



UNIVERSIDAD DE MÁLAGA



Grado en Ingeniería Informática

Diseño y desarrollo de una aplicación para compartir coche
con Flutter

Design and development of a car sharing app with Flutter

Realizado por
Bryan Velicka Leka

Tutorizado por
Francisco José Jaime Rodríguez

Departamento
Lenguajes y ciencias de la computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, mayo 2023



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA**

**DISEÑO Y DESARROLLO DE UNA APLICACIÓN
PARA COMPARTIR COCHE CON FLUTTER**

**DESIGN AND DEVELOPMENT OF A CAR SHARING
APP WITH FLUTTER**

Realizado por
Bryan Velicka Leka

Tutorizado por
Francisco José Jaime Rodríguez

Departamento
Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, MAYO DE 2023

Fecha defensa: junio de 2023

Abstract

Current students at the University of Malaga have formed groups on instant messaging applications with the aim of car sharing. These groups arose from the need of some students to go to university from distant or poorly communicated places by public transport and the mutual benefit that is achieved if the price of fuel is paid between the driver and passengers. This also has the environmental benefit of having four people in one car instead of just one. The fact that car sharing offers are managed by instant messaging brings with it the difficulty of managing trips and a number of other problems associated with having public messaging groups. The purpose of this work is to facilitate the management and discovery of rides and to solve some of the problems encountered by designing and developing a mobile application in the Flutter development framework. The target platform of the application is going to be Android but the development framework has been selected to make it compatible with more platforms.

Keywords: Flutter, Car sharing, Mobile application, Students

Resumen

Actualmente los alumnos de la Universidad de Málaga han formado grupos en aplicaciones de mensajería instantánea con el objetivo de compartir coche. Dichos grupos surgieron de la necesidad de algunos alumnos de ir a la universidad desde lugares lejanos o mal comunicados por transporte público y el beneficio mutuo que se consigue si se paga el precio del carburante entre el conductor y los pasajeros. Esto además tiene el beneficio medioambiental que supone que en un coche vayan cuatro personas en vez de solo una. El hecho de que las ofertas para compartir coche se gestionen por mensajería instantánea trae consigo la dificultad de gestionar los viajes y otra serie de problemas relacionados con tener grupos de mensajería públicos. El propósito de este trabajo de fin de grado es facilitar la gestión y descubrimiento de viajes y solucionar algunos de los problemas encontrados mediante el diseño y desarrollo de una aplicación móvil en el marco de desarrollo Flutter. La plataforma objetivo de la aplicación va a ser Android pero se ha seleccionado un marco de desarrollo para poder hacerla compatible con más plataformas.

Palabras clave: Flutter, Compartir coche, Aplicación móvil,
Alumnos

Índice

1. Introducción	11
1.1. Motivación	11
1.2. Objetivos	12
1.3. Estructura del documento	12
2. Tecnologías y metodologías de trabajo usadas	15
2.1. Tecnologías usadas	15
2.1.1. Flutter	15
2.1.2. Dart	15
2.1.3. Supabase	16
2.1.4. Postgres	16
2.1.5. PL/SQL	16
2.1.6. Git	16
2.1.7. Visual studio code	17
2.1.8. Android Studio	17
2.1.9. Modelio	17
2.1.10. Visual paradigm	17
2.1.11. Overleaf	18
2.1.12. Krita	18
2.1.13. Pencil project	18
2.1.14. Trello	18
2.1.15. Oracle datamodeler	18
2.2. Metodología de trabajo	19
3. Diseño	21
3.1. Requisitos	21
3.1.1. RF-1. Iniciar sesión	23
3.1.2. RF-1.1. Iniciar sesión con Discord	23

