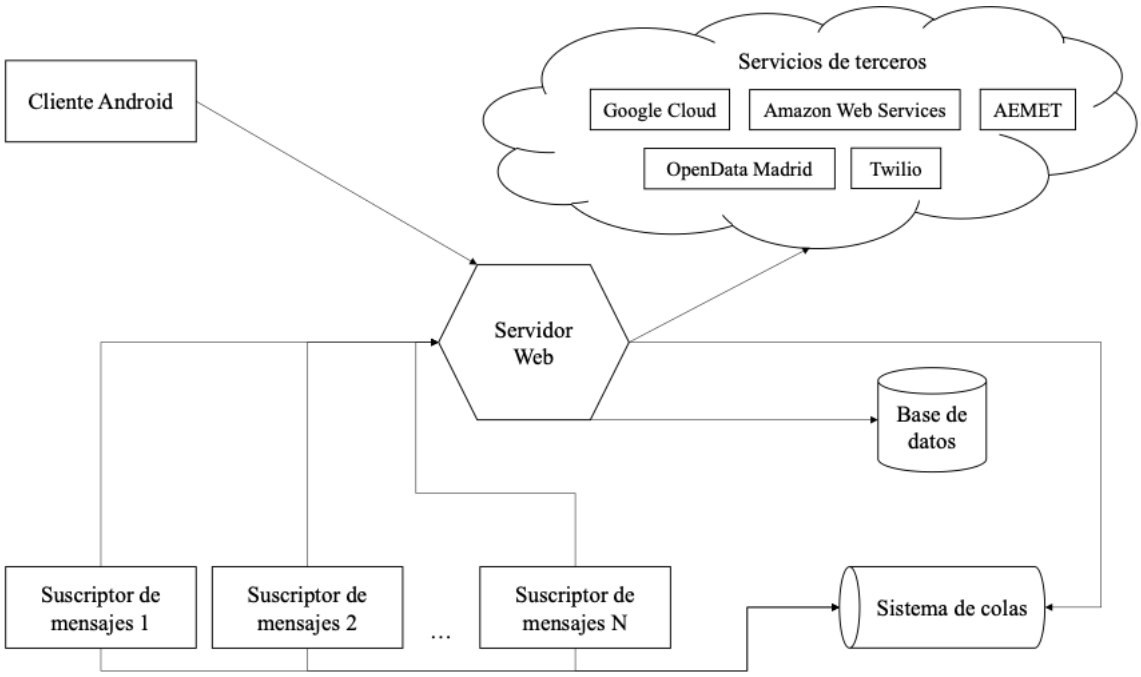


Figura 10 - Arquitectura del sistema

Como se observa en la Figura 10, el cliente Android del sistema interactúa con el servidor a través de una API que expone todos los servicios disponibles en este. Para uso de la API, los clientes utilizan el protocolo de comunicación HTTP [18].

En el lado servidor, este procesa las peticiones de cada cliente y ejecuta la lógica de negocio correspondiente a cada caso de uso, abstrayendo a los clientes de los procesos particulares de cada acción.

Para ciertas tareas relacionadas con el almacenamiento de imágenes, la obtención de datos meteorológicos y el envío de correos electrónicos, el servidor web hace uso de servicios de terceros que pueden verse representados en la ilustración en forma de nube. La interacción con estos servicios es realizada a través de los servicios web expuestos por los mismos.

El servidor web requiere de procesamiento asíncrono de ciertas tareas que se consideran bloqueantes y empeorarían la experiencia del usuario por la lentitud en la respuesta. En este punto el servidor encola las tareas en través de RabbitMQ [7], el *broker* de mensajería utilizado en el proyecto.

Por último, los suscriptores de mensajes son notificados por el sistema de colas para el procesamiento de los mensajes, ejecutando estos los casos de uso correspondiente.

6.2. Patrones de diseño y arquitectónicos

En esta sección se enumeran y contextualizan los diferentes patrones utilizados para el diseño y construcción del sistema.

6.2.1. Patrones arquitectónicos

Los patrones arquitectónicos describen de forma abstracta la solución a un problema común en el diseño de componentes software. A diferencia de los patrones de diseño, los arquitectónicos intentan dar solución a nivel de sistema y/o componentes de este.

- **MVP (Modelo-Vista-Presentador)**

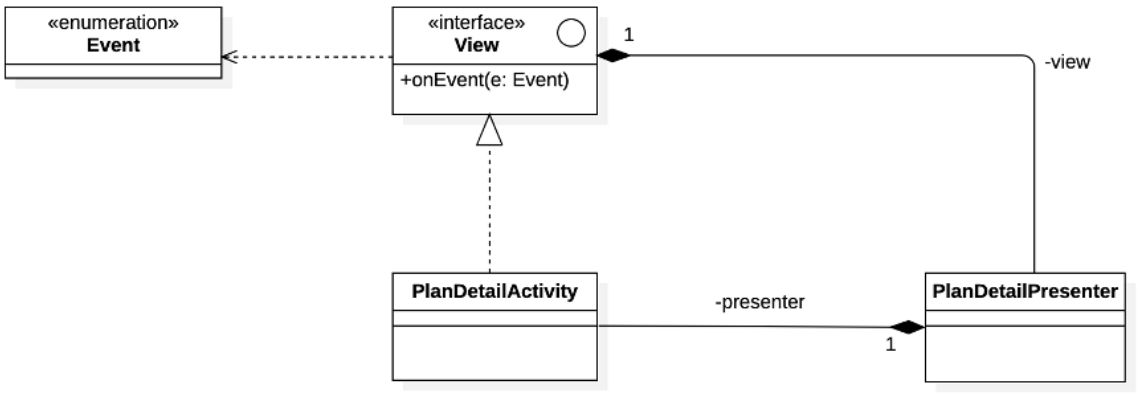


Figura 11 Aplicación MVP en el detalle del plan

El patrón Modelo-Vista-Presentador [19] ha sido aplicado para separar la parte de la vista (Activity o Fragment en el SDK de Android), con respecto de las llamadas al servidor. Es el presentador el que interactúa con la capa de servicios, devolviendo una respuesta a las vistas para que sean actualizadas con los datos. Se puede observar en la Figura 11 un diagrama UML con un ejemplo de este patrón aplicado en el proyecto.

- **Hexagonal Architecture / Ports and Adapters**

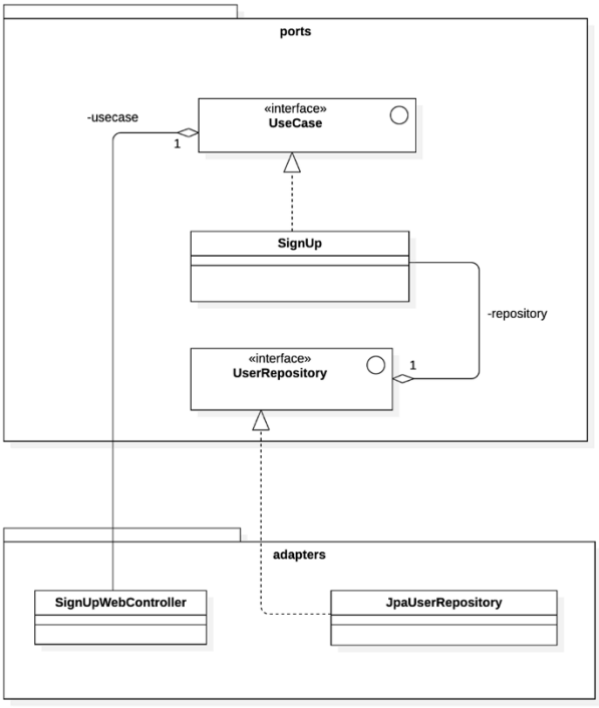


Figura 12 Ejemplo de puerto y adaptador en el servidor web

Hexagonal Architecture (también conocida como Ports & Adapters) es una arquitectura acuñada en el año 2005 por Alistair Cockburn [20].

Los casos de uso del lado servidor han sido desarrollados como puertos que son utilizados por los controladores web, que actúan de adaptadores. Además, las clases que interactúan con servicios de terceros también han sido diseñadas como adaptadores. Se puede observar la implementación de este patrón en la Figura 12.

- **Service Layer**

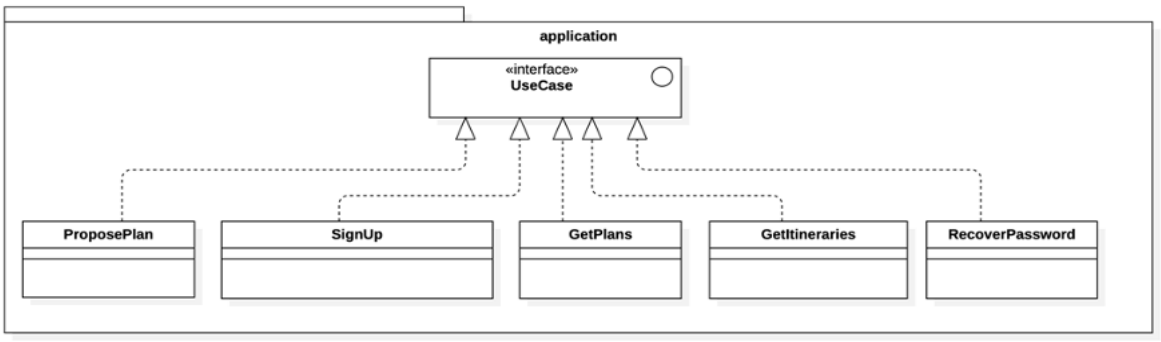


Figura 13 Capa de aplicación en el servidor conteniendo los casos de uso

En el lado del servidor, toda la lógica de negocio está expuesta a través de una Service Layer [59] de clases que representan cada uno de los casos de uso

disponibles en la aplicación. Estos casos de uso son utilizados por los clientes. En el diagrama de la Figura 13 se muestran algunos de los componentes que forman parte de la capa de servicios.

6.2.2. Patrones de Diseño

- **Adapter**

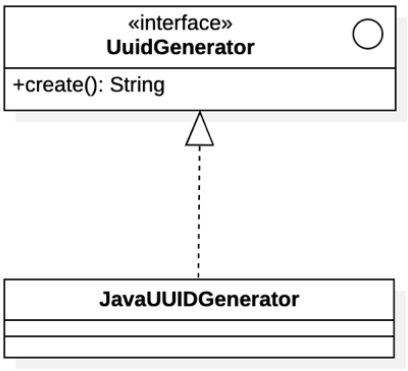


Figura 14 Adaptador para la generación de identificadores

El patrón *Adapter* [22] ha sido aplicado en el lado del servidor de tal forma que las dependencias con librerías de terceros o APIs que puedan suponer un problema para la mantenibilidad son adaptadas a través de una interfaz para evitar la dependencia de la capa de negocio con estas.

En la Figura 14 se observa un ejemplo en el que la generación de identificadores a través de la librería de Java ha sido adaptada.

- **Repository**

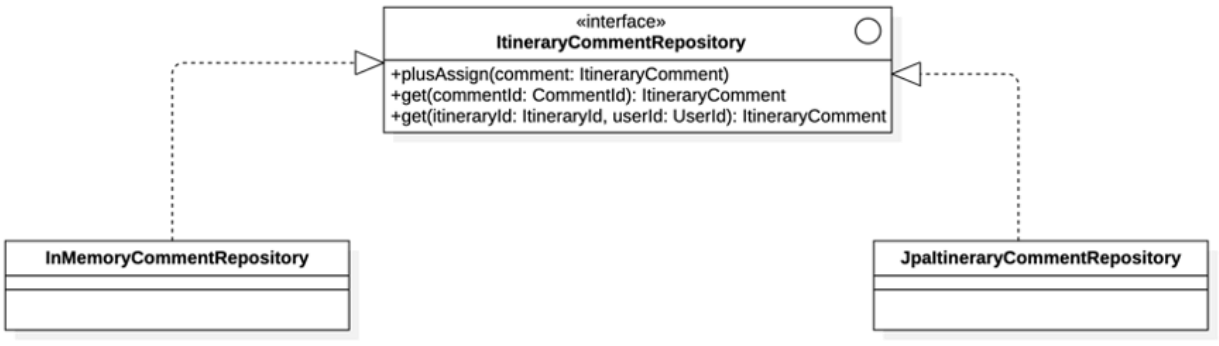


Figura 15 - Repository para la persistencia de los comentarios

El patrón Repository [21][23][24] en el lado del servidor hace posible que todos los objetos de negocio sean persistidos y recuperados a través de una clase diseñada según este, de forma que los servicios de aplicación interactúan con las clases

relacionadas con la persistencia de igual forma que lo harían con una colección en memoria. La Figura 15 muestra la implementación de este patrón.

- **Singleton**

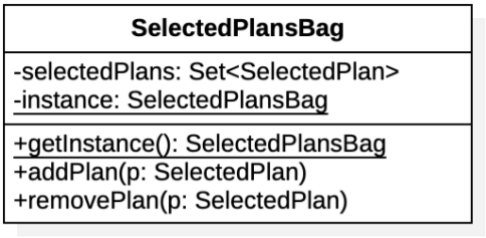


Figura 16 Singleton utilizado para el flujo de crear itinerario

El patrón de diseño Singleton [25] ha sido utilizado tanto el lado cliente como en el lado servidor. En la aplicación Android ha sido usado para disponer de una instancia única de la API de Retrofit para hacer las llamadas al servidor.

Además, también ha sido utilizado para almacenar en memoria los planes seleccionados en el flujo de creación de un itinerario, tal y como se muestra en la Figura 16.

Por otro lado, en el lado servidor, al iniciarse este, el contenedor de dependencias del marco de trabajo crea una instancia de cada uno de los servicios de aplicación, generando de facto un Singleton por cada uno de ellos.

- **Chain of Responsability**

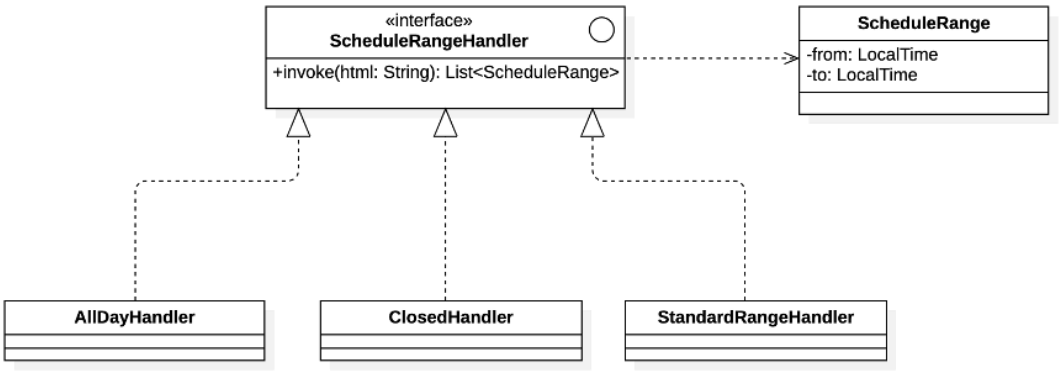


Figura 17 Uso del patrón para la extracción de los horarios

El patrón *Chain of Responsability* [26] ha sido aplicado debido a los diferentes formatos que existen en la forma de mostrar los horarios y afluencias de personas en los resultados del buscador Google.

La cadena de objetos intenta procesar el HTML y en caso de no corresponder al esperado, delega en el siguiente objeto de la cadena, y así hasta que una de las clases

será la que interprete el formato y lo transforme a un objeto. En la Figura 17 pueden verse las tres clases que componen la cadena.

- **Entity (DDD)**

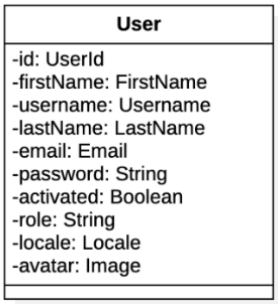


Figura 18 La entidad que representa a los usuarios

En el servidor, las clases que representan el dominio de la aplicación (Plan, Itinerario, Usuario, etc.) y su lógica de negocio asociada, son representadas como objetos de negocio, o lo que es igual, Entities [23][24].

- **Value Object (DDD)**

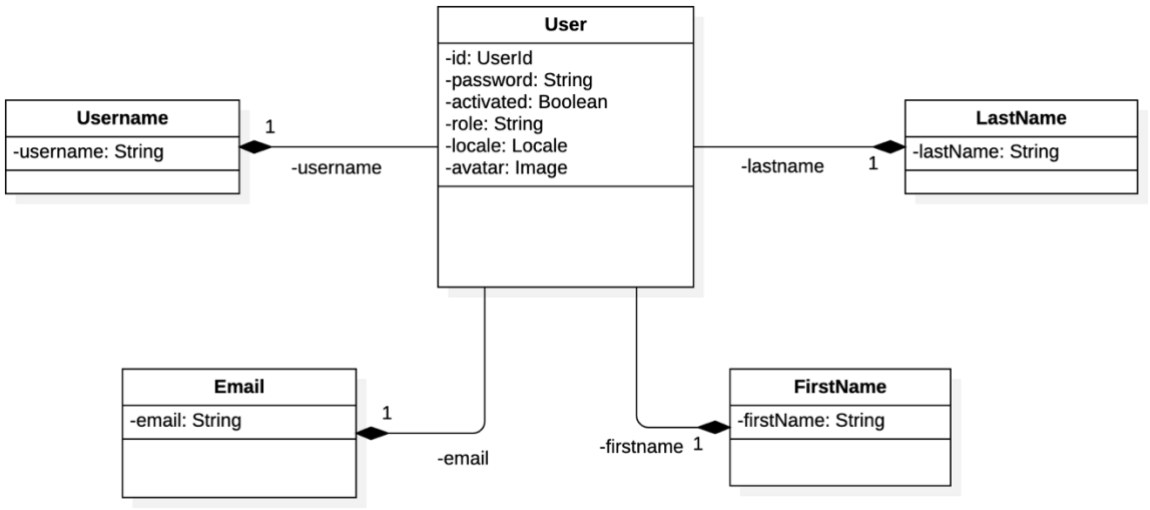


Figura 19 Los atributos del usuario son representados como value objects

En el lado del servidor para facilitar y hacer más específico el control de errores, cada uno de los atributos de las entidades de negocio, han sido modelados como Value Object [23][24] y no como tipos primitivos, validándose dentro de ellos las reglas de negocio asociadas.

En la Figura 19 se representan los Value Object que componen parte de la entidad User.

- **Domain Event**

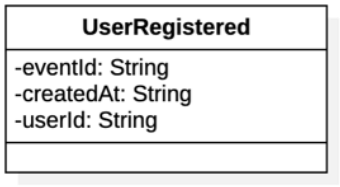


Figura 20 Evento creado al registrar un nuevo usuario

En el lado del servidor, e han modelado *Domain Events* [23][24] relacionados con acciones en los casos de uso, por ejemplo, cuando un usuario se registra, el evento *UserRegistered* es utilizado para acciones derivadas como el envío del correo electrónico de confirmación de cuenta.

La Figura 20 muestra la implementación del patrón mediante un diagrama UML.

- **Layer supertype**

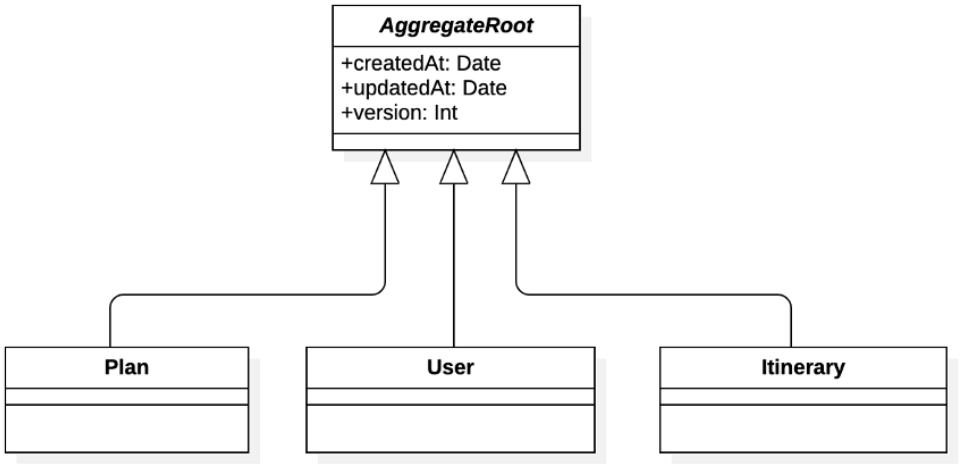


Figura 21 Entidades de negocio compartiendo una misma Layer Supertype

En el lado del servidor, se ha modelado una clase común a todas las entidades de negocio, actuando como *Layer Supertype* [21]. De esta forma todas las entidades comparten métodos y atributos comunes a través de una clase abstracta de la que heredan. En la Figura 21 se muestran las entidades Plan, User e Itinerary como tres ejemplos de especializaciones de la clase abstracta AggregateRoot.

- **DTO (Data Transfer Object)**

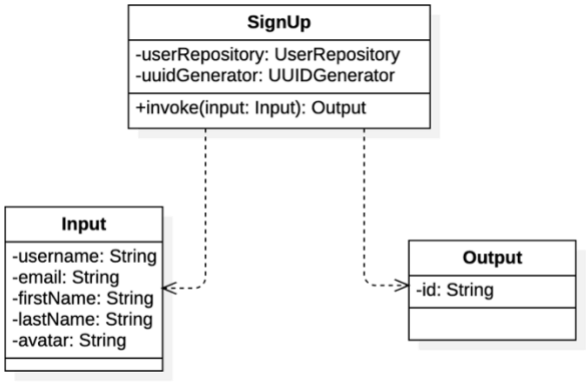


Figura 22 Ejemplo de DTO en los parámetros de un servicio de aplicación

Tanto en la aplicación Android como en el servidor, todos los parámetros de entrada y los datos de retorno de cada uno de los casos de uso son modelados como DTO [21], de forma que estos actúan de estructuras de datos que evitan exponer la representación interna de los objetos de negocio.

En la Figura 22 se observa el caso de uso SignUp, el cuál expone como tipo de retorno un DTO de tipo Output y recibe como parámetro de entrada un DTO de tipo Input, abstrayendo así a los clientes de las clases internas de las que hace uso la clase SignUp.

- **Inyección de dependencias**

```
@UseCase
class SignUp(private val userRepository: UserRepository,
             private val passwordEncoder: PasswordEncoder,
             private val imageBucket: ImageBucket,
             private val eventBus: DomainEventBus,
             private val localeRepository: LocaleRepository,
             val uuidGenerator: UUIDGenerator) { ...
```

Figura 23 Ejemplo de caso de uso recibiendo las dependencias a través de spring boot

Los servicios de aplicación que están expuestos en el lado del servidor contienen dependencias en su constructor que son inyectadas al iniciar el servidor web.

Esta inyección es realizada a través del marco de trabajo, el cual contiene un contenedor de dependencias que explora todas las dependencias de las clases a través de anotaciones. En la Figura 23 se muestra un caso de uso el cual recibe las dependencias que requiere a través de su constructor, que será invocado por el marco de trabajo.

Capítulo 7. Diseño

En este capítulo, se van a describir los principales aspectos acerca de la funcionalidad y el diseño realizado para el desarrollo del proyecto.

7.1 Colores

Para la elección del color de la aplicación se estableció una línea de diseño sencilla, basada en el mínimo de colores posibles y que a su vez fuese llamativa, para ello se escogió 3 colores base, el primero el blanco debido a su sencillez. El objetivo de los otros dos colores era que contrastasen con el blanco. Así pues, se escogió el negro y un naranja pomelo, tal como se observa en la Figura 25. En la Figura 24 se observan los 3 tipos de fuentes que se han utilizado, se optó por estas debido a su sencillez.

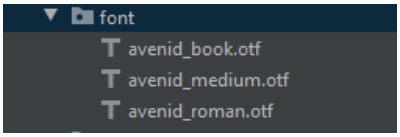


Figura 24 - Fuentes usadas

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#1D000000</color>
    <color name="colorPrimaryDark">#25000000</color>
    <color name="colorAccent">#FF5722</color>
    <color name="colorAccentTransparent">#CCFF5722</color>
    <color name="bottomNavViewColor">#FDFBF8</color>
    <color name="opaqueBlack">#B4000000</color>
    <color name="black">#000000</color>
    <color name="opaqueBlackLight">#51000000</color>
    <color name="red">#FF0000</color>
    <color name="white">#FFFFFF</color>
</resources>
```

Figura 25 – Archivo configuración android donde se especifican los colores de la aplicación

7.2 Funcionalidad de la aplicación

En este apartado se describen las funcionalidades implementadas, así como las vistas más importantes de la aplicación.

7.2.1 Dashboard

Es la pantalla principal de la aplicación. Al acceder a esta pantalla, se muestra unos listados con una serie de planes. Estos muestran varios listados: planes populares, planes por categorías, planes recomendados e itinerarios populares, como se observa en la Figura 26. La lista de planes recomendados se basa en un algoritmo inteligente del que se hablará más adelante.

Los planes y los itinerarios populares son aquellos que coinciden con una media aritmética de todas sus valoraciones superior a 3. Las valoraciones son realizadas por usuarios registrados sobre cada itinerario y plan.

En cada listado de los mencionados se muestran única y exclusivamente 10 planes o itinerarios, aunque es posible acceder al resto de ellos pulsando al final de la lista sobre la flecha que se muestra en la Figura 27.

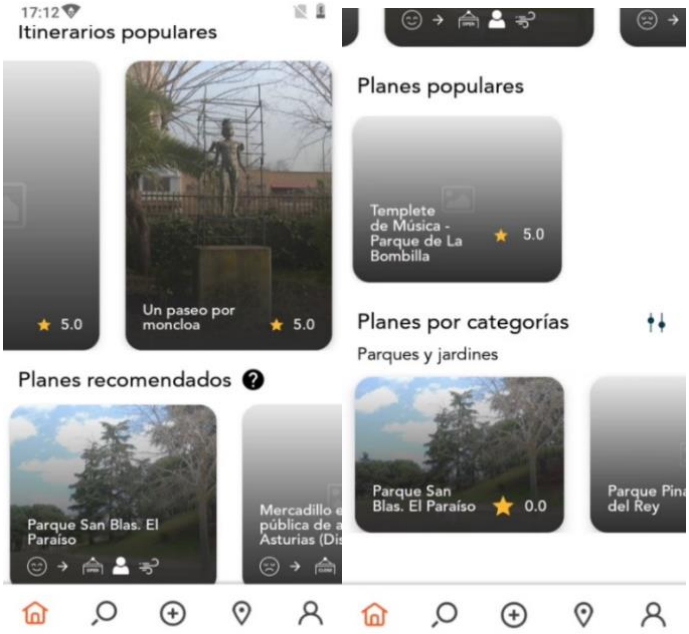


Figura 26 – Dashboard

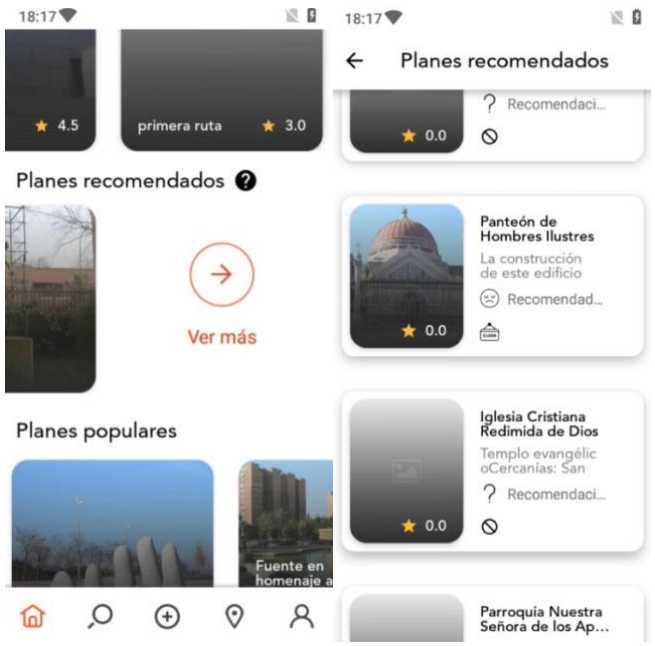


Figura 27 - Ver más planes o itinerarios

Para mostrar las listas de planes e itinerarios se utilizó un *RecyclerView*, el componente gráfico de Android que permite crear listas de elementos. Cada lista depende de una clase *Adapter*, que es el lugar donde los datos de cada elemento

de la lista son insertados en la interfaz gráfica, para ello se creó un *Adapter* genérico.

Las funciones más importantes son la que se encarga de añadir los datos a los elementos para que cuando se haga *scroll* en la lista se muestren, cómo se observa en la Figura 28.

```
final override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        if (holder !is LoadingVH) {
            onBind(holder as T, items[position])
        } else {
            holder.progressBar.visibility = if (isLoading) View.VISIBLE else View.GONE
        }
    }
```

Figura 28 - Función onBindViewHolder

En la figura anterior, se observa que para cargar los datos se llama a una función *bind* que es la encargada de añadir esos datos a los nuevos elementos de la lista como bien se observa en la Figura 29 y Figura 30.

```
override fun onBindViewHolder(holder: ItineraryPlanVH, position: Int) {
        val item = items[position]

        holder.bind(items[position])
        holder.itemView.setOnClickListener {
            listener.onItineraryPlanClick(item)
        }
    }
```

Figura 29 - Sobreescritura onBindViewHolder

```
class ItineraryPlanVH(val binding: ItemItineraryPlanBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(item: ItineraryPlan) {
            binding.itiPlanNameTv.text = item.name
            binding.iniHourTv.text = item.iniTime
            binding.endHourTv.text = item.endTime

            if (item.imageUrl == null) {
                Glide.with(itemView)
                    .load(R.drawable.not_image_available)
                    .into(binding.itiPlanIv)
            } else {
                Glide.with(itemView)
                    .load(item.imageUrl)
                    .into(binding.itiPlanIv)
            }
        }
    }
```

Figura 30 - Llenado de datos en adapter

Por otro lado, para consumir el *endpoint* con la información proporcionada desde la API se ha hecho uso de una librería, Retrofit, que permite realizar

funciones GET, PUT, DELETE... y transforma la información obtenida a clases POJO.

En la Figura 31 se puede observar la manera en la que se ha implementado una petición a la API.

