ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

APLICACIÓN ANDROID PARA EL DESPLAZAMIENTO A LA UMA

ANDROID APPLICATION FOR THE DISPLACEMENT TO THE UMA

Realizado por
Rodrigo Represa Represa
Tutorizado por
Luis Manuel Llopis Torres
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Junio 2019

Fecha defensa:

Fdo.: El Secretario del Tribunal

Resumen

Hoy en día uno de los principales problemas que surgen a aquellos alumnos que se desplazan diariamente a las diferentes facultades y edificios de la Universidad de Málaga, sobre todo desde localidades anexas a la capital, es encontrar un medio de transporte eficaz y rentable que les permitan llegar a las aulas en un período razonable de tiempo.

En contra posición, existe una serie de alumnos que se desplaza semanalmente en vehículos privados, con la posibilidad de compartir algunas de las plazas libres de las que dispone.

Algo que tienen en común éstos alumnos es la facilidad de acceder a tecnologías ligadas con el sector de la telefonía móvil, como por ejemplo el sistema operativo Android, líder del mercado móvil a nivel mundial.

Este proyecto pretende ofrecer una solución estable y de calidad para aquellos usuarios de tecnologías Android que puedan facilitar a otros el transporte a cualquier campus o edificio de la Universidad de Málaga, así como para aquellos que deseen optar por un transporte privado compartido.

Además, la finalidad del proyecto también es la de promover el uso de vehículo compartido, favoreciendo a disminuir la contaminación, pues se pueden poner en contacto alumnos que dispongan ambos de vehículos privados y sin embargo prefieran compartirlo.

Palabras clave: Aplicación Android nativa, Firebase, movilidad, Smartphone, vehículo, compartir.

Abstract

Nowadays one of the main problems that appear in those students who travel daily to the different Faculties and buildings of the University of Malaga, especially from locations near the capital, is to find an efficient and economical transport that will allow them to reach the classrooms in a reasonable period of time.

On the other hand, there is a series of students who travel weekly in private vehicles with the possibility of sharing some of the available vacancies.

Something that these students have in common is the possibility of accessing technologies associated with the mobile telephony sector, such as the Android operating system, leader in the mobile market worldwide.

This project offers a stable and quality solution for users of Android technologies that can facilitate other users transport to any campus or building of the University of Malaga and for those who wish to opt for a shared private transport.

In addition, the purpose of the project is also to promote the use of shared vehicles that reduce pollution, students who enjoy both private vehicles and prefer to share it can be contacted.

Keywords: Native Android application, Firebase, mobility, Smartphone, vehicle, share.

Índice General

Índice Ilustraciones

Índice Tablas

Capítulo 1
Introducción

1.1 Motivación

1.2 Objetivos

1.3 Estructura de la memoria

1.4 Estado del arte

1.5 Tecnologías a utilizar

- 1.5.1 Android
- 1.5.2 Firebase
- 1.5.3 Google Cloud Platform
- 1.5.4 Java
- 1.5.5 Nodejs

1.1 Motivación

Es un hecho contrastado que hoy en día el acceso a tecnologías móviles en poblaciones desarrolladas es cada vez más fácil. En España, el porcentaje de usuarios poseedores de móviles asciende a aproximadamente el 80% de la población total. Éste desarrollo tecnológico permite que cada vez sea más frecuente encontrar soluciones o ayudas tecnológicas a problemas diarios que antes no encontraban solución.

Un campo que no se ve exento de éste crecimiento tecnológico es el campo de la movilidad. Todas las personas en algún momento de su vida necesitan hacer uso de algún tipo de transporte, ya sea privado o público, para llegar a un destino. Forma parte de nuestra vida diaria y es normal que a medida que avanza la tecnología, ésta se aplique a la movilidad para mejorarla y facilitar el acceso al transporte.

Este desarrollo tecnológico permite que hoy estemos a solo unos pocos gestos a través de nuestros dispositivos móviles de acceder a un vehículo. Sin embargo, hay casos en los que la tecnología no se ha puesto al servicio del usuario, existiendo problemáticas ligadas al transporte que podrían ser resueltas.

Una de está problemáticas es el acceso a la Universidad de Málaga por parte de aquellos usuarios que carecen de vehículo privado. Es por ello que la principal motivación de éste proyecto es facilitar, por medio de una aplicación Android, una solución para que usuarios de la Universidad de Málaga puedan compartir vehículo de forma eficaz, favoreciendo además a la disminución de emisión de gases de efecto invernadero.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene como objetivo el desarrollo de una aplicación Android nativa para la creación de viajes, con destino a algún campus o edificio de la Universidad de Málaga, en la que cada usuario pueda acceder a través de su cuenta de Google y unirse a aquellos viajes que más le convengan.

Por lo tanto, como objetivo principal encontramos la elaboración de un sistema que permita poner en contacto a aquellas personas poseedoras de un vehículo, con aquellas personas que quieran hacer uso de alguna plaza para llegar al destino que más les convenga.

El hecho de facilitar la puesta en contacto de conductor y pasajeros supone que, como objetivo secundario, encontremos la creación de una interfaz de usuario que permita al usuario final una consecución fácil e intuitiva de sus objetivos.

Por último destacar que, aunque no sea objetivo principal, es importante saber que el desarrollo de ésta aplicación está ligada a una mayor concienciación sobre el medio ambiente y su cuidado.

1.3 Estructura de la memoria

La memoria se ha dividido en dos partes: la principal, estructurada por capítulos y que contiene cada etapa del desarrollo software y, una última parte, dos anexos destinados al usuario final.

El contenido principal de la memoria ha sido dividido por capítulos:

Capítulo 1. Introducción: en este capítulo se hace una pequeña explicación del proyecto, qué es lo que ha motivado a su realización y qué se pretende conseguir con él. Se habla sobre la estructura de la memoria y sobre las tecnologías utilizadas para el desarrollo del proyecto, así como de las diferentes soluciones que encontramos en el mercado para abordar la problemática de la movilidad.

Capítulo 2. Planificación: primera de las etapas del desarrollo software. Se explica cómo se ha abordado esta fase y la metodología utilizada en el desarrollo del producto.

Capítulo 3. Análisis: en éste capítulo encontramos plasmados todos los requisitos que se han ido recogiendo a lo largo del desarrollo de los prototipos, que han dado lugar al sistema final. Fruto de esos requisitos también se muestran los casos de uso a implementar.

Capítulo 4. Diseño: para un mejor entendimiento del proyecto, se explica la arquitectura utilizada. Por otro lado, también encontramos diferentes claves que se han tenido en cuenta a la hora de diseñar tanto el cliente como el servidor.

Capítulo 5. Implementación: en este capítulo se hace un recorrido por cada una de las pantallas que conforman la aplicación, explicando con más detalle su funcionamiento e implementación. Para concluir, se explican brevemente las librerías que respaldan la implementación.

Capítulo 6. Pruebas: éste capítulo es un reflejo de los documentos que surgen a la hora de realizar las pruebas con los usuarios finales. En él encontramos los diferentes errores surgidos y nuevas funciones a implementar.

Capítulo 7. Conclusiones: opinión personal sobre el proyecto. Se hace un recorrido personal sobre lo que ha supuesto el desarrollo del trabajo, desde que se tiene la idea hasta su realización.

Capítulo 8. Líneas Futuras: se muestran posibles mejoras del proyecto así como ideas nuevas a implementar.

El resto de la memoria lo conforman dos anexos: un manual de instalación para facilitar esta tarea al usuario final, y un manual de usuario con una explicación de las diferentes acciones y pantallas.

1.4 Estado del arte

Dado que la aplicación que se desarrolla en este proyecto está enfocada a viajar a destinos relacionados con la Universidad de Málaga, no se encuentra ninguna aplicación en el mercado que cubra exactamente ésta necesidad. La mayoría de aplicaciones están centradas en la compartición de vehículo para trayectos largos. Sin embargo, dado que podrían ser utilizadas para éste caso de uso, se va a proceder a analizarlas.

BlaBlaCar

Principal aplicación para la compartición de vehículos en España. Con más de 10 millones de descargas, permite a los conductores viajar a cualquier destino deseado, ahorrándose lo invertido en gasolina, mediante el alquiler de las plazas de su vehículo.

BlaBlacar [1] hace uso de una interfaz bonita y amigable para facilitar la tarea de creación o búsqueda de viajes. Además permite la puesta en contacto con los pasajeros de un mismo vehículo a través de un chat. El principal problema que tiene, aplicado al caso de uso que se expone en éste proyecto, es que está destinado a trayectos largos, pudiendo acceder a la Universidad de Málaga a cambio de un coste elevado, por lo que no sería rentable para el usuario.

Amovens

Plataforma española para el alquiler y renting de coches a particulares y compartición de vehículos. Si bien es cierto que nació para dar solución a viajes diarios con destino a Universidades y empresas, hoy en día permite viajar por todo el territorio nacional.

Mediante una interfaz sencilla y soporte tanto móvil como web, Amovens [2] permite acceder a trayectos que vayan a realizar otras personas o crear uno de una forma fácil. Aplicándola al hecho de moverse diariamente a la Universidad, surge el mismo problema que con BlaBlacar, el coste semanal sería bastante alto, pues los usuarios buscan beneficiarse con los trayectos.

Journify

Su reclamo principal es dar solución a aquellas personas que realicen diariamente el mismo trayecto, como puede ser ir a la Universidad. Con una interfaz colorida y quizás poco intuitiva podemos crear viajes como conductor hacia destinos que realicemos diariamente.

A priori, Journify [3] parece ser la aplicación que más conviene a aquellos usuarios que diariamente viajen a la UMA, sin embargo todavía no se encuentra disponible en Málaga, por lo que no se adapta a las necesidades del proyecto.

1.5 Tecnologías a utilizar

En este apartado se expondrán las diferentes tecnologías que han hecho posible el desarrollo del proyecto. A pesar que más adelante se detallará el uso de alguna de estas tecnologías, se va a proceder a explicar en qué consisten y sus características principales.

1.5.1 Android

Android [4] es un sistema operativo basado en Linux, un núcleo de sistema operativo libre, multiplataforma y gratuito. Principalmente diseñado para su uso en teléfonos móviles y tabletas, hoy en día es común su implementación en relojes inteligentes, televisiones o automóviles.

Tras la compra por parte de Google en 2005, éste sistema operativo comienza a posicionarse como uno de los sistemas operativos móviles más vendidos, puesto que ocupa desde 2010 hasta la actual fecha.

En noviembre de 2007 junto con la primera versión de Android, es lanzado el kit de desarrollo de software SDK, que abriría nuevas posibilidades en el mercado al permitir a los desarrolladores la creación de sus propias aplicaciones.

¿Por qué Android?

Android año tras año se consolidad como el sistema operativo más utilizado en España. A finales del cuarto trimestre de 2018 se calculó que el porcentaje de teléfonos móviles inteligentes con sistema operativo Android era del 89,9%, como se indica en la Ilustración 1. Viendo estas cifras parece obvio que si se quiere desarrollar una aplicación destinada al mercado español, se prefiera desarrollar antes su versión bajo el sistema operativo Android que bajo otro sistema operativo.

Ventajas desarrollar en Android

- Gracias a que Android es un proyecto de software libre y por tanto pueda instalarse en la mayoría de dispositivos actuales, se asegura que el mercado disponible siempre va ser alto.
- Android permite la instalación de aplicaciones de terceros bajo responsabilidad del usuario, facilitando que en fase de desarrollo se pueda distribuir el proyecto para realizar pruebas en diferentes dispositivos.
- Uno de los lenguajes principales para desarrollar aplicaciones nativas en Android y utilizado en éste proyecto es Java, a día uno de los lenguajes de programación más utilizados en todo el mundo.
- Debido a su gran cantidad de usuarios, Android cuenta con una de las mayores comunidades de desarrollo a nivel mundial, algo que hace que sea fácil consultar en Internet tutoriales, pedir ayuda o consejo…
- Android pertenece a Google, una de las empresas tecnológicas más grandes del planeta. Esto permite que tenga un gran apoyo y respaldo y esté continuamente en crecimiento aumentando las posibilidades para los desarrolladores.
- Relacionado con el punto anterior, Android cuenta con entorno de desarrollo específicamente creado para él, Android Studio. Ésta herramienta facilita la creación de aplicaciones, así como la implementación de herramientas directamente relacionadas con Google como Firebase.
- Como hemos mencionado, la comunidad Android es enorme, esto hace que el número de librerías que se crean para éste sistema operativo, de las que pueden hacer uso los desarrolladores, sea cada vez mayor.

Desventajas desarrollar en Android

- La fragmentación es a día de hoy uno de los principales problemas de Android. Existen una gran cantidad de versiones publicadas en el mercado, y no todas las aplicaciones están optimizadas para aquellas versiones más antiguas. Para asegurase de que la aplicación que se ha desarrollado es instalable en la mayoría de los teléfonos Android, se ha utilizado con SDK mínimo la versión 21, pudiendo ser instalable en el 84% de los dispositivos mundiales.

- El hecho de que sea un sistema operativo de código libre, permite que existan infinidad de fabricantes que hagan uso de él. Esto supone que cualquier aplicación vaya a correr en terminales con diferente hardware y funcionalidades, al contrario que ocurren con otros sistemas operativos.

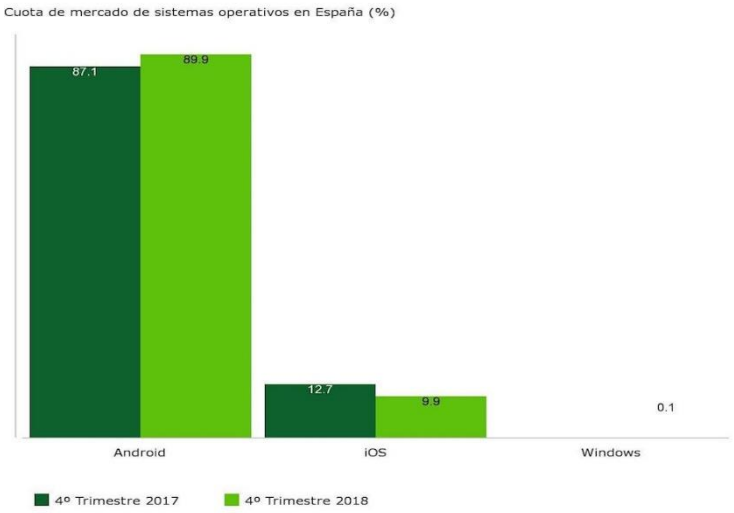

Ilustración 1: cuota de mercado Android en España

Fuente https://eloutput.com/noticias/moviles/telefonos-android-espana-cuota-2018/

1.5.2 Firebase

Firebase [5] es una plataforma Sas (Software como Servicio) adquirida por Google en 2014, con la que se pone a disposición al usuario de una forma fácil, eficaz y gratuita el acceso, almacenamiento y sincronización de datos en la nube para todo tipo de aplicaciones. Son numerosos los recursos que ésta plataforma ofrece al usuario para complementar sus proyectos. En el capítulo de diseño se verá más en profundidad las diferentes tecnologías de las que éste proyecto se beneficia a través de Firebase.

