

TRABAJO DE FIN DE GRADO

FACULTAD DE INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

Aplicación Android para turismo personalizado
en Madrid

*Android application for personalized tourism in
Madrid*



UNIVERSIDAD
COMPLUTENSE
MADRID

Autor: Álvaro Urda Díaz
Tutor: Antonio Sarasa Cabezuelo

MADRID, CURSO 2020–2021

Resumen

Con el aumento del uso de *smartphones* cada vez son herramientas más accesibles para la población y más útiles para desplazarse por las distintas ciudades del mundo.

Aprovechando esta situación se ha desarrollado esta aplicación para el sistema operativo Android la cual permite dentro del área que ocupa Madrid capital que el usuario pueda obtener recomendaciones de distintas localizaciones , puntuar las mismas, agregar comentarios y obtener la dirección del lugar para visitarlo.

También el usuario podrá agregar sus propias localizaciones que considere interesantes y mostrar información sobre las mismas como descripción, fotos, dirección, etc.

Palabras clave

Aplicación Android, mapa, turismo, Madrid, categorías, comentarios, información, puntuación, imágenes, ruta.

Abstract

With the increase in the use of smartphones, they are becoming more and more accessible for the overall population and more useful to move around the different cities of the world.

Taking advantage of this situation, this application has been developed for the Android operating system which allows that within the area occupied by the city of Madrid the user can obtain recommendations for different locations, rate them, add comments and get the address of the place in order to visit it.

The user could also add his own locations that he considers interesting and show some information about them such as description, photos, address, etc.

Keywords

Android application, map, tourism, Madrid, categories, comments, information, score, images, route.

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Estructura de la memoria	2
1. Introduction	3
1.1. Motivation	3
1.2. Objectives	3
1.3. State of the art	3
2. Servicios y aplicaciones de referencia	5
3. Herramientas y tecnologías empleadas	6
3.1. Android Studio	6
3.2. GitHub	6
3.3. Base de datos	6
3.4. Firebase	7
3.4.1. Firebase Authentication	7
3.4.2. Firebase Storage	7
3.4.3. Firebase Cloud Functions	7
3.5. Google Maps Platform API	8
4. Especificación de requisitos	9
4.1. Actores	9
4.2. Módulo de servicios	10
4.3. Módulo de información	15
5. Arquitectura del proyecto	22
6. Modelo de datos	24
6.1. Usuarios	24
6.2. Etiquetas	26
6.3. Lugares	27
6.4. Storage	31
7. Diseño de la aplicación	32
7.1. Elementos de la aplicación	32
7.2. Tipografía	36
7.3. Objetivos del diseño de la aplicación	36
7.4. Iconos	41
8. Implementación	42
8.1. Implementación en Android	42
8.1.1. Actividades	42
8.1.2. Fragmentos	44
8.1.3. Diseños de interfaces	46
8.1.4. Servicios	46
8.2. Función Javascript	80

8.3. Navegación de vistas y patrón de la aplicación	81
8.3.1. Navegación de vistas	81
8.3.2. MVVM	82
9. Evaluación	83
10. Conclusiones y trabajo futuro	92
10.1. Conclusiones	92
10.1. Conclusions	92
10.2. Trabajo futuro	93
Índice de figuras	94
Referencias	97
Apéndice	98
Guía instalación de la aplicación	98
Guía de uso de la aplicación	99
Repositorio código - GitHub	116

1. Introducción

En este capítulo se explicará la motivación del proyecto y los objetivos planteados. Así mismo también se describe la estructura de la memoria a realizar.

1.1. Motivación

En general, las guías de turismo recomiendan visitar los mismos lugares (museos, restaurantes, edificios...) diferenciándose por los comentarios que hacen o puntuaciones que dan a cada lugar. En este sentido, estos documentos son creados por profesionales del turismo, por lo que responde a una información “estándar” u oficial. Sin embargo, hay dos factores que se deben tener en cuenta. Por una parte, en las ciudades hay más lugares que los propuestos por las guías turísticas, y por otra parte, la información que puede ofrecer un turista resultado de su visita puede enriquecer a la información oficial y servir a otro turista futuro. Es por todo ello, que sería interesante disponer de una guía turística “alternativa” que recogiera información de lugares que visitar de una ciudad, y cuya información hubiera sido proporcionada por los propios turistas como resultado de su experiencia.

En este proyecto se propone crear una aplicación que implementa las ideas planteadas anteriormente. Se buscaría que la aplicación permita a los usuarios compartir sus experiencias turísticas de forma que las visitas que se proponen no tienen que coincidir con las que recomiendan la mayoría de guías turísticas. Así mismo el sistema permite encontrar a otros usuarios con gustos parecidos.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación móvil Android que sea capaz de adaptarse a las preferencias de cada usuario para recibir recomendaciones personalizadas y que este pueda compartir sus lugares turísticos preferidos con otros usuarios y expresar su opinión de los de otros. Este objetivo se refina en los siguientes objetivos más específicos:

- Permitir al usuario seleccionar sus preferencias lo cual permitirá recibir recomendaciones orientadas al turismo en función a sus gustos.
- Mantener seguimiento de los lugares preferidos por el usuario.
- Ofrecer la posibilidad a cada usuario de participar de forma activa en la aplicación ya sea puntuando o comentando los lugares visitados.
- Añadir información sobre los lugares visitados pudiendo aportar datos que eran desconocidos por otros usuarios.
- Ser capaz de almacenar la información en bases de datos, de esta forma si el usuario desinstala la aplicación tendrá todo de nuevo disponible con su usuario y contraseña.

1.3. Estructura de la memoria

A continuación se mostrará la estructura de la memoria del proyecto, explicando la temática de cada capítulo.

1. En el capítulo 1 se describe la motivación y objetivos del proyecto y la estructura de la memoria.
2. En el capítulo 2 se comparan aplicaciones y servicios similares al proyecto en desarrollo.
3. En el capítulo 3 se analizan las distintas herramientas seleccionadas para el desarrollo de la aplicación.
4. En el capítulo 4 se describe la especificación de requisitos.
5. En el capítulo 5 se especifica el modelo de datos de la aplicación.
6. En el capítulo 6 se especifica la arquitectura de la aplicación.
7. El capítulo 7 muestra el diseño de la aplicación.
8. El capítulo 8 muestra la implementación de la aplicación.
9. En el capítulo 9 se realiza una evaluación de la aplicación por usuarios y encuestas realizadas a los mismos.
10. En el capítulo 10 se muestran las conclusiones finales del proyecto.
11. En el apéndice se muestra una guía de instalación y otra de uso de la aplicación con ejemplos mostrando la interfaz de usuario.

1. Introduction

This chapter will explain the motivation behind the project and its objectives. It will also describe the structure of the report to be carried out.

1.1. Motivation

Overall, tourist guides recommend visiting the same places (museums, restaurants, buildings, etc.), differentiated by the comments they make or the scores they give to each place. In this sense, these documents are created by tourism professionals, so they respond to “standard” or official information. However, there are two factors that must be taken into account. On one hand, there are more places in cities than those proposed by tourist guides, and on the other hand, the information that a tourist can offer as a result of his or her visit can enrich the official information and serve another tourist in the future. For all these reasons, it would be interesting to have an “alternative” tourist guide that would gather information on places to visit in a city, and whose information would have been provided by the tourists themselves as a result of their experience.

This project proposes to create an application that implements the ideas outlined above. The aim is for the application to allow users to share their experiences in such a way that the proposed visits do not have to coincide with those that are included in most tourist guides. The system also allows you to find other users with similar tastes.

1.2. Objectives

The main objective of this project is to develop an Android mobile application that is able to adapt to the preferences of each user in order to receive personalised recommendations and to allow users to share their favourite tourist sites with others and express their opinion of those of others. This objective is refined into the following more concrete objectives:

- Allow the user to select their preferences, which will enable them to receive tourism-oriented recommendations based on their tastes.
- Keep track of the user’s preferred locations.
- Offer the possibility for each user to actively participate in the application either by rating or commenting on the places visited.
- Adding information about the places visited and being able to provide information that was unknown to other users.
- Be able to store the information in databases, so that if the user uninstalls the application, everything will be available again with his username and password.

1.3. State of the art

The structure of the project report will be shown below, explaining the subject matter of each chapter.

1. Chapter 1 describes the motivation and objectives of the project and the structure of the report.

2. Chapter 2 compares applications and services similar to the project under development.
3. Chapter 3 discusses the different tools selected for the development of the application.
4. Chapter 4 describes the requirements specification.
5. Chapter 5 specifies both the architecture and the data model of the application.
6. Chapter 6 shows the design and implementation of the application as well as the structure and use of the database.
7. In chapter 7, an evaluation of the application by users and user surveys are carried out.
8. Chapter 8 shows the final conclusions of the project.
9. The appendix shows an user guide of the application with examples showing the user interface.

2. Servicios y aplicaciones de referencia

Han sido tenidos en cuenta una serie de servicios y aplicaciones como referencia para el desarrollo de la misma.

Un servicio web tomado como referencia a pesar de tener una finalidad completamente diferente ha sido YouTube, o mejor dicho la funcionalidad de YouTube de listas de reproducción, o también llamadas “playlists”. Las listas de reproducción permiten al usuario seleccionar una serie de vídeos con una temática en común y agrupar esos vídeos para encontrarlos o reproducirlos posteriormente.

Otra herramienta de YouTube utilizada como referencia para este proyecto ha sido el sistema de etiquetas de YouTube. Un vídeo puede tener una o varias etiquetas o categorías en función de su temática. De esta forma ha sido hecho con los distintos lugares en la aplicación, cada lugar puede tener varias etiquetas y estos se pueden filtrar en función de las mismas. Así será más sencillo hacer recomendaciones al usuario en función de sus gustos.

Por último se ha tomado como referencia la tienda online Amazon. Esta tienda permite filtrar las búsquedas de productos en función de la puntuación dada por los usuarios la cual puede ir de una a cinco estrellas, si un producto tiene cinco estrellas pero varios votos (que pueden ir de unos pocos a miles) será considerado por la tienda más relevante que otro que tenga cinco estrellas pero solo un par de votos por ejemplo. Esta forma de organizar los lugares por puntuación ha sido tomada en consideración para el desarrollo del proyecto.

3. Herramientas y tecnologías empleadas

En esta sección se especificarán las distintas herramientas empleadas para el desarrollo de la aplicación móvil.

3.1. Android Studio

Una de las herramientas más importantes ha sido el entorno de desarrollo Android Studio. Este software creado por Google como indica su nombre permite el desarrollo de aplicaciones para el sistema operativo Android en todas sus variantes: Smartphone, smartwatch, tv inteligente. . .

Se ha utilizado este software para desarrollar una aplicación para smartphone debido a que permite la creación de aplicaciones nativas para Android permitiendo un funcionamiento óptimo de la misma.

Android Studio[17] utiliza dos lenguajes de programación, Java y Kotlin. Ambos hacen uso de la máquina virtual de Java (JVM – Java Virtual Machine), también permite el uso del lenguaje XML utilizado para definir recursos como librerías a descargar, cadenas de texto, variable globales, diseño gráfico de los distintos componentes de la aplicación con los que interactúa el usuario.

En este proyecto se ha optado por el uso de Kotlin sobre Java[19], esto se debe a varios motivos, entre ellos que al utilizar ambos la JVM permite usar clases creadas con el lenguaje de programación Java y que para el desarrollo de aplicaciones móviles Kotlin es más práctico y flexible permitiendo escribir lo mismo que con Java pero con muchas menos líneas de código, incluso Google ha establecido en Android Studio a Kotlin como el lenguaje de programación por defecto para sus proyectos.

3.2. GitHub

Se ha utilizado la plataforma GitHub[15] para mantener copias de seguridad del código y mantener distintas ramas de desarrollo. Esto ha permitido añadir al proyecto nuevas funciones de forma simultánea sin poner en riesgo a el resto de la aplicación.

Se ha utilizado la versión de Git[14] que viene instalada en Android Studio ya que este provee una versión de Git con una interfaz de usuario muy intuitiva. Esta permite seleccionar de una forma muy sencilla ramas, que archivos seleccionar al hacer subir código al repositorio, etc.

3.3. Base de datos

La base de datos es una parte esencial para el funcionamiento de la aplicación, se busca que la información almacenada a nivel local sea la mínima posible. De esta forma si se necesita hacer una actualización de los servicios o el usuario cambia de smartphone esto no tendría un impacto negativo en su experiencia de uso. Además esto permite al administrador editar o eliminar información si se da la necesidad.

Se ha seleccionado una base de datos llamada *Firebase Realtime Database*[11]. Esta es un tipo de base de datos gratuita que tiene soporte para Android, iOS y Web. La información se almacena en formato JSON y cualquier cambio en los datos se refleja inmediatamente en todas las plataformas a las que ofrece servicios. Al pertenecer a Google provee un servicio especialmente eficiente a Android y al necesitar descargar información de una forma casi inmediata *Firebase Realtime Database* es idónea.

Es necesario mencionar que la base de datos solo es una herramienta dentro de una plataforma llamada *Firebase* que provee otros servicios como un sistema de autenticación utilizado para registrarse y acceder a la aplicación de forma segura, un sistema de almacenamiento en la nube que permite almacenar archivos como imágenes y un sistema de funciones en la nube las cuales permiten modificar la información de la base de datos de forma automática ya sea para editar, eliminar o agregar campos de información.

Para desplegar las funciones en la nube se ha utilizado la consola de comandos de Firebase (Firebase CLI) de Node.js, estas funciones han sido programadas utilizando el lenguaje de programación Javascript siendo el lenguaje de programación exigido para las mismas.

3.4. Firebase

Firebase[10] es una plataforma creada por Google que provee un conjunto de servicios para el desarrollo de aplicaciones web. La plataforma se encuentra en la nube y está integrada con Google Cloud Platform. *Realtime Database* es uno de los servicios que provee esta plataforma, pero hay más.

3.4.1. Firebase Authentication

Permite crear cuentas y acceder a las mismas de forma segura para todas las aplicaciones que utilicen los servicios de Firebase. Esto permite acceder al resto de los servicios de Firebase incluida la base de datos. También permite crear nuevos usuarios usando cuentas de Google, Twitter, Facebook, Github, etc.

Se ha utilizado este servicio para poder registrar nuevos usuarios y acceder a los servicios de la aplicación.

3.4.2. Firebase Storage

Servicio que permite almacenar contenido en la nube y que permite desde las distintas plataformas a las que Firebase provee servicio cargar y descargar objetos como imágenes, vídeos, archivos de texto, etc.

Se ha utilizado esta herramienta para cargar y descargar imágenes de las localizaciones que los usuarios podrán crear.

3.4.3. Firebase Cloud Functions

Esta herramienta está pensada para desplegar funciones sin depender de un servidor y funcionar de forma conjunta junto a el resto de servicios de Firebase. Esto permite al desarrollador cambiar cosas sobre la marcha si las condiciones del proyecto así lo requieren ya sea para añadir

nuevas funcionalidades o modificar las ya existentes.

Cloud Functions[5] permite funcionar como *back-end* para las distintas aplicaciones a las que Firebase provee servicios. Se pueden crear funciones que detecten si en *Firebase Authentication* un usuario nuevo ha sido creado, si ha sido añadido un objeto nuevo a *Firebase Storage* o si se ha dado un cambio en la base de datos de Firebase. Pudiendo crear distintas respuestas a estos eventos, como modificar la base de datos, mandar un email o una notificación, etc.

Para este proyecto se han buscado utilizar estas funciones de forma que actualicen ciertos elementos de la base de datos cuando sea necesario. Estas funciones han sido escritas utilizando el lenguaje de programación Javascript[18].

3.5. Google Maps Platform API

Se ha utilizado los servicios y funciones proveídos por Google Maps[8] para poder establecer y encontrar la localización de los lugares deseados, la ubicación del actual del usuario de la aplicación y permite además acceder al buscador de direcciones de Google que en función de un texto introducido intenta predecir cual es la dirección que el usuario busca.

Con esto cuando el usuario desee agregar una dirección solo tendrá que buscar la dirección y aparecerá en el mapa, posteriormente podrá agregarla a la información del lugar.

4. Especificación de requisitos

En este capítulo se especificarán los casos de uso de la aplicación a desarrollar en el TFG.

4.1. Actores

A continuación, se describen los actores que participarán en los casos de uso.

- **Usuarios no registrados:**

Aquel usuario sin cuenta, únicamente puede registrarse en la aplicación.

- **Usuarios registrados:**

Aquel usuario que dispone de cuenta en la aplicación y puede hacer uso de todos los servicios de la aplicación.

- **Administrador:**

Aquel usuario encargado de realizar el mantenimiento de la aplicación.

La funcionalidad de la aplicación se agrupa en los siguientes módulos:

- Módulo de servicios.

- Módulo de información.

En las siguientes secciones se describirán los casos de uso asociados a cada uno de los módulos.

4.2. Módulo de servicios

En este módulo se agrupan las funciones referente a los servicios que ofrece la aplicación al usuario. En la figura 1 se muestran especificados los casos de uso y los actores que intervienen.



Figura 1: Diagrama casos de uso módulo servicios

A continuación se mostrarán los casos de uso.

Función	Crear cuenta
Identificador	M-1
Actor	Usuario no registrado
Descripción	El usuario sin registrar ha de introducir un email y una contraseña para poder registrarse en la aplicación y acceder a sus servicios.
Precondición	No puede insertar cualquier texto en el espacio de email
Secuencia estándar	1. El usuario se encontrará en una pantalla en la que se verán 3 cajas de texto. 2. En la primera caja introduce el correo deseado. 3. En la segunda la contraseña y en la tercera introduce de nuevo la contraseña para asegurarse de que está bien escrita.
Postcondición	Permite al usuario acceder a los servicios de la app sin necesidad de ir a la pantalla de inicio de sesión.
Excepciones	Dejar uno de los campos en blanco impedirá que se avance a la siguiente pantalla.

Función	Iniciar sesión
Identificador	M-2
Actor	Usuario registrado, Administrador
Descripción	El usuario introduce un correo electrónico y una contraseña para poder acceder a la aplicación.
Precondición	Debe de haber una cuenta creada.
Secuencia estándar	El usuario en la pantalla del login deberá introducir en dos cajas de texto su email y contraseña y dar al botón de aceptar.
Postcondición	El usuario podrá acceder a los servicios de la aplicación.
Excepciones	Si el usuario introduce mal el email, la contraseña o deja algún campo en blanco se le pedirá que vuelva a introducir los datos correctamente.

Función	Cerrar Sesión
Identificador	M-3
Actor	Usuario registrado, Administrador
Descripción	El usuario sale de su sesión en la aplicación
Precondición	Debe haber iniciado sesión.
Secuencia estándar	1. En un menú lateral de la aplicación (probablemente menú lateral) tendrá la opción de cerrar sesión en un botón. 2. Al darle a ese botón aparecerá un mensaje preguntando al usuario si desea salir de la aplicación con los botones aceptar y cancelar. 3. Al darle a aceptar se sale de la aplicación.
Postcondición	Dejan de recibirse los servicios de la aplicación.
Excepciones	Si se da a cancelar se sigue dentro de la aplicación.

Función	Selección de preferencias
Identificador	M-4
Actor	Usuario no registrado, Usuario registrado, Administrador
Descripción	El usuario en función de sus intereses seleccionará una serie de etiquetas para que la aplicación en función de sus gustos le haga recomendaciones inteligentes.
Precondición	
Secuencia estándar	Tras crear una cuenta el usuario le aparecerá una pantalla con las características a seleccionar deseadas, después de seleccionar estas características le dará al botón de aceptar e irá a la pantalla principal.
Postcondición	El usuario recibirá recomendaciones en función a sus gustos.
Excepciones	No se pueden dar, el usuario obligatoriamente ha de seleccionar como mínimo una preferencia.

Función	Ver recomendaciones
Identificador	M-5
Actor	Administrador, Usuario registrado.
Descripción	El usuario puede ver todas las recomendaciones personalizadas de lugares a visitar en función de varios datos. inteligentes.
Precondición	Haber seleccionado las preferencias del usuario y estar conectado a internet para obtener la información necesaria para poder hacer las recomendaciones.
Secuencia estándar	El usuario entra en la pantalla de recomendaciones y ahí se encuentra las recomendaciones personalizadas de la app divididas en categorías o secciones (restaurantes, museos, lugares para pasar la tarde, lugares al aire libre, etc). En función de sus necesidades puede recorrer las distintas categorías y seleccionar el lugar que más le interese.
Postcondición	Al seleccionar un lugar llevará al usuario al perfil de ese lugar.
Excepciones	Si el teléfono no está conectado a internet no se podrán hacer recomendaciones.

Función	Darse de baja
Identificador	M-6
Actor	Administrador, Usuario registrado.
Descripción	El usuario elimina su cuenta de la aplicación. inteligentes.
Precondición	Debe de haber una cuenta creada y haber iniciado sesión.
Secuencia estándar	1. En el menú lateral de la aplicación el usuario tendrá un botón para borrar la cuenta. 2. Si el usuario pulsa el botón, le aparecerá una ventana preguntando si está seguro de su decisión. 3. Si selecciona que está seguro se cerrará la sesión y se borrará la cuenta del usuario de la base de datos.
Postcondición	El usuario no podrá acceder a los servicios propios de un usuario registrado.
Excepciones	Si dice que no desea eliminar su cuenta en la ventana emergente no pasará nada.

Función	Ver información lugar
Identificador	M-7
Actor	Administrador, Usuario registrado.
Descripción	Los usuarios ven información sobre una localización concreta. inteligentes.
Precondición	Seleccionar una localización de las recomendadas por la aplicación.
Secuencia estándar	1. Se muestra información del lugar seleccionado en concreto (nombre, descripción, puntuación general..). 2. Aparece la opción de puntuar el lugar y ver su localización, además de añadirlo a una lista de favoritos.

4.3. Módulo de información

En este módulo se agrupan aquellas funciones que permiten al usuario ver, añadir o modificar información. En el diagrama de casos de la figura 2 se muestran las funciones y actores involucrados en este módulo.

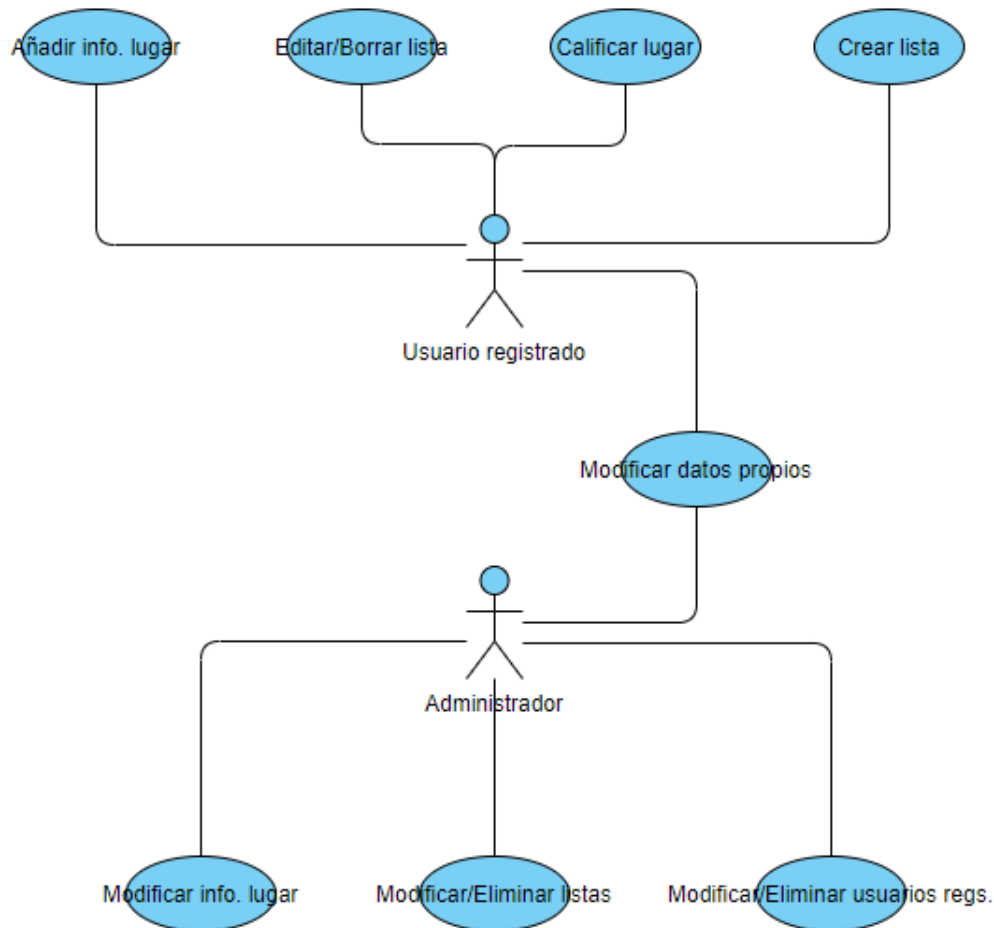


Figura 2: Diagrama casos de uso módulo información

A continuación se mostrarán los casos de uso.

Función	Añadir información lugar
Identificador	M-8
Actor	Usuario registrado, Administrador.
Descripción	Agregar información sobre una localización concreta.
Precondición	Encontrarse en pantalla dar alta lugar.
Secuencia estándar	1. El usuario al dar de alta lugar tendrá que introducir información sobre el lugar en cuestión (nombre, descripción, localización...) 2. Si el usuario pulsa el botón de subir, podrá ver la información del lugar.
Postcondición	La información será visible para el resto de los usuarios.
Excepciones	Si el texto está vacío el botón Subir estará bloqueado.

Función	Calificar lugar
Identificador	M-9
Actor	Usuario registrado, Administrador.
Descripción	Se le da una puntuación a un lugar.
Precondición	Hay que estar en la pantalla de la vista del lugar que se desea puntuar.
Secuencia estándar	1. Buscar la sección de puntuación dentro del perfil y establecer la puntuación deseada.
Postcondición	Se sube la puntuación a una base de datos, si había una puntuación previa la sustituirá.
Excepciones	Si no se establece ninguna puntuación el botón de puntuar permanecerá bloqueado.

Función	Crear lista
Identificador	M-10
Actor	Usuario registrado, Administrador.
Descripción	El usuario crea una o varias listas con los lugares turísticos que desee.
Precondición	Hay que estar en la pantalla de listas de usuario.
Secuencia estándar	1. . En la pantalla de listas del usuario el usuario selecciona el botón de crear nueva lista y la llama como desee. 2. Al crear la lista o después es posible agregar nuevos lugares o eliminar los ya presentes.
Postcondición	En la pantalla de listas aparecerá la nueva lista disponible.

Función	Editar/Borrar lista
Identificador	M-11
Actor	Usuario registrado, Administrador.
Descripción	El usuario puede editar las listas ya creadas modificando su contenido o borrarlas directamente.
Precondición	La lista a modificar ha de existir.
Secuencia estándar	1. Se selecciona una de las listas en la pantalla de listas. 2. El usuario podrá eliminar elementos de la lista o pulsando un botón de añadir, agregar nuevos lugares. 3. Al pulsar añadir le llevará a la pantalla de recomendaciones para que añada los sitios que desee.
Postcondición	Los cambios realizados se mostrarán en la lista seleccionada anteriormente disponible.

Función	Modificar/Eliminar usuarios registrados
Identificador	M-12
Actor	Administrador.
Descripción	El administrador podrá modificar la información de los distintos usuarios registrados o borrar directamente sus perfiles.
Precondición	Los usuarios a eliminar han de ser usuarios registrados.
Secuencia estándar	1. Se selecciona a un usuario registrado. 2. Aparecerá por cada usuario una serie de campos con su información personal, datos añadidos por ellos y la opción de dar de baja su cuenta abajo del todo. 3. Si el administrador desea editar la información del usuario puede ir a cualquiera de esos campos y hacer como en el caso de Editar las listas de usuarios. 4. Si desea dar de baja al usuario aparecerá una ventana preguntando si está seguro, al dar que si el usuario se dará de baja.
Postcondición	Los cambios realizados afectarán directamente a la base de datos y a todos los usuarios.

Función	Modificar información lugar
Identificador	M-13
Actor	Usuario registrado, Administrador.
Descripción	El administrador o usuario creador del lugar podrá editar la información de un lugar determinado (por ejemplo si un usuario ha puesto un comentario de mal gusto).
Precondición	Hay que seleccionar una localización de las recomendadas por la aplicación.
Secuencia estándar	1. El administrador o usuario verá la información proveída por todos los usuarios registrados. 2. En todos los textos aparecerá la opción de editarlos con un botón. 3. Al pulsarlo le llevará a otra vista con un cuadro de texto donde podrá escribir y editar el texto previamente escrito. 4. Abajo tendrá un botón llamado Subir, que al pulsarlo esa información será visible para el resto de los usuarios registrados y no registrados.
Postcondición	Los cambios realizados serán visibles a todos los usuarios.
Excepciones	Si se sale de la edición antes de dar terminar no se guardan los cambios.