```
@GET("/itineraries/popular")
suspend fun getPopularItineraries(
    @Query("s") pageSize: Int,
    @Query("p") page: Int,
    @Query("lang") lang: String
): GetPopularItinerariesResponse
```

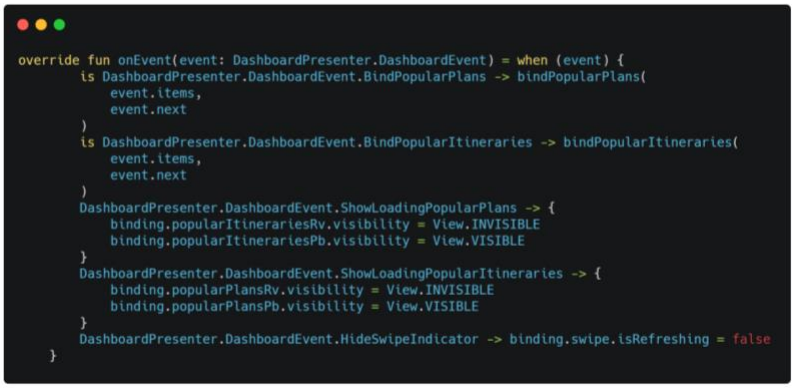
Figura 31 - Petición desde cliente a API

```
class GetPopularItineraries(
    private val sBackend: ItineraryRepository,
    gson: Gson
) : UseCase<GetPopularItineraries.Input, GetPopularItinerariesResponse>(gson) {

    data class Input(
        val page: Int = 1,
        val pageSize: Int = 10
    )

    override suspend fun execute(params: Input): GetPopularItinerariesResponse {
        return sBackend.getPopularItineraries(
            page = params.page,
            pageSize = params.pageSize,
            lang = Locale.getDefault().language
        )
    }
}
```

Figura 32 - Caso de uso para tratar datos obtenidos por una petición

Una vez se ha realizado la petición se ha de tratar los datos obtenidos, como se observa en la Figura 32, el caso de uso devuelve al *presenter*, que es donde se implementa la lógica de cada caso de uso, los datos obtenidos por la corrutina al hacer la petición a la API.

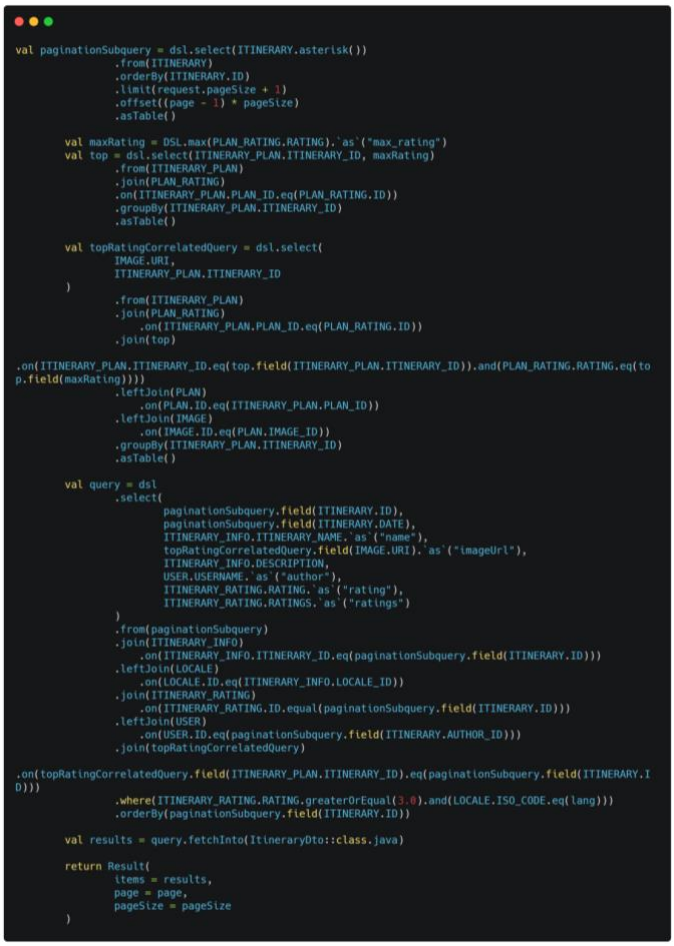
Al manejar varias listas en una misma vista, como es el caso del *dashboard*, se ha implementado una función que identifica el evento que está recibiendo el *presenter* para saber de qué tipo son los datos obtenidos de la API. Como se observa en la Figura 33 el gestor de eventos se encarga de llamar a la función correspondiente para cargar los datos de cada ítem del *adapter* que posteriormente se engancha con el *RecyclerView* para que se vea la lista.



```
override fun onEvent(event: DashboardPresenter.DashboardEvent) = when (event) {
        is DashboardPresenter.DashboardEvent.BindPopularPlans -> bindPopularPlans(
            event.items,
            event.next
        )
        is DashboardPresenter.DashboardEvent.BindPopularItineraries -> bindPopularItineraries(
            event.items,
            event.next
        )
        DashboardPresenter.DashboardEvent.ShowLoadingPopularPlans -> {
            binding.popularItinerariesRv.visibility = View.INVISIBLE
            binding.popularItinerariesPb.visibility = View.VISIBLE
        }
        DashboardPresenter.DashboardEvent.ShowLoadingPopularItineraries -> {
            binding.popularPlansRv.visibility = View.INVISIBLE
            binding.popularPlansPb.visibility = View.VISIBLE
        }
        DashboardPresenter.DashboardEvent.HideSwipeIndicator -> binding.swipe.isRefreshing = false
    }
```

Figura 33 - Función que controla los eventos del presenter

Se utilizan varias consultas para la devolución de los datos que necesita mostrar el cliente. En primer lugar, como la API hace uso de paginación, se requiere de una consulta que escoja todos aquellos itinerarios que correspondan a la página solicitada por el cliente. Tras esto, se obtiene la imagen que irá asociada al itinerario a través de la imagen del plan que mejor valoración tenga de entre todos los que conforman el itinerario. Por último, se extraen el resto de los datos de los itinerarios a través de una consulta y se devuelve el resultado como se observar en la Figura 34



```
val paginationSubquery = dsl.select(ITINERARY.asterisk())
                .from(ITINERARY)
                .orderBy(ITINERARY.ID)
                .limit(request.pageSize + 1)
                .offset((page - 1) * pageSize)
                .asTable()

        val maxRating = DSL.max(PLAN_RATING.RATING).`as`("max_rating")
        val top = dsl.select(ITINERARY_PLAN.ITINERARY_ID, maxRating)
                .from(ITINERARY_PLAN)
                .join(PLAN_RATING)
                .on(ITINERARY_PLAN.PLAN_ID.eq(PLAN_RATING.ID))
                .groupBy(ITINERARY_PLAN.ITINERARY_ID)
                .asTable()

        val topRatingCorrelatedQuery = dsl.select(
                IMAGE.URI,
                ITINERARY_PLAN.ITINERARY_ID
        )
                .from(ITINERARY_PLAN)
                .join(PLAN_RATING)
                    .on(ITINERARY_PLAN.PLAN_ID.eq(PLAN_RATING.ID))
                .join(top)

.on(ITINERARY_PLAN.ITINERARY_ID.eq(top.field(ITINERARY_PLAN.ITINERARY_ID)).and(PLAN_RATING.RATING.eq(to
p.field(maxRating))))
                .leftJoin(PLAN)
                    .on(PLAN.ID.eq(ITINERARY_PLAN.PLAN_ID))
                .leftJoin(IMAGE)
                    .on(IMAGE.ID.eq(PLAN.IMAGE_ID))
                .groupBy(ITINERARY_PLAN.ITINERARY_ID)
                .asTable()

        val query = dsl
                .select(
                        paginationSubquery.field(ITINERARY.ID),
                        paginationSubquery.field(ITINERARY.DATE),
                        ITINERARY_INFO.ITINERARY_NAME.`as`("name"),
                        topRatingCorrelatedQuery.field(IMAGE.URI).`as`("imageUrl"),
                        ITINERARY_INFO.DESCRIPTION,
                        USER.USERNAME.`as`("author"),
                        ITINERARY_RATING.RATING.`as`("rating"),
                        ITINERARY_RATING.RATINGS.`as`("ratings")
                )
                .from(paginationSubquery)
                .join(ITINERARY_INFO)
                    .on(ITINERARY_INFO.ITINERARY_ID.eq(paginationSubquery.field(ITINERARY.ID)))
                .leftJoin(LOCALE)
                    .on(LOCALE.ID.eq(ITINERARY_INFO.LOCALE_ID))
                .join(ITINERARY_RATING)
                    .on(ITINERARY_RATING.ID.equal(paginationSubquery.field(ITINERARY.ID)))
                .leftJoin(USER)
                    .on(USER.ID.eq(paginationSubquery.field(ITINERARY.AUTHOR_ID)))
                .join(topRatingCorrelatedQuery)

.on(topRatingCorrelatedQuery.field(ITINERARY_PLAN.ITINERARY_ID).eq(paginationSubquery.field(ITINERARY.I
D)))
                .where(ITINERARY_RATING.RATING.greaterOrEqual(3.0).and(LOCALE.ISO_CODE.eq(lang)))
                .orderBy(paginationSubquery.field(ITINERARY.ID))

        val results = query.fetchInto(ItineraryDto::class.java)

        return Result(
                items = results,
                page = page,
                pageSize = pageSize
        )
```

Figura 34 - Obtención de los datos de una lista de itinerarios más populares

7.2.2 Creación de un itinerario

El proceso de creación de un itinerario solo está disponible para usuarios que se hayan registrado. Es un proceso por etapas.

En la primera etapa de la creación se permite al usuario establecer que día quiere realizar dicho itinerario y a partir de que ahora desea comenzar. Tras seleccionar estos datos se le muestra al usuario cuales son los planes recomendados para ese día a esa hora, como se observar en la Figura 35. Si el usuario desease seleccionar algún plan de esos, se añadirían a una lista donde se almacenarán todos los planes seleccionados por el usuario para que pueda personalizar al máximo el itinerario.

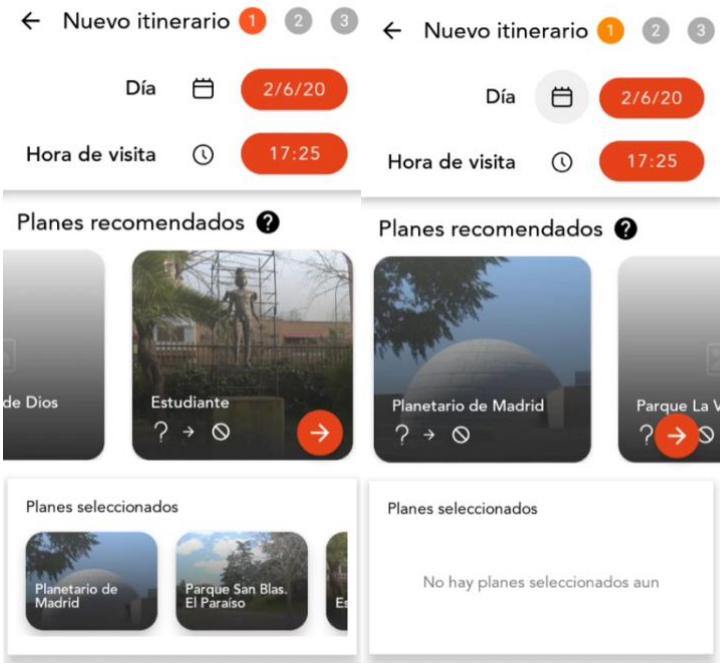
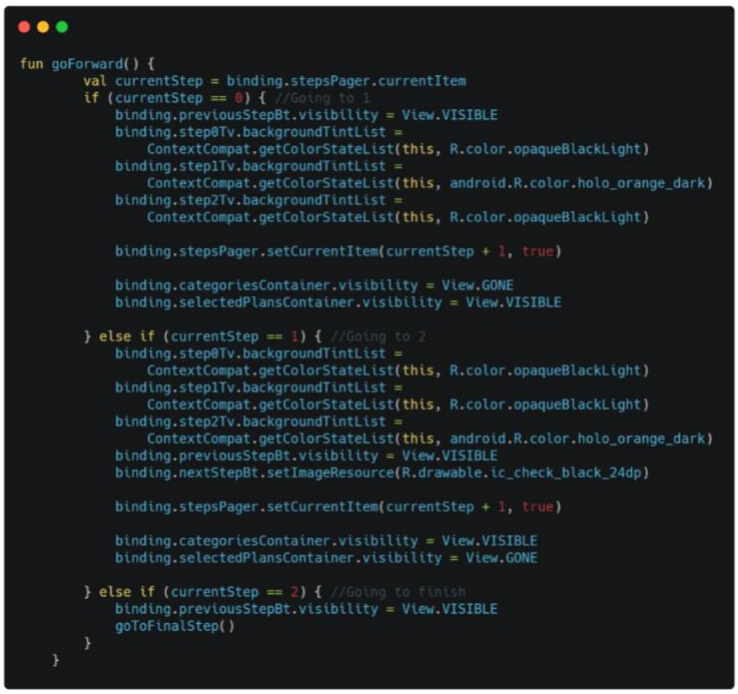


Figura 35 - Crear itinerario paso 1 añadir planes recomendados

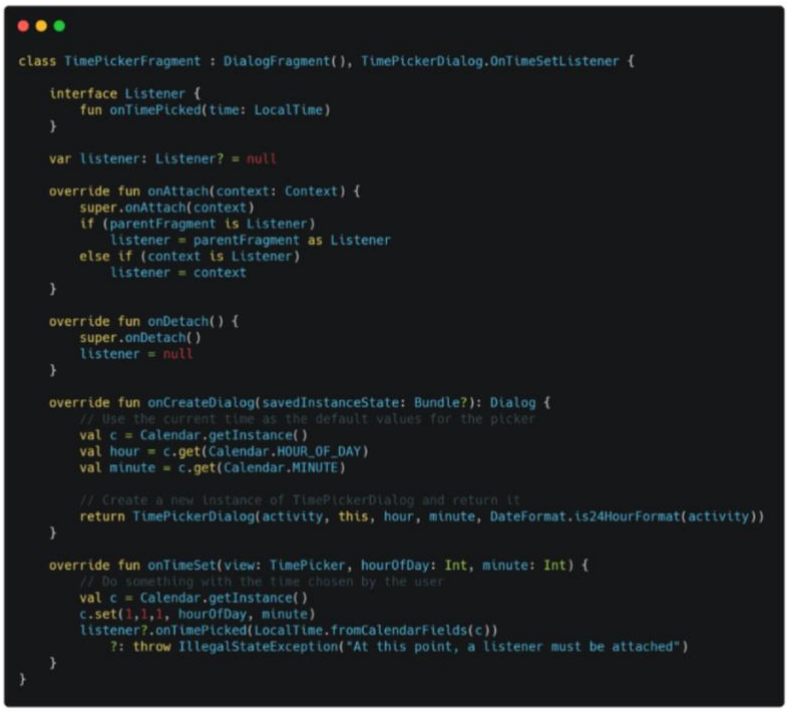
Para la implementación de esta funcionalidad se optó por usar un *view pager*, que permite la transición entre vistas durante el proceso entero de creación de un itinerario. Para la transición entre vistas se implementó una función que actualizase datos y se encargara del cambio de vista, como se observa en la Figura 36.



```
fun goForward() {
        val currentStep = binding.stepsPager.currentItem
        if (currentStep == 0) { //Going to 1
            binding.previousStepBt.visibility = View.VISIBLE
            binding.step0Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, R.color.opaqueBlackLight)
            binding.step1Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, android.R.color.holo_orange_dark)
            binding.step2Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, R.color.opaqueBlackLight)

            binding.stepsPager.setCurrentItem(currentStep + 1, true)

            binding.categoriesContainer.visibility = View.GONE
            binding.selectedPlansContainer.visibility = View.VISIBLE

        } else if (currentStep == 1) { //Going to 2
            binding.step0Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, R.color.opaqueBlackLight)
            binding.step1Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, R.color.opaqueBlackLight)
            binding.step2Tv.backgroundTintList =
                ContextCompat.getColorStateList(this, android.R.color.holo_orange_dark)
            binding.previousStepBt.visibility = View.VISIBLE
            binding.nextStepBt.setImageResource(R.drawable.ic_check_black_24dp)

            binding.stepsPager.setCurrentItem(currentStep + 1, true)

            binding.categoriesContainer.visibility = View.VISIBLE
            binding.selectedPlansContainer.visibility = View.GONE

        } else if (currentStep == 2) { //Going to finish
            binding.previousStepBt.visibility = View.VISIBLE
            goToFinalStep()
        }
    }
```

Figura 36 - Gestión de cambio de página de un view pager

Para la implementación de la hora de inicio y fin del itinerario, así como del día de su realización, se optó por utilizar los widgets propios de Android, Figura 37 y Figura 38 se implementó también las funciones correspondientes para el control de errores como seleccionar una fecha anterior a la actual, elegir hora final anterior a una hora inicial, o elegir horas anteriores, tal como se observa en la Figura 39.



```
class TimePickerFragment : DialogFragment(), TimePickerDialog.OnTimeSetListener {

    interface Listener {
        fun onTimePicked(time: LocalTime)
    }

    var listener: Listener? = null

    override fun onAttach(context: Context) {
        super.onAttach(context)
        if (parentFragment is Listener)
            listener = parentFragment as Listener
        else if (context is Listener)
            listener = context
    }

    override fun onDetach() {
        super.onDetach()
        listener = null
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        // Use the current time as the default values for the picker
        val c = Calendar.getInstance()
        val hour = c.get(Calendar.HOUR_OF_DAY)
        val minute = c.get(Calendar.MINUTE)

        // Create a new instance of TimePickerDialog and return it
        return TimePickerDialog(activity, this, hour, minute, DateFormat.is24HourFormat(activity))
    }

    override fun onTimeSet(view: TimePicker, hourOfDay: Int, minute: Int) {
        // Do something with the time chosen by the user
        val c = Calendar.getInstance()
        c.set(1,1,1, hourOfDay, minute)
        listener?.onTimePicked(LocalTime.fromCalendarFields(c))
            ?: throw IllegalStateException("At this point, a listener must be attached")
    }
}
```

Figura 37 - Time picker de Android

```
class DatePickerFragment : DialogFragment(), DatePickerDialog.OnDateSetListener {

    interface Listener {
        fun onDatePicked(date: LocalDate)
    }

    var listener: Listener? = null

    override fun onAttach(context: Context) {
        super.onAttach(context)
        if (parentFragment is Listener)
            listener = parentFragment as Listener
        else if (context is Listener)
            listener = context
    }

    override fun onDetach() {
        super.onDetach()
        listener = null
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        return DatePickerDialog(activity!!, this, year, month, day)
    }

    override fun onDateSet(view: DatePicker, year: Int, month: Int, day: Int) {
        val c = Calendar.getInstance()
        c.set(year, month, day)
        listener?.onDatePicked(LocalDate.fromCalendarFields(c))
            ?: throw IllegalStateException("At this point, a listener must be attached")

    }
}
```

Figura 38 - Date picker de android

```
            Snackbar.make(
                binding.root,
                "La hora de inicio debe ser menor de la de fin",
                Snackbar.LENGTH_SHORT
            ).show()
            return
```

Figura 39 - Control de errores de horarios

Para que el usuario pudiera ver los planes que había seleccionado se creó un *RecyclerView* que fuese llenándose con pulsar en un plan, como se observa en la Figura 40.

```
override fun onSmartPlanClick(plan: SmartPlan) {
        SelectedPlansBag.add(
            plan = SelectedPlan(
                id = plan.id,
                name = plan.name,
                imageUrl = plan.imageUrl,
                businessHours = plan.week.firstOrNull()?.businessHours ?: emptyList(),
                iniHour = SelectedPlansBag.last()?.endHour ?: LocalTime.now(),
                endHour = SelectedPlansBag.last()?.endHour?.plusHours(1) ?: LocalTime.now()
                    .plusHours(1)
            )
        )
        binding.selectedPlansRv.smoothScrollToPosition(selectedPlansAdapter.itemCount - 1)
    }
```

Figura 40 – Añadir planes recomendados a la mochila de planes seleccionados

En el segundo paso se permite al usuario seguir añadiendo planes a esa lista con los planes seleccionados y para ello, tal como se observa en la Figura 41, se

permite al usuario ver todos los planes que existen según la categoría por la que desee filtrar.

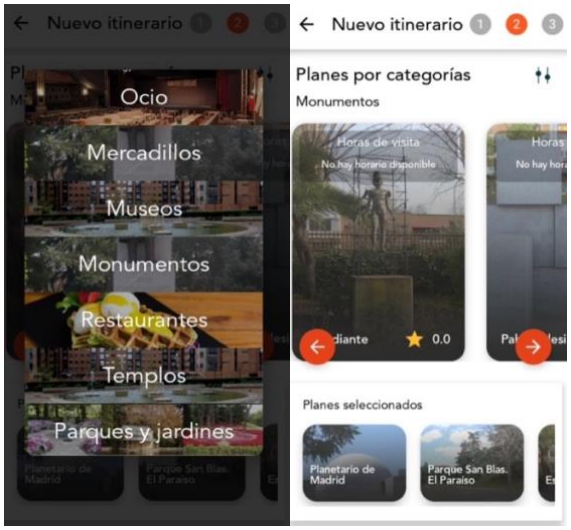


Figura 41 – Crear itinerario paso 2 añadir planes por categorías

En esta funcionalidad el proceso de mostrar a nivel de código los planes filtrados por categorías es exactamente igual que en la primera etapa, consumir en servicio dado por la API y añadir esos datos a un *RecyclerView*.

El tercer y último paso es el de personalización. Se pide al usuario establecer un nombre y una descripción para este itinerario que ha creado. Además, debe seleccionar los intervalos de horas en los que desea realizar cada plan seleccionado. Por defecto, estos se mostrarán por orden de elección y el comienzo de cada plan coincidirá con el final del plan anterior a excepción del primer plan que tendrá la hora inicial del itinerario seleccionada en el primer paso, tal como se observa en la Figura 42

Figura 42 - Crear itinerario, selección nombre y descripción

A nivel de código para la implementación de este caso de uso también se realizó como en las anteriores etapas, consumiendo un servicio y creando sus listas correspondientes, sin embargo, en la API se tuvo que crear un nuevo *endpoint* para la creación del itinerario.

Al darle al botón de enviar que está en la Figura 42Figura 43, sino ha habido ningún problema, el caso de uso crea un itinerario vacío al que le asigna un identificador autogenerado, posteriormente añade la información del itinerario como puede ser el nombre y su descripción, el usuario que lo creó y el idioma. Después comprueba que no haya planes erróneos y los añade al itinerario que posteriormente será insertado en la base de datos, como se observa en la Figura 43.

```
override fun invoke(request: Request): Response {
        val itineraryId = ItineraryId(uuidGenerator.create())
        val date = LocalDate.parse(request.date)

        val author = userRepository[UserId(request.authorId)]
                ?: throw UsernameNotFoundException("Author was not recognized")

        val itinerary = Itinerary(
                id = itineraryId,
                date = date,
                author = author
        )

        request.info.forEach {
            val name = ItineraryName(it.name)
            val desc = ItineraryDescription(it.description)
            val locale = localeRepository[it.lang]
                    ?: throw LocaleNotFoundException("Locale '${it.lang}' was not found")

            itinerary.addInfo(
                    description = desc,
                    name = name,
                    locale = locale,
                    itineraryRepository = itineraryRepository,
                    user = author
            )
        }

        if(request.categoriesIds.isEmpty())
            throw NotCategorizedItineraryException("Itinerary must be at least in one category")

        request.categoriesIds.forEach {
            val category = categoryRepository[CategoryId(it)]
                    ?: throw CategoryNotFoundException("Category $it does not exist")

            itinerary.includeIn(category,itineraryRepository)
        }

        if(request.plans.isEmpty())
            throw EmptyItineraryException("Itinerary must have at least 1 plan")

        request.plans.forEach {
            val plan = planRepository[PlanId(it.idPlan)]
                    ?: throw PlanNotFoundException("Plan ${it.idPlan} does not exist")

            val ini = LocalTime.parse(it.isoIniHour)
            val end = LocalTime.parse(it.isoEndHour)

            itinerary.add(ini = ini, end = end, plan = plan, user = author)
        }

        itineraryRepository += itinerary
        itineraryRatingRepository += ItineraryRating(itinerary)

        domainEventBus.send(itinerary.pullEvents())

        return Response(itinerary.id.toString())
    }
```

Figura 43 – Creación de un itinerario en la API

7.2.3 Geolocalización

Como bien se ha mencionado con anterioridad una de las premisas de esta aplicación es que el usuario pueda tener información de manera sencilla y completa, por ello se optó por utilizar geolocalización para que el usuario pueda ver cuáles son los planes que más cerca tiene.

Como se ve en la Figura 44, se muestra un mapa con todos los planes cercanos al usuario, además, se muestra una lista que contiene información de cada plan mostrando la distancia a la que este está, su nombre, la valoración y un botón de información que abre una vista con el detalle del plan, de la que hablaremos posteriormente.

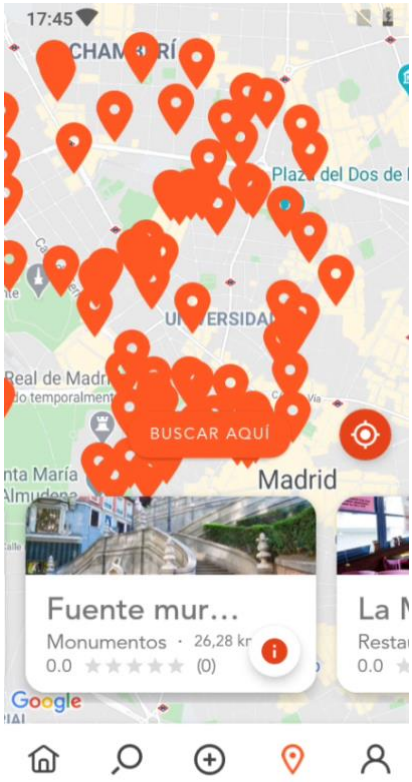


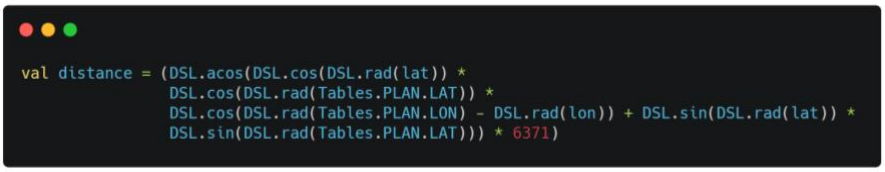
Figura 44 - Mapa interactivo para buscar planes cerca de ti

Para la implementación de esta funcionalidad en el lado del cliente se optó por utilizar la API de Google Maps. Lo primero que hace es preguntar al usuario sí quiere activar la localización o no, al activarla te centra la vista en su posición en el mapa. Posteriormente se pintan los puntos del mapa obtenidos por servicio y se muestra una lista de planes cercanos como se observa en la Figura 45.