A continuación se expondrán una serie de ventajas y desventajas del uso de ésta plataforma:

Ventajas

- Permite ahorrarse la implementación del servidor. Ofrece dos tipos de bases de datos para usar en los proyectos y todos los métodos de acceso a ellos mediante la implementación de librerías.
- Pone al servicio del usuario una API para realizar la autenticación. A través de la consola y de la implementación de librerías se puede crear un sencillo flujo de autenticación.
- Facilita la creación de funciones en el servidor. Firebase permite desarrollar en diferentes lenguajes, funciones que serán activadas por el cliente a través de una conexión HTTTP o directamente que se ejecuten periódicamente.
- A la hora de realizar el mantenimiento de un proyecto, Firebase es capaz de detectar en qué APIs se han producido fallos y permite acceder a un informe para la reproducción del fallo facilitando sus corrección.

Desventajas

- Si el flujo de datos es muy grande se verá colapsada la conexión, ya que el número de conexiones simultáneas permitidas gratuitamente es de 100.
- Posee un límite de almacenamiento.

1.5.3 Google Cloud Platform

Cloud Platform [6] es la IaaS (Infraestructura como Servicio) que pone a disposición de los usuarios el gigante Google para proveerles de diversos productos, herramientas y servicios, para implementar en aplicaciones.

Gracias a ésta plataforma se puede hacer uso de las diferentes API que ofrece Google mediante la activación en la consola web. Además permite a los usuarios hacer un seguimiento de las estadísticas de uso de los diferentes proyectos y pone a su disposición numerosas herramientas de mantenimiento.

En éste proyecto, el uso que se le ha dado a ésta plataforma es la activación de diferentes APIs relacionadas con la localización, que son claves para la consecución de un proyecto que cumpla con los requisitos y contemple todos los casos de uso.

1.5.4 Java

El principal lenguaje de programación que se ha utilizado para el desarrollo de la aplicación cliente es Java [7]. Nacido en 1995, su característica principal es la orientación a objetos. Gracias a sus numerosas ventajas como ser un lenguaje multiplataforma y compilado, Java se ha convertido hoy en día en uno de los principales lenguajes de programación a nivel mundial.

1.5.5 Nodejs

A pesar que para el servidor se ha utilizado Firebase, en el apartado de funciones ha sido necesario el uso de tecnologías de programación. Para estas tareas se ha optado por Nodejs [8]. Este entorno de ejecución para JavaScript permite la programación asíncrona, evitando la creación de sistemas basados en hilos y sus problemas derivados.

Capítulo 2
Planificación

2.1 Planificación del proyecto

2.1 Planificación del proyecto

Antes de comenzar con las etapas del desarrollo software, es importante abordar la etapa de planificación. En la ingeniería de software una buena planificación es sinónimo de haber escogido una metodología que se ajusta a los diversos factores que inciden en un proyecto: tiempo disponibles, recursos, requisitos…

Dado que el proyecto se realizará en un corto período de tiempo y con la mayoría de requisitos definidos desde un primer análisis, parece que lo más obvio es escoger una metodología evolutiva [9], que son aquellas que entrelazan las etapas de desarrollo y validación, permitiendo el desarrollo rápido de un sistema inicial para luego ser mejorado poco a poco.

Dentro de las metodologías evolutivas, existen una gran cantidad de modelos diferentes que se adaptan a las necesidades de cada proyecto. Para el desarrollo de ésta aplicación se ha elegido el modelo de prototipos, basado en el desarrollo de un primer sistema que se ajusta a los requisitos recogidos para, una vez realizados las pruebas necesarias con el usuario final, implementar nuevos requisitos y mejorar el sistema a través de los diferentes prototipos.

En el modelo de prototipos, antes de la creación de cada prototipo, es necesario realizar las siguientes etapas:

- **Análisis**: consiste en sintetizar los requisitos necesarios.
- **Diseño Rápido**: diseñar de forma eficaz la forma de implementar los requisitos recogidos.
- **Implementación**: desarrollar e implementar los requisitos.
- **Retroalimentación**: tras la prueba por parte del usuario final, se analiza su retroalimentación para volver a desarrollar otro prototipo si es necesario.

Para la realización de este proyecto se han llevado a cabo dos prototipos y un sistema final, por lo que es lógico que tanto el análisis, como la implementación y pruebas se han tenido que repetir, adaptándose así a la retroalimentación del usuario final.

Como usuario final se ha escogido un grupo conformado por cuatro estudiantes de la Universidad de Málaga, que han ido plasmando sus necesidades y opiniones a lo largo del desarrollo.

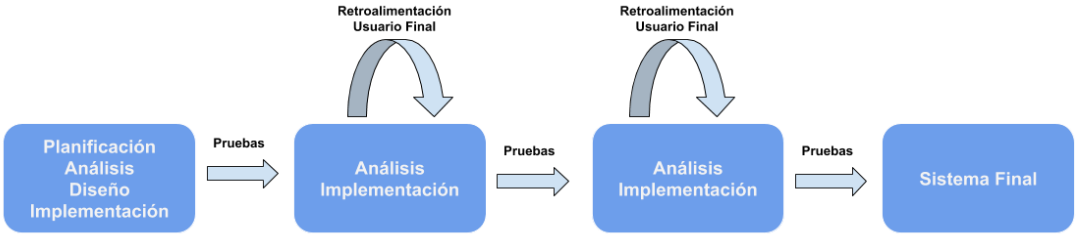

Ilustración 2: planificación evolutiva del proyecto

Fuente: elaboración propia

Capítulo 3
Análisis

3.1 Fase de análisis

Tras la elección de una metodología que se ajuste al proyecto, se lleva a cabo la fase de análisis. En esta etapa se recogen los requisitos necesarios para un correcto desarrollo del proyecto y que serán implementados en cada prototipo. La fase de análisis supone la parte más importante de un proyecto, pues es la base sobre la que se sustentan todas las demás etapas, si no se recogen bien los requisitos necesarios, la aplicación final no cumplirá con las funciones demandadas.

Como se ha mencionado antes, el hecho de implementar una metodología evolutiva, supone que puedan aparecer nuevos requisitos a implementar tras cada prueba de prototipos por parte del usuario final, por lo que a continuación se desglosará cada una de las fases de análisis previas al desarrollo de cada prototipo.

3.2 Análisis previo al primer prototipo

Antes de comenzar con el desarrollo del sistema, se lleva a cabo una recogida de requisitos necesarios para que la aplicación cumpla con sus funcionalidades principales. A continuación se mostrarán tanto los requisitos funcionales, que son aquellos que definen el comportamiento del sistema bajo unas determinadas condiciones, como los requisitos no funcionales, los que representan funcionalidades o restricciones generales de un sistema.

3.2.1 Requisitos funcionales

Gestión Usuarios

RF01	**Inicio sesión**

Versión	**1.0**
Descripción	1. El usuario debe ser capaz de iniciar sesión en la aplicación mediante sus datos de identificación de la plataforma Google. 2. El propio sistema deberá realizar la validación de los datos a través de una API. 3. El sistema debe ser capaz de permitir iniciar sesión con tantas cuentas de Google como haya introducidas en el sistema Android en ese momento. 4. El sistema debe ser capaz de recoger la respuesta a la autenticación por parte de la API, en caso de no ser correcta, deberá impedir el uso de la aplicación, en caso contrario, permitirá el uso de la aplicación procediendo a un almacenamiento de los datos del usuario en la Base de datos. 5. En caso de una autenticación correcta, el sistema redirigirá al usuario a un tutorial o a la pantalla principal, dependiendo de si es la primera instalación de la aplicación o no. 6. En caso de estar iniciada la sesión, la aplicación se encargará de redirigir al usuario a la pantalla inicial mostrando un mensaje de sesión iniciada. 7. En caso de que exista un problema con la comunicación con la API, el sistema deberá notificarlo al usuario.

Tabla 1: RF01 - Inicio Sesión

RF02	**Cierre de sesión**
Versión	**1.0**
Descripción	1. Todo usuario que se encuentre en uso de la aplicación con una sesión iniciada deberá poder cerrar la sesión. 2. Una vez el usuario pulse el botón “Cerrar sesión”, éste deberá ser redirigido a la pantalla principal. 3. Si la aplicación es abierta sin ninguna sesión iniciada, se deberá mostrar un botón con dicha acción.

Tabla 2: RF02 - Cierre de sesión

Funcionamiento

RF03	**Pantalla “Viajes”**
Versión	**1.0**
Descripción	1. La pantalla principal deberá contener como nombre la palabra “Viajes” y ésta deberá ser mostrada al usuario para una fácil identificación del contenido. 2. La pantalla principal “Viajes” deberá constar de: I. Todos aquellos viajes que ha realizado o va a realizar el usuario cuya sesión está iniciada.

	II. Existirá alguna forma de diferenciación entre “Viajes Terminados” y “Viajes sin Terminar”. III. Por cada uno de los viajes deberá mostrarse como mínimo la siguiente información: a. Foto del conductor b. Nombre del conductor c. Horas de salida d. Hora aproximada de llegada e. Origen f. Destino g. Pequeña información acerca del vehículo para una fácil identificación h. Pequeña información adicional. 3. Al pulsar sobre un viaje “Terminado” no ocurrirá nada. 4. Al pulsar sobre un viaje “Sin Terminar” deberá mostrarse un diálogo con el siguiente contenido: I. Un botón que permita al usuario visualizar el recorrido en Google Maps. II. En caso de que sea conductor el usuario con la sesión iniciada, deberá permitir eliminar el viaje. En caso contrario, deberá ser permitido dejar de ser pasajero en ese viaje

Tabla 3: RF03 - Pantalla Viajes

RF04	**Pantalla “Buscar”**
Versión	**1.0**

Descripción	1. La pantalla de búsqueda debe permitir al usuario filtrar los viajes. 2. El usuario debe poder elegir la fecha en la que se realizará el viaje a buscar. 3. El usuario debe poder elegir el destino del viaje a buscar. 4. La pantalla búsqueda debe incorporar un botón que lleve a una nueva actividad con los viajes filtrados, según los criterios establecidos. 5. La pantalla que contiene los viajes filtrados debe ser idéntica visualmente a la pantalla Viajes. 6. La pantalla con los viajes filtrados no deberá mostrar en ningún caso aquellos viajes en los que el usuario ya está subido o pertenece con el rol “Conductor”. 7. Al pulsar sobre un viaje se deberá mostrar un diálogo con los siguientes botones: a. Un botón que conduzca a Google Maps con la ruta del viaje. b. Un botón que permita al usuario “Subirse al trayecto”.

Tabla 4: RF04 - Pantalla Buscar

RF05	**Pantalla “Nuevo”**
Versión	**1.0**
Descripción	1. La pantalla “nuevo” debe permitir al usuario la creación de un nuevo viaje.

	2. El usuario podrá escoger el origen del viaje. 3. El usuario podrá escoger entre “Campus Ejido”, “Campus Teatinos” o “Ampliación Teatinos” como destino del viaje. 4. El usuario podrá escoger la hora en la que se realizará el viaje. 5. El usuario conductor deberá poder elegir las plazas que tiene disponible para el viaje. 6. Deberá aparecer un botón con el cual el usuario pueda crear el viaje con los datos insertados.

Tabla 5: RF05 - Pantalla Nuevo

RF06	**Pantalla “Chat”**
Versión	**1.0**
Descripción	1. La pantalla chat permitirá al usuario entrar en el chat de cada viaje al que pertenezca. 2. Se deberán mostrar todos los viajes a los que el usuario pertenece. 3. Al pulsar sobre un viaje se debe abrir una nueva actividad “Chat”. 4. En la actividad el usuario deberá ser capaz de enviar mensajes de texto a los pasajeros del viaje sobre el que ha pulsado.

Tabla 6: RF06 - Pantalla Chat

RF07	**Pantalla “Perfil”**
Versión	**1.0**
Descripción	1. Está pantalla permitirá al usuario la visualización de su foto y nombre. 2. La pantalla debe mostrar de alguna forma la foto asociada a Google del usuario cuya sesión está iniciada. 3. Se debe mostrar el nombre del usuario. 4. Aparecerá de alguna manera un botón que permita cerrar la sesión al usuario.

Tabla 7: RF07 - Pantalla Perfil

3.2.2 Requisitos no funcionales

RNF01	**Seguridad**
Versión	**1.0**
Descripción	1. La aplicación solo debe permitir la lectura y escritura de datos a aquellos usuarios que hayan iniciado sesión previamente. 2. Es necesario tener iniciada una sesión para hacer uso de las funcionalidades.

	3. La sesión solo se cerrará en caso de pulsar el botón con este acometido, o que el usuario borre o de sincronice la cuenta del teléfono móvil.

Tabla 8 : RNF01 - Seguridad

RNF02	**Documentos**
Versión	**1.0**
Descripción	1. Se realizará un manual de usuario indicando los pasos a seguir para el uso de la aplicación. 2. Se realizará un manual de instalación con las diferentes maneras de instalar la aplicación.

Tabla 9: RNF02 – Documentos

RNF03	**Interfaz de usuario**
Versión	**1.0**
Descripción	1. La interfaz de usuario final deberá ser lo suficientemente clara como para que el usuario realice todas las operaciones sin ningún problema. Además deberá ser

	adaptadas a los fallos de usabilidad obtenidos en las pruebas.

Tabla 10 : RNF03 - Interfaz de usuario

RNF04	**Mantenimiento**
Versión	**1.0**
Descripción	1. La aplicación hará uso en todo momento de conexión a Internet. 2. La aplicación deberá ser ejecutable en todas las versiones Android iguales o superiores a las soportadas por el SDK 21.

Tabla 11 : RNF04 - Mantenimiento

3.2.3 Casos de Uso

Tras la recogida de los requisitos de la aplicación, es necesario dejar claro los diferentes casos de uso a contemplar. Realizar un buen diagrama de casos de uso permite establecer aquellas actividades o acciones que se van a poder llevar a cabo en nuestro sistema final.

Se va a proceder a detallar cada uno de los casos de uso que se establecieron necesarios, antes del desarrollo del primer prototipo:

CU01	Iniciar Sesión
Dependencia	*No depende de ningún caso de uso.*
Precondición	No existe ninguna sesión iniciada.
Descripción	El usuario deberá poder escoger entre alguna de sus cuentas Google establecidas en el sistema móvil o introducir unos nuevos credenciales de acceso. Una vez hecho esto, se llevará a cabo el inicio de sesión mediante acceso al token de Google Oauth a través de su API.
Postcondición	Se encuentra iniciada la sesión seleccionada o cuyos datos han sido introducidos.

Tabla 12: CU01 - Iniciar Sesión

CU02	Cerrar Sesión
Dependencia	*CU1*
Precondición	Existe sesión iniciada.
Descripción	El usuario en cualquier momento debe poder cerrar su sesión mediante algún tipo de botón.
Postcondición	Se encuentra iniciada la sesión seleccionada o cuyos datos han sido introducidos.

Tabla 13 : CU02 - Cerrar Sesión

CU03	Buscar Viaje
Dependencia	*CU1*
Precondición	Debe existir un viaje que se ajuste a los parámetros de búsqueda y del que el usuario no forme parte.
Descripción	El usuario en cualquier momento puede hacer una búsqueda de viajes según fecha, destino y hora y podrá ver e interactuar con aquellos viajes que respondan a los parámetros establecidos.

