Desarrollo de una aplicación Android de valor añadido para el turismo de Madrid

Development of an Android value-added service for Madrid tourism

Trabajo de Fin de Grado
Curso 2020–2021

Autores
Juan Antonio Escobar de los Ángeles
Daniel Nasim Santos Ouafki
Jin Tao Peng Zhou
Jin Wang Xu

Director
Antonio Sarasa Cabezuelo

Tutor
Antonio Sarasa Cabezuelo

Trabajo de Fin de Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de una aplicación Android de valor añadido para el turismo de Madrid

Development of an Android value-added service for Madrid tourism

Grado en Ingeniería Informática

Autores
Juan Antonio Escobar de los Ángeles
Daniel Nasim Santos Ouafki
Jin Tao Peng Zhou
Jin Wang Xu

Director
Antonio Sarasa Cabezuelo

Tutor
Antonio Sarasa Cabezuelo

Convocatoria: *Junio 2021*

Trabajo de Fin de Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

7 de junio de 2021

Un experto es alguien que te explica algo sencillo de forma confusa de tal manera que te hace pensar que la confusión sea culpa tuya.

William Castle

Agradecimientos

0.1 Daniel Nasim Santos Ouafki

Quiero agradecer a mi familia por su cariño y apoyo en este nuevo e importante período que ha constituido la universidad. También a los profesores que he tenido, tanto en la universidad como fuera de ella, por haber fomentado en el desarrollo de mi curiosidad y conocimientos.

0.2 Juan Antonio Escobar de los Ángeles

Quiero agradecer a mi familia el apoyo que me ha dado a lo largo de mi carrera académica y que siempre haya hecho todo lo posible para que pueda seguir estudiando.

0.3 Jin Tao Peng Zhou

Lo primero de todo, quiero agradecer a mis compañeros del TFG por el buen trabajo que han hecho.

También quiero agradecer a mi familia, en especial a mi hermana Qi Ting Peng Zhou, por el apoyo constante que he recibido de ellos. Sin ellos no estaría aquí en este momento.

Por último quiero agraceder a todos los profesores que he tenido a lo largo de la carrera por brindarme muchos conocimientos.

0.4 Jin Wang Xu

Personalmente quiero agradecer lo mucho que ha significado para mí poder trabajar una última vez con mis compañeros. También a mi pareja y familia por el entusiasmo y apoyo constante durante toda mi carrera universitaria.

0.5 Todos

Queremos agradecer a nuestro tutor del proyecto del TFG, Antonio Sarasa por ayudarnos y aconsejarnos en todo momento durante el desarrollo de nuestro último proyecto como estudiantes universitarios de la carrera de Ingeniería Informática.

También queremos dar las gracias a todas las personas que nos han ayudado durante el proceso de la evaluación de la aplicación que nos dedicaron su tiempo para ayudarnos con el proyecto.

Resumen

El presente trabajo consiste en una aplicación Android para facilitar el turismo en Madrid.

Para obtener opiniones reales de ciudadanos madrileños, se ha implementado un sistema de valoraciones en la aplicación.

El usuario dispone de un sistema de búsqueda por texto, por voz, por cercanía o por categorías y puede seleccionar los lugares para ver sus detalles y realizar reseñas sobre estos.

Cabe mencionar que se ha realizado un sistema de amigos para poder recomendar lugares del portal o creados por otros usuarios a amigos de la aplicación.

Palabras clave

- Turismo
- Twitter
- Android
- APIs
- Lugares
- Flask
- ETL
- Madrid
- MVVM

Abstract

The present work consists of an android application to promote and make tourism easier in Madrid.

In order to get real opinions from Madrid citizens, the application is implemented with a rating system.

The user has a text and voice search system that could be used by nearness or categories. The user also could select places in order to see their details and make ratings about others.

It is worth mentioning that a friend system has been created to be able to recommend places on the portal or places created by other users.

Keywords

- Tourism
- Twitter
- Android
- APIs
- Places
- Flask
- ETL
- Madrid
- MVVM

Índice

Lista de Figuras

Lista de Tablas

Chapter 1

Introduction

1.1 Introduction

Nowadays, technologies have become essential for society. Many companies, use technologies to perform tasks faster, investing less money. Inspired by these concepts, a good amount of technologies were used for the purpose of communicating in social media where this information can be use in other types of services.

With the data offered by these technologies, an application has been developed that helps users to further enrich their culture about Madrid, for example, with the city's monuments information. Thanks to the application, users will be able to recommend places to their friends in order to discover new places in their city with a navigation system from wherever they are.

1.2 Motivation

Before the coronavirus pandemic, tourism was the order of the day. During any time of the year, people from all over the world travel to see new places, try different cuisines, expand their culture, etc.

Despite all the restrictions, once the situation is stabilized, the people are hopeful and certain that tourism will once again fill the streets with people. For this, a multitude of websites, apps and other services will be used to allow tourists to visit the different places.

Usually, tourists who travel to Madrid have problems looking for places or monuments leading to poor trip planning and a bad experience.

Although it may not seem like it, this can give many serious problems, the tourist can get into inappropriate streets where their physical integrity is at risk, they can be scammed by restaurants or people on the street, or even get lost in the city without knowing how to return or what to do.

Tourists need, in some way, the help of citizens themselves to know what places to visit and where to eat to avoid bad experiences.

With this problems in mind, it has been decided to make an application where the users can visit peacefully Madrid, taking into account the opinions of the app and Twitter, so the stay in Madrid is more friendly and pleasant. So the motivation to make the application is to make easier to the user to choose places of interest in Madrid, keep a history of places visited and be able to recommend places to other users.

1.3 Goals

The goal is to develop an application capable of providing information about the tourist spots of Madrid and through text and voice to be able to search these places.

Another objective is to provide realistic data on places of interest in Madrid with an application that is easy to use and accessible. With the server part easily applicable to other devices with different interfaces and the development of an Android application with smartphones's common features.

In this way, information provided by the city hall of Madrid can be accessed in a clustered and organized way, something that is very difficult from the original webpage itself.

1.4 Work plan

First of all, we must fulfill a research of the technologies that we are going to apply:

- Discuss what is the best schema to use for both the application and the database
- API technologies
- Android libraries for an API connection and handling of HTTP requests
- Android SDK elements

The application will only be available on Android platforms. An extended investigation will be carried out about basic programming in Android with Android Studio as the IDE to understand how it works and start planning the design and implementation of the application.

Once the objectives are clear, the first step will be the creation and implementation of a web service where valuable data is extracted through HTTP requests for the correct functionality of our application.

At the same time, it is also necessary to collect real data on tourism within Madrid.

Later on, an user interface and a back-end application will be created for the interaction with the extracted data from the web service.

Once the project comes to an end, testing and evaluation will be carried out with actual people to ensure the application is free from errors

Capítulo 1

Introducción

1.1 Introducción

En la actualidad, las tecnologías se han convertido en algo imprescindible para la sociedad. Se utilizan para los negocios de una empresa, donde utilizan las tecnologías para realizar tareas más rápido e invirtiendo menos dinero. También se usa para que las personas se comuniquen en redes sociales, en las cuales se puede usar esta información para otro tipo de servicios.

Con los datos que ofrecen estas tecnologías, se ha desarrollado una aplicación que ayude a los usuarios a enriquecer más su cultura sobre Madrid con información sobre los monumentos de la ciudad. Gracias a la aplicación, los usuarios podrán recomendar lugares a sus amigos para que así puedan conocer nuevos sitios en su ciudad con un sistema de navegación desde donde se encuentren.

1.2 Motivación

Antes de los trágicos acontecimientos provocados por la pandemia del coronavirus, el turismo estaba a la orden del día. Durante todas la épocas del año, gente de todo el mundo viajaba para conocer nuevos lugares, probar distintas gastronomías, ampliar su cultura, etc.

A pesar de todas las restricciones, una vez la tormenta haya amainado, las personas tienen la esperanza y la certeza de que el turismo volverá a llenar las calles de gente. Para ello se usarán multitud de webs, app y otros servicios para permitir a los turistas visitar los diferentes lugares de una ciudad.

Generalmente los turistas que viajan a Madrid tienen problemas para buscar lugares o monumentos y esto conlleva a una mala planificación del viaje y una mala experiencia.

Aunque no lo parezca, esto puede dar serios problemas. El turista se pue-

de adentrar en calles inadecuadas donde peligra su integridad física, puede ser estafado por restaurantes o personas de la calle, o incluso perderse en la ciudad sin tener idea de cómo volver o qué hacer.

Los turistas necesitan de alguna forma la ayuda de los propios ciudadanos para conocer qué lugares visitar, dónde comer, para evitar malas experiencias.

Con estos problemas en mente, se ha decidido crear una aplicación donde los usuarios puedan visitar tranquilamente la ciudad de Madrid, teniendo en cuenta las opiniones de la propia aplicación y las de Twitter acerca de los lugares, alojamientos y restaurantes para que su visita a Madrid sea más acogedora y agradable. Por lo que la motivación para hacer la aplicación es facilitar al usuario a elegir lugares de interés de Madrid, mantener un historial de lugares visitados y poder recomendar lugares a otros usuarios.

1.3 Objetivos

El objetivo es desarrollar una aplicación capaz de proporcionar información acerca de los puntos turísticos de Madrid y poder buscar estos lugares con un buscador por texto y voz.

También se quiere proporcionar datos realistas sobre los lugares de Madrid a los que poder ir, con una aplicación que sea fácil de usar y accesible. Con la parte del servidor fácilmente aplicable a otros dispositivos que tengan otras interfaces y con el desarrollo de una aplicación en Android con utilidades propias de los dispositivos móviles.

De esta manera, se puede acceder a una amplia información que ha sido proporcionada por datos del Ayuntamiento de Madrid de una manera agrupada y ordenada, algo que es muy difícil desde la propia página web.

1.4 Plan de trabajo

Para empezar, se debe realizar una investigación sobre las tecnologías que se van a utilizar:

- Qué estructura usar tanto para la aplicación como para la base de datos.
- Tecnologías de la API.
- Librerías Android para la conexión a la API y manejo de peticiones HTTP.
- Elementos del SDK de Android.

Como la aplicación móvil estará únicamente disponible en plataformas Android, se realiza una investigación acerca de la programación en Android con **Android Studio [30]** para conocer el funcionamiento y planificar el diseño e implementación de la aplicación.

Una vez claro los objetivos, se empieza con la creación e implementación de un servicio web donde se extraen datos significativos mediante peticiones HTTP para nuestra aplicación y analizar estos datos.

Al mismo tiempo, es necesario también recopilar datos reales sobre el turismo dentro de Madrid.

Posteriormente se debe realizar la interfaz del usuario y la aplicación con la que va a interactuar con los datos extraídos.

Finalmente se hacen pruebas y se arreglan errores que se encuentren.

Capítulo 2

Estado del arte

El turismo es un tema bastante común a día de hoy, por lo que no es sorpresa que ya existan múltiples aplicaciones en distintas plataformas para facilitar las visitas turísticas. Entre ellas, se encuentran algunas aplicaciones que se asemejan al proyecto realizado.

Como aplicaciones de páginas web, existen algunas como:

- 101viajes [47]
  - Es una página web en la cual los usuarios pueden ver lugares de interés en Madrid: museos, mercados, restaurantes, bares, etc. También tiene una sección para alojamientos y otra para realizar excursiones.
- Disfruta Madrid [25]
  - Es una página web para turistas nacionales que quieren visitar Madrid, en la cual se ofertan diversas actividades como si de un guía turístico se tratase. Ofrece consejos sobre las tiendas, lugares, alojamientos, e incluso una sección para estancias cortas llamada "Madrid en dos días". También tiene una versión para Android e IOS. Existen también otras guías para Valladolid y Sevilla del mismo tipo, las cuales se indican en la propia página.

Como aplicaciones móviles, encontramos aplicaciones como:

- Madrid Turismo [21]
  - Es una aplicación para Android, en la cual se pueden ver lugares, observar detalles de un lugar, marcarlos como visitados o no visitados, email y teléfono de contacto, diversas fotografías e incluso audios cortos sobre los lugares. También contiene información

acerca de los planos de las redes de transportes y sobre el clima en Madrid.

- MADRID City Guide, Offline Maps, Tours and Hotels [63]
  - Es una aplicación móvil donde muestra lugares de Madrid. Puedes buscar lugares por nombre o por categorías, ver recomendaciones de otros turistas del lugar, guardar lugares para más tarde visitarlos y buscar entradas de pago para visitar el lugar.

Cabe destacar que todas las aplicaciones mencionadas tienen en común que muestran lugares para visitar en Madrid y un mapa que muestra donde se ubica el lugar.

Capítulo 3

Especificación de requisitos

En este capítulo se van a explicar los requisitos de la aplicación.

3.1 Actores

En la aplicación se han definido los siguientes actores:

- **Usuarios no registrados**: Es aquel que no dispone de cuenta en el sistema. Puede acceder a las funciones e información pública.
- **Usuarios registrados**: Es aquel que dispone de cuenta en el sistema. Tiene acceso a todas las funciones como usuario.
- **Administrador**: Es la persona encargada del mantenimiento de la aplicación.

3.2 Módulos

La funcionalidad de la aplicación se ha agrupado en módulos funcionales:

- Módulo usuario.
- Módulo lugares.

En los siguientes apartados se desarrollan cada uno de estos módulos.

3.2.1 Módulo Usuario

En este módulo se ha agrupado toda la funcionalidad referente a la gestión de la información de los usuarios.

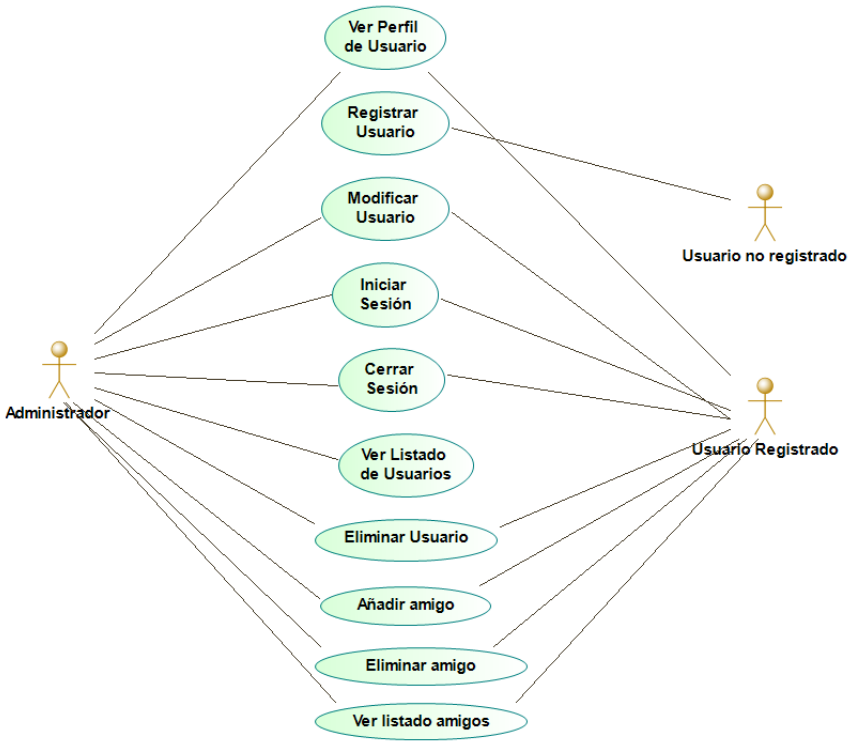


Figura 3.1: Módulo usuario

Como se muestra en la figura 3.1, un usuario no registrado solo puede registrarse en la aplicación ya que el resto de funciones es necesario haber iniciado sesión. Un usuario registrado puede ver su perfil, modificar su perfil, iniciar sesión, cerrar sesión, borrar su propia cuenta y también tiene un sistema de amigos donde puede añadir, eliminar o ver la lista de amigos. El administrador además puede ver perfiles de otros usuarios y eliminar cuentas de usuarios que no se comporten bien.

3.2.2 Módulo Lugares

En este módulo se ha agrupado toda la funcionalidad referente a la gestión de la información e interacción de los lugares.

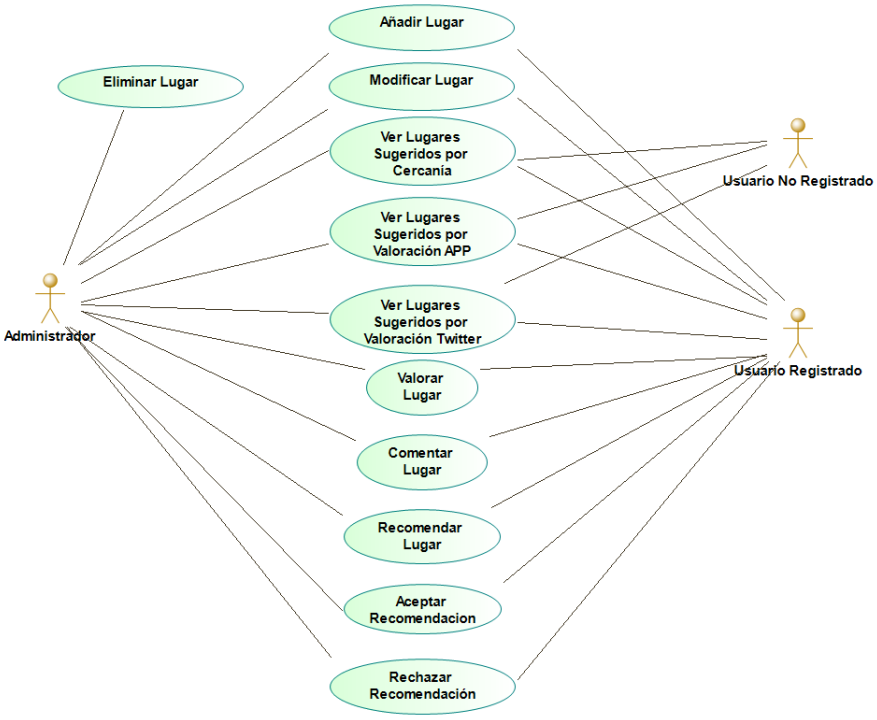


Figura 3.2: Módulo Lugares

En la figura 3.2 se muestra las acciones que pueden realizar los usuarios. El usuario no registrado puede ver lugares que la aplicación sugiere por cercanía al lugar que se encuentra el usuario, ver lugares que mejor han calificado los usuarios con una cuenta dentro de la aplicación y ver lugares que recomienda la gente que comenta en Twitter. El usuario registrado también puede añadir un lugar, modificar un lugar, valorar un lugar que tiene relación con los lugares mejores calificados dentro de la aplicación, comentar un lugar y recomendar un lugar a otro usuario con el sistema de amigos. El administrador además puede borrar un lugar.

3.3 Diagramas de flujo

En este apartado se especificarán algunos diagramas de flujo de la aplicación.

3.3.1 Crear lugar

Dentro de la aplicación, en la pantalla principal, se muestra un icono en la parte superior izquierda del menú para la creación de un nuevo lugar. Una vez pulsado el botón, se muestra un formulario donde el usuario deberá rellenar. Todos los campos son obligatorios excepto las imágenes. Una vez

rellenado los campos, el usuario debe pulsar el botón de crear el lugar y si ningún campo es erróneo, se crea el nuevo lugar y se redirige a la pantalla principal.

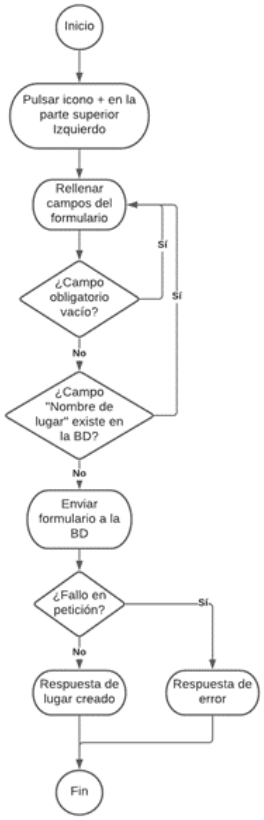


Figura 3.3: Diagrama de creación de un lugar

La figura 3.3, representa el diagrama de flujo correspondiente a la creación del lugar.

3.3.2 Recomendar lugar

Al pulsar en cualquier lugar de la lista de lugares, la aplicación muestra una nueva pantalla con información más detallada del lugar. En la nueva pantalla, hay un botón en el menú con el icono de tres puntos refiriéndose textualmente a "más opciones" que al pulsar en ella muestra una opción de recomendar un lugar. Una vez se pincha en la opción, se muestra la lista de amigos que el usuario tiene dentro la aplicación. Una vez que muestre la lista, se selecciona a uno o varios usuarios para recomendar el lugar y se pulsa el botón de enviar. A cada usuario que ya se le había mandado la recomendación, la aplicación devuelve una respuesta sin éxito. En caso

contrario, se devuelve una respuesta con éxito. En caso de que el usuario no tenga amigos en la aplicación, se muestra un texto donde se describe que no hay amigos para recomendar.

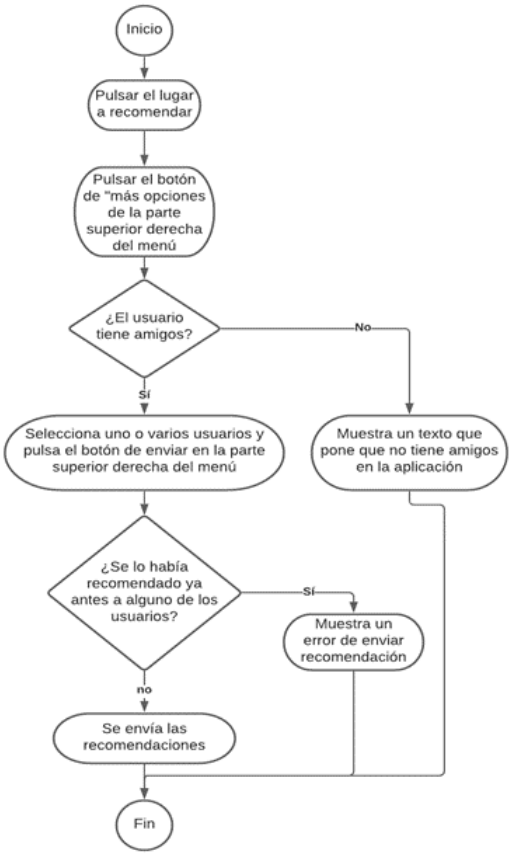


Figura 3.4: Diagrama de recomendación de un lugar

La figura 3.4 muestra el diagrama de flujo correspondiente a la recomendación de un lugar.

3.3.3 Aceptar o rechazar petición de amistad

En la pantalla principal se muestra un icono de menú lateral que se puede mostrar pulsando en el icono o desplazando lateralmente el dedo hacia la derecha. En el menú lateral aparece un apartado de amigos. En el apartado de amigos se muestra un TabLayout con las opciones de "lista de amigos" y de "peticiones de amistad". En el apartado de peticiones de amistad se muestra una lista de peticiones de amistad pendientes por aceptar o rechazar. Tanto si se acepta o se rechaza la petición de amistad se muestra un mensaje indicando que se ha realizado la petición.

En caso de que no tenga peticiones de amistad pendientes se muestra un texto indicando que no tiene peticiones pendientes.

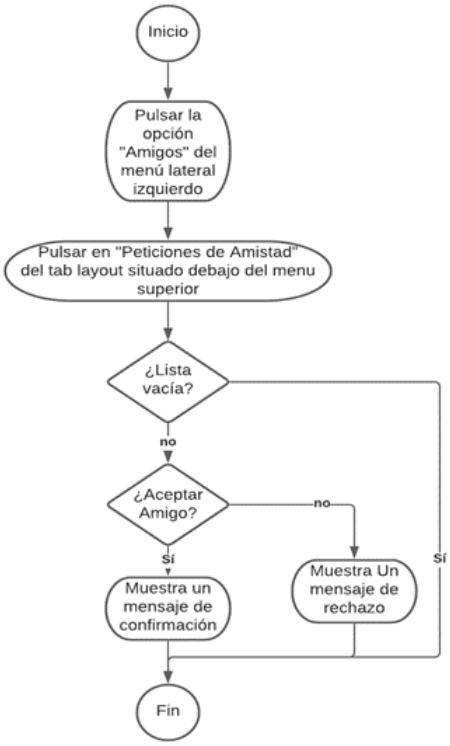


Figura 3.5: Diagrama de peticiones de amistad

En la figura 3.5 se observa el diagrama de flujo correspondiente a aceptar o rechazar peticiones de amistad.

3.4 Análisis de requisitos

En esta fase se realiza el análisis de requisitos de la aplicación, en la cual se tuvo que contar con los requisitos funcionales y no funcionales:

- Los requisitos funcionales son los referidos a las acciones del usuario, lo que puede hacer, y las funcionalidades que tendrá la aplicación.

- Los requisitos no funcionales son los referidos a la manera en la que se desarrolla la aplicación, tales como las tecnologías, lenguajes de programación y plataformas a usar.

3.4.1 Requisitos funcionales

Los requisitos funcionales de la aplicación han sido los siguientes:

- Un usuario nuevo puede crear una cuenta en la aplicación a través de un formulario.
- Un usuario registrado puede iniciar sesión en la aplicación con su nombre de usuario y contraseña.
- Un usuario puede cerrar sesión.
- Un usuario puede ver su información desde el perfil de usuario.
- Un usuario puede cambiar su contraseña desde el perfil de usuario.
- Un usuario puede cambiar la información de su cuenta desde el perfil de usuario.
- Un usuario puede borrar su cuenta desde su perfil de usuario.
- Un administrador puede ver el listado de usuarios de la aplicación.
- Un administrador puede borrar la cuenta de un usuario de la aplicación.
- Un usuario puede ver los lugares sugeridos por la aplicación según su cercanía.
- Un usuario puede ver los lugares sugeridos por la aplicación según su categoría.
- Un usuario puede ver los lugares sugeridos por la aplicación según su valoración en Twitter.
- Un usuario puede buscar por texto un lugar.
- Un usuario puede buscar por voz un lugar.
- Un usuario puede ver un lugar concreto.
- Un usuario puede añadir a favoritos un lugar.
- Un usuario puede marcar un lugar como visitado.
- Un usuario puede ver un lugar en el mapa y navegar hacia él.
- Un usuario puede ver los comentarios que existen sobre un lugar.
- Un usuario puede ver las valoraciones de un lugar.
- Un usuario puede comentar sobre un lugar y añadir su valoración.

- Un usuario puede añadir un amigo a través de un formulario.
- Un usuario puede ver su lista de amigos.
- Un usuario puede aceptar o rechazar una solicitud de amigo.
- Un usuario puede recomendar un lugar a un amigo a través de un formulario.
- Un usuario puede ver las recomendaciones recibidas y aceptarlas o rechazarlas.
- Un usuario puede ver los lugares que ha visitado
- Un usuario puede ver los lugares que tiene pendientes por visitar
- Un usuario puede cambiar el idioma de la aplicación

3.4.2 Requisitos no funcionales

Los requisitos no funcionales de nuestra aplicación han sido los siguientes:

- La aplicación es desarrollada para Android.
- Se utiliza el lenguaje de programación Java para la aplicación Android.
- Se utiliza el lenguaje de programación interpretado Python para el servidor.
- Se utiliza el framework Flask [53] para realizar el servidor en Python.
- Se utiliza Flask-SQLAlchemy [20] como ORM en el servidor.
- Se utiliza Mapbox [41] para la realización de mapas en la aplicación.
- Se utiliza el patrón de diseño MVVM [62] para separar vistas y modelos.
- Se utilizan los patrones DTO y DAO para tratar los modelos de datos.
- Se realizan las conexiones con el servidor desde la App a través de OkHttp [57].
- La base de datos se realiza en SQL.
- Se utiliza PHPMyAdmin [23] para manejar la base de datos.

Capítulo 4

Tecnologías

En este capítulo se van a explicar las herramientas utilizadas en la aplicación.

4.1 API REST

Es una de las arquitecturas más utilizadas para implementar un servicio web. Para poder desarrollar el servicio web, se utiliza el micro-framework que se explica a continuación.

Un servicio web sirve para facilitar una comunicación entre dos aplicaciones, en este caso, entre la aplicación Android y la base de datos. Esta comunicación se basa en enviar peticiones HTTP para recibir unas respuestas en formato JSON, con los recursos demandados por la aplicación. Se utiliza API REST ya que son potentes y ligeras, así que es ideal para recibir datos lo antes posible. Existen otros servicios webs que simulan el comportamiento de una API, como por ejemplo SOAP. Comparando API REST frente a otros servicios webs, obtiene una ventaja por su simplicidad y fácilidad de uso. SOAP por ejemplo, es muy usado en las empresas actuales ya que también ofrecen una capa de seguridad en las peticiones, pero lo hace más pesado y no presenta una flexibilidad igual que API REST.

4.2 Flask [53]

Es un framework cuya principal ventaja frente a otros como Django es su simplicidad a la hora de desplegarlo, ya que con unas pocas líneas de código se puede desarrollar el servidor web indicando la ruta a la que se debe dirigir el usuario y la función que se ejecutará. Otra gran ventaja es su tamaño, ya que por ejemplo Django ocupa mucho más espacio para realizar tareas simples, por lo que se suele decir que Flask es un "micro" framework. También aporta una estructura al proyecto ya que todos los proyectos que usan Flask

se construyen de la misma manera haciendo más fácil así la colaboración entre desarrolladores. Además tiene una serie de extensiones para ampliar su funcionalidad si así se desea, como por ejemplo flask-Sqlalchemy. Flask proporciona una manera simple de lanzar un servidor de desarrollo para poder hacer pruebas con la comunicación del Frontend, mostrando las peticiones y las respuestas del servidor, además, es sencillo para la creación de API REST y está desarrollado en código abierto, el cuál se puede encontrar en github.

4.3 Flask-SQLAlchemy [20]

Flask-Sqlalchemy permite utilizar modelos para crear, modificar, eliminar o consultar datos de una base de datos como por ejemplo MySQL de manera sencilla con una clase que permite realizar operaciones sin necesidad de escribir las consultas directamente, sino utilizando unas funciones propias que se usan de manera intuitiva para simplificar el uso.

4.4 MySQL [45]

MySQL es una base de datos relacional muy usada para entornos de desarrollo web por su sencillez a la hora de usarlo y tiene muy buen rendimiento por la velocidad de respuesta a la hora de hacer peticiones. Contiene a su vez varias capas de seguridad, con contraseñas encriptadas y mantener un sistema de permisos entre los usuarios con acceso a la base de datos.

A diferencia de otras bases de datos, las bases de datos no relacionales como MongoDB o Cassandra, tienden a usarse en proyectos que guarden grandes cantidades de datos o con una estructura dinámica.

4.5 PHPmyadmin [23]

La herramienta phpMyAdmin maneja la administración de la base de datos con MySQL, obteniendo muchas ventajas. Estas ventajas son:

- Es una herramienta gratuita.
- Tiene una interfaz web sencilla y fácil de usar.
- Contiene todas las funcionalidades de MySQL.

También tiene desventajas a la hora de usarlo. Por ejemplo, su uso para administrar bases de datos está limitado a utilizarse sólo en MySQL.

4.6 Android Studio [30]

Android Studio es una herramienta con muchas opciones y ayudas para el desarrollo de aplicaciones Android con java y Kotlin, incluyendo un conversor de código de un idioma a otro [9]. También ofrece una pestaña en la que incluyen las funciones y los atributos de una clase para poder navegar más fácilmente en el código. También contiene emuladores de dispositivos android, dando la posibilidad de personalizar la memoria RAM, el uso de CPU, y otras características. También es útil su debugger, el cual permite analizar muy bien los fallos que puedan ocurrir en ejecución, con un log bastante extenso y descriptivo.

Una parte bastante importante son las dependencias de los archivos Gradle, en caso de manipular distintos elementos como por ejemplo la herramienta SDK de Mapbox, para incluir las credenciales del usuario con Maven y las dependencias.

4.7 Visual Studio Code [43]

Visual Studio Code es un editor de código mundialmente conocido para desarrollar scripts en varios lenguajes de programación, proporcionando una interfaz gráfica simple. Ofrece un control de Git integrado, para mantener actualizado el repositorio de proyectos. A nivel de código, incluye una finalización de código inteligente y una refactorización de código.

Visual Studio Code posee un soporte técnico sólido gracias a la extensa comunidad que apoya el uso de la herramienta.

4.8 GitHub [58]

Es la plataforma más popular para alojar y gestionar proyectos que necesiten una colaboración activa entre varios miembros. Esta herramienta se utiliza principalmente para mejorar el flujo de trabajo entre los miembros de un proyecto y tener un sistema de control de versiones. Lo más llamativo de GitHub es que da la posibilidad de colaborar y realizar cambios en proyectos compartidos manteniendo una monitorización del desarrollo del código.

4.9 Python [54]

El lenguaje Python es mundialmente conocido debido a la sencilla legibilidad y ámplia documentación oficial, con bastantes librerías que aportan distintas funcionalidades.

Además, Python iba a resultar más sencillo e intuitivo de usar a la hora de realizar código frente a otros lenguajes. Actualmente hay bastantes apli-

caciones que hacen uso de Python para el desarrollo de sus servidores, como Instagram, que usa este lenguaje con el framework Django.

4.10 API de Twitter

La API de Twitter **Tweepy [59]** se usa para extraer tweets reales sobre opiniones de usuarios de la red social y así poder dar a conocer dentro de la aplicación Android una opinión más realista.

La ventaja más notable frente a otras APIs de redes sociales como Facebook o Instagram es la gran cantidad de información que puede ofrecer. También porque Twitter es conocido mundialmente como una plataforma donde los usuarios comparten sus opiniones personales sobre prácticamente todo, por lo que contiene datos esenciales que podrán ser útiles a nuevos turistas que visiten Madrid.

Respecto a la API de Twitter, si se utiliza una versión gratuita, conlleva a tener restricciones de uso y una limitada cantidad de 180 peticiones por 15 minutos.

4.11 API de Google

Se ha aplicado diversas APIs de Google tanto para el procesamiento de datos como para el uso de la aplicación Android.