Función	Modificar datos propios
Identificador	M-14
Actor	Usuario registrado, Administrador.
Descripción	Tanto el usuario como el administrador podrán editar la información proveída por sí mismos (nombre, contraseña, preferencias)
Precondición	La cuenta tiene que haber sido configurada correctamente al principio.
Secuencia estándar	1. En la pantalla de perfil de usuario aparecerá la opción ajustes donde se verá la opción de editar datos usuario. 2. Al seleccionar esa opción el usuario podrá ver toda la información previamente dada en cajas de textos y abajo del todo un boton guardar 3. Al dar a guardar se guardan esos cambios en la base de datos.
Postcondición	Los cambios se reflejarán en la cuenta del usuario, por ejemplo si se cambia la contraseña no se podrá volver a entrar con la antigua.

5. Arquitectura del proyecto

El sistema ha sido diseñado utilizando la aplicación de Android y los servicios de Firebase, Firebase es lo que se conoce como un *Backend as a Service* o BaaS[4].

La aplicación Android funciona a partir de la versión 6.0 de Android (SDK 23), esto lo hace compatible según el propio IDE de Android Studio con aproximadamente un 85 % de los dispositivos disponibles a nivel global.

BaaS provee a las aplicaciones móviles o servicios web una forma de integrarse con distintos servicios, almacenamiento en la nube, etc. El desarrollo de la parte backend (notificaciones, integración con redes sociales, bases de datos) queda en manos de un tercero y los desarrolladores se encargan del lado del cliente, en este caso la aplicación móvil.



Figura 3: Backend as a Service

Luego se encuentran los servicios *Functions as a Service* o FaaS[13]. Estos servicios permiten desplegar funciones sobre la marcha que se ejecutarían de forma automática al reaccionar a determinados eventos establecidos por el desarrollador, esto permite agregar o modificar micro-servicios sobre la marcha en función de las necesidades del desarrollador y los usuarios.

La aplicación desarrollada en este proyecto es una aplicación sin servidor o *Serverless* ya que se utilizan un conjunto de servicios gratuitos backend (Firebase) como bases de datos, autenticación de usuarios, almacenamiento de objetos, etc.

También hace uso de servicios los servicios FaaS proveídos por Firebase llamados *Firebase Cloud Functions*. Estos servicios se caracterizan por:

- Estar integrados con los servicios de Firebase restantes, por ejemplo permiten desplegar funciones que en el caso de este proyecto reaccionan a cambios en ciertas secciones de la base de datos, y la modifican en consecuencia
- El código de las funciones a desplegar se escribe en Javascript o Typescript

- El desarrollo es simple, ya que se provee un conjunto de librerías que facilitan el desarrollo de las funciones.

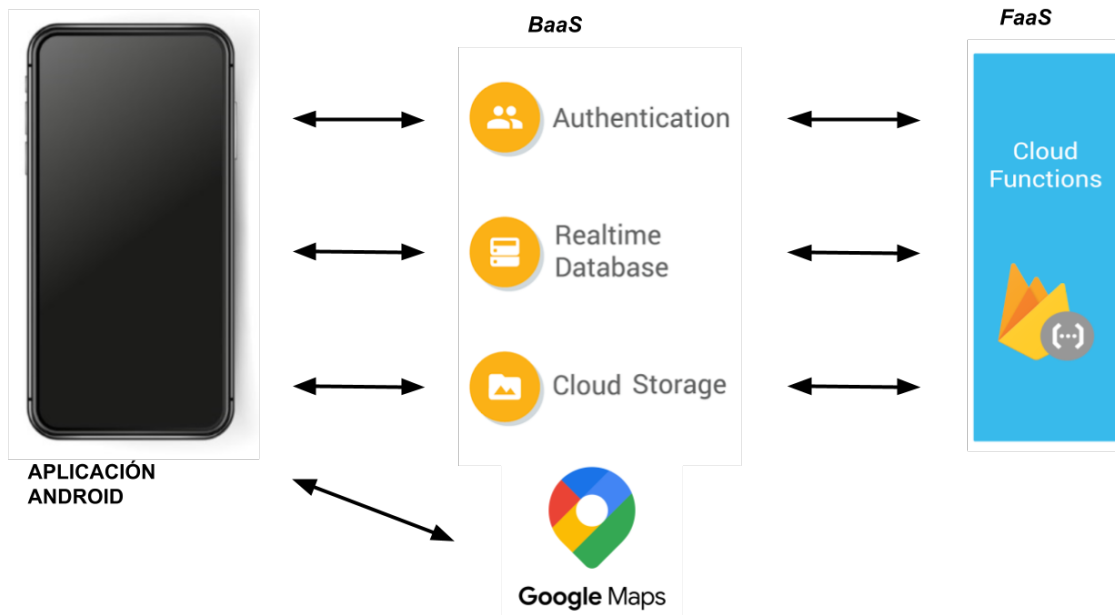


Figura 4: Esquema arquitectura

Los servicios BaaS proporcionados por Firebase serían los especificados en la sección 3.4, la cual trata las distintas herramientas ofrecidas por Firebase. También se hace uso de la API de Google Maps la cual es un BaaS ya que permite a la aplicación hacer llamadas a la API de Google Maps de forma gratuita, se explica más en detalle en la sección 3.5.

Los servicios FaaS son aquellos mencionados de Firebase Cloud Functions de los cuales se habla en la sección 3.4.3.

6. Modelo de datos

En este capítulo se describirá que modelo de datos ha sido empleado. Los datos que se mostrarán a continuación son los que se encuentran en la base de datos de Firebase. A continuación se procederá a describirlos.

6.1. Usuarios



Figura 5: Campo usuario en la base de datos

Como se puede ver en la captura cada usuario tiene un código de identificación único, y para acceder a la información del usuario deseado es necesario conocer su id de antemano. Los campos de información de cada usuario son:

- **ID del usuario (*userId*):**
El código único de cada usuario. Este es creado de forma automática por Firebase cuando se registra una cuenta nueva. Este código se utiliza principalmente en el código de la aplicación para consultar si el usuario en cuestión tiene permiso para ciertas actividades.
- **Email del usuario (*userMail*):**
El correo electrónico utilizado por el usuario al registrarse en la aplicación y que utilizará para acceder a los servicios de la misma, junto con su contraseña.
- **Nombre del usuario (*userName*):**
Nombre del usuario introducido por el mismo al registrarse en la aplicación.

- **Contraseña del usuario (*userPassword*):**

Contraseña codificada del usuario, la contraseña introducida para el registro y acceder a los servicios de la aplicación se encuentra codificada en caso de que el usuario desee modificarla más adelante.

- **Preferencias del usuario (*userPrefs*):**

Las preferencias son numéricas debido a que existe un campo denominado ETIQUETAS en la base de datos donde cada número está asociado a un campo determinado. Estas preferencias se seleccionan al registrar una cuenta o tras modificarse posteriormente desde la aplicación. Los números a los que hacen referencia se pueden ver en la figura 6.

- **Tipo de usuario (*userType*):**

Hay tres tipos de usuario especificados en la sección “Especificación de requisitos”. El usuario no registrado, el usuario registrado y el administrador. Si aparece NORMAL se refiere al usuario registrado, si aparece ADMIN es un administrador.

La información aquí mostrada tras registrarse o acceder de forma habitual a la aplicación se almacenará en la aplicación de Android en un objeto denominado User.

6.2. Etiquetas

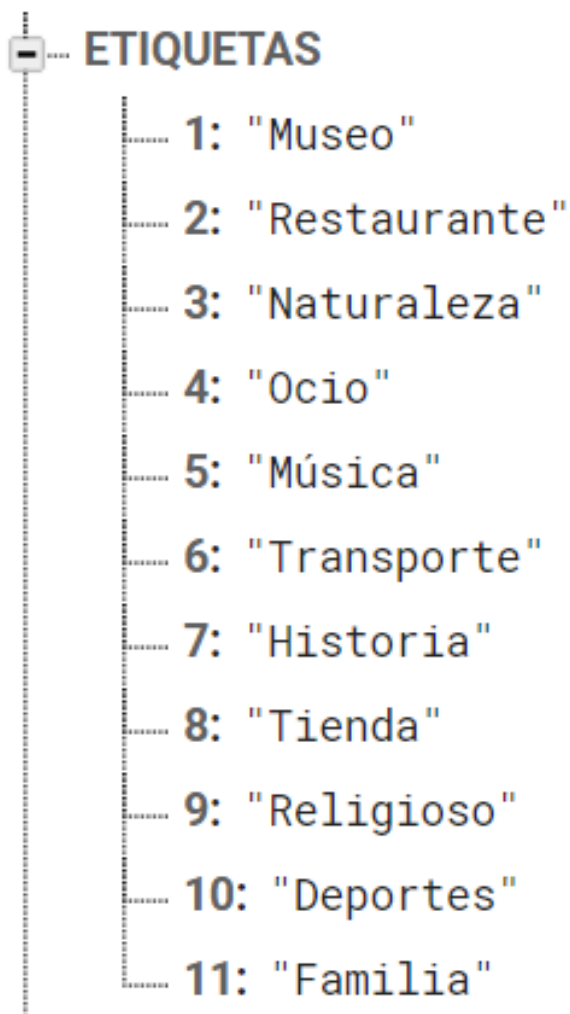


Figura 6: Campo etiquetas en la base de datos

En este campo se establecen unas etiquetas que sirven para establecer los gustos o preferencias de los usuarios y las categorías de los lugares, los usuarios pueden tener varias preferencias y los lugares pueden caer simultáneamente en varias categorías.

Al encontrarse en la base de datos si se desean actualizar o añadir nuevas etiquetas, estos cambios se reflejarían de forma inmediata en la aplicación móvil.

6.3. Lugares



Figura 7: Campo lugares en la base de datos

- **Coordenadas del lugar (*placeCoordinates*):**
Se guardan las coordenadas del lugar en cuestión.
- **ID del creador (*placeCreator*):**
Id del usuario que dio de alta el lugar en cuestión a través de la plicación, este usuario tiene permisos especiales para eliminar o editar el lugar.
- **Descripción del lugar (*placeDescription*):**
Aquí se encuentra una breve descripción del lugar que los usuarios interesados podrán leer.
- **ID del lugar (*placeId*):**
Al igual que los usuarios cada lugar tiene su propio código de identificación único. Es mucho más sencillo y rápido buscar el lugar por su ID en caso de que sea necesario hacerlo en la base de datos.
- **Nombre del lugar (*placeName*):**
El nombre del lugar dado por el usuario.
- **Ruta imágenes (*placePictures*):**
Es la ruta en la que se almacenan las imágenes del lugar en Firebase Storage, si se desea

descargar estas imágenes solo hace falta descargarlas utilizando la ruta proveída en este campo.

- **Puntuación lugar (*placeScore*):**

Es la puntuación media calculada por las calificaciones dadas por los usuarios, la media se calcula utilizando los datos almacenados en la base de datos haciendo uso de las funciones en Google Cloud Services.

- **Categorías del lugar (*placeTags*):**

Son las categorías asignadas a cada lugar por su creador a la hora de dar de alta el lugar, estos números hacen referencia a los campos de la sección etiquetas en la figura 6. Un lugar puede tener varias categorías simultaneas.

En los propios lugares hay un campo llamado *placeComments* donde se almacenan todos los comentarios que hablan del lugar en cuestión.



Figura 8: Campo lugares en la base de datos

El primer código (el marcado en rojo) es el id del usuario que ha mandado el comentario, se pone de esta forma ya que un usuario puede comentar varias veces para un mismo lugar y todos sus comentarios irían a parar a la sección con su id particular. También dentro de este campo guardamos la puntuación que el usuario ha dado al lugar, si no lo ha puntuado ya que es opcional el campo aparecerá vacío.

El segundo id (marcado en azul) es el id del comentario, cada comentario tiene su propio código de identificación, de esta forma es más sencillo encontrarlos.

- **ID comentario (*commentId*):**
Id único del comentario.
- **Texto del comentario (*commentTxt*):**
Texto del comentario.
- **ID creador comentario (*commentUserId*):**
Id del usuario que escribió el comentario.
- **Nombre creador comentario (*commentUserName*):**
Nombre del usuario que escribió el comentario, esto se debe a que el nombre del usuario aparecerá en el comentario.

Este último campo se utiliza de forma auxiliar, se utiliza principalmente para calcular la media de las puntuaciones de cada lugar. Se encuentra primero el ID del lugar que tiene las

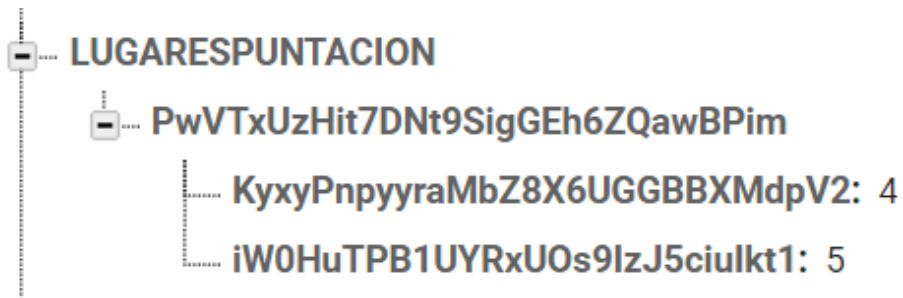


Figura 9: Campo puntación lugares en la base de datos

puntuaciones, y dentro de esta sección los IDs de los usuarios que han puntuados el lugar y la puntuación dada por cada uno.

6.4. Storage

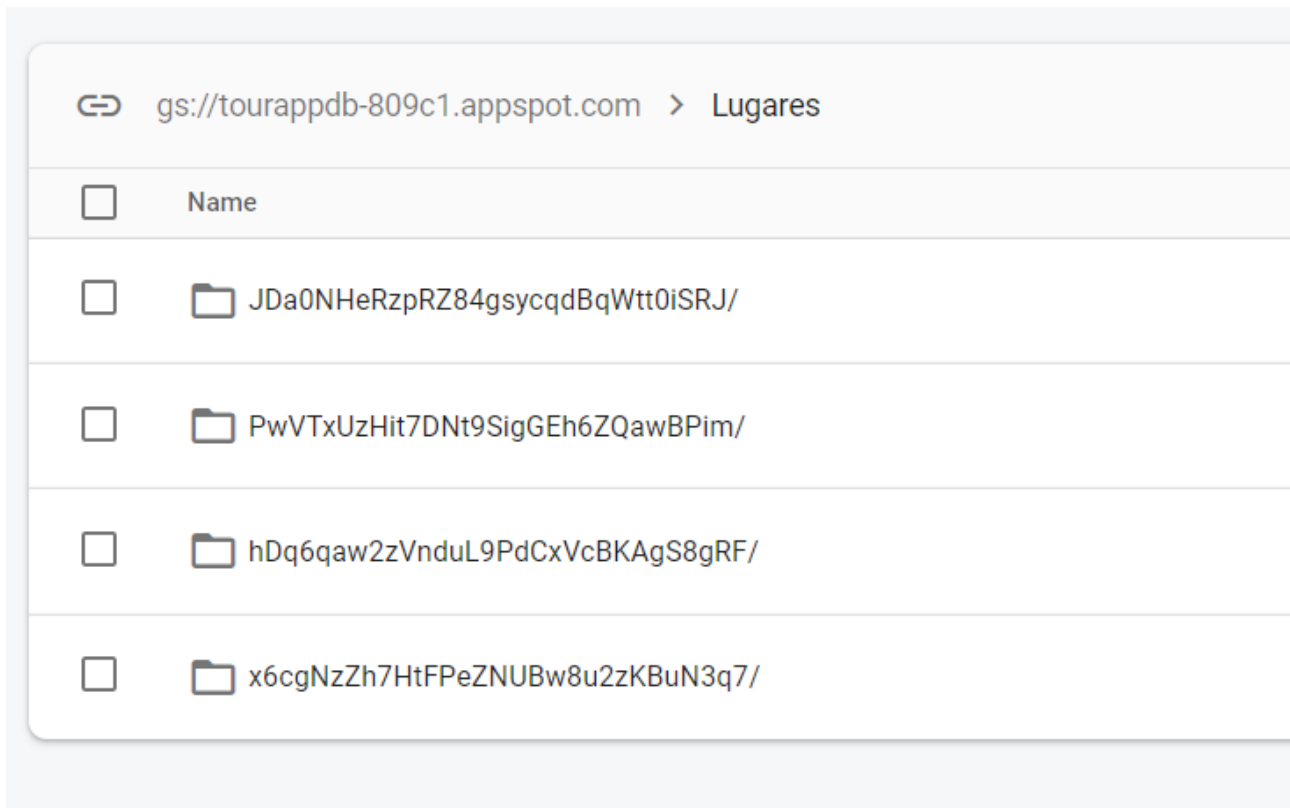


Figura 10: Carpetas de imágenes en los lugares

En estas carpetas de Firebase Storage se almacenan las imágenes de los distintos lugares subidas por los usuarios al dar de alta el lugar. Cada carpeta se encuentra identificada por el ID del lugar. La ruta se puede encontrar en el campo *placePictures* en la base de datos en la sección de Lugares (6.3).

7. Diseño de la aplicación

El diseño de una aplicación es una parte esencial de la misma, un buen diseño supone la diferencia entre poder utilizar la aplicación con facilidad o no, que la aplicación sea atractiva al usuario o no lo sea, en otras palabras determina que una la experiencia de usuario sea buena o no. También es necesario tener en cuenta que los usuarios determinan si una aplicación es fiable o no basándose únicamente en el aspecto de la misma[20], los usuario no darán muchas oportunidades a la aplicación es por esto que el diseño es un campo crucial.

A continuación se mostrarán los distintos elementos del diseño de la interfaz de la aplicación.

7.1. Elementos de la aplicación

La navegación dentro de una aplicación móvil es un elemento esencial. La forma en que estos elementos se muestran al usuario es de una gran importancia ya que esto puede influir en si el usuario quiere seguir utilizando la aplicación o no. Se mostraran a continuación los elementos gráficos utilizados para que el usuario pueda moverse entre las distintas vista e interactuar con la aplicación y la información que esta le provee.

Menú lateral: El menú lateral o deslizante se utiliza como menú principal de la aplicación. Desde el mismo es posible acceder a el resto de funcionalidades de la aplicación. Este menú se caracteriza por su icono de tres líneas en forma de hamburguesa que al pulsarlo o deslizar el dedo desde el extremo izquierdo de la pantalla hasta el centro de la misma se muestra el menú que ocupa la mitas de la pantalla. Esto provee una gran ventaja ya que ahorra mucho espacio ya que está por defecto oculto y si se desea interactuar con las opciones del mismo basta con invocarlo de la forma especificada anteriormente.

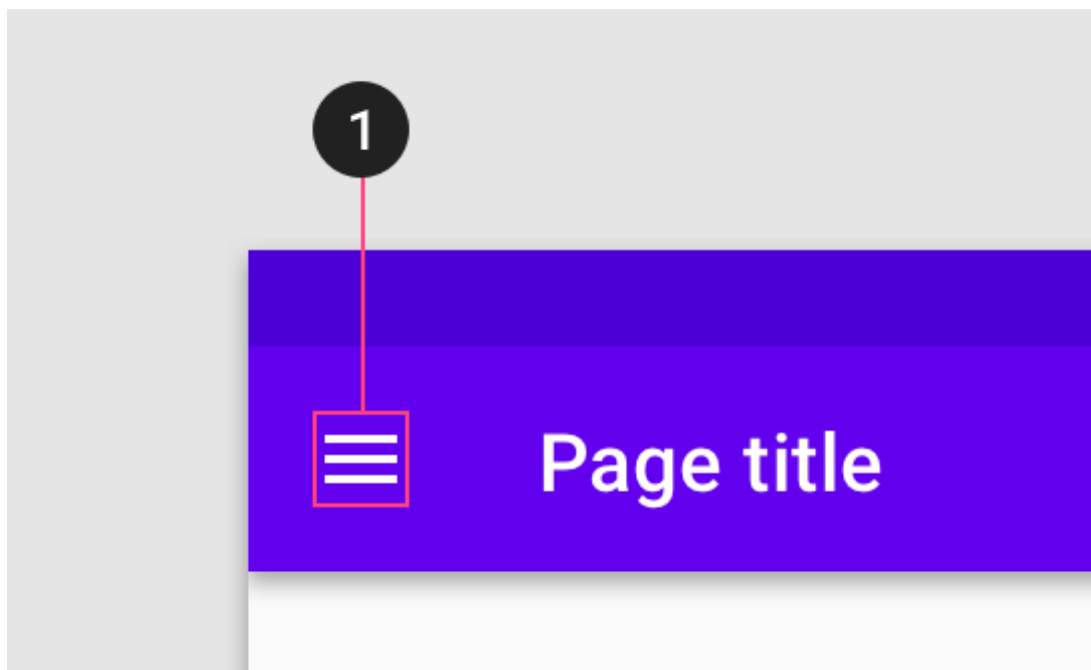


Figura 11: Botón menú lateral

Menú flotante: El menú flotante o menú popup es un tipo de menú que es muy similar al menú lateral en tanto que es muy sencillo de mostrar y ocultar y por lo tanto no piden demasiado espacio. Este tipo de menús se caracterizan por que a diferencia del anterior tienen solamente unas pocas opciones. Los menús flotantes se suelen invocar pulsando un botón y este aparecerá encima del botón desde el cual se invocó - respetando las dimensiones de la pantalla – mostrando las opciones disponibles al usuario. Generalmente este tipo de menús aparecen en las listas dinámicas de la aplicación de las cuales se hablará más adelante.

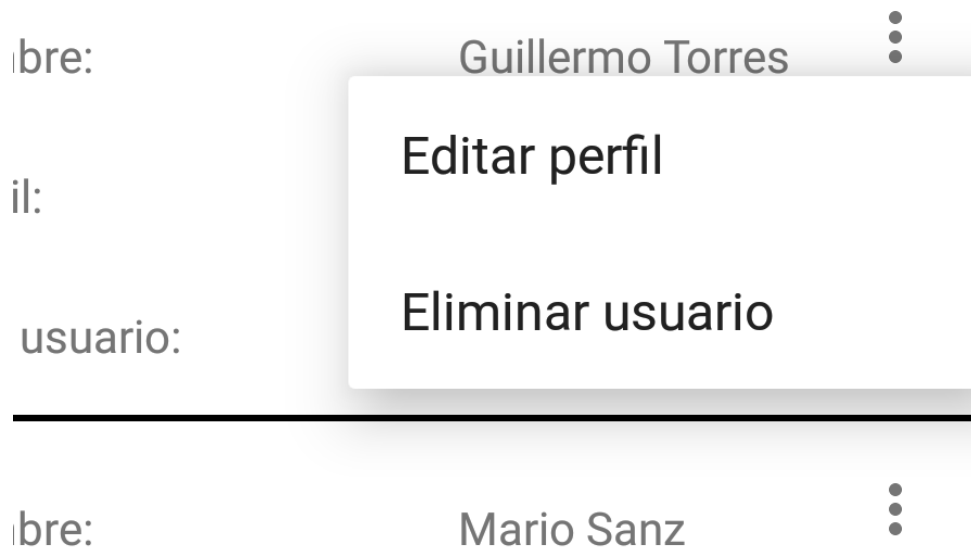


Figura 12: Muestra menú flotante

Listas dinámicas (Recycler View): Las listas dinámicas son como dice su nombre unas listas de elementos con los que el usuario puede interactuar. El programador decide como se ven los elementos individuales de estas listas y provee los datos necesarios a mostrar en las mismas, después Recycler View se encarga de crear los elementos de la lista de forma dinámica en función del diseño y los datos proveídos. Cada elemento de la lista permite que se interactué de forma independiente con el mismo y este permite un desplazamiento vertical para ver todos los elementos de la lista hasta alcanzar los límites superiores e inferiores.

En cada elemento de la lista se provee distintos tipos de información y otros elementos interactivos como botones laterales los cuales invocan un menú flotante mencionado anteriormente. En el caso de la figura 13 muestro el nombre del lugar, su puntuación general y una vista anticipada de su descripción.

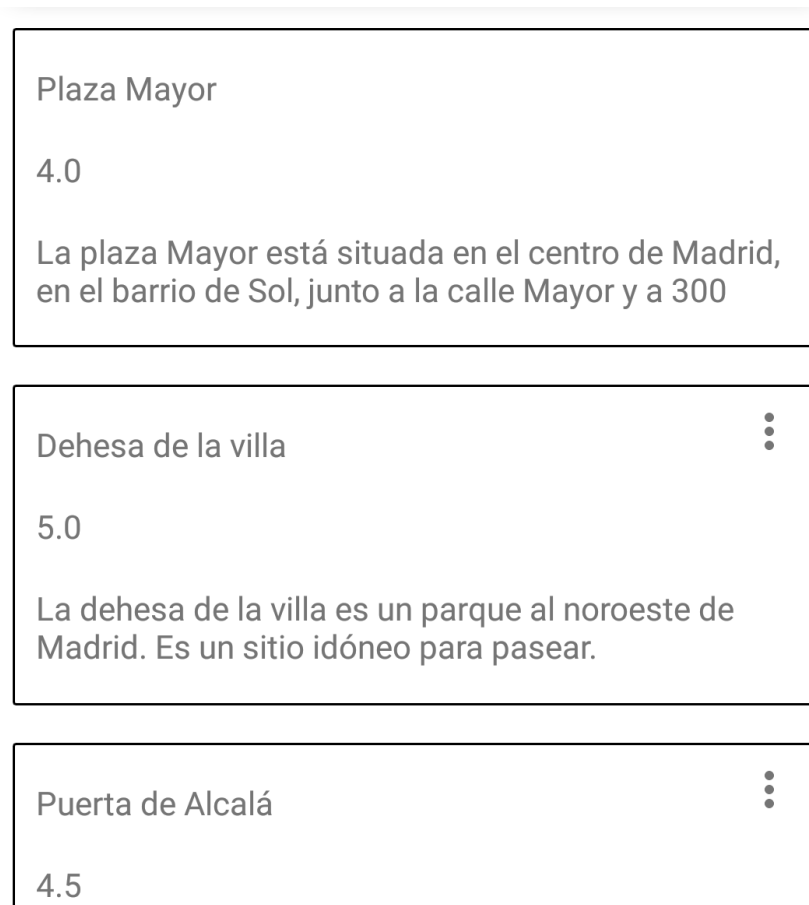


Figura 13: Muestra lista dinámica

Botones: En la aplicación hay distintos tipos de botones, que en función de en que pantalla o tipo de vista se encuentren pueden dar lugar a un tipo de interacción u otra desde llevar al usuario a otra pantalla o invocar menús, como en el caso de el menú lateral y el menú flotante.

Ventanas flotantes (ventanas popup): Este tipo de ventanas suelen aparecer para cerciorarse de que el usuario está seguro de una decisión u otros motivos. Es un tipo de ventana muy práctico ya que por lo general no se requiere interactuar mucho tiempo con ese elemento en cuestión, entonces se invoca la ventana, el usuario interactúa con los elementos de la misma - los cuales han sido decididos previamente por el programador – y una vez terminada la interacción esta desaparece permitiendo ahorrar espacio de pantalla y tener una vista más limpia.

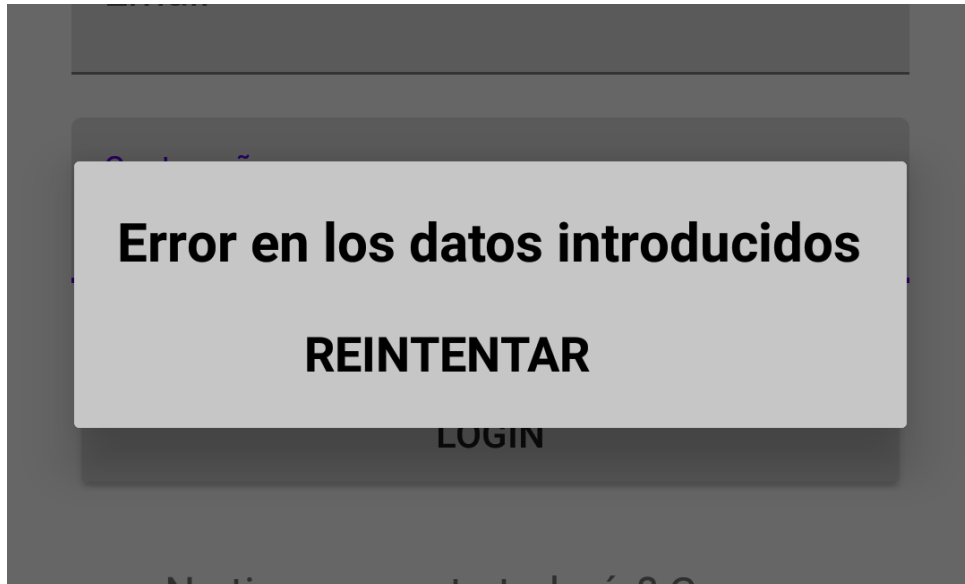


Figura 14: Ejemplo ventana flotante

Barra de puntuación: Este elemento visual sirve para ver y establecer una puntuación determinada, como se puede ver en la figura 15 consiste en un conjunto de estrellas en las cuales se puede establecer el valor de una puntuación e incluso interactuar con las mismas para establecer un valor deseado. En la aplicación se ha establecido que la barra de puntuación tenga cinco estrellas y por lo tanto las puntuaciones irán de cero a cinco.