Postcondición	Se mostrará los viajes que se ajustan a los parámetros establecidos.

Tabla 14 : CU03 - Buscar Viaje

CU04	Unirse Viaje
Dependencia	CU1 y CU3
Precondición	Debe haberse iniciado una búsqueda de viajes.
Descripción	Una vez el usuario se encuentre ante el listado de viajes que responde a sus criterios de búsqueda, pulsando sobre un viaje, siempre y cuando haya plazas disponibles, el usuario debe ser capaz de unirse al viaje.
Postcondición	Se añadirá al usuario como pasajero del viaje.

Tabla 15 : CU04 - Unirse Viaje

CU05	Publicar Viaje
Dependencia	*CU1*

Precondición	*No existe precondición*
Descripción	El usuario debe poder crear un viaje que se ajuste a los parámetros introducidos.
Postcondición	Se creará en la base de datos un nuevo viaje cuyo conductor será el usuario.

Tabla 16 : CU05 - Publicar Viaje

CU06	Ver Ruta
Dependencia	*CU1, CU3 o CU4 o CU5*
Precondición	Debe existir al menos un viaje sobre el que consultar ruta.
Descripción	Pulsando sobre un viaje al que el usuario pertenece o ha buscado, el usuario debe ser capaz de ver la ruta mediante la aplicación Google Maps.
Postcondición	Se abrirá la aplicación Google Maps con el trayecto del viaje consultado.

Tabla 17 : CU06 - Ver Ruta

CU07	Chatear
Dependencia	*CU1, CU5 o CU4*
Precondición	El usuario debe pertenecer a un viaje no terminado.
Descripción	El usuario debe ser capaz de chatear con el resto de pasajeros de un viaje.
Postcondición	Se creará en la base de datos la conversación mantenida entre pasajeros.

Tabla 18 : CU07 – Chatear

CU08	Eliminar
Dependencia	*CU1, CU5*
Precondición	Ser conductor de un viaje.
Descripción	El usuario debe ser capaz de eliminar un viaje.

Postcondición	Se eliminará de la base de datos el viaje.

Tabla 19 : CU08 – Eliminar

CU09	Salir
Dependencia	*CU1, CU4*
Precondición	Ser pasajero de un viaje.
Descripción	El usuario debe ser capaz de dejar de pertenecer a un viaje.
Postcondición	Se eliminará de los pasajeros de ese viaje al usuario.

Tabla 20 : CU9 - Salir

3.3 Análisis previo al Segundo Prototipo

Tras el primer análisis e implementación de un primer prototipo se llevó a cabo las pruebas con un grupo de usuarios, alumnos de la Universidad de Málaga. Como resultado surgieron nuevos requerimientos y casos de uso que mejorarían la experiencia de usuario y sacarían mayor rendimiento a la funcionalidad de la aplicación.

3.3.1 Requisitos funcionales

Funcionamiento

RF03	**Pantalla “Viajes”**
Versión	**2.0**
Descripción	1. Ya no será necesario mostrar aquellos viajes que han terminado, sólo se mostrarán los viajes a los que pertenezca el usuario cuya sesión esté iniciada y que no hayan terminado. 2. Si el usuario pulsa sobre un viaje cuyo conductor es él mismo, se le permitirá pulsar sobre un botón “Terminar” que simule la terminación del viaje. 3. Si el usuario no pertenece a ningún viaje, se deberá mostrar de forma visual esto.

Tabla 21: RF03 - Pantalla Viajes v2.0

RF04	**Pantalla “Buscar”**
Versión	**2.0**

Descripción	1. El usuario debe poder elegir la hora a la que se realizará el viaje, mostrando todos los viajes que se ajusten a los demás criterios y que comiencen a la misma hora o pasada de la introducida. 2. Al pulsar sobre un viaje se deberá mostrar un diálogo con los siguientes botones, siempre y cuando el viaje tenga plazas disponibles: a. Un botón que conduzca a Google Maps con la ruta del viaje. b. Un botón que permita al usuario “Subirse al trayecto”. 3. Si el resultado de la búsqueda es vacío, se indicará al usuario que no hay ningún viaje que se ajuste a sus criterios. 4. La pantalla que muestra el resultado de la búsqueda deberá de contener algún tipo de icono que permita el cierre de la misma. 5. Una vez se haya producido la unión a un viaje, se deberá cerrar la pantalla resultado de la búsqueda.

Tabla 22: RF04 - Pantalla Buscar v2.0

RF05	**Pantalla “Nuevo”**
Versión	**2.0**

Descripción	1. Al introducir un origen, sólo se debe permitir la selección de calles perteneciente a la provincia de Málaga. 2. El usuario deberá poder introducir el número de plazas disponibles que posee. 3. El usuario podrá introducir una descripción que facilite la identificación de su vehículo así como una nota para los demás pasajeros. 4. Se debe facilitar al usuario la selección de una hora para realizar un viaje de vuelta en caso de que seleccione ésta opción. 5. El usuario podrá escoger la hora en la que se realizará el viaje. 6. El sistema debe ser capaz de calcular la hora de llegada mediante el uso de una API.

Tabla 23: RF05 - Pantalla Nuevo v2.0

RF06	**Pantalla “Chat”**
Versión	**2.0**
Descripción	1. Se distinguirá de alguna manera la identidad del usuario de cada mensaje dentro de un chat.

Tabla 24 : RF06 - Pantalla Chat v2.0

RF07	**Pantalla “Perfil”**
Versión	**2.0**
Descripción	1. Se mostrarán un histórico de todos los viajes realizados por el usuario cuya sesión esté iniciada y que hayan terminado. 2. Se permitirá de alguna manera la votación de aquellos viajes en los que el usuario haya sido pasajero y han terminado.

Tabla 25 : RF07 - Pantalla Perfil

3.3.2 Casos de Uso

Posterior a las pruebas y recogidas de requisitos es inevitable que surjan nuevos casos de uso que haya que implementar. Es cierto que en comparación con los requisitos que surgen tras la realización de un prototipo, el número de casos de uso que surgen es menor, pero su recogida es necesario para detallar perfectamente el sistema final.

CU10	Terminar viaje
Dependencia	*CU1, CU5*

Precondición	Ser conductor de un viaje no terminado.
Descripción	El usuario debe ser capaz de terminar un viaje.
Postcondición	El estado del viaje pasará a Terminado.

Tabla 26 : CU10 -Terminar Viaje

CU11	Puntuar
Dependencia	*CU1, CU4*
Precondición	Ser pasajero de un viaje terminado.
Descripción	El usuario debe ser capaz de puntuar al conductor de un viaje terminado.
Postcondición	Se añadirá la puntuación al conductor.

Tabla 27 : CU11: Puntuar

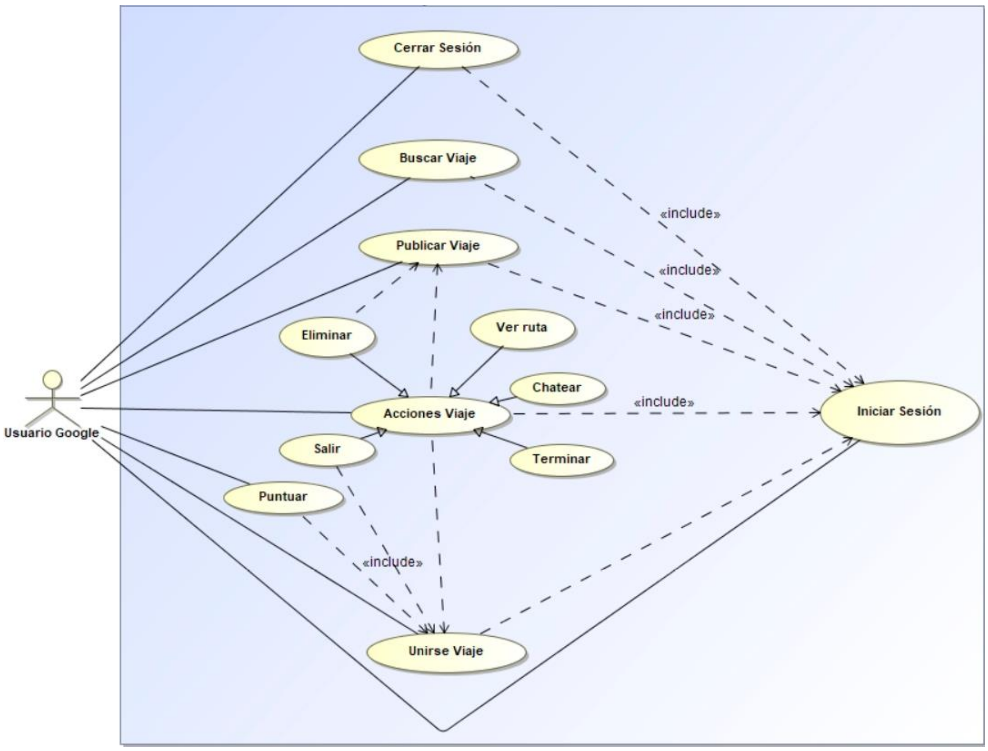

Ilustración 3: diagrama final de casos de usos

3.4 Análisis previo al Sistema Final

Una vez implementado nuevamente los requerimientos adicionales y contemplados los nuevos casos de uso, se lleva a cabo unas nuevas pruebas sobre el prototipo realizado. Como resultado de estas pruebas surgen una serie de nuevos requisitos:

3.4.1 Requisitos Funcionales

Funcionamiento

RF03	**Pantalla "Viajes"**
Versión	**3.0**
Descripción	1. Cada viaje deberá de mostrar también el número de "me gusta" y "no me gusta" asociados al conductor del viaje así como la fecha a realizar el viaje.

Tabla 28 : RF03 - Pantalla Viajes v3.0

RF04	**Pantalla "Buscar"**
Versión	**3.0**

Descripción	1. Una vez pulsado en “Unirse al Viaje”, si el éste contiene trayecto de vuelta, preguntar al usuario si quieres realizarla también.

Tabla 29 : RF04 - Pantalla Buscar v3.0

RF06	**Pantalla “Chat”**
Versión	**3.0**
Descripción	1. Se notificará al usuario cuando hablan por un chat al que pertenece.

Tabla 30 : RF06 - Pantalla Chat v3.0

RF08	**General**
Versión	**1.0**
Descripción	1. La aplicación no deberá permitir la rotación de pantalla. 2. Desarrollar una función en el lado del servidor que termine los viajes de cada día.

Tabla 31 : RF08 - General

Capítulo 4
Diseño

4.1 Fase de diseño

En este capítulo se explicará las decisiones tomadas para la constitución del diseño del proyecto, tanto a nivel arquitectónico como a nivel de interfaz. El diseño permite explicar de forma general como funcionará el sistema de forma que satisfaga los casos de usos y requisitos a tener en cuenta.

A pesar de que el proyecto ha sufrido cambios notables conforme se desarrollaban nuevos prototipos, el diseño no ha sufrido muchas variaciones desde su planteamiento inicial. Las tecnologías que desde un primer momento se plantearon para la implementación del cliente y del servidor, han sido lo suficientemente versátiles como para abordar los nuevos requisitos, casos de usos y problemas que han surgido a los largo del desarrollo.

4.2 Arquitectura aplicación

La arquitectura que se ha escogido para el desarrollo del proyecto ha sido la conocida como cliente-servidor. Las aplicaciones que usan este tipo de arquitecturas hacen uso de un cliente que ejecuta consultas a un servidor, esperando una respuesta.

En el caso de este proyecto, el cliente es la misma aplicación Android, que permitirá mediante una interfaz de usuario hacer uso del código desarrollado para poder ejecutar las diferentes acciones, con el fin de esperar una respuesta del servidor.

Por otra parte, en este tipo de arquitecturas es importante tener un servidor donde ejecutaremos el resto de código y que permita comunicarse con el cliente para darle respuesta, así como hacer labores de persistencia. En el caso de este proyecto, el lado del servidor se encuentra totalmente instaurado en la plataforma Firebase, permitiendo la comunicación con el cliente y la realización de las tareas de persistencia, además de muchas otras, como se muestra en la Ilustración 4.

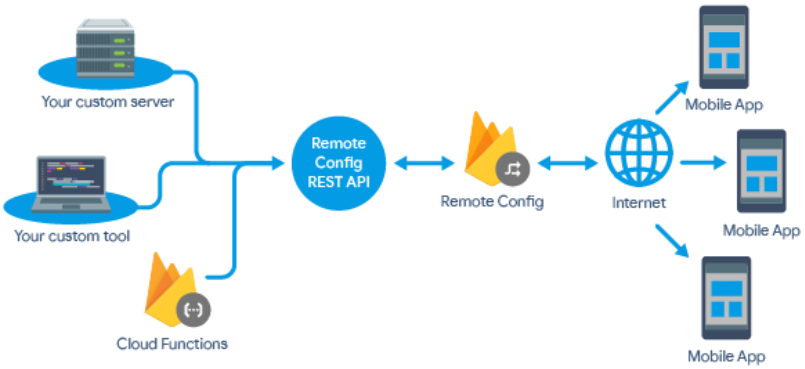

Ilustración 4 : diagrama cliente-servidor con Firebase

Fuente: https://firebase.google.com/docs

4.2.1 Diseño del servidor

La versatilidad y el gran número de posibilidades que ofrece Firebase, así como su reducido coste para proyectos a pequeña escala, suponen que sea una tecnología perfecta para implementar en éste proyecto.

Se va a proceder a exponer los diferentes servicios de los que se beneficia el proyecto, en el lado del servidor:

Autenticación

Bajo los estándares OAuth 2.0 y OpenID Connect, Firebase Authentication [10] pone al servicio del usuario una API completa para poder implementar la autenticación de usuarios, con proveedores como Google, Facebook o Twitter.

Desde el punto de vista del servidor, Firebase Authentication permite hacer un control en todo momento de los usuarios que se encuentran activos en el sistema, facilitando las tareas de mantenimiento.

Otro punto fuerte de la API es la facilidad para activar nuevos proveedores por parte del desarrollador, es decir, si se decide permitir el inicio de sesión con una cuenta de Facebook, basta con activarlo desde la consola y añadir el código propuesto en el lado del cliente. También es importante el hecho de que los credenciales de acceso no

queden en responsabilidad del desarrollador, pues en ningún momento se guardan usuarios ni contraseñas.

A la hora de restringir el acceso a la escritura y lectura de datos, de nuevo desde la consola se permite fácilmente instaurar nuevas reglas para ello.

Ésta API también supone un punto crucial en el desarrollo de UMACAR desde el lado del cliente, pues permite que el usuario que vaya hacer uso de la aplicación pueda iniciar sesión en ella directamente con su cuenta de Google, sin tener que registrarse, manteniendo su sesión iniciada y permitiendo el acceso a los datos que ofrecen los distintos proveedores hasta un cierre de sesión.

Base de datos

A la hora de alojar los datos de la aplicación en la nube Firebase propone dos tipos de esquemas para utilizar:

- **Firebase Realtime Database** [11] es una base de datos NoSQL que permite la sincronización simultánea en tiempo real de aquellos clientes que se encuentren haciendo peticiones en cualquier momento mediante el almacenamiento de datos en formato JSON.
- **Cloud Firestore** [12] surge de la necesidad de establecer un modelo más intuitivo a los datos alojados en Realtime Database. Orientado a la telefonía móvil, permite realizar consultas más rápidas y complejas.