```
private fun setUpMap(){
        markerIcon = this.context!!.getDrawable(R.drawable.ic_marker)?.let {
DisplayUtils.bitmapDescriptorFromVector(it) }
        enableLocationButton()
        if (locationPos != null)
            mlastKnownLocation = locationPos!!
        setCameraPosition(mlastKnownLocation, MapsSettings.ZOOM_LEVEL_DEFAULT)
        binding.mapPetitionButton.setOnClickListener {
            page = 1
            googleMap.cameraPosition.target
            getPoints(googleMap.cameraPosition.target)
            getNewPlaces(googleMap.cameraPosition.target, page)
            mlastPetitionLocation = googleMap.cameraPosition.target
            binding.mapPetitionButton.isEnabled = false
            binding.mapPetitionButton.visibility = View.GONE
        }
        googleMap.setOnCameraIdleListener {
            if(DistanceCalculator.distance(
                    mlastKnownLocation.latitude,
                    mlastKnownLocation.longitude,
                    googleMap.cameraPosition.target.latitude,
                    googleMap.cameraPosition.target.longitude,
                    "K") > 1)
            {
                binding.mapPetitionButton.isEnabled = true
                binding.mapPetitionButton.visibility = View.VISIBLE
            }
        }
    }
```

Figura 45 - Inicializacion del mapa

En cuanto a la API, se hizo uso de la fórmula de Harvesine [28] que actualmente es la más precisa ya que permite calcular la menor distancia entre dos puntos usando la latitud y la longitud basándose en la relación entre los lados y ángulos de triángulos esféricos, para poder ordenar los puntos del mapa en función de la distancia como se observa en la Figura 46



```
val distance = (DSL.acos(DSL.cos(DSL.rad(lat)) *
                DSL.cos(DSL.rad(Tables.PLAN.LAT)) *
                DSL.cos(DSL.rad(Tables.PLAN.LON) - DSL.rad(lon)) + DSL.sin(DSL.rad(lat)) *
                DSL.sin(DSL.rad(Tables.PLAN.LAT))) * 6371)
```

Figura 46 - Calculo de la distancia

7.2.4 Detalle de un plan / itinerario

Esta vista ofrece al usuario toda la información relacionada con cada plan e itinerario. Las vistas de ambas opciones son muy similares.

Como vemos en la Figura 47, se muestra el nombre del plan, la distancia a la que se encuentra del plan y a la derecha se muestra entre 1 y 3 iconos en función de la situación meteorológica del momento. El primer icono que se muestra indicará si es de día o es de noche, los siguientes irán en función de situaciones atmosféricas como puede ser lluvia, tormenta, nieve o viento, y después se mostrará la temperatura. También hay un icono que abre Google Maps para que el usuario pueda ver la situación exacta del plan, y otro para que pueda compartirlo con más gente.

Figura 47 - Pantalla detalle plan

Al hacer *scroll* en la vista de detalle del plan se llega a la Figura 48. En esta vista se ofrece al usuario información acerca de los horarios de ese plan y cuál es su estado actualmente, si está abierto o si está cerrado, además, hay un botón que muestra un menú plegable que informa de cuáles son los horarios durante el resto de la semana.

También se muestra un diagrama de barras indicando lo concurrido que está un plan durante toda la semana en las horas de apertura de este, junto con un mensaje informativo al hacer clic en la barra de una hora en concreto.

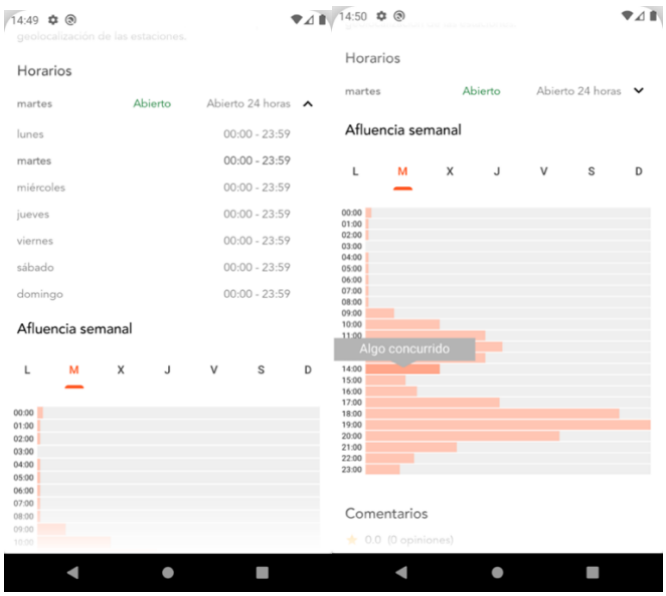


Figura 48 – Horarios y afluencias de un plan

Para la implementación de este diagrama de barras se recurrió a una biblioteca externa, *MPAndroidChart* [29].

El diagrama se llena con datos recibidos por servicio, al eje que contenía las horas se le asigna un color naranja claro y a la hora actual se le asigna un naranja más fuerte con el fin de que el usuario pueda identificar la afluencia de ese plan en ese mismo instante de manera sencilla, como se observa en la Figura 49.

```
data.affluences.reversed().forEach { xAxisLabel.add(it.hour) }
                val xAxis = chart.xAxis
                xAxis.labelCount = data.affluences.size
                xAxis.valueFormatter = IndexAxisValueFormatter(xAxisLabel)


                xAxis.setDrawGridLines(false)
                xAxis.position = XAxis.XAxisPosition.BOTTOM
                xAxis.isEnabled = true
                xAxis.setDrawAxisLine(false)

                val yLeft = chart.axisLeft
                //Set the minimum and maximum bar lengths as per the values that they represent
                yLeft.axisMaximum = 100f
                yLeft.axisMinimum = 0f
                yLeft.isEnabled = false

                //Set label count to 5 as we are displaying 5 star rating
                colors = MutableList(xAxisLabel.size) { Color.parseColor("#FFFFC5B3") }

                //Now add the labels to be added on the vertical axis

                val yRight = chart.axisRight
                yRight.setDrawAxisLine(true)
                yRight.setDrawGridLines(false)
                yRight.isEnabled = false

                //Set bar entries and add necessary formatting
                // setGraphData()

                //Add animation to the graph
                chart.animateY(2000)

                val entries = ArrayList<BarEntry>()

                val dateFormatter = ISODateTimeFormat.hourMinute()
                val now = LocalTime.now(DateTimeZone.forID(timeZone))
                data.affluences.reversed().forEachIndexed { index, affluence ->
                    val affluenceHour = dateFormatter.parseLocalTime(affluence.hour)
                    if (now.hourOfDay == affluenceHour.hourOfDay) {
                        colors[index] = Color.parseColor("#FF5722")
                    }
                    entries.add(BarEntry(index.toFloat(), affluence.percent.toFloat(), affluence))
                }
```

Figura 49 - inserción datos en eje x e y

En el lado del servidor se encontró un problema a la hora de realizar consultas a la base de datos sobre las afluencias que tenían como horario de apertura, por ejemplo, entre las cinco de la tarde y las tres de la mañana del día siguiente, ya que al realizar la *query* daba error porque no reconocía que las tres de la mañana del día siguiente fuese un número mayor que las cinco de la tarde.

La solución fue trabajar con minutos en lugar de con horas, pero surgió otro problema con los planes en los que abrían un domingo y cerraban el lunes de madrugada, puesto que la *query* daba error ya que los minutos en los que se abría el plan eran menores que en los que cerraba debido a que el contador de minutos los lunes a las 00:00 se pone a cero. Para solucionar esto se añadió en el extremo

derecho de la *query* la suma de los minutos totales de la semana más los minutos en los que estuvo abierto el lunes, como bien se puede apreciar en la Figura 51

Por último, al llegar al final del detalle, se muestra la valoración que posee el plan sobre una lista que contiene todos los comentarios que otros usuarios han ido dejando para que la gente pueda hacerse una idea de que opinan acerca del plan.

Posteriormente se muestra otro listado, que contiene una serie de planes relacionados basado en las categorías que el plan tenga, además se le ofrece al usuario la oportunidad de valorar y comentar, como se observa en la Figura 50.

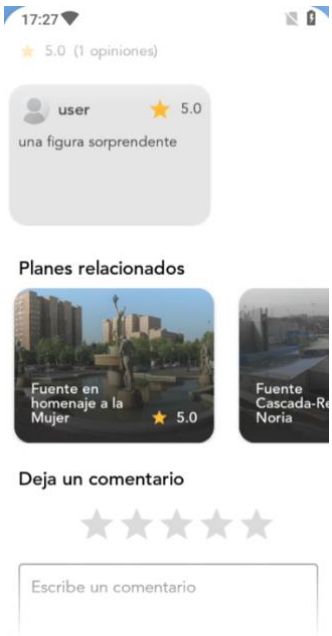


Figura 50 - Comentarios pantalla detalle plan

```
val query = dsl.select(
                PLAN_DAILY_STATS.PLAN_ID,
                PLAN_DAILY_STATS.UPDATED_AT,
                PLAN_DAILY_STATS.DAY,
                HOURS_RANGE.FROM_HOUR,
                HOURS_RANGE.FROM_MINUTES_SINCE_WEEK_START,
                HOURS_RANGE.TO_MINUTES_SINCE_WEEK_START,
                HOURS_RANGE.TO_HOUR,
                INFLUX.HOUR,
                INFLUX.STATS_DAY,
                INFLUX.PERCENT
        )
                .from(PLAN)
                .join(PLAN_DAILY_STATS)
                .on(PLAN.ID.eq(PLAN_DAILY_STATS.PLAN_ID))
                .join(HOURS_RANGE)

.on(PLAN_DAILY_STATS.PLAN_ID.eq(HOURS_RANGE.STATS_PLAN_ID).and(PLAN_DAILY_STATS.DAY.eq(HOURS_RANGE.STAT
S_DAY)))
                .leftJoin(INFLUX)
                .on(PLAN_DAILY_STATS.PLAN_ID.eq(INFLUX.STATS_PLAN_ID).and(

 cond1.or(HOURS_RANGE.TO_MINUTES_SINCE_WEEK_START.greaterOrEqual(10080).and(INFLUX.MINUTES_SINCE_WEEK_S
TART.between(value(0L),HOURS_RANGE.TO_MINUTES_SINCE_WEEK_START.minus(10080))))
                ))
                .where(PLAN_DAILY_STATS.PLAN_ID.eq(planId))
                .orderBy(PLAN_DAILY_STATS.DAY, INFLUX.MINUTES_SINCE_WEEK_START.asc())
```

Figura 51 – Control de afluencias

7.2.5 Buscar Itinerarios y planes

Esta funcionalidad se creó con la idea de dar al usuario una opción de ver otros planes e itinerarios distintos a los ofrecidos en el *dashboard*, como bien se observa en la Figura 52 al usuario se le da la opción de elegir entre varios parámetros de búsqueda como el nombre del autor, nombre del plan o itinerario, por afluencia, o por fecha. También se le da la opción de realizar la búsqueda por comando de voz en lugar de escribirlo.

Una vez se realiza la búsqueda se muestra un listado con los resultados obtenidos si es que hubiera.

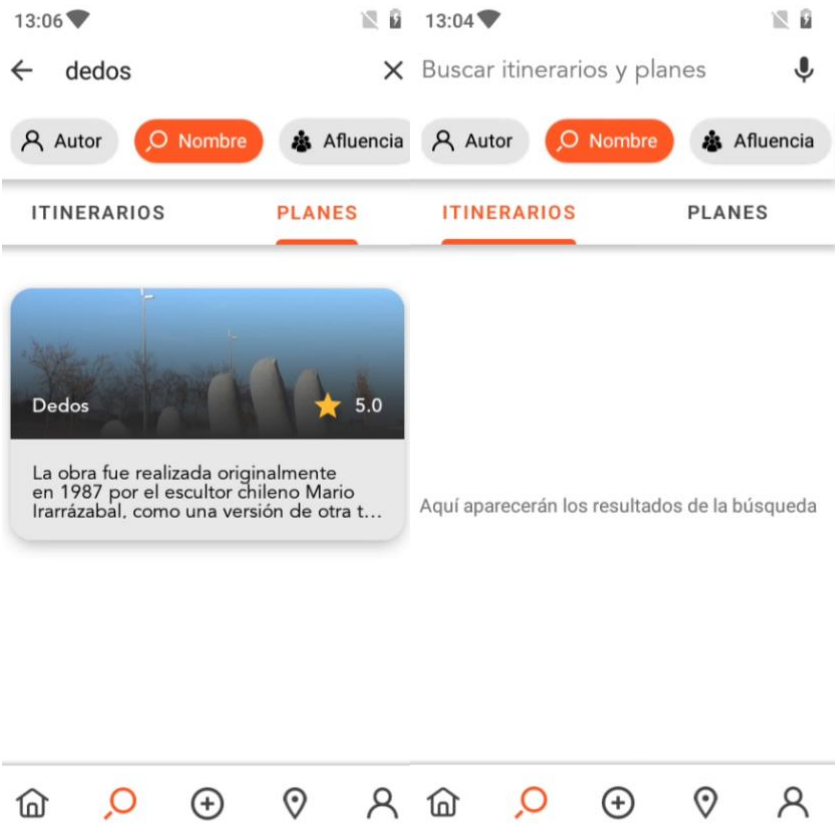
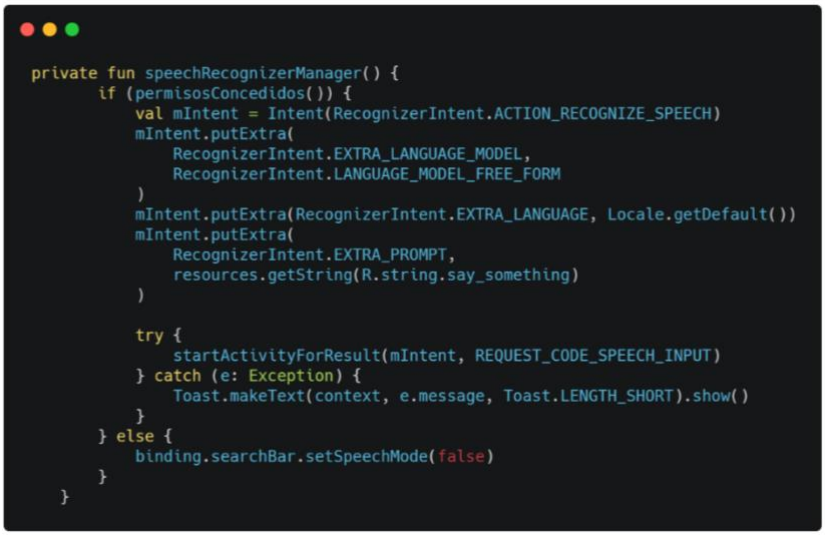


Figura 52 - Caso de uso de búsqueda

En la parte del cliente se elaboró un recycler view que mostrase todos los planes e itinerarios encontrados al realizar la búsqueda. Para el reconocimiento de voz se pide primero al usuario permiso para activarlo, si decide activarlo se crea un mánager que se encarga de gestionar el texto que le dicta el usuario y escribirlo en la barra de búsqueda como se observa en la Figura 53



```
private fun speechRecognizerManager() {
        if (permisosConcedidos()) {
            val mIntent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
            mIntent.putExtra(
                RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM
            )
            mIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault())
            mIntent.putExtra(
                RecognizerIntent.EXTRA_PROMPT,
                resources.getString(R.string.say_something)
            )

            try {
                startActivityForResult(mIntent, REQUEST_CODE_SPEECH_INPUT)
            } catch (e: Exception) {
                Toast.makeText(context, e.message, Toast.LENGTH_SHORT).show()
            }
        } else {
            binding.searchBar.setSpeechMode(false)
        }
    }
```

Figura 53- Reconocedor de voz

7.3 Desarrollo del algoritmo de planes recomendados

La funcionalidad de planes recomendados permite a un usuario conocer cuáles son los planes más adecuados para acudir en ese mismo momento.

Para la toma de decisiones, se ha hecho diseñado e implementado un algoritmo ID3 [30]. Dentro de este algoritmo, existen una serie de conceptos asociados al mismo:

- *Atributo*: son aquellos factores que influyen en la decisión.
- *Ejemplo*: combinación de datos en pares factor-valor que son utilizados para entrenar el algoritmo.
- *Clase*: los posibles valores en la toma de una decisión.

El algoritmo genera un árbol de decisión a partir de los ejemplos proporcionados. El árbol es recorrido desde la raíz hasta un nodo hoja, decidiendo en cada paso por cuál de los nodos hijos avanzar. El avance a un nodo hijo u otro dependerá del valor que tome el atributo de la muestra que se esté clasificando. El algoritmo finaliza al alcanzar un nodo hoja, que corresponderá con la clase a la que se asocia la muestra.

Para el análisis de la conveniencia de realizar o no un plan, se tienen en cuenta los siguientes factores: meteorología, afluencia de personas, naturaleza del lugar y horarios de apertura, que han sido utilizados como atributos del algoritmo.

Por otra parte, algunos de los ejemplos utilizados para el entrenamiento del algoritmo pueden observarse en la Figura 54. Estos contienen todas las posibles combinaciones de los valores posibles de cada factor y la decisión a tomar en base a cada una de estas combinaciones.

```
Opening times;Weather;Influx;Nature;Decision
Open;Snow;Empty;Outdoor;No
Open;Snow;Empty;Indoor;Yes
Open;Snow;Half;Outdoor;No
Open;Snow;Half;Indoor;Yes
Open;Snow;Full;Outdoor;No
Open;Snow;Full;Indoor;No
Open;Rain;Empty;Outdoor;No
Open;Rain;Empty;Indoor;Yes
Open;Rain;Half;Outdoor;No
Open;Rain;Half;Indoor;Yes
Open;Rain;Full;Outdoor;No
Open;Rain;Full;Indoor;No
Open;Wind;Empty;Outdoor;Yes
Open;Wind;Empty;Indoor;Yes
Open;Wind;Half;Outdoor;Yes
Open;Wind;Half;Indoor;Yes
Open;Wind;Full;Outdoor;No
Open;Wind;Full;Indoor;No
Open;Sun;Empty;Outdoor;Yes
Open;Sun;Empty;Indoor;Yes
Open;Sun;Half;Outdoor;Yes
```

Figura 54 Algunos de los ejemplos utilizados en el algoritmo ID3

Los datos que utiliza el algoritmo son dependientes de la hora y el día de consulta, y es posible que no todos estén disponibles. Esto significa que cabe la posibilidad de que para un día y hora dados no se posean datos de la afluencia de personas en ese lugar, que se desconozca si está abierto o no, e incluso que no se posean datos meteorológicos por la lejanía respecto al día actual.

En todos los casos, el algoritmo informará sobre su decisión dando alguna de las siguientes respuestas: sí, no o sin determinar. El último de los valores indica que no se posee la suficiente información como para dar un veredicto.

A continuación, en la Figura 55, se observan dos ejemplos en los que el algoritmo ha dado un veredicto concluyente. En la parte derecha de la ilustración, el veredicto es sí, debido a que en el momento de consultar el lugar está abierto, no tiene apenas afluencia de personas y la meteorología no es adversa (el cielo está nublado). Sin embargo, en la parte izquierda se puede observar que el veredicto es no, siendo el motivo la inviabilidad de acudir a visitarlo puesto que se encuentra cerrado en ese momento.

Figura 55 -Se muestra la decisión y tomada y porqué se ha llegado a ella

En cualquier otro caso, el algoritmo concluye que no dispone de los datos suficientes como para dar un veredicto y por tanto no es posible determinar si es o no recomendable acudir al lugar.

Las razones de un veredicto no concluyente son siempre motivadas por el hecho de falta de información respecto al lugar. Por ejemplo, en el caso que se muestra en la Figura 56, aunque se conoce que el lugar está abierto porque se dispone de los horarios de apertura y cierre, se desconoce la afluencia de personas, haciendo imposible dar un veredicto.

Figura 56 - En ocasiones no existe información suficiente para recomendar un plan y se asigna como no concluyente

Por último, se ha de reseñar que debido a la emergencia sanitaria por razón de la COVID-19 [31] que acontece en España en el momento en el que se escribe este documento, se ha incluido como razón de un veredicto negativo (no), el cierre temporal del lugar.

Como se explicará en los puntos que aparecen a continuación, todos los datos que alimentan este algoritmo son extraídos en procesos programados que actualizan los datos cada cierto número de horas o días. La información extraída por estos procesos es guardada en la base de datos del sistema para su posterior uso por el algoritmo.

7.4 Desarrollo del buscador de horarios y afluencias

Todos los planes que se muestran en la aplicación disponen de una vista de detalle en la que es posible visualizar los datos asociados a este. Entre estos datos se encuentran los horarios de apertura y la afluencia de personas a lo largo de la semana.

Tal y como se muestra en la Figura 57, al acceder a un plan, es posible visualizar la afluencia desglosada en días y horas además de los horarios de apertura.

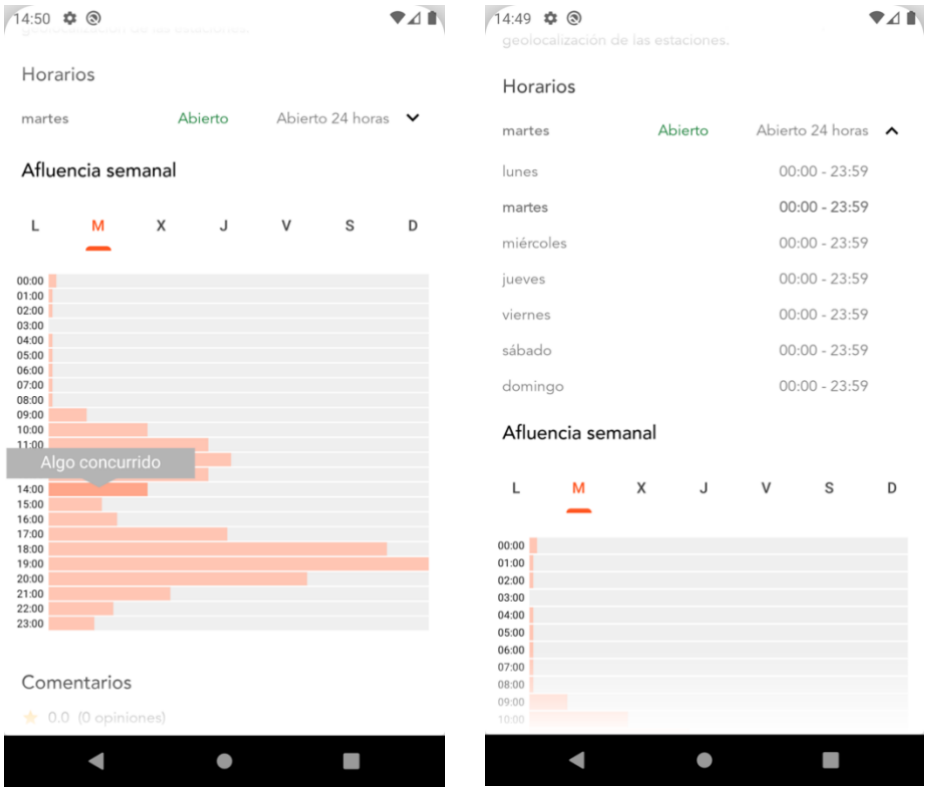


Figura 57 - Los horarios de apertura y las afluencias se muestran desglosados a lo largo de la semana

Todos estos datos son obtenidos a través de Google [32], que, aunque no dispone de una API para su acceso, sin embargo, sí se puede acceder usando el buscador. En este sentido, los datos son obtenidos a través de un proceso automatizado responsable de realizar una búsqueda en Google y extraer los resultados de la información recuperada.

La información extraída se utiliza para la visualización desglosada en la vista de detalle, y como fuente de datos del algoritmo ID3 mencionado en el punto anterior. A continuación, se describe en profundidad como se ha elaborado este algoritmo de automatización y que restricciones se ha necesitado tener en cuenta.

El proceso automático que extrae información de una página web simulando ser un humano, es una técnica conocida como Web *Scraping* [34]. Normalmente, esta clase de procesos son bloqueados estableciendo un límite de peticiones en un periodo de tiempo. En el caso de Google no es posible conocer las técnicas que utiliza que utiliza para ello, pero existe información empírica [35] de diferentes fuentes que avalan la limitación de un número de peticiones por hora.

En el momento de redactar este documento, la aplicación contiene alrededor de 1500 planes registrado en la base de datos. Esto supone un gran número de peticiones para obtener la información de horarios y afluencias, ya que es necesario realizar una búsqueda por cada uno de ellos.

Teniendo en cuenta lo anterior, se ha implementado una tarea que de forma iterativa y continua realice una búsqueda automatizada. Para evitar que el buscador bloquee las peticiones provenientes del sistema al alcanzar el límite, se ha hecho uso de servidores proxy [33] y un *broker* de mensajería.

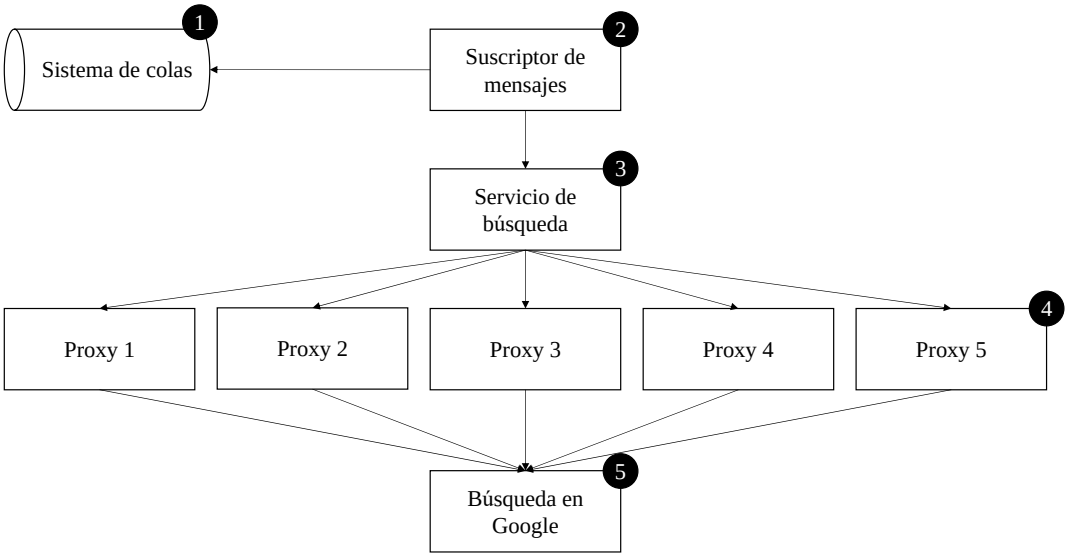


Figura 58 - Esquema general del sistema de búsqueda de horarios y afluencias

Para explicar de forma visual cómo funciona el sistema, a continuación, se describen de forma ordenada los puntos que aparecen numerados en la Figura 58:

1. **Sistema de colas**
   Cuando es insertado un mensaje en la cola dedicada a los mensajes del sistema de búsqueda, el suscriptor es notificado del nuevo mensaje e inicia el proceso. El mensaje no es eliminado de la cola hasta que este es procesado de forma correcta, y en caso de error, se re-encola de nuevo.

2. **Suscriptor de mensajes**
   El consumidor recibe los mensajes de forma secuencial, y se garantiza que el mensaje será consumido de forma correcta al menos una vez. En este sentido, como se garantiza que los mensajes serán consumidos al menos una vez, es necesario contemplar el consumo de mensajes duplicados. Para ello, cuando el consumidor recibe un mensaje, lo añade a una tabla de mensajes consumidos que hace posible consumir una única vez cada mensaje.

3. **Servicio de búsqueda**

   Antes de iniciar la búsqueda, el sistema debe buscar un servidor *proxy* al que conectarse para realizar la búsqueda en Google. Actualmente, se dispone de cinco servidores *proxy*, limitados a 10 peticiones de búsqueda por hora cada uno, para evitar que sean bloqueados por el buscador.

   Al estar estos servidores limitados a un número de peticiones por hora, es necesario interrumpir el proceso de búsqueda alcanzado el límite (5 servidores *proxy* x 10 peticiones/hora = 50 peticiones/hora).

   Para obtener conexión a un servidor *proxy* que no haya consumido todas las peticiones disponibles, el servicio de búsqueda hace uso de otro componente que controla las peticiones realizadas a través de cada servidor y el tiempo entre estas.

4. **Rotador de *proxies***

   El componente mencionado en el párrafo anterior contiene una cola de prioridad [37] que ordena los datos de los servidores *proxy* según el número de peticiones realizadas y el tiempo transcurrido desde la última vez que fue utilizado. De esta forma, cada nueva conexión que es creada, siempre se realiza mediante el servidor con más peticiones disponibles por realizar.

   Cuando todos los servidores alcanzan el límite de peticiones y no ha transcurrido al menos una hora desde el último uso, es necesario que el servicio de búsqueda quede bloqueado hasta cumplir el tiempo mínimo (la diferencia entre el tiempo transcurrido entre la última vez que fue usado y la hora actual). El hilo de ejecución queda bloqueado durante el tiempo mencionado antes de continuar el flujo de ejecución, garantizando así que no se excede del número de peticiones. Se puede observar un ejemplo de código en la Figura 59.