- **API de Google Translate [60]**: Es una API gratuita con diversas ventajas, como por ejemplo la realización de traduciones masivas, detección de lenguaje inteligente. Prácticamente la API de Google Translate no tiene competencia por su eficacia y precisión a la hora de traducir texto.
- **API de Google Speech-to-Text [31]**: La API de Google Translate presenta varias ventajas, como una alta precisión a la hora de transcribir los audios, y un reconocimiento de voz en streaming, recibiendo resultados de las transcripciones a tiempo real.

Las dos APIs son gratuitas pero con restricciones. Por ejemplo, Speech-to-Text deja únicamente 1 hora de uso por cada mes.

4.12 API de Imgur [35]

La API de Imgur ofrece la posibilidad de utilizar toda las funcionalidades disponibles dentro de la página web pero en un lenguaje de programación. La API contiene unas restrinciones que no afectan al desarrollo de la aplicación Android, pero se debe tener en cuenta. Aproximádamente, la API de

Imgur da la posibilidad de realizar 1,250 peticiones al día, ya sean publicar o consultar fotos. Las ventajas principales de la API es su rápida conexión y manipulación de datos, fácilidad de uso y servicio gratuito.

4.13 Beautiful Soup [52] y Selenium [34]

BeautifulSoup es una herramienta utilizada durante años para realizar un análisis estructural de los documentos HTML de páginas webs. La biblioteca, genera un árbol en base al contenido de la estructura del documento para realizar una extracción de datos con la metodología Web Scrapping.

Selenium es una herramienta utilizada para realizar pruebas sobre aplicaciones webs. Es compatible con la mayoría de navegadores webs para reproducir y grabar ejecuciones automáticas. oftware es de código abierto, por lo que su uso es completamente gratuito.

Aunque Selenium esté creado específicamente para realizar testing, se ha conseguido sacarle provecho para completar el procedimiento de extracción dentro del proceso ETL llevado a cabo.

4.14 XAMPP [55]

XAMPP es una abreviación de las herramientas que la componen. Está compuesto por los programas Linux, Apache, MySQL o MariaDB, PHP y Perl. Se usa para diseñar páginas web u otros proyectos sin necesidad de tener internet. Esto hace que esta herramienta sea muy usada por muchos desarrolladores.

4.15 Advanced Rest Client [44]

Es una herramienta para realizar testing sobre una API con peticiones HTTP. Esta herramientas se puede instalar directamente como una extensión de Google Chrome. Su interfaz es muy intuitiva y fácil de utilizar. En ella te permite de forma gratuita realizar cualquier petición HTTP con todos los métodos viables, POST, GET, PUT, DELETE, etc.

4.16 OkHttp3 [57]

Okhttp3 es un cliente de HTTP que proporciona una librería la cual permite realizar peticiones asíncronas al servidor, sin necesidad de preocuparse por el formato de los mensajes. Simplemente indicando parámetros como la URL, el mensaje, el tipo de mensaje que se envía y en otro hilo se recoge la respuesta del servidor. Es un proyecto de código abierto que se puede encontrar

en github. Existen otras librerías como Volley o Retrofit que hacen uso de okhttp, pero cada una tiene sus pegas a la hora de su desarrollo:

- **Volley [4]** no es tan robusto como OkHttp y no alcanza la misma velocidad ni ancho de red además de estar menos documentada.
- **Retrofit [3]** está construida sobre OkHttp así que el uso de OkHttp permite un mayor control sobre las peticiones que se lanzan al servidor.

4.17 Mapbox [41]

Mapbox es una herramienta de navegación de código abierto la cual cuenta con su propia API y SDK (software development kit) para Android e IOS. Gracias a esta herramienta, cualquier persona puede incluir mapas en sus programas ya sean aplicaciones web, Android o IOS, incluir navegación a un lugar, con personalización de los mapas, pudiendo elegir propiedades como las coordenadas iniciales, el estilo del mapa, los iconos que se mostrarán cuando una persona marque un lugar, cómo iniciar la navegación a un sitio, el icono del usuario, etc.

Mapbox frente a otras aplicaciones como Google Maps tiene una versión gratuita, tiene un mejor desempeño y mejor customización de fondos, pero Google Maps es mucho más confiable y ha estado operativo más tiempo.

Capítulo 5

Arquitectura y modelo de datos

Este capítulo explica la arquitectura de la aplicación y los modelos de datos.

5.1 Arquitectura

Esta sección presenta los patrones de diseño utilizados, manteniendo un esquema organizado y reduciendo considerablemente los errores al diseñar el software.

5.1.1 MVVM [61] [51] [40] [56]

Para el desarrollo de la aplicación Android se aplica el patrón MVVM que recomienda y promueve Android en su documentación [16]. Para ello, el proyecto está estructurado en tres partes bien diferenciadas: El modelo, la vista, y el modelo de la vista.

- La vista muestra la presentación de los datos. Además, la vista es activa, es decir, reaccionando a los cambios que ocurren en el ViewModel.
- En el modelo de la vista (ViewModel) se encuentra la lógica de la presentación y es la encargada de comunicarse con el modelo y "hacer de puente" entre vista y modelo.
- En el modelo se encuentra la lógica de negocio encargada de obtener los datos ya sea a través de una API o una conexión a una base de datos. En el proyecto se identifica como un Repositorio tal y como se especifica en la documentación de Android.

Todo ello se realiza con observadores, de manera que la vista observa al modelo de la vista, el cual observa al repositorio; la clase que contiene los objetos del modelo y que se comunica con el servidor, de manera que si

hay algún cambio en un objeto del modelo, el repositorio indicado enviará esa información a los modelos de la vista que observen el repositorio en ese momento, y cada modelo de la vista a su vez proporcionará la información a su respectiva vista.

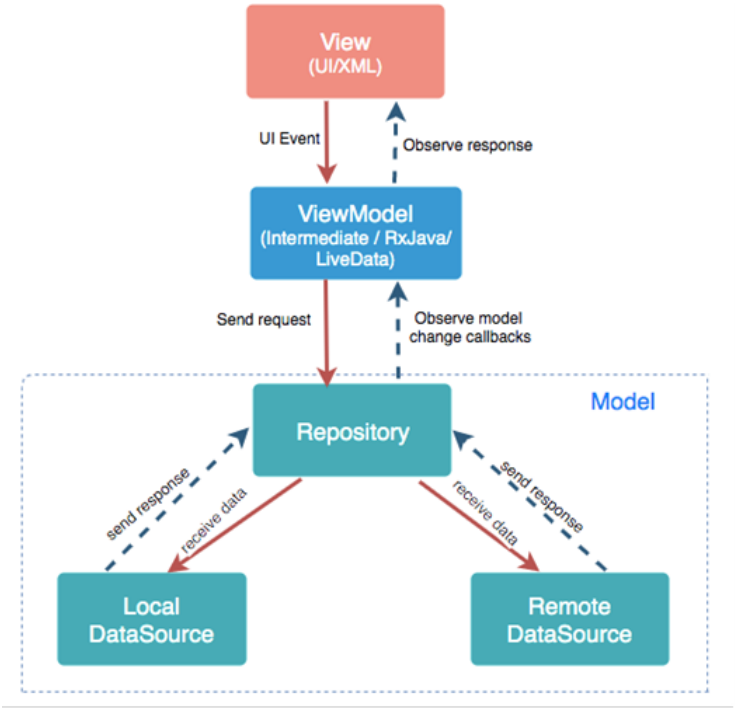


Figura 5.1: Diagrama MVVM

Para ver el patrón de una manera más clara, la figura 5.1 representa un resumen de este patrón simplificado.

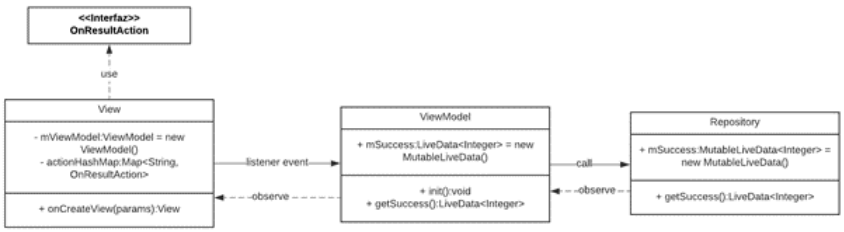


Figura 5.2: Clases del modelo MVVM

La figura 5.2 representa, de una manera más cercana, el código de la estructura que tiene cada una de las clases de este tipo.

En general, este patrón se puede resumir en acción y reacción mediante la separación de elementos de tal manera que un cambio hace que la vista se actualice.

El acoplamiento entre el ViewModel y la vista es mínimo en el patrón MVVM porque el ViewModel no necesita tener ninguna referencia a los elementos de la vista. Simplemente tiene que hacer observables los elementos que tengan una funcionalidad en ella. Sin embargo, en el patrón **MVC**, el controlador, que es el equivalente al ViewModel, sí necesita una referencia a la vista para poder actualizarla. Además, el patrón MVC es más intuitivo ya que sigue el flujo que espera un usuario: se ejecuta una operación que cambia el modelo y posteriormente el controlador actualiza la visualización del modelo. En cambio en el patrón MVVM, la vista es la que observa los cambios en el ViewModel, el cual observa los cambios en el repositorio que es quien actualiza el modelo.

5.1.2 DAO

El patrón DAO se utiliza para separar en una clase las operaciones sobre un modelo que van a interactuar con la API REST, llamadas repositorios.

Gracias a ello, se puede ver las operaciones que se realizan para cada módulo de la aplicación en funciones bien diferenciadas para cada modelo y con un uso bastante claro, con una comunicación asíncrona con el servidor para no congelar la aplicación.

5.1.3 Data Transfer Object

El patrón Data Transfer Object (**DTO**) se utiliza para poder enviar los datos de un modelo entre distintas clases de la aplicación. Para ello, se definen cada clase del modelo como una clase con métodos get y set para cada atributo, de manera que se puedan modificar y leer los datos sin problema.

5.1.4 Singleton

El patrón Singleton se utiliza para tener acceso global a la clase App para realizar gestiones que influyen a la sesión de la aplicación o ajustes en menús u otros elementos de la interfaz de la aplicación.

5.1.5 Adapter

Se usa el patrón adapter para implementar los RecyclerView. Su funcionalidad es construir una vista final a partir de unos datos para que observe el usuario.

5.2 Modelo de datos

5.2.1 Modelo Entidad Relación

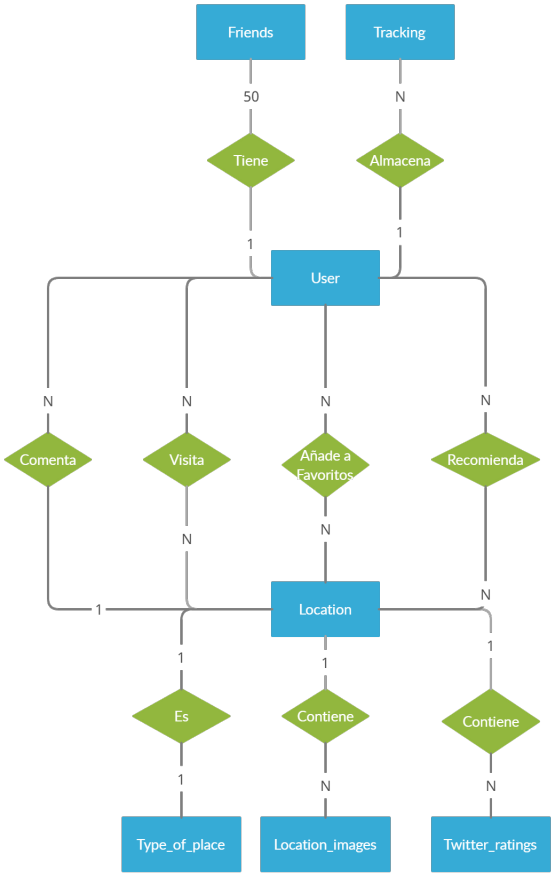


Figura 5.3: Diagrama ER de la Base de datos.

Para realizar el modelo de datos, se lleva a cabo un diagrama Entidad Relación con los principales componentes, el cual posteriormente se transforma al modelo relacional (Véase en la Figura 5.3).

5.2.2 Filas y columnas de la BBDD relacional

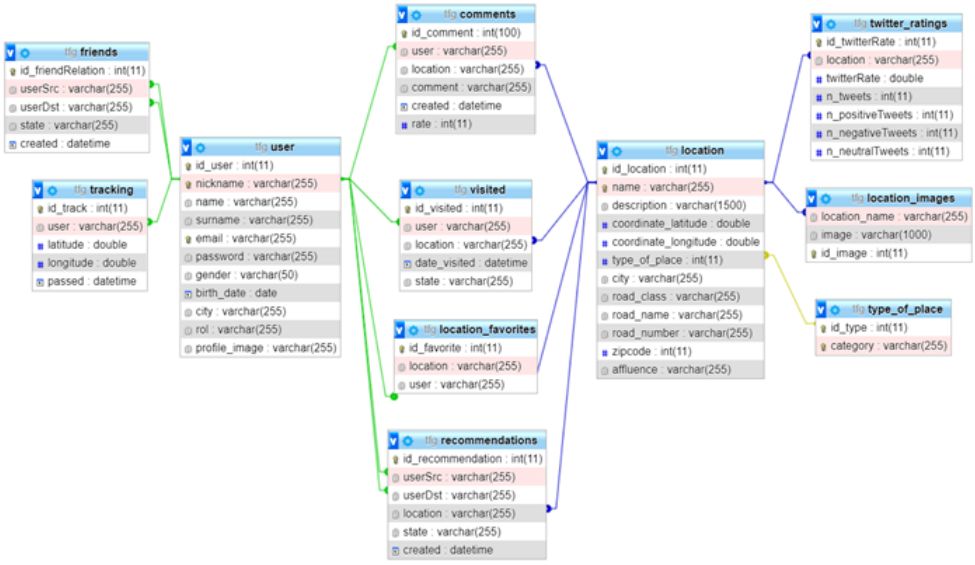


Figura 5.4: Diagrama UML de la Base de datos.

En primer lugar, se muestra el diagrama UML con la estructura de la base de datos (ver Figura 5.4). Como se puede observar, la base de datos está formada por un total de 11 tablas.

A continuación se explica el propósito de cada tabla en sus respectivos módulos, el significado de sus campos y las relaciones con otras tablas.

Módulo Usuario y Amigos

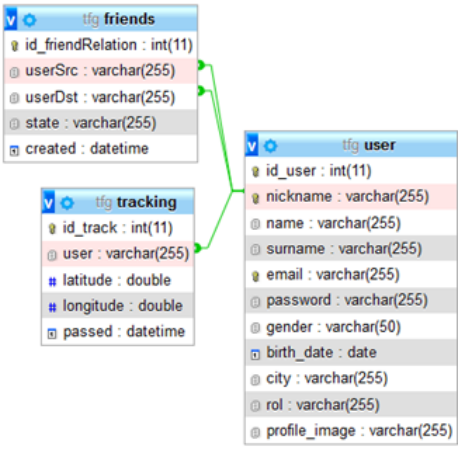


Figura 5.5: Diagrama UML del módulo de Amigos.

En este módulo existen tres tablas (ver Figura 5.5). En la tabla User se almacenan todos los usuarios registrados dentro de la aplicación. Todos tienen una identificación única y un rol que desempeña el papel de usuario cliente o usuario administrador.

User	
id_user	Se trata del identificador del usuario y es la clave primaria de toda la tabla. Cada usuario tiene un id único y se genera automáticamente cada vez que se crea un usuario nuevo.
nickname	Es el apodo del usuario que se muestra públicamente en la aplicación. Este campo es de tipo string y para no haber ninguna confusión, este campo es único por lo que dos usuarios no pueden tener el mismo apodo.
name	Es el nombre real del usuario. Esta información personal, junto con el/los apellido/s y el correo electrónico, se podrán ver en la aplicación una vez registrados, en la vista de Perfil del Usuario.
surname	Se trata de los apellidos reales del usuario. Este campo es de tipo String.

Tabla 5.1: Tabla User parte 1

User	
email	Es el email del usuario. En la aplicación, un correo electrónico solo puede estar registrado una vez, por lo que este campo es único. De esta forma, dos usuarios no pueden compartir un mismo correo electrónico.
gender	El género del usuario, que puede ser M (Mujer) o H (Hombre).
birth_date	La fecha de nacimiento del usuario en formato Date.
city	La ciudad donde vive el usuario. Por defecto Madrid es la ciudad predeterminada.
rol	El rol es la función que desempeña el usuario dentro de la aplicación. Puede ser user o admin. En caso de ser user, esta cuenta tendrá acceso a todas las funcionalidades disponibles para un usuario normal. Si es admin, tendrá permisos para realizar funciones de administrador.
profile_image	Contiene una foto del perfil del usuario. El campo contiene una URL generado por la API de Imgur para poder almacenar de forma óptima todas las imágenes de los usuarios. Como se trata de URLs, el campo es de tipo String.

Tabla 5.2: Tabla User parte 2

En la tabla Friends, se almacenan las relaciones amistosas que tienen los usuarios de la aplicación. Para no saturar demasiado la base de datos, se restringe un máximo de 50 amigos por usuario. También se almacenan aquí las solicitudes de amistad y la fecha de creación.

En la tabla tracking se almacenan los puntos de seguimiento de todos los usuarios que han dado permisos para recoger la ubicación del móvil Android. Los datos no son permanentes, ya que dichos puntos se borran periódicamente cada 1 hora.

Esta tabla sirve para realizar un seguimiento del camino recorrido por el usuario y saber qué lugares ha estado visitando.

Módulo de Lugares

La tabla de Lugares es la tabla principal de todo el módulo de Lugares (ver Figura 5.6). Es donde se almacenan todos los lugares recopilados por la página oficial de datos abiertos del Ayuntamiento de Madrid.

La tabla Location_images se trata de la tabla que almacena todas las imágenes que tiene un lugar. Para no sobrecargar la base de datos con imágenes, se han almacenado las imágenes en urls que accedan a dichas imágenes.

Friends	
id_friendRelation	Es el identificador de la relación entre dos usuarios.
userSrc	userSrc es el apodo del usuario que ha solicitado una amistad con userDst. Contiene una clave foránea con la tabla "user".
userDst	Es el apodo del usuario al que han solicitado una amistad. Tal como está definido userSrc, también contiene una clave foránea de la tabla "user".
state	Es el campo que representa la situación de la relación entre dos usuarios. Puede ser de dos tipos: P → Pendiente: Significa que userSrc ha enviado una solicitud de amistad y userDst aún no ha aceptado o rechazado la solicitud. A → Accepted: Significa que userDst ha aceptado la solicitud de amistad y los usuario ya son amigos. En caso de rechazar la solicitud de amistad, la relación se elimina completamente. El campo está definido como un varchar.
created	Created es un campo de tipo DateTime generado automáticamente cuando se envía una solicitud de amistad nueva entre los dos usuarios. En caso de aceptar la solicitud, el campo created se actualiza a la fecha cuando se aceptó la petición.

Tabla 5.3: Tabla Friends

La mayoría de urls se obtuvieron por los datasets recopilados y una minoría se generaron utilizando Selenium para recoger imágenes de los lugares con Google Fotos.

La tabla Type_of_place es donde se almacenan los tipos de lugares que existen dentro de la aplicación Android.

La tabla Comments se almacena todas las reseñas y opiniones que expresan los usuarios registrados dentro de la aplicación. También contiene un campo que permite al usuario poder calificar el lugar del 1 al 5.

La tabla Twitter_ratings contiene otras reseñas no elaboradas por los usuarios de la aplicación, si no por usuarios que han escrito tweets dentro de Twitter acerca de dichos lugares. Sin embargo, no se han podido extraer tweets de todos los lugares. Esta tabla se actualiza semanalmente de forma manual con un script en Python para actualizar las valoraciones e intentar

Tracking	
id_track	Es el identificador único de cada punto de seguimiento de un usuario. Es la clave primaria de la tabla.
user	Se refiere al apodo del usuario que está monitorizando su seguimiento. Contiene una clave foránea con la tabla de "User".
latitude	Se trata de la localización del lugar, con dirección Norte o Sur desde el ecuador y se expresa en medidas angulares que varían desde los 0^{o} del Ecuador hasta los 90^{o}N ($+90^{o}$) del polo Norte o los 90^{o}S (-90^{o}) del polo Sur.
longitude	Se trata de la localización del lugar, con dirección Este u Oeste desde el meridiano de referencia 0^{o}, o meridiano de Greenwich, expresándose en medidas angulares comprendidas desde los 0^{o} hasta 180^{o}E ($+180^{o}$) y 180^{o}W (-180^{o}).
passed	Se almacena en este campo el momento exacto que ha pasado el usuario por punto definido. Es de tipo DateTime.

Tabla 5.4: Tabla Tracking

obtener nuevos tweets acerca de los lugares.

En la tabla Visited se almacenan todos los lugares que se han visitado o están pendientes por visitar por los usuarios registrados en la aplicación. También, cuando un usuario recomienda un lugar a otro usuario y éste acepta la recomendación, se añade automáticamente el lugar como pendiente por visitar. De esta manera, amplificamos considerablemente la interacción social entre usuarios.

La tabla Location_favorites es una tabla intermedia que almacena los lugares favoritos de todos los usuarios registrados en la aplicación. Dentro de la aplicación hay una sección donde se pueda ver la lista de los lugares favoritos, y también poder seleccionar un lugar como favorito.

Por último, la tabla Recommendations contiene todas las recomendaciones que se han enviado entre los usuarios de la aplicación. Las recomendaciones se refieren a lugares que están registrados en la base de datos. En caso de que un usuario haya visitado un lugar y quisiera recomendarle a alguien, podrá recomendar a cualquier amigo que tenga dentro de la aplicación.

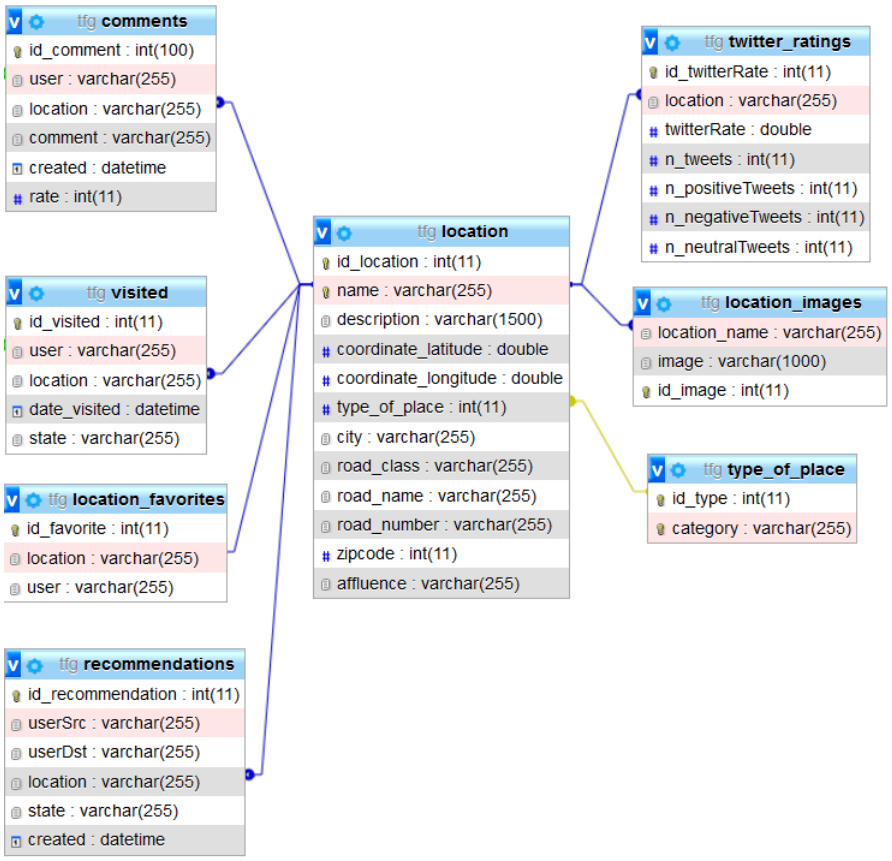


Figura 5.6: Diagrama UML del módulo de Lugares.

Location	
id_location	Es el identificador del lugar y es la clave primaria de toda la tabla.
name	Es el campo del nombre oficial definido por el Ayuntamiento de Madrid. El campo es de tipo varchar y es el nombre que se mostrará dentro de la aplicación Android.
description	Es una descripción no escrita por los miembros del proyecto, sino recopilado por los datasets recogidos por el Ayuntamiento de Madrid. La descripción suele ser una información detallada del lugar.
coordinate_latitude	Como ya he explicado es un parámetro geográfico necesario para calcular el punto exacto donde se encuentra el lugar, o una estimación en caso de ser un lugar amplio como un parque o jardín.
coordinate_longitude	Es el otro parámetro geográfico para calcular la localización que se encuentra el lugar.
type_of_place	Es el campo que contiene los identificadores de los tipos de lugares definidos. Es la clave foránea de la tabla Type_of_place que hablaremos más adelante. Como se trata de identificadores, el campo es de tipo integer.
city	Es el campo donde especifican en qué ciudad se encuentra el lugar. Como se tratan de lugares únicamente en Madrid, este campo se completa automáticamente como Madrid.
road_class	Los siguientes 4 campos definen la dirección completa del lugar. En concreto, road_class se trata de los tipos de vías públicas, pueden ser Calle, Plaza, Avenida, etc.
road_name	Es el nombre de la vía.
road_number	El número de la vía
zipcode	El código postal que tiene la vía.
affluence	Es un campo referido a la afluencia actual del lugar. Este campo no se tiene en cuenta ya que el campo se añadió para un funcionamiento de trabajo futuro.

Tabla 5.5: Tabla Location

Location_images	
id_image	Es el identificador único de la imagen en concreto. Es la clave primaria de la tabla.
location_name	El campo contiene el nombre del lugar que se relaciona con una imagen en concreto. Es la clave foránea para poder enlazar las imágenes con sus respectivos lugares.
image	Es el campo donde se almacena las urls que dan acceso a las imágenes en internet.

Tabla 5.6: Tabla Location_images

Type_of_place	
id_type	Este campo contiene los identificadores de los tipos de lugares. Se trata de la clave primaria de la tabla.
category	Aquí se almacena el tipo de lugar en formato varchar. Son los mismos tipos de lugares definidos dentro de la aplicación Android.

Tabla 5.7: Tabla Type_of_place

Comments	
id_comment	Este campo contiene los identificadores de los comentarios de los lugares. También se trata de la clave primaria de la tabla.
user	El campo user contiene el apodo del usuario que ha escrito el comentario. Se trata de la clave foránea para poder relacionarlo con la tabla "user".
location	El campo location contiene el nombre del lugar que ha comentado el usuario.Se trata de la clave foránea para poder relacionarlo con la tabla "location".
comment	Es el campo que contiene la opinión o reseña que ha escrito el usuario. El campo es de tipo varchar.
created	El campo created contiene la fecha exacta en que se publicó el comentario. El campo es de tipo Datetime por lo que se almacena la fecha con la hora.
rate	Es una calificación opcional que puede especificar el usuario dentro de la aplicación. La valoración es del 1 al 5, siendo 1 como un lugar no recomendado, y 5 como perfecto para visitar.

Tabla 5.8: Tabla Comments

Twitter_ratings	
id_twitterRate	El campo contiene los identificadores de las valoraciones recogidas por los tweets de usuarios de Twitter, acerca de los lugares registrados en la aplicación.
location	Se trata del nombre del lugar relacionado a las valoraciones generados con los tweets recogidos. Es la clave foránea que se relaciona con la tabla de "location".
twitterRate	Contiene la valoración final que es nada más que la media de tweets positivos, neutrales y negativos.
n_tweets	El campo contiene el número específico de tweets que se han podido recoger por la API de Twitter. Hay algunos lugares que no se han podido extraer ningún tweet, ya que no todos los lugares son conocidos.
n_postiveTweets	Es el campo en el que se almacenan la cantidad de tweets positivos acerca del lugar.
n_negativeTweets	Es el campo en el que se almacenan la cantidad de tweets negativos acerca del lugar.
n_neutral_Tweets	Es el campo en el que se almacenan la cantidad de tweets neutrales acerca del lugar.

Tabla 5.9: Tabla Twitter_ratings

Visited	
id_visited	Se trata del campo de identificación del lugar visitado o pendiente por visitar. Es la clave primaria de toda la tabla.
user	El campo user contiene el apodo del usuario que ha visitado o tiene el lugar como pendiente por visitar. Es una clave foránea necesaria para relacionar con la tabla "user".
location	Es el campo que contiene el nombre del lugar que se ha visitado o está pendiente por visitar. Contiene la clave foránea para relacionar con la tabla "location".
date_visited	Es el campo que contiene la fecha que se ha visitado o se ha añadido el lugar como pendiente a visitar. El campo es de tipo Datetime, para saber la fecha y la hora exacta que se ha creado.
state	Es el campo que representa la situación de la relación entre el usuario y el lugar. Puede ser de dos tipos: P → Pendiente: Significa que el lugar está pendiente a visitar por el usuario. En el momento que se visite, pasará a estado V. V → Visited: Significa que el usuario ha visitado el lugar.

Tabla 5.10: Tabla Visited

Location_favorites	
id_favorite	Es el campo identificador que es único para cada lugar favorito. Es la clave primaria de toda la tabla.
location	Este campo contiene el nombre del lugar que ha seleccionado el usuario como favorito. Es la clave foránea que se relaciona con la tabla de "location".
user	El campo user contiene el apodo del usuario que ha seleccionado un lugar como favorito. Es la clave foránea que se relaciona con la tabla de üser".

Tabla 5.11: Tabla Location_favorites

Recommendations	
id_recommendation	Se trata del identificador único de la recomendación enviada por los usuarios. Es la clave primaria de toda la tabla.
userSrc	El campo userSrc es el usuario que ha enviado la recomendación a otro usuario, almacenado en userDst. Contiene la clave foránea que relaciona con la tabla "user" y sacar información acerca del usuario.
userDst	El campo userDst es el usuario al que le han enviado la recomendación. Contiene la clave foránea que se relaciona con la tabla "user".
location	Este campo contiene el nombre del lugar que han recomendado por los usuarios. El campo contiene la clave foránea que se relaciona con la tabla "location".
state	Es el campo que representa la situación de recomendación del lugar. Puede ser de dos tipos: P → Pendiente: Significa que userSrc ha enviado una recomendación y userDst aún no lo ha aceptado o rechazado. A → Accepted: Significa que userDst ha aceptado la recomendación. Tras aceptarlo, se añadirá el lugar en su lista de lugares pendientes por visitar. En caso de rechazar la recomendación, se elimina completamente.
created	El campo created contiene la fecha que se ha enviado o aceptado la recomendación. Es de tipo Datetime por lo que se puede saber la fecha y la hora exacta.

Tabla 5.12: Tabla Recommendations

Capítulo 6

Implementación

6.1 Fase de diseño

6.1.1 Elementos Android

Este apartado explica los elementos que se han utilizado dentro de la aplicación.

6.1.1.1 Fragments [14]

Los fragmentos de Android son elementos visuales que representan la parte de una interfaz y siempre deben estar en una actividad. El proyecto está formado por una sola Activity y múltiples Fragment, ya sea a modo de componentes visuales o pantallas completas para la vista de la aplicación en sustitución de las activities.

6.1.1.2 RecyclerView (listas) [12]

RecyclerView es una vista que crea de manera dinámica, una lista de elementos en la interfaz visual. Para el uso de este elemento es necesario implementar:

- ViewHolder, esta clase constituye el elemento visual básico que se utiliza en la lista. Es necesario diseñar layout e implementar su posible lógica.
- Adapter, esta clase es la encargada de generar los ViewHolder correspondientes a un modelo de datos dado(una lista de elementos). Es la clase principal con la se comunica el RecyclerView

Se usa el RecyclerView para generar listas dinámicamente como listas de lugares o listas de usuarios.

6.1.1.3 TabLayout [15]

Es una vista que se usa junto con un ViewPager para ofrecer en la vista una serie de pestañas con las que cambiar de vista(fragmentos).

Para la implementación de este elemento es necesario crear una clase auxiliar que herede de FragmentPagerAdapter. Esta clase auxiliar(BaseTabAdapter) se puede encontrar en el paquete components del proyecto y se reutiliza en las múltiples vistas que hacen uso de una vista con pestañas.

6.1.1.4 LiveData [27]

LiveData es una clase que encapsula objetos de tal manera que se puedan observar. Esta clase observable está optimizada para los ciclos de vida de Android lo que facilita el manejo de los datos en los cambios de estado del objeto LiveData.

```
mViewModel.getSuccess().observe(getViewLifecycleOwner(), new Observer<Integer>() {
    @Override
    public void onChanged(Integer integer) {
        if(actionHashMap.containsKey(integer))
            actionHashMap.get(integer).execute();
    }
});
```

Figura 6.1: Método observe.

Para la observación de los datos se usa el método observe (ver Figura 6.1 donde se muestra en la vista para actuar según el resultado de las operaciones que se producen en el modelo.) para reaccionar a los cambios que sufra el LiveData. Este componente es esencial para aplicar el patrón MVVM ya que la vista observa los cambios que se producen en el ViewModel.

6.1.1.5 Navigation [17]

Es un componente de Android Jetpack que permite crear una navegación con una interfaz sencilla ya sea mediante comandos o la creación de grafos. Para poder implementarlo es necesario usar y configurar:

- Gráfico de navegación, es un recurso XML que contiene toda la información relacionada con la navegación, es decir, los distintos fragmentos de la interfaz por los que va a navegar el usuario.
- NavHost, es un contenedor vacío que muestra los destinos de tu gráfico de navegación.
- NavController, para administrar la navegación de la app dentro de un NavHost.

```
<com.facebook.shimmer.ShimmerFrameLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/placeList_ShimmerLayout">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="4dp">

        <include
            layout="@layout/place_item_shimmer"/>

        <include
            layout="@layout/place_item_shimmer"/>

        <include
            layout="@layout/place_item_shimmer"/>

    </LinearLayout>

</com.facebook.shimmer.ShimmerFrameLayout>
```

Figura 6.2: Elemento ShimmerFrameLayout.

6.1.1.6 Glide [37]

Glide es una librería que permite buscar, decodificar o mostrar imágenes, GIFs o videos. Se usa por la facilidad que ofrece para la carga de imágenes de una URL y su manejo de la caché de la aplicación.