3.1.3.	RF-2. Registro de usuario	23
3.1.4.	RF-2.1. Registro automático con Discord	23
3.1.5.	RF-3. Borrar cuenta de usuario	23
3.1.6.	RF-4. Cerrar sesión	23
3.1.7.	RF-4.1. Cerrar sesión automáticamente	24
3.1.8.	RF-5. Mantener sesión abierta	24
3.1.9.	RF-5.1. Renovar token automáticamente	24
3.1.10.	RF-6. Selección de posición	24
3.1.11.	RF-6.1. Mirar posición geográfica	24
3.1.12.	RF-6.2. Selección de zonas concretas	24
3.1.13.	RF-7. Crear oferta de viaje	25
3.1.14.	RF-7.1. Ver oferta de viaje	25
3.1.15.	RF-7.2. Borrar oferta de viaje	25
3.1.16.	RF-7.3. Editar oferta de viaje	25
3.1.17.	RF-7.4. Buscar oferta de viaje	26
3.1.18.	RF-7.5. Dar detalles de viaje	26
3.1.19.	RF-7.6. Aceptar viaje	26
3.1.20.	RF-7.7. Viajes periódicos	26
3.1.21.	RF-7.8. Eliminar usuarios apuntados	26
3.1.22.	RF-8. Iniciar chat entre usuarios	26
3.1.23.	RF-8.1. Ver mensajes de chat	27
3.1.24.	RF-8.2. Enviar mensajes	27
3.1.25.	RF-8.3. Recibir mensajes	27
3.1.26.	RF-8.4. Borrado de mensajes automático	27
3.1.27.	RF-9. Aplicar filtro de búsqueda de viajes	27
3.1.28.	RF-9.1. Filtrar por hora	28
3.1.29.	RF-9.2. Filtrar por posición	28
3.1.30.	RF-9.3. Eliminar filtro	28
3.1.31.	RF-10. Cambiar tema de color	28
3.1.32.	RNF-1. Registro de usuario con Discord sencillo	29
3.1.33.	RNF-2. Intuitividad	29

3.1.34. RNF-3. Eficiencia	29
3.1.35. RNF-4. Fluidez	29
3.1.36. RNF-5. Interfaz agradable a la vista	29
3.1.37. RNF-6. Fácil adaptación del inicio de sesión a nuevos procedimientos	29
3.1.38. RNF-7. Modularidad	30
3.1.39. RNF-8. Seguridad	30
3.2. Casos de uso	31
3.2.1. Diagrama	31
3.2.2. Iniciar sesión	32
3.2.3. Iniciar sesión con Discord	33
3.2.4. Registro de usuario	34
3.2.5. Registro automático con Discord	36
3.2.6. Borrar cuenta	37
3.2.7. Cerrar sesión	38
3.2.8. Ver oferta de viaje	39
3.2.9. Crear oferta de viaje	40
3.2.10. Editar oferta de viaje	42
3.2.11. Borrar oferta de viaje	43
3.2.12. Aceptar viaje	44
3.2.13. Dar detalles de viaje	45
3.2.14. Eliminar usuarios apuntados	46
3.2.15. Selección de zonas concretas	47
3.2.16. Selección de posición	48
3.2.17. Aplicar filtro de búsqueda	49
3.2.18. Filtrar por posición	50
3.2.19. Filtrar por hora	51
3.2.20. Eliminar filtro	52
3.2.21. Iniciar chat entre usuarios	53
3.2.22. Ver mensajes de chat	54
3.2.23. Enviar mensajes	55
3.2.24. Recibir mensajes	56

3.2.25. Cambiar tema de color	57
3.3. Diagrama de base de datos	58
3.3.1. Tabla viaje	58
3.3.2. Tabla usuario	60
3.3.3. Tabla chat	62
3.3.4. Tabla mensaje	62
3.3.5. Tabla Auth.Users	63
3.3.6. Relaciones	63
3.3.7. Funciones creadas	64
3.3.8. Triggers	66
3.3.9. Otras configuraciones en Supabase	68
3.4. Diagramas de clase	77
3.4.1. Modelo	77
3.4.2. Vista	79
3.4.3. Control	80
3.5. Diagramas de secuencia	82
3.5.1. Secuencia de inicio de sesión	82
3.5.2. Secuencia de registro de usuario	83
3.5.3. Secuencia de crear oferta de viaje	84
3.5.4. Secuencia de editar oferta	85
3.5.5. Secuencia de filtrar por posición	86
3.5.6. Secuencia de enviar mensaje	87
3.5.7. Secuencia de cambiar tema de color	88
3.6. Diagramas de estado	89
3.6.1. Diagrama de estado de iniciar sesión	89
3.6.2. Diagrama de estado de editar oferta	90
3.6.3. Diagrama de estado de borrar oferta	91
3.6.4. Diagrama de estado de filtrar por hora	92
3.6.5. Diagrama de estado de enviar mensaje	93
3.7. Maquetas de software	94