En este proyecto se ha hecho uso de ambos tipos de esquemas para implementar la persistencia. La simpleza y rapidez de trabajo que ofrece Realtime Database se ajusta perfectamente a los requisitos que supone la implementación de un chat entre usuarios en cualquier aplicación, es por ello que las conversaciones entre los diferentes pasajeros de un viaje se alojarán en este servicio, dejando el resto de la persistencia de mano de Cloud Firestore, que permite mayor estructuración de los datos, así como lo realización de consultas más elaboradas por medio de índices.

A continuación se hará un desglose de las diferentes entidades que se almacenan en ambas bases de datos, es importante destacar que ambos esquemas son no relacionales, por lo que es necesario hacer uso de identificadores para las relaciones:

Realtime Database

- **Chat:** por cada viaje creado se genera un id aleatorio y se crea una nueva entidad chat en la base de datos que alojará cada uno de los mensajes que se escriban en él.

  Atributos:

  - **Id**: cadena de texto que contiene el id de la sala chat para su referencia.

- **Mensaje:** representa cada uno de los mensajes que se escriben en un chat.

  Atributos:

  - **autorUID:** UID de Google del usuario que escribe el mensaje.
  - **fotoPerfil:** cadena de texto con la dirección web de la foto de perfil del usuario que escribe el mensaje.
  - **mensaje**: cadena de texto con el contenido del mensaje.
  - **nombre**: nombre del autor del mensaje.

Cloud Firestore:

- **Usuario:** por cada usuario diferente que inicia sesión en algún momento se crea una nueva entidad Usuario con los siguientes atributos:

  Atributos:

  - **nombre**: cadena de texto con el nombre del usuario en su cuenta Google.
  - **usuario:** cadena de texto con el UID Google asociado al usuario.
  - **token:** token de sesión asociado al usuario.
  - **likes:** número de “me gusta” que ha recibido el usuario como conductor.
  - **unlikes**: número de “no me gusta” que ha recibido el usuario como conductor.
  - **viajes**: matriz de cadenas de textos que representan cada uno de los viajes que ha realizado el usuario.

- **Viajes**: por cada viaje que se publica en la aplicación se crea una nueva entidad con los siguientes atributos:

  Atributos

  - **creador:** contiene el nombre del creador del viaje.
  - **creadorId:** cadena de texto con el UID Google del creador.
  - **year:** cadena de texto con el año en el que se realizará el viaje.
  - **month:** cadena de texto con el mes en el que se realizará el viaje.
  - **day:** cadena de texto con el día en el que se realizará el viaje.
  - **coche:** cadena de texto que permite facilitar la identificación del coche.
  - **anotación:** cadena de texto que permite al conductor recordar algo importante a los pasajeros.
  - **destino:** nombre del destino del viaje.
  - **destinoId:** id asociada a Google Places del destino.
  - **origen:** nombre del origen del viaje.
  - **origenId:** id asociada a Google Places del origen.
  - **Chat:** cadena de texto que coincide con el id del chat creado para el viaje.
  - **duración:** duración calculada mediante la API Google Places del viaje.
  - **fechaCreación:** fecha creación del viaje.
  - **foto:** url de la imagen de perfil del conductor.
  - **hayVuelta:** valor boolean que estará a verdadero si hay viaje de vuelta y a falso en caso contrario.
  - **idVuelta:** contiene el id del destino de vuelta, coincidirá con el idOrigen.
  - **horaLlegada:** hora en la que se producirá la llegada estimada.
  - **horaSalida:** hora en la que se producirá la salida según la hora introducida por el usuario.
  - **minutosLlegada:** minutos en los que se producirá la llegada estimada.
  - **minutosSalida:** minutos en los que se producirá la salida según la hora introducida por el usuario.
  - **id:** cadena de texto única para cada viaje.
  - **plazas:** plazas libres en el viaje.
  - **terminado:** valor booleano que indica si un viaje ha concluido.

- **pasajeros:** matriz de cadena de texto con los identificadores de cada pasajero.
- **votaciones:** matriz de cadenas de texto con los identificadores de cada pasajero que no ha votado el viaje.
- **likes**: matriz de cadenas de texto con los identificadores de cada pasajero que ha dado me gusta al viaje.
- **unLikes:** matriz de cadenas de texto con los identificadores de cada pasajero que ha dado no me gusta al viaje.

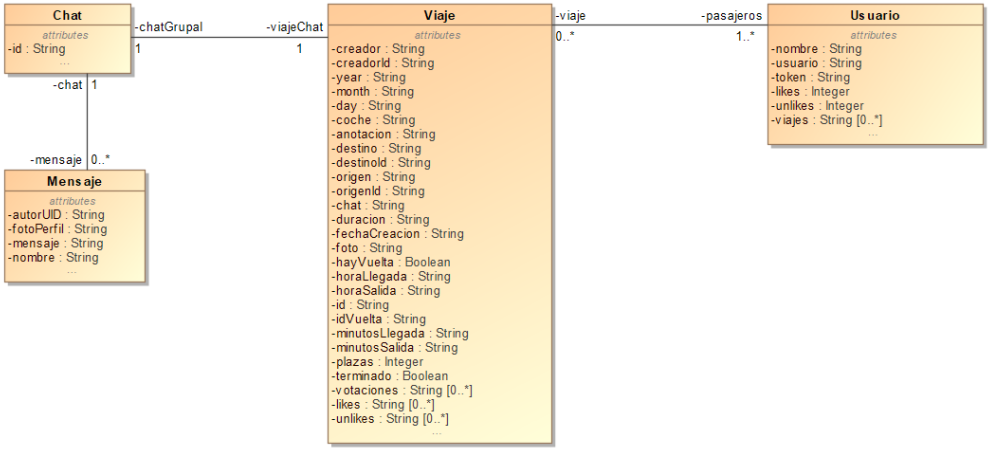

Ilustración 5: diagrama de clases

Fuente: elaboración propia

Firebase Crashlytics

Para el apartado de pruebas y mantenimiento, Firebase ofrece una herramienta con la que podemos hacer un seguimiento claro de cualquier tipo de error que pueda ocurrir en nuestra aplicación.

A través de Firebase Crashlytics [13] podremos ser informados en tiempo real de cualquier fallo que ocurra a cualquier usuario que esté utilizando la aplicación y consultar las estadísticas inmediatamente a través de la consola. Mediante una interfaz clara, podremos consultar un registro en el que se mostrará el dispositivo en el que ha ocurrido el fallo, la línea de código y más información relevante para la reproducción y posterior corrección del fallo. Otra característica que ofrece esta herramienta es la posibilidad de simular fallos en nuestra aplicación.

Firebase Cloud Messaging

Firebase pone a disposición del usuario una plataforma segura para el envío de notificaciones multiplataforma entre usuarios de un sistema.

Mediante la subscripción automáticas a “temas”, Firebase Cloud Messaging [14] permite a los usuarios de UMACAR poder recibir notificaciones cuando se envíe un mensaje por algún chat.

Firebase Functions

Para desarrollar la función en el lado del servidor que termine los viajes de forma automática se ha utilizado éste servicio.

Mediante Firebase Fuctions [15] se ha desarrollado un documento Nodejs que contiene la función a desplegar en el servidor. Ésta función se ejecuta todos los días a las 00:00 y obtiene todos los viajes del día anterior y pone el valor booleano “terminado”, alojado en la base de datos, a false.

APIs

A la hora de abordar el diseño del proyecto, surgen varias dudas en cuento a la necesidad de implementar un sistema que permita calcular el tiempo que tardará en realizarse un viaje, destinado al cálculo de la hora de llegada. El comportamiento esperado es que, cuando un usuario introduzca la hora de salida, un origen y un destino, automáticamente se calcule la hora de llegada para evitar tener que meterlo manualmente y poder cometer errores. Para ello se llevó a cabo un estudio de las diferentes APIs que ofrecen estos servicios, resultando elegida Directions API, nuevamente ofrecida por Google a través de Google Cloud Platform.

Directions API permite obtener información sobre un trayecto a través de una petición HTTP. Cuando el usuario elige un origen y un destino, se genera una url a la que se hace la petición, tomando como respuesta un JSON en el que figura, aparte de más información, el tiempo estimado que se tardará en recorrer ese trayecto en ese momento.

Diseño Seguridad

El hecho de elegir una plataforma como Firebase para desarrollar el lado del servidor supone una ventaja más a nivel de seguridad, pues no hay que preocuparse de establecer protocolos a la hora de realizar las conexiones HTTP con el propio servidor. Además, el hecho de no almacenar datos críticos del usuario como puede ser usuario y contraseña, supone que no haya que establecer un algoritmo de cifrado de la información, antes de ser almacenada.

Si se observa los métodos de seguridad que brinda la plataforma [16], vemos como se ajustan perfectamente al Reglamento general de protección de datos de la Unión Europea y contemplan los procesos de evaluación ISO 27001 y SOC1, SOC2 y SOC 3, dejando a buen recaudo cualquier información que podamos almacenar.

4.2.2 Diseño del cliente

Aunque es cierto que el lado del servidor es importante para el funcionamiento del producto, el lado del cliente es lo que al final estará en manos del usuario final. Por tanto, una implementación buena de los requisitos recogidos que satisfagan todos los casos de uso previsto, así como el diseño de una interfaz estética y funcional es crucial para que la aplicación se adapte a las necesidades del cliente y se posicione como una alternativa real para la solución del problema por la que es desarrollada.

APIs

Al igual que pasa en el servidor, en el lado del cliente surgen nuevas dudas sobre cómo abordar los diferentes requisitos que necesitan de un servicio externo para ser implementados. En este caso, surgen tres principales problemas:

1. Búsqueda de un servicio que permita mantener una sesión iniciada.
2. Un servicio que permita la incrustación de mapas en nuestra aplicación.

3. Un servicio que permita, de alguna manera, la elección de calles para poder establecer un origen.

1. El hecho de decantarse, en el lado del servidor, por Firebase Authentication hace que sea aún más fácil la implementación de un método de autenticación en el lado del cliente. Junto con Firebase Authenticaton se ofrece la posibilidad de implementar FirebaseUI Auth, un SDK que proporciona una solución de autenticación directa para el manejo y mantenimiento de flujos de autenticación de usuario a través de los diferentes proveedores.

Con FirebaseUI Auth se puede implementar de forma intuitiva una interfaz de usuario que permita el inicio y cierre de sesión por parte del usuario, así como implementar a través de las librerías que ofrece, diferentes métodos para el manejo de la sesión y por lo tanto de los datos del usuario cuya sesión está iniciada.

FirebaseUI Auth hace uso de las siguientes APIs controladas a través de la consola de Google Cloud Platform:

- **Google Identity Toolkit API**: permite usar los estándares de los diferentes proveedores para la identificación de usuarios.
- **Token Service API**: permite acceder al token de sesión para realizar peticiones al servidor.

2. A la hora de resolver la problemática de implementar un mapa con el que podamos interactuar en la misma aplicación, se decidió hacer uso de Maps SDK for Android, un conjunto de herramientas que permite acceder directamente a la API de Google Maps.

Maps SDK ofrece la posibilidad de incrustar en la aplicación mapas de Google, pudiendo interactuar no solo a través de gestos, sino también de métodos gracias a las librerías que ofrece.

3. Para que se lleve a cabo la interacción con el mapa, es necesario primero establecer un origen a la hora de crear un viaje. Para ello es necesario de una potente

herramienta que permita elegir entre las distintas calles de un territorio acotado. Aunque existen varias soluciones ante ésta problemática, la más cómoda y que mejor se ajusta al proyecto es Places API. Con Places API se permite la instauración de una actividad con una interfaz cómoda y limpia para la selección de calles. Además, una vez seleccionado el origen, nos permite interactuar con él a través de una librería, permitiéndonos el acceso a sus coordenadas, id…

4.3 Diseño Interfaz

El diseño de la interfaz supone uno de los factores más importantes para que una aplicación tenga éxito. Hoy en día es frecuente ver gran variedad de herramientas que proponen dar solución a un mismo problema, sin embargo unas triunfan y otras no. ¿Cómo es esto posible? Probablemente, en la mayoría de casos, lo que hace que un usuario se decante por una aplicación es la experiencia en cuanto a nivel de usabilidad e interfaz.

Una aplicación que no sepa adaptar su diseño a las necesidades que pueda tener el usuario al que va destinada, es susceptible de acabar desinstalada del terminal. Normalmente, cuando hablamos del diseño pensamos sólo en la interfaz de usuario, que es tremendamente importante, pues una mala elección de los colores o disposición de los elementos puede hacer incomodar al usuario y que por lo tanto deje de usar la aplicación al instante, pero va más allá. El diseño de la interfaz del usuario también debe ser valorado desde el punto de vista de la usabilidad, es decir, cómo llevamos a cabo la implementación de la interfaz de forma que la experiencia sea lo más agradable posible, ya sea en tiempos de carga, transiciones etc.

A continuación se mostrarán las decisiones generales que se han tomado en cuanto a diseño y que por tanto deben tener en común todo elemento dentro de la aplicación.

Gama cromática

A la hora de elegir una gama cromática que instaurar en los componentes de la aplicación, se ha optado por partir de un azul grisáceo oscuro hasta llegar a un blanco,

pasando por el gris y el turquesa para diferentes detalles. Los valores en hexadecimal de los diferentes colores son:

- Color Primario (Azul Grisáceo): #102027
- Color Secundario (Gris Oscuro): #484848
- Color Secundario Claro (Gris Claro): #bdbdbd
- Color Detalles (Turquesa): #baffff
- Blanco: #f5f5f5

Balances Blanco y Negro

Como podemos observar en la elección de los colores principales, ningún color es exactamente ni negro, que equivaldría al valor #00000, ni blanco, cuyo valor en hexadecimal es #FFFFF. En el caso del negro, esta decisión se debe a que el uso de colores grises saturados se asemeja más al mundo real y añade riqueza visual a los diseños, al ser combinados con elementos más oscuros o más claros. En el caso del blanco, si bien es cierto que el utilizado se asemeja más a un blanco total, la decisión de optar por valores un poco más crudos es sencilla, normalmente las pantallas de los diferentes móviles muestran los colores un poco más brillantes o más oscuros y esto, sumado al uso del brillo por parte de los usuarios para ver mejor sus pantallas, puede hacer que un elemento blanco destaque excesivamente sobre los demás, incomodando la experiencia del usuario. Por ello es mejor optar por valores un poco más oscuros y prevenir blancos brillantes.

Espacios en blanco

Junto con el anterior apartado, potenciar los espacios en blancos facilita el desarrollo de una interfaz más limpia y pulida. Es por ello que a lo largo de todas las actividades de la aplicación, se ha potenciado el uso de márgenes en los diferentes elementos, permitiendo una mayor visualización del blanco, que normalmente es sinónimo de tranquilidad en cuanto a experiencia de usuario.

Diseño semiplano

Cuando se habla de un diseño semiplano se hace referencia a la combinación de elementos de interfaz que gozan de un relieve sobre el plano, junto con elementos que carecen de este relieve. Ésta decisión viene influenciada por seguir los estándares

de interfaces recomendados en Android, Material Design, que promueve un diseño limpio y sencillo, junto con algunas sombras que indican zonas de interacción.