```
val proxy = proxies.peek().copy()
if (proxy.usages >= 10) {
    val lastUsage = proxy.lastUsage
    val diff = Duration.between(lastUsage, LocalDateTime.now())
    if (diff < Duration.ofHours(1)) {
        val waitingTime = Duration.ofHours(1).minus(diff)
        Thread.sleep(waitingTime.toMillis())
    }
    proxy.usages = 0
}

proxy.usages++
proxy.lastUsage = LocalDateTime.now()

synchronized(proxies) {
    proxies.remove()
    proxies.add(proxy)
}
```

Figura 59 - Fragmento del algoritmo rotador de proxies

5. **Búsqueda en Google**
   Por último, cuando el servicio de búsqueda adquiere una conexión mediante servidor *proxy*, realiza una búsqueda en Google extrayendo del código HTML los datos de los horarios, si está o no cerrado temporalmente y las afluencias de personas del lugar.

   En ocasiones no se dispone de ninguno de los datos o solamente existe uno de ellos, en cualquier caso, son almacenados en la base de datos una vez extraídos.

   Para la extracción de los datos del código fuente de la página de resultados de búsqueda, ha sido necesario estudiar la estructura de la página web y extraer mediante recorrido del árbol de elementos, cada uno de los datos. Este algoritmo está sujeto a cambios que puedan producirse por parte del buscador y por tanto es una parte sensible, que previsiblemente requerirá de ser modificada en el tiempo.

7.5 Desarrollo de los proveedores de planes

El catálogo de planes de la aplicación está diseñado para contemplar la adición de nuevas ciudades con el paso del tiempo y que no sólo los administradores sean los que añadan nuevos planes, sino que también puedan hacerlo los usuarios.

De momento, todos los planes son extraídos de la API de datos abiertos que provee el Ayuntamiento de Madrid [38]. Este catálogo de datos contiene restaurantes, monumentos, museos, lugares de culto, etc. Todos ellos han sido incorporados a la aplicación a través de un proceso de extracción automatizada, que de forma periódica realiza peticiones a las distintas rutas para incorporar nuevos planes.

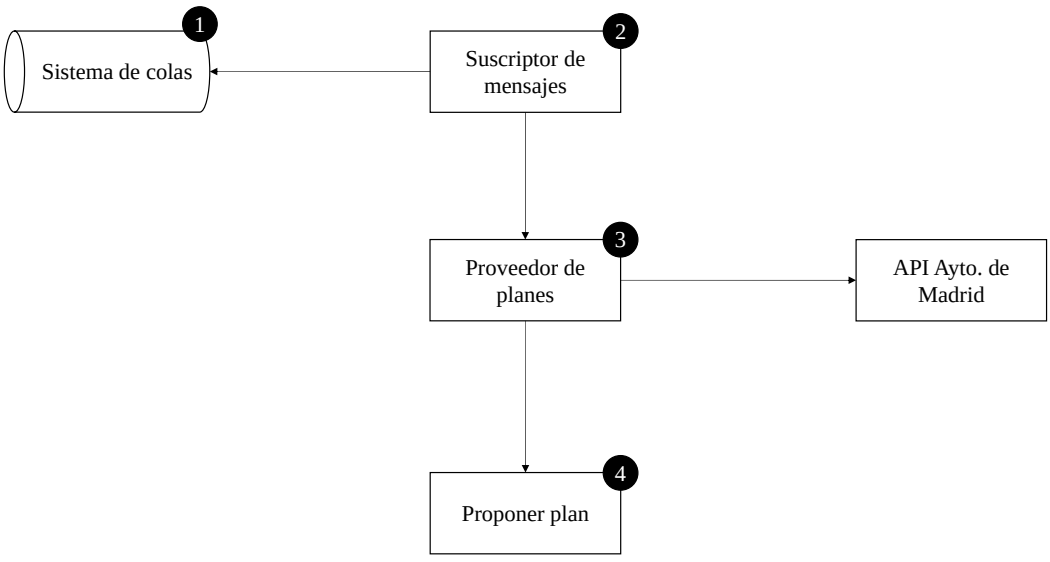


Figura 60 – Alta de planes extraídos de la API del Ayto. de Madrid

Actualmente existen alrededor de 1500 planes diferentes en el total de las distintas categorías que provee el catálogo de datos abiertos. Cada uno de ellos requiere de la

ejecución de un caso de uso que realiza diferentes acciones: comprobar si ya ha sido incorporado el plan anteriormente, guardar su imagen, incorporarlo a las categorías correspondientes, etc.

Este proceso abarca una duración total de aproximadamente 30 minutos, haciendo que necesariamente deba ser realizado de forma asíncrona para impedir bloquear al cliente que solicite el inicio del proceso.

A continuación, se enumera y explica en qué pasos se divide este proceso, siguiendo el esquema que aparece en la Figura 60:

1. **Sistema de colas**
   Cuando un administrador acciona el refresco de planes, este es enviado al sistema de colas y redirigido a la cola encargada de atender el evento.

2. **Suscriptor de mensajes**
   El suscriptor de la cola de mensajes recoge los mensajes de forma secuencial e inicia el proceso de extracción de los planes, recorriendo de forma iterativa cada uno de los componentes encargados de ello.

   En caso de error, la acción es reencolada y el proceso se inicia de nuevo. Cabe destacar que todos los planes que hubiesen sido dados de alta antes del error no son revertidos, ya que se entiende que hasta el momento de error todos los planes han verificado las pertinentes restricciones.

3. **Proveedor de planes**
   Cada componente proveedor de planes es responsable de extraer los planes de un cierto repositorio dentro del catálogo de datos abiertos. Tal y como se aprecia en la Figura 61, estos componentes implementan una misma interfaz que sirve para accionarlos.

   El esquema de funcionamiento de cada uno de estos proveedores de planes se compone de dos pasos: la petición a la API correspondiente mediante HTTP para extraer los planes del proveedor y ejecutar el caso de uso con cada uno de los planes.

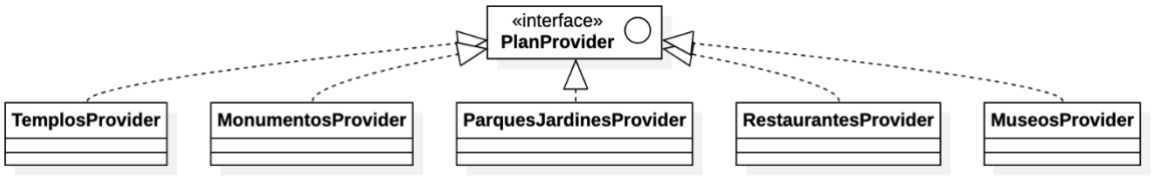


Figura 61 – Diagrama UML de los proveedores de planes

   La inclusión de nuevos proveedores es posible a través de la implementación de nuevas clases que implementen la interfaz *PlanProvider*, de forma que pasarían a formar parte del sistema automatizado de extracción.

4. **Proponer plan**
   El caso de uso es ejecutado de forma transaccional para el alta de cada uno de los nuevos planes. Es importante reseñar la naturaleza transaccional de este, puesto que supone que cada plan es dado de alta de forma independiente al resto del proceso, y el fallo de uno no supone el revertir el resto de ellos.

   Por último, tal y como se describía al principio de este punto, el sistema ha sido diseñado con la posibilidad de que no sólo se puedan integrar nuevos planes a través de mecanismos del lado del sistema, si no, que puedan ser los clientes (humanos o sistemas automáticos) los que incorporen nuevos planes a través de una ruta en la API HTTP [18] de la aplicación, método que se muestra en la Figura 62.

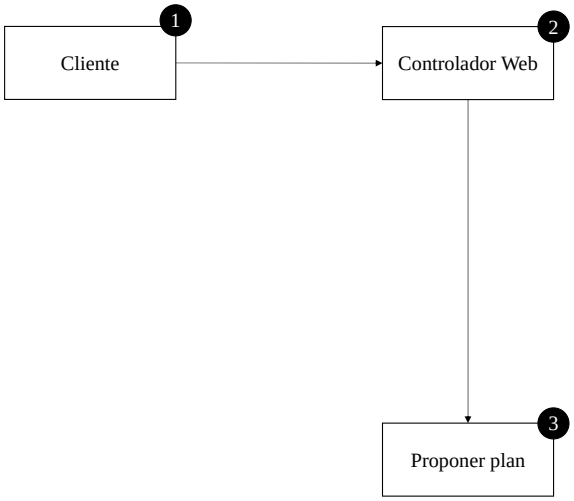


Figura 62 - Método alternativo de alta de planes a través de API REST

7.6 Internacionalización automática de los itinerarios y planes

El público al que va destinado la aplicación son personas que acuden a una ciudad para hacer turismo y quieren planear que lugares visitar. Estas personas pueden ser de cualquier parte del mundo. Por ello, todos los planes e itinerarios del catálogo son traducidos al idioma inglés. De esta forma cualquier persona que quiera consultar la información puede hacerlo tanto en español, como en inglés.

Como se ha explicado anteriormente, los planes que se muestran en la aplicación provienen del catálogo de datos abiertos de la ciudad de Madrid, los cuáles contienen todos sus datos en idioma español.

También es posible que los usuarios de la aplicación den de alta itinerarios en un idioma y por tanto es necesario la traducción al resto de idiomas disponibles (inglés y español).

Para este problema de internacionalización, se ha desarrollado a través del servicio de traducción de Google Cloud, un componente encargado de realizar la traducción de la descripción y nombre de los nuevos planes e itinerarios dados de alta.

Cuando un nuevo plan o itinerario es dado de alta, el sistema lanza un evento que es procesado de forma asíncrona y desencadena la traducción de la información del nuevo plan. A continuación, en la Figura 63 se muestra de forma detallada este proceso:

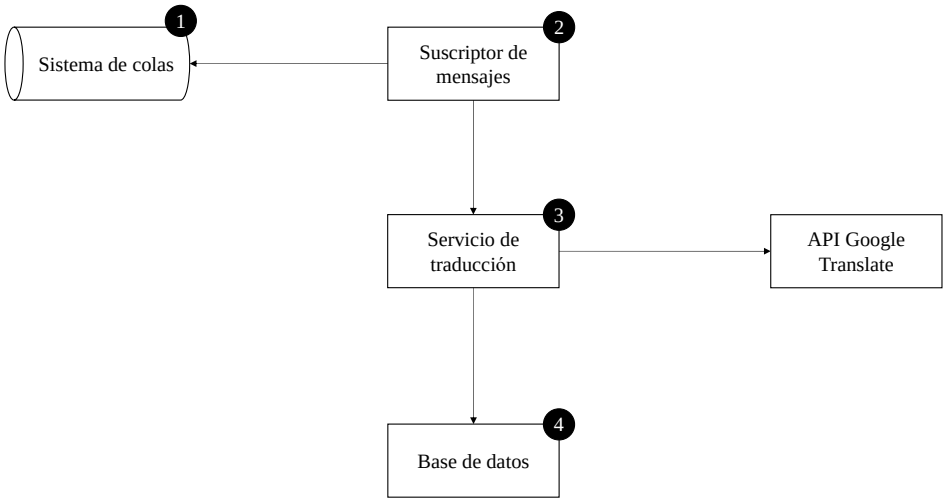


Figura 63 - Proceso de internacionalización

1. **Sistema de colas**
   Cuando un usuario da de alta un nuevo itinerario o se registra un nuevo plan, se desencadena un evento que es enviado al sistema de colas y redirigido a la cola encargada de atender el evento.

2. **Suscriptor de mensajes**
   El suscriptor de la cola de mensajes recoge los mensajes de forma secuencial y extrae la información que existe asociada al plan en la base de datos.

3. **Servicio de traducción**
   La información extraída de la base de datos es utilizada para conocer en que idiomas está disponible la traducción del plan, y en caso de que no esté en todos, se realiza una petición al servicio web de Google Translate [40] para cubrir los idiomas restantes.

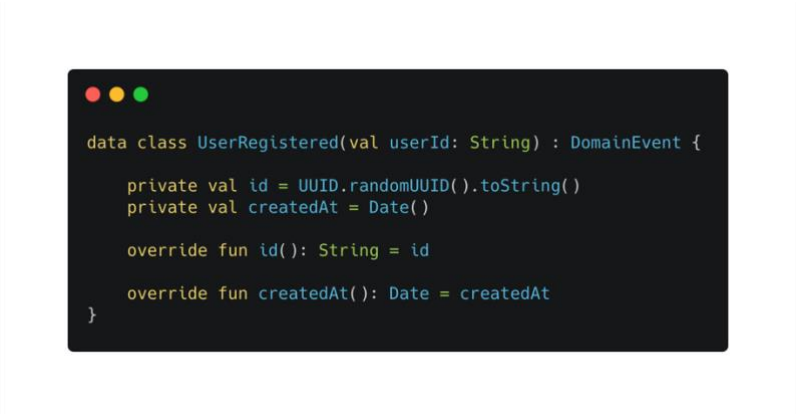
4. **Base de datos**
   Por último, la información traducida a través del servicio web de Google es almacenada en la base de datos, completando así el proceso de internacionalización y haciendo posible que la información pueda ser visualizada en distintos idiomas.

7.7 Desarrollo de la cola y suscripción a eventos

Todas las tareas enunciadas en los puntos 7.4, 7.5 y 7.6 implican el uso de un sistema de colas. Este sistema de colas es el encargado de iniciar de forma asíncrona distintos procesos de la aplicación, de forma que ciertas tareas son siempre encoladas para su ejecución diferida.

RabbitMQ garantiza el envío de mensajes al menos una vez. Esto implica que el propio sistema de colas se encarga de volver a notificar a los suscriptores en caso de error, para reprocesar el mensaje. Esto facilita el manejo de errores en operaciones asíncronas, puesto que no sólo garantiza los reintentos en caso de error, sino que también es capaz de persistir los mensajes para su supervivencia en caso de reinicios.

Las diferentes tareas han sido representadas como diferentes clases suscriptoras a eventos, donde cada evento es un hecho importante para el dominio de la aplicación ocurrido en el pasado. Un ejemplo de estos eventos puede verse en la Figura 64, donde se representa el hecho de que un nuevo usuario se ha registrado.



```
data class UserRegistered(val userId: String) : DomainEvent {

    private val id = UUID.randomUUID().toString()
    private val createdAt = Date()

    override fun id(): String = id

    override fun createdAt(): Date = createdAt
}
```

Figura 64 - Evento generado al registrarse un nuevo usuario

Los suscriptores a estos eventos son notificados por el sistema de colas y desencadenan una acción en respuesta a nuevos mensajes. Por ejemplo, en el caso del nuevo usuario registrado, uno de los suscriptores será encargado de enviar un correo electrónico de bienvenida para que el usuario pueda confirmar su cuenta.

Overview				Messages			Message rates			+/-
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**	
planadvisor.notify_password_recovery_on_password_recover_requested	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.run_plan_providers_on_run_plan_providers_triggered	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.run_plan_stats_refresh_on_run_plan_stats_refresh_triggered	classic	D	idle	395	250	645	0.00/s	0.00/s	0.00/s	
planadvisor.translate_category_info_on_category_created	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.translate_itinerary_info_on_itinerary_published	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.translate_plan_info_on_plan_published	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.update_itinerary_rating_on_itinerary_commented	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.update_weather_forecast_on_weather_forecast_update_requested	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
planadvisor.welcome_on_user_registered	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	

Figura 65 - Colas existentes en rabbitmq

Uno de los problemas que surgen con respecto a garantizar que los mensajes sean ejecutados al menos una vez es el manejo de mensajes duplicados. Es posible que un mensaje que ha sido procesado de forma correcta por el suscriptor no llegue a ser notificado al sistema de colas y que por tanto este interprete que ha fallado y que debe ser reencolado, provocando la duplicidad de la ejecución del suscriptor del mensaje.

Para abordar este problema, todos los suscriptores a una cola de mensajes almacenan que mensajes han sido procesados, de forma que, al recibir un nuevo mensaje, comprueban si ya había sido procesado. En este caso, el mensaje es descartado sin notificar error, puesto que se asume que la duplicidad de los mensajes es una casuística esperada.

```
fun handle(event: E) {
    val duplicate = deduplier?.isAlreadyHandled(event,this) ?: false
    if(!duplicate) {
        deduplier?.handled(event,this)
        on(event)
    }
    else {
        println("[${this.javaClass.simpleName}] Duplicate ignored ${event.id()}")
    }
}
```

Figura 66 - Los suscriptores gestionan la posible duplicidad de los mensajes

Otro de los problemas a abordar en la integración de este tipo de sistemas es la inclusión de transacciones distribuidas. Al tratarse la base de datos y el broker de mensajería de dos sistemas diferentes, estos no son capaces de coordinar transacciones, haciendo que no sea posible realizar acciones entre ambos de forma atómica.

Como solución a este problema se ha elaborado una cola de eventos en base de datos que acumula los eventos que son generados durante una transacción de sistema. Así, cuando se confirman, entonces los eventos son persistidos atómicamente con el resto de las acciones que se han desarrollado a la vez que estos.

Esta cola de eventos en base de datos responde a un patrón de diseño conocido como Outbox Pattern [41]. Una vez se ha confirmado la transacción (commit) pasado un tiempo indeterminado, un Message Relay (elemento Nº2 en la Figura 67) extrae los eventos almacenados y los publica en el broker de mensajería, garantizando el envío del evento al sistema de colas al menos una vez.

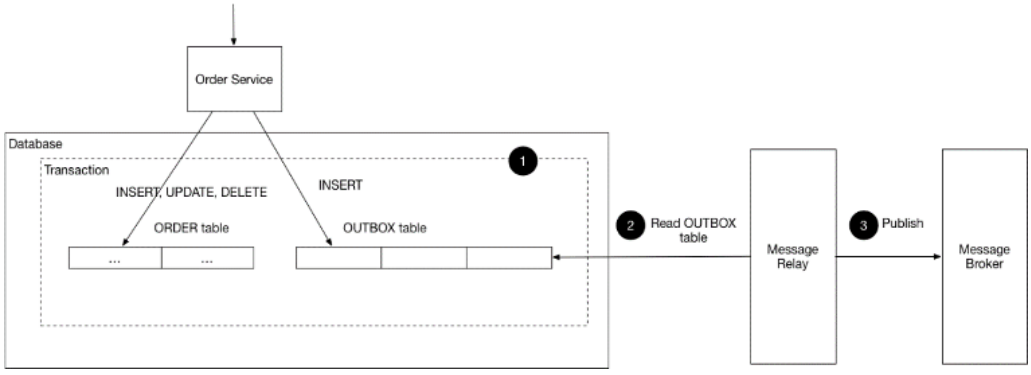


Figura 67 - Esquema general de la cola transaccional de eventos

7.8 Desarrollo de la obtención de datos meteorológicos

Una de las fuentes de datos del algoritmo de recomendaciones explicado en el punto 7.3 son las predicciones meteorológicas. Se utiliza en los casos en que el lugar es un sitio abierto o el clima es adverso, para determinar si es o no adecuado acudir (por ejemplo, en caso de lluvia o nieve).

Para la incorporación en la aplicación de estos datos, existe una tarea programada que periódicamente realiza una petición a la API REST de la AEMET [16] para obtener partes meteorológicos de hasta 7 días posteriores al actual.

```
class UpdateWeatherForecastJob(private val updateWeatherForecast: UpdateWeatherForecast) {

    @Scheduled(cron = "\${jobs.update_weather_forecast.rate_cron}", zone = "Europe/Madrid")
    fun run() { updateWeatherForecast(UpdateWeatherForecast.Request(
            fromIsoDate = LocalDate.now().toString(),
            toIsoDate = LocalDate.now().plusDays(7).toString()
    )) }