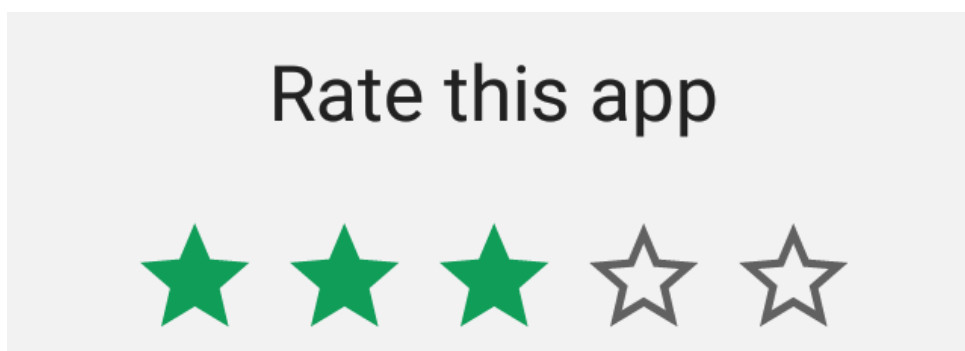


Figura 15: Ejemplo barra de puntuación

7.2. Tipografía

Para esta aplicación se ha utilizado la familia tipográfica (conjunto de tipografías con un diseño común) que venía por defecto, esta se llama *Roboto*. Esta ha sido seleccionada debido a que tiene un diseño profesional que se ve bien con la estética de la mayoría de las aplicaciones. Hay otro tipo de fuentes cuyo diseño podría considerarse más profesional o más juvenil pero debido a que no encajaban con la imagen que se quería tener de la aplicación y a que se desea que todo el mundo pueda utilizarla no únicamente adultos o jóvenes se optó por esta.

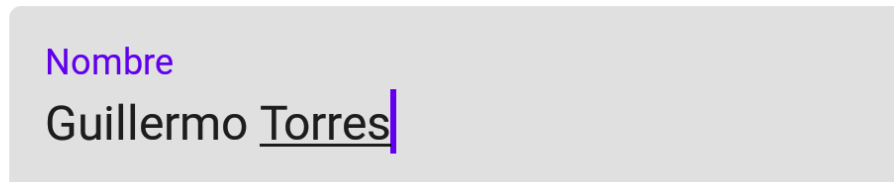
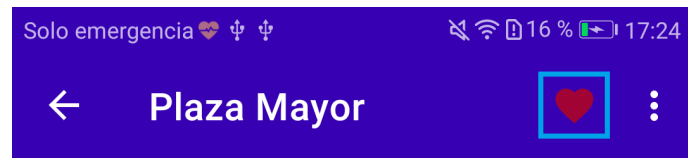


Figura 16: Muestra tipografía

7.3. Objetivos del diseño de la aplicación

El objetivo del diseño de esta aplicación ha sido el presentar al usuario con el diseño más simple y limpio posible. Se busca que el usuario de un vistazo rápido sea capaz de ver cual es la información más relevante, en lugar de ofrecerle mucha información se le muestra lo necesario ya que hay situaciones en las que mucho puede ser contraproducente.

Esto se puede ver en la pantalla de información sobre el lugar (figura 17) donde se puede contemplar lo primero el nombre del lugar tanto en la barra superior como en el primer campo de texto de la pantalla, después debajo del texto se contempla la imagen del lugar (en caso de que el usuario que lo dio de alta deseara subir una) y seguido de esto una descripción corta y la puntuación del lugar.



Plaza Mayor



La plaza Mayor está situada en el centro de Madrid, en el barrio de Sol, junto a la calle Mayor y a 300 metros de la Puerta del Sol.



Figura 17: Pantalla información lugar

El usuario puede ver todo aquello que necesita en poco tiempo. También en pantallas como los datos del usuario (figura 18) se ve la información del mismo a primera vista, siguiendo la filosofía mencionada anteriormente.

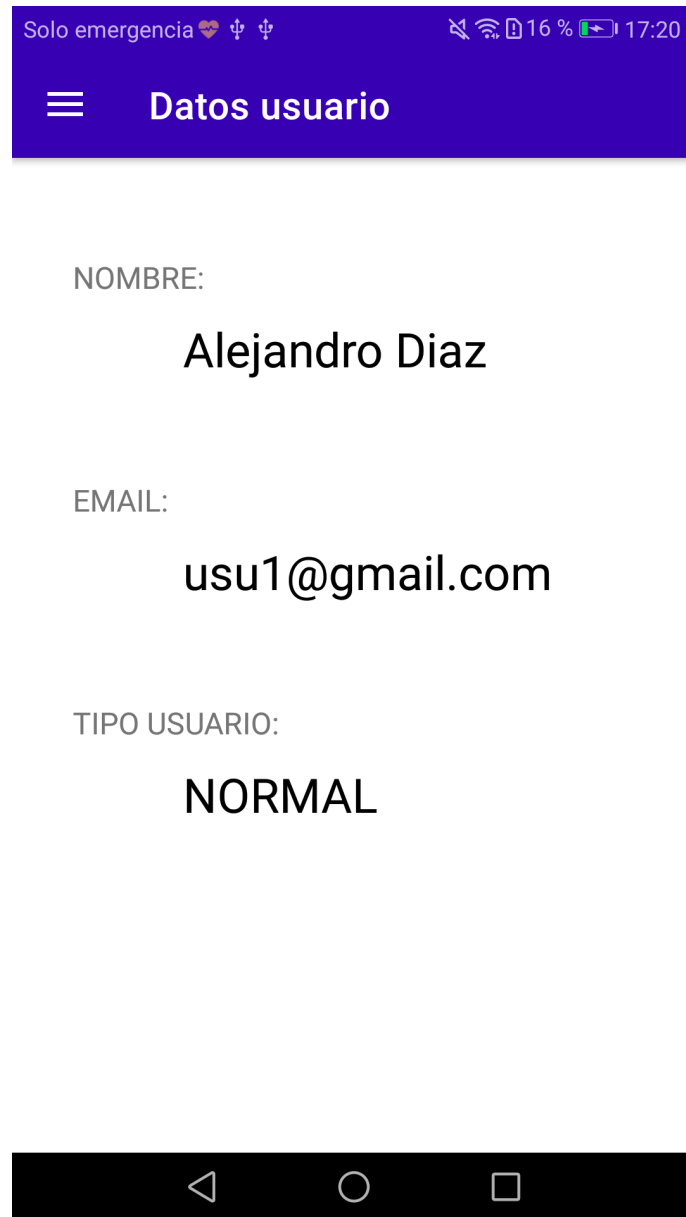


Figura 18: Pantalla datos usuario

En las listas dinámicas (figura 19) se sigue la misma idea, a la hora de enseñar los datos de los lugares en las listas, se enseña el nombre, la puntuación y un adelanto de la descripción, lo suficiente para que el usuario desee leer más de la misma y por lo tanto consultar la información del lugar.

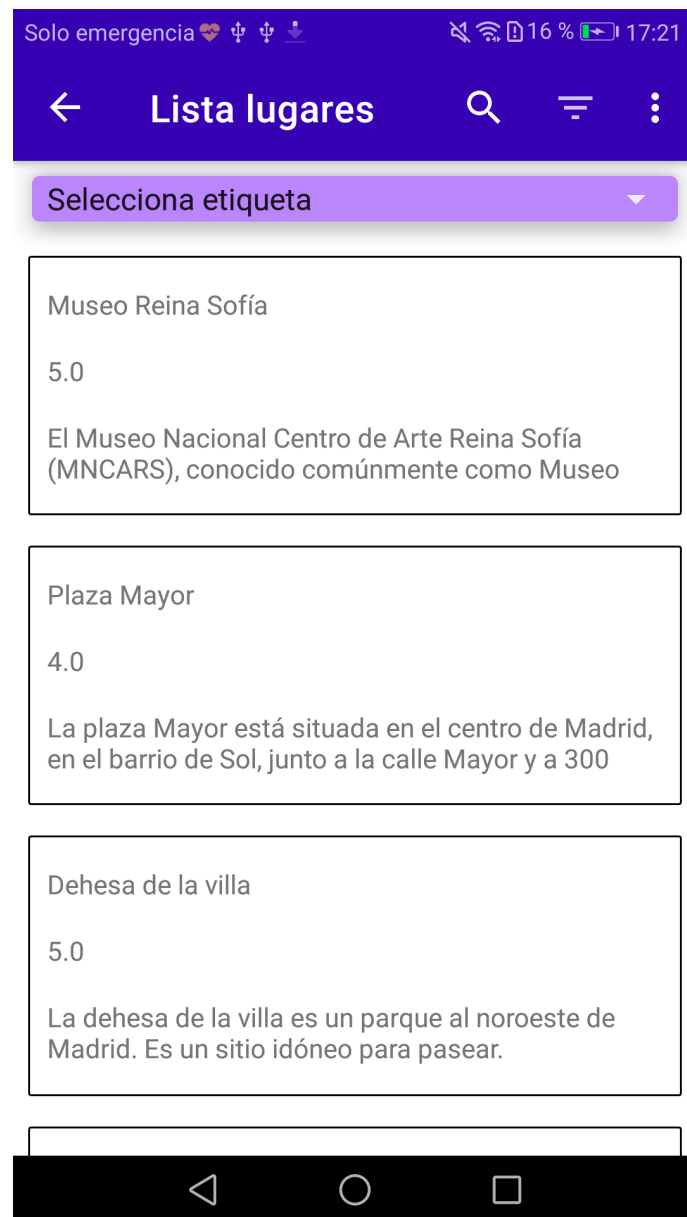


Figura 19: Pantalla de lista de lugares

La pantalla de listas de lugares muestra una serie de filtros, hay dos categorías:

Filtros por etiqueta: Filtran los lugares por el tipo de etiquetas asignadas al mismo (museo, restaurante, deportes, ...) y se encuentran en la barra morada justo de bajo de la barra superior.

Filtros por datos: Filtran los lugares en función de datos como puntuación, o si han sido creados por el usuario en cuestión.

Ambos han sido separados en dos zonas en la misma pantalla ya incluso si ambos sirven para filtrar ambos lugares, lo hacen en función de categorías distintas y se considera que esto habría hecho innecesariamente más complejo el uso de la aplicación. Al separar ambos se facilitó al usuario como deseaba filtrar.

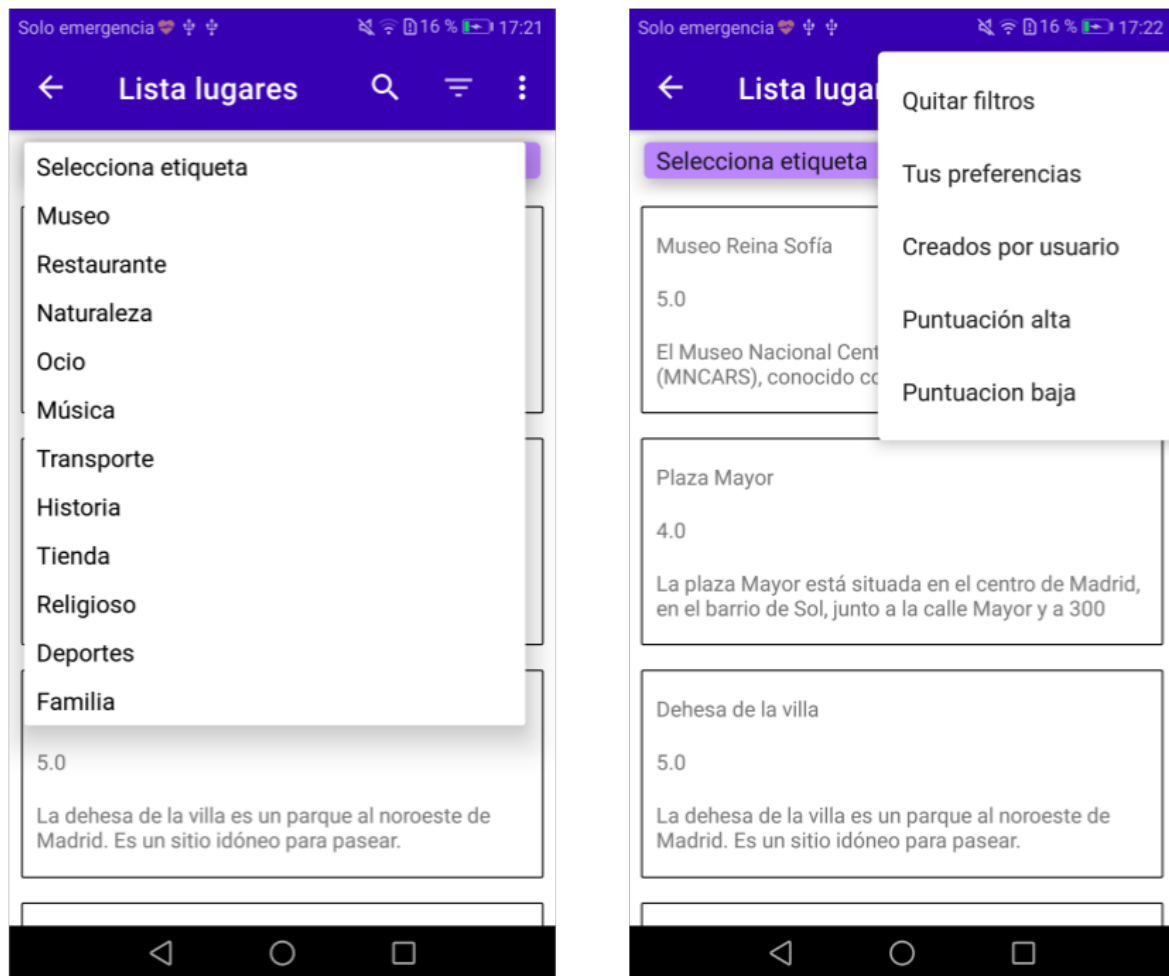


Figura 20: Filtros de listas de lugares

7.4. Iconos

La elección de iconos de la aplicación sigue la misma filosofía que los objetivos seguidos anteriormente, que se da al usuario únicamente la información justa y necesaria. Los iconos cumplen un papel muy importante en esto ya que con una elección adecuada se puede ahorrar mucho espacio de la vista y tiempo del usuario ya que intuitivamente puede saber que hace cada cosa.

Los iconos de la barra superior permiten al usuario realizar una serie de opciones sin necesidad de texto y por lo tanto da lugar a una vista más limpia y elegante. Como se contempla en la figura 21 hay un icono de búsqueda, uno de filtro y otro de opciones extras. Al ser muy similar a el diseño de otras aplicaciones de Android basta con que el usuario tenga un mínimo de experiencia para saber en que consiste cada opción sin tener que tocar nada.

Los iconos del menú lateral cumplen un objetivo similar, si es cierto que hay un texto explicando en que consiste cada opción el icono completa la información del mismo. Si el usuario no comprende el texto puede mirar el icono para hacerse una idea y viceversa.

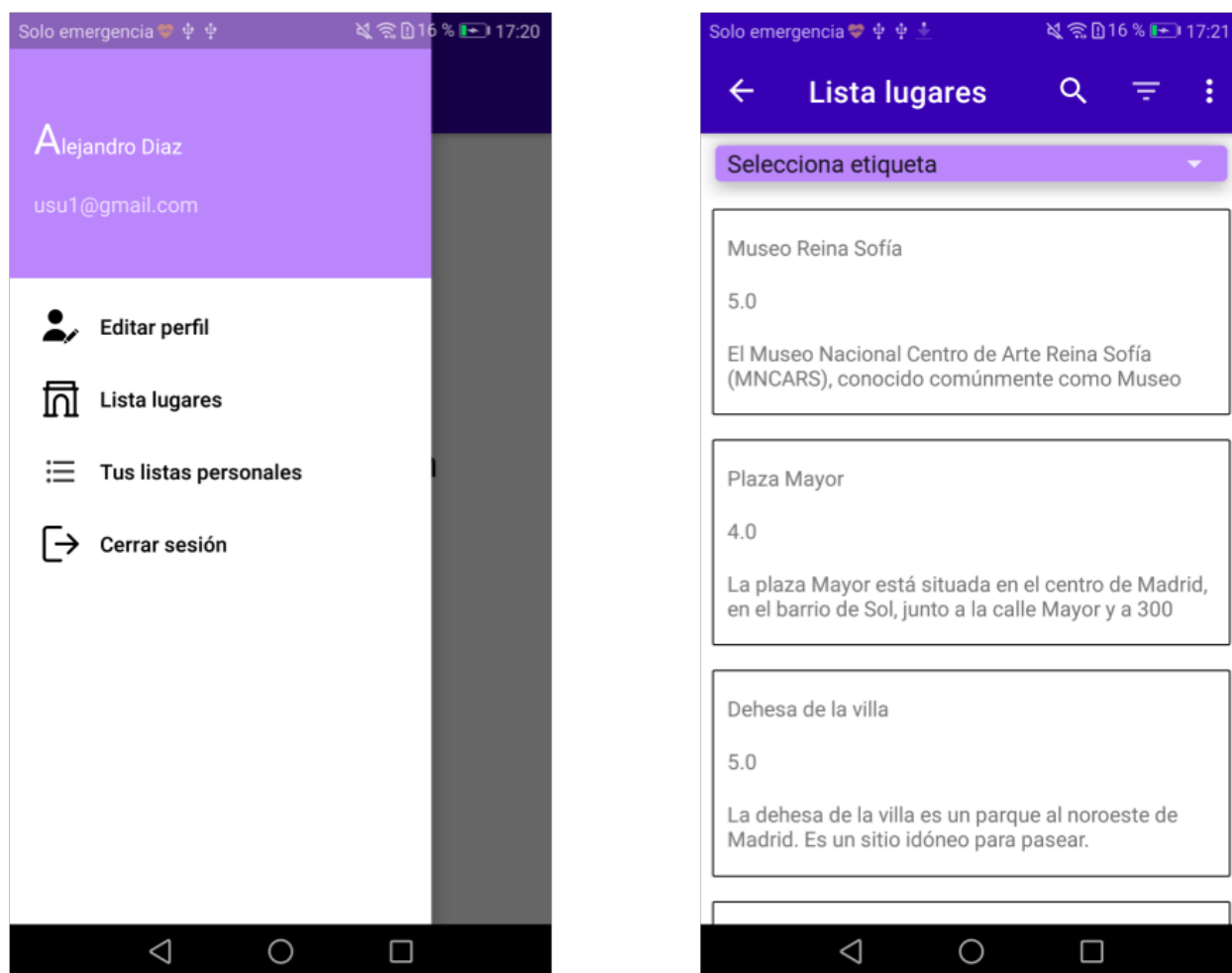


Figura 21: Iconos de muestra

8. Implementación

En esta sección se procederá a describir la implementación realizada en la aplicación móvil y las funciones desplegadas utilizando el servicio de Firebase Cloud Functions (sección 3.4.3).

8.1. Implementación en Android

Se desarrollará como se ha realizado la implementación de la aplicación de Android. Primero se mostrarán de los componentes básicos de una aplicación, estos son.

- **Actividades**
- **Fragmentos**
- **Diseño de interfaces**
- **Servicios**

Toda aplicación tiene un archivo de tipo XML llamado `AndroidManifest.xml` [7]. Es en este archivo donde se declara que permisos va a solicitar la aplicación al usuario y que va a necesitar para funcionar correctamente.

También se declaran que actividades y servicios se encuentran en la aplicación y cual sería su utilidad. En resumidas cuentas el manifiesto funcionaría como una carta de presentación para el dispositivo donde se va a instalar la aplicación.

8.1.1. Actividades

Las actividades son componentes de la aplicación que contiene la pantalla con la que los usuarios pueden interactuar, una aplicación por lo general suele constituir entre una o varias aplicaciones vinculadas entre sí.

Las actividades pasan por una serie de estados definidos en el ciclo de vida de la actividad [6], ahora explicaré los estados o eventos que componen el ciclo de vida de la actividad.

1. **onCreate()**: Se llama cuando se crea la actividad.
2. **onStart()**: Se llama cuando la actividad se hace visible al usuario.
3. **onResume()**: Se llama cuando la actividad es interactuable con el usuario.
4. **onPause()**: Se llama cuando la actividad está en pausa y se retoma la actividad anterior.
5. **onStop()**: Se llama cuando la actividad ya no es visible al usuario.
6. **onDestroy()**: Se llama antes de que la actividad sea destruida por el sistema.
7. **onRestart()**: Se llama cuando la actividad ha sido parada y se está reiniciando.

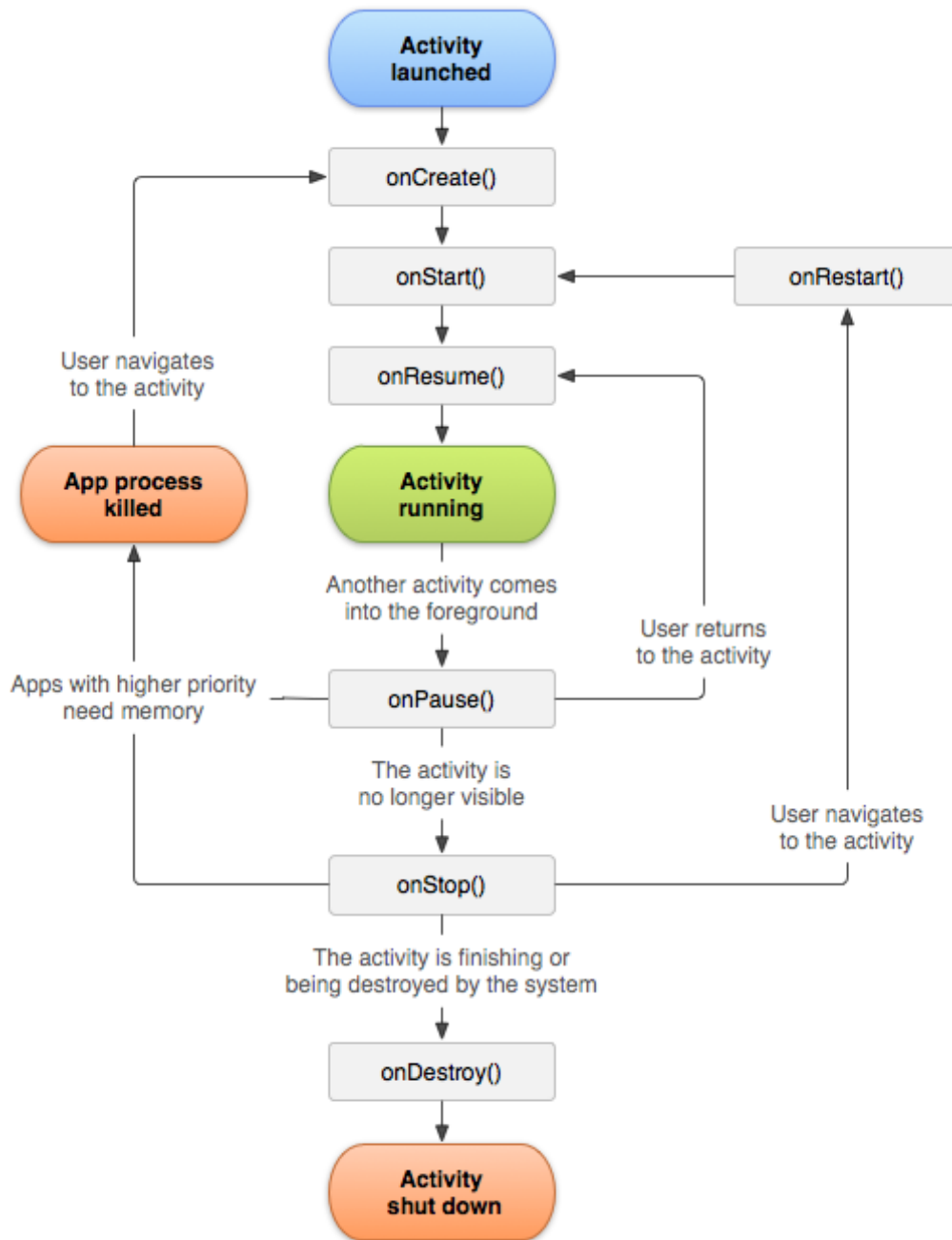


Figura 22: Diagrama del ciclo de vida de la actividad.

Una aplicación puede tener una o varias actividades, cada una con una funcionalidad distinta, para desplazarse entre ellas se utilizan unos objetos denominados intentos, los intentos permiten intercambiar objetos entre las actividades de la aplicación pudiendo así pasar información relevante de una actividad a otra.

8.1.2. Fragmentos

Una actividad es básicamente un contenedor para las vistas, el problema se da cuando una actividad se muestra en pantallas de distinto tamaño como la de un teléfono y una Tablet, para esto se encuentran los fragmentos. Los fragmentos funcionan como una especie de ‘mini actividades’ cada una con su propio conjunto de vistas.

Un fragmento ha de estar asociado a una actividad y una actividad puede tener asociados a la misma ninguno o varios fragmentos. Los fragmentos tienen su propio ciclo de vida [12] el cual se ve afectado por el ciclo de vida de la actividad en la que se encuentran integrados. Los estados únicos a los fragmentos son:

1. **onAttach()**: Se llama cuando el fragmento ha sido asociado con la actividad.
2. **onCreateView()**: Se llama cuando se crea la vista del fragmento.
3. **onActivityCreated()**: Avisa cuando *onCreate()* de la actividad ha finalizado.
4. **onDestroyView()**: Cuando la vista del fragmento está siendo eliminada.
5. **onDetach()**: Se llama cuando el fragmento se desune de la actividad.

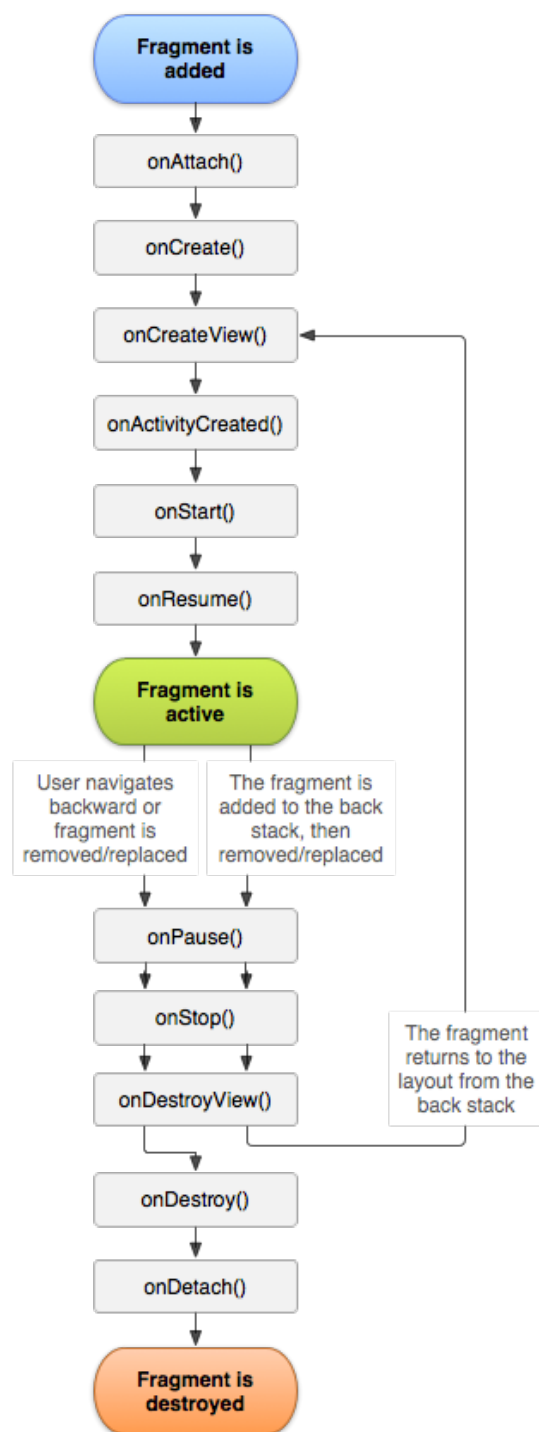


Figura 23: Diagrama del ciclo de vida del fragmento.