La decisión de optar por este diseño es la intención de trasmitir un diseño sutil y cuidado al usuario, que haga que su experiencia con la aplicación sea más agradable.

Tipografía

La elección de una tipografía, en lugar de usar las que viene por defecto en el sistema, da la sensación de que hay un trabajo y un cuidado detrás de la interfaz. En este caso, se ha optado por implementar una tipografía redonda, que sea perfectamente legible, pero que se aleje de aquellos estilos más rectos y serios. El uso de una letra redondeada hace que la interfaz sea más amigable he incita al uso por parte del usuario. Además, la combinación de diferentes estilos para una misma tipografía, ya sea negrita, light o cursiva enfatiza sobre los datos más importantes, diferenciándolos de aquellos que quizás no son tan importantes pero es necesario que estén ahí.

Iconos

El uso de iconos permite identificar de forma rápida los elementos contiguos, ahorrando espacio que iría destinado a texto. En la mayoría de actividades se hace uso de estos iconos, pues al igual que la tipografía, hace más amena la experiencia de usuario y más familiar. También es importante la combinación de iconos y texto para enfatizar a lo que se hace referencia y a modo de complemento. No obstante el uso de iconos debe ser limitado, pues el abuso de éstos puede causar rechazo por parto del usuario.

Actividades oscuras

En contraposición con el apartado en el que se habla de la importancia de dejar huecos en blanco surge éste apartado. Es importante que a la hora de utilizar espacios en blancos, lo equilibremos con otras actividades más oscuras que permitan una balanza entre colores claros y oscuros. El abuso de color blanco puede hacer creer al usuario que hay falta de interés por parte del desarrollador en crear una interfaz más cuidada y compleja. Por ello, a lo largo de la aplicación se suceden elementos o actividades más oscuras que permiten un perfecto balance entre colores.

Enfatizar Pulsaciones

A lo largo del ciclo de vida de la aplicación se hace uso del color turquesa sobre un color más oscuro para detallar dónde hemos pulsado. Esto es muy importante porque a veces el usuario no sabe si ha realizado la pulsación o no, e indicándolo con la combinación de color es un método muy efectivo y agradable a la vista.

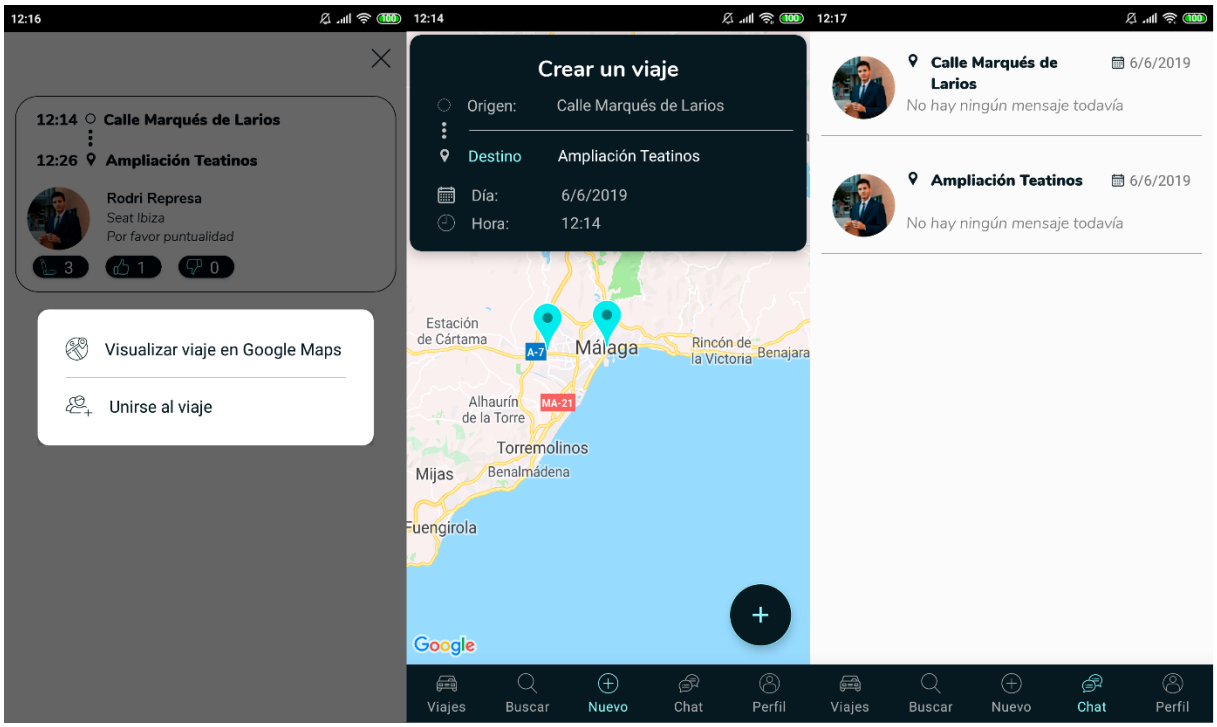

Ilustración 6 : diferentes interfaces de la aplicación

Fuente: elaboración propia

Capítulo 5
Implementación

5.1 Fase de implementación

Una vez ha queda claro tanto la parte de análisis como la de diseño, se ha de abordar la fase de implementación. En este capítulo se desarrollarán conceptos generales sobre la programación nativa en Android, para más tarde detallar cada una de las pantallas de las que se componen la aplicación. También se hará una explicación de la estructura interna para facilitar su entendimiento.

Si bien es cierto que el sistema final ha sido el resultado de numerosas fases de implementación, una por cada prototipo, en este capítulo se explicará la implementación del sistema final, que es el que reúne todos los requisitos y casos de usos contemplados en el apartado anterior.

Destacar también que se explicará la implementación del cliente, pues el servidor como se ha mencionado se alojará en Firebase.

El cliente ha sido desarrollado en el entorno de desarrollo Android Studio, utilizando el lenguaje de programación Java para la lógica y XML para la interfaz. También ha sido necesario la implementación de diferentes librerías, que serán explicadas más adelante, para la consecución de diferentes tareas.

5.2 Componentes Android

A continuación se explicarán algunos componentes típicos de Android y usados en este proyecto para un mejor entendimiento de las soluciones implementadas:

Activity

Un activity [17], también llamado actividad, es el componente básico de toda aplicación Android, equivaldría a lo que comúnmente llamamos pantalla. Una actividad se conforma de una parte lógica ejecutada en un fichero Java que permite el procesamiento de datos, conexión con el servidor, interaccionar con una Api.... A su vez, una actividad también permite la interacción directa con el usuario mediante lo que se denomina layout.

Layout

Fichero XML en el que se encuentra el código que representa la interfaz de una actividad. Para que la experiencia de usuario sea más satisfactoria, Android Studio provee de diferentes componentes propias del sistema Android para conformar nuestro layout [18] o vista. Además, una vista puede incorporar nuevos componentes a través de librerías externas.

Intent

Para poder comunicarnos entre diferentes actividades, Android SDK provee una clase denominada Intent [19] que permite, a través de un objeto, comunicarse con otra actividad y así poder transmitir información entre ambas.

Fragment

Un Fragment [20] hace referencia a una porción de una actividad que tiene su propio ciclo de vida. Así, por ejemplo, podríamos tener en una misma actividad, un fragmento que ocupe la mitad superior de la pantalla y que esté mostrando una interfaz totalmente distinta a la mitad inferior de la pantalla. Los fragmentos surgen con la necesidad de crear interfaces de usuario más dinámicas y flexibles con la aparición de las tabletas. Sin embargo, hoy en día su uso en desarrollo móvil es muy común, pues ofrece mayor eficiencia que abrir diferentes actividades continuamente, con el riesgo de no cumplir sus ciclos de vida.

RecyclerView

Componente propio de la librería Android que permite mostrar grandes listas de datos con un consumo eficiente de memoria. La decisión de utilizar recyclerview [21] para mostrar los datos se debe a que sólo consumen memoria aquellos datos que se encuentran en pantalla, evitando así un consumo excesivo de memoria RAM y por consecuente, propiciando una mala experiencia de usuario. Además, permite utilizarse junto con las librerías de Firebase, que proporcionando métodos más cómodos para la actualización en tiempo real de los datos.

Los recyclerview hacen uso de un adaptador para mostrar la interfaz de usuario y carga la lógica.

Adaptador

Por cada uno de los datos devueltos en las consultas de Firebase y mostrados a través del recyclerview, el adaptador se encarga de escoger la interfaz de usuario a mostrar y controlar la lógica de éstos, por ejemplo al ser pulsados. Es el adaptador [22] el que se encarga de mostrar los diferentes diálogos al pulsar sobre los viajes, dependiendo de si se es conductor o no.

Diálogo

Cuando hacemos referencia a un diálogo [23] hablamos de una pequeña venta que se muestra en primer plano ofreciendo información al usuario y esperando a la toma de decisiones. Sin embargo, estos diálogos también pueden ser personalizados a gusto del desarrollador para mostrar todo tipo de interfaces.

5.3 Estructura del proyecto

En este apartado se hará una breve explicación del contenido del proyecto Android para un mejor entendimiento de la estructura y se detallará brevemente la funcionalidad de cada carpeta.

- Carpeta **app/manifest/** contiene el manifiesto del proyecto. Éste documento de extensión XML contendrá cada una de las actividades que componen la aplicación, así como los permisos necesarios y características de comportamiento general.
- Carpeta **app/java/** contiene tres paquetes de proyectos, uno dedicado al código y otros dos a distintos tipos de test. Dado que las prueban se han realizado con usuarios reales y a través del servidor, se va a hacer incapié en el primer paquete.
- Dentro del primer paquete **es.represa.umacar** encontramos toda la lógica del cliente, el paquete está estructurado en diferentes carpetas según la función de cada clase:
    - **Adapter**: contiene los adaptadores de cada uno de los recyclerview utilizados.
    - **Chat**: contiene la única actividad necesaria para el desarrollo del chat.

- **Fragment**: contienen todos los fragmentos que encontramos en la aplicación y que conforman la actividad principal.
- **Holder**: por cada uno de los adaptadores existe un holder que se encarga de captar cada componente de la interfaz para su modificación.
- **Model**: contiene cada uno de los modelos de persistencia necesarios para el desarrollo del proyecto. Existe un modelo por cada clase en la base de datos: viaje, usuario, chat y mensaje.
- **Uti**l: contiene diferentes clases necesarias para realizar diferentes tareas:
    - **GuiUtil**: lleva a cabo la transición entre fragmentos.
    - **PreferenceManager:** permite saber si es la primera vez que el usuario ha abierto la aplicación.
    - **QueryUtils**: tarea asíncrona para realizar la conexión HTTP.
- En la raíz del paquete se encuentra el resto de actividades que conforman el proyecto y que serán detalladas a continuación.

5.4 Pantallas

Ahora que ya se ha expuesto los diferentes conceptos necesarios para abordar los siguientes apartados, se va a proceder a exponer y explicar la implementación de cada una de las pantallas que conforma la aplicación.

SplashScreen

- **Tipo**: Actividad.
- **Funcionalidad**: una vez abierta la aplicación, se muestra esta actividad como transición entre la pantalla principal del móvil del usuario y la pantalla principal de la aplicación. Dota de mayor seriedad e imagen y permite agilizar los tiempos de carga en segundo plano para aplicaciones pesadas.
- **Implementación**: para su implementación se ha optado por el uso de una actividad, ya que la transición es más vistosa. Se compone de un único método que carga la interfaz de usuario correspondiente y hace una espera de 5 segundos, tras la cual ejecuta mediante un Intent la actividad principal.

LoginActivity

- **Tipo**: Actividad.
- **Funcionalidad**: proporcionar al usuario una manera efectiva de iniciar sesión mediante un botón de Google.
- **Implementación**: para llevar a cabo su implementación se ha tenido en cuenta los dos posibles casos, que el usuario tenga la sesión iniciada o que por el contrario no haya ninguna sesión iniciada. Al crearse la actividad se comprobará si hay una sesión iniciada, en caso verdadero se lanzará el tutorial.

  En caso de no haber sesión iniciada, la actividad no ejecutará nada más, pues se quedará esperando a que el usuario interactúe mediante el botón de iniciar sesión. Si el usuario pulsa el botón, se llamará proveedor correspondiente, en este caso Google, para seleccionar una de las cuentas o introducir una nueva con las que iniciar sesión.

  Si todo va bien se lanzará el tutorial, pero en caso contrario se devolverá a ésta misma actividad mostrando un mensaje de error.

TutorialActivity

- **Tipo**: Actividad.
- **Funcionalidad**: mostrar un tutorial que deje claro qué ofrece la aplicación.
- **Implementación**: para implementar el tutorial se ha hecho uso del componente Android denominado viewpager. Éste componente permite mostrar en una misma actividad diferentes vistas interactuando con la pantalla, en este caso deslizando a los lados.

  Una vez se inicia la actividad lo primero que se comprueba es si el usuario ya ha pasado alguna vez por el tutorial desde que instaló la aplicación. En caso afirmativo, como no queremos mostrar el tutorial cada vez que se abra la aplicación, se cerrará la actividad y se llamará a la siguiente. En caso de que sea la primera vez, se mostrará el viewpager, que irá escogiendo las diferentes vistas según se mueva de izquierda a derecha.

La actividad está en todo momento esperando a que el usuario pulse alguno de los botones:

- En caso de que el usuario presione el botón SIGUIENTE, el viewPager pasará a la siguiente pantalla.
- Si por el contrario pulsa directamente en SALTAR, se ejecutará un método que cierra la actividad actual y abre la siguiente.

MainActivity

- **Tipo:** DetailActivity
- **Funcionalidad**: pantalla principal que permitirá moverse entre las diferentes categorías.
- **Implementación**: la actividad principal posiblemente sea la pantalla más compleja e importante. Antes de explicar la implementación de esta clase es importante entender por qué extiende de la clase DetailActivity.

  DetailActivity es una clase abstracta que representa una actividad cuyo único funcionamiento es mostrar una interfaz conformada por la barra de navegación inferior, que contendrá un icono y texto descriptivo de cada una de las secciones. El resto de la interfaz se conforma por un espacio en blanco. que será sustituido por los diferentes fragmentos de las cinco secciones una vez se pulse en el icono correspondiente. Por ejemplo si pulsamos en el botón Buscar, se espera que el espacio en blanco sea sustituido por el fragmento que representa la sección Buscar, quedando la barra inferior en la misma posición. Para lograr este comportamiento basta con ejecutar un método implementado a través de la clase GuiUtil.

  El fragmento por defecto que cargará esta actividad será el asociado a los viajes, denominado CarFragment.

  También, al ser la primera actividad que se ejecuta una vez esté la sesión iniciada, se hace una comprobación de si el usuario existe en la base de datos, si no es así lo crea. Destacar que está actividad será la encargada de reaccionar ante las pulsaciones del usuario en la barra inferior, y por tanto está

constantemente a la espera de que el usuario pulse en una de las secciones para hacer la transición, remplazando el espacio en blanco por el fragmento de la sección pulsada.