6.1.1.7 Shimmer [33]

Una librería para añadir gráficos de carga para los contenidos de la aplicación.

Para usarlo es necesario crear una vista ViewHolder alternativa para los RecyclerViews de tal manera que aparece un fondo grisáceo a modo de transición mientras se realiza la carga de la lista. Este ViewHolder alternativo se usa en un elemento XML para las vistas ShimmerFrameLayout (ver Figura 6.2) cuyo efecto de transición (ver Figura 6.3) se puede manejar desde código.

6.1.1.8 Action Bar [5]

Figura 6.4: Menú superior.

Es el menú superior de nuestra aplicación (ver Figura 6.4), contiene funcionalidades importantes como la búsqueda por texto o por voz. Para saber si una funcionalidad se debe incluir en él, se debe utilizar un esquema **FIT**: Si se va a utlizar con **F**recuencia, si es **I**mportante para los usuarios y si es **T**ípica, es decir, que los usuarios esperan encontrarla en ese lugar.

Figura 6.3: Efecto de shimmer en la lista de lugares de la aplicación.

Figura 6.5: Menú inferior.

6.1.1.9 Bottom Navigation Bar [11]

El menú inferior de nuestra aplicación (ver Figura 6.5), contiene tres botones distintos para poder moverse de unos fragmentos a otros con facilidad, siendo los más importantes para el usuario para que pueda hacerlo de manera fácil. Además resalta la funcionalidad sobre la cual está el usuario para que le sea más fácil distinguirla del resto.

6.1.1.10 Refresh [10]

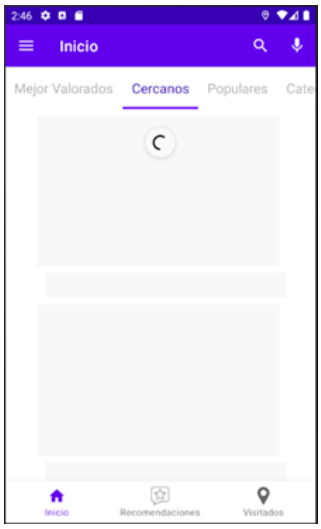


Figura 6.6: Efecto Refresh.

El elemento Refresh (ver Figura 6.6) sirve para actualizar las listas de elementos haciendo swipe hacia arriba. También es llamado popularmente Pull-to-refresh y se utiliza para las listas de lugares.

6.1.1.11 Ripple [22]

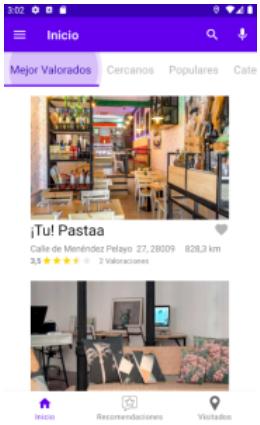


Figura 6.7: Efecto Ripple.

El efecto Ripple (ver Figura 6.7) sirve para que el usuario pueda percibir que ha realizado una pulsación en un determinado lugar, para ello se muestra un efecto que simboliza una onda en el agua que comienza en el lugar de

la pulsación. Este efecto se usa en diversos botones como en los ítems del menú, los comentarios, etc.

6.1.1.12 Gestos

En una aplicación Android se utilizan diversos gestos sobre la pantalla según las necesidades. En este caso, se utilizan los siguientes:

- **Toque**: Sirve para seleccionar un elemento y que cumpla la función que le ha sido asignada por defecto. Se realiza haciendo una breve pulsación sobre el elemento indicado.
- **Deslizar**: Sirve para moverse entre los diversos elementos de una lista si se realiza verticalmente y para mostrar el menú del usuario si se realiza horizontalmente. Se realiza manteniendo pulsada la pantalla a la vez que se arrastra el dedo sin levantarlo en la dirección querida.
- **Pellizco abierto**: Sirve para ampliar el mapa una vez abierto, para mostrar un área más específica de la región. Se realiza pulsando los dos dedos sobre la pantalla y arrastrandolos en dirección contraria uno del otro.
- **Pellizco cerrado**: Sirve para reducir la vista del mapa mostrando un área más amplia de la región. Se realiza pulsando los dos dedos sobre la pantalla y arrastrando uno hacia el otro juntándolos.

6.1.2 Base de datos

Para el desarrollo de la aplicación Android, es necesario unas grandes cantidades de datos sobre turismo y localidades dentro de Madrid que sean atractivas para los usuarios primarios. Para ello, se construye una base de dato relacional con MySQL para poder almacenar todos los datos extraídos en una estructura sólida y así aumentar la robustez y la consistencia.

Los datos extraídos son completamente fijas, sin necesidad de modificar la estructura de la base de datos, por lo que no sería conveniente unas bases de datos no relacionales. No obstante, fomentamos el crecimiento del contenido de la base de datos, permitiendo tanto a los administradores como a los propios usuarios de la aplicación Android, poder insertar nuevos lugares que se vayan encontrando durante su viaje a Madrid y que no estén registrados ya en la aplicación.

6.2 Clases principales

La aplicación MadridPlaces se compone de varios paquetes donde se recoge tanto la lógica de las vistas como la de negocio. La aplicacion contiene gran

cantidad de clases y paquetes, los más importantes para la compresión de la arquitectura de la aplicación son los siguientes(ver Figura 6.8):

- **Paquete ui.** Donde se puede encontrar todos los paquetes con las clases necesarias para la creación de las vistas

- **Paquete repositories.** En el se encuentran las clases que se conectan con la API y envían los datos a la vista

- **Paquete components.** Paquete donde se encuentan algunas clases genéricas de la aplicación

- **Paquete models.** En este paquete se pueden ver la implementación de las clases DTO que se usan en la aplicación

- **Paquete utilities.** Paquete donde encontramos clases usadas para la definición de estados, gestión de permisos o información asociada a la aplicación como los lenguajes de la aplicación.

En los siguientes puntos se explican con mayor detalle los distintos paquetes y clases que componen la aplicación.

6.2.1 Main Activity

Actividad principal de la aplicación donde se configuran los menús, fragmentos, navegación y permisos de la aplicación.

6.2.2 App

Clase Singleton que se utiliza para gestionar operaciones de la sesión del usuario, comprobar la conexión a internet, realizar pequeñas configuraciones en la vista o pedir permisos al usuario.

6.2.3 SessionManager

Clase con los objetos y funciones necesarias para la gestión de los datos de sesión del usuario. En esta clase se guardan las credenciales del usuario [13] y el idioma de la aplicación .

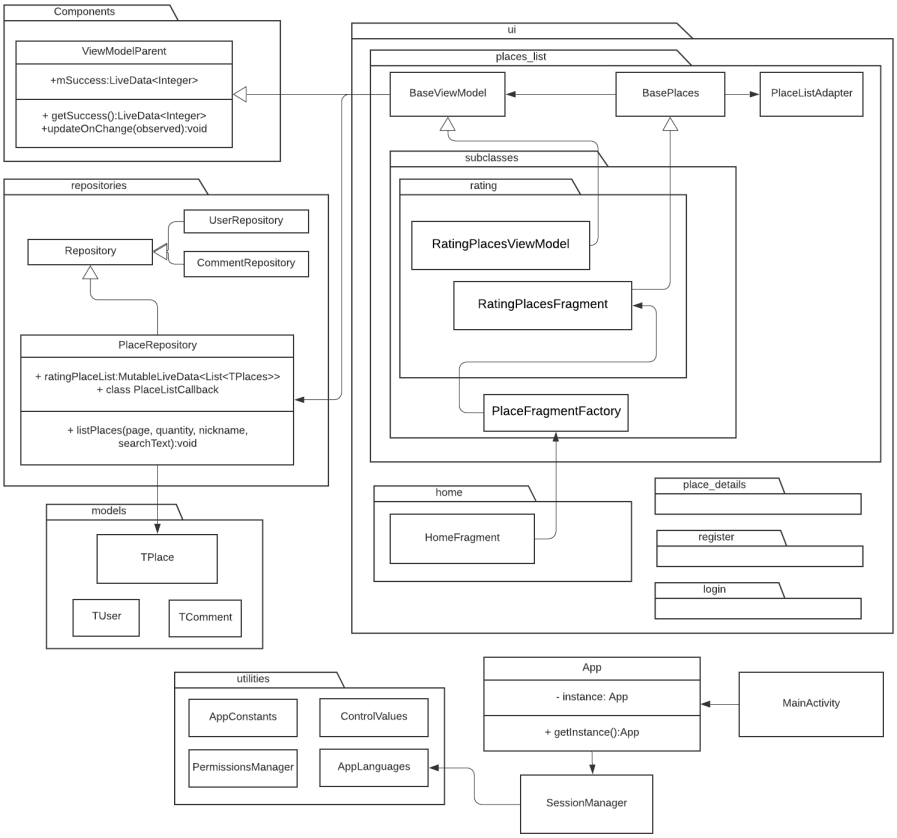


Figura 6.8: Diagrama de clases principales del proyecto.

6.2.4 Paquete components

```
public abstract class ViewModelParent extends ViewModel {

    protected LiveData<Integer> mSuccess = new MutableLiveData<>();

    public abstract void init();

    public LiveData<Integer> getSuccess() { return mSuccess; }

    /**
     * Devuelve un liveData que se actualiza cada vez que el valor de observed cambia
     */
    public <T> LiveData<T> updateOnChange(MutableLiveData<T> observed){
        return Transformations.switchMap(
                observed,
                newValue -> getLiveDataFromNewValue(newValue)
        );
    }

    private <T> LiveData<T> getLiveDataFromNewValue(T value) {
        return new MutableLiveData<>(value);
    }
```

Figura 6.9: Clase ViewModelParent.

En este paquete se encuentran unos elementos que se reutilizan en el proyecto como:

- BaseTabAdapter [8]. Un adapter genérico para los TabLayout de la aplicación.
- LogoutObserver. Una interfaz para implementar una acción cuando se produzca un login en los distintos fragmentos
- ViewModelParent (ver Figura 6.9). Clase abstracta usada en todos los ViewModel del proyecto. Contiene la lógica necesaria para reaccionar a los cambios de LiveDara que produce el repositorio.

6.2.5 Paquete networking

En este paquete se encuentra la clase **SimpleRequest**. En esta clase se incluyen las funciones necesarias para conectarse a la API, pudiendo saber si el servidor es accesible y también crear peticiones pasando atributos como la URL y los parámetros en formato JSON. También existe una función para crear las llamadas y otra para recoger la respuesta.

Esta clase utiliza OkHttp que como ya se ha explicado anteriormente, es un cliente para el manejo de peticiones Http. Esta clase abstrae el funcionamiento básico de OkHttp que se utiliza en la aplicación, creando una petición a través de los datos que se van a pasar en peticiones POST o GET, dependiendo del endpoint de la API.

La clase **SimpleRequest** se centra en 2 métodos:

- buildRequest(String postBodyString, String method, String route) (ver Figura 6.10). Utilizada para crear una petición con los datos que se le proporcionan, una Cuerpo para la petición en caso de que el método sea POST, el método con el que se corresponde la petición y la ruta del servidor.
- createCall(Request request) (ver Figura 6.11). Necesaria para preparar y crear una llamada con los datos de una petición ya construida con **buildRequest(...)**.

6.2.6 Paquete repositories

En este paquete se encuentra una serie de clases helpers en un paquete con el mismo nombre. Estas clases se encargan de construir la información JSON que necesita lanzar la aplicación a la API según el endpoint usado. Hay un helper por cada repositorio y cada uno hereda de una clase padre **Repository** que contiene una base con los atributos que deben tener todos los repositorios para el control de los resultados de las peticiones.

```java
public class SimpleRequest {

    private static final String DIR_PROTOCOL = "http";
    private static final String IP_ADDRESS = "10.0.2.2";
    private static final int PORT = 5000;
    private static final int TTL_SECONDS = 40;
    private static final int TTL_MSECONDS = 3000;
    private static final String SERVER_URL = DIR_PROTOCOL + "://" + IP_ADDRESS + ":" + PORT;
    private static final String DIRECTORY = "_templateImages";
    private static final String IMAGE_DIRECTORY = SERVER_URL + "/" + DIRECTORY;

    private volatile String response;

    public SimpleRequest(){
        response = null;
    }

    public Call createCall(Request request){
        OkHttpClient client = new OkHttpClient.Builder()
                .retryOnConnectionFailure(true)
                .connectTimeout(TTL_SECONDS, TimeUnit.SECONDS)
                .readTimeout(TTL_SECONDS, TimeUnit.SECONDS)
                .writeTimeout(TTL_SECONDS, TimeUnit.SECONDS)
                .build();
        Call call = client.newCall(request);

        return  call;
    }
```

Figura 6.10: Clase SimpleRequest.

Además del anterior paquete, se encontran las clases repositorio encargadas de lanzar las peticiones Http al servidor usando la abstracción que ofrece SimpleRequest.

Estos **MutableLiveData** encapsulan la información que necesitan los Viewmodel y son observados por estos de tal manera que reaccionan cuando se produzca un cambio en los datos.

```
public Request buildRequest(String postBodyString, String method, String route) {

    MediaType mediaType = MediaType.parse("application/json; charset=utf-8");
    RequestBody body = RequestBody.create(postBodyString, mediaType);

    Request request = null;

    if(method.equals(AppConstants.METHOD_GET)){
        request = new Request.Builder()
                .get()
                .url(SERVER_URL + route)//Ej http://10.0.0.2:5000/login
                .header( name: "Accept",  value: "application/json")
                .header( name: "Content-Type",  value: "application/json")
                .header( name: "Connection",  value: "close")
                .build();
    }else{
        request = new Request.Builder()
                .method(method, body)
                .url(SERVER_URL + route)//Ej http://10.0.0.2:5000/login
                .header( name: "Accept",  value: "application/json")
                .header( name: "Content-Type",  value: "application/json")
                .header( name: "Connection",  value: "close")
                .build();
    }

    return request;
}
```

Figura 6.11: Método buildrequest.

```
public void registerUser(TUser user) {

    String postBodyString = user.jsonToString();
    SimpleRequest simpleRequest = new SimpleRequest();
    Request request = simpleRequest.buildRequest(
            postBodyString,
            AppConstants.METHOD_POST,  route: "/registration/"
    );
    Call call = simpleRequest.createCall(request);

    call.enqueue(new Callback() {
        @Override
        public void onFailure(Call call, IOException e) {
            e.printStackTrace();
            mSuccess.postValue(ControlValues.REGISTER_USER_FAIL);
            call.cancel();
        }
        @Override
        public void onResponse(Call call, final Response response) throws IOException {
            if (!response.isSuccessful()) {
                throw new IOException("Unexpected code " + response);
            }
            mSuccess.postValue(ControlValues.REGISTER_USER_OK);
        }
    });
}
```

Figura 6.12: Método para registro de un usuario en el repositorio de Usuarios.

La estructura básica para realizar una llamada al servidor desde el método de un repositorio se divide en [38] [56]:

1. Construir el body de la petición si el método fuese POST. Para ello, es necesario transformar los datos necesarios en un formato JSON que pueda interpretar la API

2. Construir la clase SimpleRequest con el body, el método de la aplicación y el endpoint de la operación.

3. Crear una llamada personalizada con la que actualizar los datos que necesitará el ViewModel

Un ejemplo de esto se puede ver en la Figura 6.12.

6.2.6.1 UserRepository

```
public class UserRepository extends Repository{

    private MutableLiveData<TUser> mUser = new MutableLiveData<>();
    private MutableLiveData<List<TUser>> mListUsers = new MutableLiveData<>();
    private MutableLiveData<Pair<Integer,Integer>> mCountProfileCommentsAndHistory = new MutableLiveData<>();

    public void registerUser(TUser user) {...}

    public void loginUser(String nickname, String password) {...}

    public void deleteUser(String nickname){...}

    public void modifyUser(TUser u) {...}

    public void listUsers(int page, int quantum, String searchText, int sortType) {...}

    public void getCommentsAndHistoryCount(String nickname) {...}
```

Figura 6.13: Clase UserRepository.

Este paquete (ver Figura 6.13) es el encargado de realizar las peticiones relacionadas con el usuario.

Los datos que deberán observar los ViewModel de la aplicación son:

- Un usuario, el logueado en la aplicación.

- Una lista de usuarios, para el administrador.

- El número de comentarios y lugares visitados por el usuario para su perfil.

6.2.6.2 UserFriendRepository

```
public class UserFriendRepository extends Repository{

    private MutableLiveData<List<TRequestFriend>> mFriendRequestList = new MutableLiveData<>();
    private MutableLiveData<List<TRequestFriend>> mFriendList = new MutableLiveData<>();

    class FriendListCallBack implements Callback {...}

    public void deleteFriend(String userToDelete, String currentUser) {...}

    public void friendList(String username) {...}

    public void declineFriendRequest(String userOrigin, String userDest) {...}

    public void acceptFriendRequest(String userOrigin, String userDest) {...}

    public void friendRequestList(String username) {...}

    public void sendFriendRequest(String username){...}
```

Figura 6.14: Clase UserFriendRepository.

Este repositorio (ver Figura 6.14) es el encargado de realizar las peticiones relacionadas con las relaciones sociales del usuario con otras personas registradas en la aplicación. Los datos que deben observar los ViewModel de la aplicación son:

- La lista de amigos del usuario que ha iniciado sesión.
- La lista de peticiones de amistad que el usuario ha recibido.

6.2.6.3 RecommendationRepository

```
public class RecommendationRepository extends Repository{

    private MutableLiveData<List<TRecommendation>> mRecommendationsList = new MutableLiveData<>();
    private MutableLiveData<List<TRecommendation>> mPendingRecommendationsList = new MutableLiveData<>();

    class RecommendationsListCallBack implements Callback {...}

    public void listRecom(int page, int quantity, String nickname) {...}

    public void acceptPendingRecom(String placeName, String userOrigin, String userDest){...}

    public void denyPendingRecom(String placeName, String userOrigin, String userDest){...}

    public void listPendingRecom(int page, int quantity, String nickname) {...}

    public void sendRecommendation(String userOrigin, String userDest, String place) {...}

    public MutableLiveData<List<TRecommendation>> getmRecommendationsList() {...}

    public MutableLiveData<List<TRecommendation>> getmPendingRecommendationsList() {...}
}
```

Figura 6.15: Clase RecommendationRepository.

En este repositorio (ver Figura 6.15) están definidos los métodos necesarios para el manejo de las recomendaciones de lugares entre usuarios.

La información a la que los ViewModel reaccionan son:

- La lista de las recomendaciones de lugares que ha realizado el usuario en uso de la aplicación.
- La lista de recomendaciones enviadas por amigos a las que el usuario logueado aún no ha contestado.

6.2.6.4 CommentRepository

```
public class CommentRepository extends Repository {

    private MutableLiveData<List<TComment>> mCommentList = new MutableLiveData<>();

    public void listComments(String placeName, int page, int quant) {...}

    public void appendComments(String placeName, int page, int quant) {...}

    public void newComment (String userName, String content, String placeName, float rate) {...}

    public void deleteComment(TComment comment, int position) {...}
```

Figura 6.16: Clase CommentRepository.

En este repositorio (ver Figura 6.16) se encuentran los métodos relacionados con los comentarios que los usuarios escriben sobre los diferentes lugares de la aplicación. En este repositorio solo es necesario una lista de comentarios que usa el ViewModel relacionado con la pantalla de los detalles de un lugar.

6.2.6.5 PlaceRepository

Este es el repositorio principal (ver Figura 6.17) de la aplicación y el que más métodos posee, es el encargado de realizar todas las peticiones relacionadas con los lugares existentes en la base de datos del sistema. Los datos que necesitarán los ViewModel son los siguientes:

- Distintas listas de lugares dependiendo de la necesidad del usuario, la aplicación presenta lugares por: valoración, cercanía, opinión en twitter, añadidos a favoritos por el usuario, de una categoría dada, visitados o pendientes por visitar por el usuario.
- Lista de categorías posibles para un lugar. Por ejemplo: restaurante, museo, parque, etc...

En este repositorio se disponen de muchos métodos que devuelven listas según la pantalla en la que se encuentre el usuario, para ello se crea un

```
public class PlaceRepository extends Repository{

    private MutableLiveData<List<TPlace>> mPlacesList = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mTwitterPlacesList = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mNearestPlacesList = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mFavouritesPlacesList = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mCategoriesPlacesList = new MutableLiveData<>();
    private MutableLiveData<List<String>> mCategoriesList = new MutableLiveData<>();
    private MutableLiveData<TPlace> mPlace = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mPlaceVisited = new MutableLiveData<>();
    private MutableLiveData<List<TPlace>> mPlacesPendingToVisit = new MutableLiveData<>();

    //Callback personalizado tanto para las listas de lugares
    class PlaceListCallBack implements Callback{...}

    public void getPlaceByName(String placeName, String username) {...}

    public void addPlace(TPlace place){...}

    public void setPlaceToPendingVisited(TPlace place, String username) {...}

    public void modifyPlace(TPlace place, String oldName) throws JSONException {...}

    public void deletePlace(String placeName){...}

    public void getCategories() {...}

    public void setFavOnPlace(TPlace place, String username) {...}

    public void setVisitedOnPlace(TPlace place, String username) {...}
```

Figura 6.17: Clase PlaceRepository.

callback genérico que es usado por todas las funciones que se encargan de actualizar las listas del repositorio. Este tipo de callback genérico también se usa para los otros repositorios adaptándose al tipo de lista que necesitan.

```
class PlaceListCallBack implements Callback{

    private SimpleRequest simpleRequest;
    private MutableLiveData<List<TPlace>> placeList;

    public PlaceListCallBack(SimpleRequest simpleRequest, MutableLiveData<List<TPlace>> placeList){
        this.simpleRequest = simpleRequest;
        this.placeList = placeList;
    }

    @Override
    public void onFailure(@NotNull Call call, @NotNull IOException e) {...}

    @Override
    public void onResponse(@NotNull Call call, @NotNull Response response) throws IOException {...}
}
```

Figura 6.19: Clase PlaceListCallBack.

```
//si no hubo problemas...
List<TPlace> auxList = placeList.getValue();
List<TPlace> listFromResponse = PlaceRepositoryHelper.getListFromResponse(res);
if(listFromResponse == null){
    Log.d( tag: "PlaceListCallback",  msg: "Lista recibida nula...");
    return;
}
if(listFromResponse.isEmpty()){
    mSuccess.postValue(ControlValues.NO_MORE_PLACES_TO_LIST);
    return;
}
if (auxList == null){ //Lista vacia se crea
    placeList.postValue(PlaceRepositoryHelper.getListFromResponse(res));
}
else{ //Lista creada, se añade la lista recibida a la lista actual
    auxList.addAll(listFromResponse);
    placeList.postValue(auxList);
}
mSuccess.postValue(ControlValues.LIST_PLACES_OK);
```

Figura 6.20: Método onResponse si no hay errores.

```
public void listPlaces(int page, int quantity, String nickname, String searchText) {...}

public void listPlacesCategories(int page, int quantity, String nickname, String category, String searchText) {...}

public void listFavouritesPlaces(int page, int quantity, String nickname, String searchText) {...}

public void listTwitterPlaces(int page, int quantity, String nickname, String searchText) {...}

public void listNearestPlaces(int page, int quantity, String nickname, List<Double> point, String searchText) {...}

public void listNearestCategories(int page, int quant, String nickname, String category, String searchText, List<Double> point) {

public void listVisitedPlaces(int page, int quantity, String nickname, String searchText) {...}

public void listPendingToVisit(int page, int quantity, String nickname, String searchText) {...}
```

Figura 6.21: Métodos de actualización de lugares del repositorio.

El PlaceListCallBack (ver Figura 6.19) implementa la interfaz Callback que usa OkHttp para realizar llamadas al servidor, esta se compone de 2 métodos, uno para fallos y otro para procesar la respuesta (ver Figura 6.20). Para poder realizar la petición en el callback es necesario la instancia de SimpleRequest que se ha generado anteriormente y la lista de lugares que se quiere modificar (ver Figura 6.21) según el método que llama el ViewModel. Al procesar la respuesta se sigue la lógica de la figura 6.18.

6.2.7 Paquete models

Es el paquete en el que se encuentran las definiciones de todas las clases Transfer.

6.2.7.1 Transfers

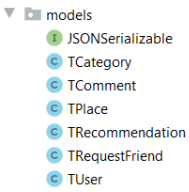


Figura 6.22: Clases transfer utilizadas en el proyecto.

Para cada modelo hay una clase transfer (ver Figura 6.22) con los atributos necesario y métodos set y get como indica el patrón DTO.

```
public interface JSONSerializable {
    JSONObject toJSON();
    String jsonToString();
}
```

Figura 6.23: Interfaz JSONSerializable.

Todas las clases transfer del proyecto implementan la interfaz JSONSerializable (ver Figura 6.23) necesaria para transformar los datos de un objeto a JSON y poder enviarlos en una petición http al servidor.

```
@Override
public int describeContents() { return 0; }

@Override
public void writeToParcel(Parcel dest, int flags) {...}

public TUser(Parcel in) {...}

public static final Creator<TUser> CREATOR = new Creator<TUser>() {...};
```

Figura 6.24: Métodos implementados para la clase TUser.

Además, para poder pasar objetos entre los fragmentos de la aplicación mediante la clase Bundle [1] que encapsula objetos, es necesario implementar la interfaz Parcelable [2] con los métodos y atributos que se pueden ver en la figura 6.24.

6.2.8 Paquete services

```
public class LocationTrack extends Service implements LocationListener {

    private Activity mActivity;
    private PermissionsManager permissionsManager;

    //Permissions
    private boolean checkGPS = false;
    private boolean checkNetwork = false;
    private boolean canGetLocation = false;

    //Data
    private Location loc;
    private double latitude;
    private double longitude;

    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; //m
    private static final long MIN_TIME_BW_UPDATES = 5000; //ms

    protected LocationManager locationManager;

    public LocationTrack(Activity mActivity) {...}

    //Los permisos se controlan con anterioridad con otras clases, ej: PermissionsManager
    @SuppressLint("MissingPermission")
    private Location getLocation() {...}
```

Figura 6.25: Atributos y método básico de LocationTrack.

En este paquete se encuentra la clase LocationTrack [6] (ver Figura 6.25). Esta clase actúa como un servicio Android [7] y hereda de Service, lo que le permite realizar operaciones en segundo plano.

En este caso, se usa para recoger las coordenadas del usuario con una frecuencia definida en los atributos de la clase. Solo está activa cuando el usuario acepta los permisos de localización de la aplicación.

Además usa la interfaz de LocationListener, para actualizar las coordenadas del usuario necesarias para obtener los lugares por cercanía.

6.2.9 Paquete ui

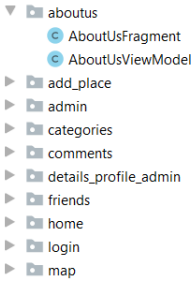


Figura 6.26: Vistas de la Aplicación.

Este paquete contiene los elementos de la vista (ver Figura 6.26) y los modelos de la vista(ViewModels) para comunicarse con los repositorios. Están organizados en un paquete por pantalla de la aplicación, donde cada paquete contiene, como mínimo, una clase Fragment y un ViewModel para comunicarse con el repositorio y efectuar las operaciones necesarias mediante la API.

6.2.10 Paquete aboutus

En este paquete se encuentra el fragmento que muestra información sobre la amplitud y motivo del proyecto y los integrantes del equipo.

6.2.11 Paquete add_place

En este paquete se encuentra el fragmento con la definición del formulario para la creación de un lugar.

Además, para poder seleccionar la ubicación correcta del lugar, dispone de una pantalla llamada AddPlaceMapboxActivity, que pide los permisos de geolocalización al usuario si no los posee y donde puede seleccionar el punto del mapa donde se encuentre el lugar que quiera subir a la aplicación.

Para poder utilizar la localización del usuario, se debe implementar las interfaces PermissionsManager y OnMapReadyCallback de Mapbox SDK. La primera hace referencia a los permisos que la aplicación pide al usuario al iniciar el mapa siendo en este caso los de localización y la segunda, a la acción del mapa cuando el usuario realiza acciones sobre el mapa.

Cuenta con diversas funciones en esta clase:

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    permissionsManager.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

Figura 6.27: Método onRequestPermissionsResult.

onRequestPermissionsResult (ver Figura 6.27) realiza la comprobación de los permisos que ha dado o denegado el usuario a la aplicación.

```
@Override
public void onPermissionResult(boolean granted) {
    if (granted && mapboxMap != null) {
        Style style = mapboxMap.getStyle();
        if (style != null) {
            enableLocationPlugin(style);
        }
    } else {
        Toast.makeText(this, R.string.user_location_permission_not_granted, Toast.LENGTH_LONG).show();
        finish();
    }
}
```

Figura 6.28: Método onPermissionResult.

En la función onPermissionResult (ver Figura 6.28) se activa o desactiva la función de localización según los permisos que haya dado el usuario.

```
@SuppressWarnings( {"MissingPermission"})
private void enableLocationPlugin(@NonNull Style loadedMapStyle) {
    // Check if permissions are enabled and if not request
    if (PermissionsManager.areLocationPermissionsGranted(this)) {

        // Get an instance of the component. Adding in LocationComponentOptions is also an optional
        // parameter
        LocationComponent locationComponent = mapboxMap.getLocationComponent();
        locationComponent.activateLocationComponent(LocationComponentActivationOptions.builder(
                this, loadedMapStyle).build());
        locationComponent.setLocationComponentEnabled(true);

        // Set the component's camera mode
        locationComponent.setCameraMode(CameraMode.TRACKING);
        locationComponent.setRenderMode(RenderMode.NORMAL);

    } else {
        permissionsManager = new PermissionsManager(this);
        permissionsManager.requestLocationPermissions(this);
    }
}
```

Figura 6.29: Método enableLocationPlugin.

La función enableLocationPlugin (ver Figura 6.29) activa el componente de localización del usuario si se han dado permisos, o pide los permisos en caso contrario. Esta función se utiliza en los casos que se utilice la localización del usuario en la aplicación.

Las otras dos funciones importantes son reverseGeocode, la cual transforma los datos recogidos por el mapa al confirmar la localización para enviarlos al formulario de manera entendible (código postal, calle, coordenadas, etc) y onMapReady que incluye la función onClick para cuando el usuario pulsa el botón para seleccionar el lugar.

6.2.12 Paquete admin

Contiene los archivos para la administración de los usuarios, donde muestra la lista de usuarios en la aplicación. Esta carpeta está formada por el fragmento AdminFragment, el viewModel AdminViewModel y el adaptador

UserListAdapter que sirve para dibujar los distintos usuarios en un **RecyclerView**.

```
public class UserListAdapter extends RecyclerView.Adapter<UserListAdapter.MyViewHolder> {
    private Activity activity;
    private List<TUser> listUser;
    private OnListObserver onListObserver;

    public UserListAdapter(Activity activity, List<TUser> listUser, OnListObserver onListListener) {
        this.activity = activity;
        this.listUser = listUser;
        this.onListObserver = onListListener;
    }

    public interface OnListObserver {
        void OnListClick(int position);
    }
```

Figura 6.30: Clase UserListAdapter con la interfaz para el Observer.

```
public class MyViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    ImageView iv_imgProfile;
    TextView tv_nameProfile;
    TextView tv_entireNameProfile;
    TextView tv_emailProfile;
    TextView tv_birthdayProfile;
    TextView tv_genderProfile;
    OnListObserver listObserver;

    public MyViewHolder(View itemView, OnListObserver onListObserver) {...}

    @Override
    public void onClick(View v) { this.listObserver.OnListClick(getAdapterPosition()); }
}
```

Figura 6.31: Clase MyViewHolder llamando al Observer.

```
@Override
public void OnListClick(int position) {
    Bundle bundle = new Bundle();
    TUser user = listUser.get(position);
    bundle.putParcelable(AppConstants.BUNDLE_PROFILE_LIST_DETAILS, user);

    //Le pasamos el bundle
    Navigation.findNavController(root).navigate(R.id.detailFragment, bundle);
}
```

Figura 6.32: Método OnListClick en AdminFragment.

Además para para que el Adapter pueda provocar acciones sobre la API, se implementa el patrón Observer (ver Figura 6.32) de tal manera que AdminFragment observa la interacción del usuario sobre los elementos de la vista (ver Figura 6.31) del UserListAdapter (ver Figura 6.30).

6.2.13 Paquete categories

En este paquete está implementado la parte de la vista que muestra la lista de categorías de los lugares. La lista de categorías está definida con la clase **CategoriesAdapter** y usa el patrón observer para poder reaccionar a las pulsaciones del usuario en alguno de los elementos de la lista. Esta pulsación llevará al usuario a una pantalla con los lugares de la categoría seleccionada.

6.2.14 Paquete comments

Contiene ficheros que sirven para mostrar la lista de comentarios de un lugar. Está el fragmento CommentsFragment se llama en el fragmento PlaceDetailFragment que es la vista principal donde muestra los detalles del lugar, el viewModel CommentsViemModel y el adaptador para cargar la lista CommentsListAdapter.

Al igual que en caso anteriores se aplica el patrón observer para que sea el fragmento el que notifique al ViewModel de realizar las operaciones correspondientes, en este caso, borrar un comentario.

6.2.15 Paquete details_profile_admin

Contiene las clases que hacen que se muestre el perfil de otro usuario. Los ficheros son detailFragment y detailViewModel.

6.2.16 Paquete friends

En este paquete, se implementan los fragmentos necesarios para la interfaz de los amigos de un usuario, en él podemos encontrar:

```
private void prepareViewPager() {
    BaseTabAdapter visitedTabAdapter = new BaseTabAdapter(getChildFragmentManager());

    FriendsFragmentFactory friendsFragmentFactory = new FriendsFragmentFactory();

    for(int i = 0; i < tabTitlesList.size(); i++){
        Fragment placeListFragment = friendsFragmentFactory.getInstance(tabTitlesList.get(i));
        visitedTabAdapter.addFragment(placeListFragment, tabTitlesList.get(i));
    }

    viewPager.setAdapter(visitedTabAdapter);
}
```

Figura 6.33: Método de asignación de fragmentos por apartado del ViewPager.