}
```

Figura 68 - Tarea programada para obtener datos meteorológicos

Esta tarea es ejecutada cada 6 horas, período tras el que la AEMET actualiza el parte meteorológico. Cabe destacar que esta API tiene un límite de peticiones por minuto que en ningún caso se sobrepasa, puesto que los datos son extraídos tan solo una vez cada 6 horas y almacenados para el uso de la aplicación y sus clientes. A continuación, se explican con más detalle los pasos enumerados en la Figura 69 que describen el proceso completo:

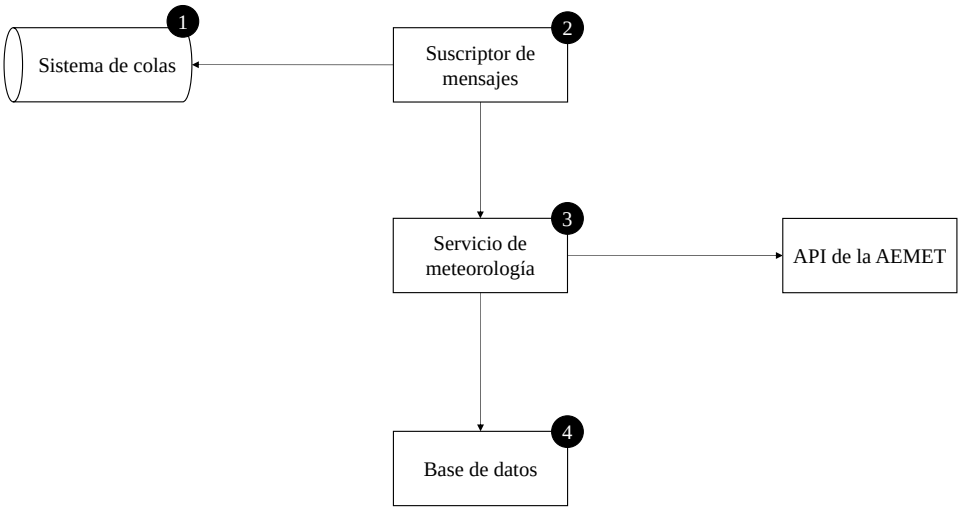


Figura 69 - Esquema general del proceso de extracción de partes meteorológicos

1. **Sistema de colas**
   Cuando la tarea programada o un administrador acciona el refresco de planes, este es enviado al sistema de colas y redirigido a la cola encargada de atender el evento.

2. **Suscriptor de mensajes**
   El suscriptor de la cola de mensajes recoge el mensaje y realiza una petición al servicio de meteorología, que actualmente es la AEMET.

3. **Servicio de meteorología**
   El servicio de la AEMET provee predicciones meteorológicas limitadas en el tiempo, alcanzando como máximo hasta los 7 días posteriores a la fecha actual.

   El componente encargado de la comunicación con la API de la AEMET además interpreta los datos y los transforma para adaptarlos al formato en el que se guardan en la base de datos del sistema.

4. **Base de datos**
   La información acerca de las previsiones meteorológicas queda guardada por horas y de forma normalizada en la base de datos hasta la siguiente ejecución de la tarea programada.

7.9 Desarrollo y documentación de API REST

En este apartado se explicará el diseño de la API REST. La API está formada por cinco grupos: Planes, Itinerarios, Mapa, Usuarios, Categorías, Errores, Backoffice, Imágenes, Recomendaciones inteligentes y Autenticación, que a su vez están formados por varios endpoints.

7.9.1 Planes

En las tablas que se muestran a continuación se establece la relación entre cada endpoint utilizado en el grupo de planes y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en las Tabla 33 y Tabla 34

Endpoint	GET	POST	PUT	DELETE
api/plans/{planId}	X			
api/plans/popular	X			
api/plans/{id}/stats	X			
api/plans/{id}/comments	X	X		
api/plans/search	X			
api/plans/near	X			
api/plans/{planId}/related	X			
api/plans/near/points	X			

Tabla 33 - Tipos de endpoints usados en planes

Descripción de cada endpoint

Endpoint	Descripción
api/plans/{planId}	Obtención de los datos de un plan
api/plans/popular	Obtención de los datos de un plan popular
api/plans/{id}/stats	Obtención de horarios y afluencia de un plan
api/plans/{id}/comments	Obtención e inserción de los comentarios de un plan
api/plans/search	Buscar planes según un filtro en la bbdd
api/plans/near	Obtener planes cercanos
api/plans/{planId}/related	Obtener todos los planes relacionados con un plan dado
api/plans/near/points	Obtener localizaciones de planes cercanos

Tabla 34 - Descripción de los endpoints usados en planes

7.9.2 Itinerarios

En las tablas que se muestran a continuación se establece la relación entre cada endpoint utilizado en el grupo de itinerarios y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en las tablas que aparecen a continuación (ver Tabla 35 y Tabla 36).

Endpoint	GET	POST	PUT	DELETE
api/itineraries/{itineraryId}/comments	X	X		
api/itineraries/{itineraryId}/{lang}			X	
api/itineraries		X		
api/itineraries/search	X			
api/itineraries/{itineraries}/plans		X		
api/itineraries/{itineraries}/plans/{planId}				X
api/itineraries/popular	X			
api/me/itineraries	X			
api/itineraries/{itineraryId}	X			

Tabla 35 - Tipos de endpoints utilizados en itinerarios

Descripción de cada *endpoint*

Endpoint	Descripción
api/itineraries/{itineraryId}/comments	Obtención e inserción de los comentarios de un itinerario dado
api/itineraries/{itineraryId}/{lang}	Inserción de un lenguaje en un itinerario dado
api/itineraries	Inserción de un itinerario
api/itineraries/search	Obtención de itinerarios según unos filtros
api/itineraries/{itineraries}/plans	Añadir planes a un itinerario dado
api/itineraries/{itineraries}/plans/{planId}	Eliminar planes de un itinerario
api/itineraries/popular	Obtener planes populares de un itinerario
api/me/itineraries	Obtener itinerarios creados por mi

Tabla 36 - Descripción de endpoints utilizados en itinerarios

7.9.3 Categorías

En las tablas que se muestran a continuación se establece la relación entre cada *endpoint* utilizado en el grupo de categorías y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en la Tabla 37 y Tabla 38.

Endpoint	GET	POST	PUT	DELETE
api/categories/{id}/plans	X			
api/categories	X			

Tabla 37 - Tipos de endpoints utilizados en categorías

Descripción de cada *endpoint*

Endpoint	Descripción
api/categories/{id}/plans	Obtención de las categorías de un plan id
api/categories	Obtención de los datos de un plan popular

Tabla 38 - Descripción de endpoints utilizados en categorías

7.9.4 Recomendaciones inteligentes

En las tablas que se muestran a continuación se establece la relación entre cada endpoint utilizado en el grupo de recomendaciones inteligentes y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en las

Tabla 39 y Tabla 40.

Endpoint	GET	POST	PUT	DELETE
api/plans/recommendation	X			

Tabla 39 - Tipos de endpoints utilizados en recomendaciones inteligentes

Descripción de cada *endpoint*

Endpoint	Descripción
api/plans/recommendation	Obtención de los planes recomendados

Tabla 40 - Descripción de endpoints utilizados en recomendaciones inteligentes

7.9.5 Autenticación

En las tablas que se muestran a continuación se establece la relación entre cada endpoint utilizado en el grupo de autenticación y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en las Tabla 41 y Tabla 42.

Endpoint	GET	POST	PUT	DELETE
api/auth/refresh		X		
api/auth/sign-in		X		
api/auth/sign-out		X		
api/auth/recover				
api/users/{id}/itineraries		X		
api/me	X			

Tabla 41 - Tipos de endpoints utilizados en autenticación

Descripción de cada *endpoint*

Endpoint	Descripción
api/auth/refresh	Refrescar el token de autenticación
api/auth/sign-in	Iniciar sesión
api/auth/sign-out	Cerrar sesión
api/auth/recover	Petición para el restablecimiento de la contraseña
api/auth/sign-up	Registrarse
api/me	Obtener mis datos de perfil

Tabla 42 - Descripción de endpoints utilizados en autenticación

7.9.6 Backoffice

En las tablas que se muestran a continuación se establece la relación entre cada *endpoint* utilizado en el grupo de backoffice y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en la Tabla 43 y Tabla 44.

Endpoint	GET	POST	PUT	DELETE
api/images		X		
api/admin/stats/refresh		X		
api/admin/weather/refresh		X		
api/admin/providers/run		X		

Tabla 43 - Tipos de endpoints utilizados en backoffice

Descripción de cada *endpoint*

Endpoint	Descripción
api/images	Inserción de imágenes
api/admin/stats/refresh	Actualizar afluencias de los planes
api/admin/weather/refresh	Actualizar datos meteorología
api/admin/providers/run	Obtención de los planes del api del ayuntamiento de Madrid

Tabla 44 - Descripción de endpoints utilizados en backoffice

7.9.7 Errors

En las tablas que se muestran a continuación se establece la relación entre cada *endpoint* utilizado en el grupo de errores y los métodos HTTP asociados a estos, así como una breve explicación de la funcionalidad de cada uno de ellos, tal como se puede observar en la Tabla 45 y Tabla 46.

Endpoint	GET	POST	PUT	DELETE
api/errors	X			

Tabla 45 - Tipos de endpoints utilizados en errors

Descripción de cada *endpoint*

Endpoint	Descripción
api/errors	Obtención de errores

Tabla 46 - Descripción de endpoints utilizados en errors

7.9.8 Swagger

Para la exposición de los distintos *endpoints* disponibles para los clientes, se ha optado por el uso de *Swagger* [42], una herramienta de autogeneración de documentación a través de anotaciones. Esta herramienta genera una interfaz web que permite probar los diferentes *endpoints* y visualizar los parámetros de entrada y salida, además de los códigos de error.

En la interfaz que nos proporciona, podemos acceder a todos los *endpoints* que se hayan configurado en la API, como bien podemos observar en la Figura 70

Para poder probarlo es necesario hacer clic en el *endpoint* que queremos probar, suponiendo que necesitase datos de entrada simplemente habría que rellenarlo o crear un objeto JSON y probarlo, como se puede observar en la Figura 71.

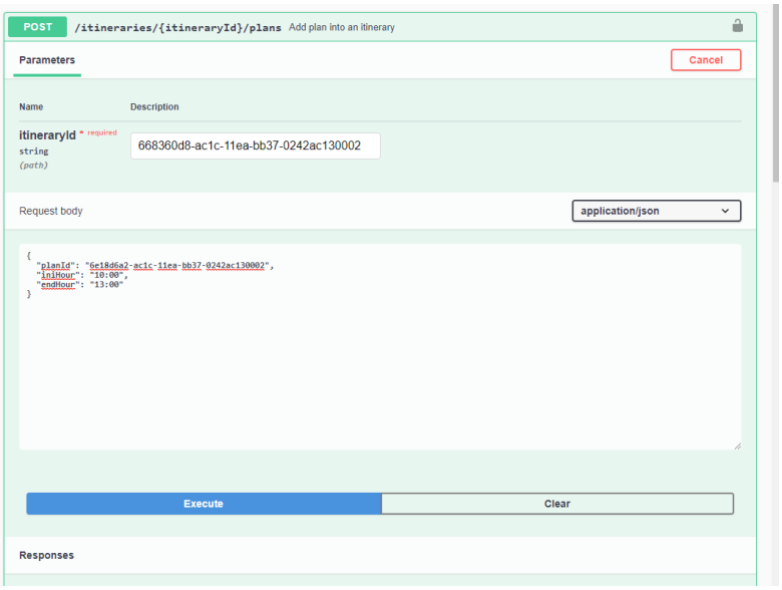


Figura 70 - Petición desde Swagger

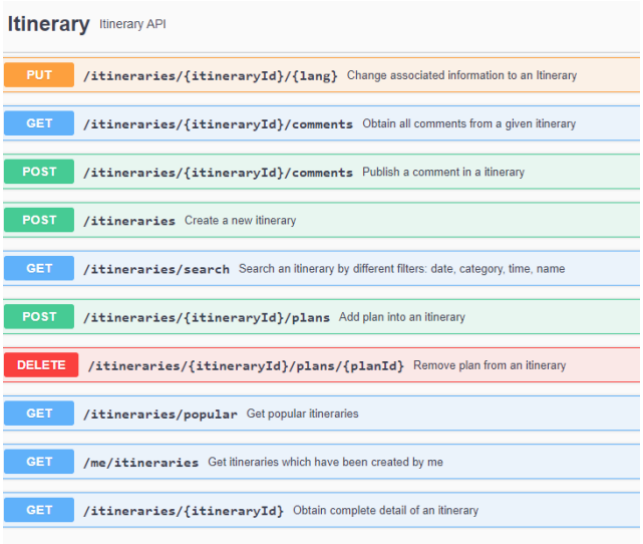


Figura 71 - Listado de algunos de los endpoints en Swagger

Capítulo 8. Evaluación

En este capítulo se analizarán los resultados obtenidos de una evaluación de la usabilidad realizada a algunos usuarios a los que se les ha propuesto utilizar la aplicación durante unos días y así poder observar la satisfacción de estos al usarla.

1. Metodología

Para realizar la evaluación, se ha utilizado la herramienta de *Google Forms* [63]. La evaluación ha sido realizada por once usuarios. Tal como se observa en la Figura 72 el 81,8% de los usuarios que han realizado la evaluación tienen entre 20 y 30 años, el 9,1% tiene entre 30 y 40 años, y otro 9,1% tienen entre 40 y 50 años. Entre los usuarios había personas que estaban muy familiarizadas con el mundo de la tecnología, personas que usaban diferentes aplicaciones diariamente desde el punto de vista de un usuario medio, y personas poco familiarizadas con la tecnología.

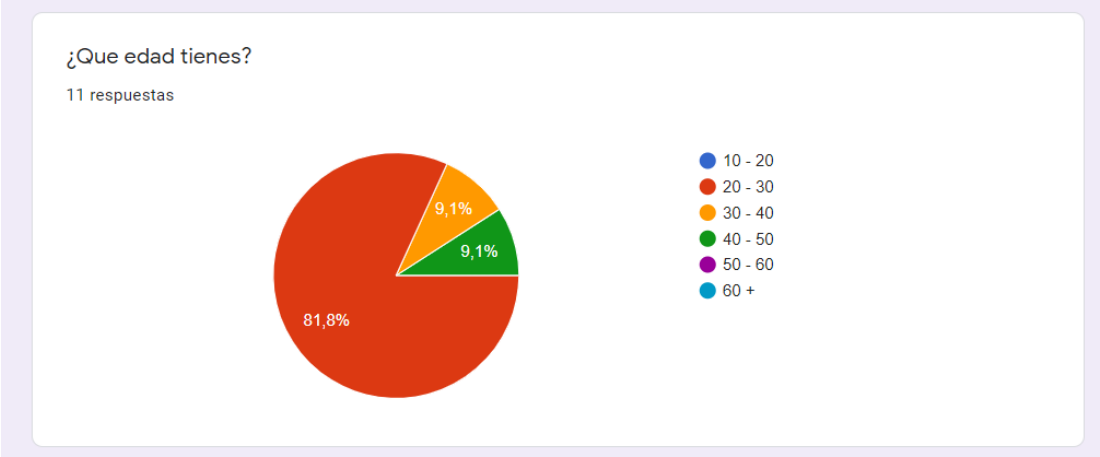


Figura 72 - Información sobre la edad de los usuarios encuestados

El tipo de preguntas realizadas han sido de respuesta sí/no o preguntas subjetivas acerca de las diferentes características principales de la aplicación, cambios que realizarían o funciones que desearían tener en el futuro. En concreto, las preguntas se refieren a 6 aspectos sobre la aplicación: El inicio de sesión y registros, la pantalla principal, el proceso de crear un itinerario, buscar itinerarios, el mapa interactivo, y el uso de la aplicación y nuevas características. Las preguntas propuestas se pueden consultar en el ANEXO III. Preguntas .

Por último, comentar que, para realizar la evaluación, se les dio acceso a la aplicación para que la pudieran utilizarla y también un enlace al cuestionario.

2. Resultados obtenidos

En esta sección se mostrarán los resultados obtenidos de la evaluación realizada agrupados por el tipo de preguntas realizadas.

2.1. Inicio de sesión y Registro

En esta característica, el 100% los usuarios que han probado la aplicación han considerado que el proceso de inicio de sesión tanto el de registrarse, era intuitivo, y que no cambiarían nada (ver Figura 73 y Figura 74).

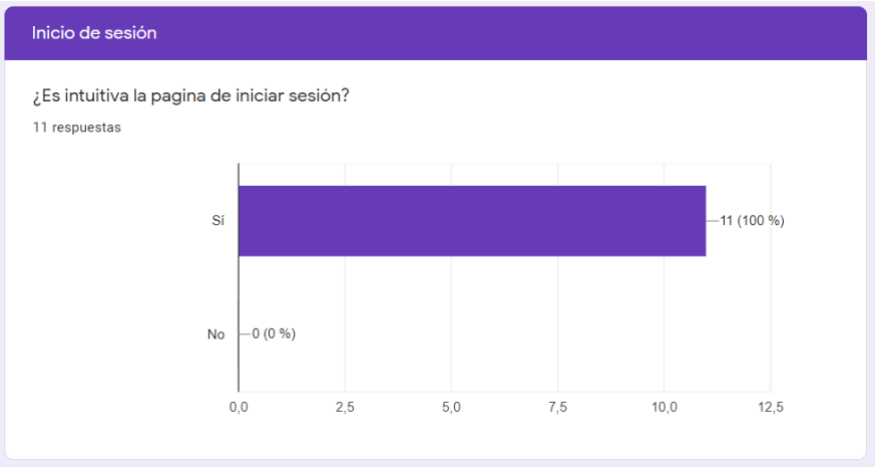


Figura 73 - Encuesta I inicio de sesión y registro

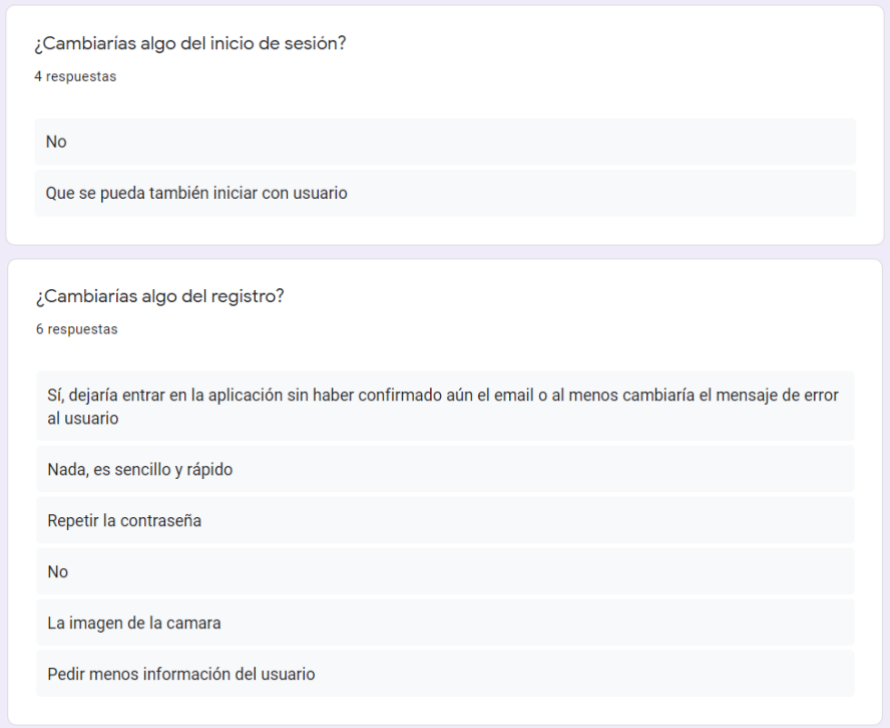


Figura 74 - Encuesta II inicio de sesión y registro

2.2. Pantalla Principal

En este apartado, al 72,7% de los usuarios les ha parecido intuitivo, al 18,2% no, y al 9,1% les ha parecido intuitivo el menú, pero no las listas que se muestran en él. Además de que se han propuesto varias mejoras para la aplicación donde la característica por la que más se opta por los usuarios es la de añadir información acerca de los apartados que aparecen en los menús (ver Figura 75).

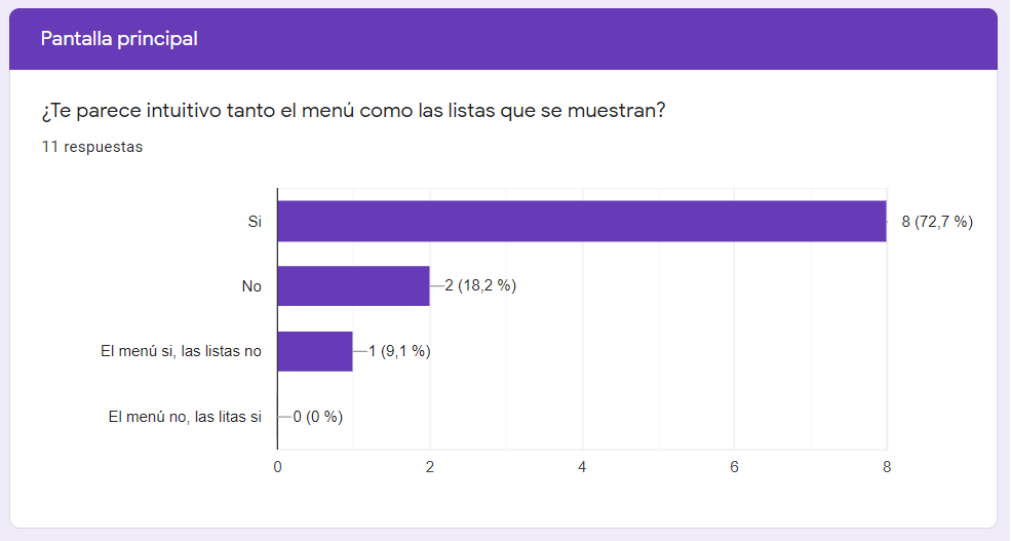


Figura 75 – Satisfacción Pantalla principal

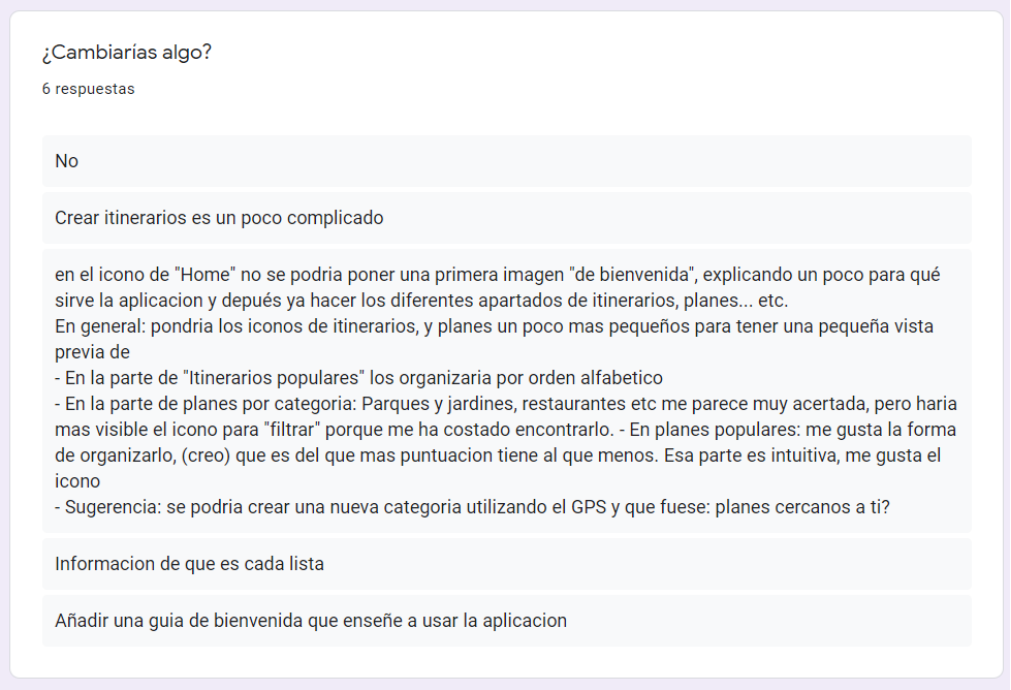


Figura 76 - Encuesta pantalla principal

2.3. Crear Itinerario

En este punto, al 54,5% de los usuarios les ha parecido intuitivo el proceso de crear un itinerario, pero al 45,5 % de los usuarios les ha parecido que la creación de un itinerario era bastante complicada y un proceso muy largo (ver Figura 78, Figura 79 y Figura 77).

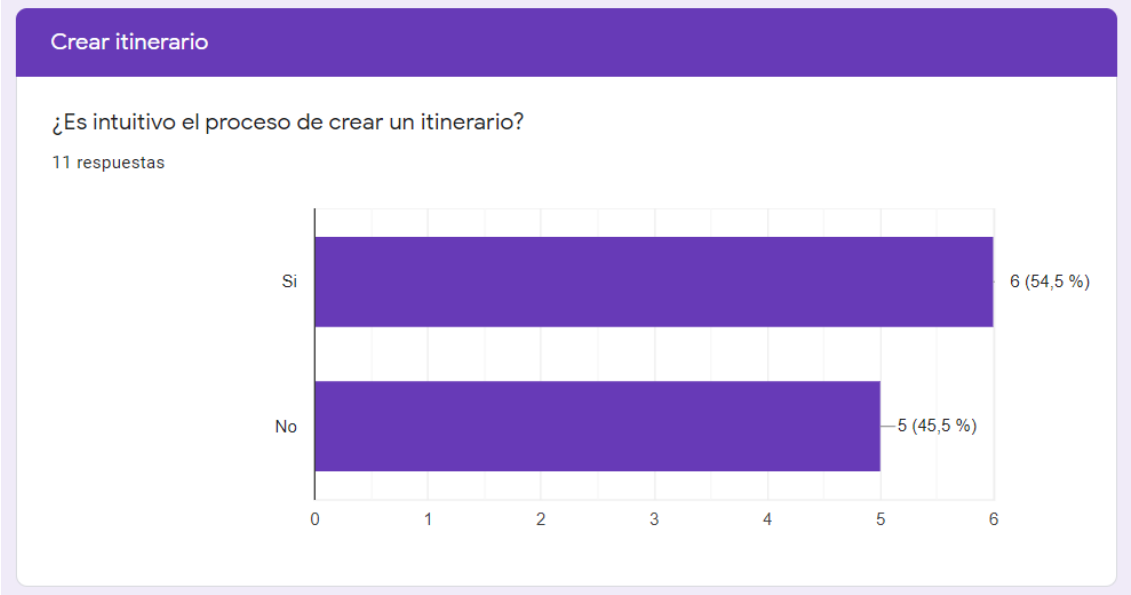


Figura 77 - Satisfacción crear itinerario

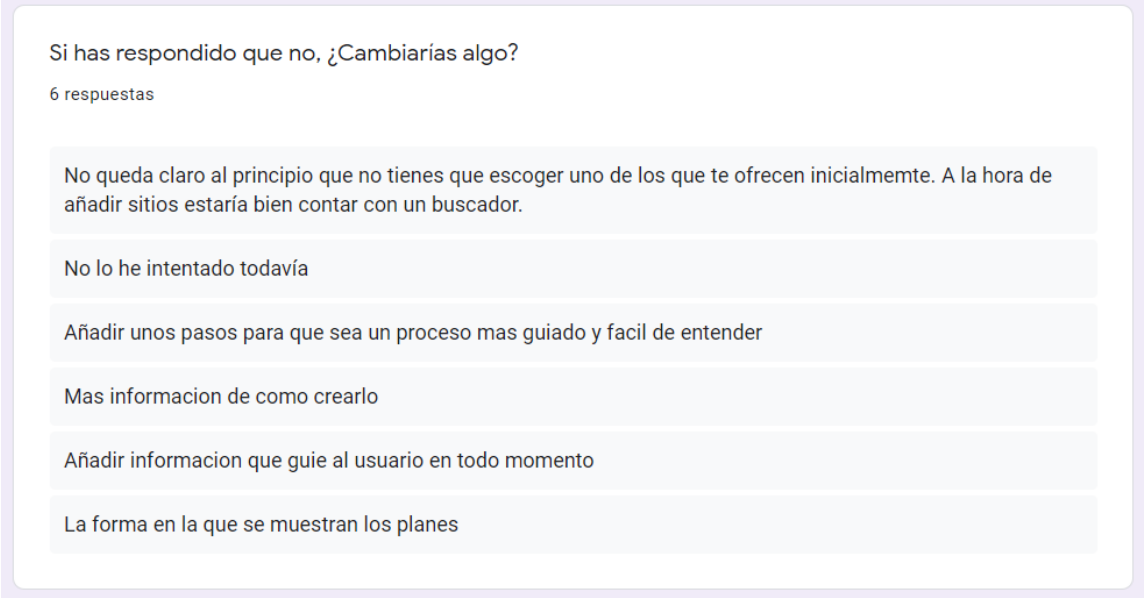


Figura 78 - Encuesta I crear itinerario

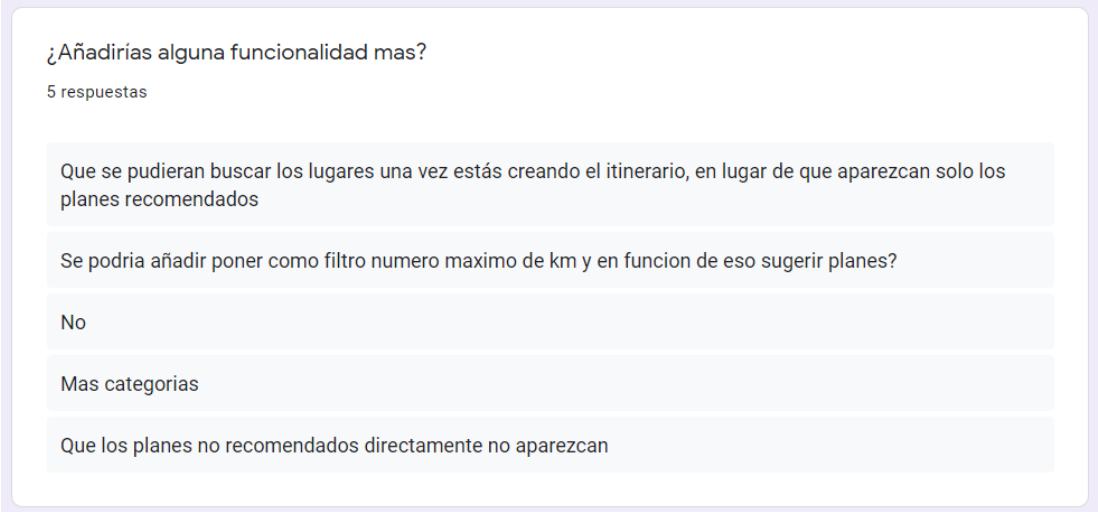


Figura 79 - Encuesta II crear itinerario

2.4. Buscar itinerario

En el apartado de buscar itinerario, el 100% de los usuarios ha opinado que era un proceso intuitivo, aunque también se han propuesto varias ampliaciones (ver Figura 80 y Figura 81).

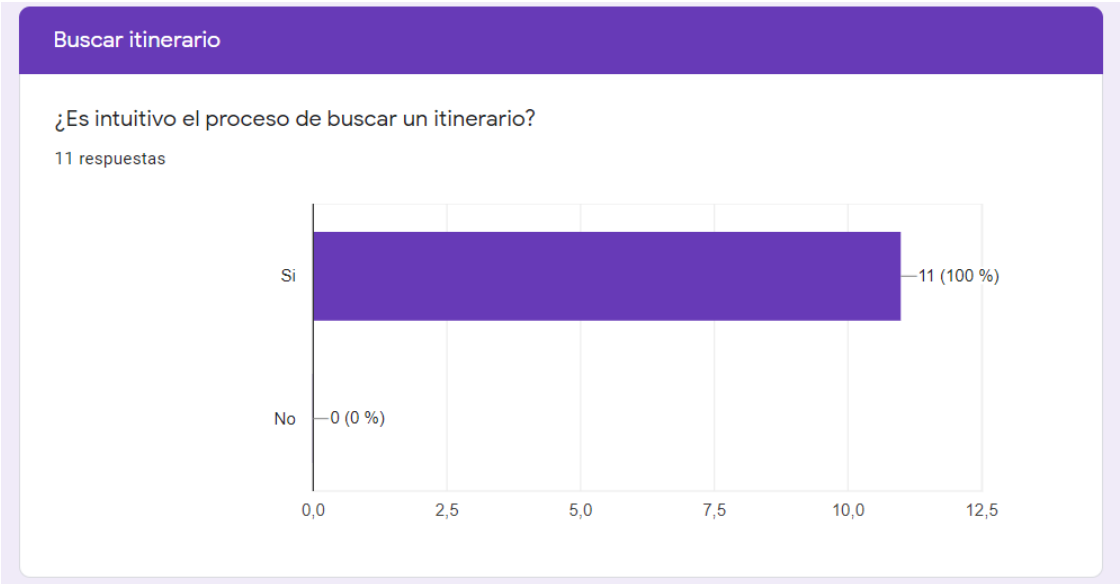


Figura 80 - Encuesta I búsquedas

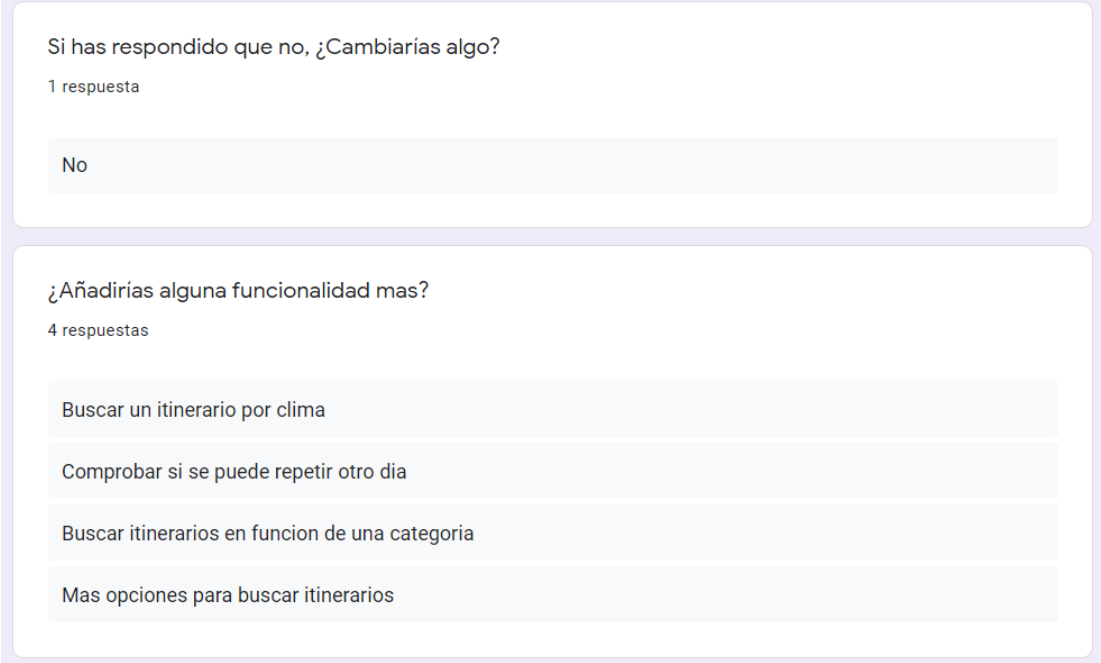


Figura 81 - Encuesta II búsquedas

2.5. Mapa

En esta característica el 100% de los usuarios que han realizado la encuesta han votado que es intuitivo el proceso de los planes en el mapa y en que aporta valor para la aplicación. Además, también se han recogido opiniones para posibles ampliaciones futuras (ver Figura 82 y Figura 83).

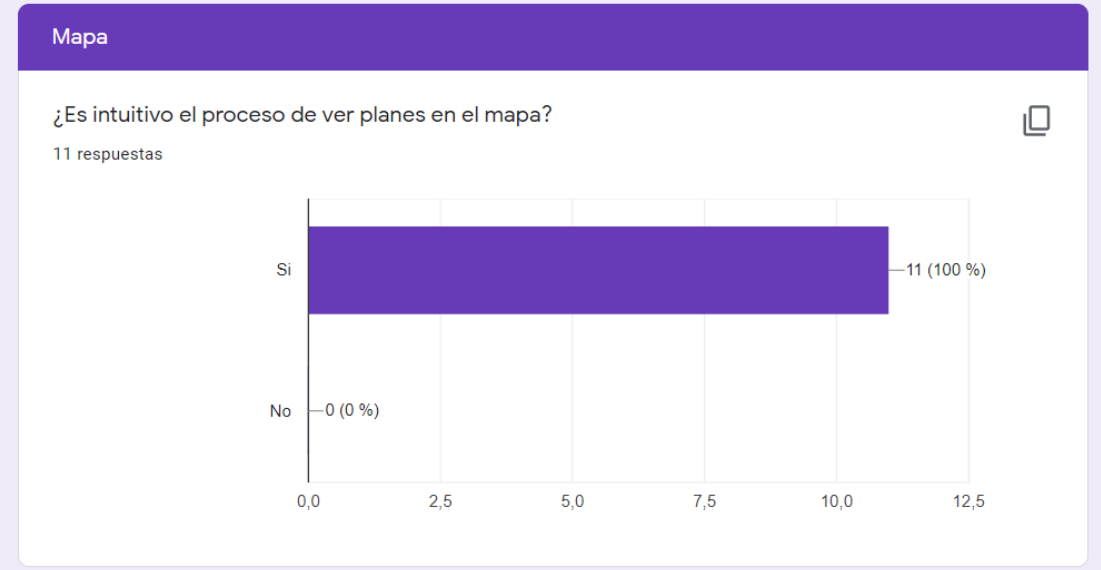
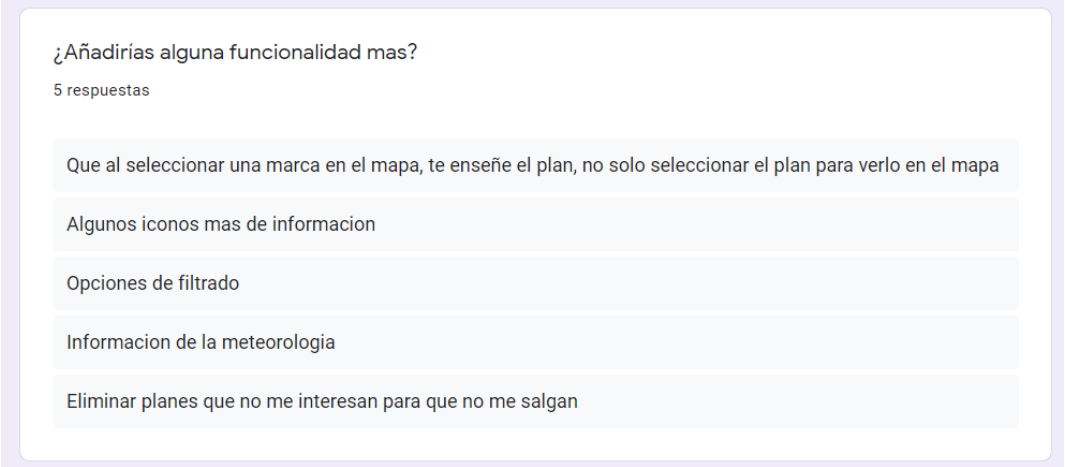


Figura 82 - Encuesta I mapa interactivo



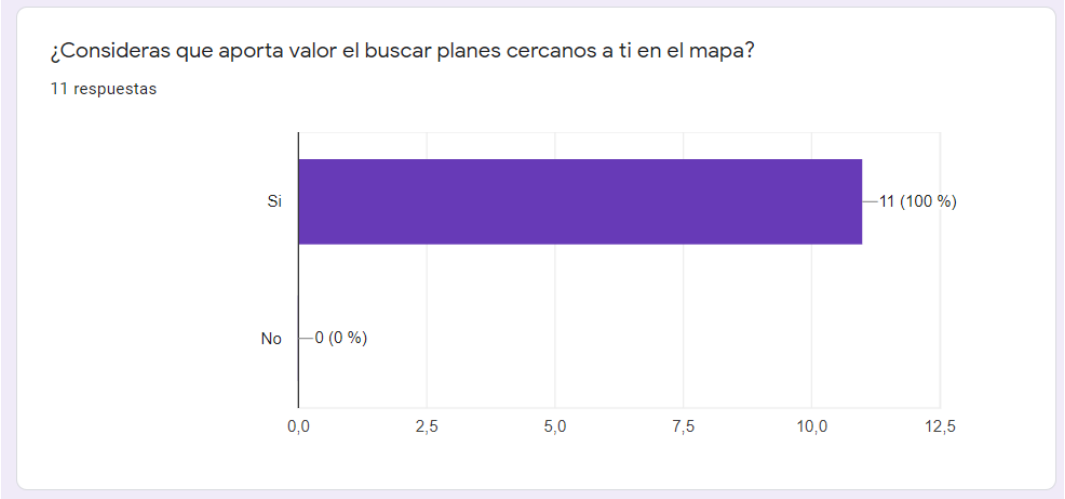


Figura 83 -Encuesta II mapa interactivo

2.6. Uso de la aplicación

Finalmente, se les preguntó a los usuarios si utilizarían la aplicación regularmente para planificar sus excursiones y visitas a Madrid, además de preguntarles cual sería la funcionalidad más importante para implementar próximamente. (ver Figura 84).

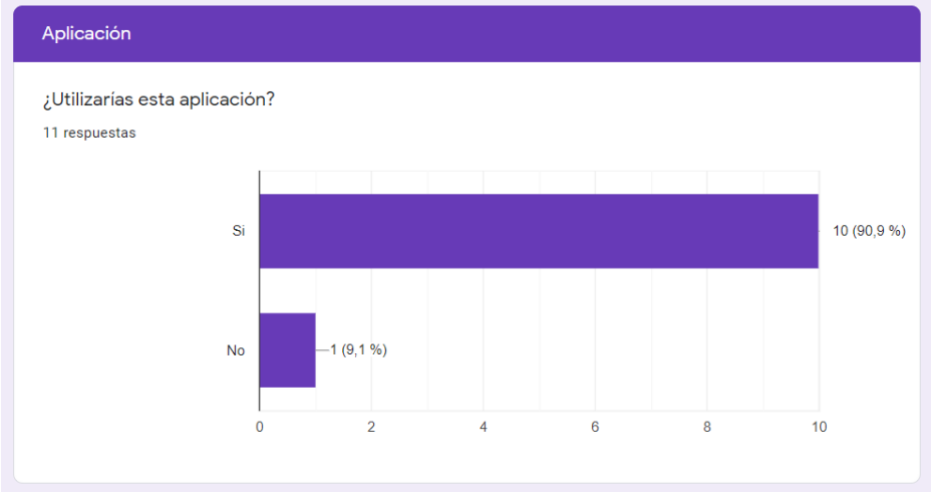


Figura 84 - Satisfacción sobre la aplicación

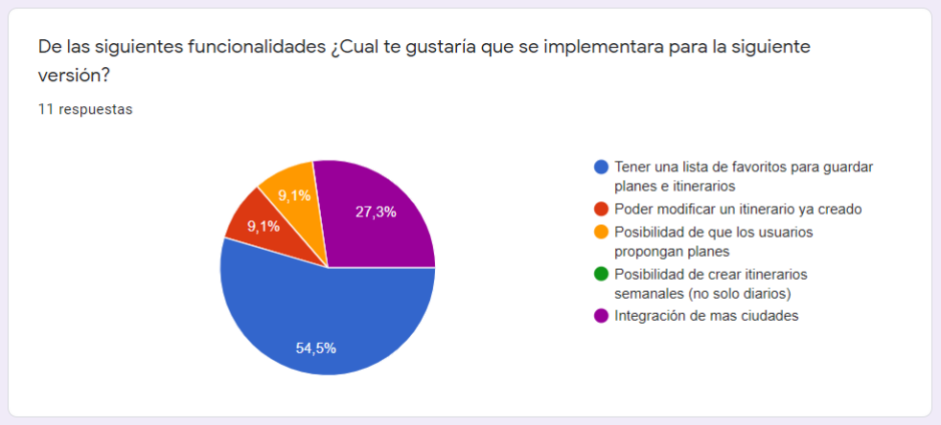


Figura 85 - Resultados de próximas funcionalidades

El 100% de los usuarios estaban satisfechos con la aplicación y la funcionalidad futura que ganó con un 54,5% de los votos, fue la de poder añadir planes e itinerarios a una lista de favoritos de los usuarios (ver Figura 85), un 9,1% de los usuarios votó a poder modificar un itinerario ya creado, un 9,1% votó a que los usuarios pudieran proponer planes, y un 27,3% votó a la integración de más ciudades.

Capítulo 9. Conclusiones y trabajo futuro.

9.1. Conclusiones

En este proyecto se ha desarrollado un servicio de valor añadido que permite a los visitantes y ciudadanos de la ciudad de Madrid acceder a la información de planes de ocio que tienen lugar en dicha ciudad de manera sencilla, además permite crear conjuntos de planes con el fin de planificar visitas turísticas más enriquecedoras.

Los usuarios podrán intercambiar opiniones sobre los lugares o conjuntos de planes que han visitado, lo que permite que el resto de los usuarios puedan contar con información de valor antes de organizar su visita a la ciudad de Madrid.

El servicio proporciona su propia recomendación basada en información que se ha conseguido congregar en el sistema, es decir, información que fuera del servicio no se encuentra de manera conjunta, esto permite que el usuario pueda consumir toda esta información en un solo servicio, uno de los principales objetivos del proyecto. Por último, el sistema ofrece funcionalidad basada en geolocalización para permitir a los usuarios un acceso rápido a los planes de ocio en función del lugar de estancia o de residencia.

Se considera que el servicio aquí desarrollado aporta a los usuarios el valor que inicialmente se ideó, llegando a cumplir así los objetivos generales de este proyecto. Además, se han planteado una serie de ideas para continuar mejorando la calidad de servicio, estas se describen a continuación.

9.2. Trabajo futuro

La funcionalidad del proyecto se podría mejorar de la siguiente manera:

- **Integración de más ciudades:** La aplicación está orientada a la ciudad de Madrid, sin embargo, se podría extender su funcionalidad con el objetivo de ofrecer un servicio similar para otras ciudades.

- **Añadir más clientes:** Actualmente solo se dispone de una interfaz Android para el uso de los servicios del proyecto, se podría desarrollar otro cliente como por ejemplo una interfaz para dispositivos móviles iOS.

- **Módulo de administración:** En este momento, cualquier cambio que haya que realizarse en los datos, se debe de realizar directamente sobre la base de datos o sobre el código fuente de la aplicación, en este sentido, se podría desarrollar un módulo de administración para que se puedan realizar acciones como eliminar planes o itinerarios, eliminar usuarios, eliminar comentarios de planes o itinerarios, o modificar datos que se deseen.

- **Ampliación del algoritmo ID3 con datos históricos:** Existen APIs que facilitan la obtención de datos históricos de varias décadas atrás. En este sentido se podría

elaborar un modelo predictivo en base a datos históricos de clima o afluencia, realizando predicciones sobre días lejanos al actual.

- **Proposición de planes:** En el servicio solo están recogidos los lugares turísticos que existen en la base de datos de Madrid, pero podrían existir lugares no registrados interesantes de ser visitados. Es por ello por lo que podría implementarse una opción para que los usuarios registrados pudieran proponer lugares para ser añadidos al servicio.

- **Funcionalidad favoritos:** Actualmente los usuarios registrados de la aplicación no disponen de la opción de poder guardar en su perfil, los planes y los itinerarios que les hayan gustado. En este sentido, se podría extender la funcionalidad para que los usuarios puedan ver en su perfil los planes y los itinerarios que han marcado como favoritos.

- **Reporte de errores:** Desarrollar funcionalidad que permita a los usuarios emitir incidencias en caso de que la consistencia o veracidad de los datos se vea comprometida.

- **Posibilidad de hacer itinerarios semanales (no solo diarios):** Actualmente los usuarios registrados pueden crear itinerarios para un día concreto. En este sentido se podría extender la funcionalidad de forma que los usuarios registrados pudieran crear itinerarios semanales (de más de un día de duración) por si disponen de varios días en la ciudad y quieren aprovechar al máximo el tiempo.

Chapter 9. Conclusions and future work

9.1. Conclusions

This project has developed a value-added service that allows visitors and citizens of the city of Madrid to access information on leisure plans taking place in that city in a simple way, also allows the creation of sets of plans in order to organize more enriching tourist visits.

Users will be able to exchange opinions about the places or sets of plans they have visited, which allows the rest of the users to have valuable information before organizing their visit to the city of Madrid.

The service provides its own recommendation based on information that has been gathered in the system, that is, information that outside the service is not found together, this allows the user to consume all this information in a single service, one of the main objectives of the project. Finally, the system offers functionality based on geolocation to allow users quick access to leisure plans depending on the place of stay or residence.

It is considered that the service developed here provides users with the value initially conceived, thus fulfilling the general objectives of this project. In addition, a series of ideas have been put forward to continue improving the quality of service, these are described below.

9.2. Future work

The project functionality could be improved in the following way:

- **Integration of more cities:** The application is oriented to the city of Madrid. However, its functionality could be extended in order to offer a similar service for other cities.

- **Adding more clients:** Currently only one Android interface is available for the use of the project's services, another client could be developed such as an interface for iOS mobile devices.

- **Administration module:** At this time, any change that needs to be made to the data, must be made directly on the database or on the source code of the application, in this sense, an administration module could be developed so that actions such as removing plans or routes, removing users, removing comments from plans or routes, or modifying data can be done.

- **Extension of the ID3 algorithm with historical data:** There are APIs that facilitate obtaining historical data from several decades ago. In this sense, a predictive model could be elaborated based on historical climate or flow data, making predictions about days far from the current one.

- **Plan proposal:** The service only includes the tourist places that exist in the Madrid database, but there could be interesting unregistered places to visit. That is why an option could be implemented so that registered users could propose places to be added to the service.

- **Favorite functionality:** Currently, registered users of the application do not have the option to save in their profile the plans and itineraries they have liked. In this sense, the functionality could be extended so that users can see in their profile the plans and itineraries they have marked as favorites.

- **Bug reporting:** Develop functionality that allows users to issue bugs in case the consistency or veracity of the data is compromised.

- **Possibility of making weekly itineraries (not only daily):** Currently registered users can create itineraries for a specific day. In this sense, the functionality could be extended so that registered users could create weekly itineraries (lasting more than one day) in case they have several days in the city and want to make the most of their time.

Capítulo 10. Aportaciones individuales

10.1. Juan Antonio Ávila Catalán

- **Ingeniería de requisitos**
  De forma periódica, a través de reuniones con el resto de los compañeros, participó en el planteamiento inicial de la aplicación, proponiendo algunos de los casos de uso que han sido desarrollados.

  También fue partícipe de reuniones con el director del TFG para la resolución de dudas respecto a algunos de los casos de uso y formato de la memoria.

- **Desarrollo**

  - **Modelado de los datos**
    Participó junto al resto de compañeros, en el diseño del modelo entidad-relación sobre el que se basa el modelo de datos de la aplicación.

  - **Aplicación cliente Android**
    Participó en la corrección de errores y puesta en marcha de mejoras relacionadas con la notificación de excepciones, gestión de la conectividad en el dispositivo móvil y uso de marcos de trabajo para la gestión de tareas asíncronas.

    Integró el sistema de autenticación mediante token presente en los servicios web del servidor, tanto el inicio de sesión, como el registro, además de la actualización en segundo plano del token cuando este expira, haciendo innecesario que el usuario requiera de introducir de nuevo sus credenciales.

  - **Diseño e implementación de la API REST**
    Fue el responsable de integrar e implementar la autorización y autenticación en el proyecto, elaborando los *endpoints* relacionados con las funcionalidades del rol administrador y usuarios autenticados entre otros.

    Este trabajo también incluye el estudio y posterior integración del módulo de seguridad que integra el marco de trabajo *Spring Boot*, el cual hace uso de roles para la diferenciación entre los distintos actores del sistema y requiere de una serie de componentes para su puesta en marcha.

  - **RabbitMQ**
    Dado el gran número de tareas que requerían de un largo tiempo de procesamiento, planteó el uso de RabbitMQ para resolver los problemas derivados de procesar tareas de larga duración.

    También fue quién investigó los problemas derivados del uso de un sistema de colas y cuáles eran las posibles soluciones, aplicándolas en el proyecto.

    Además, ha sido el responsable de integrarla con el marco de trabajo *Spring Boot*.

- **Extracción e importación de datos de API ayuntamiento**