CarFragment

- **Tipo**: Fragmento.
- **Funcionalidad:** mostrar cada uno de los viajes no terminados a los que el usuario, cuya sesión está iniciada, pertenece como conductor o pasajero.
- **Implementación**: para desarrollar este fragmento ha sido necesario contemplar las dos únicas posibilidades: que el usuario tenga viajes que mostrar o que no los tenga. Para saber esto, cuando se inicia el fragmento, se hace una consulta a la base de datos alojada en Firebase que devuelve el conjunto de viajes en los que el usuario aparece como conductor o pasajero. En el caso de que éste conjunto sea vacío, se procederá a mostrar un icono y mensaje que haga saber al usuario que carece de viajes. En caso contrario se procederá a mostrar cada uno de los viajes pertenecientes al conjunto devuelto por la consulta. La manera en la que se muestran es sencilla y eficaz, pues contiene toda la información necesaria para la identificación del viaje.

  Para mostrar cada uno de los viajes se ha utilizado el componente recyclerview implementado con Firebase, que se encuentra permanentemente escuchando cambios en los viajes. Por ejemplo si un viaje es eliminado, ya sea por nosotros mismos o por otro conductor, desaparecerá al instante de la pantalla.

  El recyclerview hace uso de un adaptador que se encarga de mostrar la interfaz y estar a la espera de que el usuario pulse sobre alguno de los viajes. En caso de que se produzca esto, se muestra un diálogo en el que nuevamente el contenido dependerá de si el usuario es conductor o pasajero. Para ello basta con comparar mediante los identificadores de usuario y a través de la librería Firebase si el usuario actual es el usuario que está guardado como conductor o no.

Si el identificador del usuario es igual al del conductor, se mostrará un botón que permita visualizar el trayecto mediante la aplicación Google Maps, para ello basta con lanzar un intent, pasando como parámetros el origen y el destino. Se mostrará otro botón que permita al usuario eliminar el viaje, desapareciendo de la base de datos y por lo tanto no será accesible por ningún pasajero. Por último habrá un botón que permita terminar el viaje, que hace pasar el viaje al estado terminado y permitiendo su votación a los pasajeros. En el caso de que los identificadores no coincidan, habrá un botón con la misma funcionalidad del Google Maps y otro que permita al usuario salirse del viaje, por si ya no le interesa ir al destino. En este último caso lo único que hay que hacer es eliminar al usuario de la matriz pasajeros alojada en la base de datos, mediante una consulta.

SearchFragment

- **Tipo**: Fragmento
- **Funcionalidad**: permitir realizar una búsqueda parametrizada de viajes.
- **Implementación**: al ejecutarse el fragmento carga una interfaz que funciona a modo de formulario. El sistema esperará a que el usuario pulse en las diferentes acciones para responder.

  Si el usuario pulsa en la fecha, se mostrará un calendario limitado a una semana para elegir cuando quiere viajar.

  Si pulsa sobre el Destino se mostrará un diálogo que permita elegir cualquiera de los destinos ofrecidos.

  Si pulsa sobre la hora se mostrará un diálogo personalizado, que permite escoger a qué hora quiere realizar el viaje.

  Si pulsa sobre la flecha puede ocurrir dos cosas, si el usuario no ha rellenado todos los pasos anteriores se mostrará un diálogo informando de esto al usuario, en caso contrario la clase lanzará un Intent lanzando la actividad SearchActivity y pasándole como parámetros cada una de los datos rellenados por el usuario.

SearchActivity

- **Tipo:** Actividad**.**
- **Funcionalidad:** mostrar del mismo modo que se hacer en CarFragment cada uno de los viajes que coinciden con los parámetros de búsqueda del usuario.
- **Implementación**: para desarrollar ésta actividad basta con recoger los parámetros que se pasan a través del Intent y ejecutar una consulta que devuelva todos los viajes que se amolden a éstos filtros y en los que el usuario no aparezca ni como conductor ni como pasajero. Una vez tengamos el conjunto de viajes ocurre exactamente igual que al mostrar los viajes del usuario, si el resultado es vacío se mostrará un icono y mensaje informando de esto al usuario. En caso adverso se mostrarán todos los viajes mediante un RecyclerView. La diferencia es que el dialogo que se muestra al pulsar sobre un viaje es distinto:

  Si el viaje no cuenta con plazas disponibles se mostrará un mensaje informativo al usuario impidiendo interactuar con el propio viaje.

  Si el viaje cuenta con plazas disponibles el diálogo ofrecerá la posibilidad de informarse sobre el trayecto al usuario mediante la aplicación Google Maps. También ofrecerá la posibilidad de unirse al viaje y formar parte de los pasajeros. En caso de pulsar este último botón, se realizará la consulta necesaria para guardar al usuario como pasajero y pueden ocurrir dos cosas:

    - Si el viaje cuenta con trayecto de vuelta, se mostrará un nuevo diálogo informando de esto al usuario y permitiendo unirse a la vuelta, ejecutando nuevamente una consulta a través de Firebase.
    - En caso de no contar con trayecto de vuelta, se cerrará la actividad.

  Esta actividad también cuenta con un botón situado en la esquina superior derecha que permite al sistema estar pendiente de si el usuario lo pulsa, en este caso se procederá al cierre de la actividad. El hecho de elegir una implementación a través de una actividad en vez de un fragmento responde a temas de usabilidad, pues para el usuario es más diferenciable en este caso

una navegación a través de actividades que nuevamente sustituyendo un fragment, pues dota de mayor navegabilidad y diferenciación a la aplicación.

AddFragment

- **Tipo:** Fragmento.
- **Funcionalidad:** permitir a cualquier usuario la creación de un viaje.
- **Implementación:** este fragmento supone una de las clases más complejas en cuento a código. Al igual que en la sección Buscar, éste fragmento funcionará a modo de formulario.

  Cuando iniciamos el fragmento, lo primero que hace la clase es cargar el mapa a través de la librería correspondiente. Para ello se pasa como parámetro la latitud y longitud de la capital Malagueña y se pasa como parámetro al objeto GoogleMap ofrecido por la librería, permitiendo que el mapa se cree centrado en estas coordenadas. También se ajusta el zoom y se activa la interacción para poder navegar por él a través de gestos.

  Una vez realizado esto y sin que todavía el usuario le haya dado tiempo a interactuar, la clase sigue ejecutando métodos. En este caso se va a proceder a crear un objeto Viaje con los datos invariantes y que no debe rellenar el usuario, preparándolo así para su inserción en la base de datos una vez el usuario se decida a ello. Se crea tanto un viaje de ida como un posible viaje de vuelta.

  Una vez carga la interfaz el usuario ya puede interactuar sobre ella:

  Al pulsar sobre Origen se abrirá una actividad proporcionada por la librería Places API que devolverá la dirección seleccionada por el usuario al Fragment. Al abrir esta actividad se controla mediante código que sólo se propongan calles de la provincia de Málaga, enviando como parámetros de longitud las coordenadas que delimitan la provincia. Una vez se tiene el origen se calcula su id para la posterior petición http y se muestra en el mapa un icono identificativo sobre la calle introducida.

Al pulsar sobre destino nuevamente se ofrecerá al usuario la elección de un destino mediante un diálogo. Al igual que en el paso anterior, se guarda la id del destino en el objeto Viaje y se muestra un identificador sobre el mapa.

Al igual que ocurre en el fragmento de búsqueda, si el usuario pulsa sobre la fecha o la hora se abrirá un diálogo que le permita escoger respectivos parámetros.

En todo momento la clase se encuentra a la espera de que el usuario pulse sobre el botón flotante + que se encuentra en las esquina inferior derecha. Si el usuario pulsa sin haber escogido los cuatro parámetros anteriores se notificarán mediante un diálogo esto y se esperará a una nueva pulsación. En cambio si todo es correcto, se muestra un diálogo personalizado:

- **Plazas disponibles**: permite al usuario elegir cuantas plazas dispone para los usuarios que quieran realizar el trayecto.
- **Anotación**: permite al usuario ofrecer cualquier información relevante sobre el trayecto.
- **Modelo coche:** permite al usuario hacer una referencia sobre el coche, ya sea color, modelo…
- **Ida y vuelta**: permite al conductor crear un viaje de vuelta a la hora que se selecciona en la misma sección.
- **Crear:** ejecuta la consulta necesaria para la creación del viaje o los viajes en la base de datos.

Mientras se muestra el diálogo la propia clase hacer una conexión HTTP mediante la API DIrections de Google que le devuelve un JSON con información sobre el trayecto. Ésta conexión se hace mediante una hebra que se ejecuta en segundo plano y que lleva a cabo la interpretación del JSON y devuelve a nuestro fragmento la duración del viaje. Con esta duración se calcula el tiempo de llegada estimado y se guarda en el objeto Viaje.

ChatFragment

- **Tipo:** Fragmento.
- **Funcionalidad:** mostrar por cada uno de los viajes no terminados a los que pertenece el usuario un acceso al chat grupal de pasajeros.
- **Implementación**: este fragmento es muy sencillo. Una vez iniciado realiza una consulta a las base de datos para obtener todos los viajes en los que el usuario es conductor o pasajero y que no están terminados para después mostrarlos a través de un RecyclerView mediante una interfaz muy sencilla. En caso de no haber ningún viaje se mostrará un texto y un viaje descriptivo.

  Si el chat contiene viajes, se mostrará el último mensaje escrito en el chat para que el usuario no tenga que acceder a él si no quiere. Al pinchar sobre un chat se procederá a lanzar mediante un Intent la siguiente actividad.

ChatActivity

- **Tipo**: Actividad.
- **Funcionalidad:** mostrar cada uno de los mensajes que se han escrito en ese chat.
- **Implementación**: para la implementación primero se lleva a cargo la visualización de la interfaz y personalización según los datos de ese chat. Una vez hecho esto se muestra mediante un RecyclerView todos los mensajes que se han escrito en el chat y se enfoca en el último para evitar hacer scroll. En el adaptador cada vez que se ejecute el método que carga la interfaz del mensaje se comprueba si ese mensaje es del usuario o pertenece a otro pasajero. En caso de ser del usuario se mostrará a la derecha con una interfaz simple, en caso contrario se mostrará a la izquierda con una interfaz más compleja, que permite identificar al usuario mediante su foto y nombre.

  En todo momento la clase se encuentra a la espera de que el usuario pulse sobre:

- **Flecha**: situada en la esquina superior izquierda permite cerrar la actividad y volver al listado de chats.
- **Mensaje**: permite la inserción de texto o emoticonos para su envío.
- **Icono enviar**: situado abajo a la derecha permite enviar el mensaje introducido, en caso de que éste sea vacío no se enviará nada.

ProfileFragment

- **Tipo:** Fragmento.
- **Funcionalidad:** permitir el cierre de sesión y ver un historial de los viajes realizados así como su votación.
- **Implementación**: una vez ejecutado se encarga de cargar la foto de perfil y nombre del usuario cuya sesión está iniciada. Al hacer esto se procederá a la consulta de todos aquellos viajes en los que ha estado el usuario y que estén terminado y se procederán a su visualización. El adaptador será el encargado de decidir si mostrar o no el método de votación:

  Si el usuario no es conductor, se consulta si ya ha procedido a la votación y en caso afirmativo se detalla el icono que selecciona, en caso contrario no se detalla ninguno y se permite la votación.

  Si el usuario es conductor no se permitirá la votación ni se mostrarán iconos.

  Desde su ejecución el fragmento está pendiente de que el usuario pulse sobre cerrar sesión, produciéndose la llamada a la API y cerrando la sesión.

5.5 Librerías

La implementación de estas pantallas así como de la interfaz de usuario y las demás clase javas no podría haber sido posible sin el soporte de una serie de librerías.

Una librería es una herramienta externa al SDK Android que permite añadir funcionalidad a nuestro entorno así como importar nuevos componentes. El uso de

librería en el desarrollo de aplicaciones Android es muy común hoy en día ya que permiten ahorrase trabajo o directamente realizar acciones que de otra manera no serían posibles.

Para diferenciar el tipo de librería vamos a hablar de librerías funcionales, aquellas que añaden función a nuestro código y librerías de interfaz, aquellas que permiten añadir componentes a nuestra interfaz.

5.5.1 Librerías funcionales

Firebase-Core

Permite implementar el SDK principal de Firebase así como añadir datos analíticos a la aplicación para su posterior consulta a través de la consola.

Firebase-Auth

Permite añadir métodos de manejo de tokens de autenticación. EL uso de esta librería hace posible la autenticación a través de Google y su conexión con el servidor.

Firebase-UI-Auth

De la mano con la anterior librería, permite añadir métodos de interfaz ligados a la autenticación. Sin la implementación de FirebaUI se tendrían que crear los componentes de autenticación según el proveedor y la carga de perfiles, lo que sería algo muy complejo.

Play-Services-Auth

Última librería asociada a la autenticación. Para el manejo de sesión Google obliga a que nuestra aplicación esté subida a Google Play, aunque sea de manera oculta. Mediante ésta librería se permite hacer las comprobaciones pertinentes para su funcionamiento. Si queremos sacar una versión estable de la aplicación es totalmente necesaria.

Picasso

La librería Picasso [24] permite la carga de imágenes de forma eficiente en nuestro código. Además contiene una serie de métodos para manipular imágenes facilitándo las tareas y aprovechando al máximo los recursos del sistema.

Places

Importa los métodos necesarios para la incorporación de Places Api [25]. Gracias a esta librería podemos seleccionar el origen de nuestros viajes.

Firestore

Permite acceder a los objetos necesarios para la conexión con Cloud Firestore.

Firestore-UI

Permite establecer conexión entre nuestra interfaz y Cloud Firestore para que en todo momento tengamos actualizados nuestros elementos en pantalla y a tiempo real.

Firebase-Database

Permite acceder a los objetos necesarios para la conexión con Real Time Database.

Crashlytics

Mediante la importación de esta librería podemos registrar todos los errores que ocurren en la aplicación y hacer un seguimiento de éstos mediante la consola.

Cloud Messaging

Permite enviar notificaciones a los usuarios mediante una llamada HTTP al servidor.

5.5.2 Librerías de diseño

RecyclerView

Permite incrustar en nuestro layout los recyclerview necesarios para el listado de datos. También permite acceder a diferentes objetos para el manejo de éstos desde código.

Android-Design

Brinda la posibilidad de introducir los últimos componentes desarrollados por el equipo de Android como por ejemplo el componente CardView.

CircleImageView

Permite el acceso al componente CircleImage. Éste componente permite incrustar una imagen sobre una vista redonda.

Capítulo 6
Pruebas

6.1 Fase de pruebas

Tras la realización de cada uno de los prototipos que implementasen los correspondientes casos de uso y requisitos, se llevó a cabo la fase de pruebas. Al desarrollar una metodología evolutiva, el proceso de pruebas se ha realizado con un grupo reducido de usuarios, estudiantes de la Universidad de Málaga, que han probado cada prototipo, dando su opinión y ofreciendo nuevas posibles mejoras, que se convertirían en nuevos requisitos.

A la hora de estructurar las pruebas se hará de igual manera que con la fase de análisis, se expondrán los problemas que han surgido una vez el grupo de usuario ha probado el prototipo, explicando cada posible fallo o mejora y proponiendo una solución.

A continuación se mostrarán los diferentes errores y nuevas implementaciones que se integrarían en los requisitos, tal y como se recogieron en el documento.