- Paquete subclasses. En este paquete se encuentran los fragmentos y adapter relaciones con la lista de amigos(FriendListFragment) y la lista de peticiones de amistad(FriendRequestListFragment) de un usuario.

Además, hay una clase que simula a una factoría para que devuelva un tipo de fragmento dependiendo de un tag dado. Esta factoría será utilizada por FriendsFragment.

- FriendsViewModel. Es un ViewModel común que usan tanto FriendListFragment, FriendRequestListFragment como AddFriendFragment para las peticiones al repositorio de las correspondientes listas.

- FriendsFragment. Un fragmento con un TabLayout vinculado a un ViewPager en el que se cargan los fragmentos FriendListFragment y FriendRequestListFragment. En él se usa un adapter genérico para la creación del ViewPager. El método de creación del adapter se puede observar en la Figura 6.33.

- AddFriendFragment. Un fragmento que contiene un formulario donde es necesario rellenar el nombre de usuario de la persona a la que se quiere agregar. Si no existe un nombre de usuario escrito recibe un mensaje de error mandando la petición de amistad. En caso contrario se recibe un mensaje exitoso.

6.2.17 Paquete home

```
private void prepareViewPager() {
    BaseTabAdapter homeTabAdapter = new BaseTabAdapter(getChildFragmentManager());

    PlaceFragmentFactory placeFragmentFactory = new PlaceFragmentFactory();

    for(int i = 0; i < tabTitlesList.size()-1; i++){
        Fragment placeListFragment = placeFragmentFactory.getInstance(tabTitlesList.get(i),  category: null);
        homeTabAdapter.addFragment(placeListFragment, tabTitlesList.get(i));
    }
    Fragment categoryFragment = new CategoriesFragment();
    int last = (tabTitlesList.size()-1);
    homeTabAdapter.addFragment(categoryFragment, tabTitlesList.get(last));

    viewPager.setAdapter(homeTabAdapter);
}
```

Figura 6.34: Método prepareViewPager.

Es el paquete en el que se encuentra el HomeFragment la vista principal de la aplicación. En ella se usa una clase que simula una factoría de fragments de lista de lugares y se construye un adapter con las correspondientes vistas para un TabLayout. A diferencia del caso anterior, el TabLayout se compone de 3 pantallas de listas de lugares y un último fragmento con la lista de categorías disponibles, en la figura 6.34 se muestra el código encargado de construir el anterior TabLayout.

6.2.18 Paquete login

Contiene las clases LoginFragment y loginViewModel para mostrar el formulario de inicio de sesión del usuario.

6.2.19 Paquete map

Contiene la clase MapboxActivity, que sirve para mostrar el mapa, el lugar destino, la localización del usuario y poder trazar la ruta y navegación desde el usuario al destino. En este caso se utiliza un activity en lugar de un fragment para la creación del mapa.

La clase contiene un menú con opciones a la derecha las cuales sirven para cambiar los estilos del mapa, pudiendo mostrar el tráfico, modo oscuro, vista satélite, etc.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_map_style, menu);
    return true;
}
```

Figura 6.35: Método onCreateOptionsMenu.

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
// Handle item selection
        switch (item.getItemId()) {
            case R.id.menu_streets:
                mapboxMap.setStyle(Style.MAPBOX_STREETS);
                return true;
            case R.id.menu_dark:
                mapboxMap.setStyle(Style.DARK);
                return true;
            case R.id.menu_light:
                mapboxMap.setStyle(Style.LIGHT);
                return true;
            case R.id.menu_outdoors:
                mapboxMap.setStyle(Style.OUTDOORS);
                return true;
            case R.id.menu_satellite:
                mapboxMap.setStyle(Style.SATELLITE);
                return true;
            case R.id.menu_satellite_streets:
                mapboxMap.setStyle(Style.SATELLITE_STREETS)
                return true;
            case R.id.menu_traffic:
                mapboxMap.setStyle(Style.TRAFFIC_DAY);
                return true;
            case android.R.id.home:
                finish();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

Figura 6.36: Método onOptionsItemSelected.

Con la función onCreateOptionsMenu (ver Figura 6.35) obtiene el menú y con onOptionsItemSelected cambia los estilos según cuál se seleccione.

El funcionamiento de los permisos es el mismo que al crear un lugar, por lo que no necesita mayor explicación.

```
buttonRoute.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Point originPoint = Point.fromLngLat(locationComponent.getLastKnownLocation().getLongitude(),
                locationComponent.getLastKnownLocation().getLatitude());
        getRoute(originPoint, destPoint);
        button.setEnabled(true);
        //button.setBackgroundResource(R.color.mapboxBlue);
    }
});
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        boolean simulateRoute = true;
        NavigationLauncherOptions options = NavigationLauncherOptions.builder()
                .directionsRoute(currentRoute)
                .shouldSimulateRoute(simulateRoute)
                .build();
        // Call this method with Context from within an Activity
        NavigationLauncher.startNavigation(MapboxActivity.this, options);
    }
});
putDestPoint(destPoint);
```

Figura 6.37: Método setOnClickListener de la ruta.

La parte más importante de este apartado es la creación de las rutas [28] y la navegación, para las cuales hay dos funciones onclick para cada uno de los botones. El primero, con el origen la localización del usuario y destino la del lugar, crea una ruta. El segundo, inicia una navegación cambiando el aspecto en el que se muestra el mapa, como se puede observar en la figura 6.37.

6.2.20 Paquete modify_place

Contiene las clases ModifyPlaceFragment y ModifyPlaceViewModel que muestran el formulario para modificar un lugar, así como ModifyPlaceMapboxActivity el cual tiene como fin poder cambiar las coordenadas de un lugar eligiendolas en el mapa.

El funcionamiento es el mismo que para add_place con la diferencia de que en este caso se pasa un objeto con información sobre el lugar para obtener las coordenadas iniciales.

6.2.21 Paquete place_details

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    inflater.inflate(R.menu.option_for_place_navigation_menu, menu);

    modifyPlace = menu.findItem(R.id.modify_place_menu_button);
    modifyPlace.setOnMenuItemClickListener(item -> {
        Bundle bundle = new Bundle();
        bundle.putParcelable(AppConstants.BUNDLE_PLACE_DETAILS, place);
        Navigation.findNavController(root).navigate(R.id.modifyPlaceFragment, bundle);
        return true;
    });
    deletePlace = menu.findItem(R.id.delete_place_menu_button);
    deletePlace.setOnMenuItemClickListener(item -> {
        deleteDialog();
        return true;
    });
    toPendingVisited = menu.findItem(R.id.pending_visited_menu_item);
    toPendingVisited.setOnMenuItemClickListener(item -> {
        mViewModel.placeToPendingVisited(place, App.getInstance().getUsername());
        return true;
    });
    MenuItem recommendPlace = menu.findItem(R.id.recommend_place_menu_item);
    recommendPlace.setOnMenuItemClickListener(item -> {
        onRecomClick();
        return false;
    });
```

Figura 6.38: Método para la creación del menú de opciones en los detalles de un Lugar.

Contiene el fragmento PlaceDetailFragment donde se muestran los detalles de un lugar. Hace uso de la clase SliderAdapter [18] para mostrar un carrusel de imágenes del lugar. Para crear el menú se necesita realizar un inflate para buscar los botones y asginar una acción a los botones (ver Figura 6.38). También se encuentra la clase PlaceDetailViewModel que permite diversas operaciones:

- Borrar un lugar si se es administrador.
- Marcar el lugar como favorito.
- Marcar el lugar como visitado.
- Recomendar el lugar a un amigo.

6.2.22 Paquete place_list

En este paquete se encuentran todas las clases relacionadas con las pantallas de listas de lugares de la aplicación. Es uno de los paquetes que más lógica recoge de la aplicación además de ser de las más usadas.

Para la construcción de las listas es necesario una clase PlaceListAdapter, en esta clase se define el elemento ViewHolder de la lista de lugares y la manera de construirla.

En MadridPlaces hay multitud de pantallas de lugares que poseen la misma estructura visual, la única diferencia entre ellas son los lugares que se presentan. Las listas pueden ser sobre los lugares más populares en la aplicación, los más cercanos, los populares en Twitter o los lugares de un tipo concreto como podrían ser un parque o un restaurante.

```
public abstract class BasePlaces extends Fragment implements PlaceListAdapter.OnPlaceListener, LogoutObserver {

    protected BaseViewModel mViewModel;
    protected View root;
    protected HashMap<Integer, OnResultAction> actionHashMap;

    protected MenuItem search_place;
    protected MenuItem mic_search_place;
    protected MenuItem addPlace;
    protected MenuItem nearestPlaces;

    protected SearchView searchView;

    //UI Elements
    protected SwipeRefreshLayout swipeRefreshLayout;
    protected NestedScrollView nestedScrollView;
    protected RecyclerView recyclerView;
    protected ProgressBar progressBar;
    protected ShimmerFrameLayout shimmerFrameLayout;

    protected List<TPlace> placeList;
    protected PlaceListAdapter placeListAdapter;

    protected int page = 1, limit = 3, quantum = 3;

    protected String search_text = "";
    private boolean endOfList;
```

Figura 6.39: Atributos de la clase BasePlaces.

Para poder reutilizar la misma vista y evitar repetir código se ha creado una clase abstracta BasePlaces (ver Figura 6.39) que es un fragmento que recoge una abstracción general de la vista de lugares de tal manera que las clases que hereden de BasePlaces sólo se tienen que preocupar de llamar al método correcto en el ViewModel con el que rellenar la lista de lugares del PlaceListAdapter.

```
//Funciones a implementar en los hijos según el tipo de lugares a mostrar
public abstract void listPlaces();
public abstract BaseViewModel getViewModelToParent();
```

Figura 6.40: Métodos abstractos de BasePlaces.

Los hijos tienen que implementar su propio método listPlaces() (ver Figura 6.40) que se llama en BasePlaces al llegar al fin de página, al usar el buscador o al refrescar los resultados con el SwipeRefreshLayout.

Además, es necesario implementar un método getViewModelToParent() (ver Figura 6.40) que lo que hace es devolver un ViewModel que hereda de BaseViewModel de tal manera que desde BasePlaces se pueda realizar una llamada al ViewModel que realize la operación correspondiente notificando al Repositorio.

```
public abstract class BaseViewModel extends ViewModelParent {
    protected PlaceRepository placeRepository;
    protected LiveData<List<TPlace>> mPlacesList = new MutableLiveData<>();

    protected abstract MutableLiveData<List<TPlace>> getPlaceListToParent();
    public abstract void listPlaceToParent(int page, int quant, String nickname, String searchText);

    @Override
    public void init() {
        placeRepository = new PlaceRepository();

        mSuccess = super.updateOnChange(placeRepository.getSuccess());
        mPlacesList = super.updateOnChange(getPlaceListToParent());
    }

    public void listPlaces(int page, int quant, String nickname, String serchText){
        listPlaceToParent(page, quant, nickname, serchText);
    }

    public void setFavOnPlace(TPlace place, String username) {
        placeRepository.setFavOnPlace(place, username);
    }

    public LiveData<List<TPlace>> getPlacesList(){ return mPlacesList; }

}
```

Figura 6.41: Clase abstracta de BaseViewModel.

A su vez las clases que hereden de BaseViewModel (ver Figura 6.41) deben implementar su propio método listPlaceToParent() para realizar la llamada correcta al Repositorio.

Así al definir esta estructura, se aprovecha la resolución de tipos en tiempo de ejecución que ofrece el poliformismo. Así, las clases hijas son muy sencillas y pueden implementar distinta lógica para la carga de los lugares.

```java
public class RatingPlacesFragment extends BasePlaces {

    public RatingPlacesFragment(){
        super();
        placeList = new ArrayList<>();
    }
    @Override
    public void listPlaces() {
        super.mViewModel.listPlaces(page, quantum, App.getInstance().getUsername(), search_text);
    }

    @Override
    public BaseViewModel getViewModelToParent() {
        RatingPlaceViewModel rvm = new ViewModelProvider( owner: this).get(RatingPlaceViewModel.class);
        return rvm;
    }
}
```

Figura 6.42: Fragmento RatingPlacesFragment que hereda de BasePlaces.

```java
public class RatingPlaceViewModel extends BaseViewModel {
    @Override
    protected MutableLiveData<List<TPlace>> getPlaceListToParent() {
        return placeRepository.getPlacesList();
    }

    @Override
    public void listPlaceToParent(int page, int quant, String nickname, String searchText) {
        placeRepository.listPlaces(page, quant, nickname, searchText);
    }
}
```

Figura 6.43: ViewModel de RatingPlacesFragment que hereda de BaseViewModel.

En el caso de los lugares por valoración en la aplicación hay una clase muy sencilla. Para implementar otros lugares como los favoritos de un usuario, por la valoración en twitter o por cercanía la estructura es la misma que la de las Figuras 6.42 y 6.43.

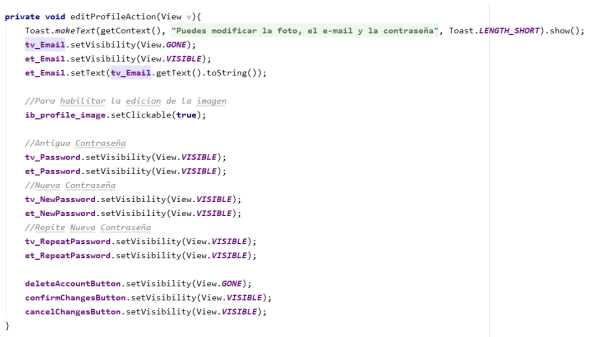
```
public class CategoryPlacesFragment extends BasePlaces {

    protected String category;

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

        TCategory tCategory = (TCategory) getArguments().getParcelable(AppConstants.BUNDLE_CATEGORY_TYPE);
        category = tCategory.getName();

        //Poner el nombre de la categoria en la toolbar
        AppCompatActivity appCompatActivity = (AppCompatActivity) getActivity();
        ActionBar actionBar = appCompatActivity.getSupportActionBar();
        actionBar.setTitle(category);

        //Primero es necesario instanciar el tipo de categoria de este fragmento para que la llamada a super
        //no ejecute con valores nulos la llamada a la BD
        View superView =  super.onCreateView(inflater, container, savedInstanceState);

        return superView;
    }

    @Override
    public void listPlaces() { super.mViewModel.listPlaces(page, quantum, App.getInstance().getUsername(), search_text);  }

    @Override
    public BaseViewModel getViewModelToParent() {...}

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {...}
```

Figura 6.44: Fragmento de la lista de los lugares por categoria.

En cambio es posible que haya alguna pantalla de lista de lugares que necesite algún parámetro extra o algo más de lógica, como es el caso de la lista según una categoría seleccionada (ver Figura 6.44). La abstracción con BasePlaces permite seguir teniendo una estructura limpia y que no repite código.

6.2.23 Paquete profile

Contiene las clases ProfileFragment y ProfileViewModel para la vista del perfil del usuario.



```
private void editProfileAction(View v){
    Toast.makeText(getContext(), "Puedes modificar la foto, el e-mail y la contraseña", Toast.LENGTH_SHORT).show();
    tv_Email.setVisibility(View.GONE);
    et_Email.setVisibility(View.VISIBLE);
    et_Email.setText(tv_Email.getText().toString());

    //Para habilitar la edicion de la imagen
    ib_profile_image.setClickable(true);

    //Antigua Contraseña
    tv_Password.setVisibility(View.VISIBLE);
    et_Password.setVisibility(View.VISIBLE);
    //Nueva Contraseña
    tv_NewPassword.setVisibility(View.VISIBLE);
    et_NewPassword.setVisibility(View.VISIBLE);
    //Repite Nueva Contraseña
    tv_RepeatPassword.setVisibility(View.VISIBLE);
    et_RepeatPassword.setVisibility(View.VISIBLE);

    deleteAccountButton.setVisibility(View.GONE);
    confirmChangesButton.setVisibility(View.VISIBLE);
    cancelChangesButton.setVisibility(View.VISIBLE);
}
```

Figura 6.45: Método que se ejecuta al editar el perfil.

La vista se ha realizado de forma que al intentar modificar el perfil, no se cambie de fragmento. Para ello se hace uso de las visibilidades de los objetos. Por ejemplo, la Figura 6.45 muestra qué objetos se hacen visibles o no visibles cuando se pulsa en el botón de editar el perfil.

6.2.24 Paquete recommendations

Contiene las clases RecommendationsFragment y RecommendationsViewModel, para crear la vista de la pestaña de recomendaciones y enlazar los datos con la vista. Además en la carpeta subclases contiene:

- RecommendationsFragmentFactory. Clase que simula una factoría para la creación de fragmentos en el ViewPager del TabLayout de la vista
- Paquete my_recommendations: MyRecommendationsAdapter y MyRecommendationsFragment para mostrar las recomendaciones del usuario.
- Paquete pending_recommendations: PendingRecommendationsFragment y PendingRecommendationsViewModel para mostrar las recomendaciones pendientes por aceptar o rechazar del usuario.

6.2.25 Paquete register

Contiene las clases RegisterFragment y RegisterViewModel para la vista del formulario de registro en la aplicación.

```
try {
    uri = data.getData();
    bitmap = MediaStore.Images.Media.getBitmap(getContext().getContentResolver(), uri);
} catch (IOException e) {
    e.printStackTrace();
}
showImages();
```

Figura 6.46: Try/Catch que recoge el dato y lo convierte a Bitmap.

En el formulario se pide opcionalmente añadir una imagen de perfil [19]. Se usa la función insertImagesFromGallery() que crea una actividad para seleccionar una o varias imágenes de la galería. Una vez que el usuario ha seleccionado la imagen, se recogen los datos en la función onActivityResult(int requestCode, int resultCode, @Nullable Intent data) donde el dato se convierte a Bitmap. La figura 6.46 representa como se recoge el dato y se convierte a Bitmap.

Una vez que el usuario se registre, para guardar la imágen, se convierte el Bitmap a un String y el servidor se encarga de volver a construir la imagen para guardarla como un enlace.

6.2.26 Paquete sendRecommendation

Contiene las clases SendRecomendationFragment y SendRecomendationViewModel, además de tener la clase SendRecomendationAdapter que sirve para enlazar los datos de la lista de amigos con la vista del formulario para enviar recomendaciones.

6.2.27 Paquete settings

Contiene las clases SettingsFragment y SettingsViewModel para la vista de la pantalla de opciones de la aplicación. En el fragmento Settings están implementadas las funciones relacionadas con el control de los permisos de geolocalización del usuario y la configuración del idioma de la aplicación.

6.2.28 Paquete visited

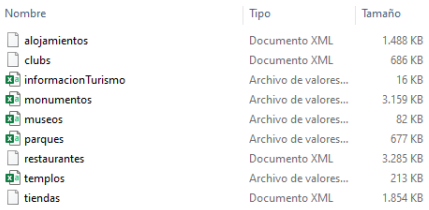
Contiene las clases VisitedFragment y VisitedViewModel para crear la vista con la pestaña de Lugares Visitados y Lugares Pendientes de Visitar. Además tiene las siguientes clases en la carpeta subclasses:

- **VisitedFragmentFactory**: Clase que simula una factoría para la creación de fragmentos en el ViewPager del TabLayout de la vista
- **Paquete pendingVisited**: En el se encuentran las clases PendingVisitedFragment y PendingVisitedViewModel para la vista de los lugares pendientes por visitar del usuario. Estas clases heredan de BasePlaces y BaseViewModel respectivamente
- **Paquete visitedPlaces**: En el se encuentran las clases VisitedPlacesFragment y VisitedPlacesViewModel para la vista de los lugares visitados por el usuario. Estas clases heredan de BasePlaces y BaseViewModel respectivamente.

6.3 Implementación de la recopilación de datos

En este apartado se implementa una recolección de datos con diversas herramientas para poblar la base de datos y usarlo en la aplicación Android:

- Con Python utilizando el editor Visual Studio Code, se recolecta información utilizando librerías como:
  - Pandas [42], para la lectura de datasets en csv.
  - ElementTree [49], para la lectura de datasets en xml.
  - MySqldb [26], para cargar los datos obtenidos a la base de datos.
- Con Web Scraping con la librería Beautifulsoup [52] y la librería Selenium [34] se extraen las imágenes de todos los lugares realizando peticiones HTTP.



Nombre	Tipo	Tamaño
alojamientos	Documento XML	1.488 KB
clubs	Documento XML	686 KB
informacionTurismo	Archivo de valores...	16 KB
monumentos	Archivo de valores...	3.159 KB
museos	Archivo de valores...	82 KB
parques	Archivo de valores...	677 KB
restaurantes	Documento XML	3.285 KB
templos	Archivo de valores...	213 KB
tiendas	Documento XML	1.854 KB

Figura 6.47: Lista de datasets sin procesar.

Existen un total de nueve datasets en formato CSV y XML (ver Figura 6.47), todos recogidos de la página oficial de datos abierto del Ayuntamiento de Madrid.

Cada dataset contiene una cabecera con más de 20 columnas de información relevante acerca del lugar. Se descartan columnas irrelevantes para el desarollo de la aplicación y se crea un script en Python para transformar la información, a datos productivos para la aplicación.

```
print("Comenzando el preproceso de los datasets...")

templos = pd.DataFrame(columns=['Nombre','Descripcion','Latitud', 'Longitud', 'Tipo',
                                'Clase_vial', 'Nombre_vial', 'Numero_vial', 'Codigo_postal'])

dataTemplos = pd.read_csv("datasets/templos.csv", sep='delimiter', header=None, engine='python')

for i in range(0, len(dataTemplos)):
    prueba = dataTemplos.iloc[i, 0] #Recoge toda la fila
    res = limpiezaTemplos(prueba)
    if(res[1] == ""):
        res[1] = "Sin descripcion"
    auxTipo = res[4].split("/")
    if('http' not in res[2] and 'Ntilde' not in res[2] and len(auxTipo) != 1):
        templos = templos.append({'Nombre' : res[0],
                                  'Descripcion' : res[1],
                                  'Latitud' : res[2],
                                  'Longitud' : res[2],
                                  'Tipo' : auxTipo[3],
                                  'Clase_vial' : res[5],
                                  'Nombre_vial' : res[6],
                                  'Numero_vial' : res[7],
                                  'Codigo_postal' : res[8]}, ignore_index=True)

dataTemplos.drop(dataTemplos.columns, axis=1)

templos.sort_values(by=['Nombre'], inplace=True)

templos.to_csv("datasets_procesados/TemplosProcesados.csv",index=False, encoding='utf-8-sig', sep= '\t')
print("CSV de Templos creados")
```

Figura 6.48: Limpieza de datos (Parte 1).

El primer script en Python que se muestra en la Figura 6.48, genera nuevos datasets en formato CSV completando tres funciones vitales para la transformación de datos, reducir la cantidad de columnas, sustituir valores nulos y reemplazar caracteres especiales.

En la siguiente Figura 6.49 se muestra el proceso de transformación del datasets referidos a los templos en Madrid.

```
templos = pd.DataFrame(columns=['Nombre','Descripcion','Latitud', 'Longitud', 'Tipo',
                                'Clase_vial', 'Nombre_vial', 'Numero_vial', 'Codigo_postal'])

dataTemplos = pd.read_csv("datasets/templos.csv", sep='delimiter', header=None, engine='python')
```

Figura 6.49: Limpieza de datos (Parte 2).

En primer lugar, se crea un dataframe con una cabecera inicial para estructurar los datos que se almacenarán dentro. Luego, es necesario leer el dataset que está almacenado en una carpeta llamada "datasets". Para ello se usa la librería Pandas.

La librería Pandas es de software libre es muy conocida en el mundo de Big Data para la manipulación y análisis de datos de forma sencilla y eficiente.

```
def limiezaTemplos(dataTemplos):
    separate = re.split(r';', dataTemplos)
    nombre = separate[1].replace('"', '')
    descripcion = separate[2].replace('"', '') if separate[2] != '' else "Sin especificar"
    clase_vial = separate[10].replace('"', '') if separate[10] != '' else "Sin especificar"
    nombre_vial = separate[9].replace('"', '') if separate[9] != '' else "Sin especificar"
    numero_vial = separate[12].replace('"', '') if separate[12] != '' and separate[12] != 's/n' else '0'
    codigo_postal = separate[19].replace('"', '') if separate[19] != '' and separate[19] != '0' else '00000'
    latitud = separate[24].replace('"', '') if separate[24] != '' else '0'
    longitud = separate[25].replace('"', '') if separate[25] != '' else '0'
    tipo = separate[29].replace('"', '') if separate[29] != '' else "Sin especificar"
    url = separate[8].replace('"', '') #Únicamente existe una foto por cada templo
    return [nombre, descripcion, latitud, longitud, tipo, clase_vial, nombre_vial, numero_vial, codigo_postal, url]
```

Figura 6.50: Limpieza de datos (Parte 3).

Tras almacenar en una variable la lectura del dataset, se realiza una iteración de cada fila del contenido, ejecutando una función definida que sirve para limpiar datos nulos, caracteres innecesarios y reducir el número de valores de la fila especificada (ver Figura 6.50).

```
templos.sort_values(by=['Nombre'], inplace=True)

templos.to_csv("datasets_procesados/TemplosProcesados.csv",index=False, encoding='utf-8-sig', sep= '\t')
print("CSV de Templos creados")
```

Figura 6.51: Limpieza de datos (Parte 4).

Después de procesar las filas, se añade el nuevo contenido dentro del dataframe creado, y se crea un dataset ordenado por el nombre de los lugares (ver Figura 6.51).

No es posible realizar el mismo procedimiento de transformación para todos los tipos de lugares ya que cada datasets tienen una estructura diferente y peculiaridades únicas que incapacitan una ejecución común para todos los datos.

Una vez transformados los datos, es necesario extraer las imágenes relacionadas a los lugares, que están almacenados en la columna Url de los

conjuntos de datos. Para ello, se utiliza la técnica de web scraping con la librería Beautiful Soup. Esta librería sirve para extraer datos de documentos HTML analizando de forma automática la indexación de la web y convirtiéndolo en un árbol con todos los elementos de la página.

extraccionImagenes.py

```
def Monumentos():
    dataMonumentos = pd.read_csv("monumentos.csv", sep='delimiter', header=None, engine='python'
    for i in range(1, len(dataMonumentos)):
        prueba = dataMonumentos.iloc[i, 0]
        separate = re.split(r'";"', prueba)
        name = separate[0].split('"')[1]
        url = separate[6].replace('"', '') #Url para hacer web Scrapping

        r = requests.get(url)
        html = r.text
        soup=BeautifulSoup(html,'html.parser')
        allImg = soup.find_all('a', class_="carouselNoticia-link")
        if(len(allImg) == 0):
            clase_vial = separate[8]
            nombre_vial = separate[7]
            numero_vial = separate[10] if separate[10] != '' and separate[10] != 's/n'else '0'
            codigo_postal = separate[17]
            listaSelenium.append({"locationName" : name,
                                  "search" :
                                      name + ", monumento " +
                                      clase_vial + " " +
                                      nombre_vial + " " +
                                      numero_vial + ", " +
                                      codigo_postal
                                  })
        for a in allImg:
            image = "https://patrimonioypaisaje.madrid.es" + a["href"]
            #data=requests.get(image)   # En el caso de volver a BLOB
            #photo=data.content
            sql=" INSERT IGNORE INTO location_images (location_name, image) VALUES (%s, %s)"
            cursor.execute(sql, (name, image))
            mydb.commit()
```

Figura 6.52: Web Scraping (Parte 1).

A continuación se explica la implementación de la técnica Web Scraping (ver Figura 6.52).

Se crea un script en Python llamado “extraccionImagenes.py” donde carga las imágenes extraídas con web scraping y Selenium, a la base de datos con Mysqldb.

Para lugares extraídos que no posean imágenes, se aplica el uso de la herramienta Selenium para extraer, con el navegador de Google Chrome, imágenes de forma automática.

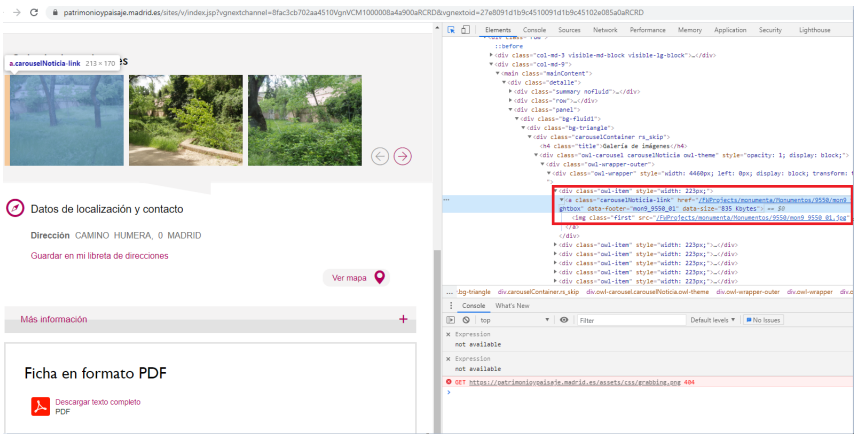


Figura 6.53: Ejemplo de análisis del DOM Web.

Para la parte de web scraping, se extraen las urls del dataset de monumentos.csv. Se usa la librería urllib.request [50] para realizar una conexión HTTP con la url y se crea la estructura del documento web en forma de árbol con BeautifulSoup (ver Figura 6.53) para su análisis.

```
r = requests.get(url)
html = r.text
soup=BeautifulSoup(html,'html.parser')
allImg = soup.find_all('a', class_="carouselNoticia-link")
for a in allImg:
    image = "https://patrimonioypaisaje.madrid.es" + a["href"]
    sql=" INSERT IGNORE INTO location_images (location_name, image) VALUES (%s, %s)"
    cursor.execute(sql, (name, image))
    mydb.commit()
```

Figura 6.54: Web Scraping (Parte 2).

La estrategia aplicada con la técnica de web scraping es simple. En primer lugar, se realiza un análisis de la estructura de la página web para identificar la localización de las imágenes y en qué etiquetas pertenece. En el ejemplo de la Figura 6.54, se observa que las imágenes están en un elemento <a>con etiqueta "carouselNoticia-link".

```
def main(): #Uso de Selenium exclusivamente para datos sin imágenes
    selectLugar() #Devuelve la lista de los nombres de todos los lugares exceptos los monumentos
    print(len(listaSelenium))
    driver = webdriver.Chrome() #Accede al navegador
    for name in listaSelenium:
        inp = name["search"] #Recoge el nombre del lugar con su direccion para aumentar la precisión de búsqueda
        url = 'https://www.google.com/search?q='
        +str(inp)+
        '&source=lnms&tbm=isch&sa=X&ved=2ahUKEwie44_AnqLpAhUhBWMBHUFGD90Q_AUoAXoECBUQAw&biw=1920&bih=947'
        #Recoge 5 fotos scrappeando Google fotos y los almacena en formato BLOB en la BD
        find_urls(name["locationName"],url,driver,5)
```

Figura 6.55: Selenium (Parte 1).

Después de realizar el análisis, se recogen las urls que contengan las imágenes y las cargan en la base de datos con la librería Mysqldb. Existen una

minoría de casos donde no existe ninguna imagen dentro de la página web, por lo que se recurre al mismo procedimiento aplicando Selenium para la recopilación de imágenes, como se observa en la Figura 6.55.

Por lo general, Selenium es una herramienta que está pensada para realizar pruebas en aplicaciones basadas en web. Sin embargo, puede ser utilizado también para automatizar funciones en un navegador web. Por esa razón, parece útil para poder extraer imágenes de Google eficientemente.

```
listaSelenium = [] #Lista global que contendrá todos los lugares que serán procesados con Selenium

def find_urls(inp,url,driver,iterate):
    try:
        driver.get(url)
        for j in range (1,iterate+1):
            imgurl = driver.find_element_by_xpath('//div//div//div//div//div/'
            +'/div//div//div//div//div['+str(j)+']//a[1]//div[1]//img[1]')

            imgurl.click() #Hace un click como un usuario normal
            time.sleep(1) #Sleep para evitar el captcha de Google
            img = driver.find_element_by_xpath('//body/div[2]/c-wiz/div[4]/div[2]/div[3]/div/'
            +'div/div[3]/div[2]/c-wiz/div/div[1]/div[1]/div[1]/div[2]/a/img').get_attribute("src")

            time.sleep(1)
            if("http" in img):
                print(img)
                sql=" INSERT IGNORE INTO location_images (location_name, image) VALUES (%s, %s)"
                cursor.execute(sql, (inp, img))
                mydb.commit()
    except:
    #Para saber qué lugares están fallando y probablemente se deba almacenar fotos manualmentes
        print(inp)
        pass
```

Figura 6.56: Selenium (Parte 2).

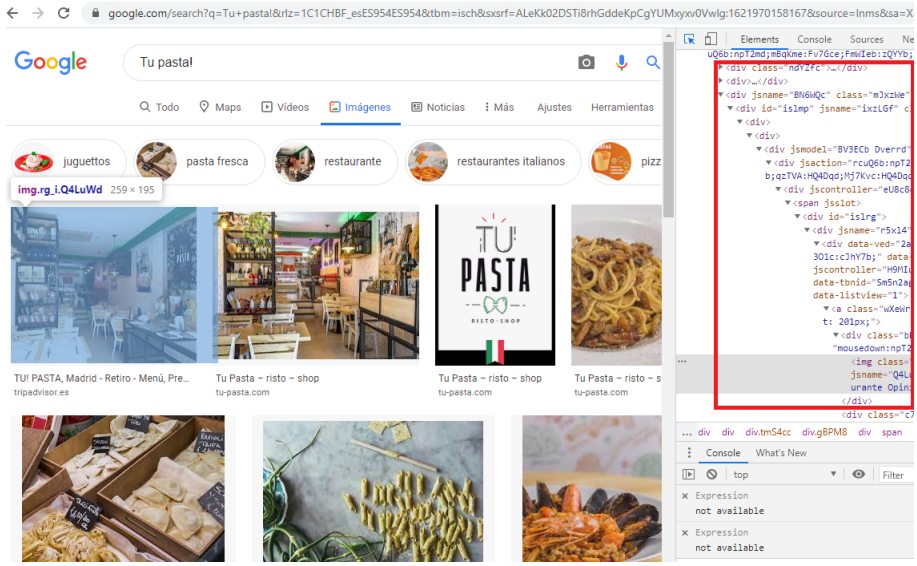


Figura 6.57: Selenium (Parte 3).