8.1.3. Diseños de interfaces

Los diseños de interfaces se encargan de mostrar los elementos gráficos de la aplicación, con los que el usuario va a interactuar, es a través de las actividades y fragmentos que se les proporciona que información mostrar y como reaccionar al interactuar con el usuario, pero es a través de estos diseños que se definen los diseños.

Estos diseños se definen en documentos de formato XML [9], ahí se incluyen botones, cajas de texto, cajas de dialogo, menús, etc. Estos componentes van dentro de un conjunto de vistas marcados por unas etiquetas denominadas *layouts*.

Hay muchos tipos de *layouts* y cada uno describe como se distribuirán los componentes dentro de sí:

- Linear Layout.
- Table Layout.
- Frame Layout.
- Relative Layout.
- Constraint Layout

8.1.4. Servicios

Los servicios son componentes de una aplicación en Android que se ejecutan en segundo plano y estos pueden continuar activos incluso si la aplicación está en segundo plano o cerrada. Estos pueden interactuar con proveedores de servicios, reproducir audios, o manejar transacciones de red.

Existen tres tipos de servicios:

- **Servicio en primer plano:** Los servicios en primer plano son aquellos que el usuario puede notar incluso si no se encuentra interactuando con la aplicación directamente, por ejemplo las aplicaciones de música se pueden mantener activas incluso si el teléfono se encuentra bloqueado.
- **Servicio en segundo plano:** Un servicio en segundo plano es aquel que el usuario no nota, si por ejemplo guarda o escribe en archivos del teléfono.
- **Enlace:** Es muy parecido a los servicios en segundo plano, ofrece una interfaz *cliente-servidor* que hace que el servicio envíe solicitudes y reciba resultados.

Esta es la estructura que presenta por lo general un proyecto de una aplicación Android. Ahora se mostrará la implementación de las funcionalidades especificadas en la subsección 4.2 y la subsección 4.3.

M-1: Crear cuenta

Al iniciar la aplicación esta pedirá al usuario que introduzca un usuario y una contraseña, en caso de no disponer de cuenta ofrecerá la opción de crear una. Pedirá al usuario que introduzca un nombre, un correo electrónico y una contraseña. Estos campos se utilizarán para crear al usuario y darlo de alta, el usuario se guardará en la variable *user* de tipo User, como se puede ver en la figura 24.

```
fun onRegisterClick() {  
  
    model.addNewUser(  
        userName = input_name.text.toString(),  
        userType = Constants.NORMAL,  
        userMail = input_email.text.toString(),  
        userPassword = input_password.text.toString()  
    )  
  
}
```

Figura 24: Obtención de datos

```
/**  
 * Funcion para autenticar al usuario y añadirlo en DB  
 */  
private fun onSignUp() {  
  
    var usrPassAux = user.userPassword  
  
    if(user.userMail.isNotEmpty() && user.userPassword.isNotEmpty()) {  
        mFirebaseAuth.createUserWithEmailAndPassword(user.userMail, user.userPassword)  
            .addOnCompleteListener {result ->  
  
                if(result.isSuccessful) {  
                    Log.v( tag: "FIREBASE_LOGIN", msg: "SUCCESS_LOGIN")  
                    //uploadUserData(usrPassAux)  
                    loginAfterSignUp(usrPassAux)  
                }  
                else {  
                    flagCreate.value = false  
                    Log.v( tag: "FIREBASE_LOGIN", msg: "ERROR, BAD CREDENCIALES")  
                }  
            }  
  
        //mFirebaseAuth.signInWithEmailAndPassword(userMail,userPassword)  
    }  
}
```

Figura 25: Alta del usuario

En la figura 25 se puede ver como una vez creado el objeto tipo User se da de alta en la base de datos. Posteriormente dado de alta en el sistema de Firebase realizara un inicio de sesión automático con el usuario recién creado.

M-2: Iniciar sesión

Al abrir la aplicación esta pedirá al usuario que introduzca su correo electrónico asociado a su cuenta y su contraseña correspondiente, estos se almacenaran en unas variables que se utilizaran posteriormente para permitir al usuario acceder a los servicios de la base de datos. Se puede ver en la figura 26.

Si algún campo se encuentra vacío mostrará un error en la pantalla de la aplicación, en caso contrario procederá a comprobar la validez de los datos.

```
fun onLoginClick(v: View){
    when {
        email.text.toString().isEmpty() || password.text.toString().isEmpty() -> {
            if (email.text.toString().isEmpty()) tiEmail.error = "Campo obligatorio" else tiEmail.error = null
            if (password.text.toString().isEmpty()) tiPassword.error = "Campo obligatorio" else tiPassword.error = null
        } else -> {
            //cl_loading_user.visibility = View.VISIBLE
            v.hideKeyboard()
            tiEmail.error = null
            tiPassword.error = null
            model.loginData.email = email.text.toString() //Guardo el valor introducido (email) en el viewModel
            model.loginData.password = password.text.toString() //Guardo el valor introducido (password) en el viewModel
            model.onLoginClick()
        }
    }
}
```

Figura 26: Lectura de datos del usuario

```
mFirebaseAuth?.signInWithEmailAndPassword(loginData.email, loginData.password)
?.addOnCompleteListener { it: Task<AuthResult>
    if(it.isSuccessful) {
        //loadClient(mFirebaseAuth.uid.toString())
        Log.v( tag: "FIREBASE_LOGIN", msg: "SUCCESS_LOGIN")
        loginNotifv.value = true
    }
    else {
        //SharedPreferencesManager.setSomeBooleanValues(Constants.SAVELOGIN, false)
        Log.v( tag: "FIREBASE_LOGIN", msg: "ERROR, BAD CREDENCIALES")
        loginNotifv.value = false
    }
}
```

Figura 27: Accediendo a los servicios de Firebase

Se puede ver en la figura 27 que una vez obtenidos los datos se intentará acceder al los servicios de Firebase para poder acceder a la base de datos donde se encuentra almacenada toda la información sobre el usuario y los distintos lugares.

```
private fun loginFinished(loginSuccess : Boolean) {  
  
    if (loginSuccess){//Si hace login correctamente  
  
        val intent = Intent( packageContext: this, MainActivity::class.java)//Entramos a pantallan principal  
        intent.putExtra( name: "MyUser", user)  
        //intent.putExtra(Constants.USERS, model.user)  
        startActivity(intent)  
        finish()  
        //Cerramos la actividad del login  
    } else //Si no hace login  
  
        email.text.clear()  
        password.text.clear()  
  
        val dialogBuilder = AlertDialog.Builder( context: this)  
        val dialogView = layoutInflater.inflate(R.layout.dialog_error_login, root: null)  
        val b = dialogBuilder.setView(dialogView).create()  
  
        dialogView.btnAceptarDialog.setOnClickListener { it: View!  
            b.dismiss()  
        }  
  
        b.setCancelable(false)  
        b.show()  
    }  
}
```

Figura 28: Fin de login

En caso de que el login se haya realizado sin ningún problema se llevará al usuario a la pantalla principal, donde tendrá acceso al resto de funciones de la aplicación. En caso contrario se mostrará al usuario una ventana flotante (*dialogView* en la figura 28) que indicará que los datos introducidos no son válidos.

M-3: Cerrar sesión

En la actividad principal (*MainActivity.kt*) al seleccionar la opción del menú lateral de cerrar sesión tal y como se ve en la figura 29. Ejecutará la función para cerrar la sesión del usuario y salir de la aplicación.

```
override fun onNavigationItemSelected(item: MenuItem): Boolean {  
    when (item.itemId) {  
        R.id.nav_edituser -> {  
            useredit = user  
            nav_host_fragment.view?.let { view ->  
                Navigation.findNavController(view).navigate(R.id.action_userDataFragment_to_userOptionsFragment)  
            }  
        }  
        R.id.nav_userlist -> {  
            nav_host_fragment.view?.let { view ->  
                Navigation.findNavController(view).navigate(R.id.action_userDataFragment_to_listAuxFragment)  
            }  
        }  
        R.id.nav_placelist -> {  
            nav_host_fragment.view?.let { view ->  
                val bundle = Bundle()  
                bundle.putBoolean("CustomList", false) //Descargamos toda la lista de lugares  
                Navigation.findNavController(view).navigate(R.id.action_userDataFragment_to_placeListFragment, bundle)  
            }  
        }  
        R.id.opt_edituser -> {  
            nav_host_fragment.view?.let { view ->  
                Navigation.findNavController(view).navigate(R.id.action_listAuxFragment_to_editUserFragment)  
            }  
        }  
        R.id.opt_deleteuser -> {  
            model.deleteUserData(useredit, useredit.userPassword, user)  
        }  
        R.id.nav_user_custom_lists -> {  
            nav_host_fragment.view?.let { view ->  
                Navigation.findNavController(view).navigate(R.id.action_userDataFragment_to_userListOfListsFragment)  
            }  
        }  
        R.id.nav_logout -> showDialogLogout(item)  
    }  
    return true  
}
```

Figura 29: Opción menú *Cerrar sesión*

Esta función (figura 30) mostrará una ventana flotante (*dialogView*) en la que se pregunta al usuario si desea salir de la aplicación. Si selecciona btnAceptar cerrará los servicios de Firebase y llevará al usuario de nuevo a la pantalla del login. Si selecciona btnCerrar simplemente se cerrará la ventana flotante *dialogView*.

```

/**
 * Funcion que infla una modal para desloguear al usuario de la app
 */
private fun showDialogLogout(itemId: MenuItem) {

    val dialogBuilder = AlertDialog.Builder(context: this)
    val dialogView = layoutInflater.inflate(R.layout.dialog_logout, root: null)
    val b = dialogBuilder.setView(dialogView).create()

    dialogView.btnCerrar.setOnClickListener{ it: View!
        itemId.is-checked = false
        b.dismiss()
    }

    dialogView.btnAceptar.setOnClickListener{ it: View!
        logout()
        startActivity(Intent(packageContext: this, LoginActivity::class.java))

        b.dismiss()
        finish()
    }

    b.setCancelable(false)
    b.show()
}

```

Figura 30: Función cierre sesión

M-4: Selección de preferencias

Al dar de alta una cuenta el usuario tendrá que seleccionar una serie de preferencias para que la aplicación pueda filtrar en función de sus gustos. En la figura 31 se ve que cuando se selecciona el objeto de tipo `RadioButton` (el cual es un tipo de botón como dice su nombre) la opción seleccionada se agrega a una lista llamada `arrayRadioBtn` donde se almacenan las preferencias seleccionadas.

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
    listTags?.get(position)?.let { it: String  
        this.parent?.let { parent ->  
  
            val select = listTagsSelected?.get(position) == true  
            holder.bind(it, select, parent)  
  
            val radioBtn = holder.getRadioBtn()  
            radioBtn.setOnClickListener { it: View!  
                listTagsSelected!![position] = !listTagsSelected?.get(position)!!  
                Log.d( tag: "TAGS_CLICKED", msg: "Ha sido pulsado ${radioBtn.text}")  
                arrayRadioBtn[position].isChecked = listTagsSelected?.get(position) == true  
  
                notifyItemChanged(position)  
            }  
  
            arrayRadioBtn.add(radioBtn)  
        }  
    }  
}
```

Figura 31: Selección preferencias

En la imagen 32 se ve que se obtienen las preferencias en forma de string este se sube a la base de datos. En la variable `prefRef` se guarda la dirección dentro de la propia base de datos donde se desea almacenar las preferencias.

```

private fun uploadUserPrefs(userPass: String) {
    val prefRef = FirebaseDatabase.getInstance().getReference(Constants.U

    val msg = getMsg()
    user.userPrefs = msg
    user.arrayPrefs = getPrefs(msg)

    prefRef.setValue(msg).addOnCompleteListener { it: Task<Void!>
        if(it.isSuccessful) {
            Log.v( tag: "FIREBASE_BBDD", msg: "SUCCESS_TAGS_UPLOAD")
            flagCreate.value = true
            //loginAfterSignUp(userPass)
        }
    }
}
}

```

Figura 32: Dada de alta preferencias

M-5: Ver recomendaciones

La aplicación descarga de la base de datos una colección de lugares, los cuales el usuario podrá filtrar posteriormente. En la imagen 33 se ve como se descarga la información de los lugares de la base de datos.

Se provee al código con la ruta de la información de los lugares en la base de datos y este descarga la información de los mismos, aunque se respeta un máximo de lugares a descargar establecido en la constante *MAX DATABASE ITEMS*. Solo se descargarán otros *MAX DATABASE ITEMS* cuando el usuario toca el fondo de la pantalla (ya que los lugares se presentan como una lista).

Se puede ver en la imagen 34 los criterios a la hora de filtrar los lugares, estos son: (*NINGUNO, CREADOS POR USUARIO, PUNTUACIÓN ALTA, PUNTUACIÓN BAJA, PREFERENCIAS DEL USUARIO*).

```

fun getPlaceList(listCodes: ArrayList<String>) {

    listPlace.clear()
    listCodes.clear()
    val placeRef = FirebaseDatabase.getInstance().getReference(Constants.PLACES)
    val userRef = FirebaseDatabase.getInstance().getReference(Constants.USERS)
    var placeAux: Place

    var customList = false
    if(!listCodes.isEmpty())
        customList = true

    mListenerPlace = object : ValueEventListener {
        override fun onCancelled(error: DatabaseError) {
            Log.v( tag: "FIREBASE_BBDD_USER", msg: "ERROR AL DESCARGAR INFO")
        }

        override fun onDataChange(snapshot: DataSnapshot) {

            while(placeIndex < snapshot.childrenCount && descargas < Constants.MAX_DATABASE_ITEMS) {

                //Descargamos el lugar
                placeAux = getPlaceData(snapshot.children.elementAt(placeIndex))
                listPlace.add(placeAux)

                placeIndex++
                descargas++

                setPlaceList()
            }
        }
    }
}

```

Figura 33: Descarga información de lugares

```

enum class FilterCategory {
    NONE, USERID, HIGHRATE, LOWRATE, USERPREFS
}

```

Figura 34: Criterios de filtros

En la imagen 35 se pueden ver las opciones del menú para filtrar los lugares según los filtros descritos anteriormente, o si simplemente no se desea que se filtre absolutamente nada.

```
R.id.filter_by_current_user-> {  
    categoryFilter = PlaceListViewModel.FilterCategory.USERID  
    viewModel.filterByCategory(positionSpinner, categoryFilter)  
}  
R.id.filter_high_rate->{  
    categoryFilter = PlaceListViewModel.FilterCategory.HIGHRATE  
    viewModel.filterByCategory(positionSpinner, categoryFilter)  
}  
R.id.filter_low_rate->{  
    categoryFilter = PlaceListViewModel.FilterCategory.LOWRATE  
    viewModel.filterByCategory(positionSpinner, categoryFilter)  
}  
R.id.filter_by_prefs->{  
    categoryFilter = PlaceListViewModel.FilterCategory.USERPREFS  
    viewModel.filterByCategory(positionSpinner, categoryFilter)  
}  
R.id.none_filter-> {  
    categoryFilter = PlaceListViewModel.FilterCategory.NONE  
    viewModel.filterByCategory(positionSpinner, categoryFilter)  
}
```

Figura 35: Menú filtración

M-6: Darse de baja

```
fun deleteAccountPopup() {
    val dialogBuilder = AlertDialog.Builder(context: this)
    val dialogView = LayoutInflater.inflate(R.layout.dialog_delete_account, root: null)
    val b = dialogBuilder.setView(dialogView).create()

    dialogView.btn_delete_cancel.setOnClickListener { it: View!
        dialogView.btn_delete_accept.isEnabled = false
        b.dismiss()
    }

    dialogView.btn_delete_accept.setOnClickListener { it: View!
        dialogView.btn_delete_cancel.isEnabled = false
        darDeBaja()

        b.dismiss()
    }

    b.setCancelable(false)
    b.show()
}
```

Figura 36: Darse de baja (Ventana flotante)

```
private fun darDeBaja() {
    val userRef = FirebaseDatabase.getInstance().getReference(Constants.USERS)
    mFirebaseAuth.currentUser?.let { fuser ->
        userRef.child(fuser.uid).removeValue().addOnCompleteListener { it1 ->
            if(it1.isSuccessful) {
                mFirebaseAuth.currentUser!!.delete().addOnSuccessListener { it: Void!
                    startActivity(Intent(packageContext: this, LoginActivity::class.java))
                    finish()
                }
            }
        }
    }
}
```

Figura 37: Darse de baja

Se ve en la imagen 36 que aparece una ventana flotante (*dialogView*) en la cual al pulsar la opción *btn delete accept* invoca la función mostrada en la imagen 37, *darDeBaja()*. La cual elimina la información del usuario de la base de datos y al confirmar que lo ha hecho correctamente pide a Firebase que el usuario sea eliminado de la lista de aquellos que pueden iniciar sesión.

M-7: Ver información lugar

En los objeto de tipo Place se almacenará toda la información de un lugar concreto (figura 38), se pueden ver los siguientes argumentos: *ID del lugar, nombre, descripción del lugar, ID del usuario creador, puntuación, ruta de imágenes del lugar en la BBDD, coordenadas, categorías y lista de comentarios.*

```
class Place(  
    var placeId: String = "",  
    var placeName: String = "",  
    var placeDescription: String = "",  
    var placeCreator: String = "",  
    var placeScore: Double = 0.0,  
    var placePictures: String = "",  
    var placeCoordinates: String = "0.0,0.0",  
    var placeTags: String = "",  
    var placeComments: MutableMap<String, Comment> = mutableMapOf()  
): Serializable {  
  
    var arrayTags: ArrayList<Int> = arrayListOf()
```

Figura 38: Estructura clase Place

```
(context as MainActivity).toolbar.title = viewModel.place.placeName  
  
tv_placeName.text = viewModel.place.placeName  
tv_placeDescription.text = viewModel.place.placeDescription  
place_rating_bar.rating = viewModel.place.placeScore.toFloat()  
place_rating_bar.isEnabled = false
```

Figura 39: Asociación de información lugar a elementos gráficos

Se ve que en la figura 39 se asocia la información almacenada en el objeto place de tipo Place en los distintos elementos gráficos de la vista de la aplicación. Y se establece que no se pueda interactuar directamente con la barra de puntuación ya que se hará a través de otra vista.

En la figura 40 se contempla como se descargan las imágenes de la ruta almacenada en el objeto de tipo Place. Contempla si hay varias imágenes y las descarga todas (aunque solo se puede subir un máximo de imágenes).

```

fun getImages(path: String) {
    if (path != "") {

        val storageRef = FirebaseStorage.getInstance().getReference(path)
        val maxDownloadBytes: Long = 1024 * 1024

        storageRef.listAll().addOnSuccessListener { listResult ->

            for ((index, fileRef) in listResult.items.withIndex()) {
                fileRef.getBytes(maxDownloadBytes).addOnSuccessListener { bytes ->
                    val bmp: Bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.size)
                    myBitmapPlaceImg[index] = bmp

                    if(index == listResult.items.lastIndex)
                        imagesDownloaded.value = true
                }
            }
        }
    }
}

```

Figura 40: Descarga imágenes

M-8: Añadir información lugar

En la figura 41 se ve como se se obtiene la información y esta se agrega en un objeto de tipo Place para ser subido a la base de datos. Si campos como el nombre o la descripción están vacíos dará error. En caso contrario mostrará un error en cualquiera de los campos obligatorios que se encuentren vacíos.

Si se detectan imágenes se agregará una ruta en la que serán almacenadas en la base de datos, en caso contrario ese campo estará vacío.

```
private fun darAltaLugar() {
    val tags = getStringTags()
    if(tags == "") {
        Toast.makeText((context as MainActivity), text: "Has de seleccionar al menos una etiqueta", Toast.LENGTH_SHORT).show()
    }
    else {
        val name: String = et_place_name.text.toString()
        val info: String = et_place_info.text.toString()
        val latitude = this.latLng.latitude
        val longitude = this.latLng.longitude

        if(name.isBlank()) {
            et_place_name.error = "Tienes que escribir un nombre"
        }
        if(info.isBlank()) {
            et_place_info.error = "Tienes que dar una descripción"
        }

        if(!name.isBlank() && !info.isBlank()) {
            et_place_name.error = null
            et_place_info.error = null

            if (this.cnt_images > 0)
                this.pathImage = "Lugares/${viewModel.id_lugar}/"

            val myPlace = Place(viewModel.id_lugar, name, info, viewModel.user.userId, placeScore: 5.0, this.viewModel.uploadPlace(myPlace))
        }
    }
}
```

Figura 41: Creación del lugar

Se muestra en la figura 42 como se seleccionan imágenes del propio smartphone, si no se han superado el máximo de imágenes definido en la constante *MAX IMAGES UPLOAD* se permite al usuario acceder a una aplicación para seleccionar la imagen deseada. La aplicación será la que tenga el dispositivo Android establecida por defecto.

```

/**
 * Abrimos el selector de imágenes*/
private fun openFileChooser() {

    if(cnt_images < Constants.MAX_IMAGES_UPLOAD) {
        val intent = Intent()
        intent.type = "image/*"
        intent.action = Intent.ACTION_GET_CONTENT
        startActivityForResult(intent, PICK_IMAGE_REQUEST)
    }
    else
        Toast.makeText((context as MainActivity), text: "El máximo de imágenes es ${Constants.MAX_IMAGES_UPLOAD}")
}

```

Figura 42: Selección de imágenes

Aquí se muestra (imagen 43) como se abre una actividad de google maps para seleccionar las coordenadas del lugar deseado.

```

private fun openMap() {

    val mapIntent = Intent((context as MainActivity), MapsActivity::class.java)

    mapIntent.putExtra(name: "AddNewPlace", value: true)
    mapIntent.putExtra(name: "MyUser", viewModel.user)

    mapIntent.resolveActivity((activity as MainActivity).packageManager)?.let {
        startActivityForResult(mapIntent, Constants.OPEN_MAP_REQUEST)
    }
}

```

Figura 43: Selección coordenadas

Una vez creado el lugar (imagen 41) se da de alta en la base de datos, si los datos de lugar se han subido correctamente y hay una ruta establecida para las imágenes se procederá a subir las imágenes seleccionadas por el usuario a la base de datos en la ruta establecida.

```
fun uploadPlace(place: Place) {  
    val refPlace: DatabaseReference = FirebaseDatabase.getInstance().getReference(Constants.PLACES)  
    refPlace.child(place.placeId).setValue(place).addOnCompleteListener { it: Task<Void!> ->  
        if(it.isSuccessful) {  
            if(myMapPlaceImg.isEmpty()) {  
                //Pongo observer a true  
                placeUploaded.value = true  
            }  
            else {  
                myMapPlaceImg.iterator().forEach { entry ->  
                    uploadImage(entry.value, myMapImgExtension[entry.key], entry.key, place.placeName)  
                }  
            }  
        }  
    }  
}
```

Figura 44: Dada de alta en BBDD del lugar

M-9: Calificar lugar

En la figura 45 se puede ver como se invoca una ventana flotante que muestra un objeto de tipo *ratingBar* donde el usuario puede marcar la puntuación deseada (de cero a cinco estrellas) y al pulsar el botón *btn score accept* se invoca la función de la figura 46 donde la puntuación dada por el usuario se subirá a la base de datos en la ruta específica para el lugar en cuestión.

```
fun ratePlace(place_model: PlaceDataViewModel) {  
  
    val dialogBuilder = AlertDialog.Builder(context: this)  
    val dialogView = layoutInflater.inflate(R.layout.dialog_score_place, root: null)  
    val b = dialogBuilder.setView(dialogView).create()  
  
    dialogView.btn_score_cancel.setOnClickListener { it: View!  
        dialogView.btn_score_accept.isEnabled = false  
        b.dismiss()  
    }  
  
    dialogView.btn_score_accept.setOnClickListener { it: View!  
        dialogView.btn_score_cancel.isEnabled = false  
        dialogView.ratingBar.isEnabled = false  
        val rating = dialogView.ratingBar.rating  
  
        place_model.uploadScoreUser(rating)  
  
        b.dismiss()  
    }  
  
    b.setCancelable(false)  
    b.show()  
}
```

Figura 45: Ventana flotante puntuación

```

fun uploadScoreUser(score: Float) {
    val placeRef = FirebaseDatabase.getInstance().getReference(Constants.PLACES).child(
        Constants.PLACECOMMENTS
    ).child(user.userId)

    val scoreRef = FirebaseDatabase.getInstance().getReference(Constants.PLACESCORES).child(
        Constants.PLACECOMMENTS
    ).child(user.userId)

    placeRef.child(Constants.USERPLACESCORE).setValue(score).addOnCompleteListener { it: Task<Void>?
        if(it.isSuccessful) {
            Log.v( tag: "FIREBASE_BBDD", msg: "SUCCESS_ADD_SCORE")
            scoreRef.setValue(score).addOnSuccessListener { it: Void!
                Log.v( tag: "FIREBASE_BBDD", msg: "SUCCESS_ADD_SCORE")
            }
        }
        else
            Log.v( tag: "FIREBASE_BBDD", msg: "ERROR_ADD_SCORE")
    }
}

```

Figura 46: Subida de puntuación a BBDD

M-10: Crear lista

En la vista de las lista del usuario hay un botón para llamar al fragmento que sirve para crear una nueva lista y darla de alta en la base de datos (figura 47). A este fragmento se le pasa una lista de los códigos de las otras listas (cada lista se encuentra identificado por un id único) para que al generar un nuevo código este no se repita.

Al añadir elementos a una lista será posible filtrarlos tal y como se especificó en 35. Se puede ver que en la figura 48 se pueden filtrar estos lugares por categoría para seleccionar los lugares más interesantes sin perder demasiado tiempo.

```
fab_add_list.setOnClickListener { btnView ->
    view.let { it: View?
        if (it != null) {
            val bundle = Bundle()
            bundle.putBoolean("newList", true)
            bundle.putStringArrayList("listCodes", viewModel.listCodes)
            Navigation.findNavController(it).navigate(R.id.action_userListOf
        }
    }
}
```

Figura 47: Acceso a crear lista

```
private fun configSpinner() {

    arrayAdapter = ArrayAdapter((context as MainActivity), android.R.layout.simple_spinner_item, arrayListTags)
    arrayAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)

    spinner_places_filter.adapter = arrayAdapter
    spinner_places_filter.setOnItemClickListener(object : AdapterView.OnItemClickListener {
        override fun onItemClick(arg0: AdapterView<*>?, arg1: View?,
            pos: Int, arg3: Long) {

            val msupplier: String = spinner_places_filter.selectedItem.toString()
            if (positionSpinner != pos) {
                if (pos >= 0 && pos < arrayListTags.size) {
                    positionSpinner = pos
                    viewModel.filterPlaceList(pos)
                }
            }
        }
    })
}
```

Figura 48: Filtros lugares

En la figura 49 se describe que pasará cuando el usuario entre en contacto con el elemento de la lista de lugares ya sea para añadir o eliminar de la lista, si desea eliminarlo aparecerá un icono que mostrará que puede ser añadido de nuevo si así lo desea y se eliminará de la lista. Si desea añadirlo aparecerá un icono que muestra que puede eliminar ese lugar de la lista y se agregará a la lista en cuestión.

```
override fun onBindViewHolder(holder: RecyclerView.Adapter.ViewHolder, position: Int) {  
    listPlace?.get(position)?.let { it: Place  
        this.parent?.let { parent ->  
            holder.bind(it, parent)  
  
            var imageButton = holder.view.findViewById<ImageButton>(R.id.imb_add_list)  
            mapImageButton[position] = imageButton  
  
            if(arrayListSelected[position]) {  
                mapImageButton[position]?.setImageResource(R.drawable.ic_cancel_icon)  
            }  
            else {  
                mapImageButton[position]?.setImageResource(R.drawable.ic_add_list_icon)  
            }  
            addElement(position, arrayListSelected[position])  
  
            imageButton?.setOnClickListener { it: View!  
                arrayListSelected[position] = !arrayListSelected[position]  
  
                if(arrayListSelected[position]) {  
                    mapImageButton[position]?.setImageResource(R.drawable.ic_cancel_icon)  
                }  
                else {  
                    mapImageButton[position]?.setImageResource(R.drawable.ic_add_list_icon)  
                }  
  
                addElement(position, arrayListSelected[position])  
                notifyDataSetChanged()  
            }  
        }  
    }  
}
```

Figura 49: Interacción con los elementos de la lista de lugares

En 50 se sube a la base de datos la lista con los lugares deseados por el usuario, si es nueva lista (*Booleano newList*) se subirá a la base de datos en una ruta completamente nueva, si es una ya existente que ha sido modificada se subirá a una ruta ya existente sobrescribiendo la anterior.