6.2 Pruebas primer prototipo

Tras la realización del primer prototipo se entregó a cada usuario un manual en el que explicaba la finalidad de la aplicación y se hacía una explicación de cada una de las funciones de ésta. El primer prototipo no implementaba la interfaz final, por lo que se dejó claro que lo que se pretendía era analizar la funcionabilidad de la aplicación.

Errores

ERR01	Controlar Viajes Largos
Descripción	Cuando se crea un viaje largo o “imposible”, por ejemplo desde Estados Unidos, la aplicación falla y se reinicia.

Explicación	La clase encargada de interpretar la cadena de texto con la duración devuelta por la API en ese momento, está diseñada para obtener distancias existentes, obviando que podría ser devuelto un valor nulo.
Corrección	Basta con añadir la posibilidad de que si es devuelto un valor nulo se indique por pantalla que ese origen no es posible seleccionarlo. Sin embargo, al analizar el mercado al que va destinada la aplicación, es mejor añadir un requisito nuevo: sólo se podrán escoger calles de la provincia de Málaga, pues al final es a donde va destinada la aplicación. De esta manera se elimina la posibilidad de que la duración sea nula porque desde cualquier punto de la provincia se puede llegar en vehículo a los diferentes destinos.

Tabla 32 : ERR01 - Controlar Viajes Largos

ERR02	Número plazas igual a 1
Descripción	Al crear un viaje con una plaza disponible la aplicación se cierra y no crea el viaje.
Explicación	La selección de plazas disponibles se hace mediante un componente nativo de Android. Este componente permite elegir un número entre un rango a modo de carrusel, sin embargo devuelve un número menor puesto que los almacena en un array. Por ejemplo si queremos una sola plaza devuelve la posición del

	uno que sería la posición cero del array y la aplicación no permite crear un viaje sin plazas pues no tiene sentido.
Corrección	Realizar un componente manual en la nueva interfaz que permitiese escoger plazas de forma atractiva y entre los valores uno y seis.

Tabla 33 : ERR02 - Número plazas igual a 1

Nuevas implementaciones

RF	Ida y Vuelta
Descripción	Añadir la posibilidad de crear un viaje de vuelta.
Implementación	A la hora de desarrollar el segundo prototipo se implementará un botón que permita escoger si se quiere realizar el trayecto de vuelta y la posibilidad de escoger una hora.

Tabla 34 : RF - Ida y Vuelta

RF	Proximidad
Descripción	Cuando se inerte una calle como origen, que la API ofrezca primero los sitios más cercanos.

Implementación	Al solucionar el error 1 se implementó esto también.

Tabla 35 : RF – Proximidad

RF	Valoración
Descripción	Elaborar un sistema de puntuación al conductor una vez haya terminado el viaje. Cuando un usuario busque viajes pueda ver la valoración del conductor.
Implementación	Se decide implantar un sistema de votación a favor o en contra en vez de un sistema de puntuación numérico a través de estrellas.

Tabla 36 : RF – Valoración

6.3 Pruebas Segundo Prototipo

En el segundo prototipo se implementaron los nuevos requisitos surgidos de las pruebas, se depuró el código y se corrigieron los anteriores errores. Nuevamente se entregó un documento a los usuarios que probaría la aplicación. En este documento se hacía hincapié en que ya no sólo se analizaría la funcionalidad, sino que también la interfaz de usuario implementada.

Errores

ERR03	El mes siempre es Abril
Descripción	Cuando se crea el viaje se guarda en la base de datos el mes incorrecto.
Explicación	El diálogo que permite la selección de meses enumera a éstos del cero al once en vez del uno al doce como estamos acostumbrados. Al hacer las pruebas en el mes de mayo, el mes que se guarda en la base de datos siempre es uno menos, abril.
, **Corrección**	Cuando se lleve a cabo la inserción del mes en el modelo viaje para su después guardado en la base de datos, se debe guardar un mes más, si el diálogo devuelve un cuatro se debe guardar en la base de datos un cinco.

Tabla 37 : ERR03 - El mes siempre es Abril

ERR04	Unirse a la vuelta
Descripción	En ningún momento se puede unir a un viaje de vuelta.
Explicación	A pesar de haber implementado la posibilidad de crear un viaje de vuelta, en ningún momento se puede llevar a cabo la unión a un trayecto de vuelta, quedando siempre vacío de pasajeros.

, **Corrección**	Cuando un usuario pulse en “Unirse al viaje” en caso de existir un viaje de vuelta para esa ida, mostrar un diálogo informando al usuario de éste hecho y permitiéndole pertenecer al viaje de vuelta.

Tabla 38 : ERR04 - Unirse a la vuelta

ERR04	Nombre estático
Descripción	El nombre del usuario en el Perfil no se carga de la base de datos, siempre es el mismo de ejemplo.
Explicación	Cuando se carga el fragmento perfil y se lleva a cabo la carga de la foto de perfil en ningún momento se asocia el nombre del usuario cuya sesión está iniciada al cuadro de texto destinado a ello.
Corrección	Basta con recuperar el nombre del objeto FirebaseUser y mostrarlo a través del cuadro de texto destinado a ello.

Tabla 39 : ERR04 - Nombre estático

ERR05	Modo rotación reinicia la app

Descripción	Al rotar la pantalla se produce un reinicio de la aplicación que dificulta una buena experiencia de usuario.
Explicación	En Android el ciclo de vida de las actividades viene impuesto por el sistema. Sin embargo se puede manipular su comportamiento mediante una serie de clases. Un comportamiento que viene impuesto es que cuando se gire el móvil rotando la pantalla se reinicie la aplicación y con ella la carga de datos.
Corrección	Basta con añadir una línea en el manifiesto para que no se permita la rotación de pantalla en la aplicación, pues no interesa en éste proyecto esa funcionalidad.

Tabla 40 : ERR05 - Modo rotación reinicia la app

Nuevas implementaciones

Interfaz

RF	**Fechas en viajes**
Descripción	Añadir a los viajes la fecha en la que se va a realizar.
Implementación	Añadir al lado de los “no me gusta” un nuevo cuadro que muestre la fecha de forma atractiva

Tabla 41 : RF - Fechas en viajes

Capítulo 7
Conclusiones

7.1 Conclusiones

7.1 Conclusiones

Desde que aprendí a desarrollar aplicaciones móviles supe que era la tecnología a utilizar a la hora de desarrollar mi trabajo final de grado. Estuve pensando varias ideas para implementar, pero sin duda elegí esta porque pienso que es una herramienta muy útil para la Universidad que ayudaría a muchos alumnos e incluso profesores y otros trabajadores. También otro punto a favor de éste proyecto era la idea de aportar mi grano de arena a la ayuda de la conservación del medio ambiente, una problemática de hoy en día y con la que estoy muy concienciado y que con la tecnología, podemos ser capaces de revertir o por lo menos ayudar a que no se deteriore más nuestro mundo.

Desde el punto de vista de producto, creo que se han cumplido todas las expectativas, incluso superándolas. La aplicación que se ha desarrollado es totalmente funcional y está preparada para ser usada por todo aquél que lo necesite. Creo que el acabado en cuanto a usabilidad y funcionalidad es bastante bueno.

En un principio abordar la aplicación no fue fácil, nunca había trabajado con tecnologías como Firebase y carecía totalmente de conocimientos sobre ella. Sin embargo nunca dudé en realizar éste proyecto, si algo he aprendido desarrollándolo es que si se tienen los medios, todo es posible en el mundo de la programación.

Las fases de análisis y diseño fueron muy fáciles de abordar en comparación con la implementación, en la que surgen muchas dudas. Sin embargo, el ser constante todos los días y querer realizar mi propuesta me ha servido para aprender muchas tecnologías y manejos de APIs y librerías. El diseño de la interfaz también fue un quebradero de cabeza, pues me considero una persona perfeccionista y nunca quedaba a gusto con el resultado, aunque creo que el producto final es bastante profesional, pero siempre mejorable.

En resumen, abordar este proyecto ha supuesto un reto bastante ambicioso para mí dado el corto período en el que se ha desarrollado, pero no me arrepiento de nada en realizarlo. Informándome a través de internet me he empapado de nuevos conocimientos que me servirán en mi vida laboral.

Abordar tecnologías de las que carecemos conocimiento nos desarrolla más como Ingenieros de Software y nos ayuda a tener esa ambición por mejorar y aprender cosas nuevas que creo que es tan necesaria en ésta profesión.

Capítulo 8
Líneas futuras

8.1 Líneas futuras

8.1 Líneas Futuras

A la hora de hablar sobre futuros trabajos dentro del proyecto se va a diferencia entre el cliente y el servidor:

Cliente

Aunque el acabado de la aplicación sea bastante bueno no quiere decir que no sea mejorable. En cuanto a funcionalidad se podría establecer un método a la hora de crear viajes que indicase el municipio o barrio desde el cual se iniciará el viaje, facilitando la búsqueda a los usuarios.

El sistema de votación, aunque es funcional quizás no sea del todo exhaustivo, por lo que se podría elaborar una serie de métodos de puntuación numérica a través de estrellas. A medida que se use la aplicación irán surgiendo nuevas propuestas o requisitos que sería bueno contemplar para adaptar la aplicación a todo usuario.

La autenticación es otro tema importante, se podría aumentar el número de proveedores con los que autenticarse, ya sea Facebook o Twitter, aunque lo más lógico sería establecer un inicio de sesión con la cuenta de usuario de la UMA de cada alumno, profesor o trabajador.

Desde el punto de vista de la interfaz, su mejora es evidente. Aunque está bien implementada, dar paso a un diseño realizado por un profesional que elabore sus propios componentes y enfocados a la usabilidad de esta aplicación en concreto sería un gran avance.

Por otro lado, aunque Android es el sistema operativo móvil más utilizado, existen más aparte de él. Otro sistema operativo muy importante es IOS. Realizar la misma aplicación para dispositivos IOS aunando el servidor sería una gran mejora.

Servidor

En cuento al servidor, las herramientas que facilitan Firebase y Google Cloud Platform son muy potentes. Sin embargo, puede que un día su uso no sea rentable. Para ello estaría bien la creación de un propio servidor que ofrezca las mismas características que Firebase ofrece al proyecto e incluso aumentarlas en función de las nuevas necesidades.

También sería interesante la creación de algún tipo de algoritmo que cifrase el guardado de la información del usuario en la base de datos, aunque no sea crítica.

Ligado al tema del cliente, en cuanto a autenticación sería bastante bueno la creación de un método para la creación de usuario y contraseña con la que podamos manejar nuestros propios datos a guardar en la base de datos.

Creo que al ser un proyecto que puede evolucionar e ir adaptándose a las diferentes necesidad pueden ir surgiendo nuevas mejoras a parte de estas. Lo que sí está claro es que es una aplicación compleja y que deberá ser mantenida para corregir y evitar errores en un futuro así como responder a las nuevas actualizaciones del sistema operativo.

Capítulo 9
Referencias Bibliográficas

[1] https://www.blablacar.es/
[2] https://amovens.com/
[3] https://journify.es/
[4] https://developer.android.com/guide?hl=ES
[5] https://firebase.google.com/?hl=es-419
[6] https://cloud.google.com/why-google-cloud/?hl=es
[7] https://www.java.com/es/download/faq/whatis_java.xml
[8] https://nodejs.org/es/about/
[9] http://jorgetrejos.blogspot.com/2010/08/modelo-evolutivo.html
[10] https://firebase.google.com/docs/auth/?hl=es-419
[11] https://firebase.google.com/docs/database?hl=es-419
[12] https://firebase.google.com/docs/firestore?hl=es-419
[13] https://firebase.google.com/docs/crashlytics/?hl=es-419
[14] https://firebase.google.com/docs/cloud-messaging
[15] https://firebase.google.com/docs/functions
[16] https://firebase.google.com/support/privacy?hl=es-419
[17] https://developer.android.com/guide/components/activities.html?hl=ES
[18] https://developer.android.com/guide/topics/ui/declaring-layout?hl=es-419
[19] https://developer.android.com/training/basics/firstapp/starting-activity?hl=es
[20] https://developer.android.com/guide/components/fragments?hl=es-419
[21] https://developer.android.com/guide/topics/ui/layout/recyclerview
[22] https://developer.android.com/reference/android/widget/Adapter
[23] https://developer.android.com/guide/topics/ui/dialogs?hl=es-419
[24] https://square.github.io/picasso/
[25] https://developers.google.com/places/web-service/intro

Anexo A
Manual Instalación

A.1 Instalación a través de APK

Se procederá a detallar los pasos necesarios para instalar la aplicación a través del archivo .apk que se ha otorgado. Dado que la aplicación se encuentra en el Play Store registrada en fase Alpha, se debe activar una serie de opciones para el correcto funcionamiento.

1. **Activar "Orígenes desconocidos"**

   - Entrar en el menú de **Ajustes** de la barra de notificaciones.
   - Entrar en el apartado **Seguridad.**
   - Entrar en el apartado **Administración de dispositivos.**
   - Activar **Orígenes Desconocidos.**

2. **Desactivar "Play Protect"**

   - Acceder al **Play Store**.
   - En el menú lateral pulsar sobre **Play Protect**.
   - Acceder a los ajustes.
   - Desactivar opciones.

3. **Instalar APK**

   Una vez tengamos estos pasos realizados, basta con meter en la memoria del teléfono el archivo .apk y desde el Gestor de archivos instalarlo**.**

A.2 Instalación a través del código

Se procederá a detallar los pasos necesarios para instalar la aplicación a través del código facilitado. En este caso no hace falta activar ninguna opción, ya que el propio IDE comunica al teléfono de que se trata de un archivo de pruebas y por tanto puede hacer uso de los servicios de Google aunque se encuentre en fase Alpha.

1. **Importar proyecto en Android Studio**

   - Guardar el proyecto facilitado en una carpeta del ordenador.
   - En Android Studio pulsar en **File -> Open.**
   - Abrir la ruta del proyecto.

2. **Compilar proyecto**

   - Pulsar sobre el botón del martillo verde.

3. **Activar Depuración/Instalación USB**

   - En el teléfono dirigirse a la ruta **Ajustes -> Opciones de desarrollador.**
   - Activar **Depuración USB**.
   - Si el teléfono posee la **opción Instalar vía USB**, activarla también.

4. **Instalar proyecto**

   - Pulsar sobre el botón reproducir de color verde.

Anexo B
Manual Usuario

B.1 Inicio de sesión

1. Abrir la aplicación desde el menú del teléfono móvil.
2. Una vez cargada la pantalla principal, pulsar sobre el botón “**Iniciar Sesión**”
3. Pulsar sobre la cuenta Google deseada para interactuar con la aplicación.
4. Esperar a ser redirigido a la pantalla principal “**Viajes**”.