Para la implementación de Selenium, se genera una variable global, que contiene una lista de lugares que no se han podido extraer imágenes vía Web

Scraping o por datasets. Una vez establecido la lista, se itera en un bucle para realizar la búsqueda automática de los diferentes lugares (ver Figura 6.56).

Normalmente, Google bloquea la interfaz si se realizan peticiones aceleradas y comprueba que no es un robot el que realiza estas peticiones.

Para evitar este inconveniente, se establece un sleep() [48] para ralentizar el proceso.

Para usar Selenium, se debe realizar un análisis del DOM de la interfaz de Google para extraer las imágenes, algo parecido al proceso que realizamos con Web Scraping (ver Figura 6.57).

Google Chrome actualiza el DOM frecuentemente, por lo que será necesario actualizar el script para que Selenium pueda ejecutar el proceso automatizado de forma correcta.

Tras analizar la estructura del documento, se prepara el driver de Selenium para que busque por Google Fotos, el nombre y la dirección del lugar deseado, y realizar clicks a las 5 primeras imágenes para extraer sus urls y cargarlos en la base de datos.

El tiempo de ejecución total del script es bastante elevado, ya que hay una gran cantidad de lugares sin imágenes que han sido necesario el uso de Selenium. De media, la ejecución de la extracción de imágenes de un lugar es de 10 segundos.

Existe un segundo script "extraccionImagenes2.py" donde realiza una segunda extracción de los lugares que no se han podido recoger ninguna imagen por razones desconocidas.

cargaDeDatos.py

```
#Inserta todas las categorias de los lugares
def insertarTipos(URL):

    lastCategory = []
    csv_data = pd.read_csv(URL + ".csv", sep='delimiter', header=None,
                           engine='python', encoding='utf-8-sig')
    for i in range(1, len(csv_data)):
        prueba = csv_data.iloc[i, 0]
        separate = re.split(r'\t', prueba)
        #Quitar algunos casos donde el tab no funciona bien
        sep = re.sub('[*0-9]', '', separate[4])
        #Cambia los nombres de las categorias
        separate[4] = categorias(separate[4])
        if(sep != "-." and separate[4] not in lastCategory):
            cursor.execute("INSERT IGNORE INTO type_of_place (category) VALUES (%s)", [separate[4]])
            lastCategory.append(separate[4])
    mydb.commit()
```

Figura 6.58: Método que carga las categorías.

Una vez se han limpiado, extraído y transformado los datos, se pasa al último paso del proceso ETL, que es la carga de los datos a la base de datos relacional (ver Figura 6.58).

El script que se encarga de realizar la carga de todos los lugares de la tabla "locations" de la base de datos se llama "cargaDeDatos.py". Este script se compone principalmente de la carga esencial de dos componentes.

- El primero es cargar de todas las categorías que puedan existir dentro de los datos transformados de los originales. Se insertan en la tabla "type_of_place".

- El segundo es insertar todos los lugares en la base de datos, teniendo en cuenta el id único de la categoría que pertenezca los lugares insertados.

```
def insertarSQL(URL): #Inserta todos los lugares
    csv_data = pd.read_csv(URL + ".csv", sep='delimiter', header=None, engine='python',  encoding='utf-8-sig')
    for i in range(1, len(csv_data)):
        prueba = csv_data.iloc[i, 0]
        separate = re.split(r'\t', prueba)
        #Descartar manualmente caracteres raros
        separate[0] = separate[0].replace('&Aacute;lbora', 'Á')
        separate[0] = separate[0].replace('&aacute;lbora', 'á')
        separate[1] = re.sub('\u0301', '', separate[1])
        separate[1] = re.sub('\u0303', '', separate[1])
        separate[0] = re.sub('\u014c', 'O', separate[0])
        separate[1] = re.sub('\u014c', 'O', separate[1])
        separate[1] = re.sub('\u200b', '', separate[1])
        separate[1] = re.sub('\u2010', '', separate[1])
        separate[1] = re.sub('\x96', '', separate[1])
        separate[1] = re.sub('\x93', '', separate[1])
        separate[1] = re.sub('\x94', '', separate[1])
        separate[1] = re.sub('\x85', '', separate[1])
        if(len(separate) == 9 and (separate[2] != '' or separate[3] != '') and separate[2].count('.') == 1):
            #Cambia los nombres de las categorias
            separate[4] = categorias(separate[4])
            #Recoge la categoria de la tabla de tipos
            cursor.execute( "SELECT * FROM type_of_place WHERE category LIKE %s", [separate[4]])
            tipo = cursor.fetchone()
            sql2 = "INSERT IGNORE INTO location ("name,
                                                 description,
                                                 coordinate_latitude,
                                                 coordinate_longitude,
                                                 type_of_place,
                                                 road_class,
                                                 road_name,
                                                 road_number,
                                                 zipcode")
                                          VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
```

Figura 6.59: Método que carga los lugares.

Para insertar las categorías de los lugares (ver Figura 6.59), es necesario recorrer los datasets generados, descartando todas las que sean duplicadas. Cuando se insertan, generan un id único para cada categoría que serán valores necesarios para insertar en la tabla de los lugares y de esta forma tener una relación con la tabla de "type_of_place".

En la segunda parte del script, estan las inserciones de los lugares en la tabla de "location". Antes de insertar los datos, se realiza un filtrado de caracteres especiales para no cometer ningún error a la hora de insertar datos dentro de la base de datos y no sea posible realizar una corrección manual.

A continuación, se explica cómo se extraen los tweets de todos los lugares y posteriormente, se realiza un análisis del sentimiento.

```
#Lectura del .env
load_dotenv()
consumer_key = os.getenv("consumer_key")
consumer_secret = os.getenv("consumer_secret")
access_token =os.getenv("access_token")
access_token_secret = os.getenv("access_token_secret")

#Autentificación
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

#Usamos la variable api para utilizar distintas funcionalidades
api = tweepy.API(auth, wait_on_rate_limit=True)

#Inicializamos la API de Google traductor
translator = google_translator()
```

Figura 6.60: Autenticación para el uso de la API de Twitter.

Para empezar, se configura la autenticación de la cuenta registrada de la API de Twitter [59] (ver Figura 6.60). Todos los parámetros están almacenados en un archivo de configuración .env no almacenado en el repositorio para aumentar la privacidad de la cuenta. También se configura una variable "translator"para preparar el entorno de la API Google Traductor [60]. Es necesario traducir los tweets en inglés para realizar el análisis del sentimiento con TextBlob.

```
#Recogemos de la BD los nombres de todos los lugares
cursor.execute( "SELECT name FROM location") # --> ~5000 lugares
locations = cursor.fetchall()

for location in locations:
    try:
        #Generamos los últimos 10 tweets acerca del lugar (parametro modificable)
        tweets = tweepy.Cursor(api.search, q=location[0], tweet_mode='extended').items(10)
        listRates = []
        n_positiveTweets = 0
        n_negativeTweets = 0
        n_neutralTweets = 0

        avgRate = 0 #Si no hay tweets, avgRate = 0
        for tweet in tweets:
            cleanedTweet = p.clean(tweet.full_text) #Preprocesa el texto, quita los #, urls y emojis.

            translation = cleanedTweet

            if(len(cleanedTweet) > 3): #Para hacer el análisis como mínimo es necesario 3 letras
                try:
                    lang = translator.detect(cleanedTweet)
                    #TextBlob no analiza frases en español, por lo que hay que traducirlo. Se perderá un poco de precisión
                    if(lang  == 'es'):
                        translation = translator.translate(cleanedTweet)
                except Exception as e:
                    print("Ha habido un error:", repr(e))
                    pass
```

Figura 6.61: Limpieza y traducción de tweets.

La biblioteca Textblob [39] se usa para procesar texto y realizar procesamiento de lenguaje natural, como clasificación, traducción y análisis del sentimiento (ver Figura 6.61). No obstante, es necesario preprocesar el texto de los tweets, ya pueden contener Hashtags, Retweets e incluso emojis.

El script se encarga de la recogida de todos los lugares almacenados en la base de datos con una query SQL simple y se almacena en una lista.

A continuación se itera la lista creada, para buscar como máximo 10 tweets acerca del nombre del lugar. Hay casos en los que no existe ningún tweet ya que el lugar puede no ser conocido o que directamente la gente no haya twitteado nunca.

Posteriormente, se aplica el uso de la librería Preprocessor [46] para

limpiar sin dificultad un tweet y convertirlo en texto claro, sin emojis ni hashtags. La funcionalidad de esta biblioteca sirve específicamente para este propósito. La herramienta simplifica el código sin necesidad de utilizar expresiones regulares para limpiar el texto.

```
analyzed = TextBlob(translation)
#print(analyzed) # Comprobar la traducción
#print(analyzed.tags) # --> Se puede ver el análisis sintáctico de la frase
#print(analyzed.sentiment)
rate = classifyPolarity(analyzed.sentiment.polarity)
listRates.append(rate)
sentiment = detectSentiment(analyzed.sentiment.polarity)
if(sentiment == 0):
    n_neutralTweets = n_neutralTweets + 1
elif(sentiment == 1):
    n_positiveTweets = n_positiveTweets + 1
elif(sentiment == -1):
    n_negativeTweets = n_negativeTweets + 1
```

Figura 6.62: Análisis del sentimiento (Parte 1).

Finalmente se realiza un cálculo de la polaridad del tweet. La polaridad no es más que una puntuación que define lo positivo o lo negativo que es el tweet. El resultado es un rango entre 1 a -1, siendo 1 muy positivo y -1 muy negativo (ver Figura 6.62).

```
def classifyPolarity(polarity):
    rate = 0
    #Vamos a clasificarlo del 1 al 5, siendo 1 muy negativo y 5 muy positivo
    if (polarity == 0):  # 3
        rate = 3
    elif (polarity > 0 and polarity <= 0.5): #4
        rate = 4
    elif (polarity > 0.5 and polarity <= 1): #5
        rate = 5
    elif (polarity > -0.5 and polarity <= 0): #2
        rate = 2
    elif (polarity > -1 and polarity <= -0.5): #1
        rate = 1
    return rate
```

Figura 6.63: Análisis del sentimiento (Parte 2).

En la aplicación, se utiliza dichos tweets como comentarios extras referidos a los lugares registrados en la base de datos. Para ello, se clasifica y recoge la polaridad para generar comentarios valorados del 1 al 5 (ver Figura 6.63).

Tras el cálculo de la polaridad de todos los tweets, se insertan en la tabla "twitter_ratings" la cantidad de tweets, neutrales, negativos y positivos, y una media de las valoraciones generadas con las polaridades de los tweets.

6.4 Implementación de la API REST

En este apartado se explicará detalladamente las funcionalidades creadas para la comunicación efectiva entre la aplicación Android con la base de datos MySQL [45].

API REST es un servicio web capaz de conectar mediante peticiones HTTP entre dos programas. Se utiliza Flask como micro-framework para crear el servicio API REST por la facilidad de uso y envíos de peticiones rápidas gracias a su estructura ligera.

Para un buen diseño de API REST, se tuvo en cuenta la metodología KISS (Keep It Simple, Stupid), que se basa en dejar lo más simple posible la API y no hacer falta documentación para explicar cómo funcionan las llamadas.

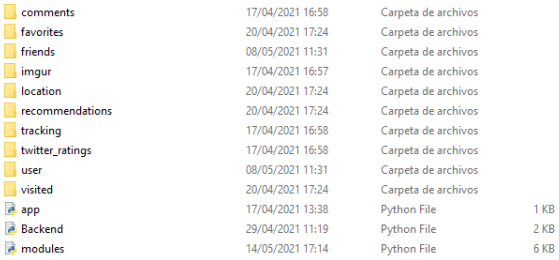


Figura 6.64: Directorio de la estructura del servicio web.

Primero se mostrará el contenido de la carpeta Backend del proyecto como se ve en la Figura 6.64.

Backend.py

```
#Configuracion del flask
from app import app
#Clases que contienen las rutas de las peticiones
from user.userClass import userClass
from location.locationClass import locationClass
from comments.commentsClass import commentsClass
from visited.visitedClass import visitedClass
from tracking.trackingClass import trackingClass
from twitter_ratings.twitter_ratingsClass import twitter_ratingsClass
from favorites.favoritesClass import favoritesClass
from recommendations.recommendationsClass import recommendationsClass
from friends.friendsClass import friendsClass

#Todas las rutas divididas en Blueprints
app.register_blueprint(userClass)
app.register_blueprint(commentsClass)
app.register_blueprint(visitedClass)
app.register_blueprint(locationClass)
app.register_blueprint(trackingClass)
app.register_blueprint(twitter_ratingsClass)
app.register_blueprint(favoritesClass)
app.register_blueprint(recommendationsClass)
app.register_blueprint(friendsClass)

#Host 0.0.0.0 permite a cualquier máquina local interaccionar con el Flask

app.run(host="0.0.0.0", port=5000, debug=True, threaded=True)
```

Figura 6.65: Contenido principal del script backend.py.

El diseño del servicio web se compone principalmente de 4 partes:

- backend.py (ver Figura 6.65): Es el script que enlaza todos los compo-

nentes del servidor. Se encarga de unificar todas las clases que contienen las rutas que invocan peticiones HTTP. Cada clase está relacionada con su tabla en la base de datos. También se inicializa el servicio web en un puerto específico y en local, por lo que no tiene conexión externa y no podrá funcionar de forma online.

- app.py: Se encarga de configurar la conexión con la base de datos, con la librería flask-sqlAlchemy [36]. Aquí se configura la URI para definir el acceso a la base de datos y si fuera necesario, también poder definir un usuario para acceder identificado dentro de MySQL.

- modules.py: Es el script donde se definen todas las tablas y modelos que se relacionan con las tablas dentro de la base de datos de MySQL. Flask-sqlAlchemy [20] define dichos modelos para tener claro cómo es la estructura de la base de datos que se conecta con el servicio web y así facilitar toda manipulación de datos como por ejemplo las inserciones y selecciones.

- El resto de las carpetas almacenan las urls que realizan las llamadas HTTP para devolver en formato JSON todo lo que la aplicación necesite de la base de datos. También contienen funciones auxiliares para facilitar y ahorrar código, y enlazar funciones de otros modelos para recoger información de diferentes tablas. Por ejemplo, para llamadas que necesiten toda la información sobre un lugar, necesitará también un listado de comentarios referidos a ese lugar.

app.py

```
from flask_sqlalchemy import SQLAlchemy
from flask import Flask

app = Flask(__name__)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:@localhost/TFG'
```

Figura 6.66: Configuración con la conexión a la base de datos.

A continuación se explica el desarrollo del programa principal que inicia el servicio web.

Inicialmente se configura la conexión de la base de datos con el script .ªpp.py"(ver Figura 6.66) y se importan todas las clases que contienen las rutas definidas para las peticiones HTTP que realiza la aplicación para poder recoger toda la información deseada en formato JSON. Finalmente con la función run(), se ejecuta el inicio del servicio web.

```python
modules.py
from flask_sqlalchemy import SQLAlchemy
from app import app

sqlAlchemy = SQLAlchemy(app)

########################################   Clases ########################################

class user(sqlAlchemy.Model):
    __tablename__ = 'user'
    id_user = sqlAlchemy.Column(sqlAlchemy.Integer(), primary_key = True)
    nickname = sqlAlchemy.Column(sqlAlchemy.String(255))
    name = sqlAlchemy.Column(sqlAlchemy.String(255))
    surname = sqlAlchemy.Column(sqlAlchemy.String(255))
    email = sqlAlchemy.Column(sqlAlchemy.String(255))
    password = sqlAlchemy.Column(sqlAlchemy.String(255))
    gender = sqlAlchemy.Column(sqlAlchemy.String(50))
    birth_date = sqlAlchemy.Column(sqlAlchemy.DateTime)
    city = sqlAlchemy.Column(sqlAlchemy.String(255), default="Madrid")
    rol = sqlAlchemy.Column(sqlAlchemy.String(255), default="user")
    profile_image = sqlAlchemy.Column(sqlAlchemy.String(255), nullable=True)

class location(sqlAlchemy.Model):
    __tablename__ = 'location'
    id_location = sqlAlchemy.Column(sqlAlchemy.Integer(), primary_key = True)
    name = sqlAlchemy.Column(sqlAlchemy.String(255))
    description = sqlAlchemy.Column(sqlAlchemy.String(1500))
    coordinate_latitude = sqlAlchemy.Column(sqlAlchemy.Float())
    coordinate_longitude = sqlAlchemy.Column(sqlAlchemy.Float())
    type_of_place = sqlAlchemy.Column(sqlAlchemy.Integer())
    city = sqlAlchemy.Column(sqlAlchemy.String(255), default="Madrid")
    road_class = sqlAlchemy.Column(sqlAlchemy.String(255))
    road_name = sqlAlchemy.Column(sqlAlchemy.String(255))
    road_number = sqlAlchemy.Column(sqlAlchemy.String(255))
    zipcode = sqlAlchemy.Column(sqlAlchemy.Integer())
    affluence = sqlAlchemy.Column(sqlAlchemy.String(255))
```

Figura 6.67: Estructuras de clases declaradas en modules.py.

El script "modules.py" es fundamental para que Flask entienda todas las clases y sus campos, y poder manipular los datos de la base de datos sin errores. Como se puede observar en la Figura 6.67, se puede ver 2 modelos que corresponden a las tablas "user" y "location" de la base de datos relacional MySQL. Es completamente necesario llamar de la misma forma el campo __tablename__ como la tabla en la base de datos al que se refiere. También es importante declarar las mismas columnas y en caso de ser String, definir el mismo tamaño.

El resto del contenido del directorio está dividido por carpetas, 1 por cada tabla relacionada en la base de datos para organizar toda la distribución de peticiones HTTP. Para este ejemplo se mostrará el contenido de la carpeta "location" (ver figura 6.68):

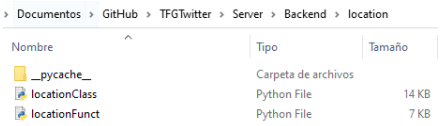


Figura 6.68: Scripts que contienen las peticiones HTTP y funciones auxiliares.

- NameTableClass.py: Esta clase contiene todas las peticiones HTTP que manipulan y muestran los datos de sus respectivas tablas en la base

de datos. Contiene también las funciones CRUD para crear, borrar, seleccionar y modificar cualquier dato que solicite la aplicación.

- NameTableFunct.py: Es un script que contiene funciones auxiliares para dar apoyo, evitar la repetición de código y mejorar la lectura del código de las peticiones HTTP.

Para factorizar los componentes del servicio web, se ha utilizado Blueprints, un objeto definido por Flask que sirve para simplificar y organizar el funcionamiento completo del servidor, y crear un esquema centralizado de los componentes. Se puede observar en la Figura 6.65 cómo se unifica todos los objetos blueprints en el script principal "backend.py".

A continuación, se explicará los diferentes métodos HTTP:

HTTP Verb	CRUD	Entire Collection (e.g. /customers)
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.

Figura 6.69: Los diferentes métodos HTTP.

Las peticiones HTTP contienen siempre un método. Para cada verbo definido en un CRUD, se deben utilizar métodos HTTP diferentes, como podéis observar en la Figura 6.69:

- El método POST es el más usado para generar nuevos recursos. En el caso de resultar una llamada exitosa. Habitualmente, necesitan parámetros para crear los recursos nuevos.
- GET se utiliza mayoritariamente para obtener recursos.
- PUT/PATCH se utiliza para actualizar o modificar los recursos dentro de la base de datos. También es habitual enviar parámetros dentro de la petición.
- DELETE es el método necesario para realizar una eliminación de algún recurso de la base de datos.

Todas las peticiones devolverán un estado que define el resultado de la petición HTTP. En total hay 5 familias de estados, cada uno con una cabecera numérica diferente:

- 1xx: Este estado sirve para informar al desarrollador de que se está procesando la petición.
- 2xx: Son mensajes de éxito. Por ejemplo el más habitual → 200 OK.
- 3xx: Es el estado que informa al desarrollador que la url ha sufrido una redirección.
- 4xx: Envían mensajes de errores al cliente sobre permisos o errores de sintaxis.
- 5xx Es el peor estado posible, ya que significa que el servidor ha fallado por causas independientes del cliente. Por ejemplo → 500 Internal Server Error.

A continuación se explica las peticiones HTTP más usadas por la aplicación Android.

```
@locationClass.route('/location/listLocations', methods=['GET', 'POST'])
def listLocations():
    json_data = request.get_json()
    page, quant, user, search = LocationFunct.initParameters(json_data)
    try:
        if(LocationFunct.checkPagination(page, quant, search) is False):
            return jsonify(exito = "true", list = [])

        places = modules.sqlAlchemy.session
                .query(modules.location)
                .outerjoin(modules.comments, modules.location.name == modules.comments.location)
                .filter(modules.location.name.like(search))
                .group_by(modules.location.name)
                .order_by(modules.sqlAlchemy.func.avg(modules.comments.rate).desc())
                .paginate(per_page=quant, page=page)

        if(places is not None):
            all_items = places.items
            lista = []
            for place in all_items:
                obj = LocationFunct.completeList(place, user)
                lista.append(obj)
            print("success")
            return jsonify(
                    exito = "true",
                    list = lista)

        print("failure")
        return jsonify(exito = "false")
    except Exception as e:
        print("Error: ", repr(e))
        return jsonify(exito = "false")
```

Figura 6.70: Petición HTTP que lista los lugares según las valoraciones.

En la Figura 6.70, se muestra la primera llamada que realiza la aplicación Android en el momento que se inicia, se trata de la petición que devuelve el listado de los lugares mejor valorados por los usuarios de la aplicación Android.

La petición necesita que la aplicación envíe al servicio web 4 parámetros, el número de página, la cantidad de lugares (por defecto son 3 lugares), el usuario cliente que ha solicitado la respuesta (en caso de no estar registrado es un anónimo), y el texto de la búsqueda integrada en la aplicación en caso

de que el usuario haya hecho uso del funcionamiento de búsqueda por texto o por voz.

El contenido de la función, consiste en crear una query con funcionalidades de la API de flask-sqlAlchemy para poder realizar una búsqueda de los lugares que se están buscando, ordenado por la valoración del lugar.

Finalmente, se crea una lista con los resultados obtenidos por la query, y se envían en formato JSON a la aplicación Android.

```
def completeList(place, user):
    visited = VisitedFunct.isVisited(place.name, user)
    n_comments = CommentFunct.numberOfComments(place.name)
    imageList = listImages(place.name)
    avgRate = CommentFunct.averageRate(place.name)
    favorite = FavoritesFunct.isFavorite(user, place)
    obj = {"name" : place.name,
    "description":place.description,
    "coordinate_latitude":place.coordinate_latitude,
    "coordinate_longitude":place.coordinate_longitude,
    "type_of_place":maptIntToCategory(place.type_of_place),
    "city":place.city,
    "road_class":place.road_class,
    "road_name":place.road_name,
    "road_number":place.road_number,
    "zipcode":place.zipcode,
    "affluence":place.affluence,
    "imageList" : imageList,
    "rate" : avgRate,
    "n_comments" : n_comments,
    "favorite" : favorite,
    "visited" : visited}
    return obj
```

Figura 6.71: Función auxiliar que crea la respuesta a la petición HTTP.

Para poder crear la lista que se envía a la aplicación Android, (ver en la Figura 6.71), se debe crear un diccionario con la información completa de un lugar. Los datos que se devuelven es la lista de imágenes, datos del lugar, comentarios, etc.

```
#Devolver una lista de 30 lugares más próximos
@locationClass.route('/location/listByProximity', methods=['GET', 'POST'])
def listByProximity():
    json_data = request.get_json()
    userLatitude, userLongitude, radius, nPlaces, user, search = LocationFunct.initParametersProximity(json_data)
    try:
        user_coords = (userLatitude, userLongitude)
        places = modules.location.query.filter(modules.location.name.like(search)).all()
        lista = []
        if(places is not None):
            for place in places:
                place_coords = (place.coordinate_latitude, place.coordinate_longitude)
                #Distancia calculada entre el usuario y el lugar en METROS
                distance = geodesic(user_coords, place_coords).meters
                if(distance <= radius): #Descartamos los lugares que no estén en el radio
                    obj = LocationFunct.completeList(place, user)
                    obj["distance"] = distance
                    lista.append(obj)
            lista.sort(key=dts)
            print("success")
            return jsonify(
                    exito = "true",
                    #Devuelve nPlaces o lo que haya disponible
                    list = lista[0:nPlaces] if nPlaces <= len(lista) else lista[0:len(lista)])
        return jsonify(exito = "false")
    except Exception as e:
        print("Error: ", repr(e))
        return jsonify(exito = "false")
```

Figura 6.72: Petición HTTP que lista los lugares por proximidad.

La petición HTTP que se muestra en la Figura 6.72 se encarga de enviar

los 30 lugares más próximos al usuario que está utilizando la aplicación Android. Para ello, se necesitan principalmente las coordenadas del usuario, y el nombre del usuario. Para calcular las distancias entre el usuario y los lugares, se itera cada lugar registrado dentro de la base de datos, y se utiliza la biblioteca geopy [29] que, dado 2 coordenadas, te calcula la distancia en metros o en kilómetros de ellas.

Tras realizar el cálculo de las distancias, se ordenan los lugares por la distancia, y se recogen únicamente los 30 primeros. Sin embargo, para reconocer un lugar próximo al usuario, es necesario que la distancia entre estos dos puntos, no sea mayor que 2 kilómetros.

```
#Servirá en el momento de crear una cuenta nueva o para modificar una contraseña
def passwordCipher(password):
    #Generamos la salt aleatoria
    #Es necesario tenerlo como bytes para hacer el hash
    hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
    return hashed

def passwordVerify(password, pwdCipher): #Comprueba si la contraseña es correcta
    if (bcrypt.checkpw(password.encode(), pwdCipher.encode()) is False):
        return False
    return True


#Login
@userClass.route('/login/', methods=['GET', 'POST'])
def login():
    json_data = request.get_json()
    nickname = json_data["nickname"]
    password = json_data["password"]

    userQuery = modules.user.query.filter_by(nickname = nickname).first()

    if (userQuery is not None):
        if (passwordVerify(password, userQuery.password) is True):
            print("success")
            return UserFunct.jsonifiedList(userQuery, password)
        else:
          print("Contraseña incorrecta")
    else:
      print("No existe el usuario")
    return jsonify(exito = "false")
```

Figura 6.73: Petición HTTP que verifica el inicio de sesión de un usuario.

La petición HTTP que se muestra en la Figura 6.73, sirve para enviar a la aplicación Android una respuesta del intento de incio de sesión por parte de un usuario.

La aplicación Android debe de enviar 2 parámetros acerca del usuario, el apodo del usuario y la contraseña proporcionada. Las contraseñas que están almacenadas dentro de la base de datos están cifradas con el algoritmo Blowfish gracias a la librería Bcrypt [24] de Python.

Bcrypt es capaz de verificar si dos contraseñas son iguales con la codificación en base 64 de estos dos. En caso de que el usuario y la contraseña coincidan con el usuario original, mandará un mensaje en formato JSON de que puede iniciar sesión.

```python
#Modificar Usuario
@userClass.route('/modifyUser/', methods=['PUT'])
def modifyUser():
    json_data = request.get_json()
    nickname = json_data["nickname"]
    name = json_data["name"]
    surname = json_data["surname"]
    email = json_data["email"]
    password = json_data["password"]
    gender = json_data["gender"]
    birth_date = json_data["birth_date"]
    b = datetime.datetime.strptime(birth_date, '%Y-%m-%d')
    profile_image = json_data["profile_image"]
    pwdCipher = passwordCipher(password)
    try:
        modifiedUser = modules.user.query.filter_by(nickname=nickname).first()
        modifiedUser.nickname = nickname
        modifiedUser.name = name
        modifiedUser.surname = surname
        modifiedUser.email = email
        modifiedUser.password = pwdCipher
        modifiedUser.gender = gender
        modifiedUser.birth_date = b
        if(profile_image != ""):
            if("http" not in profile_image):
                image = UserFunct.decode64Img(profile_image)
                url = UserFunct.uploadImg(image)
                UserFunct.delImgTemp(image)
                modifiedUser.profile_image = url
            else:
                modifiedUser.profile_image = profile_image
            modules.sqlAlchemy.session.commit()
        return UserFunct.jsonifiedList(modifiedUser, password)
    except Exception as e:
        print("Error modificando usuarios:", repr(e))
        return jsonify(exito = "false")
```

Figura 6.74: Petición HTTP para la modificación de los datos de un usuario.

La petición HTTP que se muestra en la figura 6.74 sirve para modificar el perfil del usuario. Observad que el método HTTP es diferente (PUT) al resto ya que se trata de una modificación y no de una lectura de recursos.

Durante una modificación de un perfil de usuario, es posible cambiar de foto de perfil.

Las imágenes que se almacenan en la base de datos están en formato url para no exceder más de lo necesario el tamaño de la base de datos. Como la imagen que se modificará no está en formato url, se ha hecho uso de la API de Imgur, para recoger la imagen y subirlo a la API de Imgur.

```python
def decode64Img(codedImg): #i para enumerar todas las imágenes
  image = base64.b64decode(str(codedImg))
  img = Image.open(io.BytesIO(image))
  img.save("perfil.png", "png")
  return "perfil.png"

def uploadImg(imgTemp):
  #ID público de mi cuenta de IMGUR
  CLIENT_ID = "9447315b37b3ece"
  im = pyimgur.Imgur(CLIENT_ID)
  uploaded_image = im.upload_image(imgTemp, title="TFG Madrid Places")
  return uploaded_image.link

def delImgTemp(imgTemp):
  try:
    os.remove(imgTemp)

  except Exception as e:
    print("Error eliminando la imagen temporal: ", repr(e))
```

Figura 6.75: Listado de funciones para el uso de la API de Imgur.

Para el procedimiento de la subida a la API de Imgur [35], es necesario codificar la imagen en base64 y almacenarla en formato PNG, usando la función decode64Img().

A continuación se recoge la imagen generada utilizando la función uploadImg() y se sube con el ID del cliente que vinculado a una cuenta de la página oficial de la API de Imgur. El ID del cliente es completamente público por lo que no hay problema en implementarlo dentro del código.

Para no almacenar imágenes temporales dentro del directorio, se boraan con la función delImgTemp().

Todas estas funciones auxiliares están en el script de "userFunct.py", se puede observar el contenido en la Figura 6.75.

A continuación se explicará el almacenamiento de los puntos de seguimiento de los usuarios que acepten el uso de la ubicación de su dispositivo móvil.

```python
@trackingClass.route('/user/pointTracked', methods=['POST'])
def pointTracked():
    json_data = request.get_json()
    user = json_data["user"]
    latitude = json_data["latitude"]
    longitude = json_data["longitude"]

    createTrack = modules.tracking(user = user, latitude = latitude, longitude = longitude)

    try:
        tcQuery = modules.tracking.query.filter_by(user = user, latitude = latitude, longitude = longitude).first()
        if(tcQuery is None):
            modules.sqlAlchemy.session.add(createTrack)
            modules.sqlAlchemy.session.commit()
            return jsonify(
                        exito = "true",
                        user = createTrack.user,
                        latitude = createTrack.latitude,
                        longitude = createTrack.longitude,
                        passed = createTrack.passed)
        else:
            return jsonify(exito = "true")
    except Exception as e:
        print("Error insertando la nueva fila :", repr(e))
        return jsonify(exito = "false")
```

Figura 6.76: Petición HTTP para insertar el seguimiento del usuario.

La aplicación Android llama a la petición HTTP de la Figura 6.76 de forma automática, sin que el usuario necesite realizar alguna acción. Cada 5 segundos se almacena las coordenadas del dispositivo móvil para crear un seguimiento del usuario y así poder tener un mayor precisión a la hora de mostrar los lugares más cercanos al usuario.

Para ello, la petición HTTP necesita 2 parámetros, las coordenadas y el apodo del usuario. Si se ha creado correctamente, envía un mensaje positivo a la aplicación para confirmar que se ha insertado correctamente en la base de datos.

```
#Tarea automatica que elimina el tracking cada 1 hora

scheduler = BackgroundScheduler()
scheduler.add_job(func=TrackingFunct.deleteTrack, trigger="interval", seconds=3600)
scheduler.start()

# Shut down the scheduler when exiting the app
atexit.register(lambda: scheduler.shutdown())
```

Figura 6.77: Función automática con Scheduler para borrar el seguimiento del usuario.

Tras un periodo de tiempo extenso, no es necesario que sigan en la base de datos. Por ende, se ha creado una función automática que eliminará todos los puntos de seguimientos de todos los usuarios que hayan estado 1 hora o más, almacenados dentro de la base de datos.

Esta función, como se puede observar en la Figura 6.77 se ejecuta en intervalos de 3600 segundos. Para la creación de la función, se ha hecho uso de la librería Apscheduler [32] que sirve para programar periódicamente la ejecución de un trozo de código dentro de un script en Python.