```
fun uploadListPlace(namePlace: String) {
    val ref = FirebaseDatabase.getInstance().getReference(Constants.USERS).child(paths)
    var id: String

    if(newList) {
        id = ListPlaces().generateId()
        while (listCodes.contains(id) && (id.take(n: 3) == "FAV")) {
            id = ListPlaces().generateId()
        }
    }
    else
        id = listId

    val arrayCodes: ArrayList<String> = arrayListOf()
    if(listPlacesSelected != null) {
        for(elem in listPlacesSelected!!.iterator()) {
            arrayCodes.add(elem.placeId)
        }
    }

    val list = ListPlaces(namePlace, id, arrayCodes)
    ref.child(id).setValue(list.toObject()).addOnCompleteListener { it: Task<Void!>
        if(it.isSuccessful) {
            listUploaded.value = true
        }
    }
}
```

Figura 50: Subida de lista a BBDD

M-11: Editar/Borrar lista

Las listas de los usuarios pueden ser eliminadas o editadas (ya sea editar el nombre de la lista o solo sus elementos). En la lista de las listas personalizadas de cada usuario tal y como se muestra en la figura 51, donde se puede ver que se invoca un menú popup con dos opciones. Eliminar la lista con la opción *opt eliminar lista lugares* y editar el nombre mediante la opción *opt cambiar nom lista lugares*.

En la figura 52 se ver como se elimina la lista en cuestión , hay que concretar que una lista solo puede ser editada o eliminada por el usuario que la creó o uno de los administradores de la aplicación, simplemente encuentra la ruta de la lista y la elimina de la base de datos.

```

popupMenu.setOnMenuItemClickListener { item->
    when(item.itemId) {
        R.id.opt_eliminar_lista_lugares -> {
            popupMenu.dismiss()
            val nombre = arrayNames[position]
            model.deleteList(arrayCodigos[position], position)
            Toast.makeText(parent.context, text: "Lista $nombre eliminada", Toast.LENGTH_SHORT).show()
            true ^setOnMenuItemClickListener
        }
        R.id.opt_cambiar_nom_lista_lugares -> {
            popupMenu.dismiss()
            (parent.context as MainActivity).editListNamePopup(arrayCodigos[position], position, arrayNa
            true ^setOnMenuItemClickListener
        }
        else -> {
            popupMenu.dismiss();
            false ^setOnMenuItemClickListener
        }
    }
}

```

Figura 51: Menú flotante editar/borrar listas

```

fun deleteList(listId: String, position: Int) {
    val ref: DatabaseReference
    if(user.userType == Constants.ADMIN) {
        val uid = listCreators[position]
        ref = FirebaseDatabase.getInstance().getReference(path: "${Constants.USER
    }
    else
        ref = FirebaseDatabase.getInstance().getReference(Constants.USERS).child

    ref.removeValue().addOnCompleteListener { it: Task<Void!>
        if(it.isSuccessful) {
            Log.v(tag: "FIREBASE_BBDD", msg: "SUCCESS_DEL_CUSTOM_USER_LIST")
            setLists()
        }
        else {
            Log.v(tag: "FIREBASE_BBDD", msg: "ERROR_DEL_CUSTOM_USER_LIST")
        }
    }
}

```

Figura 52: Función borrado lista de la BBDD

En la figura 53 se muestra como al editar el nombre de la lista aparecerá una ventana flotante con un campo de texto en el cual se escribiría el nuevo nombre de la lista. Si el campo donde se escribe le nombre está vacío simplemente cierra la ventana sin hacer ningún cambio.

```

fun editListNamePopup(listId: String, position: Int, oldName: String, model:
    val dialogBuilder = AlertDialog.Builder(context: this)
    val dialogView = layoutInflater.inflate(R.layout.dialog_edit_list_name,
    val b = dialogBuilder.setView(dialogView).create()

    dialogView.et_edit_list_name.setText(oldName)

    dialogView.btn_edit_cancel.setOnClickListener { it: View!
        dialogView.btn_edit_accept.isEnabled = false
        b.dismiss()
    }

    dialogView.btn_edit_accept.setOnClickListener { it: View!
        dialogView.btn_edit_cancel.isEnabled = false
        val textName = dialogView.et_edit_list_name.text.toString()

        if(!textName.isBlank())
            model.changeListName(listId, textName, position)

        b.dismiss()
    }

    b.setCancelable(false)
    b.show()

```

Figura 53: Ventana flotante de cambio nombre lista

El código mostrado en la figura 54 muestra que la pantalla de añadir nuevos elementos a la lista es la misma que la de editar lista. Es el mismo fragmento que el mostrado en la especificación anterior **M-10**.

La diferencia es que el booleano *newList* sería FALSO ya que no se está creando una nueva lista sino que se está editando, tal y como se muestra en la figura 55. Se puede ver en esa imagen como se proporciona una lista de los elemento previamente seleccionados para que se marquen de forma automática.

```
fab_add_place.setOnClickListener { it: View!
    view.let { it: View?
        if (it != null) {
            viewModel.initialDownload = true
            val bundle = Bundle()
            bundle.putBoolean("newList", false)
            bundle.putString("listId", viewModel.listCode)
            bundle.putString("listName", nameBar)
            bundle.putStringArrayList("listSelected", viewModel.keysPlaces)
            bundle.putStringArrayList("listCodes", arrayListOf())
            Navigation.findNavController(it).navigate(R.id.action_customPlaceList
        }
    }
}
```

Figura 54: Botón agregar nuevos elementos / edición lista

```
if(!viewModel.newList) {
    viewModel.listSelected = arguments?.get("listSelected") as ArrayList<String>
    viewModel.listName = arguments?.get("listName") as String
    viewModel.listId = arguments?.get("listId") as String
}
```

Figura 55: Opción de lista a editar

M-12: Modificar/Eliminar usuarios registrados

```
R.id.nav_userlist -> {  
    nav_host_fragment.view?.let { view ->  
        Navigation.findNavController(view).navigate(R.id.action_userDataFragment_to_listAuxFragment)  
    }  
}
```

Figura 56: Opción lista usuarios menú lateral

```
if((user.userType == Constants.ADMIN) && (it.userId != user.userId)) {  
    btn.setOnClickListener { view ->  
        //creating a popup menu  
        val popup = PopupMenu(view.context, btn)  
        popup.inflate(R.menu.user_list_menu)  
        //adding click listener  
        popup.setOnMenuItemClickListener { item ->  
            when (item.itemId) {  
                R.id.opt_edituser -> callActivity(parent.context, item, listUser!![position])  
                R.id.opt_deleteuser -> callActivity(parent.context, item, listUser!![position])  
                else -> false ^setOnMenuItemClickListener  
            }  
        }  
        //displaying the popup  
        popup.show()  
    }  
}  
else {  
    btn.isEnabled = false  
    btn.isVisible = false  
}
```

Figura 57: Menú flotante para elementos de la lista usuarios

```
R.id.opt_edituser -> {  
    nav_host_fragment.view?.let { view ->  
        Navigation.findNavController(view).navigate(R.id.action_listUser_to_editUser)  
    }  
}  
R.id.opt_deleteuser -> {  
    model.deleteUserData(useredit, useredit.userPassword, user)  
}
```

Figura 58: Opciones de modificación de usuarios

```

mFirebaseAuth.signOut()
FirebaseAuth.getInstance().signInWithCredential(credential).addOnCompleteListener { it: Task<AuthResult!>
    if(it.isSuccessful) {
        val userRef = FirebaseDatabase.getInstance().getReference( path: "USUARIOS/${userDel.userId}")
        userRef.removeValue().addOnCompleteListener { it2 ->
            if(it2.isSuccessful) {
                FirebaseAuth.getInstance().currentUser?.delete()?.addOnCompleteListener { task ->
                    if(task.isSuccessful) {
                        FirebaseAuth.getInstance().signInWithEmailAndPassword(currentuser.userMail, psswd)
                        userEdited.value = true
                    }
                }
            }
        }
    }
}

```

Figura 59: Borrado de datos del usuario

En la figura 56 se muestra la opción del menú lateral para acceder a la lista de todos los usuario, la cual solo es accesible si el usuario es de tipo Administrador.

Se contempla en la figura 57 que aparecerá un menú flotante para cada elemento de la lista de usuarios en el que se dará la opción de editar los datos del usuario (*opt edituser*) o eliminar al usuario (*opt deleteuser*). Se puede ver que solo puede aparecer este menú flotante si el usuario es de tipo Administrador o si el usuario que esta utilizando la aplicación en el momento desea modificar sus propios datos.

En la figura 58 se ve como se invocan en el caso de editar los datos del usuario (*opt edituser*) se invoca el fragmento *EditUserFragment* en el cual se editan los datos del usuario (se habla de la implementación de este fragmento en el caso de uso **M-14: Modificar datos propios**). La segunda opción (*opt deleteuser*) invoca la función *deleteUserData*.

La función *deleteUserData* se muestra en la figura 59 donde se muestra como el administrador temporalmente se sale de los servicios de la cuenta su usuario para conectarse como el usuario a eliminar. Esto se debe a que la plataforma Firebase solo permite dar de baja a un usuario si primero este está conectado a los servicios de Firebase con sus credenciales. Una vez conectado como el usuario a eliminar elimina su información de la base de datos y de la lista de cuentas que pueden acceder a los servicios de la aplicación (*FirebaseAuth*). TRas realizar esos pasos se vuelve a conectar automáticamente con su cuenta de administrador.

M-13: Modificar información lugar

```
private fun initSetup() {

    btn_tags.setOnClickListener { it: View!
        showDialogTags()
    }

    et_name_watcher = object : TextWatcher {
        override fun onTextChanged(s: CharSequence, start: Int, before: I
        override fun beforeTextChanged(s: CharSequence, start: Int, count
        override fun afterTextChanged(s: Editable) {
            if (et_place_name.text.isBlank()) {
                et_place_name.error = "Tienes que escribir un nombre"
            } else {
                et_place_name.error = null
            }
        }
    }
    et_place_name.addTextChangedListener(et_name_watcher)

    et_info_watcher = object : TextWatcher {
        override fun onTextChanged(s: CharSequence, start: Int, before: I
        override fun beforeTextChanged(s: CharSequence, start: Int, count
        override fun afterTextChanged(s: Editable) {
            if (et_place_info.text.isBlank()) {
                et_place_info.error = "Tienes que dar una descripción"
            } else {
                et_place_info.error = null
            }
        }
    }
    et_place_info.addTextChangedListener(et_info_watcher)

    btn_images.setOnClickListener { it: View!
        if (!Utils.checkPermission(activity as MainActivity))
            view?.let { it1 -> Utils.askforPermission(it1) }
        else
            openFileChooser()
    }
}
```

Figura 60: Configuración elementos gráficos 1

```

btn_images.setOnClickListener { it: View!
    if (!Utils.checkPermission(activity as MainActivity))
        view?.let { it1 -> Utils.askforPermission(it1) }
    else
        openFileChooser()
}

imgb_upload.setOnClickListener { it: View!
    darAltaLugar()
}

```

Figura 61: Configuración elementos gráficos 2

Esta implementación es muy similar a la del caso de uso **M-8: Añadir información de lugar**.

En las figuras 60 y 61 se puede ver la configuración de la vista con la que interactúa el usuario. Para los campos de texto (*et place name* y *et place info*) se crean unos observadores de texto que si alguno de los dos campos o ambos se encuentran vacíos mostrarán un mensaje de error y no será posible subir la imagen. El botón *btn images* permitirá al usuario abrir el selector de ficheros y seleccionar la imagen o imágenes deseadas.

```

private fun darAltaLugar() {
    val tags = getStringTags()
    if(tags == "") {
        Toast.makeText((context as MainActivity), text: "Has de seleccionar tags")
    }
    else {
        val name: String = et_place_name.text.toString()
        val info: String = et_place_info.text.toString()
        val latitude = this.latLng.latitude
        val longitude = this.latLng.longitude

        if(name.isBlank()) {
            et_place_name.error = "Tienes que escribir un nombre"
        }
        if(info.isBlank()) {
            et_place_info.error = "Tienes que dar una descripción"
        }

        if(!name.isBlank() && !info.isBlank()) {
            et_place_name.error = null
            et_place_info.error = null

            if (this.cnt_images > 0)
                this.pathImage = "Lugares/${viewModel.id_lugar}/"

            place.placeName = name
            place.placeDescription = info
            place.placeCoordinates = "${latitude},${longitude}"
            place.placeTags = tags
            //val myPlace = Place(viewModel.id_lugar, name, info, view
            viewModel.modifyPlace(place)
        }
    }
}
}

```

Figura 62: Recopilación datos usuario

Las figuras 62 y 63 muestran respectivamente la recopilación de datos y comprobaciones

```

//Eliminamos los comentarios para que no se suban con el resto del objeto place de forma erronea
place.placeComments.clear()

refPlace.setValue(place).addOnCompleteListener { it: Task<Void!>
    if(it.isSuccessful) {

        refPlace.child(Constants.PLACECOMMENTS).child(user.userId).setValue(comments)
        if(myMapPlaceImg.isEmpty()) {
            //Pongo observer a true
            placeUploaded.value = true
        }
        else {
            myMapPlaceImg.iterator().forEach { entry ->
                uploadImage(entry.value, myMapImgExtension[entry.key], entry.key, place.placeName)
            }
        }
    }
}

```

Figura 63: Subida información a BBDD

para que no se suba información incompleta (por ejemplo los campos *name* e *info* si se encuentran vacíos alguno de los dos no será posible dar de alta la información), y la subida de esa información a la base de datos.

M-14: Modificar datos propios

Se observa en esta pantalla de opciones de usuario (64) como hay tres opciones distintas en función del botón que se llegue a pulsar:

- *Btn edit user data*: Se pulsa editar datos del usuario.
- *Btn edit prefs*: Se pulsa modificar las preferencias del usuario (establecidas al crear la cuenta por primera vez).
- *Btn delete account*: Se pulsa para dar de baja al usuario de los servicios de la aplicación.

```
btn_edit_user_data.setOnClickListener { it: View!
    nav_host_fragment.view?.let { view ->
        (activity as MainActivity).useredit = (activity as MainActivity).user
        Navigation.findNavController(view).navigate(R.id.action_userOptionsFrag
    }
}

btn_edit_prefs.setOnClickListener { it: View!
    nav_host_fragment.view?.let { view ->
        Navigation.findNavController(view).navigate(R.id.action_userOptionsFrag
    }
}

btn_delete_account.setOnClickListener { it: View!
    (activity as MainActivity).deleteAccountPopup()
}
```

Figura 64: Opciones de usuario

En la figura 65 se ve como se comprueba que la contraseña al ser introducida dos veces (la segunda vez es por seguridad, para garantizar que el usuario recuerda la misma), sean exactamente iguales. En caso contrario devolverá un error mostrando un texto en el campo de la segunda contraseña indicando que ambos campos deben coincidir.

Si ocurre que no coinciden no será posible subir los cambios a la base de datos.

```

/**
 * Funcione que valida si dos campos contienen la misma cadena
 */
private fun setShameStringError(et0: EditText, et1: EditText){
    when {
        Validation.checkShameString(et0.text.toString(),et1.text.toString()) -> {
            tiPasswordSecure.error = null
            et1.tag = true
        }
        else -> {
            tiPasswordSecure.error = "Las contraseñas deben coincidir"
            et1.tag = false
        }
    }
}
}

```

Figura 65: Comprobación de contraseña valida

El código de la figura 66 sube a la base de datos las modificaciones a la información del usuario.

```

private fun onUploadData() {
    val userAux = User(input_name.text.toString(),input_password.text.to
    val currentuser = (activity as MainActivity).user
    enableElements( value: false)
    viewModel.uploadUserData(userAux, user.userPassword, currentuser)
}

```

Figura 66: Dada de alta de información en BBDD

Se ve en esta imagen (figura 67) como se muestra una ventana flotante que pregunta al usuario si desea darse de baja de los servicios de la aplicación, al seleccionar la opción de aceptar (*btn delete accept*) se invoca la función de la figura 68 donde se puede ver como se eliminan los datos de la base de datos y de la lista de usuario que pueden acceder a la aplicación y se saca al usuario de la misma de vuelta a la pantalla de login solo que esta vez necesitaría dar de alta una nueva función para poder acceder.

```

fun deleteAccountPopup() {
    val dialogBuilder = AlertDialog.Builder(context: this)
    val dialogView = inflater.inflate(R.layout.dialog_delete_ac
    val b = dialogBuilder.setView(dialogView).create()

    dialogView.btn_delete_cancel.setOnClickListener { it: View!
        dialogView.btn_delete_accept.isEnabled = false
        b.dismiss()
    }

    dialogView.btn_delete_accept.setOnClickListener { it: View!
        dialogView.btn_delete_cancel.isEnabled = false
        darDeBaja()

        b.dismiss()
    }

    b.setCancelable(false)
    b.show()
}

```

Figura 67: Ventana flotante - dar de baja

```

private fun darDeBaja() {
    val userRef = FirebaseDatabase.getInstance().getReference(Constants.USERS)
    FirebaseAuth.currentUser?.let { fuser ->
        userRef.child(fuser.uid).removeValue().addOnCompleteListener { it1->
            if(it1.isSuccessful) {
                FirebaseAuth.currentUser!!.delete().addOnSuccessListener { it: Void!
                    startActivity(Intent(packageContext: this, LoginActivity::class.java))
                    finish()
                }
            }
        }
    }
}
}

```

Figura 68: Dada de baja del usuario de la BBDD

Se contempla en 69 como el usuario tras seleccionar las etiquetas/preferencias deseadas (muy similar a lo que ocurre en M-1 al darse de alta) invoca a la función de la figura 70 donde se actualizan en la base de datos las preferencias del usuario.

```
private fun getSelectedTags() {  
    val array = viewModel.user.arrayPrefs  
    for(num in array.iterator()) {  
        viewModel.myAdapter.listTagsSelected?.set(num, true)  
    }  
    viewModel.myAdapter.listTagsSelected?.let { viewModel.myAdapter.uploadSetListSelected(it) }  
    viewModel.myAdapter.notifyDataSetChanged()  
}
```

Figura 69: Obtención de preferencias seleccionadas por el usuario

```
fun uploadUserPrefs() {  
    val prefRef = FirebaseDatabase.getInstance().getReference(Constants.USERPREFS)  
  
    val msg = getMsg()  
    user.userPrefs = msg  
    user.arrayPrefs = getPrefs(msg)  
  
    prefRef.setValue(msg).addOnCompleteListener { it: Task<Void!>  
        if(it.isSuccessful) {  
            Log.v( tag: "FIREBASE_BBDD", msg: "SUCCESS_TAGS_UPLOAD")  
        }  
    }  
}
```

Figura 70: Actualización en BBDD de preferencias usuario

8.2. Función Javascript

Se mostrará cual es la función de Javascript de la cual se habló en el apartado de *Firebase Cloud Functions* (subsección 3.4.3). Para detectar los eventos dados en la base de datos en tiempo real existen cuatro funciones [1].

- **onWrite**: Se activa cuando se crean, actualizan o borran datos en Realtime Database.
- **onCreate**: Se activa cuando se crean nuevos datos en Realtime Database.
- **onUpdate**: Se activa cuando se actualizan los datos ya existentes en Realtime Database.
- **onDelete**: Se activa cuando se eliminan datos en Realtime Database.

Listing 1: Ejemplo de declaración evento *onWrite*