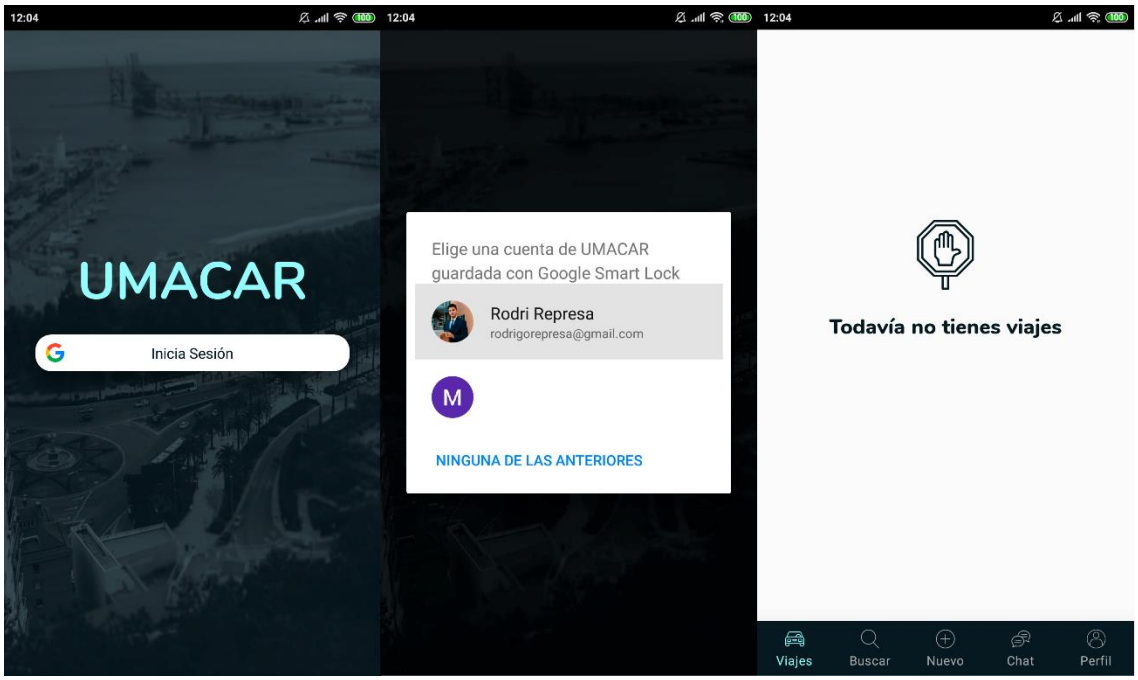

Ilustración 7 : Inicio Sesión

B.2 Crear un viaje

1. En la barra inferior pulsar sobre “**Nuevo”**.
2. Rellenar todos los datos necesarios para la creación del viaje.
3. Pulsar sobre el botón situado en la esquina inferior derecha **+**.
4. Nuevamente, rellenar los datos del diálogo.
5. Pulsar en el botón “**Crear”** de la esquina inferior derecha del diálogo.

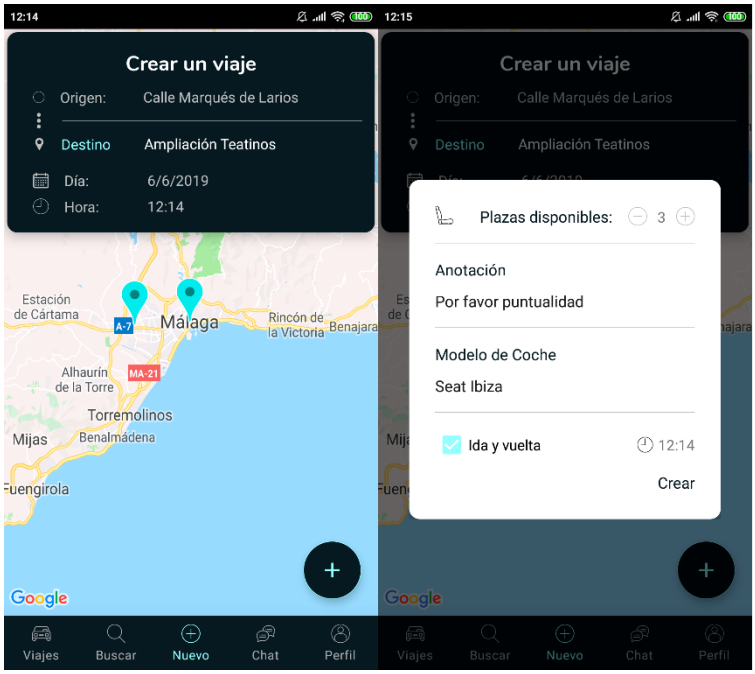

Ilustración 8: creación de viaje

B.3 Unirse a un viaje

1. En la barra inferior pulsar sobre “**Buscar”**.
2. Rellenar todos los datos necesarios para la búsqueda del viaje.
3. Pulsar sobre el botón “**->”**.
4. Pulsar sobre el viaje que deseamos.
5. Pulsar en “**Unirse al viaje**”.

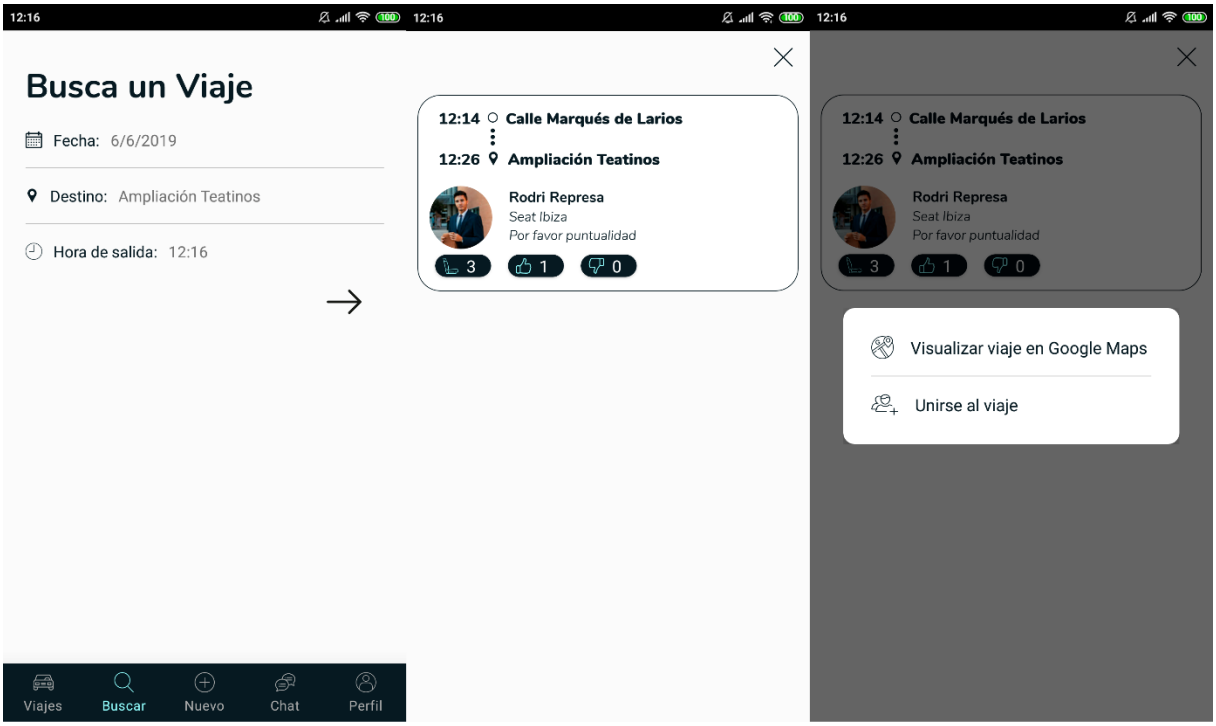

Ilustración 9 : unirse a un viaje

B.4 Acceder a un chat

1. En la barra inferior pulsar sobre “**Chat”**.
2. Aparecerán todos los viajes **No Terminados** a los que se pertenece.
3. Pulsar sobre el chat deseado.
4. Pulsando sobre “**Mensaje”** se podrá realizar el envío de texto y emoticonos.

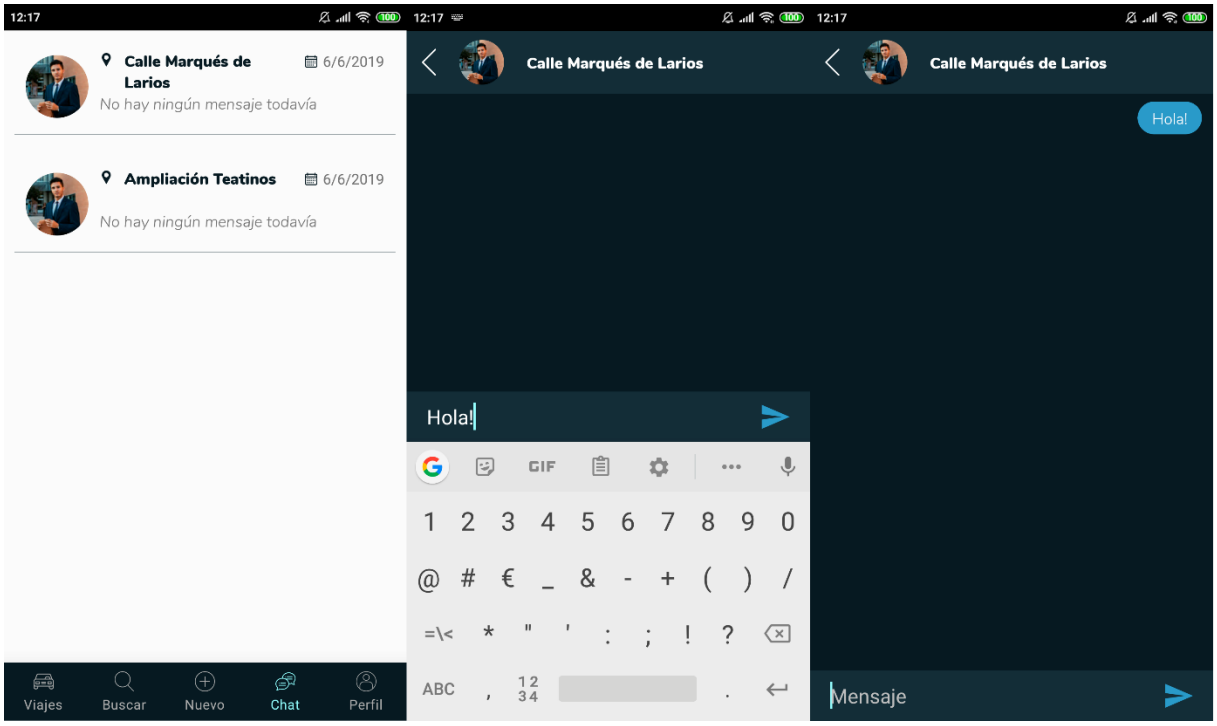

Ilustración 10 : acceder al chat

B.5 Eliminar/Terminar/Salir de un viaje

1. En la barra inferior pulsar sobre “**Viajes”**.
2. Aparecerán todos los viajes **No Terminados** a los que se pertenece.
3. Pulsar sobre el viaje deseado.
4. Pulsar en “**Eliminar/Terminar/Salir del viaje**”, según el rol que se tenga en el viaje.

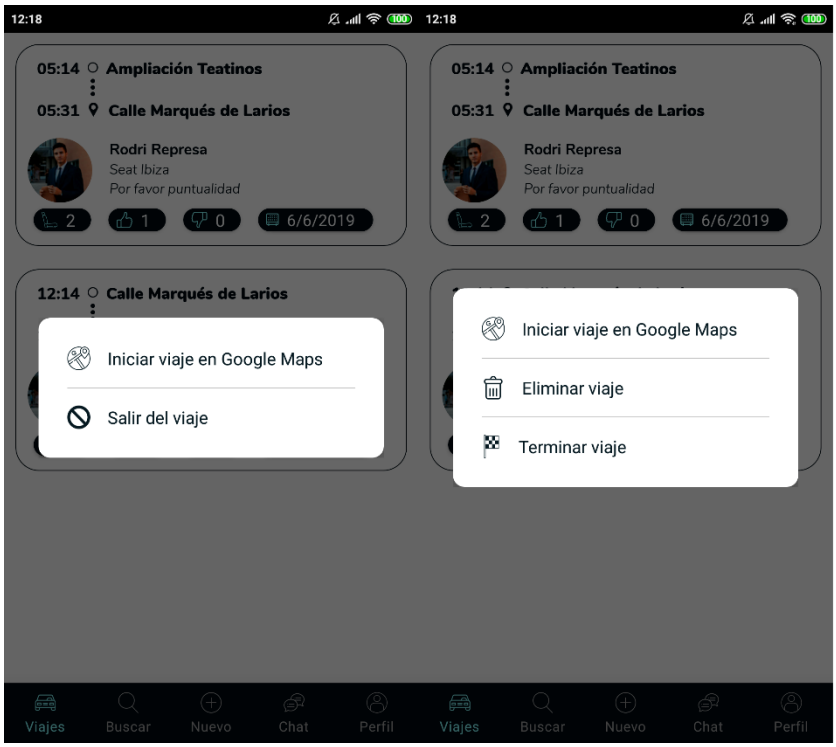

Ilustración 11 : interacción con viaje

B.6 Votar un viaje

1. En la barra inferior pulsar sobre “**Perfil”**.
2. Aparecerán todos los viajes **Terminados** a los que se ha pertenecido.
3. Pulsar sobre el **icono de votación** deseado para votar al correspondiente conductor.

B.7 Cerrar Sesión

1. En la barra inferior pulsar sobre “**Perfil”**.
2. Pulsar en “**Cerrar sesión”**, situado debajo de la foto.

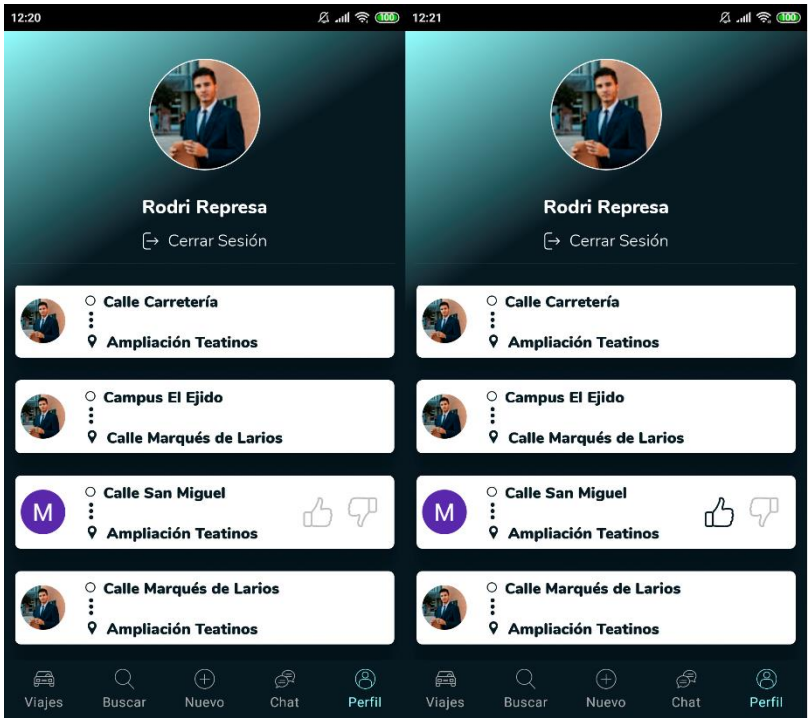

Ilustración 12 : votar viaje y cerrar sesión