6.5 Acceso al repositorio Github del proyecto

En esta sección se muestra el enlace al repositorio Github del proyecto, donde se puede ver públicamente el código tanto de la aplicación Android como la implementación del servicio API REST explicado en este capítulo.

https://github.com/GeoDNSO/MadridPlaces

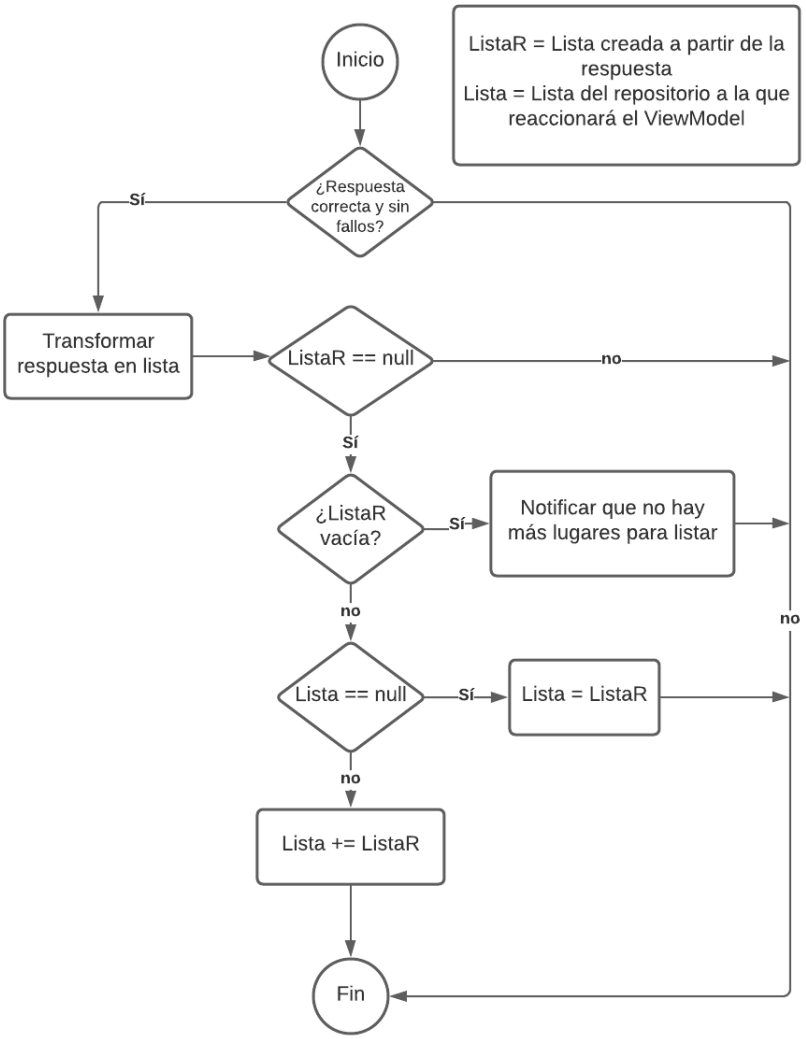


Figura 6.18: Diagrama de flujo de actualización de lista de lugares.

Capítulo 7

Evaluación

En este apartado se exponen los resultados de una evaluación realizada a una serie de usuarios para verificar y valorar la usabilidad de la aplicación.

Se ha realizado una evaluación de la usabilidad de la aplicación en una muestra formada por siete personas de edades comprendidas entre 18 y 25 años, de las cuáles 2 eran 2 mujeres y 5 hombres.

Para la realización de la evaluación se ha pedido a los usuarios que realicen una serie de operaciones para analizar la soltura con la que se desenvuelven en el uso de la aplicación.

Las operaciones que se pidieron realizar a los usuarios fueron las siguientes:

1. Registro en la aplicación
2. Inicio de sesión
3. Buscar lugares por algún filtro (valoración, cercania, categoria, etc.)
4. Marcar lugar como favorito
5. Leer la descripción de un lugar
6. Escribir un comentario/reseña sobre un lugar
7. Acceder a la lista de lugares favoritos
8. Modificar un lugar
9. Borrar un lugar
10. Crear un lugar
11. Añadir un amigo

12. Recomendar un lugar a un amigo

13. Aceptar una recomendación

14. Marcar un lugar como visitado

15. Gestionar la lista de usuarios de la aplicación con acceso de administración

16. Cambiar idioma de la aplicación

Una vez finalizada la prueba en la aplicación, los usuarios valorarón la usabilidad de la aplicación mediante un formulario. Para ello el formulario se compone de varias afirmaciones con multiples respuestas del 1 al 5 para valorar la conformidad del usuario con las actividades realizadas (siendo 1 = totalmente en desacuerdo y 5 = totalmente de acuerdo).

Los resultados obtenidos de dicho test han sido los siguientes:

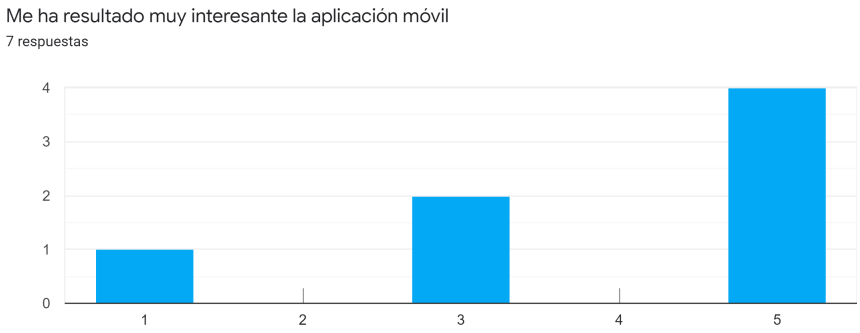


Figura 7.1: Pregunta 1.

Con estos resultados (ver Figura 7.1) se puede decir que la aplicación ha resultado algo interesante. A cuatro de siete personas les ha parecido interesante la aplicación. Hay dos personas a las que no les interesa pero tampoco les disgusta y a una persona que no le ha parecido interesante.

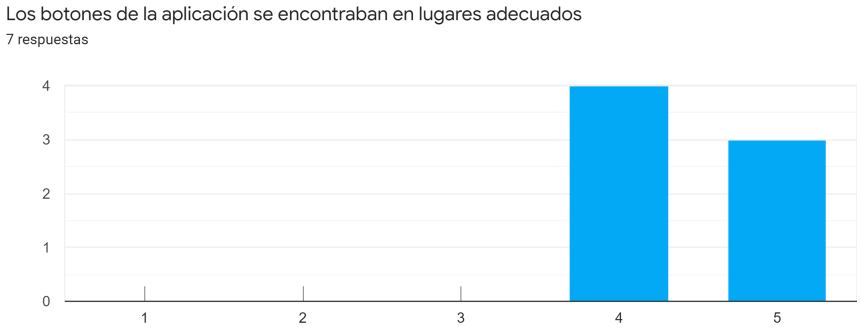


Figura 7.2: Pregunta 2.

Con los resultados de esta pregunta (ver Figura 7.2) se puede afirmar que los botones de la interfaz están situados de tal manera que resulta cómodo e intuitivo a la hora de navegar por la aplicación. A 3 de las siete personas les ha parecido perfecto el lugar de los botones y al resto de las personas evaluadas les ha parecido adecuado o correcto la posición de los botones.

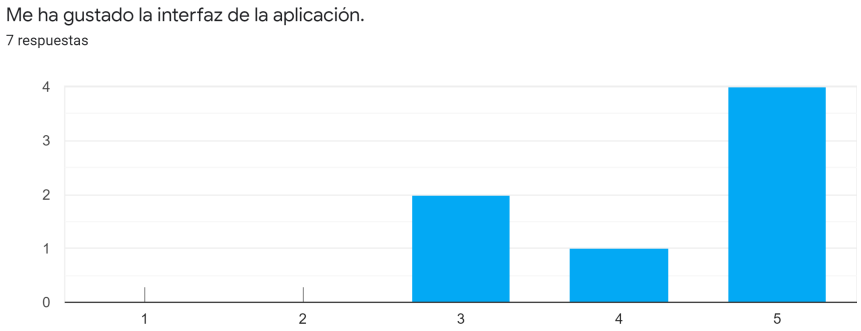


Figura 7.3: Pregunta 3.

Viendo la gráfica (ver Figura 7.3) podemos observar que la interfaz es más o menos la correcta donde cuatro de las siete personas piensan que la interfaz está bien y el resto no le parece del todo correcto, pero tampoco incorrecto. Por lo que se podría mejorar algo la interfaz de la aplicación.

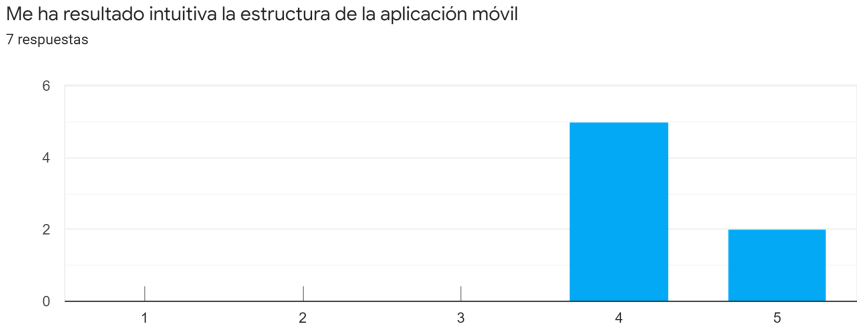


Figura 7.4: Pregunta 4.

Con estos resultados (ver Figura 7.4) podemos afirmar que la aplicación presenta una estructura amigable para el usuario aunque se podría perfeccionar. También podemos sacar en conclusión que es sencillo diferenciar dónde debe estar cada función.

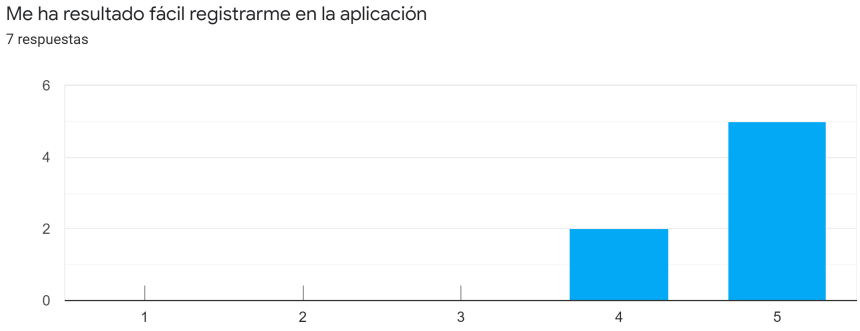


Figura 7.5: Pregunta 5.

Viendo la gráfica (ver Figura 7.5) podemos sacar en conclusión que el registro en la aplicación es accesible e intuitivo para los usuarios.

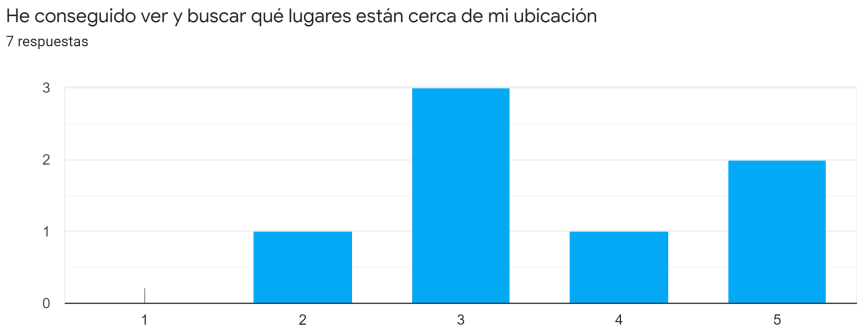


Figura 7.6: Pregunta 6.

En este caso (ver Figura 7.6) podemos observar que hay diferencia de opiniones entre los usuarios, ya que los usuarios esperaban que se encontrara en otra pestaña distinta en el TabLayout a la de lugares populares, pero no todos lo vieron al instante.

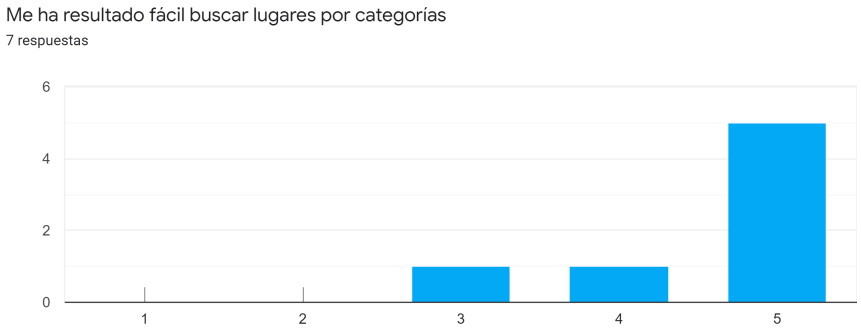


Figura 7.7: Pregunta 7.

Con estos resultados (ver Figura 7.7) podemos decir que los usuarios no tienen problemas al buscar los lugares por categorías, el único problema es que la pestaña está más oculta y hay que deslizar el menú para verlo. Los mejores resultados pueden deberse a que ya habían visto el menú de lugares cercanos anteriormente.

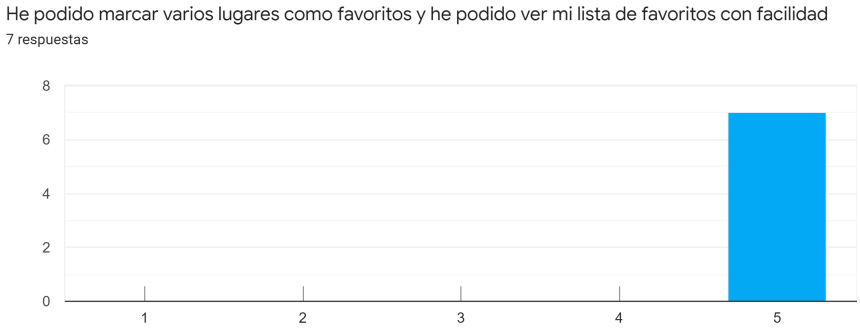


Figura 7.8: Pregunta 8.

Los usuarios han encontrado fácilmente cómo añadir un lugar a favoritos y ver la lista por la simplicidad de los iconos (ver Figura 7.8).

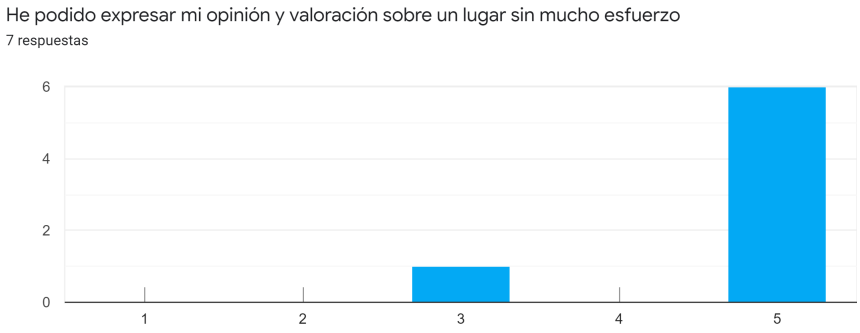


Figura 7.9: Pregunta 9.

La mayoría de los usuarios encuentran bastante fácil escribir reseñas sobre lugares salvo uno, lo cual indica que es intuitivo para casi todos los usuarios (ver Figura 7.9).

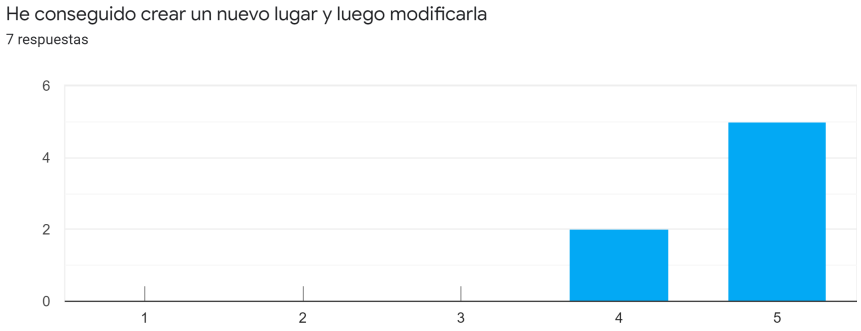


Figura 7.10: Pregunta 10.

Los usuarios encuentran fácil la creación de lugares nuevos y modificación, ya que el formulario contiene unos pasos fáciles e intuitivos (ver Figura 7.10).

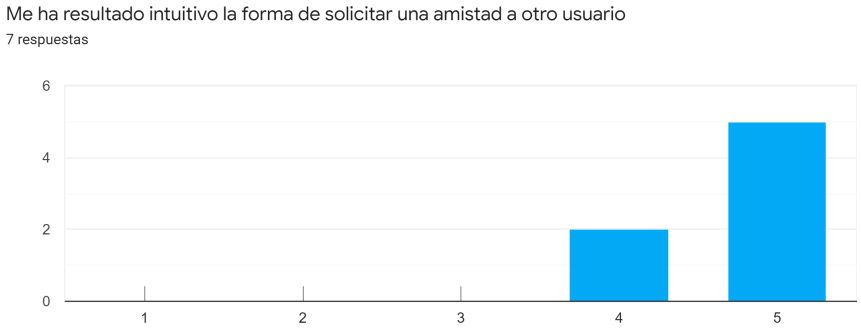


Figura 7.11: Pregunta 11.

Los resultados (ver Figura 7.11) en este caso indican que las peticiones de amistad no suponen ningún problema para los usuarios porque está implementado de manera simple y de manera típica para el usuario promedio.

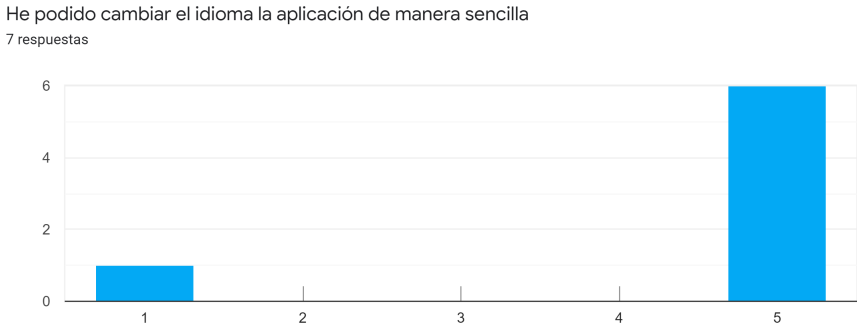


Figura 7.12: Pregunta 12.

Viendo los resultados (ver Figura 7.12), podemos destacar que la gran mayoría de los usuarios pudieron cambiar el idioma sin problemas ya que es una opción que se encuentra típicamente en el apartado de ajustes.

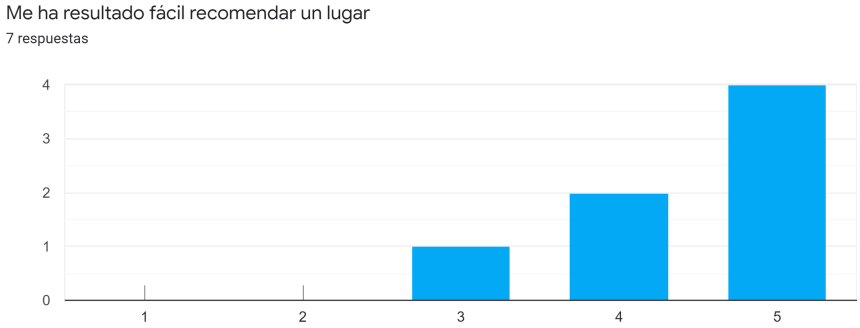


Figura 7.13: Pregunta 13.

Generalmente los usuarios no encuentran de manera complicada la recomendación de un lugar, aunque no es muy intuitivo ya que no es típico que se encuentre en un apartado genérico como en nuestra aplicación (ver Figura 7.13).

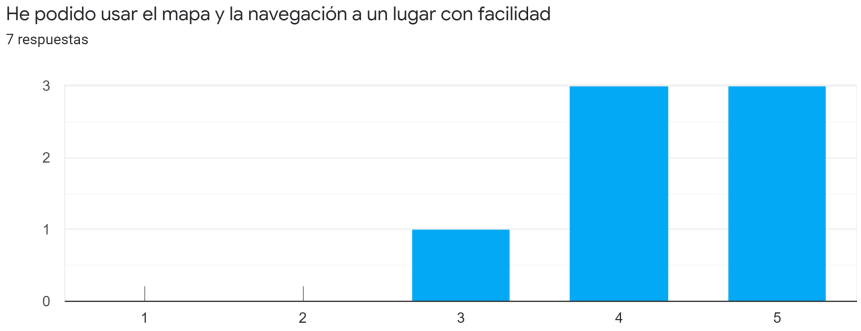


Figura 7.14: Pregunta 14.

Generalmente el uso del mapa no ha dado demasiados problemas a nuestros usuarios, aunque el icono del mapa no destaca mucho para los usuarios (ver Figura 7.14).

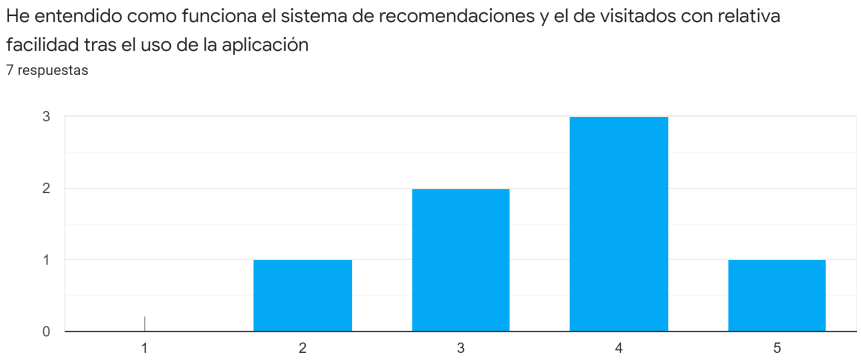


Figura 7.15: Pregunta 15.

Se puede observar que los usuarios han tenido diversos problemas, sobretodo con los lugares pendientes por visitar porque se encuentra en el apartado de visitados, por lo que nos recomiendan realizar un tercer apartado para esta categoría para una mayor claridad (ver Figura 7.15).

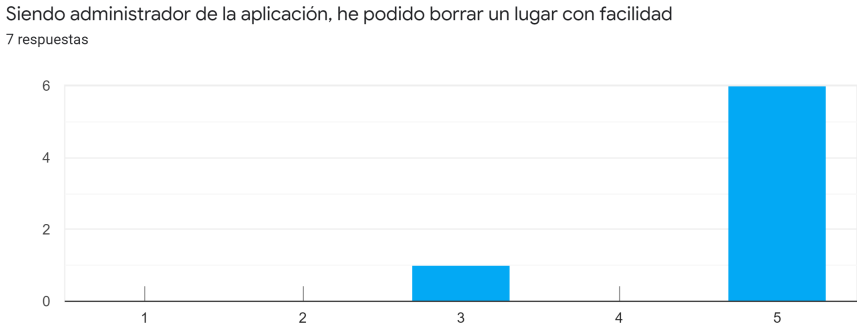


Figura 7.16: Pregunta 16.

Con estos resultados (ver Figura 7.16) sacamos en conclusión que la eliminación de lugares no es problemática para los usuarios ya que es muy visible el icono de eliminar un lugar, por lo que es intuitivo.

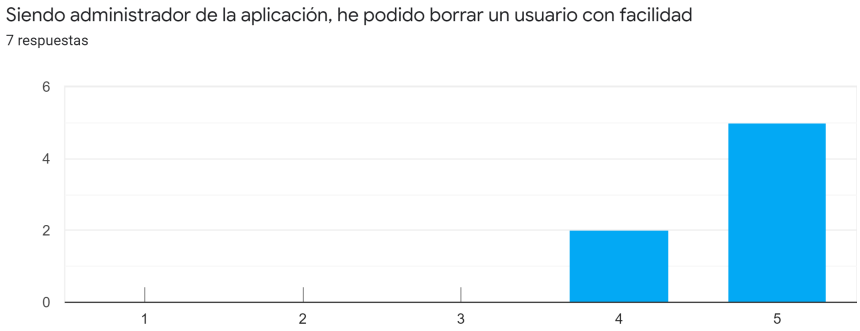


Figura 7.17: Pregunta 17.

Se puede concluir que nuestros usuarios han podido eliminar a usuarios de la aplicación de manera sencilla porque los pasos son los que un usuario normal espera en una aplicación (ver Figura 7.17).

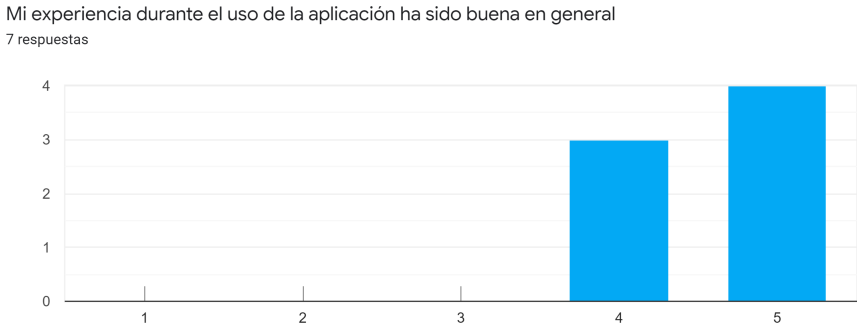


Figura 7.18: Pregunta 18.

En general, se puede decir que a todos los usuarios les ha agradado utilizar nuestra aplicación con algunos matices como la pestaña de lugares por visitar (ver Figura 7.18).

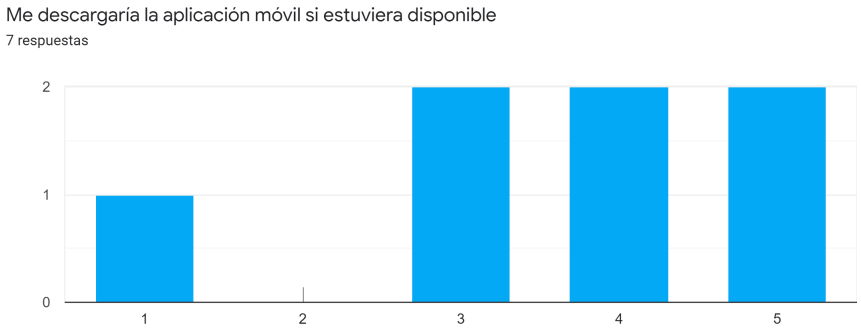


Figura 7.19: Pregunta 19.

En general, los usuarios, salvo una excepción, piensan en descargar la aplicación si estuviera disponible ya que les resulta útil para encontrar lugares en la ciudad que visitar (ver Figura 7.19).

Al final del cuestionario, se deja un apartado para que los usuarios puedan dar un feedback de manera que se pueda mejorar la aplicación en el futuro, estas han sido sus sugerencias:

- Mejoras mínimas en la interfaz gráfica a la hora de aceptar recomendaciones.
- Algún lugar que había no existe y me dio falsas ilusiones

- Sugerencias: Mi primera sugerencia sería dotar a ciertos individuos de mayor inteligencia pues habrá personas que no le serán tan intuitivo este programa. También, podrías dejar el programa así y, crear a la vez, una forma más fácil para los niños que se llamaría Madrid places kids.

- Si, haber puesto las recomendaciones aceptadas en el mismo apartado que las recibidas y las mías

- Poner algún comentario de que está visitado cuando se marca

- A la hora de hacer una recomendación lo más intuitivo como usuario ha sido entrar primero en la pestaña de recomendaciones, antes que entrar en un lugar y recomendarlo, opino que no hay que cambiarlo, sino que meter un botón de recomendar en vez de tener que buscarlo en los tres puntitos.

- Varias sugerencias: Cuando he creado una valoración, no me salía a continuación. Además si había otras valoraciones tampoco me salían, y ver las valoraciones de otros usuarios me parece importante. Al seleccionar el apartado de lugares cercanos, han tardado demasiado en aparecer. Al crear un lugar nuevo en el mapa no sabía cómo seleccionarlo, ya que no se selecciona pulsando, sino arrastrando el dedo en la pantalla. Me ha costado encontrar la lista de pendientes por visitar, ya que no es muy intuitivo si el icono se llama "visitado".

Las mejoras podrían ser las siguientes:

- Realizar una interfaz más intuitiva para el usuario a la hora de realizar recomendaciones y aceptarlas

- Crear un filtro a la hora de crear lugares para que unas personas se dediquen a confirmar su existencia.

- En general, mejorar el apartado de las recomendaciones para que sea más intuitivo.

Capítulo 8

Conclusiones y trabajo futuro

8.1 Conclusiones

Se ha implementado una aplicación orientada hacia el turismo que presenta como principales funcionalidades: la búsqueda de lugares ya sea por valoración, cercanía, popularidad en redes sociales, tipo de lugar, texto o voz. También, con el propósito de promover el turismo y la colaboración entre usuarios, estos pueden marcar lugares como favoritos, añadirlos a un historial de lugares visitados, añadir amigos o recomendar lugares a otros usuarios. Asimismo, es posible cambiar el idioma de la aplicación con lo que usuarios extranjeros pueden utilizar la aplicación sin tener en cuenta la barrera lingüística.

La aplicación desarrollada presenta como ventajas que es accesible e intuitiva gracias a la distribución de los elementos en la vista y a opciones como la selección de lenguaje. Además, permite una búsqueda rápida de lugares con distintos filtros y la colaboración entre usuarios con el sistema de recomendaciones y reseñas.

A pesar de todas estas ventajas y objetivos conseguidos, la aplicación MadridPlaces no es perfecta, ya que no proporciona gran cantidad de idiomas, y la información de los distintos tipos de lugares tienen un mismo formato. También es necesario tener en cuenta que la base de datos de la aplicación no posee todos los lugares de Madrid y que además el ámbito se limita a Madrid.

8.2 Trabajo futuro

- Añadir un mapa de lugares cercanos
  - Una idea para hacer más visual los datos de nuestra aplicación es que en vez de mostrar solo un listado de los lugares cercanos, se pueda mostrar un mapa utilizando mapbox con los lugares en unos kilómetros a la redonda, donde se muestre el tipo de lugar y su puntuación según los usuarios.
- Actualizar tecnologías
  - En el uso del patrón MVVM, se utilizó las clases que ofrece Android para el control del ciclo de vida de la aplicación y los LiveData. A medida que se investigó más sobre este patrón se descubrió que RxJava es una librería que permitiría implementar la observación del repositorio de una manera más sencilla. RxJava se basa en objetos observables y objetos que observan a estos observables de tal manera que se pueda reaccionar a sus cambios.
    La implementación de RxJava supondría un cambio en la arquitectura que llevaría algo de tiempo, pero permitiría mejorar la limpieza y organización del código.
    La principal diferencia entre RxJava y la estructura de switchMap del proyecto es que RxJava permite controlar de una mejor forma las operaciones cuando ocurre algún error sin necesidad de códigos de estado.
- Gestión de datos
  - Actualmente la aplicación realiza una petición HTTP al servidor cada vez que se navega a un fragmento, por lo que esto supone un gasto de datos en el dispositivo del usuario. Para solucionar este problema, sería necesario usar una base de datos SQLite en la que guardar los datos que se reciben de la API de tal manera que cuando el usuario vuelva a usar un fragmento se usen los datos guardados en caché y no malgastar datos.
    Para ello sería necesario utilizar Room que ofrece una abstracción sobre SQLite para manejar fácilmente los modelos de datos de la aplicación.
- Multilenguaje
  - Ahora mismo nuestra aplicación provee soporte en otros idiomas como demostración de la posibilidad de dar servicio a personas extranjeras que realizan turismo en Madrid. Estas traducciones se han realizado de manera automática gracias a la herramienta `https://asrt.gluege.boerde.de/` creada por `https:`

`//github.com/Ra-Na` y no son perfectas. Por lo tanto, sería necesario realizar un correcto repaso y traducción de los recursos de texto de la aplicación.

Además, en casos de idiomas como el árabe en el que la lectura se realiza de derecha a izquierda sería necesario la reorganización de las vistas para la comodidad de los usuarios.

- Servidor en la nube
  - Actualmente, el servidor se ejecuta en local en los ordenadores de cada uno. En un futuro sería necesario tener el servidor alojado en la nube para poder trabajar de manera más cómoda sin perder tiempo en actualizaciones de la Base de datos o del servidor en cada uno de los equipos y para poder simular mejor un entorno de producción en el que hacer pruebas más realistas.
- Interfaz personalizada para los tipos de lugares
  - Actualmente nuestra aplicación provee datos generales para todos los lugares. Los más generales e importantes serían:
    * La **descripción** para informar al usuario de los servicios del lugar
    * **Dirección**, para permitir al usuario conocer la ubicación del lugar y poder llegar a él.
    * **Comentarios**, para poder ver la valoración de otras personas y poder decidir visitar o no el lugar.

    El siguiente paso para mejorar el servicio que ofrece nuestra aplicación a los usuarios sería mostrar más información de interés dependiendo del tipo de lugar que este viendo el usuario. Por ejemplo:
    * Si el lugar es un restaurante, poder mostrar información sobre la carta y posibles alérgenos de los alimentos que se sirven además del precio medio por persona.
    * Si el lugar es museo, mostrar horarios y tarifas de las visitas.

    Para lograr todo esto sería necesario buscar nuevas fuentes de datos o crear una aplicación web secundaria en la que los propietarios de estos establecimientos puedan añadir esta información con el incentivo de la publicidad y ayuda a futuros clientes. Para realizar todo esto sería necesario ponerse en contacto con estos propietarios.

    Además, sería necesario cambiar la estructura de la base de datos para cada tipo de lugar existente en el sistema.
- Ampliar el ámbito

  - Ahora mismo la aplicación solo está pensada para la ciudad de Madrid, pero podría ampliarse el ámbito a toda españa o incluso a nivel internacional. El principal problema que habría que solucionar al implementar esta mejora serían las fuentes donde obtener todos los datos necesarios con los que ofrecer un servicio actualizado a los usuarios.

- Fuentes de datos

  - Actualmente hay más de 5000 lugares registrados en la base de datos, pero es una cifra insuficiente si se quiere ofrecer una experiencia más completa a los usuarios que utilicen la aplicación. Se podría, por ejemplo, añadir las cadenas multinacionales de comida rápida como McDonald 's. Sin embargo, es complicado buscar fuentes de datos compatibles a la estructura de datos ya integrada en el proyecto. Otra mejora es utilizar otras APIs relacionadas a redes sociales para aumentar la opinión pública acerca de los lugares registrados en la base de datos.

- Elegir entre distintas rutas para ir a un lugar

  - Siguiendo con el trabajo realizado sobre mapbox, se podría dar la posibilidad de elegir la ruta a un destino de un usuario. Para ello, se podría dar la opción de escoger entre tres rutas desde el mapa por distintas calles y/o carreteras a gusto del usuario, y después comenzar la navegación al lugar por el camino indicado.

Chapter 8

Conclusions and future work

8.1 Conclusions

An application oriented towards tourism has been implemented that presents as main functionalities: the search for places either by rating, proximity, popularity in social networks, type of place, text or voice. Also, in order to promote tourism and collaboration between users, they can mark places as favorites, add them to a history of visited places, add friends or recommend places to other users. Also, it is possible to change the language of the application so that foreign users can use the application without taking into account the language barrier.