4. Desarrollo	107
4.1. Sprint 1	108
4.2. Sprint 2	116
4.3. Sprint 3	123
4.4. Sprint 4	134
5. Validación y verificación	147
5.1. Validación	147
5.2. Verificación	148
6. Conclusiones y Líneas Futuras	153
6.1. Conclusiones	153
6.2. Líneas Futuras	154
Apéndice A. Manual de Instalación	161
Apéndice B. Manual de Usuario	163
B.1. Inicio de sesión y registro	163
B.2. Creando y editando un viaje	166
B.3. Buscar un viaje, verlo y reservarlo	171
B.4. Usando el chat	173
B.5. Configuraciones	175
Apéndice C. Diagramas de casos de uso anteriores	177
Apéndice D. Diagramas de bases de datos anteriores	181
Apéndice E. Diagramas de clases anteriores	185
E.1. Datos	185
E.2. Vista	186
E.3. Control	187
Apéndice F. Diagramas de secuencia anteriores	189
Apéndice G. Diagramas de estado anteriores	195

1

Introducción

1.1. Motivación

A la universidad de Málaga acuden alumnos de muchas zonas distintas que pueden estar a gran distancia. Muchas de estas personas no tienen un coche y su única opción es coger el transporte publico que puede resultar en viaje de una hora o más para llegar. Por otro lado las personas que tienen y usan su coche para acudir a la universidad se han encontrado presionados por la rápida subida de los precios del carburante. Debido a esto los alumnos de la universidad crearon grupos en aplicaciones de mensajería instantánea como *Whatsapp* para compartir coche. De esta forma los alumnos sin coche pueden llegar de manera más rápida y cómoda a la universidad y los conductores no se tienen que preocupar por el precio del carburante porque este se paga entre todos los viajeros.

El uso de una aplicación de mensajería instantánea para ofrecer viajes trae consigo varios problemas:

- Cuando hay muchas personas ofreciendo viajes, es difícil buscar un viaje en concreto que se requiera.
- Para la persona que ofrece el viaje es difícil gestionar quien va con él cuando lleva a varias personas.
- Al ser un grupo de mensajería instantánea, el número de teléfono es visible para todos los integrantes, lo cual no es ideal para la privacidad.
- Los grupos tienen un límite de miembros.
- Como los viajes se ofrecen por mensajes que no se pueden borrar, a veces se da el caso de que un viaje ya no tiene plazas disponibles aunque es reciente. A raíz de lo

mismo, si después de ofrecer un viaje el usuario quiere cambiarlo o no hacerlo, lo único que puede hacer es escribir un nuevo mensaje.

1.2. Objetivos

El objetivo por tanto de este trabajo de fin de grado es diseñar y desarrollar una aplicación móvil *Android* destinada a la publicación, búsqueda y gestión de viajes para compartir coche para los alumnos de la universidad de Málaga. Se busca desarrollar una aplicación que cumpla con el objetivo y que no tenga los problemas inherentes de usar una aplicación de mensajería instantánea. Además se busca que la aplicación sea fácil e intuitiva de usar para alguien que haya usado previamente el grupo en la aplicación de mensajería instantánea.

1.3. Estructura del documento

En esta sección se explica la estructura de este documento que cubre los distintos documentos de diseño y fases del desarrollo de la aplicación.

1. **Introducción:** Se describen las motivaciones para realizar el trabajo de fin de grado, los objetivos que se quieren alcanzar y la estructura del documento.
2. **Tecnologías y metodologías de trabajo usadas:** Se enumeran y describen las tecnologías que han sido usadas en el trabajo fin de grado y las metodologías usadas en cada fase.
3. **Diseño:** Se muestran y explican los documentos generados relacionados con el diseño de la aplicación a desarrollar. Los documentos que se pueden encontrar en esta sección son los de requisitos, diagrama y especificación de casos de uso, diagramas de clase, diagramas de secuencia y diagramas de estado.
4. **Desarrollo:** Se muestra el proceso de desarrollo seguido y lo conseguido en cada fase.
5. **Validación y verificación:** Se validan y verifican que los requisitos han sido cumplidos y se explican las pruebas realizadas.

6. Conclusiones y Líneas futuras: Se muestran el resultado final al que se ha llegado y se dan posibles mejoras que se podrían hacer.