```
1 exports.calcMediaPuntuacion = functions
2   .database.ref("/LUGARESPUNTUACION/{placeId}")
3   .onWrite((change, context) => {
4     let punct = 0;
5     let punctTotal = 0;
6     const numChild = change.after.numChildren();
7
8     // OBTENEMOS TODAS LAS PUNTUACIONES DEL LUGAR
9     change.after.forEach((child) => {
10      punct = child.val();
11      punctTotal = punctTotal + punct;
12    });
13    const num = Math.ceil((punctTotal / numChild));
14    const obj = {"placeScore": num};
15    const snapAux = change.after.ref.parent.parent;
16    return snapAux.child("/LUGARES/"+change.after.key)
17      .update(obj);
18  });
```

Hay dos tipos de detección de eventos, uno más general para cualquier tipo de cambio (*onWrite*), o para cada evento de forma específica (*onCreate*, *onUpdate*, *onDelete*), en el caso de este proyecto utilizamos el evento más general.

Indicamos que cuando se cree algo en el nodo `‘/LUGARESPUNTUACION/{placeId}’` se ejecute esta función, `‘{placeId}’` indica que la función se aplica a cualquier nodo dentro de `‘/LUGARESPUNTUACION/’`. `{placeId}` indica el nodo particular en el que se ha dado el cambio. Ese nodo almacena el id de un lugar, a continuación se calcula la media de todas las puntuaciones de ese lugar y se actualiza el valor de la puntuación general.

“change” contiene los datos del nodo que ha sido modificado, a partir del mismo obtenemos la información deseada, en este caso serían las puntuaciones dadas por los usuarios al lugar indicado por `‘{placeId}’`. “change.before” contiene la información del nodo antes del cambio y “change.after” contiene la información del nodo después del cambio.

8.3. Navegación de vistas y patrón de la aplicación

8.3.1. Navegación de vistas

Android Jetpack es un conjunto de bibliotecas, librerías, guías cuyo objetivo es facilitar el desarrollo de aplicaciones a los programadores. En Android Jetpack hay un componente llamado *navegador*, este componente como dice su nombre navegar entre las distintas partes del contenido de tu aplicación. Este componente se utiliza para navegar entre las distintas vistas de la aplicación de forma ágil y sencilla.

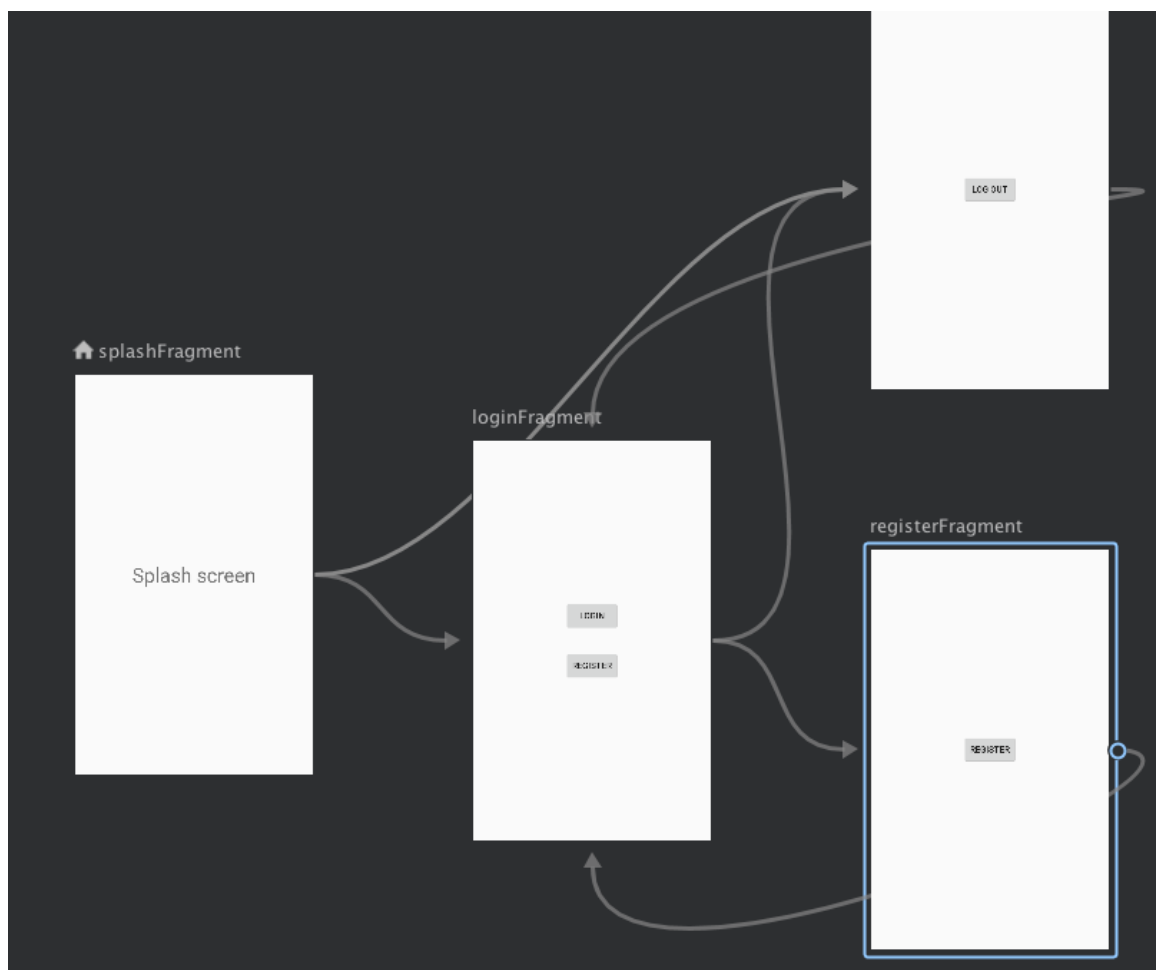


Figura 71: Ejemplo gráfico de navegación

En la aplicación hay un componente llamado *NavHost* el cual es un contenedor donde se insertarán las distintas vistas, nuestro *NavHost* es nuestra actividad principal llamada *MainActivity*, se insertarán en ella distintos fragmentos intercambiándose entre sí a medida que el usuario se desplace por la aplicación.

Se define como se intercambiarán estos fragmentos en el gráfico de navegación, este es un documento en formato XML donde se definen las acciones posibles dentro de cada fragmento.

En esta imagen se ve como no es posible ir de un fragmento a todos, sino que hay unas rutas determinadas que decide el desarrollador, no es posible ir de la pantalla splash a la pantalla de registros, hay que pasar primero por la pantalla del login.

8.3.2. MVVM

Para esta aplicación se ha seguido un patrón de diseño llamado Modelo – Vista – Modelo de vista (MVVM), este patrón permite separar la parte de la lógica del proyecto y la parte visual del mismo, que es con la que interactúa el usuario. Ahora explicaré los tres componentes que forman este patrón:

- **Modelo:** Es la capa de datos, contiene la información necesaria pero no provee directamente la información ni la manipula.
- **Vista:** : Es la parte de la aplicación que interactúa con el usuario y muestra la información deseada, pero la vista no posee ninguna lógica.
- **Modelo de vista:** Es un intermediario entre la vista y el modelo, contiene la lógica de la vista y solicita información al modelo comunicando estos dos de forma indirecta.

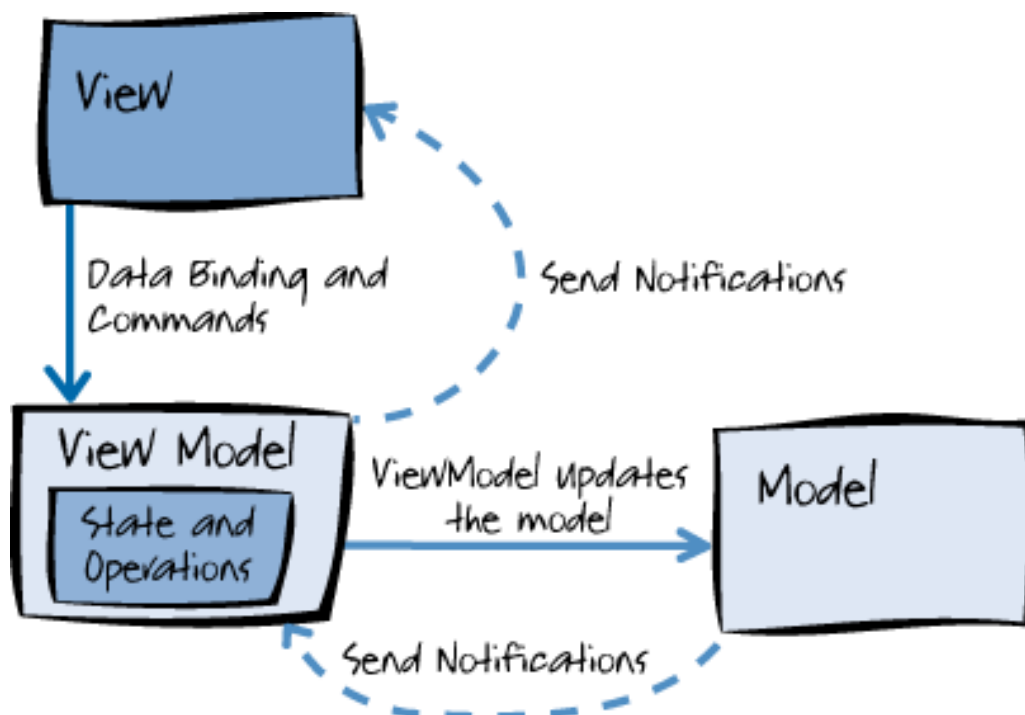


Figura 72: Diagrama del patrón MVVM

Algo que se muestra en la imagen es que la comunicación no tiene porque ser directa en el patrón MVVM. Vista se puede comunicar directamente con Modelo de vista, y este a su vez con Modelo, pero la comunicación no es directa en el sentido contrario. Modelo no puede comunicarse con Modelo de vista y este no puede comunicarse con Vista, de hecho no sabe ni que existe una vista en primer lugar. Así se hace posible la separación de la lógica y la vista.

Para que Vista pueda obtener los resultados y datos proporcionados por Modelo de vista existe un tipo de objeto llamado `MutableLiveData`, este es un objeto que puede ser observado y cada vez que hay un cambio en su valor el observador del mismo es notificado automáticamente. Este es el patrón utilizado en la aplicación, de esta forma al estar muy modularizado es más sencillo realizar modificaciones sin que afecte al resto del código.

9. Evaluación

Se dio a probar la aplicación a cinco usuarios distintos para que evaluaran la experiencia con la misma, tras realizar una serie de tareas se les hizo una serie de preguntas respecto a la aplicación. Esto se hizo con la intención de encontrar errores de diseño o cosas a mejorar.

Usuario 1

- **¿Como de intuitiva te pareció la aplicación? (del 1 al 5):** 4.
- **¿Que piensas del diseño de la aplicación? (del 1 al 5):** 3.5.
- **Aspectos positivos de la aplicación:** Por lo general es simple de utilizar, te permite subir información de una manera sencilla y consultar información nueva.
- **Aspectos negativos:** A la hora de crear una cuenta no sabía como acceder desde la pantalla de login a la pantalla de crear cuenta. En general proceso “dar de alta” un poco confuso.
- **Posibles sugerencias:** Mejorar diseño de algunas partes de la app como la opción de “añadir lugar sea más clara” y mejorar pantalla “dar de alta”.

Usuario 2

- **¿Como de intuitiva te pareció la aplicación? (del 1 al 5):** 4.5.
- **¿Que piensas del diseño de la aplicación? (del 1 al 5):** 4.
- **Aspectos positivos de la aplicación:** Fácil de usar y útil.
- **Aspectos negativos:** Algunos iconos son un poco confusos, imágenes tardaban en cargar algunas veces.
- **Posibles sugerencias:** El diseño puede ser mejor estéticamente, se puede dejar más claro donde se filtran algunas cosas.

Usuario 3

- **¿Como de intuitiva te pareció la aplicación? (del 1 al 5):** 4.
- **¿Que piensas del diseño de la aplicación? (del 1 al 5):** 4.
- **Aspectos positivos de la aplicación:** Útil para gente que desee viajar y no ir a los lugares más “turísticos”.
- **Aspectos negativos:** Algunos elementos pueden ser confusos, el botón de editar lista de usuarios tiene un icono un poco confuso ya que parece que solo sirve para añadir elementos.

Para poder ver la ruta en google maps he de tener el dedo pulsado sobre el marcador (y eso lo desconocía).

- **Posibles sugerencias:** Modificar iconos de botones y explicar o especificar como interactuar con algunos elementos gráficos.

Usuario 4

- **¿Como de intuitiva te pareció la aplicación? (del 1 al 5):** 4
- **¿Que piensas del diseño de la aplicación? (del 1 al 5):** 4
- **Aspectos positivos de la aplicación:** Puedes ver información interesante de lugares que de otra forma no podrías haber conocido.
- **Aspectos negativos:** A veces la información tarda un poco en descargarse como las imágenes, al crear lugares a primera vista no sabía que campos eran obligatorios y cuales no.
- **Posibles sugerencias:** Especificar que campos son obligatorios y cuales no lo son (para añadir lugar).

Usuario 5

- **¿Como de intuitiva te pareció la aplicación? (del 1 al 5):** 4.5.
- **¿Que piensas del diseño de la aplicación? (del 1 al 5):** 4.
- **Aspectos positivos de la aplicación:** Permite guardar lugares interesantes y permite filtrar lugares según tus intereses.
- **Aspectos negativos:** No permite compartir por otras redes sociales u aplicaciones la información del lugar, lo cual sería muy interesante (sobre todo para atraer a la gente que no tiene esta aplicación).
- **Posibles sugerencias:** Agregar en un futuro esa funcionalidad. Poder desde la vista del lugar añadir el lugar a una de las listas ya existentes.

Tras ver esto a los distintos usuarios se les pidió que realizasen una serie de tareas con la aplicación, estas eran:

1. Crear una cuenta de la aplicación.
2. Seleccionar preferencias.
3. Acceder a la lista de lugares.
4. Filtrar lugares por “Museos”.
5. Quitar filtro Museos.
6. Filtrar puntuación alta.
7. Seleccionar un lugar.
8. Puntuar lugar.
9. Añadir a favoritos.
10. Escribir comentario.
11. Agregar lugar nuevo.
12. Volver a pantalla principal.
13. Ir a lista personalizados.
14. Crear nueva lista personalizada.
15. Cambiar nombre lista.
16. Cerrar sesión.

Una vez realizadas las tareas se pidió a los usuario que completasen un cuestionario anónimo y online (utilizando la plataforma de *Google Forms* [16]) donde puntuarían del 1 al 5 distintos aspectos de la aplicación.

¿Cuál es tu opinión del diseño de la aplicación?

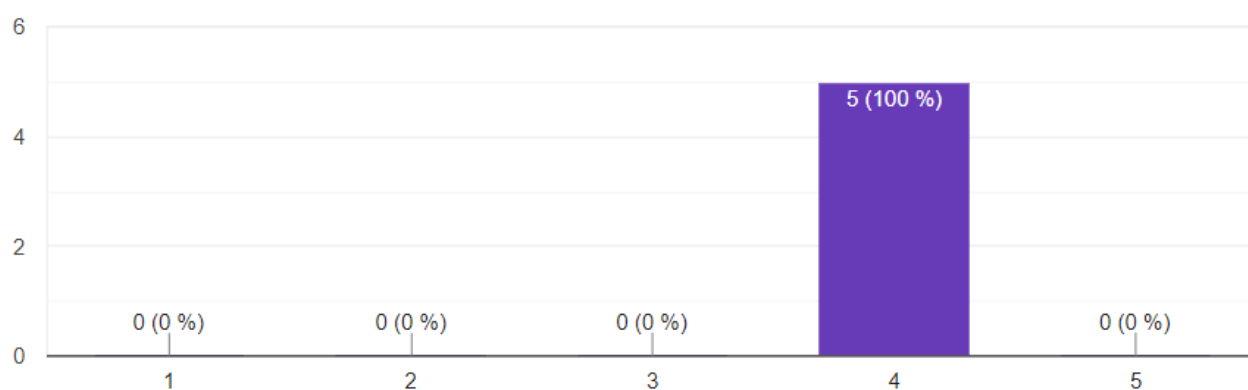


Figura 73: Pregunta 1.

¿Cuál es tu nivel de satisfacción con la aplicación?

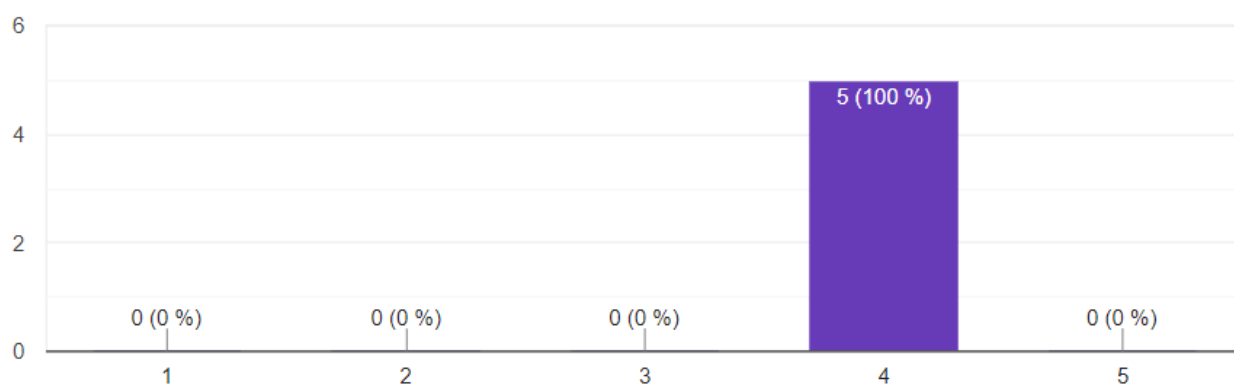


Figura 74: Pregunta 2.

¿Cómo de sencillo consideras que es añadir nuevos lugares a las ya existentes?

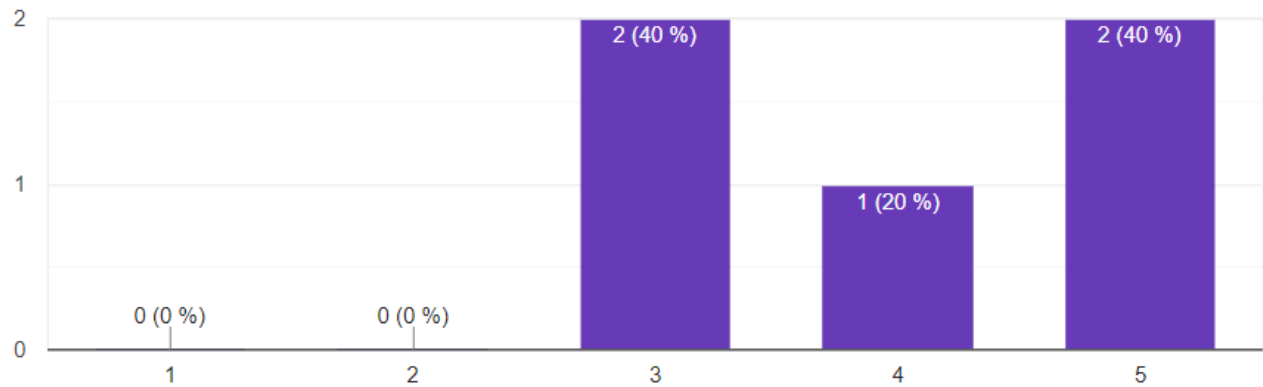


Figura 75: Pregunta 3.

Con esta pregunta se puede ver que algunos de los usuario no tenían muy claros que campos eran obligatorios de completar y cuales no lo eran.

¿Cómo de fácil te ha resultado buscar un lugar específico?

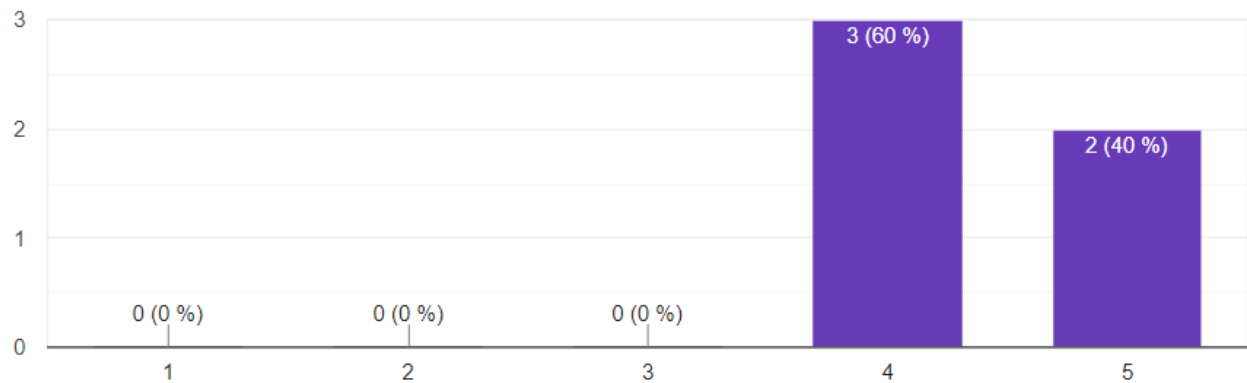


Figura 76: Pregunta 4.

¿Estas satisfecho con la forma en que se muestra la información?

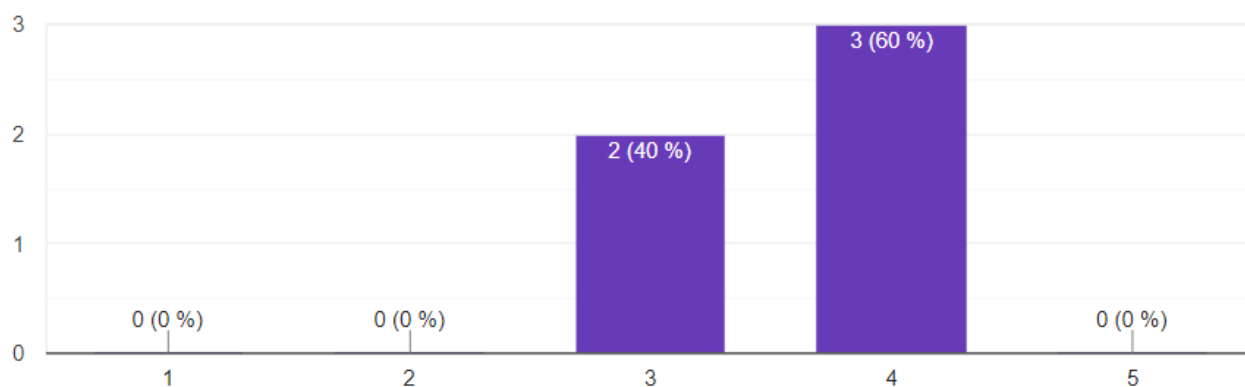


Figura 77: Pregunta 5.

Se puede ver que hay algunos usuarios que consideran que podría tener un diseño distinto, estéticamente más profesional.

¿Cómo de sencillo ha sido filtrar lugares según tus deseos?

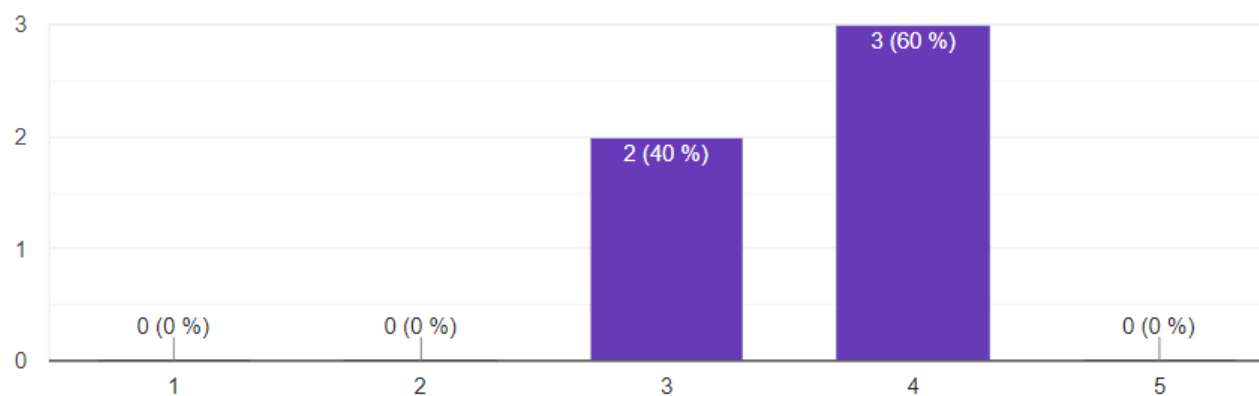


Figura 78: Pregunta 6.

Se contempla que en general los usuarios que han probado la aplicación están satisfechos con como se filtran los resultados de la aplicación.

¿Cómo de sencillo ha sido encontrar la ruta al lugar deseado?

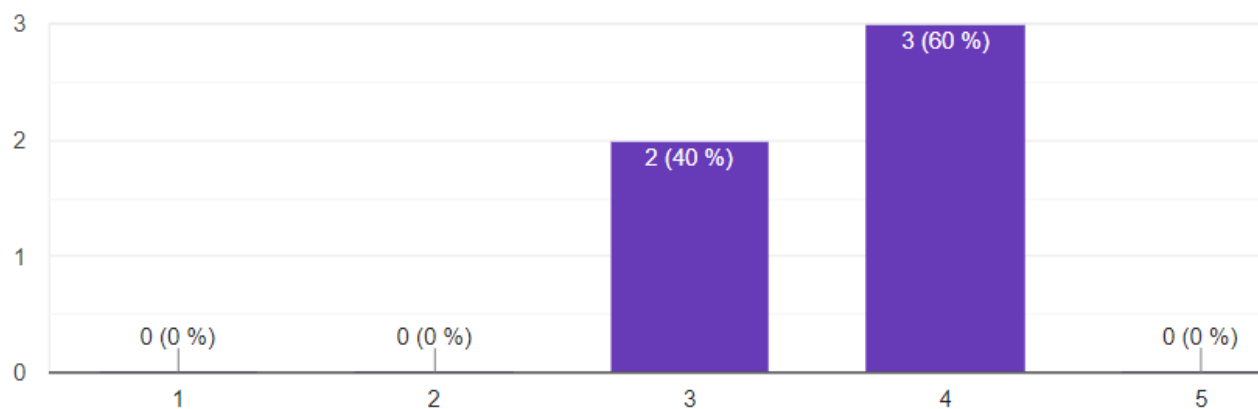


Figura 79: Pregunta 7.

¿Has encontrado facilidad al crear y almacenar lugares en las listas personalizadas?

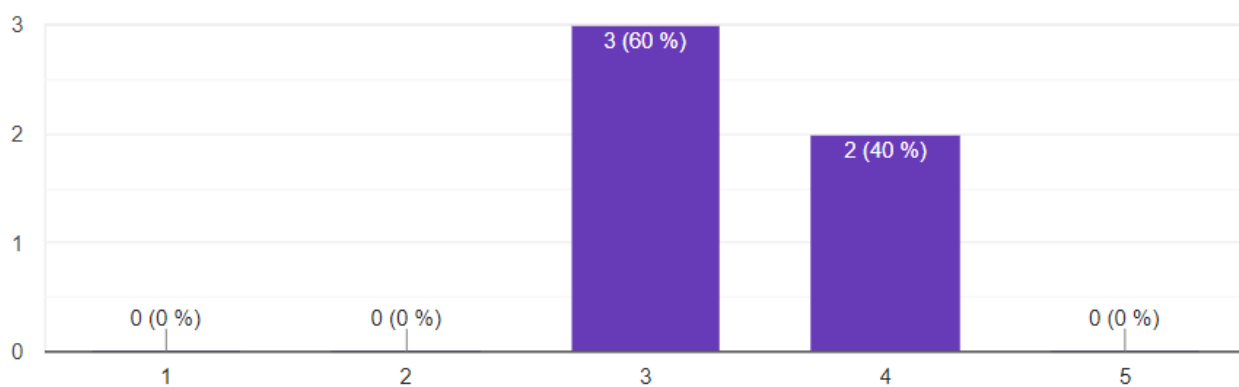


Figura 80: Pregunta 8.

¿Has podido editar estas listas sin problemas?

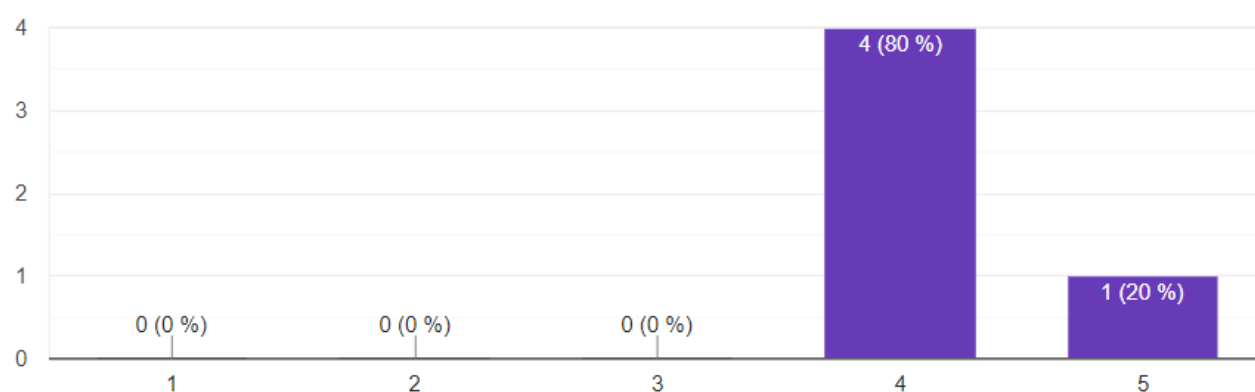


Figura 81: Pregunta 9.