  Elaboró el diseño escogido para los proveedores de planes, estableciendo el uso de la librería *Retrofit* para la conexión con servicios web de terceros.

- **Internacionalización automática de los planes**
  Elaboró e implemento el modelo de datos y la conexión con el servicio de terceros necesario para la automatización de las traducciones de planes e itinerarios.

- **Extracción e importación de datos de horarios y afluencias**
  Diseñó e implementó el algoritmo responsable de la extracción de los horarios y afluencias de la aplicación, además del uso de servidores proxy.

- **Algoritmo de planes recomendados**
  Implementó el algoritmo ID3 responsable de la funcionalidad de planes recomendados.

- **Extracción e importación de datos de la AEMET**
  Diseñó e implementó el algoritmo de recogida periódica de datos meteorológico, estableciendo el diseño de la base de datos necesario para el almacenamiento de estos datos.

- **Implementación de Swagger**
  Incorporó al proyecto la librería *Swagger* para la generación automática de documentación siguiendo el estándar Open API.

- **Despliegue de la API**
  Fue propulsor y responsable del uso de *Docker* para facilitar el despliegue de los artefactos necesarios para la ejecución del lado servidor. También desarrolló el flujo de trabajo utilizado en el servidor de integración continua.

- **Memoria**
  Elaboró los puntos correspondientes a la arquitectura utilizada en la aplicación, además de los relacionados con los patrones de diseño utilizados en la misma, contextualizándolos y elaborando los diagramas UML que aparecen en ellos.

  También redactó los apartados que corresponden al desarrollo de los algoritmos y ha sido partícipe en la corrección de errores gramaticales y de estilo.

10.2. Alberto Fernández-Baíllo Rodríguez

- **Ingeniería de requisitos**
  Participó en el *brainstorming* inicial para la consolidación de las bases de este proyecto, posteriormente creo junto a sus compañeros un listado con todos los casos de uso que se quería realizar y se votó cuales formarían el MVP de la aplicación.

  Tras tener un listado con los casos de uso que se querían desarrollar, participo en el establecimiento de requisitos de usuario que este debía tener, y se elaboró una especificación de requisitos para la posterior aprobación del cliente, que en este caso era nuestro tutor.

  Una vez establecida la base del proyecto, participo en un proceso de investigación buscando en internet APIs que pudieran ser de utilidad, así como páginas web que pudieran ser de gran utilidad.

- **Desarrollo**

  - **Modelo de datos:**
    Participo junto a sus compañeros en la elaboración del diagrama de entidades el cual ha sido la base del proyecto

  - **Extracción e importación de datos:**
    En el lado del servidor participó en la implementación y extracción de datos de la API pública del ayuntamiento para su utilización, entre ellos algunos como restaurantes, templos católicos y no católicos y la información meteorológica de la API de la AEMET mediante el uso de retrofit, que es una librería que nos permite consumir estas APIS para obtener sus resultados transformándolos posteriormente en clases POJO para su utilización.

  - **Desarrollo API REST:**
    Tras la extracción de estos datos participó en el desarrollo de varios casos de uso para su futura consumición por la parte de cliente.

  - **Desarrollo Android:**
    En el lado del cliente participó en la elaboración de la vista principal creando *recyclerviews* (listas de Android) para poder ilustrar los planes populares, y los planes por categorías, posteriormente creó una opción al final de cada lista para que el usuario pudiera ver todas las opciones posibles, para ello creó otro *recyclerview* encapsulado para que todos los listados pudieran usarlo y así evitar duplicación de código. También creo parte de la lógica para consumir los *endpoints* creados con anterioridad en la parte de servidor y utilizarlos en cliente.

    También fijó las bases de los diseños de las pantallas de iniciar sesión y crear usuario.

    Junto con sus compañeros retocó los detalles de plan e itinerarios y además creo las opciones para que se mostrasen toda la información relacionada con los

horarios de ese plan durante ese día y en eses mismo momento, como durante toda la semana. Posteriormente, utilizando la librería *MPAndroidChart* elaboró las tablas de barras para mostrar la afluencia que tenías es plan durante las horas en las que estaba abierto durante toda la semana.

Creó también la vista base de buscar planes e itinerarios, así como su lógica para la consumición de esos *endpoints* y la ilustración de las listas con la información obtenida.

Ayudo también en una creación base de la pantalla inicial anterior al inicio de sesión con la utilización de una librería llamada *Lottie* para mostrar una breve animación con el nombre de la aplicación.

- **Memoria**
  Participó en la elaboración de varias tablas y restructuración de gran parte del documento, así como la elaboración del punto uno hablando sobre la motivación y los objetivos, y la tecnología empleada, así como el siete hablando de los colores, tipografía, las funcionalidades de la aplicación y el desarrollo y documentación de la API REST. Posteriormente participó en la corrección y puesta a punto de esta.

10.3. Íñigo García-Conde Trueba

- **Ingeniería de requisitos**

  Participo en las sesiones iniciales de *brainstorming* en las que se comenzó a dar ideas sobre que funcionalidad se debería desarrollar para el proyecto, todo desde un punto de vista funcional.
  Una vez que se decidió de manera no definitiva que funcionalidad estaría dentro del sistema se comenzó con la captura de requisitos, formando de esta manera los casos de uso final junto a sus compañeros. En esta fase el alumno aporto ideas para la resolución de problemas tecnológicos y funcionales que fueron surgiendo. También ayudo a la priorización de los casos de uso en función del valor que aportaban al sistema y de las complejidades tecnológicas que suponían.

  Se participo de forma activa en el conjunto de tecnologías que se utilizarían en el proyecto, aportó ideas propias y escucho e intento comprender aquellas tecnologías que el alumno no conocía.

  Por último, junto a sus compañeros, como en todo el proceso de ingeniería de requisitos, ayudó a realizar un estudio sobre las fuentes de datos, el conjunto de datos que conformarían el sistema y de qué manera se deberían guardar y tratar.

  Cabe destacar que durante la realización del proyecto fueron surgiendo problemas que no se habían planteado y que tanto el alumno como el equipo tuvieron que ir encontrando soluciones para mitigarlos, continuando así en cierto modo ciertas tareas de ingeniería de requisitos.

- **Desarrollo**

  - **Modelado de datos**
    La primera parte en la cual aporto el alumno fue en la formación de las estructuras de datos. Tras el estudio que realizó el equipo se fueron creando los distintos modelos de datos.

  - **Extracción e importación de datos de API ayuntamiento**
    Toda esta primera fase fue acompañada del desarrollo de las extracciones de datos de las APIs del Ayuntamiento de Madrid que fueron añadidas, como por ejemplo la parte de los museos. Todo esto se realizó con la ayuda de *Retrofit*, tecnología propuesta por un compañero.

  - **Desarrollo API REST**
    Desarrollo distintos casos de uso, así como todos los elementos necesarios para añadirlos a la API REST. Colaboró activamente en la captura de errores y estandarización de estos, aunque no en su versión final. Inicio el desarrollo de la implementación del módulo de seguridad de *Spring Boot* y ayudo en los inicios de su desarrollo.

- **Cliente Android**

  Se encargó de la implementación del mapa interactivo de la aplicación. Esto conllevo el estudio de la API de Google Maps y sus elementos. El desarrollo de los casos de uso que alimentan el mapa y su funcionamiento.

  Por último, se desarrolló un componente global para la aplicación que diera acceso a la ubicación en tiempo real al resto de componentes del cliente Android para poder disfrutar de esta funcionalidad.

- **Memoria**

  En el apartado de la menoría el alumno ha desarrollado el punto 4 donde se recogen las distintas tablas de casos de uso en función de los actores del sistema.

  El punto 5, modelado de datos donde junto a sus compañeros realizó los diagramas entidad-relación y se explicó las distintas tablas de la implementación del diagrama antes mencionado junto a sus atributos o columnas.

  El punto 12 donde se proporcionan las guías de instalación del sistema.

  Por último, la revisión y corrección de elementos de toda la memoria junto a sus compañeros.

10.4. Cristo Fernando López Cabañas

- **Ingeniería de requisitos**

  Participó de manera activa junto a sus compañeros en reuniones diarias y semanales tanto en la decisión de las tecnologías que se utilizarían en la realización del proyecto, como en el diseño de la aplicación, además del establecimiento de las bases en las que ésta iba a desarrollarse. Estas actividades consistieron en reuniones de grupo y reuniones con el director del TFG para la resolución de dudas que iban surgiendo.

  También participó en la elaboración de bocetos iniciales para cada una de las vistas de la aplicación y en la creación de las entidades principales de ésta (Planes, Itinerarios, Usuarios, etc. y lo que representa cada una.

  Una vez establecidas las bases, participó en elaboración de una lista de funcionalidades que podría llegar a tener el sistema y así obtener una visión del máximo alcance que se podría alcanzar, así como las funcionalidades que se intentaría llegar a implementar realmente.

- **Desarrollo**

  - **Modelado de los datos**
    Con el resto de los compañeros, participó en la elaboración de diagramas para la representación de los datos que iba a tener la aplicación y la estructura de estos.

  - **Extracción e importación de datos de API ayuntamiento**
    Junto con otros compañeros, ayudo a la extracción de datos de la API del ayuntamiento de Madrid (parques, jardines, y mercadillos) realizando peticiones con la librería *Retrofit*, y participó en la normalización de estos datos para su utilización en la aplicación y almacenaje en la base de datos.

  - **Implementación de la API REST**
    Participó en el desarrollo de *endpoints* en el lado del servidor (por ejemplo, el módulo Itinerario) utilizando los datos extraídos de la API del ayuntamiento para que, mediante peticiones, estos datos pudieran ser enviados a un cliente. Una vez había implementados suficientes *endpoints*, decidió pasarse a seguir desarrollando en la parte del cliente Android.

  - **Diseño e implementación del cliente Android**
    Inicialmente, juntos a los demás compañeros, se estableció el diseño que seguiría la aplicación para la implementación de sus funcionalidades. Se optó por seguir el patrón MVP en toda la aplicación.

    Se encargó del diseño de la pantalla principal (Home) de la aplicación añadiendo listas de itinerarios populares y planes seleccionados por el algoritmo de recomendación, realizando peticiones a la API REST y tratando los datos que llegaban.

En unas primeras versiones, debido a que los datos no estaban completos, con la ayuda de un compañero, diseñó un repositorio auxiliar que creaba datos ficticios para así poder independizar la capa del cliente del servidor y poder realizar pruebas con estos datos de una manera más sencilla. Después esta capa ficticia se cambió por las llamadas reales al servidor con la librería *Retrofit*.

Además, añadió un menú inferior para poder cambiar entre las diferentes pantallas de la aplicación.

Fue responsable de la elaboración inicial de las vistas de detalle de plan además del detalle de los itinerarios.

Diseñó la vista del perfil del usuario, realizando una versión inicial donde aparecerían sus itinerarios creados.

Se encargó de rediseñar la pantalla de búsqueda completamente y a completar las llamadas al servidor.

Finalmente elaboró el diseño de la vista de crear un itinerario, así como su proceso y ayudó en la corrección de errores que se fueron encontrando.

- **Memoria**

Participó en la redacción de las tablas de los casos de uso además de en la revisión de la memoria corrigiendo errores gramaticales.

Elaboró el punto 2 de la memoria donde se habla del software más popular similar al desarrollado en este proyecto destacando sus ventajas e inconvenientes sobre éste.

Ayudó a la contextualización de patrones en el punto 6, contextualizando los que han sido usados en el lado del cliente.

Elaboró el punto 8, donde se habla sobre la evaluación realizada a diversos usuarios que probaron la aplicación y respondieron unas preguntas de un cuestionario.

Redactó el punto 9, donde se presentan las funcionalidades que se podrían implementar en un futuro, así como una descripción de cada una de ellas.

Bibliografía

[1]. Civitatis. https://www.civitatis.com/es/. Recuperado el día 14 de Junio de 2020.

[2]. TripAdvisor. https://www.tripadvisor.es/. Recuperado el día 14 de Junio de 2020.

[3]. Booking. https://www.booking.com. Recuperado el día 14 de Junio de 2020.

[4]. Minube. https://www.minube.com/. Recuperado el día 14 de Junio de 2020.

[5]. Android Studio. https://es.wikipedia.org/wiki/Android_Studio. Recuperado el día 14 de Junio de 2020.

[6]. Spring Boot. https://spring.io/projects/spring-boot. Recuperado el día 14 de Junio de 2020.

[7]. RabbitMQ. https://es.wikipedia.org/wiki/RabbitMQ. Recuperado el día 14 de Junio de 2020.

[8]. MariaDB. https://mariadb.org/. Recuperado el día 14 de Junio de 2020.

[9]. MySQL. https://www.mysql.com/. Recuperado el día 14 de Junio de 2020.

[10]. IntelliJ IDEA. https://www.jetbrains.com/es-es/idea/. Recuperado el día 14 de Junio de 2020.

[11]. JavaEE. https://es.wikipedia.org/wiki/Java_EE. Recuperado el día 14 de Junio de 2020.

[12]. jOOQ. https://www.jooq.org. Recuperado el día 14 de Junio de 2020.

[13]. Git. https://git-scm.com/. Recuperado el día 14 de Junio de 2020.

[14]. Kotlin. https://kotlinlang.org/. Recuperado el día 14 de Junio de 2020.

[15]. Codigo ISO 639-1. https://es.wikipedia.org/wiki/ISO_639-1/. Recuperado el día 14 de Junio de 2020.

[16]. Aemet Open Data. https://opendata.aemet.es/. Recuperado el día 14 de Junio de 2020.

[17]. Cliente-servidor. https://es.wikipedia.org/wiki/Cliente-servidor. Recuperado el día 14 de Junio de 2020.

[18]. HTTP. https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto. Recuperado el día 14 de Junio de 2020.

[19]. MVP. https://martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp. Recuperado el día 14 de Junio de 2020.

[20]. Cockburn, Alistair. Hexagonal Architecture. https://alistair.cockburn.us/hexagonal-architecture/. Recuperado el día 14 de Junio de 2020.

[21]. Fowler, Martin. Patterns of Enterprise Application Architecture. s.l. : Addison-Wesley international, 2002.

[22]. Gamma, Erich. Design Patterns: Elements of Reusable Object-Oriented Software. s.l. : Addison Wesley professional, 1994.

[23]. Evans, Eric. Domain-Driven Design. s.l. : Pearson Educacion, 2003.

[24]. Vernon, Vaughn. Implementing Domain-Driven Design. s.l. : Addison-Wesley Educational. 10.

[25]. Singleton. https://es.wikipedia.org/wiki/Singleton. Recuperado el día 14 de Junio de 2020.

[26]. Cadena de responsabilidad. https://es.wikipedia.org/wiki/Cadena_de_responsabilidad. Recuperado el día 14 de Junio de 2020.

[27]. Deepak Alur, Dan Malks, John Crupi. Core J2EE Patterns: Best Practices and Design Strategies. s.l. : Prentice Hall, 2003.

[28]. Harvesine. Movable type script. https://www.movable-type.co.uk/scripts/latlong.html. Recuperado el día 14 de Junio de 2020.

[29]. MPAndroidChart. https://github.com/PhilJay/MPAndroidChart. Recuperado el día 14 de Junio de 2020.

[30]. ID3. https://es.wikipedia.org/wiki/Algoritmo_ID3. Recuperado el día 14 de Junio de 2020.

[31]. BOE. Real Decreto 463/2020, declaración del estado de alarma por la COVID-19. https://www.boe.es/diario_boe/txt.php?id=BOE-A-2020-3692. Recuperado el día 14 de Junio de 2020.

[32]. Buscador de Google. https://es.wikipedia.org/wiki/Buscador_de_Google30.

[33]. Servidor Proxy. https://es.wikipedia.org/wiki/Servidor_proxy. Recuperado el día 14 de Junio de 2020.

[34]. Scraping. https://es.wikipedia.org/wiki/Web_scraping. Recuperado el día 14 de Junio de 2020.

[35]. Search Engine Scraper. http://scraping.compunect.com/. Recuperado el día 14 de Junio de 2020.

[36]. Servidor Proxy. https://es.wikipedia.org/wiki/Servidor_proxy. Recuperado el día 14 de Junio de 2020.

[37]. Cola de prioridad. https://es.wikipedia.org/wiki/Cola_de_prioridades. Recuperado el día 14 de Junio de 2020.

[38]. Portal de Datos Abiertos. Ayto. de Madrid. https://datos.madrid.es/portal/site/egob/. Recuperado el día 14 de Junio de 2020.

[39]. API.https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones .Recuperado el día 14 de Junio de 2020.

[40]. Cloud Translation. Google Cloud. https://cloud.google.com/translate/docs. Recuperado el día 14 de Junio de 2020.

[41]. Transactional Outbox. microservices.io. https://microservices.io/patterns/data/transactional-outbox.html. Recuperado el día 14 de Junio de 2020.

[42]. Swagger. https://swagger.io/. Recuperado el día 14 de Junio de 2020.

[43]. Entrega continua. https://es.wikipedia.org/wiki/Entrega_continua.

[44]. Docker. https://es.wikipedia.org/wiki/Docker_(software). Recuperado el día 14 de Junio de 2020.

[45]. Github Actions. https://github.com/features/actions. Recuperado el día 14 de Junio de 2020.

[46]. Deployment rollback. Kubernetes. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#rolling-back-a-deployment. Recuperado el día 14 de Junio de 2020.

[47]. Spring Boot Profiles. https://docs.spring.io/spring-boot/docs/1.1.x/reference/html/boot-features-profiles.html. Recuperado el día 14 de Junio de 2020.

[48]. SendGrid. https://sendgrid.com/. Recuperado el día 14 de Junio de 2020.

[49]. Amazon AWS. https://aws.amazon.com/es/. Recuperado el día 14 de Junio de 2020.

[50]. Gerard. xUnit test patterns: Refactoring test code. s.l. : Meszaros, 2007.

[51]. AnemicDomainModel. https://martinfowler.com/bliki/AnemicDomainModel.html. Recuperado el día 14 de Junio de 2020.

[52]. Google makes Kotlin a first-class language for writing Android Apps. Tech Crunch. https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/. Recuperado el día 14 de Junio de 2020.

[53]. Github. https://es.wikipedia.org/wiki/GitHub. Recuperado el día 14 de Junio de 2020.

[54]. JPA. https://es.wikipedia.org/wiki/Java_Persistence_API. Recuperado el día 14 de Junio de 2020.

[55]. JDBC. https://es.wikipedia.org/wiki/Java_Database_Connectivity. Recuperado el día 14 de Junio de 2020.

[56]. ORM. https://es.wikipedia.org/wiki/Mapeo_objeto-relacional. Recuperado el día 14 de Junio de 2020.

[57]. Hibernate. https://hibernate.org/. Recuperado el día 14 de Junio de 2020.

[58]. MVC. https://martinfowler.com/eaaDev/uiArchs.html#ModelViewController. Recuperado el día 14 de Junio de 2020.

[59]. Service Layer. Martin Fowler. https://martinfowler.com/eaaCatalog/serviceLayer.html. Recuperado el día 14 de Junio de 2020.

[60]. Active Record. https://es.wikipedia.org/wiki/Active_record. Recuperado el día 14 de Junio de 2020.

[61]. Inversión de control. https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control. Recuperado el día 14 de Junio de 2020.

[62]. Reflection API. Oracle. https://www.oracle.com/technical-resources/articles/java/javareflection.html. Recuperado el día 14 de Junio de 2020.

[63]. Google Forms. https://www.google.es/intl/es/forms/about. Recuperado el día 14 de Junio de 2020.

[64]. Lint. https://es.wikipedia.org/wiki/Lint. Recuperado el día 14 de Junio de 2020.

[65]. EclipseLink. https://www.eclipse.org/eclipselink/. Recuperado el día 14 de Junio de 2020.

[66]. Hibernate. https://hibernate.org/. Recuperado el día 14 de Junio de 2020.

ANEXO I. Guía de instalación

1. Entornos de despliegue

Tal y como se ha reflejado en puntos anteriores, el proyecto consta de diferentes sistemas (cola de mensajería, servidor http, base de datos, etc.) que, integrados entre sí, conforman la aplicación.

Cada uno de estos sistemas requieren de ser configurados de forma independiente, además de requerir de un despliegue con necesidades distintas dependiendo de su naturaleza, esto es, no requiere de los mismos recursos ni pasos de instalación el servidor de base de datos MySQL que el servidor HTTP Java.

A pesar de que esas tareas podían ser realizadas de forma manual, esto supone un factor bloqueante para la entrega continua [43] , y, por ende, enlentece el ciclo de desarrollo. Por ello, este proyecto se ha desarrollado dando una gran importancia a la facilidad del despliegue y el uso de entornos pre productivos para la realización de pruebas.

Han sido configurados tres entornos independientes entre sí, que pueden verse gráficamente de izquierda a derecha ordenados de menor a mayor madurez del software, siendo la fase final aquella en la que los usuarios finales pueden hacer uso de este.

Cada uno de estos entornos han sido configurados en una imagen de Docker [44] que hace posible el despliegue inmediato sin apenas necesidad de intervención manual por parte del desarrollador.

A continuación, se describen cada una de estas fases y la motivación con la que han sido diseñadas, se pueden ver en la Figura 86.

Figura 86 - distintos entornos utilizados durante el proyecto

1.1 Dev

Es un entorno local, alojado en la máquina de cada desarrollador del proyecto. Este entorno es donde el desarrollador puede simular de forma idéntica a como lo haría en cualquiera de los dos siguientes, pero sin la posibilidad de entrar en conflicto con el resto del equipo.

La utilización de un primer entorno local aislado del resto del equipo se consideró fundamental para poder realizar pruebas antes de subir los nuevos cambios al repositorio remoto de código.

Para hacer uso de este primer entorno, el desarrollador tan sólo requiere de hacer uso de las configuraciones de su IDE que han sido previamente alojadas en el sistema de control de versiones, simplificando el proceso de despliegue de 4 sistemas independientes en únicamente dos pasos, tal y como se aprecia en la Figura 87.

Figura 87 - Configuración para despliegue en entorno local

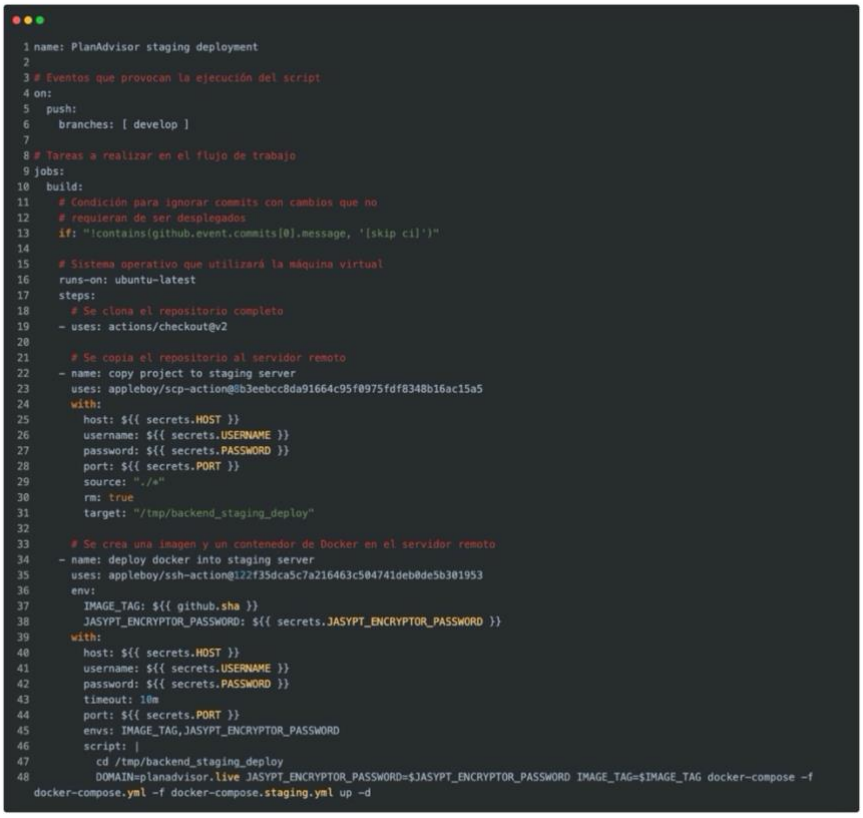
1.2 Staging

Este segundo entorno aloja el software que se considera puede estar preparado para poner a disposición de los usuarios finales pero que requiere antes de ser estabilizado.

El entorno de *staging* no se encuentra de forma local, es compartido por los desarrolladores del proyecto y es el primero en el que se integran todos los cambios realizados, de ahí su naturaleza de entorno de estabilización. No requiere de intervención manual, esto es gracias al uso de un sistema de integración continua, en este caso *Github Actions* [45], que se encarga de ejecutar un flujo de pasos en cada nuevo cambio que es publicado en la rama principal del repositorio remoto.

Este servicio de integración continua ha sido configurado a través de un script que puede verse en la Figura 88, y que está compuesto de forma resumida por los siguientes pasos:

1. Clonación del repositorio en el momento del *commit* que provocó su ejecución.
2. Copia de los ficheros al servidor remoto en el que será desplegado el sistema
3. Creación de una imagen de la aplicación del servidor y contenedores *Docker* donde los distintos sistemas estarán desplegados.