The developed application presents as advantages that it is accessible and intuitive thanks to the distribution of the elements in the view and options such as the language selection. In addition, it allows a quick search of places with different filters and the collaboration between users with the recommendation and review system.

Despite all these advantages and objectives achieved, the MadridPlaces application is not perfect, since it does not provide a large number of languages, and the information of the different place types has the same format. It is also necessary to bear in mind that the application's database does not have all the places in Madrid and that the scope is also limited to Madrid.

8.2 Future work

- Add a close places list
  - An idea to make the application data more visual is showing a map of close places using mapbox instead of a list in order to show the closest places in a determined area of a few kilometres. The map would show the place type and the user ratings.
- Update tecnologies
  - LiveData was used to make the lifecycle app control using the classes provided by Android using the MVVM pattern. After some further investigation it was discovered that RxJava library was an easier way to make this pattern. RxJava is based in the concept of observable objects and objects that observe this observables in order to react to their changes.

    The RxJava implementation would have an impact in the application architecture so it would take some time, but it would make the code cleaner and organized.

    The main difference between RxJava and SwitchMap architecture of the project is that the first allows to control the operations in a better way when some mistake happens without state code.
- Data management
  - Currently, the application makes an HTTP request to the server each time a fragment is browsed, so this is a waste of data on the user's device. To solve this problem, it would be necessary to use an SQLite database that would save the data received from the API in such a way that when the user reuses a fragment, the cached data is used and not wasting data.

    In order to do this, it would be necessary to use Room that offers an abstraction on SQLite to easily handle the application's data models.
- Multilanguage
  - Right now our application provides support in other languages as a demonstration of the possibility of providing service to foreign people who do tourism in Madrid. These translations have been done automatically thanks to the `https://asrt.gluege.boerde.de/` tool created by `https://github.com/Ra-Na` and they are not perfect. Therefore, it would be necessary to do a review and translation of the application's text resources.

In addition, in cases of languages such as Arabic in which the reading is done from right to left, it would be necessary to reorganize the views for the convenience of users.

- Cloud server
  - Currently, the server runs locally on each other's computers. In the future it would be necessary to have the server hosted in the cloud to be able to work more comfortably without wasting time on database or server updates on each of the computers and to make a production environment simulation to do more realistic tests.
- Custom interface for place types
  - Currently our application provides general data for all places. The most general and important would be:
    * La **description** to report information to the user of the place services.
    * **Direction**, to allow the user to know the location of the place and to be able to reach it.
    * **Comments**, to be able to see the ratings of other people and to be able to decide whether or not to visit the place.

    The next step to improve the service that our application offers to users would be to show more information of interest depending on the type of place the user is viewing. For example:
    * If the place is a restaurant, to be able to show information about the menu and possible allergens of the foods that are served in addition to the average price per person.
    * If the place is a museum, show schedules and visit rates.

    To achieve all this it would be necessary to search for new data sources or create a secondary web application in which the owners of these establishments can add this information with the incentive of advertising and help future customers. To do all this it would be necessary to contact these owners.

    In addition, it would be necessary to change the database structure for each type of place in the system.
- Expand the scope
  - Ahora mismo la aplicación solo está pensada para la ciudad de Madrid, pero podría ampliarse el ámbito a toda españa o incluso a nivel internacional. El principal problema que habría que solucionar al implementar esta mejora serían las fuentes donde obtener

todos los datos necesarios con los que ofrecer un servicio actualizado a los usuarios.

- Fuentes de datos
  - There are currently more than 5000 places registered in the database, but it is an insufficient number if you want to offer a more complete experience to users who use the application. It could, for example, add the multinational fast food chains such as McDonald 's. However, it is difficult to find data sources compatible with the data structure already integrated in the project. Another improvement is to use other APIs related to social networks to increase public opinion about the places registered in the database.
- Choose between different routes to go to a place
  - Continuing with the work done on mapbox, it could be possible to choose the route to a destination of a user. For this, the option could be given to choose between three routes from the map through different streets and / or highways to suit the user, and then begin navigation to the place by the indicated path.

Capítulo 9

Aportaciones individuales

9.1 Daniel Nasim Santos Ouafki

Al principio el trabajo que hicimos todos fue el de investigación. La idea era crear una demo simple con una pantalla de registro que se pudiese conectar a una Base de Datos. Después de realizar cada uno las demos comparamos las tecnologías de Android que usamos para realizar el proceso de registro y la estructura de proyecto usada. Mi papel principal en esta parte fue la de ofrecer un menú y estructura basada en Fragmentos y ViewModels.

Una vez escogida la estructura y tecnologías básicas del proyecto mi papel fue principalmente:

- Definir la estructura en el proyecto Android
- Creación de los componentes Fragments, ViewModel para la vista de la aplicación y funciones del Repositorio; necesarios para conectarse a la API. Además de la investigación e implementación de las tecnologías necesarias para poder mejorar la interfaz gráfica de la aplicación.
- Dar soporte ante los fallos que ocurrían en el desarrollo del proyecto Android y dar soluciones a la implementación de nuevas funcionalidades para la aplicación.
- Revisión final del proyecto Android, refactorizando clases y funciones y arreglando bugs y errores.
- Participación en la creación de la memoria en Google Docs, principalmente en la redacción de implementación, el patrón MVVM y revisión de otros puntos.

En general, estoy contento con mi aportación al proyecto y el conocimiento adquirido. He aprendido mucho en cuanto a desarrollo Android y

los componentes y patrones necesarios para crear una aplicación moderna y escalable. Actualmente creo que tengo un nivel de conocimiento en Android suficiente para crear distintos tipos de aplicaciones y poder seguir investigando al respecto.

También he aprendido mucho sobre el funcionamiento y creación de APIs REST con la implementación en Python usando Flask de mi compañero Jin Wang.

9.2 Juan Antonio Escobar de los Ángeles

Primero de todo, me gustaría agradecer a mis compañeros el trabajo que han realizado a lo largo del curso. Me han dado mucho apoyo moral y me han ayudado siempre que lo he necesitado, además he trabajado muy bien con ellos y no hemos tenido problemas severos entre nosotros ya que ha habido buena comunicación entre nosotros a lo largo del año.

Al comenzar, mi trabajo principal fue la investigación sobre servidores, para elegir qué lenguaje de programación y qué framework deberíamos usar, tras lo que decidimos utilizar Flask en Python después de hacer diversas pruebas. También investigué sobre qué lenguaje de programación era mejor para desarrollar en Android, tras discutir si usar Kotlin o Java decidimos usar Java por comodidad ya que nos resultaba más conocido.

También participé activamente en la selección del ORM que se utilizaría para Flask para simplificar las consultas a la base de datos. Con Flask-SQLAlchemy pudimos realizar estas consultas sin problema.

Posteriormente, mi papel fue la implementación de un patrón que separase vista, control y modelos, para lo cual hice una demo del patrón MVVM después de investigar sobre cómo llevarlo a código, utilizando LiveData, para observar el modelo. A partir de ahí participé en la creación de Fragmentos, ViewModels y Repositorios de la aplicación, centrándome sobretodo en estos dos últimos. También participé en la creación del servidor para enviar y recibir peticiones del servidor de manera correcta modificando los datos en la base de datos.

Mi tarea principal cuando nos repartimos tareas sobre las funcionalidades, fue la investigación de Mapbox para implementar un mapa que no requiriese de pago. Utilizando Mapbox SDK se pudo realizar sin mucha dificultad, adaptando códigos de ejemplo de Mapbox a nuestra aplicación, ya que había funcionalidades que no era necesario rediseñar y estaban bien implementadas en los códigos de ejemplo. También realicé pruebas en un dispositivo móvil real debido a las limitaciones de Mapbox en los emuladores de Android Studio.

También he participado generalmente en la realización de la memoria en Google Docs y su transformación a Látex.

9.3 Jin Tao Peng Zhou

Al principio del curso, todos los integrantes del grupo hicimos una investigación general de cómo funcionaba Android Studio. Una vez terminado de investigar, cada integrante del grupo hizo una pequeña versión de lo que iba a ser las pantallas de registro y login.

Una vez hechas las pantallas, el grupo se reunió para ver cómo lo había hecho cada uno y se eligió una entre todas.

Una vez se eligieron las pantallas, se decidió utilizar el patrón MVVM y se repartió trabajo a cada integrante del grupo. En mi caso, me dediqué a hacer más vistas de la modularidad del usuario (ver sección 3.2.1 de la memoria) con sus respectivos ViewModel. Por ejemplo, hice las pantallas de administrador de usuarios y ver el perfil de otro usuario. Además, implementé algunas de las funcionalidades de otras vistas relacionadas al módulo de usuario.

Cuando se terminó de realizar el trabajo correspondiente de cada miembro, se unificó y se hicieron pruebas para ver que todo funcionaba correctamente. A este punto, mis compañeros Jin Wang y Juan Antonio Escobar ya habían implementado una base de lo que iba a ser el repositorio y la base de datos puediendo de esta manera hacer pruebas con la base de datos.

Más tarde, se repartió trabajo de la modularidad de lugares (ver seccion 3.2.2 de la memoria). En este caso me encargue de seguir haciendo más vistas como la vista para añadir un lugar y ayudar a mi compañero Daniel Nasim Santos con algunas otras vistas.

Por último, me encargué de realizar la búsqueda por texto, parte del sistema de amigos y ayudar un poco en la parte de la API de Mapbox.

A lo largo del proyecto me he dedicado a probar la aplicación y apuntar los errores que iba encontrando. En caso de que hubiera algun error fácil y rápido de arreglar, se solucionaba individualmente. En caso de que fuera más díficil, el equipo se juntaba para solucionar los problemas.

En la parte de la memoria me he encargado de realizar parte de la especificación de requisitos, parte de la implementación del sistema, parte de las pruebas a usuarios e interpretación del formulario que han realizado los usuarios que han probado la aplicación y hacer el manual de instalación junto al manual del usuario. Además todo el grupo se ha encargado de revisar la memoria.

En general me he sentido muy cómodo trabajando con mis compañeros y he aprendido mucho de ellos como por ejemplo Juan Antonio Escobar me ha ayudado a comprender cómo funcionaba Mapbox, Daniel Nasim Santos me ha ayudado a comprender cómo funcionaba el manejo de parcelables y Jin Wang me ha ayudado a comprender cómo funcionaba flask y el uso de la herramienta Advanced Rest para lanzar peticiones al servidor.

9.4 Jin Wang Xu

El primer punto realizado durante el desarrollo de la aplicación fue una fase de investigación individual para familiarizarnos con el funcionamiento de Android Studio y con la estructura del patrón MVVM. Para ello, realicé un proyecto simple con un sistema de login. Esta fase de investigación la realizamos todos los miembros del equipo y juntamos todas las conclusiones obtenidas a lo largo de esta fase inicial.

Tras la fase de investigación, nos pusimos mi compañero Juan Antonio y yo con la selección y el montaje del servicio web con API REST para la comunicación entre la aplicación Android y la base de datos MySQL. Para ello decidimos utilizar Flask como framework y Flask-sqlAlchemy para la conexión con la base de datos.

Paralelamente, me encargué también de realizar una búsqueda de datos acerca de lugares potencialmente útiles para mostrar en nuestra aplicación. También me dediqué a realizar una limpieza a los conjuntos de datos extraídos, y posteriormente aplicar un proceso ETL a todos los datasets con script en Python. Durante este proceso estuve investigando herramientas, librerías y tecnologías para que me ayudaran a realizar todo este proceso.

Una vez cargados todos los datos dentro de MySQL, me encargué de realizar todo el funcionamiento de API REST, como las creaciones de las peticiones HTTP para una correcta comunicación entre Cliente y Servidor.

Por ello, todos los códigos escritos en Python fueron supervisados y creados por mí, pero con ayuda de mis compañeros a la hora de solucionar problemas.

En la parte del desarrollo de la aplicación Android, mi aportación fue ayudar a crear funciones en los repositorios del proyecto, con indicaciones proporcionadas por mis compañeros.

También en la memoria del proyecto, me encargué principalmente de realizar la implementación del servicio web, recolección de datos y herramientas aplicadas en el proyecto.

Como conclusión personal, estoy muy agradecido por haber tenido unos compañeros excepcionales, donde el ambiente siempre ha sido amigable. Gracias a ellos, he podido aprender mucho sobre Android, tanto en la parte de frontend con los fragments y el diseño de las pantallas, como en la parte de backend con el View Model, patrón MVVM y Repositorios, donde Jin Tao y Daniel Nasim han jugado un papel muy importante.

Capítulo 10

Manual de instalación

En este apartado se describen las herramientas necesarias para el uso de la aplicación.

10.1 Herramientas necesarias

10.1.1 Android Studio

Una de las herramientas necesarias es Android Studio ya que el proyecto está realizado en esta herramienta. Esta aplicación se puede descargar gratuitamente en `https://developer.android.com/studio` .

10.1.2 Python o Anaconda

Otra herramienta que es necesaria tener instalada es Python o en su lugar Anaconda. Para instalar Python está el siguiente enlace `https://www.python.org/downloads/` y para instalar Anaconda es este otro enlace `https://www.anaconda.com/products/individual#windows`.

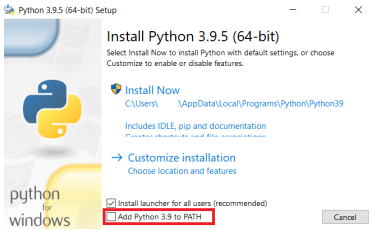


Figura 10.1: Primer paso para la instalación de Python.

En caso de tener instalado Python, es necesario tener la ruta de la extensión de Python en las variables de entorno de "Path". Por defecto, en los

campos de instalación de Python, hay un paso donde puedes hacer que se añada automáticamente a las variables de entorno como se muestra en la figura 10.1.

En caso de tener instalado Anaconda, no es necesario realizar los pasos que se realizan con python ya que Anaconda lo hace automáticamente.

10.1.3 Xampp

La última herramienta necesaria es Xampp. Esta actuará como Base de datos de la aplicación. Esta herramienta se puede descargar en el siguiente enlace: `https://www.apachefriends.org/es/index.html`.

10.2 Preparación para lanzar la aplicación

Una vez instaladas las aplicaciones necesarias, se necesitan instalar las dependencias necesarias. Para ello, es necesario situarse en el mismo directorio donde se encuentra el fichero requirements.txtz se usará el comando "pip install -r requirements.txt" para descargar las dependencias del proyecto, o ir instalando uno por uno las siguientes dependencias en el CMD (command prompt) si solo tiene instalado python o en el cmd de Anaconda si tiene instalado Anaconda:

- pip install flask
- pip install flask_sqlalchemy
- pip install bcrypt
- pip install geopy (Sirve para calcular distancias entre usuario y lugar)
- pip install pyimgur (Se usa la API de imgur para subir la foto en una url)
- pip install apscheduler (Sirve para tareas automáticas)

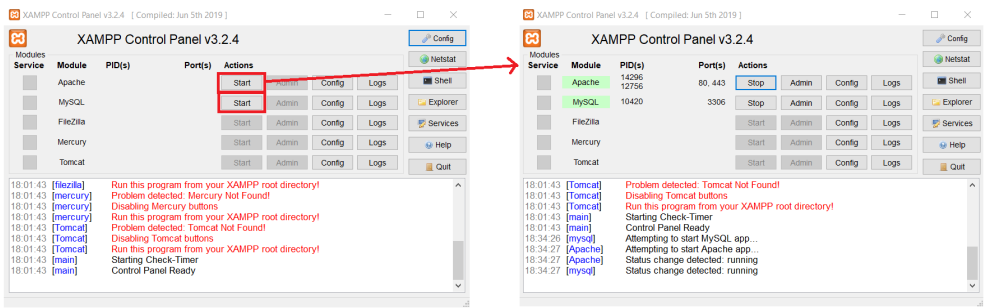


Figura 10.2: Arranque de Xampp.

Después de instalar las dependencias, se pone a ejecutar el servidor. Para ello se inicia la aplicación de xampp y se pone a ejecutar los servicios de Apache y MySQL como se muestra en la figura 10.2.

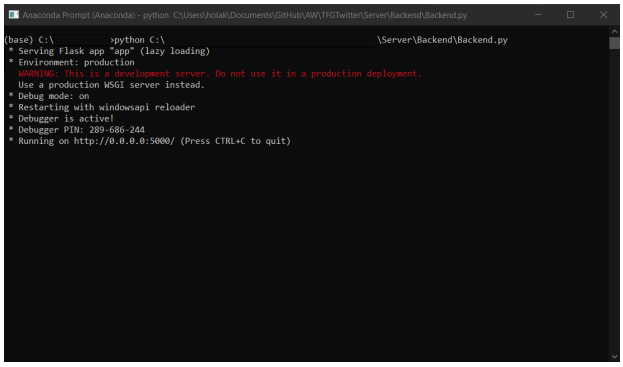


Figura 10.3: Arranque API Backend.

Una vez que estén arrancados los servicios de Xampp se abre el CMD del sistema o el CMD de Anaconda en caso de tener instalado Anaconda y se escribe el comando "python rutaDelFichero" donde la rutaDelFichero es la ruta donde se encuentre el fichero "Backend.py" como por ejemplo "C:/.../Server/Backend/Backend.py". Si todo se ha lanzado correctamente debería aparecer lo mismo que en la figura 10.3.

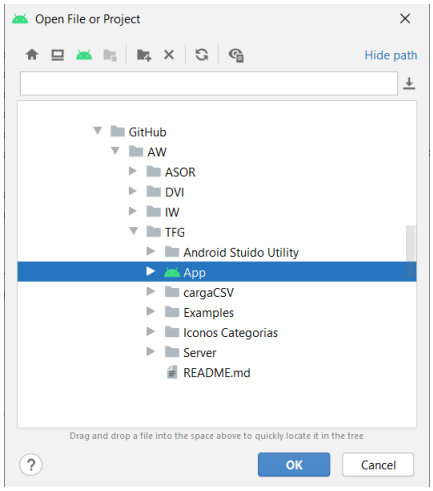


Figura 10.4: Abrir el proyecto en Android Studio.

Por último se abre Android Studio y se abre el proyecto pinchando en "File >Open..." y se busca el proyecto hasta encontrar "App" y pulsar en Ok para abrir el proyecto como se muestra en la figura 10.4.

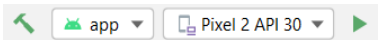


Figura 10.5: Opciones de ejecución de la aplicación en Android Studio.

Tras abrir el proyecto se puede construir el proyecto con el martillo de la figura 10.5, ejecutar el proyecto con el botón de reproducir o cambiar el emulador de móvil entre otras cosas. Se recomienda el uso de un emulador con la versión de la API 30. Además de poder lanzarlo en un emulador, se puede instalar y lanzar desde un móvil real teniendo en cuenta que se deberá conectar a la misma red tanto el ordenador que actuará como servidor como el móvil real.

Figura 10.6: Cambio de IP en Android Studio.

Cabe destacar que si se ejecuta desde un celular, hay que cambiar la dirección IP del archivo "SimpleRequest.java" como se muestra en la figura 10.6.

Capítulo 11

Manual de usuario

En este apartado se explicará detalladamente las vistas con las que el usuario podrá navegar en la aplicación para facilitarle el uso de esta.

Figura 11.1: Splash - Primera pantalla.

Al comienzo de la aplicación se muestra un "Splash" que es la primera pantalla que se le muestra al usuario mostrando el icono de la aplicación como se observa en la Figura 11.1. El "Splash" se muestra durante unos

segundos y luego se redirige a la pantalla principal.

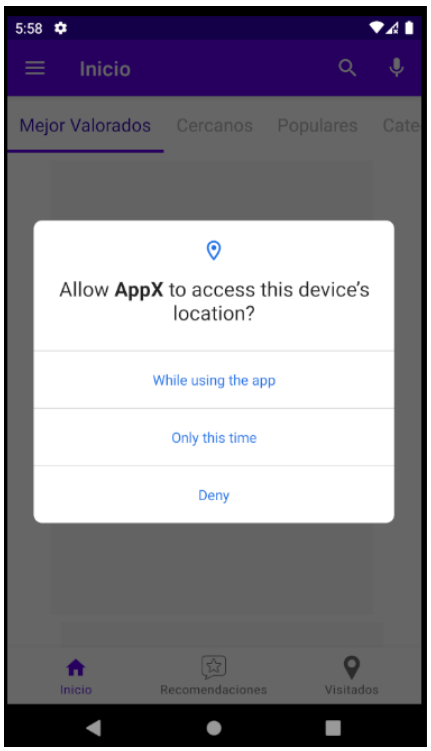


Figura 11.2: Pop Up - Ventana emergente.

La primera vez que un usuario entra en la aplicación aparece un Pop Up que es una ventana emergente donde se pregunta por el acceso a la localización del dispositivo móvil. Es recomendable activarlo para mostrar lugares cercanos de la ubicación del celular. La Figura 11.2 muestra la ventana emergente correspondiente.

Una vez en la pantalla principal aparece el Action Bar que es un elemento en la parte superior de la pantalla, el TabLayout que son pestañas que aparecen debajo del Action Bar en la parte superior, el contenido de la pestaña que por defecto muestra los lugares mejor valorados en la aplicación y un Bottom Navigation que es la barra de navegación situada en la parte inferior de la pantalla. Antes de que se carguen los lugares, se muestra un efecto de que se están cargando los lugares. A este efecto se le llama Shimmer. Una vez cargado los lugares se observa la pantalla como la que se muestra en la Figura 11.3.

Desde la pantalla principal se pueden realizar varias acciones. Pulsar el icono de menú situado en la parte izquierda del Action Bar para abrir el menú (1), pulsar en uno de los lugares para ver en más detalle el lugar (2), moverse en el TabLayout a otra pestaña (3), buscar un lugar por texto o voz (4) o cambiar de pantalla a “Recomendaciones” o “Visitados” en el Bottom

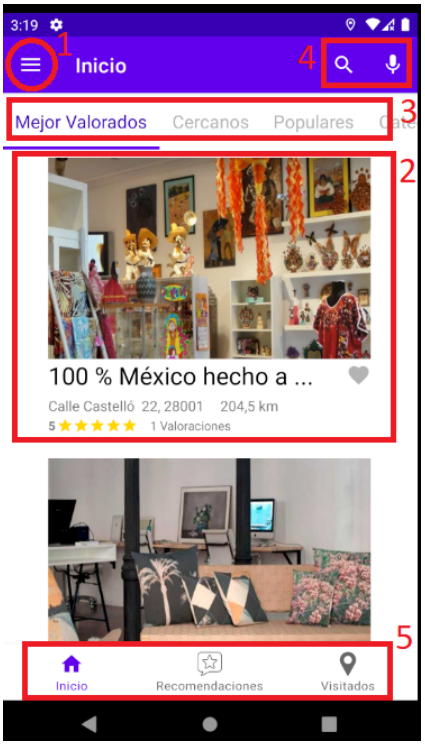


Figura 11.3: Pantalla principal.

Navigation o barra de navegación (5).

Si se pulsa el icono de menú aparece el menú como se muestra en la Figura 11.4.

Desde el menú se puede navegar a los ajustes, el inicio de sesión, el registro y a quiénes somos al no estar registrados en la aplicación. Estas vistas se muestran en la Figura 11.5.

Como se puede observar en la Figura 11.5, dentro de los ajustes podemos activar o desactivar el seguimiento del móvil o cambiar el idioma, dentro del login se muestra un formulario a rellenar para iniciar sesión, dentro del registro se muestra otro formulario para registrarse en la aplicación y en la pantalla de quiénes somos se muestra información de por qué personas están hechas el proyecto. Una vez iniciado sesión o registrado, se redirige a la pantalla principal. Todas las pantallas tienen un icono de flecha que representa volver a la pantalla anterior que es la pantalla principal.

Una vez se ha iniciado sesión, al volver abrir el menú puede mostrarse de dos maneras diferentes dependiendo del rol que tenga el usuario.

Si el usuario tiene solo el rol de usuario, le aparece un menú como la Figura 11.6.

Desde el menú se puede navegar a otras pantallas como perfil, amigos y favoritos como se enseña en la siguiente figura.

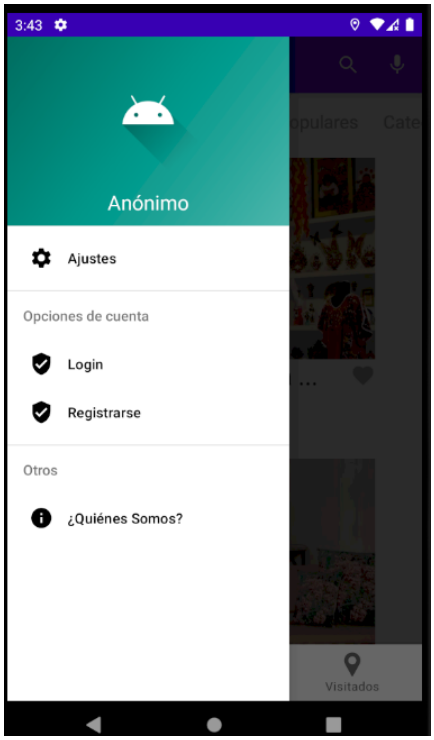


Figura 11.4: Menú sin iniciar sesión.

En la pantalla de perfil se muestran la foto, el número de sitios en favoritos, el número de sitios visitados, nombre real junto con apellidos, nombre de usuario y email. También se puede cambiar algunos datos como la foto de perfil, email o contraseña pulsando en el icono del lápiz situado en la parte superior derecha. En la Figura 11.7 se observa la vista de la pantalla.

En la pantalla de amigos se muestra un TabLayout para ver la lista de amigos y para ver las peticiones de amistad (ver Figuras 11.8 y 11.9). En la lista de amigos se puede borrar un amigo y en las peticiones de amistad se puede aceptar o rechazar una petición. Las pantallas se muestran como en las siguientes figuras. Además hay un icono en la parte superior derecha del menú para añadir amigos. En la vista de añadir amigos hay un campo para escribir el nombre del usuario. Esto se enseña en la Figura 11.10.

En la pantalla de favoritos se muestra una lista de lugares favoritos como se observa en la Figura 11.11. Se puede ver los detalles del lugar pulsando en el lugar o quitar el lugar de favoritos pinchando en el icono del corazón.

Si el usuario tiene también el rol de administrador, le aparece un menú como en la Figura 11.12 donde además de las opciones que tiene el usuario, también tiene una opción para administrar a los usuarios.

En la opción de lista de usuarios, se muestra la lista de usuarios pudiendo buscar a un usuario u ordenarlos por nombre de usuario o nombre real con

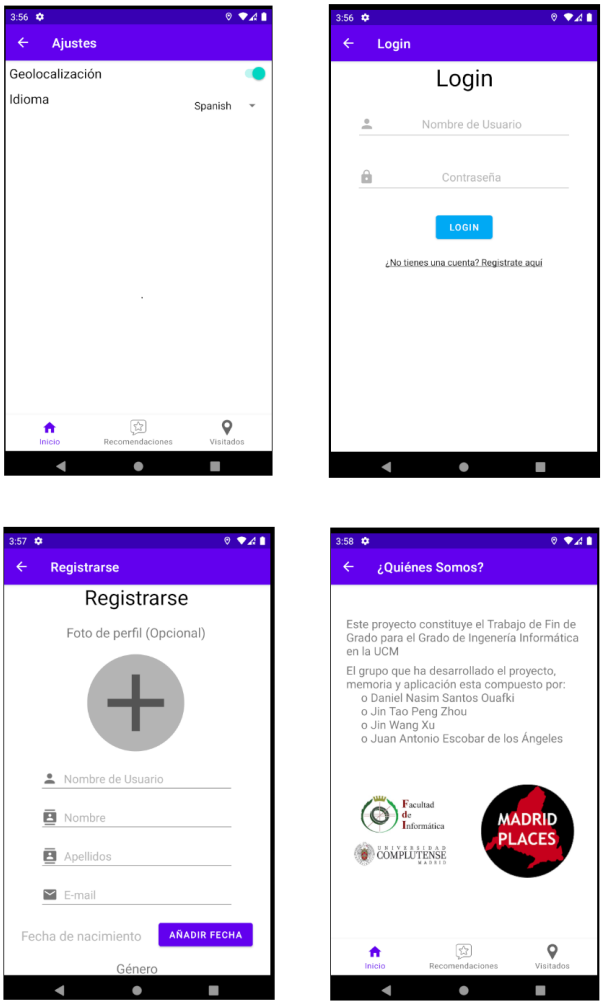


Figura 11.5: Opciones menú sin iniciar sesión.

el botón con el icono AZ situado en la parte superior derecha como se puede ver en la Figura 11.13.

Al pulsar en un usuario se muestra información del usuario como se observa en la Figura 11.14. El administrador puede borrar a un usuario siempre que tenga una causa justificada.

La segunda opción en la pantalla principal era pulsar en uno de los lugares para ver en más detalle el lugar. Dentro de la pantalla de detalles del lugar se muestra información del lugar, pudiendo modificarlo con el icono del lápiz o añadir el lugar a "pendientes por visitar" y recomendar un lugar pulsando el icono de tres puntos y la opción correspondiente. Si el usuario es administrador también puede borrar un lugar. Los usuarios sin iniciar sesión solo pueden ver la información del lugar. Véase el apartado 1 de la Figura

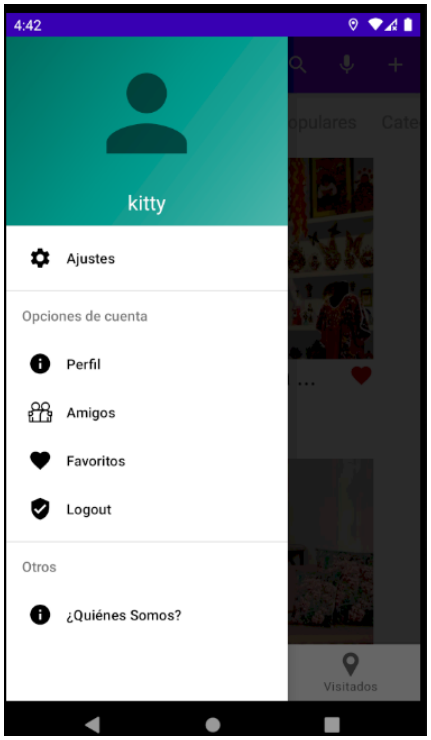


Figura 11.6: Menú iniciando sesión siendo usuario.

11.15.

Aparte hay tres iconos donde sirven para mostrar el mapa (ver Figura 11.16) y hacer una ruta desde el lugar desde donde se encuentra el usuario hasta el lugar, marcar o desmarcar como favorito el lugar o marcarlo como lugar visitado. Véase el apartado 2 de la Figura 11.15. Los usuarios sin iniciar sesión sólo podrán utilizar el mapa.

Al final de la pantalla, se puede valorar y comentar el lugar si se ha iniciado sesión y ver los comentarios escritos por otros usuarios. El usuario también puede borrar su comentario y el administrador puede borrar todos los comentarios. La Figura 11.17 muestra la vista de las acciones descritas.

La tercera opción en la pantalla principal era moverse en el TabLayout a otra pestaña para mostrar los lugares por distintos filtros como lugares populares o por categorías como se enseña en las Figuras 11.18 y 11.19.

En los lugares por categorías hay un icono en el menú situado al lado de los iconos de buscar por texto y por voz que busca lugares por proximidad.

La cuarta opción en la pantalla principal es buscar por texto y por voz. El buscador por texto no tiene más complicación que escribir el lugar a buscar, mientras que el buscador por voz si por primera vez se pulsa el icono, se abrirá una ventana emergente preguntando por los permisos de lectura de voz. Si se permite el uso, se muestra otra ventana emergente para la lectura

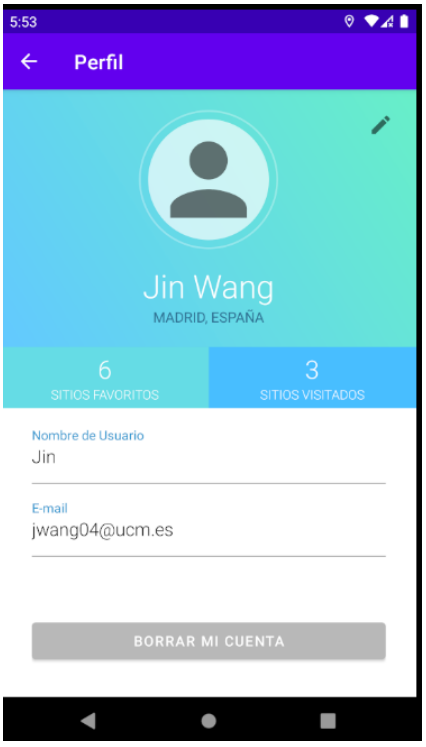


Figura 11.7: Perfil Usuario.

de voz. Véase la Figura 11.20.

Además de estas dos opciones, si el usuario ha iniciado la sesión, aparecerá un icono más para crear nuevos lugares en el Action Bar (ver Figura 11.21).

Pulsando el icono para crear nuevos lugares, muestra un formulario a rellenar para crear el lugar como se puede observar en la Figura 11.22.

La última opción en la pantalla principal es la barra de navegación en la parte inferior.

La opción de "Recomendaciones" tiene un TabLayout para ver las recomendaciones que ha hecho el usuario a otros amigos y las recomendaciones que ha recibido el usuario. Las recomendaciones que el usuario ha hecho a otros usuarios dependiendo de si el amigo la ha aceptado, rechazado o está en estado pendiente se ven de colores distintos. Si se ha aceptado, se ve de color verde y si sigue pendiente de color gris. En la parte de las recomendaciones recibidas, se pueden rechazar o aceptar. Si se aceptan se manda a la lista de pendientes por visitar (ver Figuras 11.23 y 11.24).

La opción de "Visitados" tiene un TabLayout para ver los lugares visitados y los lugares pendientes por visitar. La Figura 11.25 muestra la vista de las pantallas.