A la hora de editar la información del usuario ¿ha sido fácil?

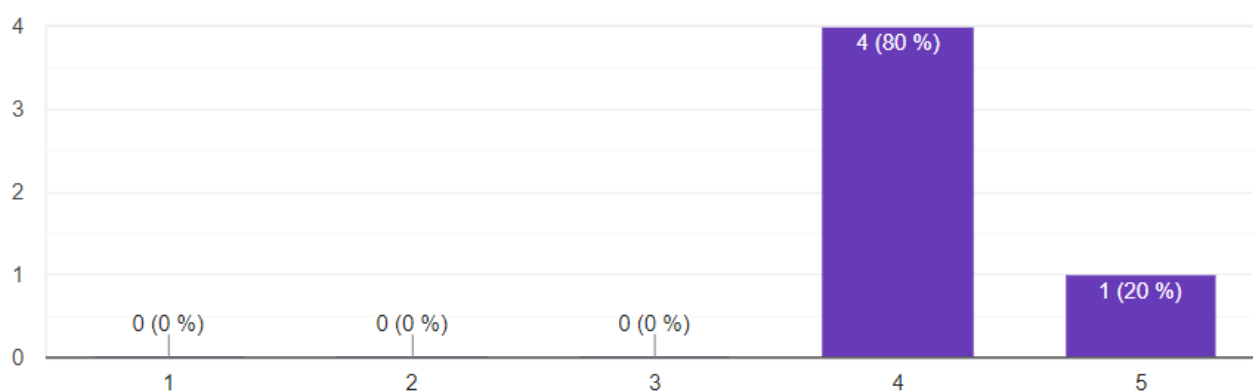


Figura 82: Pregunta 10.

¿Es sencillo editar los lugares y puntuarlos?

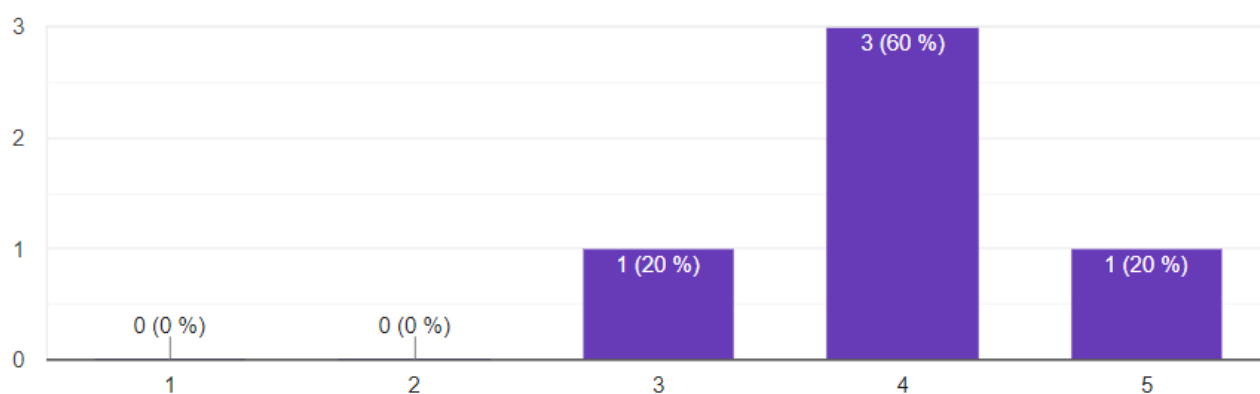


Figura 83: Pregunta 11.

Por lo general los usuarios están satisfechos con como se pueden editar lugares, pero en un caso uno de los usuarios tuvo cierta confusión al cambiar un elemento de los campos de información.

Tras contemplar los resultados de las encuestas se ha decidido que se realizarán una serie de modificaciones en la interfaz de la aplicación (iconos, posición de botones, mensajes de texto para guiar al usuario, etc).

10. Conclusiones y trabajo futuro

10.1. Conclusiones

Al finalizar el proyecto al fin fui consciente del esfuerzo que conlleva realizar una aplicación móvil. Si es verdad que había complicaciones y carecía de la experiencia y habilidades necesarias para algunos de las distintas partes del proyecto, gracias al ritmo del mismo, una apropiada división de tareas y al esfuerzo empleado pude poco a poco superar estas dificultades.

El motivo por el cual deseaba desarrollar esta aplicación es que consideraba que había muchos lugares en diferentes sitios que debido a que no tenían cierta importancia histórica (museos, castillos, iglesias, ...) se pasaban siempre por alto por no mencionar que lo que cada uno puede considerar atractivo o digno de ser visto puede variar mucho de persona en persona. Precisamente por esto deseaba que cada uno pudiera ver o dar de alta localizaciones que considerase interesantes ya que para todo siempre hay un público por muy pequeño que sea. Cada usuario en función a sus gustos particulares podría ver un sitio que se adapte a sus intereses.

Uno de los problemas de aprender nuevas disciplinas es que a medida que avanzas uno se va dando cuenta de errores o distintos campos que podrían mejorarse. Pero aún así teniendo en cuenta esto estoy satisfecho con los resultados obtenidos.

10.1. Conclusions

At the end of the project I was finally aware of the effort that goes into making a mobile application. While it is true that there were complications and I lacked the necessary experience and skills for some of the different parts of the project, thanks to the pace of the project, a proper division of the tasks and the effort put in, I was able to gradually overcome these difficulties.

The reason why I wanted to develop this application is that I considered that there were many places in different places that because they did not have a certain historical importance (museums, castles, churches, ...) were always overlooked, not to mention that what each person can consider attractive or worth seeing can vary greatly from person to person. Precisely for this reason I wanted everyone to be able to see or register locations that they considered interesting, as there is always an audience for everything, no matter how small it may be. Each user, depending on his or her particular tastes, could see a site that suits his or her own interests.

One of the problems of learning new disciplines is that as you go along, you become aware of mistakes or different areas that could be improved. But even so, taking this into account, I am satisfied with the overall results obtained.

10.2. Trabajo futuro

A continuación se verá algunas posibles mejoras o nuevas funcionalidades en las que se podría trabajar de cara al futuro:

- **Velocidad descarga imágenes:** Si es verdad que las imágenes se descargan a una velocidad decente, sería preferible que se descargasen más rápido si es posible. Esto podría conseguirse cambiando de servicio de almacenamiento de imágenes por ejemplo.
- **Modo anónimo:** Permitir que el usuario pueda acceder a ciertos servicios de la aplicación (aunque estos estén limitados) para poder consultar cierta información y probar la aplicación si así lo desea.
- **Compatibilidad redes sociales:** Que sea posible compartir información sobre los lugares a través de distintas redes sociales (*Facebook, Twitter, WhatsApp, etc*) así se daría visibilidad al lugar en cuestión y a la aplicación.
- **Vídeos cortos:** En la pantalla de información del lugar agregar la opción de subir un vídeo corto de unos segundos o un enlace a un vídeo en otra plataforma. De esta forma sería posible añadir información más completa.
- **Seguir a usuarios:** En caso de que un usuario vea que otro tiene gustos similares a los suyos, que uno pueda seguir al otro para que pueda ver los sitios que a este le gustan o que sube a la aplicación.

Índice de figuras

1.	Diagrama casos de uso módulo servicios	10
2.	Diagrama casos de uso módulo información	15
3.	Backend as a Service	22
4.	Esquema arquitectura	23
5.	Campo usuario en la base de datos	24
6.	Campo etiquetas en la base de datos	26
7.	Campo lugares en la base de datos	27
8.	Campo lugares en la base de datos	29
9.	Campo puntulación lugares en la base de datos	30
10.	Carpetas de imágenes en los lugares	31
11.	Botón menú lateral	32
12.	Muestra menú flotante	33
13.	Muestra lista dinámica	34
14.	Ejemplo ventana flotante	35
15.	Ejemplo barra de puntuación	35
16.	Muestra tipografía	36
17.	Pantalla información lugar	37
18.	Pantalla datos usuario	38
19.	Pantalla de lista de lugares	39
20.	Filtros de listas de lugares	40
21.	Iconos de muestra	41
22.	Diagrama del ciclo de vida de la actividad.	43
23.	Diagrama del ciclo de vida del fragmento.	45
24.	Obtención de datos	47
25.	Alta del usuario	47
26.	Lectura de datos del usuario	48
27.	Accediendo a los servicios de Firebase	48
28.	Fin de login	49
29.	Opción menú <i>Cerrar sesión</i>	50
30.	Función cierre sesión	51
31.	Selección preferencias	52
32.	Dada de alta preferencias	53
33.	Descarga información de lugares	54
34.	Criterios de filtros	54
35.	Menú filtración	55
36.	Darse de baja (Ventana flotante)	56
37.	Darse de baja	56
38.	Estructura clase Place	57
39.	Asociación de información lugar a elementos gráficos	57
40.	Descarga imágenes	58
41.	Creación del lugar	59
42.	Selección de imágenes	60
43.	Selección coordenadas	60
44.	Dada de alta en BBDD del lugar	61
45.	Ventana flotante puntuación	62
46.	Subida de puntuación a BBDD	63

47.	Acceso a crear lista	64
48.	Filtros lugares	64
49.	Interacción con los elementos de la lista de lugares	65
50.	Subida de lista a BBDD	66
51.	Menú flotante editar/borrar listas	67
52.	Función borrado lista de la BBDD	67
53.	Ventana flotante de cambio nombre lista	68
54.	Botón agregar nuevos elementos / edición lista	69
55.	Opción de lista a editar	69
56.	Opción lista usuarios menú lateral	70
57.	Menú flotante para elementos de la lista usuarios	70
58.	Opciones de modificación de usuarios	70
59.	Borrado de datos del usuario	71
60.	Configuración elementos gráficos 1	72
61.	Configuración elementos gráficos 2	73
62.	Recopilación datos usuario	74
63.	Subida información a BBDD	75
64.	Opciones de usuario	76
65.	Comprobación de contraseña valida	77
66.	Dada de alta de información en BBDD	77
67.	Ventana flotante - dar de baja	78
68.	Dada de baja del usuario de la BBDD	78
69.	Obtención de preferencias seleccionadas por el usuario	79
70.	Actualización en BBDD de preferencias usuario	79
71.	Ejemplo gráfico de navegación	81
72.	Diagrama del patrón MVVM	82
73.	Pregunta 1.	86
74.	Pregunta 2.	86
75.	Pregunta 3.	87
76.	Pregunta 4.	87
77.	Pregunta 5.	88
78.	Pregunta 6.	88
79.	Pregunta 7.	89
80.	Pregunta 8.	89
81.	Pregunta 9.	90
82.	Pregunta 10.	90
83.	Pregunta 11.	91
84.	Pantalla acceso y pantalla registro	100
85.	Error de acceso	101
86.	Cierre de sesión	102
87.	Pantalla datos usuario y menú lateral	103
88.	Editar información usuario	104
89.	Dar alta lugar 1	105
90.	Dar alta lugar 2	106
91.	Pantalla lugar	108
92.	Edición lugar	109
93.	Pantalla filtros lugar	111
94.	Pantalla listas usuario 1	112

95.	Pantalla listas usuario 2	114
96.	Pantalla listas usuario 3	115

Referencias

- [1] Activadores de database - firebase. <https://firebase.google.com/docs/functions/database-events>.
- [2] Allow app installations from unknown sources - samsung — verizon. <https://www.verizon.com/support/knowledge-base-222186/>.
- [3] Apk (formato) - wikipedia, la enciclopedia libre. [https://es.wikipedia.org/wiki/APK_\(formato\)](https://es.wikipedia.org/wiki/APK_(formato)).
- [4] Backend as a service - wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Backend_as_a_service.
- [5] Cloud functions para firebase. <https://firebase.google.com/docs/functions>.
- [6] Cómo interpretar el ciclo de vida de una actividad. <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [7] Descripción general del manifiesto de una app. <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419>.
- [8] Descripción general del sdk de maps para android. <https://developers.google.com/maps/documentation/android-sdk/overview?hl=es>.
- [9] Extensible markup language - wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Extensible_Markup_Language.
- [10] Firebase - wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Firebase#Firebase_Storage.
- [11] Firebase realtime database — firebase realtime database. <https://firebase.google.com/docs/database?hl=es>.
- [12] Fragment lifecycle - android developers. <https://developer.android.com/guide/fragments/lifecycle>.
- [13] Function as a service - wikipedia. https://en.wikipedia.org/wiki/Function_as_a_service.
- [14] Git - wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Git>.
- [15] Github - wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/GitHub>.
- [16] Google forms - wikipedia. https://en.wikipedia.org/wiki/Google_Forms.
- [17] Introducción a android studio — desarrolladores de android. <https://developer.android.com/studio/intro?hl=es>.
- [18] Javascript — mdn. <https://developer.mozilla.org/es/docs/Web/JavaScript>.
- [19] Kotlin vs java — openwebinars. <https://openwebinars.net/blog/kotlin-vs-java/>.
- [20] Why 7 seconds could make or break your mobile app - phiture - mobile growth consultancy and agency. <https://phiture.com/asostack/why-7-seconds-could-make-or-break-your-mobile-app-f41000fb2a17/>.

Apéndice

Guía instalación de la aplicación

En este apartado se mostrará como instalar el archivo de extensión “.apk” [3] que se proveerá en la entrega del proyecto. Los archivos “.apk” son paquetes de software para el sistema operativo Android a través del cual se instalará la aplicación deseada.