```yaml
name: PlanAdvisor staging deployment

# Eventos que provocan la ejecución del script
on:
  push:
    branches: [ develop ]

# Tareas a realizar en el flujo de trabajo
jobs:
  build:
    # Condición para ignorar commits con cambios que no
    # requieran de ser desplegados
    if: "!contains(github.event.commits[0].message, '[skip ci]')"

    # Sistema operativo que utilizará la máquina virtual
    runs-on: ubuntu-latest
    steps:
      # Se clona el repositorio completo
    - uses: actions/checkout@v2

      # Se copia el repositorio al servidor remoto
    - name: copy project to staging server
      uses: appleboy/scp-action@8b3eebcc8da91664c95f0975fdf8348b16ac15a5
      with:
        host: ${{ secrets.HOST }}
        username: ${{ secrets.USERNAME }}
        password: ${{ secrets.PASSWORD }}
        port: ${{ secrets.PORT }}
        source: "./*"
        rm: true
        target: "/tmp/backend_staging_deploy"

      # Se crea una imagen y un contenedor de Docker en el servidor remoto
    - name: deploy docker into staging server
      uses: appleboy/ssh-action@122f35dca5c7a216463c504741deb0de5b301953
      env:
        IMAGE_TAG: ${{ github.sha }}
        JASYPT_ENCRYPTOR_PASSWORD: ${{ secrets.JASYPT_ENCRYPTOR_PASSWORD }}
      with:
        host: ${{ secrets.HOST }}
        username: ${{ secrets.USERNAME }}
        password: ${{ secrets.PASSWORD }}
        timeout: 10m
        port: ${{ secrets.PORT }}
        envs: IMAGE_TAG,JASYPT_ENCRYPTOR_PASSWORD
        script: |
          cd /tmp/backend_staging_deploy
          DOMAIN=planadvisor.live JASYPT_ENCRYPTOR_PASSWORD=$JASYPT_ENCRYPTOR_PASSWORD IMAGE_TAG=$IMAGE_TAG docker-compose -f
docker-compose.yml -f docker-compose.staging.yml up -d
```

Figura 88 - Script configuración integración continua

1.3 Prod

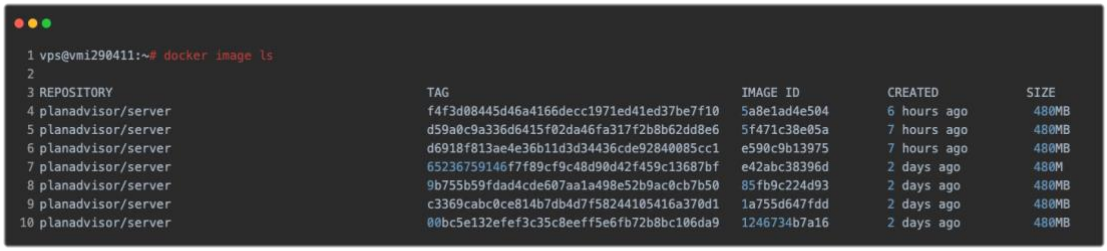
Este es el tercer y último entorno, es el utilizado por los usuarios finales de la aplicación. Está compuesto por idénticos pasos al entorno de *staging*, con la única diferencia de que su despliegue supone una instancia diferente de todos los sistemas y por tanto nunca interfiere con el entorno anterior, de forma que los datos y la red privada son distintos.

2. Historial de imágenes

Por medio del sistema de entrega continúa configurado, es posible volver a instalar una versión anterior del software en caso de cualquier problema, funcionalidad comúnmente conocida como *deployment rollback* [46].

Cada una de las imágenes desplegadas en los entornos de ‘Prod’ o ‘Staging’ quedan almacenadas en un historial que hace posible su restauración en caso de necesidad, por ejemplo, por problemas en la última versión publicada.

Para la distinción de las imágenes y su fácil recuperación, quedan marcadas con el hash del *commit* que provocó su generación, de forma que son identificadas unívocamente y sin posibilidad de sobreescritura.



```
vps@vmi290411:~# docker image ls

REPOSITORY                TAG                                                              IMAGE ID       CREATED          SIZE
planadvisor/server        f4f3d08445d46a4166decc1971ed41ed37be7f10                         5a8e1ad4e504   6 hours ago      480MB
planadvisor/server        d59a0c9a336d6415f02da46fa317f2b8b62dd8e6                         5f471c38e05a   7 hours ago      480MB
planadvisor/server        d6918f813ae4e36b11d3d34436cde92840085cc1                         e590c9b13975   7 hours ago      480MB
planadvisor/server        65236759146f7f89cf9c48d90d42f459c13687bf                         e42abc38396d   2 days ago       480M
planadvisor/server        9b755b59fdad4cde607aa1a498e52b9ac0cb7b50                         85fb9c224d93   2 days ago       480MB
planadvisor/server        c3369cabc0ce814b7db4d7f58244105416a370d1                         1a755d647fdd   2 days ago       480MB
planadvisor/server        00bc5e132efef3c35c8eeff5e6fb72b8bc106da9                         1246734b7a16   2 days ago       480MB
```

Figura 89 - Historial de imágenes creadas

3. Configuración adaptativa

La configuración adaptativa ha hecho posible el uso de distintos servicios y configuraciones adaptadas a cada entorno.

Dado que la aplicación hace uso de servicios de terceros (AWS, Sendgrid o Google Translate, entre otros) que conllevan un coste asociado por su uso, ha sido imprescindible incluir la posibilidad de utilizar *Service Stubs* [21] que sustituyan a los servicios reales para entorno preproductivos (dev y staging).

Gracias al uso de los perfiles [47] que ofrece Spring Boot, han sido elaboradas y configuradas distintas implementaciones que son utilizadas por el contenedor de dependencias función del perfil activo (ver Figura 90).

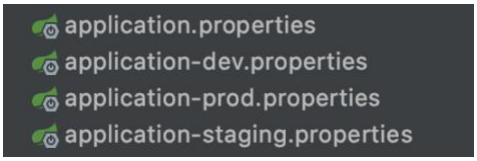


Figura 90 - Archivos de configuración para los distintos entornos

Como ejemplo de esta configuración adaptativa, se describe el servicio de email, del cual se dispone de hasta tres implementaciones diferentes, cada una de ellas utilizada por uno de los tres perfiles.

- *Fake*
  Utilizada en el entorno ‘dev’. Esta implementación imprime en la consola del desarrollador el email a enviar, de forma que nunca se realiza ningún envío real.

- *MailTrap*
  Utilizada en el entorno de ‘staging’. Es una implementación ideada para su uso interno en el entorno preproductivo. Todos los emails son enviados mediante un servidor SMTP que intercepta los emails derivándolos a un buzón de correo ‘virtual’, haciendo así que no se utilice el servicio de email real y no exista la

posibilidad de enviar por error un correo electrónico no deseado a cualquier usuario final.

- *SendGrid*
  Esta es la implementación utilizada en el entorno de producción: 'prod'. Se hace uso del servicio de correo que proporciona SendGrid [48], un proveedor de email que acarrea un coste por cada email enviado.

El uso de distintas implementaciones según el perfil ha sido utilizado en el uso de Amazon Web Services (AWS) [49], entre otros.

4. Instalación del servidor

Es posible realizar la instalación del servidor, para ello es necesario cumplir una serie de requisitos. Estos son descritos en el apartado *"ANEXO I. 4. Requisitos de instalación"*

Lo primero que se debe hacer es clonar el repositorio del proyecto desde *GitHub*. Se podrán observar los elementos de que contiene el proyecto en la Figura 91.

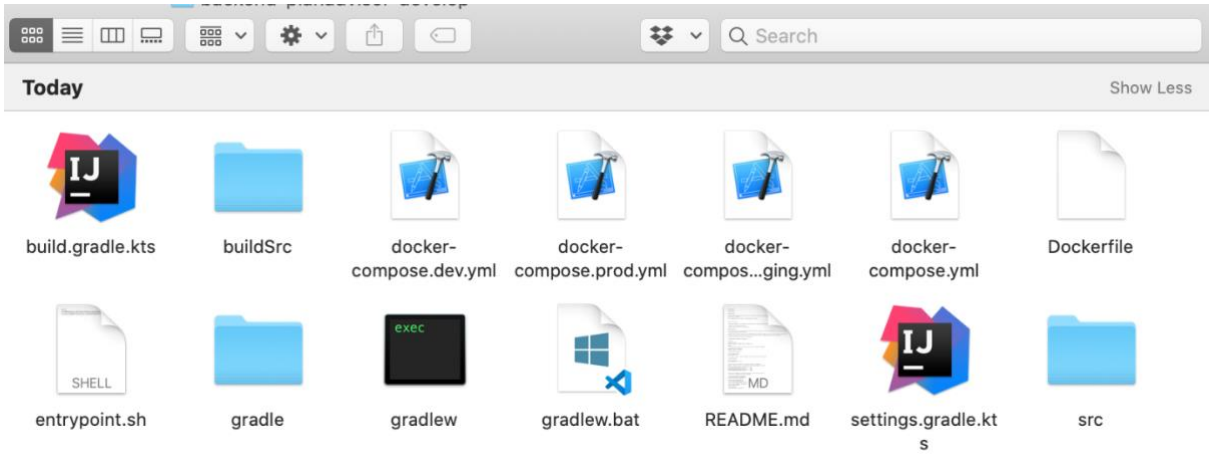


Figura 91 - Elementos del proyecto clonado

Una vez clonado el proyecto, se deberá descargar e instalar la última versión de *Docker*. El sistema se despliega a través de una imagen generada con *Docker*, esto permite empaquetar todas las dependencias y librerías del sistema y moverlo de un ordenador a otro.

Una vez instalado *Docker* procedemos a crear la imagen mediante el comando *docker-compose*. En el proyecto se puede ver el fichero "docker-compose.prod.yml". Este archivo necesita una serie de parámetros que se explican a continuación:

- DOMAIN= dominio donde se quiere realizar la instalación del sistema.

- IMAGE_TAG= nombre de la imagen, este nombre debe identificar la imagen de manera única para poder ejecutarla posteriormente.

- DB_PASS= contraseña escogida para el usuario ‘root’ de la base de datos

- RABBIT_USER= nombre de usuario escogido para el sistema de colas

- RABBIT_PASS= contraseña escogida para el sistema de colas

- JASYPT_ENCRYPTOR_PASSWORD= contraseña para descifrar las propiedades que hayan sido cifradas en el archivo de propiedades

Después de la orden *docker-compose* se deben especificar los archivos de configuración:

- compose.yml
- compose.prod.yml

El archivo “compone.prod.yml” sobrescribe determinados valores de configuración del archivo “compose.yml” medinate el flag -f. Por último, el comando up levantará todos todas las imágenes en contenedores *Docker*, el indicador -d nos permite finalizar el proceso una vez los contenedores están listos. A continuación, se muestra el comando completo:

DOMAIN=planadvisor.live JASYPT_ENCRYPTOR_PASSWORD=pass IMAGE_TAG=tag_imagen DB_PASS=contraseña_bbdd RABBIT_USER=usuario_rabbit RABBIT_PASS=pass_rabbit docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d

```
Successfully built f72ce1e831b4
Successfully tagged planadvisor/server:planAdvisor
WARNING: Image for service server was built because it did not already exist. To rebuild this image you must use `docker-compos
e build` or `docker-compose up --build`.
Creating backend-planadvisor-develop_reverse_proxy_1 ... done
Creating backend-planadvisor-develop_db_1            ... done
Creating backend-planadvisor-develop_reverse_proxy_ssl_1 ... done
Creating backend-planadvisor-develop_rabbitmq_1          ... done
Creating backend-planadvisor-develop_server_1            ... done
inigo_garcia@Admins-MacBook-Air backend-planadvisor-develop %
```

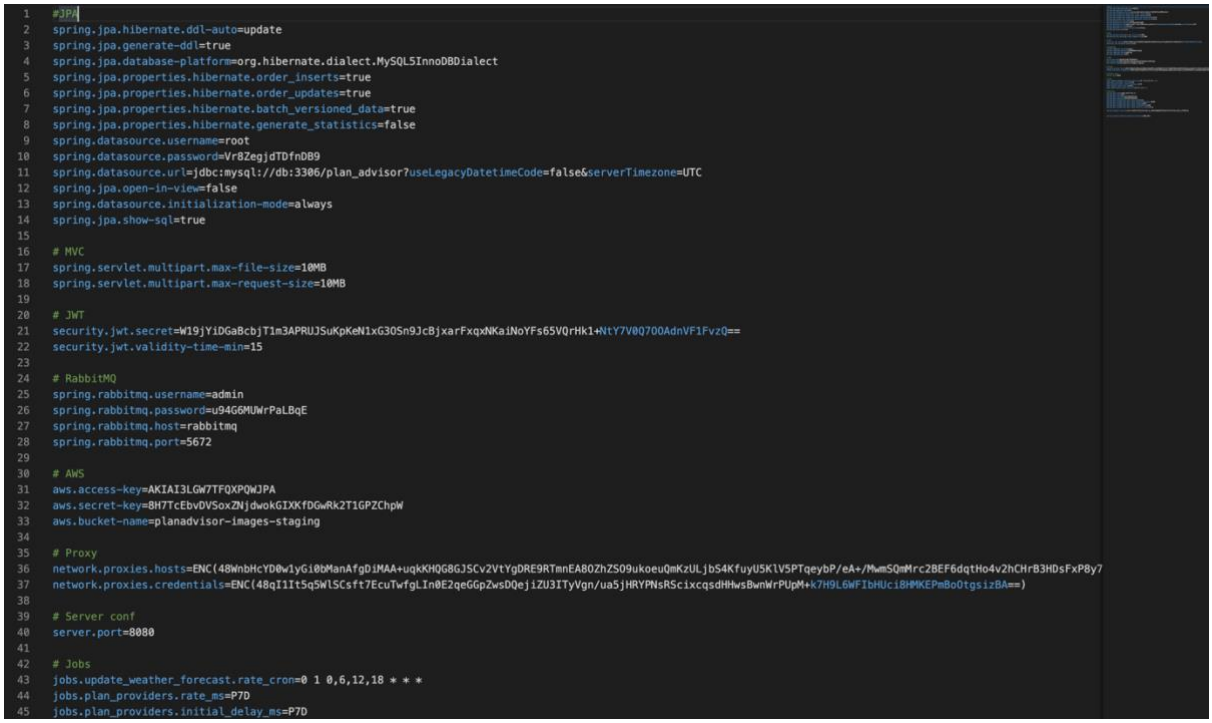
Figura 92 - Resultado ejecución docker compose

5. Requisitos de instalación

Es necesario configurar los parámetros de los distintos servicios que usa el sistema, como las claves, puertos y dominios. Estas propiedades se pueden modificar en el archivo “application.prod.properties” dentro del directorio “/src/resources”, ya que las que están ahora son de uso privado (ver Figura 93). Se puede ver por encima el formato del archivo en la ilustración. La lista de servicios que usa el sistema es la siguiente:

- AWS
- SendGrid servicio *mailing*
- Google Cloud
- AEMET

En el directorio “/src/resources” debe estar incluido la API Key de Google Cloud en formato JSON.



```
#JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.hibernate.order_inserts=true
spring.jpa.properties.hibernate.order_updates=true
spring.jpa.properties.hibernate.batch_versioned_data=true
spring.jpa.properties.hibernate.generate_statistics=false
spring.datasource.username=root
spring.datasource.password=Vr8ZegjdTDfnDB9
spring.datasource.url=jdbc:mysql://db:3306/plan_advisor?useLegacyDatetimeCode=false&serverTimezone=UTC
spring.jpa.open-in-view=false
spring.datasource.initialization-mode=always
spring.jpa.show-sql=true

# MVC
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB

# JWT
security.jwt.secret=W19jYiDGaBcbjT1m3APRUJSuKpKeN1xG3OSn9JcBjxarFxqxNKaiNoYFs65VQrHk1+NtY7V0Q7OOAdnVF1FvzQ==
security.jwt.validity-time-min=15

# RabbitMQ
spring.rabbitmq.username=admin
spring.rabbitmq.password=u94G6MUWrPaLBqE
spring.rabbitmq.host=rabbitmq
spring.rabbitmq.port=5672

# AWS
aws.access-key=AKIAI3LGW7TFQXPQWJPA
aws.secret-key=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
aws.bucket-name=planadvisor-images-staging

# Proxy
network.proxies.hosts=ENC(48WnbHcYD0w1yGi0bManAfgDiMAA+uqkKHQG8GJSCv2VtYgDRE9RTmnEA8OZhZSO9ukoeuQmKzULjbS4KfuyU5KlV5PTqeybP/eA+/MwmSQmMrc2BEF6dqtHo4v2hCHrB3HDsFxP8y7
network.proxies.credentials=ENC(48qI1It5q5WlSCsft7EcuTwfgLIn0E2qeGGpZwsDQejiZU3ITyVgn/ua5jHRYPNsRScixcqsdHHwsBwnWrPUpM+k7H9L6WFIbHUci8MMKEPmBoOtgsizBA==)