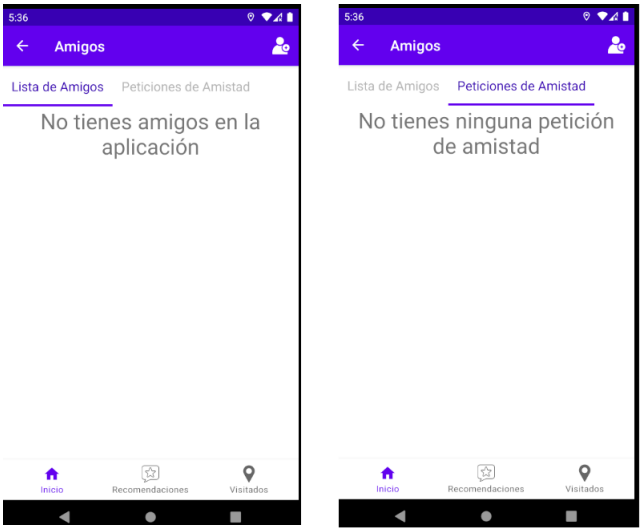


Figura 11.8: Lista de amigos y peticiones de amistad vacío.

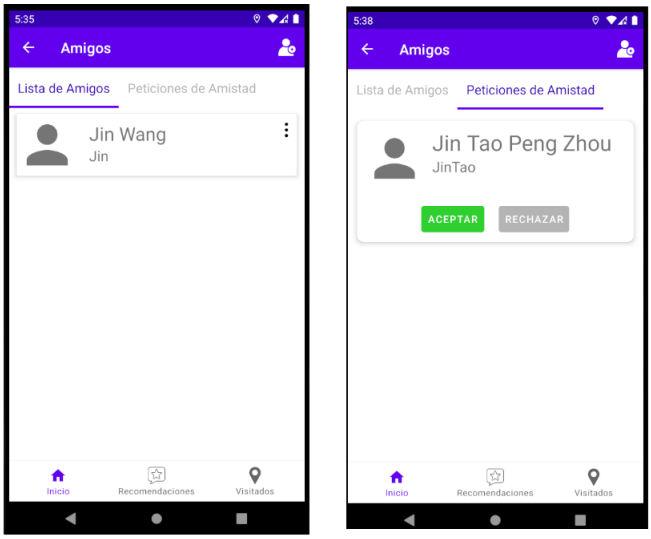


Figura 11.9: Lista de amigos y peticiones de amistad.

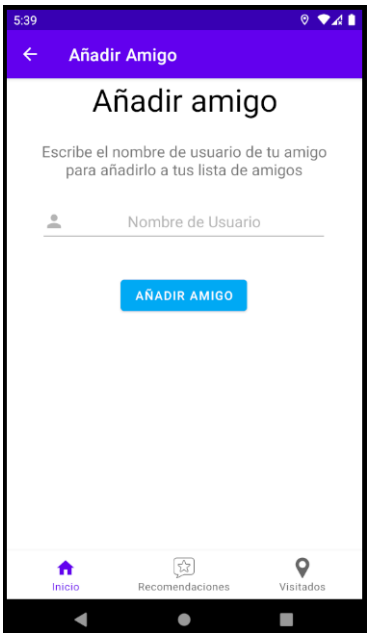


Figura 11.10: Añadir un amigo.

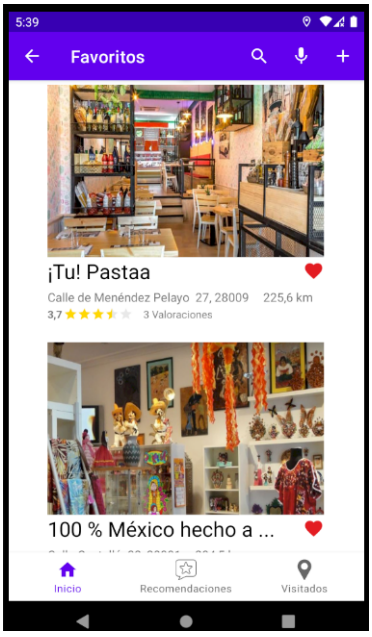


Figura 11.11: Lista de lugares en favoritos.

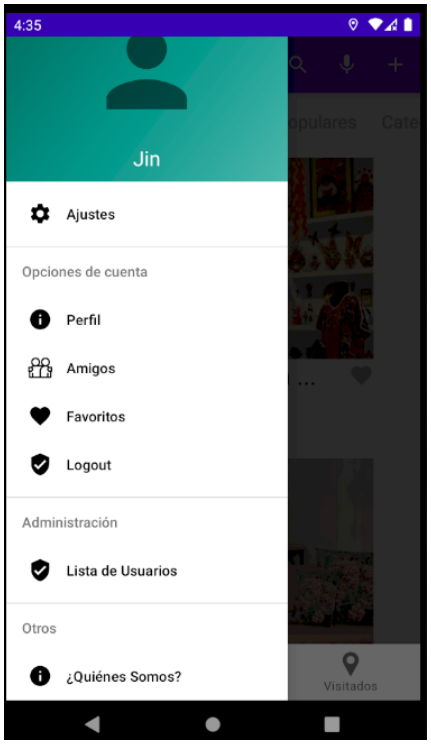


Figura 11.12: Menú iniciando sesión siendo administrador.

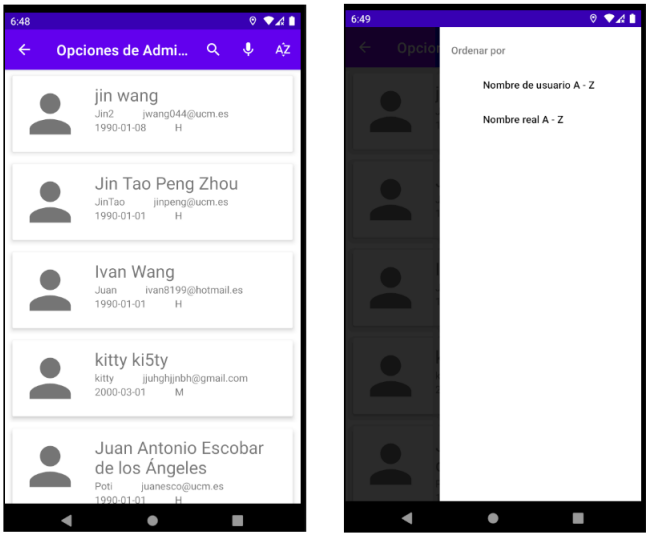


Figura 11.13: Lista de usuarios con filtro de ordenación.

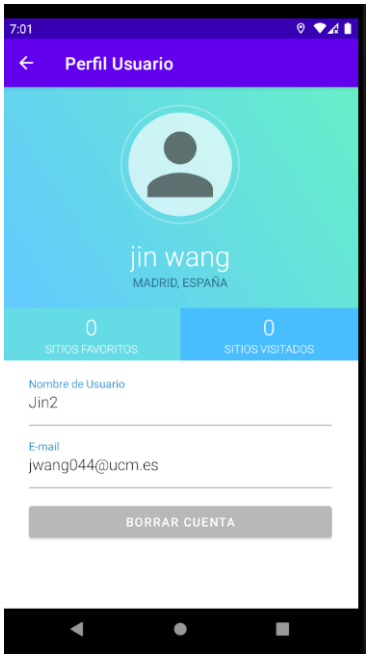


Figura 11.14: Perfil de un usuario a la vista de un administrador.

Figura 11.15: Detalles de un lugar parte 1.

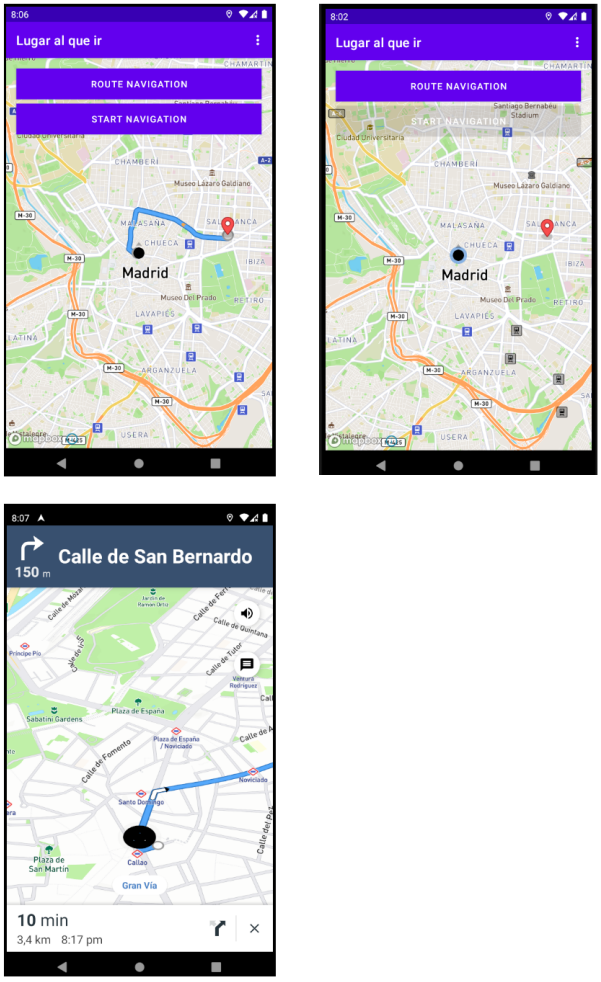


Figura 11.16: Mapa mostrando ubicación del usuario y del lugar con ruta.

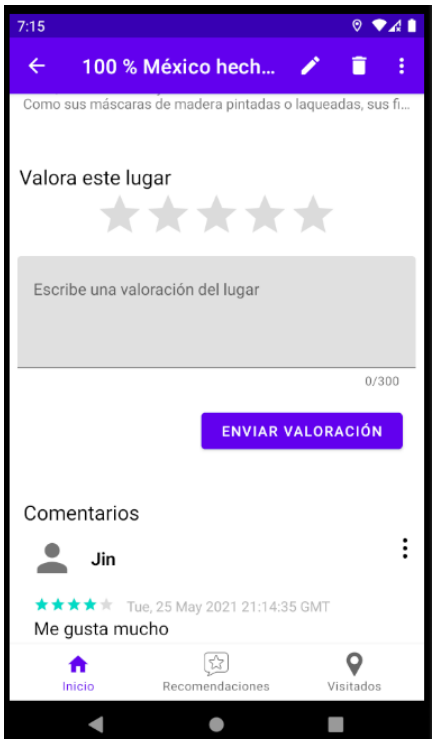


Figura 11.17: Detalles de un lugar parte 2.

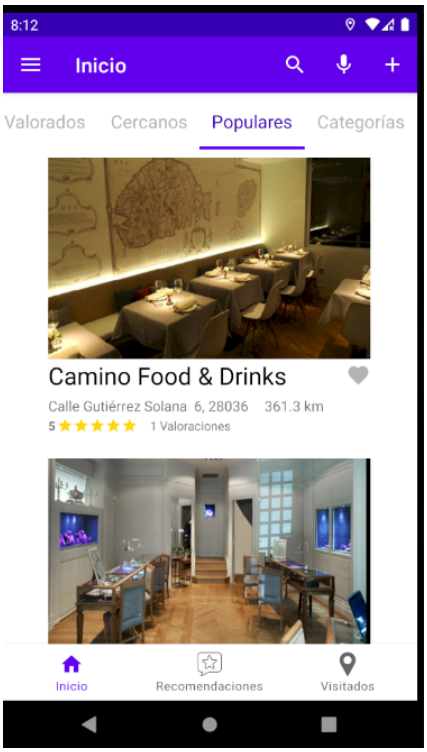


Figura 11.18: Lugares Populares.

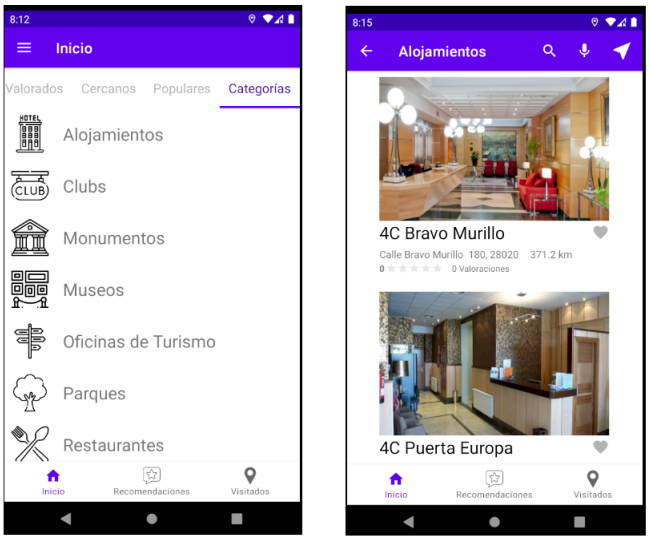


Figura 11.19: Lugares por categorías.

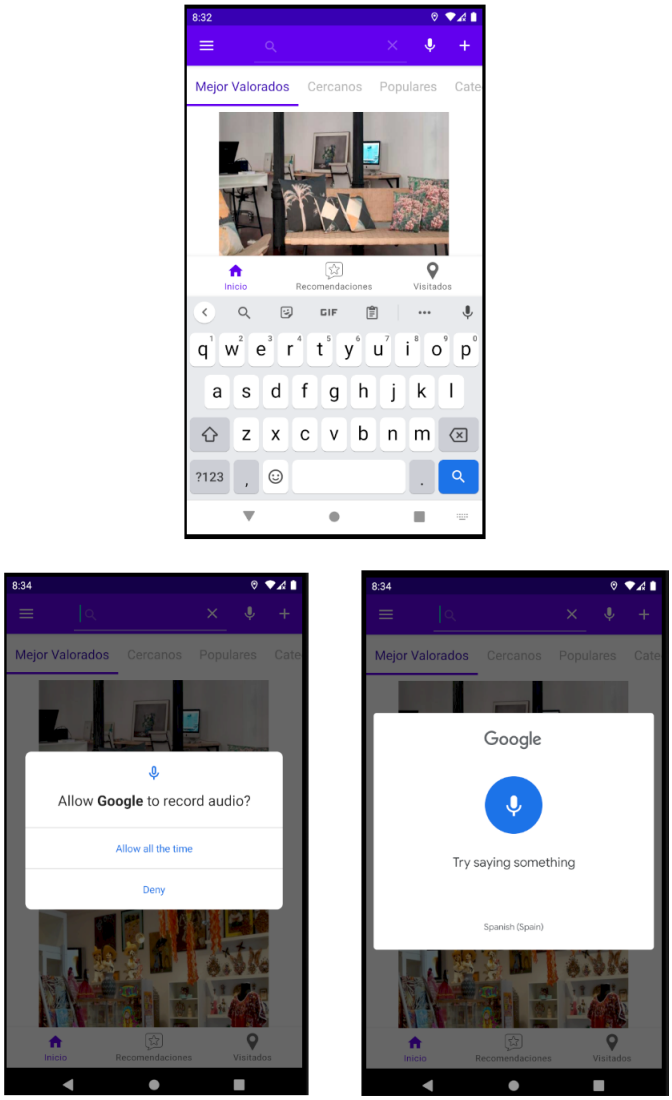


Figura 11.20: Buscador por texto y por voz.

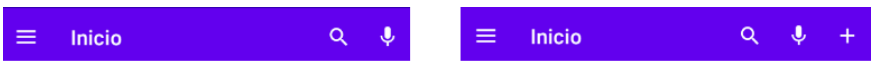


Figura 11.21: Action Bar sin iniciar sesión vs Action Bar iniciado sesión.

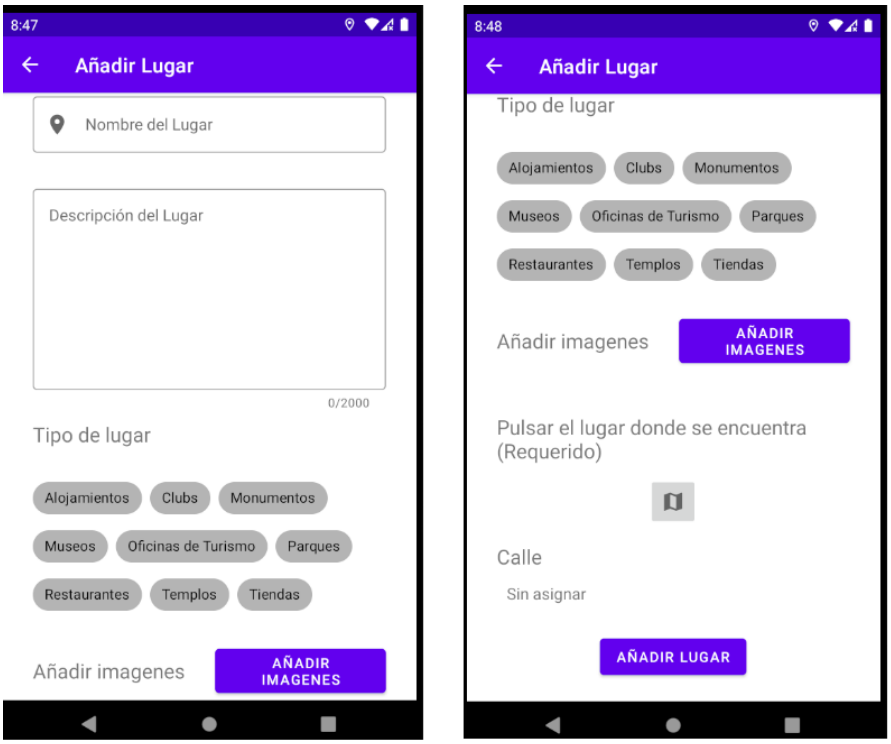


Figura 11.22: Creación de un nuevo lugar.

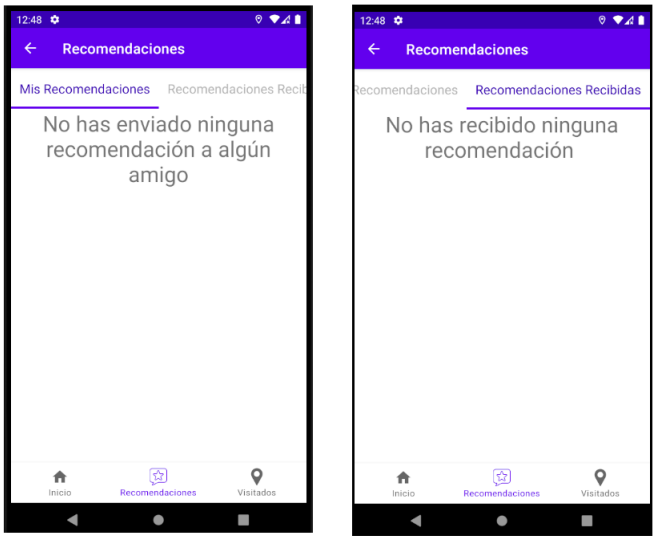


Figura 11.23: Recomendaciones a amigos y recomendaciones pendientes vacías.

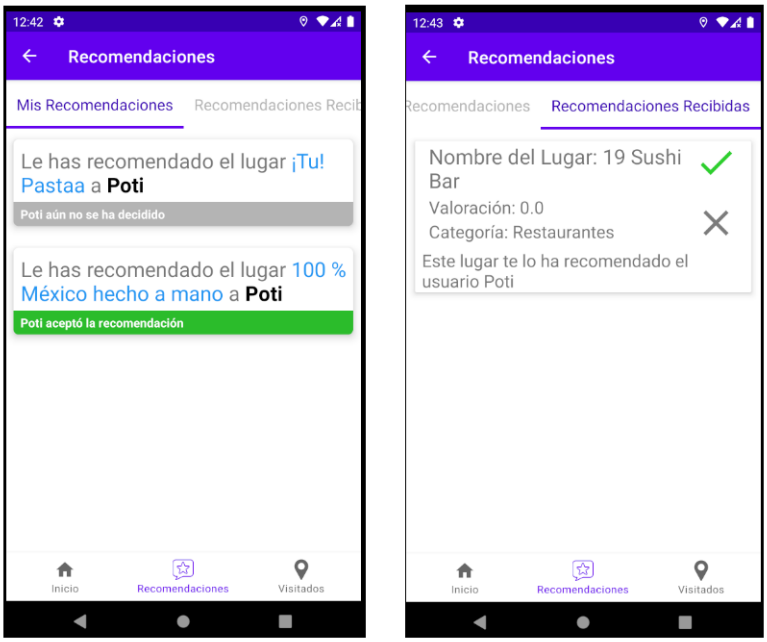


Figura 11.24: Recomendaciones a amigos y recomendaciones pendientes.

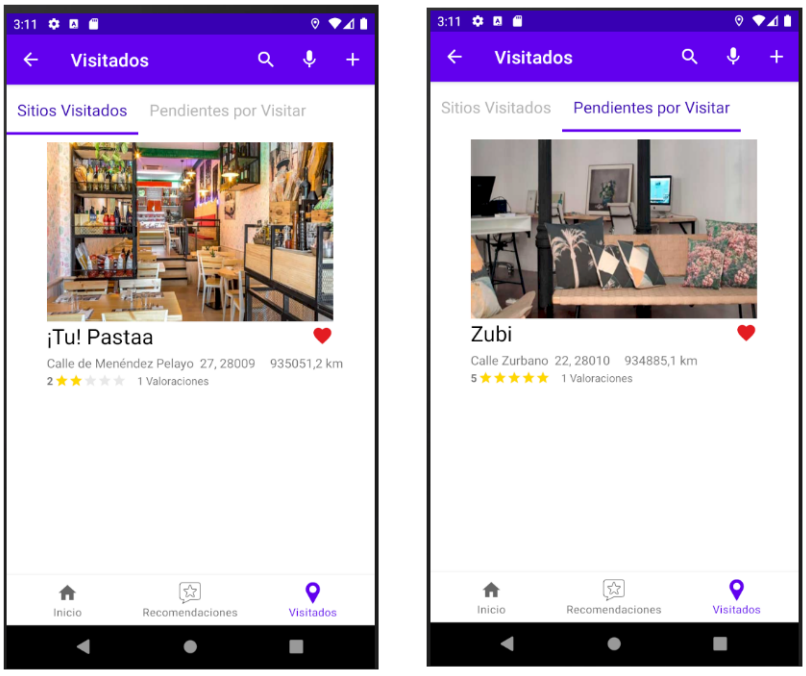


Figura 11.25: Visitados y pendientes de visitar.

Bibliografía

[1] Android. Acerca de Android App Bundle | Desarrolladores de Android. 2012. Disponible en `https://developer.android.com/guide/app-bundle?hl=es-419` (último acceso, 30 de Mayo de 2021).

[2] Android. Objetos parcelables y paquetes | Desarrolladores de Android. 2013. Disponible en `https://developer.android.com/guide/components/activities/parcelables-and-bundles?hl=es` (último acceso, 30 de Mayo de 2021).

[3] Android. Retrofit. 2013. Disponible en `https://square.github.io/retrofit/` (último acceso, 30 de Mayo de 2021).

[4] Android. Volley overview. 2013. Disponible en `https://developer.android.com/training/volley` (último acceso, 30 de Mayo de 2021).

[5] Android. Add the app bar. 2014. Disponible en `https://developer.android.com/training/appbar` (último acceso, 30 de Mayo de 2021).

[6] Android. Build location-aware apps. 2014. Disponible en `https://developer.android.com/training/location` (último acceso, 30 de Mayo de 2021).

[7] Android. Descripción general de los servicios | Desarrolladores de Android. 2014. Disponible en `https://developer.android.com/guide/components/services?hl=es-419` (último acceso, 30 de Mayo de 2021).

[8] Android. Cómo crear vistas deslizantes con pestañas mediante ViewPager. 2016. Disponible en `https://developer.android.com/guide/navigation/navigation-swipe-view?hl=es` (último acceso, 30 de Mayo de 2021).

[9] Android. Support different languages and cultures. 2016. Disponible en `https://developer.android.com/training/basics/supporting-devices/languages` (último acceso, 30 de Mayo de 2021).

[10] ANDROID. Adding Swipe-to-Refresh To Your App . 2017. Disponible en https://developer.android.com/training/swipe/add-swipe-interface (último acceso, 30 de Mayo de 2021).

[11] ANDROID. BottomNavigationView. 2017. Disponible en https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView (último acceso, 30 de Mayo de 2021).

[12] ANDROID. Cómo crear listas dinámicas con RecyclerView. 2017. Disponible en https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es (último acceso, 30 de Mayo de 2021).

[13] ANDROID. Cómo guardar datos de pares clave-valor | Desarrolladores de Android. 2017. Disponible en https://developer.android.com/training/data-storage/shared-preferences?hl=es-419 (último acceso, 30 de Mayo de 2021).

[14] ANDROID. Fragments. 2017. Disponible en https://developer.android.com/guide/fragments (último acceso, 30 de Mayo de 2021).

[15] ANDROID. TabLayout. 2017. Disponible en https://developer.android.com/reference/com/google/android/material/tabs/TabLayout (último acceso, 30 de Mayo de 2021).

[16] ANDROID. Guía de arquitectura de apps | Desarrolladores de Android. 2019. Disponible en https://developer.android.com/jetpack/guide?hl=es-419 (último acceso, 30 de Mayo de 2021).

[17] ANDROID. Navigation. 2019. Disponible en https://developer.android.com/guide/navigation (último acceso, 30 de Mayo de 2021).

[18] ANDROID CODING. How to Implement Auto Image Slider in Android Studio | AutoImageSlider | Android Coding. 2020. Disponible en https://www.youtube.com/watch?v=FU6W-IzgRpo (último acceso, 30 de Mayo de 2021).

[19] ATIF PERVAIZ. Pick One/Multiple Images From Gallery | Android Studio | Java. 2020. Disponible en https://www.youtube.com/watch?v=zhhhc1o2opE&ab_channel=AtifPervaiz (último acceso, 30 de Mayo de 2021).

[20] BAYER, M. Flask-SQLAlchemy Documentation. 2020. Disponible en https://flask-sqlalchemy.palletsprojects.com/en/2.x/ (último acceso, 30 de Mayo de 2021).

[21] CHRISTIAN. Madrid Turismo - Aplicaciones en Google Play. 2016. Disponible en https://play.google.com/store/apps/details?id=turpromadrid.principal (último acceso, 30 de Mayo de 2021).

[22] CODEPATH, G. Ripple Animation | CodePath Android Cliffnotes. 2016. Disponible en https://guides.codepath.com/android/ripple-animation (último acceso, 30 de Mayo de 2021).

[23] CONTRIBUTORS, P. phpMyAdmin. 1998. Disponible en https://www.phpmyadmin.net/downloads/ (último acceso, 30 de Mayo de 2021).

[24] DEVELOPERS, T. P. C. A. bcrypt. 2020. Disponible en https://pypi.org/project/bcrypt/ (último acceso, 30 de Mayo de 2021).

[25] DISFRUTAMADRID. Madrid - Guía de viajes y turismo en Madrid - Disfruta Madrid. 2016. Disponible en https://www.disfrutamadrid.com/ (último acceso, 30 de Mayo de 2021).

[26] DUSTMAN, A. MySQLdb User's Guide. 2012. Disponible en https://mysqlclient.readthedocs.io/user_guide.html (último acceso, 30 de Mayo de 2021).

[27] FUENTES, J. LiveData: todo sobre el componente de arquitectura Android. 2018. Disponible en https://sdos.es/blog/livedata-todo-sobre-el-componente-de-arquitectura-android (último acceso, 30 de Mayo de 2021).

[28] GAME APP STUDIO . How to Route Navigation using Mapbox in Android Studio | Game App Studio Tutorials | Android Studio. 2020. Disponible en https://www.youtube.com/watch?v=MvM2KytTG9U&ab_channel=GameAppStudio (último acceso, 30 de Mayo de 2021).

[29] GEOPY. geopy. 2020. Disponible en https://pypi.org/project/geopy/ (último acceso, 30 de Mayo de 2021).

[30] GOOGLE. Download Android Studio and SDK tools. 2013. Disponible en https://developer.android.com/studio (último acceso, 30 de Mayo de 2021).

[31] GOOGLE. API Google Speech to Text. 2017. Disponible en https://github.com/GoogleCloudPlatform/android-docs-samples/tree/master/speech/Speech (último acceso, 30 de Mayo de 2021).

[32] GRÖNHOLM, A. APScheduler 3.7.0 documentation. 2015. Disponible en https://apscheduler.readthedocs.io/en/stable/userguide.html (último acceso, 30 de Mayo de 2021).

[33] HOME, F. Shimmer for Android. 2018. Disponible en https://facebook.github.io/shimmer-android/ (último acceso, 30 de Mayo de 2021).

[34] HUGGINS, J. Documentation for Selenium. 2018. Disponible en https://www.selenium.dev/documentation/en/ (último acceso, 30 de Mayo de 2021).

[35] IMGUR. API Imgur. 2009. Disponible en https://api.imgur.com/ (último acceso, 30 de Mayo de 2021).

[36] J. Tutorial Flask - Lección 5: Base de datos con Flask SQLAlchemy. 2020. Disponible en https://j2logo.com/tutorial-flask-leccion-5-base-de-datos-con-flask-sqlalchemy/ (último acceso, 30 de Mayo de 2021).

[37] JUDD, S. bumptech/glide. 2017. Disponible en https://github.com/bumptech/glide (último acceso, 30 de Mayo de 2021).

[38] LILKENDEY, J. Consuming a REST API using Retrofit2 with the MVVM Pattern in Android. 2021. Disponible en https://learntodroid.com/consuming-a-rest-api-using-retrofit2-with-the-mvvm-pattern-in-android/ (último acceso, 30 de Mayo de 2021).

[39] LORIA, S. TextBlob 0.16.0 documentation. 2019. Disponible en https://textblob.readthedocs.io/en/dev/ (último acceso, 30 de Mayo de 2021).

[40] ÁNGEL MANEIRO, J. ViewModel, LiveData, Koin y el paradigma MVVM. 2020. Disponible en https://joseangelmaneiro.com/blog/2020/03/24/viewmodel-livedata-koin/ (último acceso, 30 de Mayo de 2021).

[41] MAPBOX. Maps, geocoding, and navigation APIs & SDKs | Mapbox. 2010. Disponible en https://www.mapbox.com/ (último acceso, 30 de Mayo de 2021).

[42] MCKINNEY, W. API reference — pandas 1.2.4 documentation. 2020. Disponible en https://pandas.pydata.org/docs/reference/index.html (último acceso, 30 de Mayo de 2021).

[43] MICROSOFT. Visual Studio Code - Code Editing. Redefined. 2016. Disponible en https://code.visualstudio.com/ (último acceso, 30 de Mayo de 2021).

[44] MULESOFT. Advanced Rest Client Documentation. 2019. Disponible en https://docs.advancedrestclient.com/using-arc/host-rules (último acceso, 30 de Mayo de 2021).

[45] MYSQL. MySQL. 1994. Disponible en https://www.mysql.com/ (último acceso, 30 de Mayo de 2021).

[46] PLAICE, E. tweet-preprocessor. 2020. Disponible en https://pypi.org/project/tweet-preprocessor/ (último acceso, 30 de Mayo de 2021).

[47] POZANCO, P. 101viajes - Reserva de tours, excursiones y visitas guiadas. 2009. Disponible en https://www.101viajes.com/ (último acceso, 30 de Mayo de 2021).

[48] PYTHON. Time Python Documentation. 2011. Disponible en https://docs.python.org/3/library/time.html (último acceso, 30 de Mayo de 2021).

[49] PYTHON. Element Tree XML API. 2016. Disponible en https://docs.python.org/3/library/xml.etree.elementtree.html (último acceso, 30 de Mayo de 2021).

[50] PYTHON. webpagelib.request documentation. 2021. Disponible en https://docs.python.org/es/3.9/library/urllib.request.html (último acceso, 30 de Mayo de 2021).

[51] RAMPRASAD. MVVM with Android Architecture Components (LiveData + ViewModel). 2020. Disponible en https://medium.com/@ramtrg/https-medium-com-ramtrg-mvvm-architecture-components-4d17d3f09bb7 (último acceso, 30 de Mayo de 2021).

[52] RICHARDSON, L. Beautiful Soup Documentation. 2017. Disponible en https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (último acceso, 30 de Mayo de 2021).

[53] RONACHER, A. Flask Documentation. 2020. Disponible en https://flask.palletsprojects.com/en/2.0.x/ (último acceso, 30 de Mayo de 2021).

[54] VAN ROSSUM, G. Python. 1991. Disponible en https://www.python.org/ (último acceso, 30 de Mayo de 2021).

[55] SEIDLER, K. O. XAMPP. 2021. Disponible en https://www.apachefriends.org/es/index.html (último acceso, 30 de Mayo de 2021).

[56] SETHI, R. ViewModel using Retrofit — MVVM Architecture - Ronak Sethi. 2020. Disponible en https://medium.com/@ronkan26/viewmodel-using-retrofit-mvvm-architecture-f759a0291b49 (último acceso, 30 de Mayo de 2021).

[57] SQUARE, INC. OkHttp. 2012. Disponible en https://square.github.io/okhttp/ (último acceso, 30 de Mayo de 2021).

[58] TOM-PRESTON-WERNER, WANSTRATH, C., HYETT, P. J. y CHACON, S. GitHub: Where the world builds software. 2008. Disponible en https://github.com/ (último acceso, 30 de Mayo de 2021).

[59] TWEEPY. Tweepy Documentation — tweepy 4.0.0-alpha documentation. 2012. Disponible en https://docs.tweepy.org/en/latest/ (último acceso, 30 de Mayo de 2021).

[60] UNKNOWN. API Google Translate. 2020. Disponible en https://pypi.org/project/google-trans-new/ (último acceso, 30 de Mayo de 2021).

[61] VERDUGO, D. Componentes de arquitectura de Android, de MVC a MVVM. 2019. Disponible en https://solidgeargroup.com/componentes-de-arquitectura-de-android-de-mvc-a-mvvm/ (último acceso, 30 de Mayo de 2021).

[62] VERDUGO, D. Componentes de arquitectura de Android, de MVC a MVVM. 2019. Disponible en https://solidgeargroup.com/componentes-de-arquitectura-de-android-de-mvc-a-mvvm/ (último acceso, 30 de Mayo de 2021).

[63] ZONZOFOX. MADRID City Guide, Offline Maps, Tours and Hotels - Apps on Google Play. 2016. Disponible en https://play.google.com/store/apps/details?id=guide.spain.madrid.prod (último acceso, 30 de Mayo de 2021).