El proceso es muy simple ya que solo es necesario dar permiso al smartphone de Android para instalar aplicaciones de *origen desconocido*. Esto se encuentra configurado de esta forma para evitar instalar aplicaciones fuera de las fuentes oficiales. Cada marca de smartphones tiene su propia forma de hacerlo pero en general todas son muy similares. Se mostrará el proceso para los teléfonos de la marca *Samsung* [2].

Android 8.x o superior

1. Accede a “Ajustes” y busca la opción “Apps”.
2. Toca el icono del menú (parte superior a la derecha).
3. Toca “Acceso especial”.
4. Toca “Instalar aplicaciones de origen desconocido”.
5. Selecciona la aplicación desconocida y pulsa el botón de “Permitir de esta fuente” para ponerlo en encendido.

Android 7.x o inferior

1. Accede a “Ajustes”.
2. Toca “Pantalla de bloqueo y seguridad”. En caso de que esta opción no esté disponible toca “Seguridad” a secas.
3. Toca el botón de “Fuente/Origen desconocido” para ponerlo en encendido.
4. Para continuar revise el aviso que aparecerá y pulse “OK”.

Guía de uso de la aplicación

Ahora se mostrará una guía de como funciona la aplicación para que quien lo desee pueda consultarla en caso de que tenga dudas de como funciona.

■ Login y registro

Lo primero que ve el usuario es la pantalla del login, si el usuario tiene una cuenta de esta aplicación ya existente simplemente necesita introducir su correo electrónico y su contraseña en los campos de texto primero y segundo respectivamente. Si alguno de esos datos fuesen erróneos aparecería una ventana indicando que los datos son erróneos.

Si el usuario no dispone de cuenta ha de pulsar el texto gris (figura 84) en la parte inferior de la pantalla del login para poder darse de alta como usuario. En esa pantalla tendrá que introducir su nombre (con o sin apellidos), su correo electrónico y deberá introducir dos veces la contraseña que desee (la segunda para garantizar que el usuario no se ha equivocado al introducirla). Tras hacer eso se mostrará al usuario una lista de preferencias donde el usuario tendrá que seleccionar mínimo una obligatoriamente para que la aplicación pueda recomendar en función de esa categoría seleccionada.

Tras realizar esa selección el usuario ya habrá sido dado de alta y automáticamente entrará en la aplicación con su cuenta recién creada.

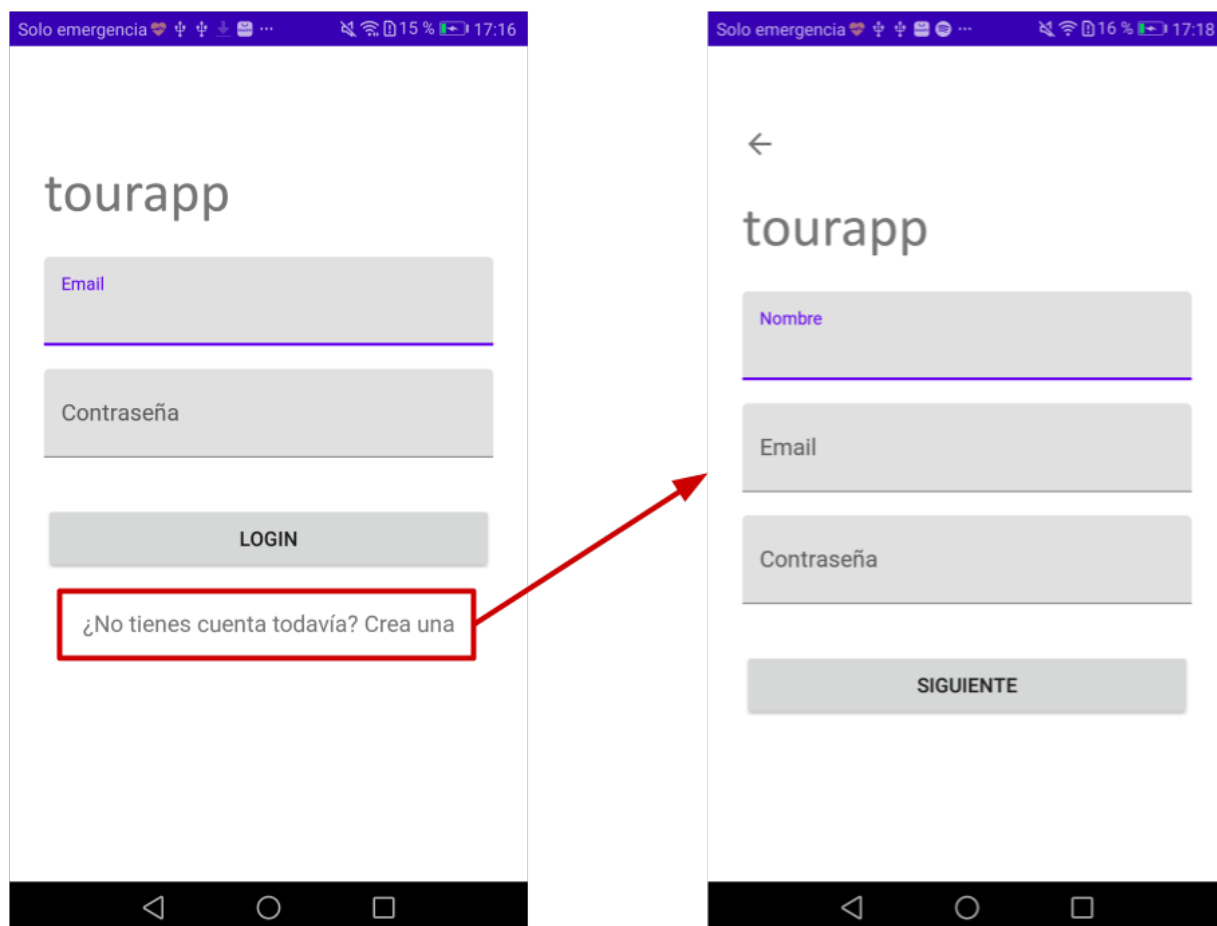


Figura 84: Pantalla acceso y pantalla registro

Si en los campos de acceso se introduce el email o la contraseña mal aparecerá el siguiente mensaje de la figura 85.

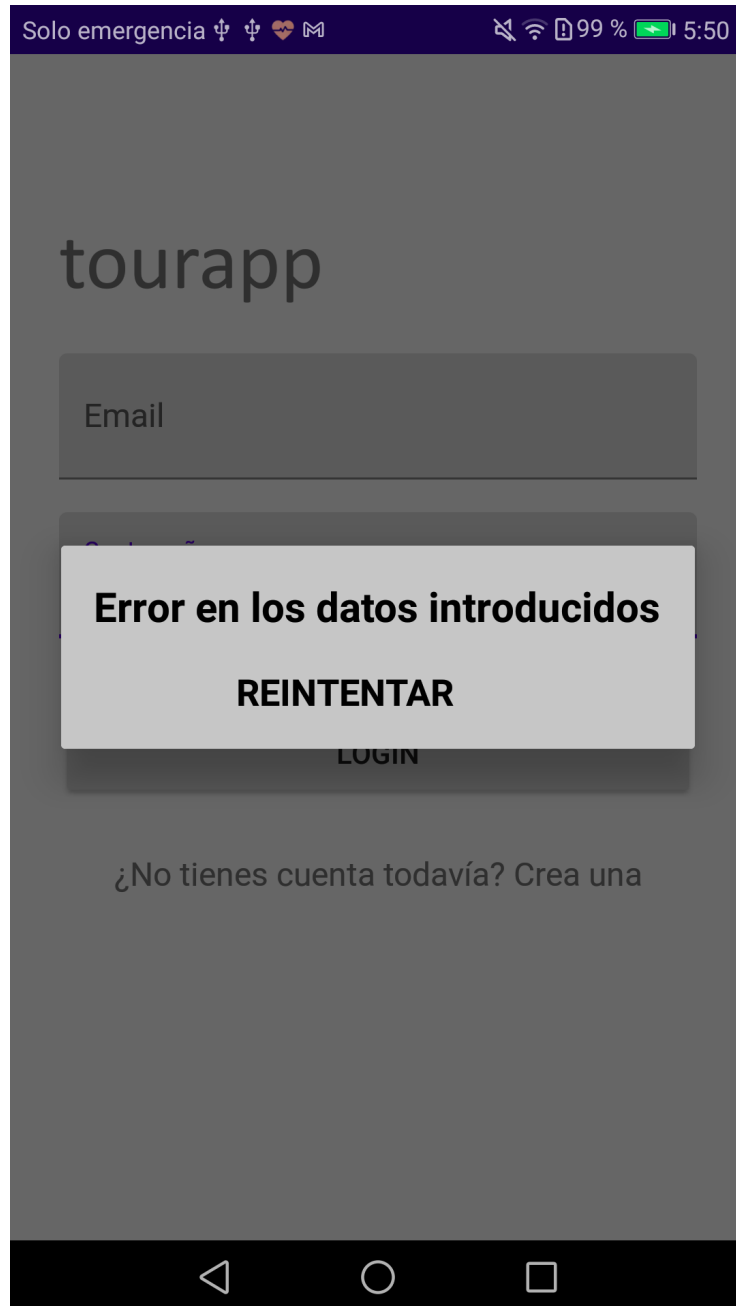


Figura 85: Error de acceso

■ Cerrar sesión

En el menú lateral abajo del todo aparece la opción de cerrar sesión. Al seleccionar esta opción aparecerá una ventana flotante preguntando al usuario si desea cerrar sesión. Al seleccionar "Aceptar" se le llevará de nuevo a la pantalla del login.

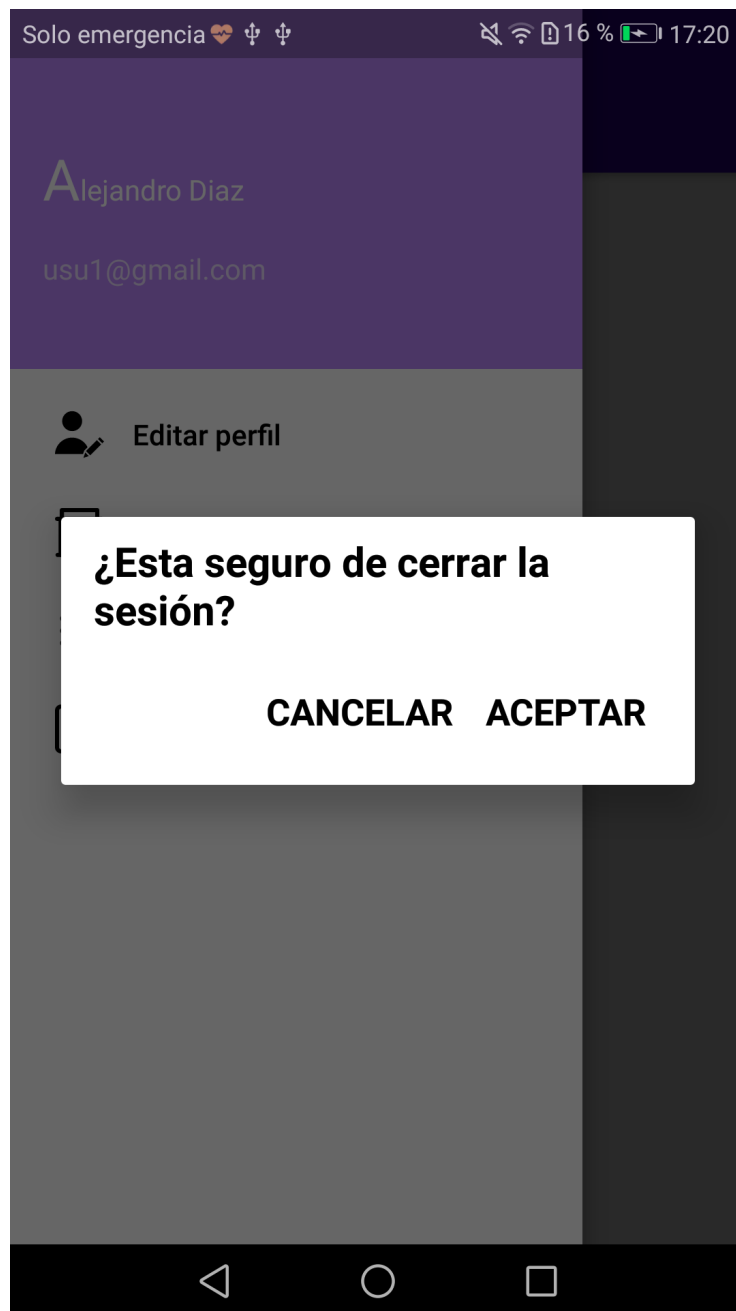


Figura 86: Cierre de sesión

■ Datos usuario y edición de datos

La primera pantalla (figura 87) que se le muestra al usuario es la pantalla de datos del usuario. En esta pantalla se muestran los datos del usuario que son su nombre, correo electrónico y tipo de usuario.

En la esquina superior izquierda de la pantalla se encuentra un botón de tres líneas horizontales con forma de hamburguesa, al pulsar ese botón aparecerá un menú lateral con varias opciones. Para hacer desaparecer el menú basta con tocar la parte de la pantalla que no está ocupada por el menú o deslizar con el dedo hacia la izquierda de la pantalla.

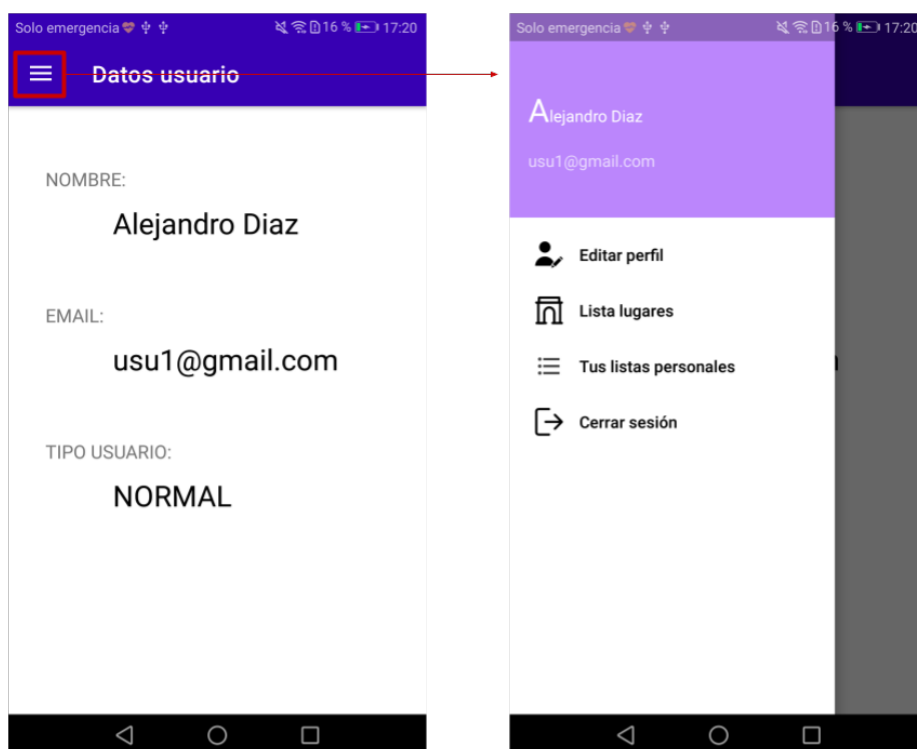


Figura 87: Pantalla datos usuario y menú lateral

Al pulsar la opción "Editar perfil" se llevará al usuario a una pantalla donde podrá editar todos los datos referentes a la información de su cuenta (figura 88). En la primera opción el usuario podrá editar su nombre y contraseña, en la segunda opción podrá editar sus preferencias y en la tercera opción podrá dar de baja su cuenta de los servicios de la aplicación, al hacer eso aparecerá una ventana preguntando el usuario si está seguro de su decisión.

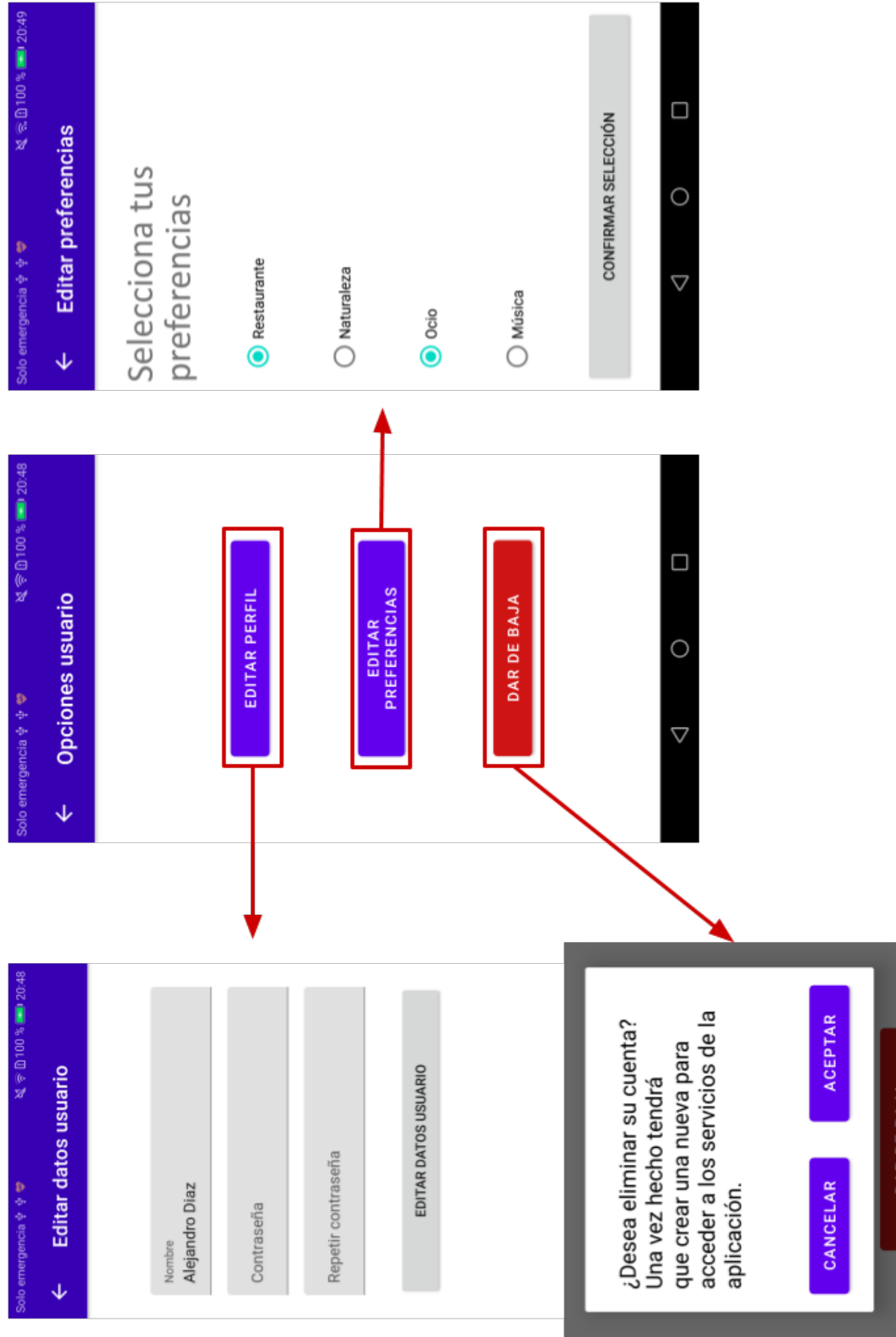


Figura 88: Editar información usuario

▪ Lista de lugares (dar de alta lugar)

La segunda opción del menú lateral es la lista de lugares, ahí mostrará los lugares que se hayan dado de alta por otros usuarios.

En la barra superior en la esquina derecha hay un botón con forma de tres puntos situados verticalmente que al pulsarlos muestran al usuario la opción de dar de alta un lugar.

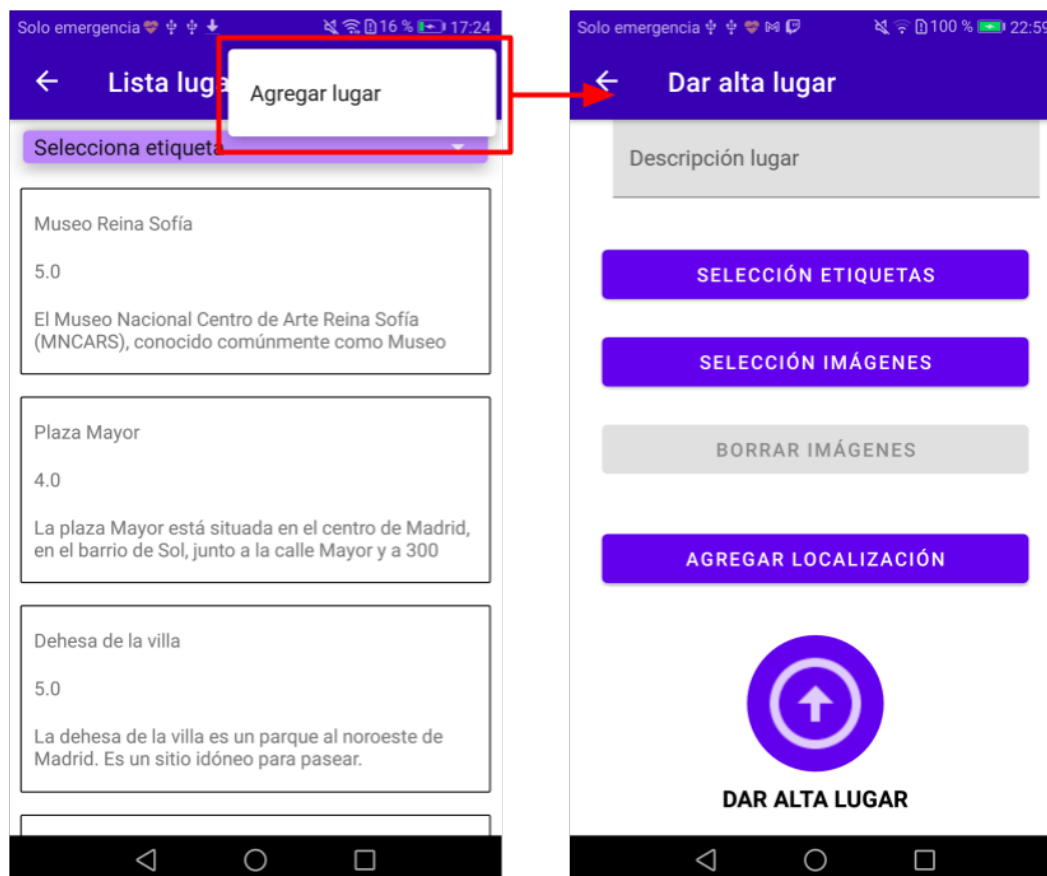


Figura 89: Dar alta lugar 1

Se llevará al usuario a una pantalla (figura 90) donde deberá introducir el nombre del lugar deseado, la descripción del mismo, una etiquetas o categorías para poder clasificar el lugar fácilmente. También se le puede agregar unas imágenes (máximo son cuatro) y debajo tiene un botón en el cual se pueden eliminar las mismas.

El botón agregar localización abre una vista de *google maps* donde es posible seleccionar una ubicación deseada y esta se almacenaría en la base datos de la aplicación. Para seleccionar el lugar deseado basta con mantener el dedo pulsado en ese sitio y aparecerá un marcador, tras eso se selecciona el botón inferior de agregar localización y la misma ya se ha guardado.

Por último para dar de alta el lugar basta con pulsar el botón inferior circular y la información se subirá, si faltase información obligatoria aparecería un mensaje en pantalla indicando que faltan datos y no se subiría la misma.

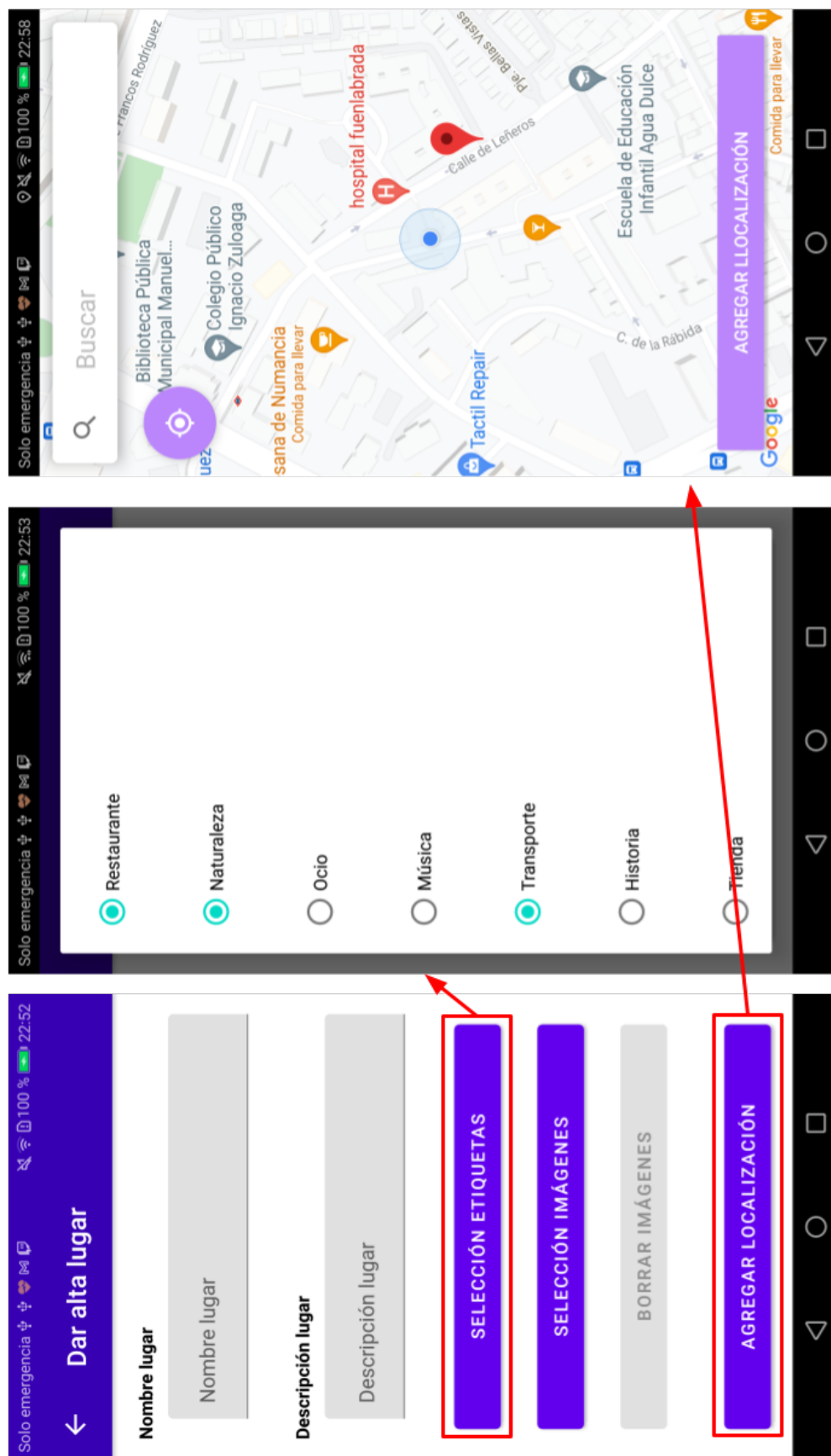


Figura 90: Dar alta lugar 2

■ Vista del lugar

La vista del lugar (figura 91) provee información del mismo, se accede a esta pantalla pulsando un elemento de la lista de lugares. Hay un botón en forma de corazón en la barra superior esta permite al usuario añadir el lugar a la lista de favoritos.

En la pantalla se puede interactuar con tres elementos, la puntuación de estrellas al pulsarla durante un tiempo mostrará una ventana flotante donde se podrá modificar la puntuación y subirla a la base de datos, o no hacer nada. Luego están los botones de comentarios donde se lleva al usuario a la pantalla de comentarios y el botón de localización que abre una vista de *google maps* donde el usuario puede ver la localización del lugar en cuestión en un marcador y si pulsa ese marcador se le dará la opción de buscar una ruta hacia ese lugar.

La pantalla de comentarios es muy similar a la de cualquier plataforma online. Se verán todos los comentarios de los usuarios y abajo del todo se encuentra una barra donde el usuario podrá escribir un comentario y subirlo. Los comentarios escritos por el propio usuario tendrán un botón disponible (en la esquina derecha de los mismos) donde se podrán eliminar o editar.

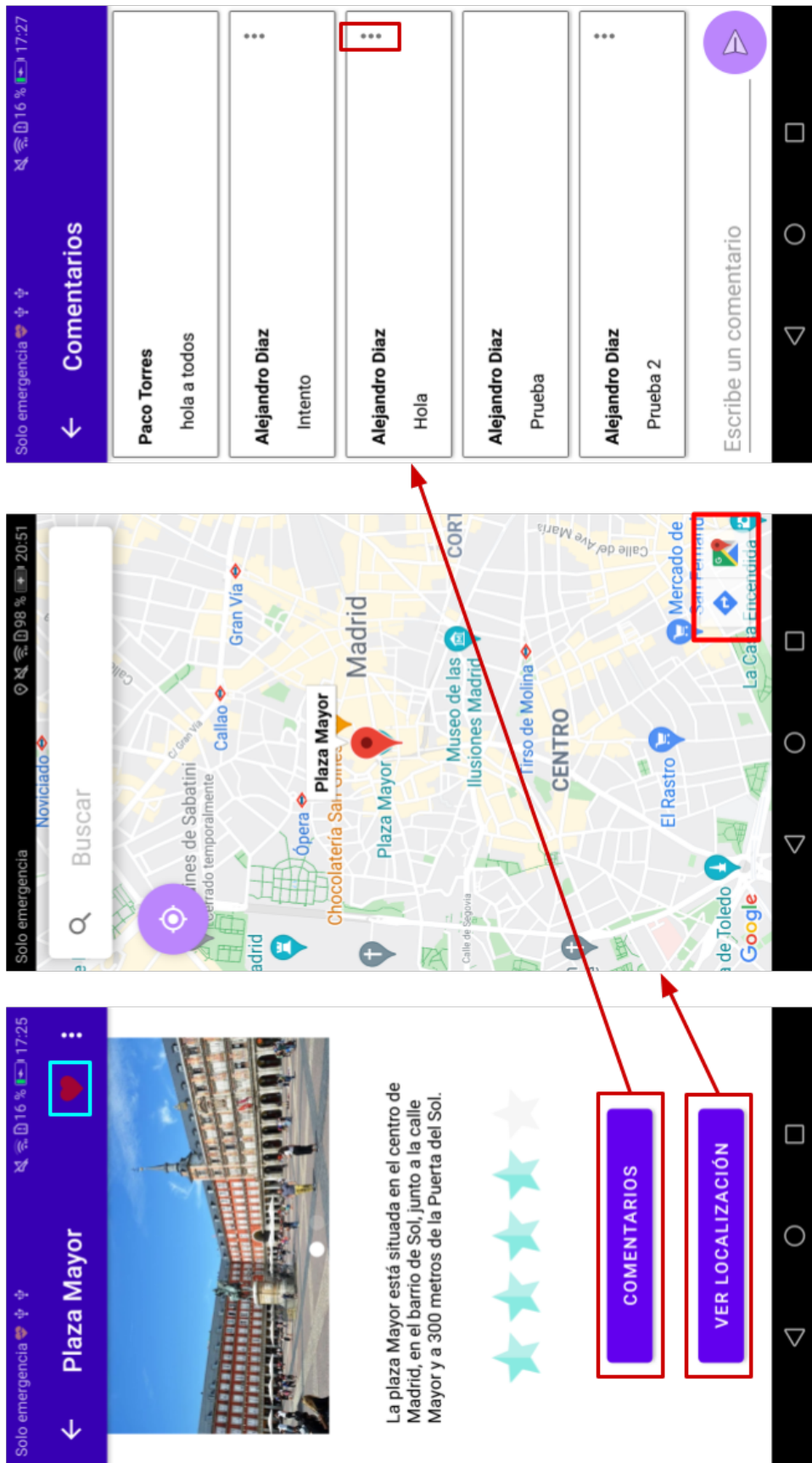


Figura 91: Pantalla lugar

▪ Lista de lugares (editar lugar)

Si eres en el creador de un lugar o un usuario tipo administrador, en la pantalla de la vista del lugar en la parte superior derecha de la misma aparecerá un botón en el que al pulsarlo aparecerá una ventana donde se dará la opción de eliminar el lugar en cuestión o editar el mismo. Si el usuario selecciona eliminar el sitio le mostrará una ventana preguntando al usuario si está seguro de su decisión.

Si selecciona editar lugar se le llevará a una pantalla igual a la de dar alta lugar donde podrá modificar la información del mismo y actualizarlos en la base de datos.

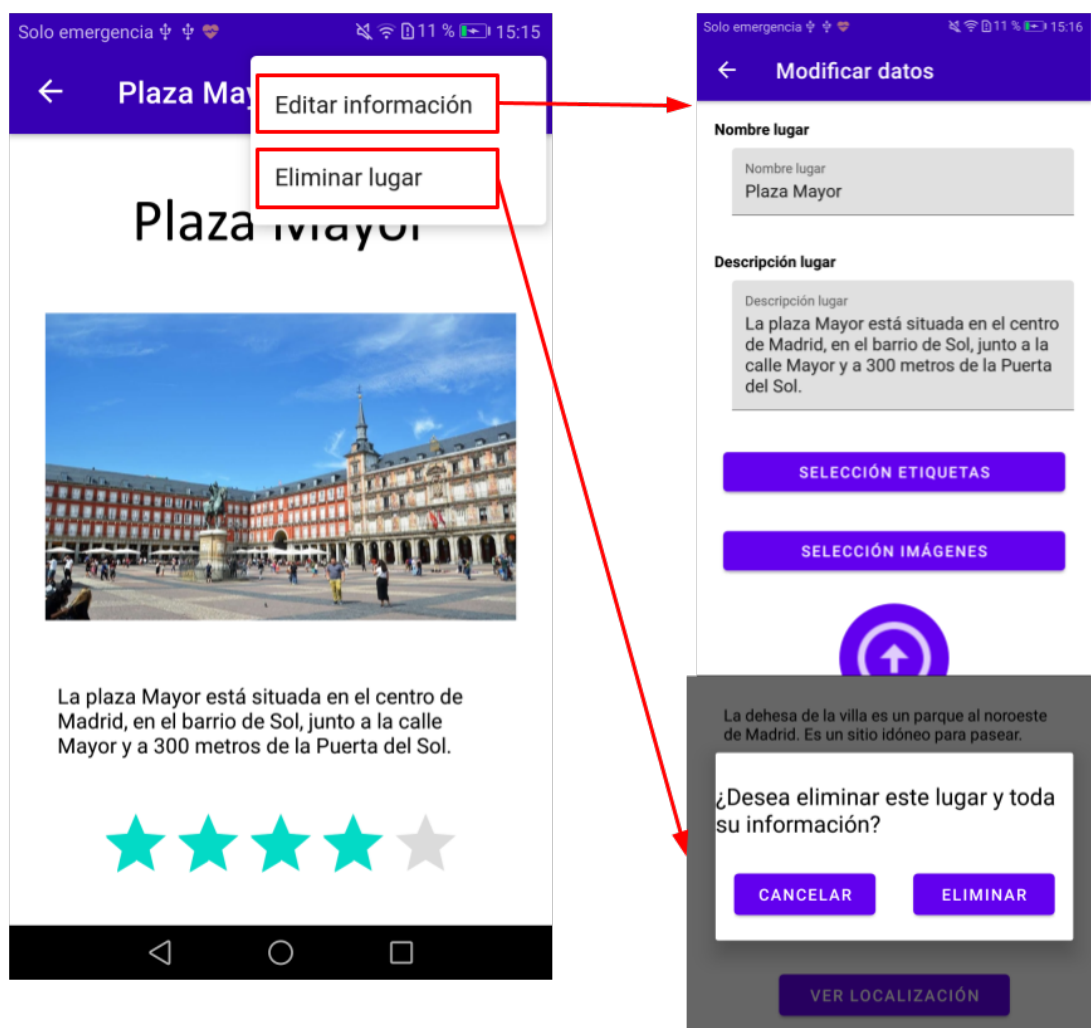


Figura 92: Edición lugar

▪ Lista de lugares (filtros)

En la barra superior (figura 93) hay el icono de una lupa la cual al pulsarla permite buscar lugares por su nombre, y justo al lado de ese icono hay un icono de filtros, el cual permite filtrar según los siguientes criterios: por categorías/etiquetas o en función a otros criterios, como la puntuación más alta, el número de categorías que coinciden con las del usuario que está utilizando la aplicación.

Justo debajo de esa barra hay otra barra de filtros que al pulsarla se despliegan todas las categorías de lugares existentes, al seleccionar una se filtran los lugares por la categoría seleccionada.

Es posible utilizar simultáneamente la búsqueda y los dos tipos de filtros distintos.

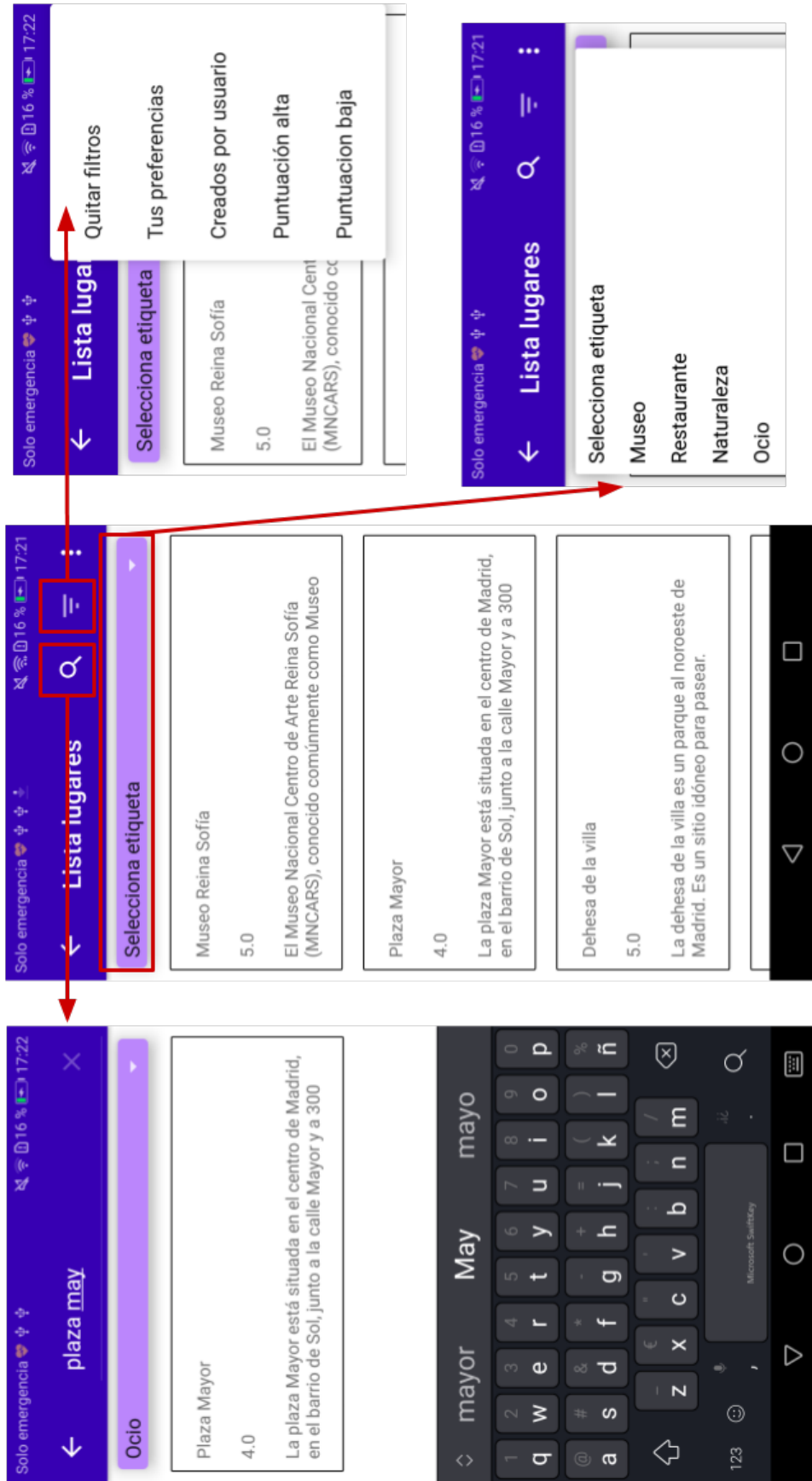


Figura 93: Pantalla filtros lugar

■ Listas personalizables

Aparecerán una colección de listas de lugares en las cuales los usuarios podrán almacenar los distintos lugares según la temática que ellos deseen. Aparecerá la lista de favoritos mencionada antes y se podrá crear una nueva de cero.

Las listas ya existentes permiten que se puedan renombrar, eliminar. Al pulsar el botón lateral del elemento de la lista aparecerá un menú ofreciendo al usuario ambas posibilidades.

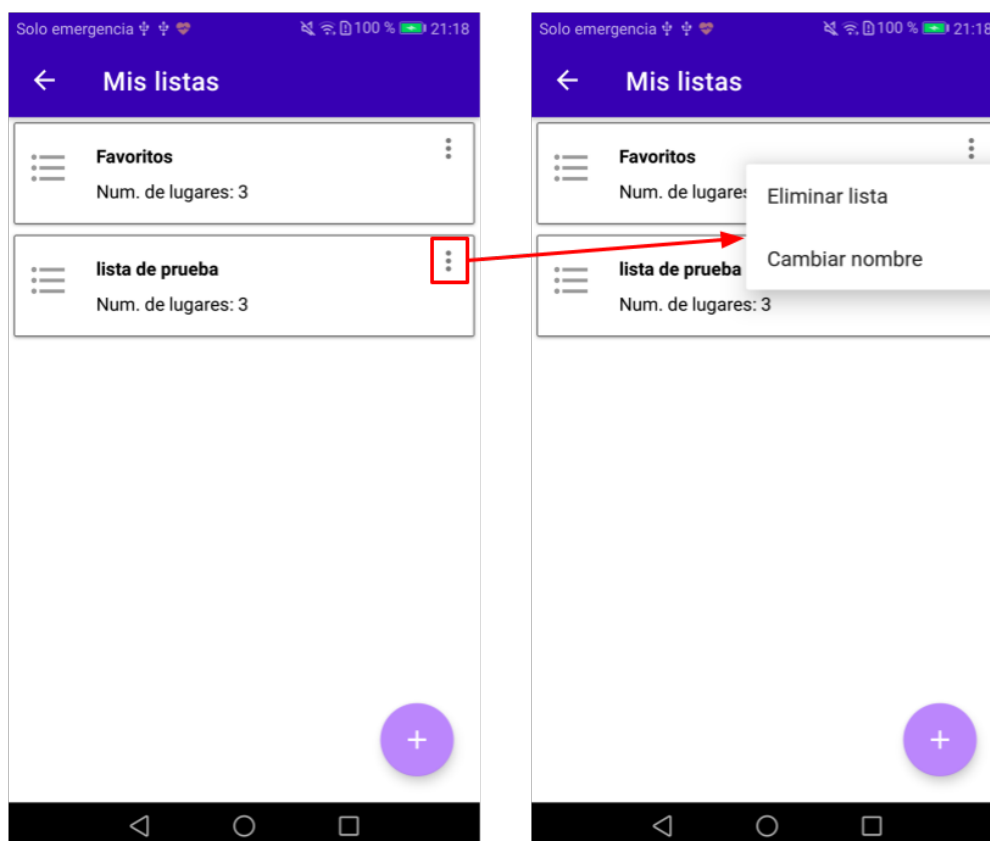


Figura 94: Pantalla listas usuario 1

Para crear una lista nueva en la esquina inferior derecha hay un botón circular (figura 95) que al pulsarlo lleva al usuario a la pantalla de creación de listas, la cual es muy similar a la pantalla de lista de lugares vista antes. Esta pantalla permite filtrar y buscar lugares como la otra y cada elemento de la lista tiene un botón con una "equis" o un "más", indicando si el lugar se ha añadido o no a la lista.

Al pulsar "confirmar lista" se le pedirá al usuario que le de un nombre, una vez hecho esto la lista de dará de alta en la base de datos y aparecerá entre las listas disponibles.

Al seleccionar una de las listas de lugares (figura 96) llevará al usuario a una pantalla igual a la de la lista de lugares (la que aparece disponible en el menú lateral). La pantalla permitiría ver los lugares pertenecientes a a lista, buscarlos y filtrarlos, además de que en la esquina inferior derecha hay un botón circular que permite editar la lista. Esta pantalla de edición es muy

similar a la de la figura 95 la cual permite seleccionar o de-seleccionar los elementos deseados. Al pulsar "confirmar lista" se actualizan los datos de la misma.

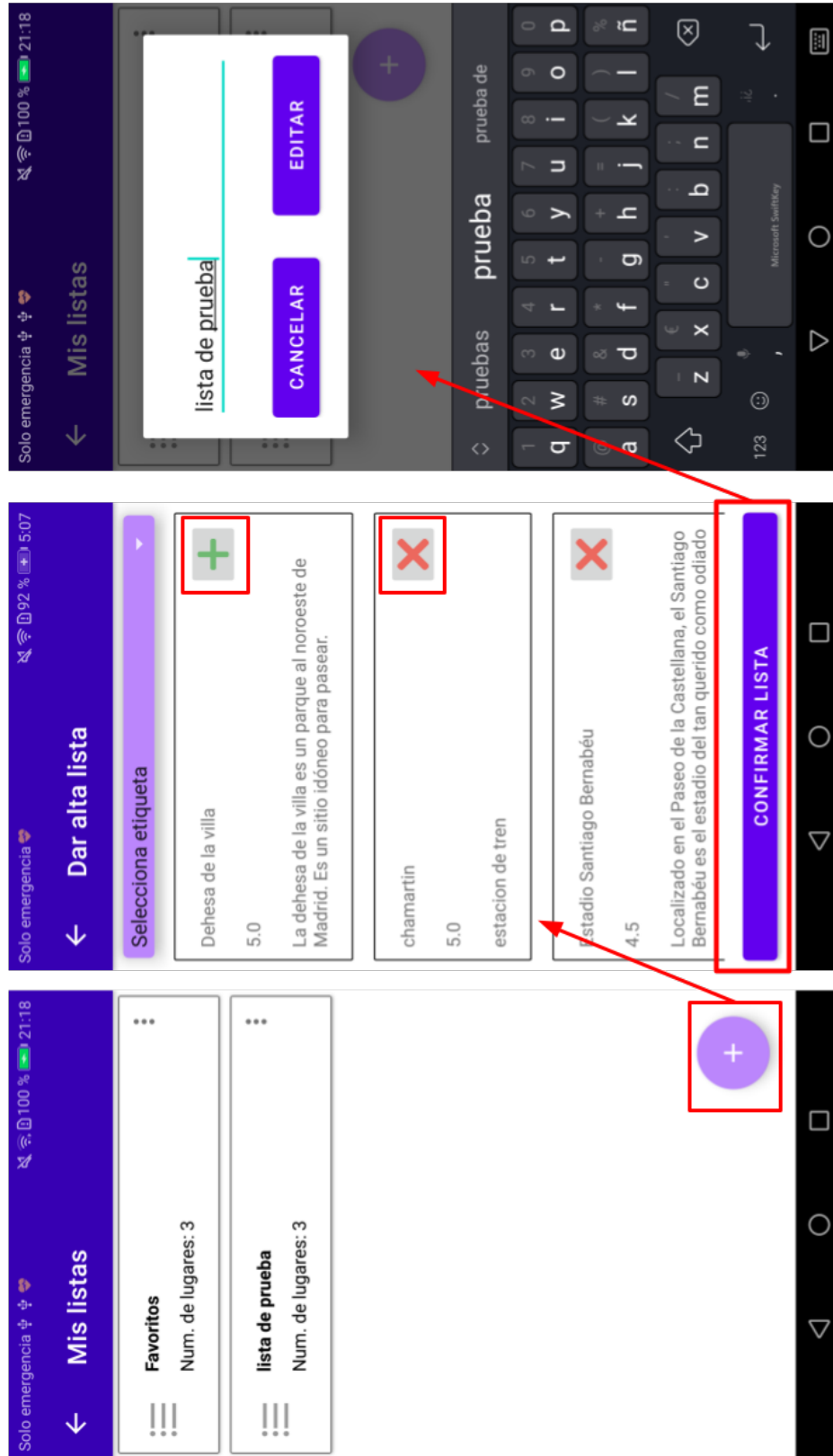


Figura 95: Pantalla listas usuario 2

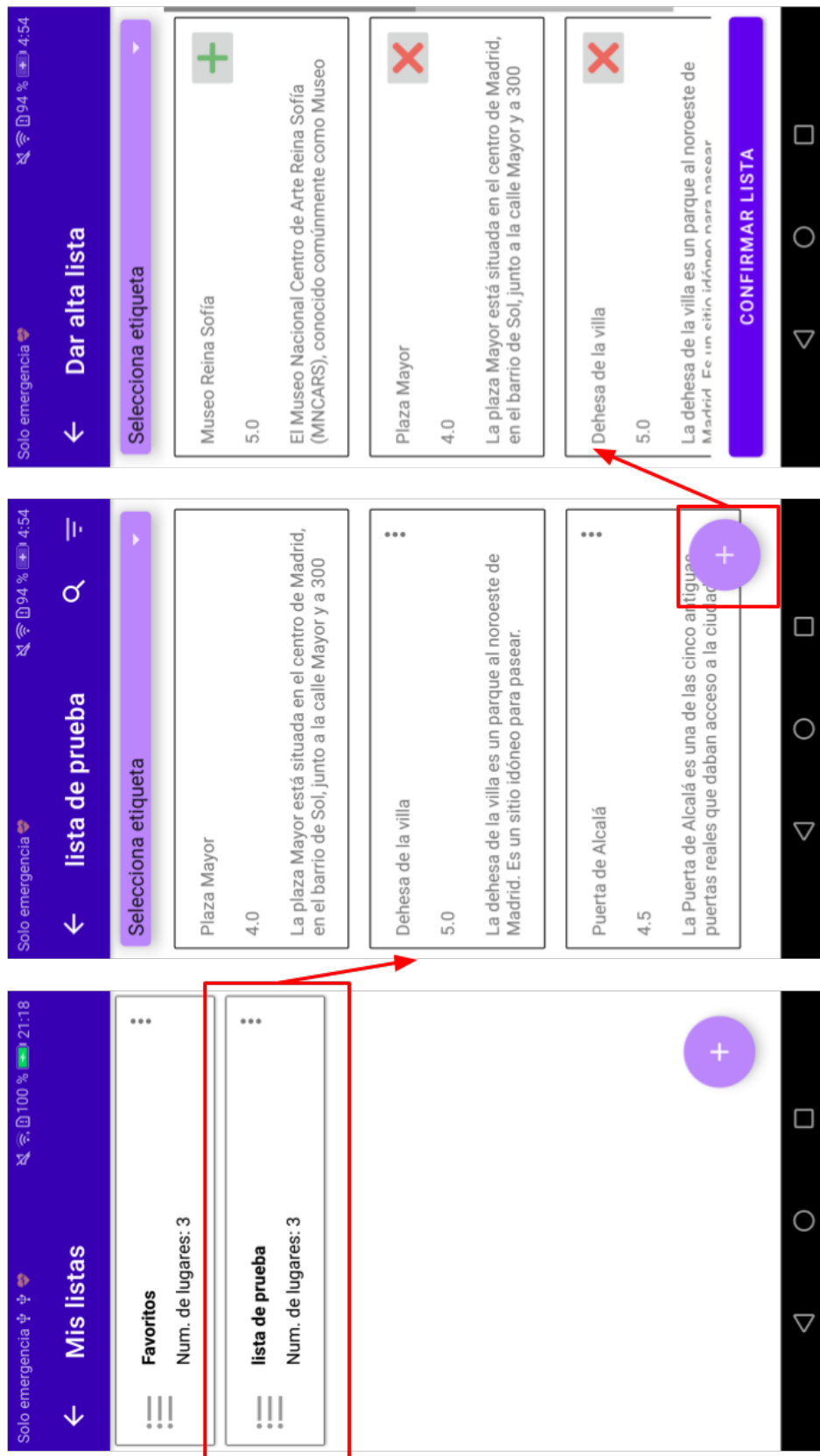


Figura 96: Pantalla listas usuario 3

Repositorio código - GitHub

El código fuente de la aplicación se encuentra disponible para consultar en el siguiente enlace de *GitHub*:

tourapp, GitHub - (<https://github.com/Alvox12/touristApp>).