# Server conf
server.port=8080

# Jobs
jobs.update_weather_forecast.rate_cron=0 1 0,6,12,18 * * *
jobs.plan_providers.rate_ms=P7D
jobs.plan_providers.initial_delay_ms=P7D
```

Figura 93 - Archivo “application.prod.properties"

ANEXO II. Guía del usuario

Una vez instalado el archivo de instalación de la aplicación, el cual se ha obtenido del repositorio donde está alojado el código, se abre la aplicación y se hace visible el menú de Inicio de sesión, donde es posible optar por varias alternativas para el uso de la aplicación.

Se puede iniciar sesión directamente si se dispone de una cuenta, introduciendo el correo y la contraseña con la que se ha creado la cuenta. En caso de que se haya olvidado la contraseña, también es posible recibir un correo para restablecerla.

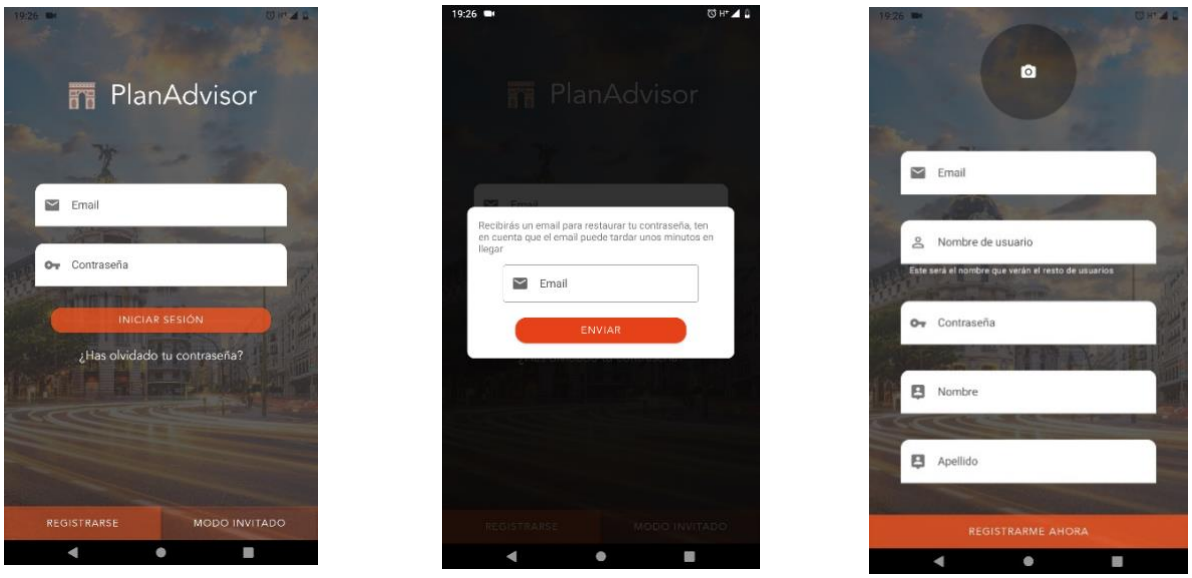


Figura 94 - Pantallas de registro, login y recuperar contraseña

En caso de que no se disponga de una cuenta, en la parte inferior del inicio de sesión se han añadido dos opciones más, la posibilidad de registrarse rellenando un formulario con una foto (opcional), nombre de usuario, contraseña, nombre y apellido, y también, en caso de que no se desee crear una cuenta, se puede acceder a la aplicación en modo invitado, aunque sin poder utilizar todos los servicios disponibles. La Figura 94 muestra las pantallas para que sea más intuitivo.

Cuando se accede a la aplicación, aparece la pantalla de inicio donde se ven los itinerarios populares, planes recomendados, planes populares y planes por categorías (ver ilustración 80).

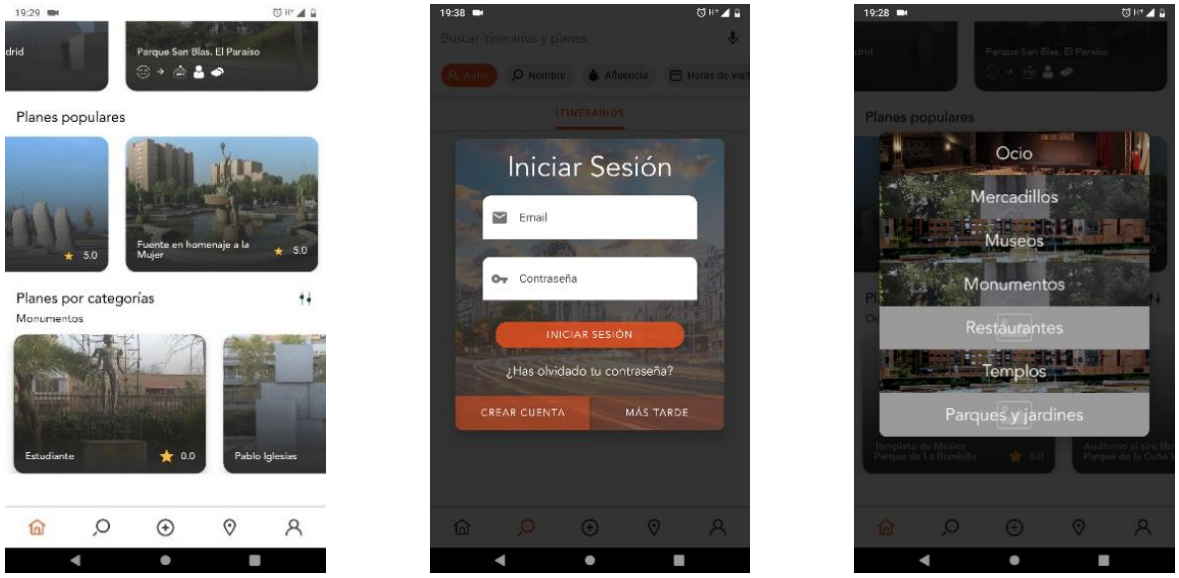


Figura 95 - Pantallas home, *popup* inicio de sesión y filtro por categorías

En la pantalla principal se pueden seleccionar los diferentes planes e itinerarios que aparecen. Una vez seleccionados, aparecerá el nombre, la descripción y los comentarios de estos (ver Figura 95).

En el caso de los planes un botón para ir a Google Maps y ver su ubicación, un botón de compartir, su horario y afluencia semanal, y los planes relacionados que hay con éste. Además, si este lugar está cerrado por alguna causa, se mostrará un mensaje con el motivo. Se pueden observar estas pantallas en la Figura 96.

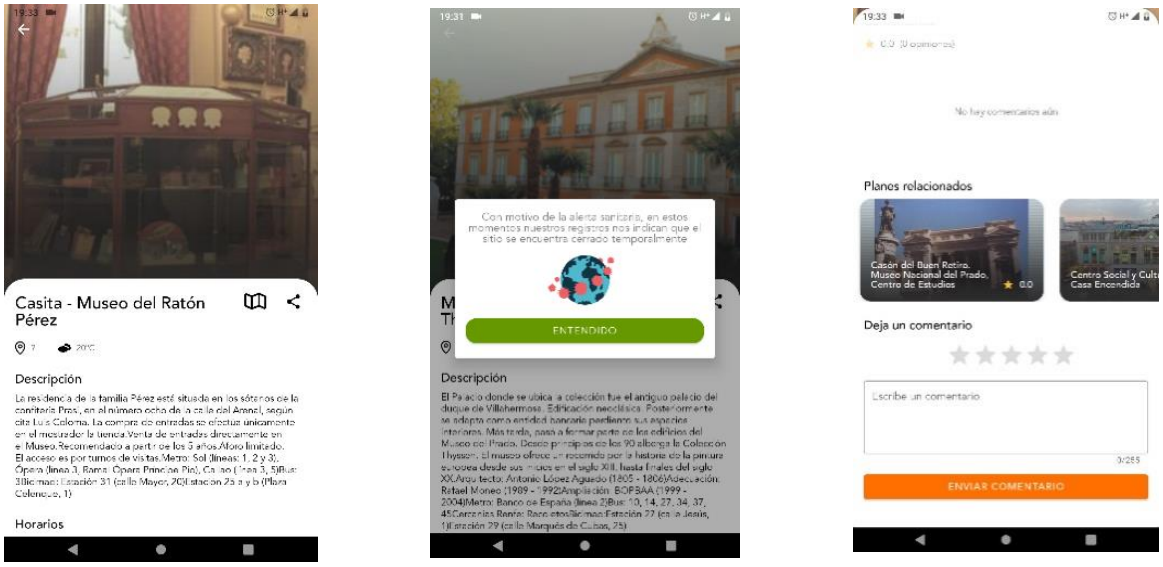


Figura 96 - Detalle plan

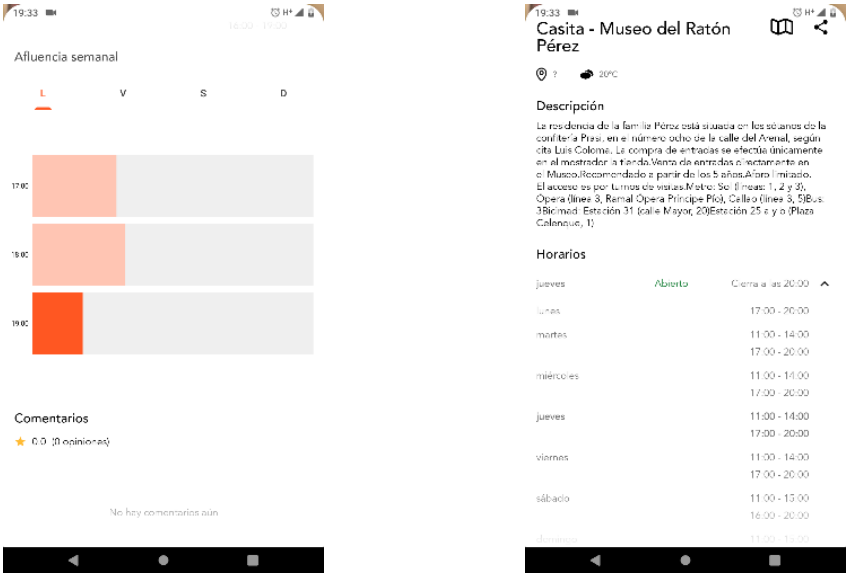


Figura 97 - Afluencias y horarios de un plan

Y en el caso de los itinerarios, aparecen las actividades que están dentro del itinerario (ver Figura 98).

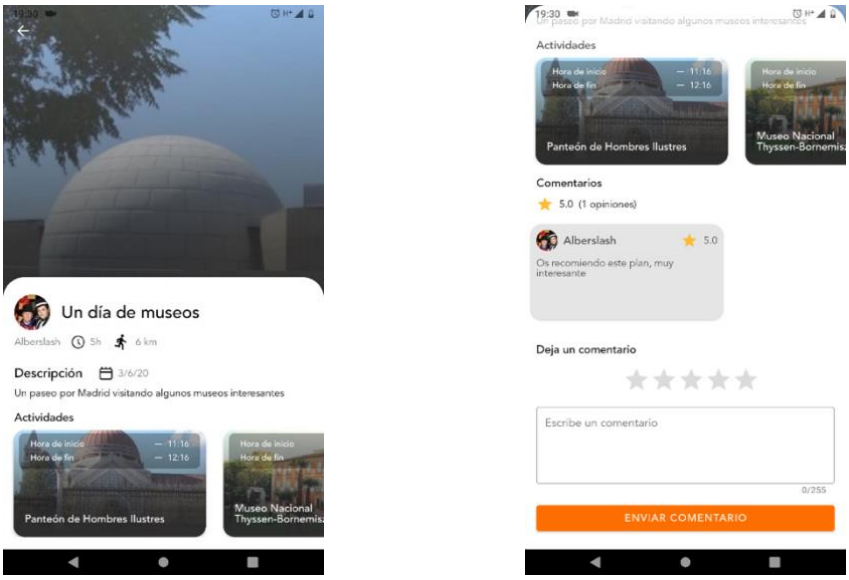


Figura 98 - Detalle itinerario

Si se vuelve a la pantalla principal y se hace clic sobre el icono de la lupa, se accede al apartado de buscar, ahí se pueden buscar tanto planes como itinerarios con unos parámetros ya sean el nombre, el autor para los itinerarios, el horario, o la afluencia. Los parámetros de la afluencia, horario y fecha se pueden modificar para realizar búsquedas más exhaustivas. Se pueden observar estas pantallas en la Figura 99.

En caso de hacer clic en los elementos resultantes de la búsqueda, se pasará ver en detalle el elemento seleccionado (el itinerario o el plan).

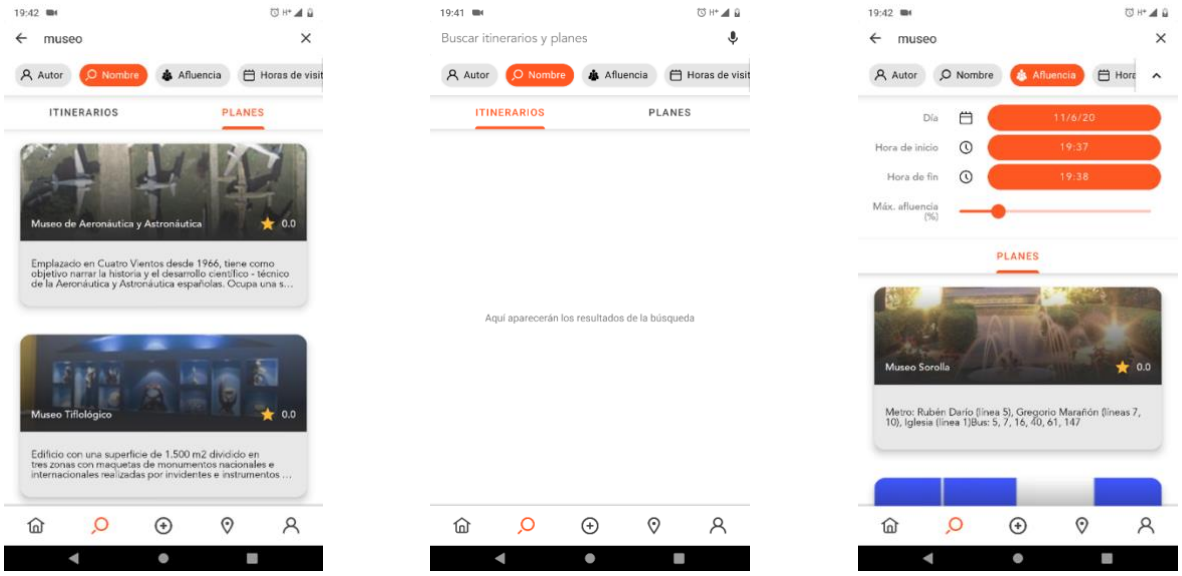


Figura 99 – Búsqueda de planes e itinerarios

De nuevo en la pantalla principal, hay otros 2 iconos, el del usuario, situado a la derecha del todo es el que dirige al usuario al perfil, donde aparecen los itinerarios que el usuario ha creado. También se puede desplegar un mapa pinchando en el icono situado a la izquierda del icono del usuario y aparece un mapa con los planes disponibles que se encuentran más cerca del usuario (ver Figura 100).

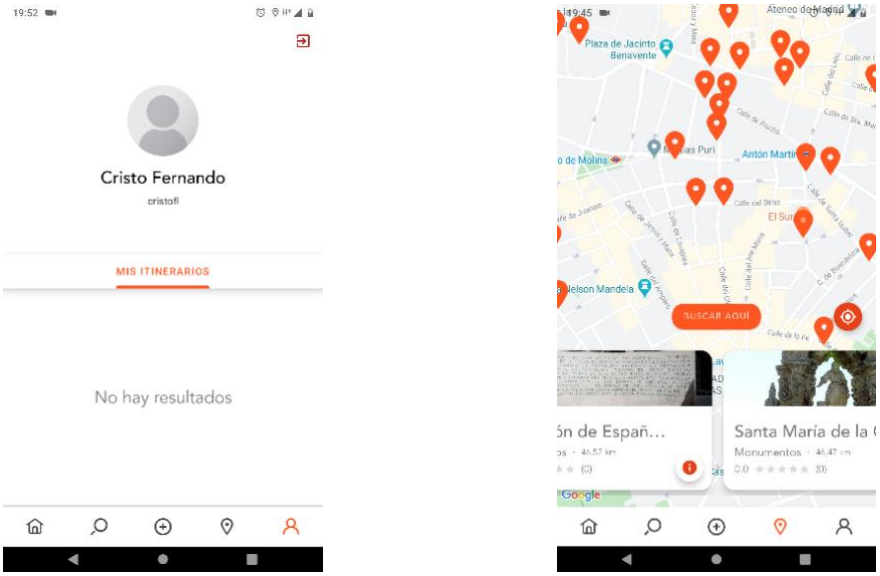


Figura 100 - Perfil de usuario y mapa interactivo

El último punto que destacar es la creación de itinerarios, para poder crear un itinerario es necesario que el usuario haya iniciado sesión.

El proceso consta de varios pasos, primero el usuario debe de introducir una fecha y una franja horaria en la cual quiere realizar actividades. En ese momento aparecerá una lista horizontal con los planes que el sistema recomienda para ese día en esas horas.

Si se selecciona alguno de esos planes, se irán añadiendo a otra lista auxiliar visible en todo momento para el usuario para que éste pueda visualizar los planes que vaya seleccionando, si alguno de los planes en esta lista es clicado, se eliminara.

El segundo paso, consiste en que el usuario seleccione planes en función a las categorías disponibles, puede ir cambiando entre estas para que vayan apareciendo diferentes planes.

El tercer y último paso es que el usuario seleccione a que categorías quiere que pertenezca el itinerario.

Finalmente, aparece una lista vertical con los planes seleccionados por el usuario con las horas provisionales a las que realizara las actividades, estas horas pueden ser modificadas. El usuario debe introducir un nombre y una descripción del itinerario, entonces puede pulsar el botón para crear el itinerario y guardarlo en su perfil para cuando quiera visualizar. Puede visualizar el proceso en la Figura 101.

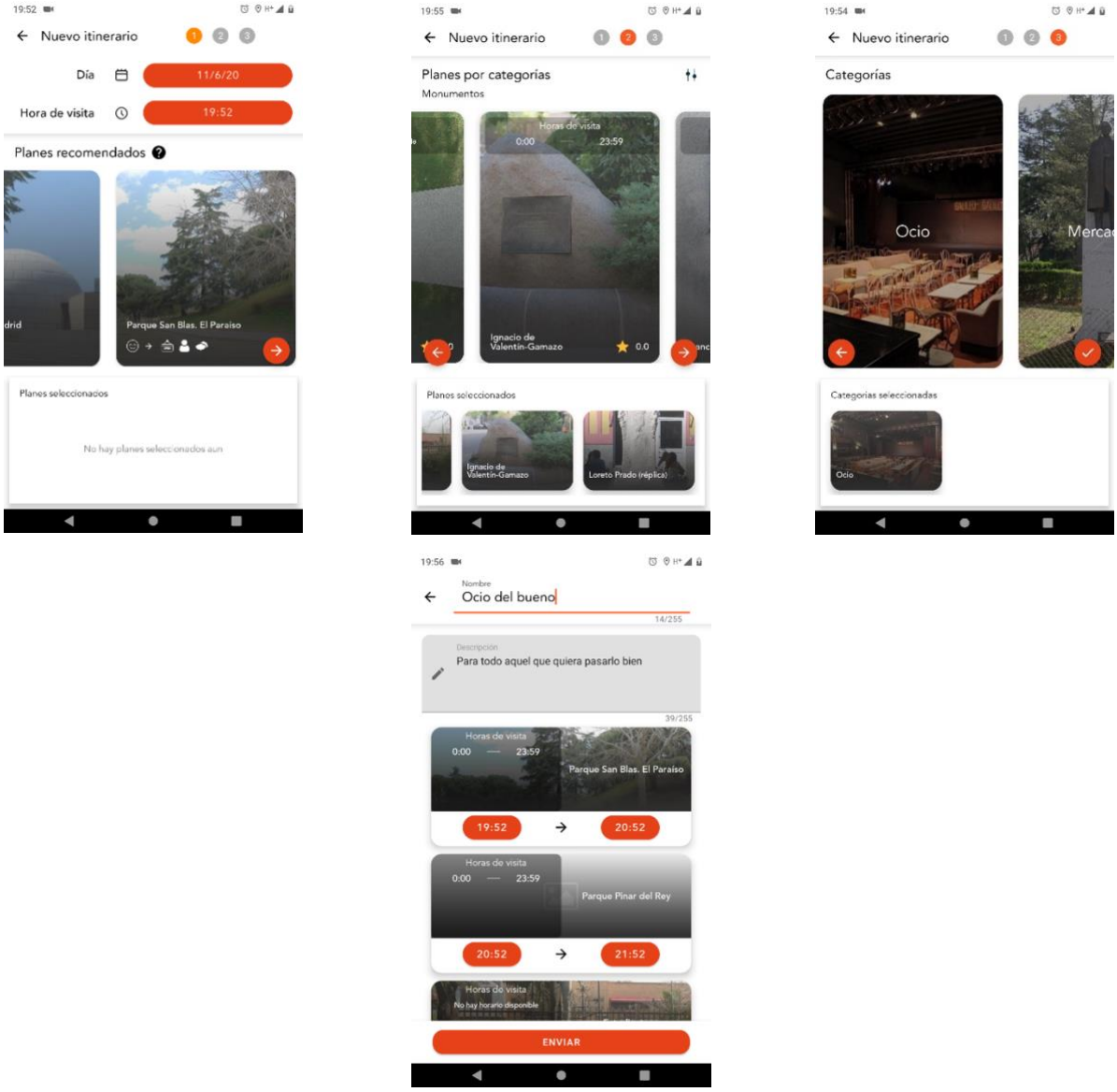


Figura 101 - Pasos creación itinerario

ANEXO III. Preguntas de la evaluación

A continuación, se muestran las preguntas realizadas a los usuarios en la evaluación de la usabilidad realizada.

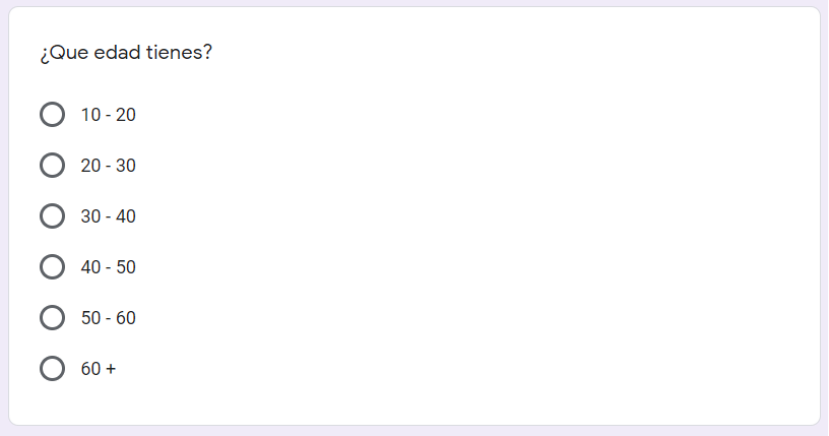


Figura 102 - Primera pregunta formulario, Edad del usuario

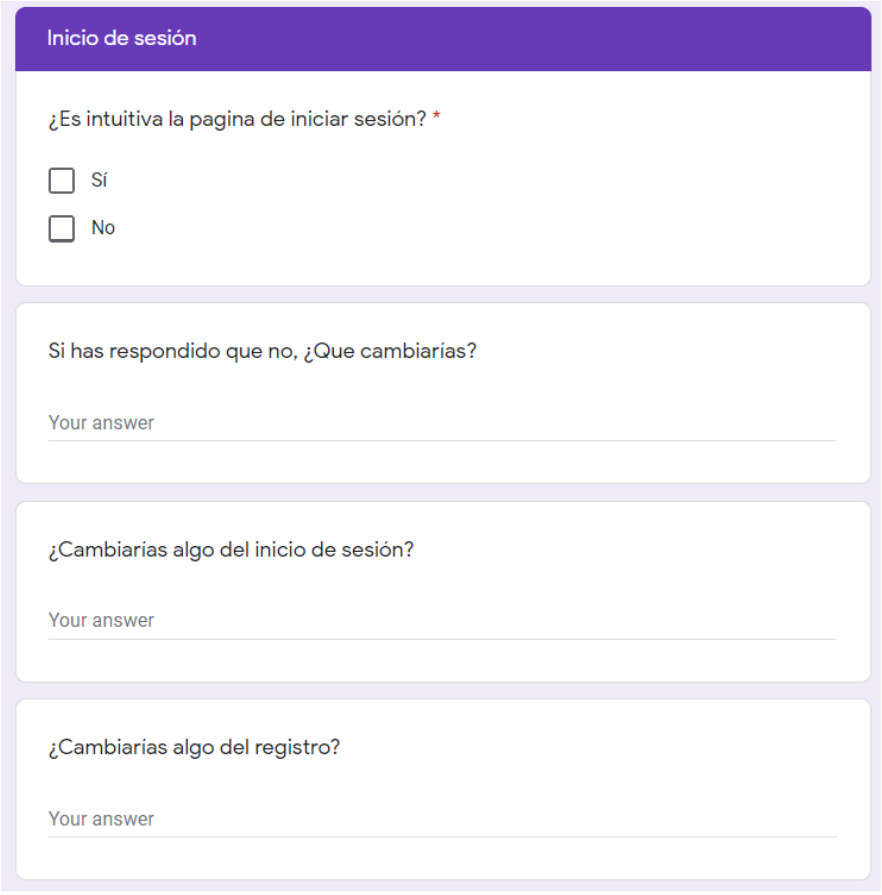


Figura 103 - Preguntas sobre la característica de inicio de sesión y registro

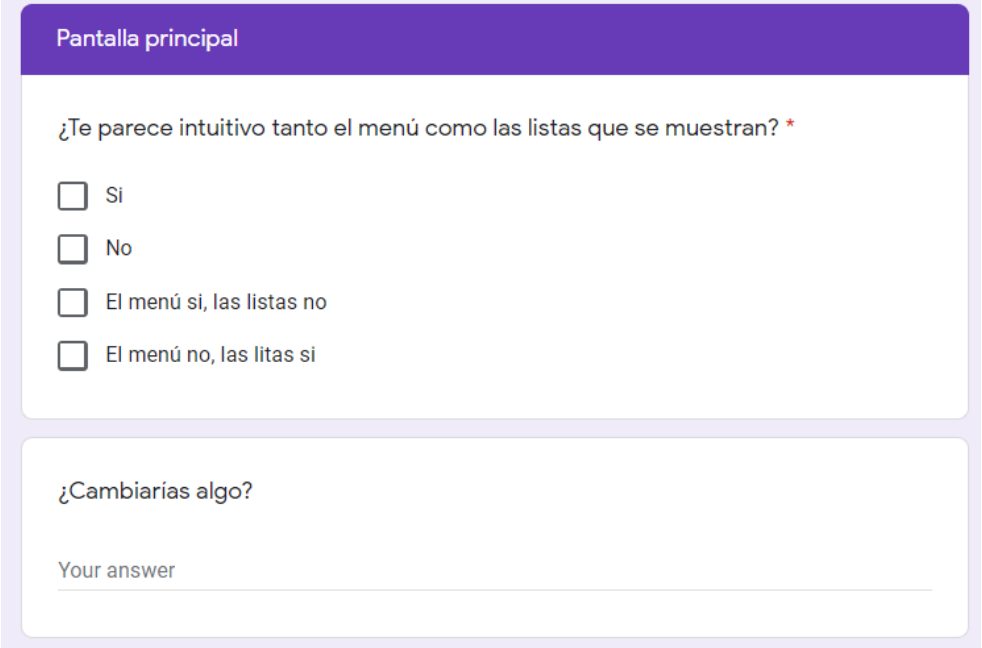


Figura 104 - Preguntas sobre la pantalla principal

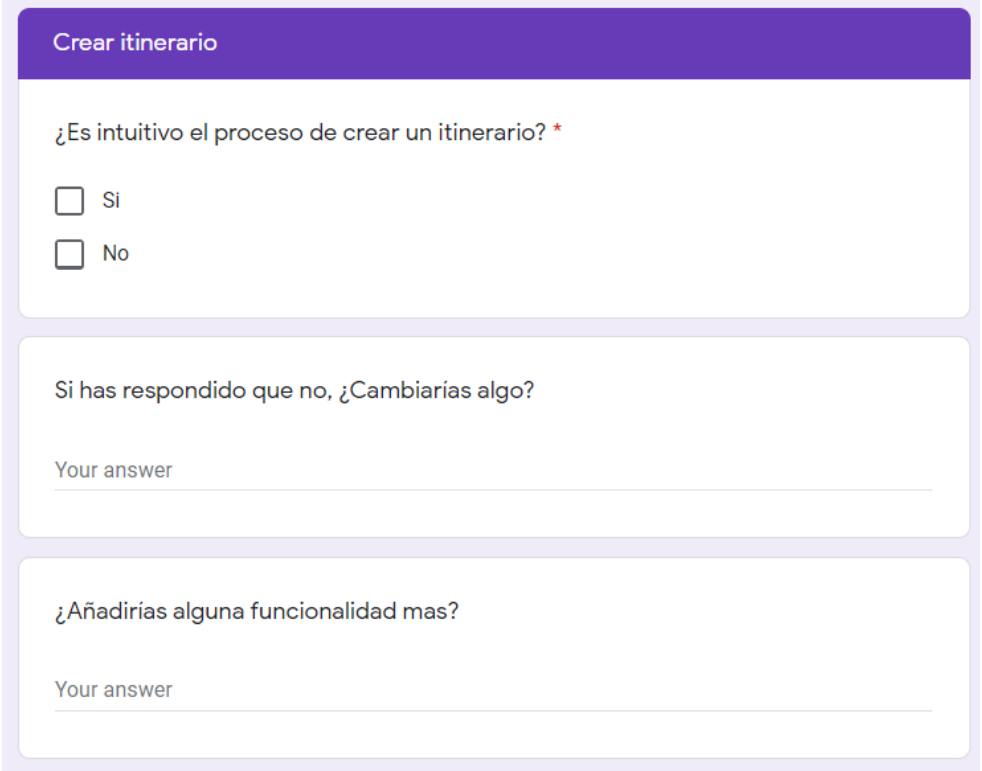


Figura 105 - Preguntas sobre crear itinerario

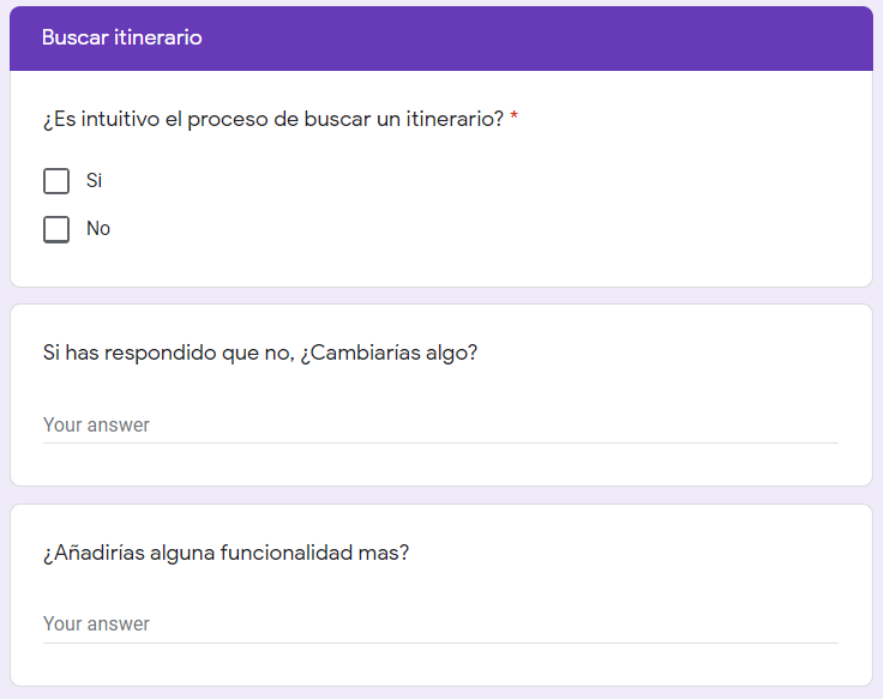


Figura 106 - Preguntas sobre buscar itinerario

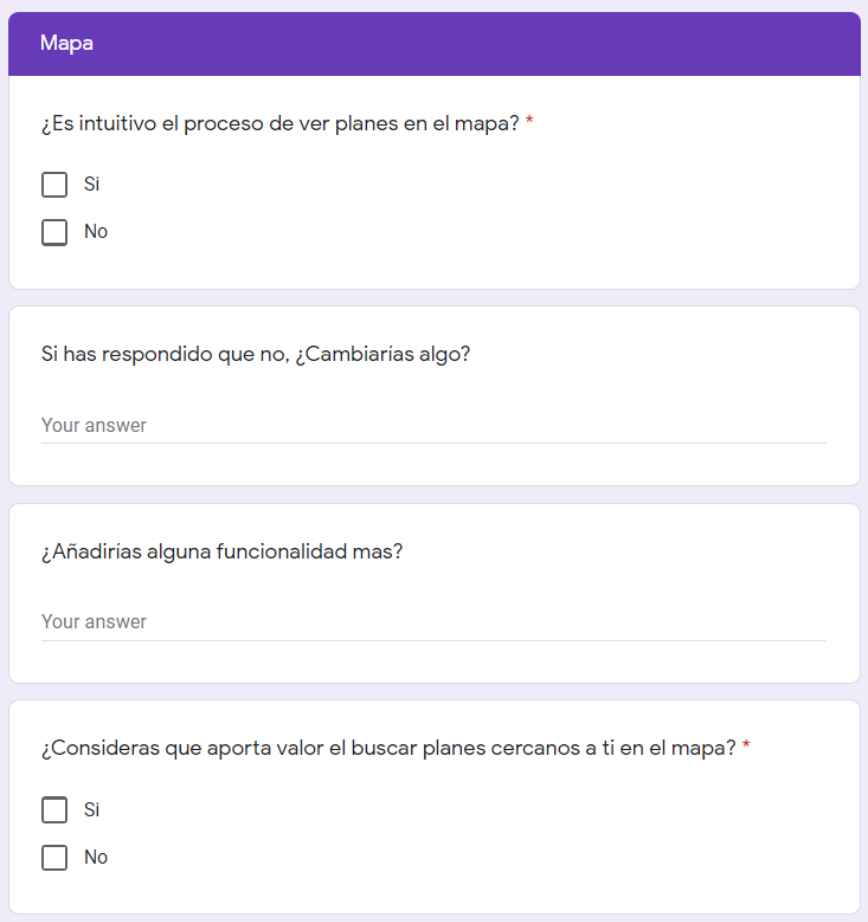


Figura 107 - Preguntas sobre el mapa interactivo

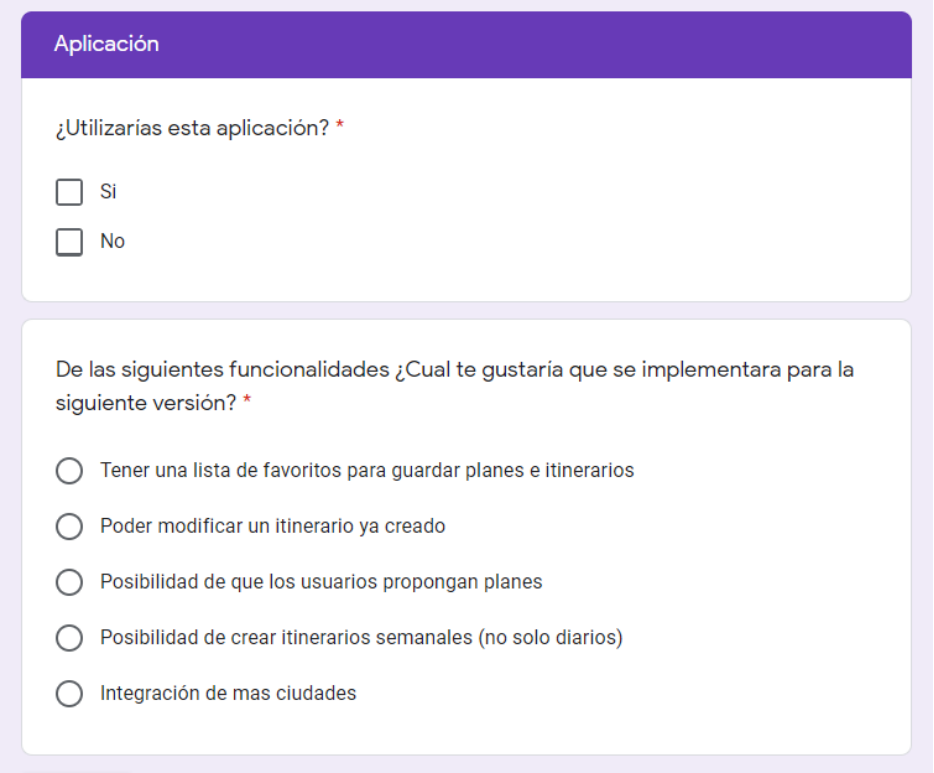


Figura 108 - Preguntas sobre el trabajo futuro y el uso de la aplicación