En cuanto a los apéndices, se pueden encontrar los documentos de diseño intermedios junto con un manual de instalación y un manual de usuario.

- **Manual de instalación:** En este apartado se explica como instalar la aplicación en un dispositivo móvil Android.
- **Manual de usuario:** En este apartado se explica como usar la aplicación para un usuario nuevo.
- **Documentos de diseño intermedios:** Estos documentos son los que se han hecho antes del documento final que aparece en su respectiva sección de la sección de diseño. Aquí se incluyen apéndices del documento de requisitos, casos de uso, diagramas de clase, diagramas de secuencia y diagramas de estado.

2

Tecnologías y metodologías de trabajo usadas

2.1. Tecnologías usadas

En esta sección se describen las tecnologías que han sido usadas para la realización de este trabajo de fin de grado.

2.1.1. Flutter

Flutter es un marco de desarrollo de código abierto que nació inicialmente para dispositivos móviles que tiene como objetivo el hacer interfaces adaptables a varios formatos de pantalla y sistemas operativos y manejar su estado de manera sencilla. [1]

Se ha elegido este marco de desarrollo para facilitar la compatibilidad con más plataformas en un futuro y por su facilidad para hacer interfaces de usuario con aspectos modernos.

2.1.2. Dart

Dart es un lenguaje de programación orientado a objetos optimizado para crear interfaces de usuario. Dart usa un recolector de basura y usa un tipado estático de datos, es decir, se asegura de que las variables siempre tienen el tipo estático definido. Además cuenta con una función llamada *"Sound null safety"* que hace que una variable no pueda tener un valor de tipo *null* a no ser que se especifique explícitamente. [2]

Se ha usado Dart debido a que es el lenguaje de programación nativo para hacer una aplicación con [Flutter](#).

2.1.3. Supabase

Supabase es un “*backend as a service*” de código abierto que intenta ser una alternativa a Firebase. Ofrece una API para interaccionar con todas las funciones de las que dispone, entre las que se encuentran, una base de datos relacional [Postgres](#), gestión de la autenticación, almacenamiento y funciones en tiempo real. Además ofrece varias librerías para desarrollar en diferentes marcos de desarrollo entre los que se encuentra [Flutter](#). [3]

Se ha elegido usar Supabase por su facilidad para crear proyectos y la extensa documentación que ofrece. También cuenta con la ventaja de usar una base de datos relacional con la que se tiene una mayor familiaridad que con bases de datos NoSQL [5] como la que usa Firebase. Otra ventaja con la que cuenta Supabase es que no es necesario introducir un método de pago para usar todas sus funciones.[3, 4]

2.1.4. Postgres

Postgres es una base de datos relacional de código abierto. [6]

Se ha usado esta base de datos porque es la que usa [Supabase](#).

2.1.5. PL/SQL

PL/SQL es un lenguaje de procedimientos diseñado para trabajar con sentencias SQL. Con ello se pueden hacer funciones en la base de datos que se ejecuten cuando ocurra un evento o llamarlas externamente para que se ejecuten. [7]

Se ha usado PL/SQL porque se tiene experiencia con él y porque es uno de los lenguajes que soporta [Supabase](#) y [Postgres](#).

2.1.6. Git

Git es un controlador de versiones gratuito y de código abierto. [8]

Se ha usado Git para llevar el control de las versiones durante el desarrollo de la aplicación. Junto a Git se ha usado el repositorio remoto Github [9].

2.1.7. Visual studio code

Visual Studio Code es un editor de código multiplataforma y ligero que soporta diversos lenguajes de programación y auto completado de código. Además soporta extensiones de forma que si un lenguaje no es soportado por defecto, esta funcionalidad puede ser añadida con una extensión. [10]

Se ha usado este editor durante el desarrollo del código por su sencillez y posibilidades de ser personalizado. En este proyecto ha sido necesario usar una extensión para tener soporte para el lenguaje **Dart** y **Flutter**

2.1.8. Android Studio

Android Studio es el entorno de desarrollo integrado oficial para desarrollar aplicaciones para Android. [11]

En este proyecto no se ha usado esta herramienta para el desarrollo del código, sin embargo se ha usado su herramienta de dispositivos virtuales [12] para ir compilando el proyecto desde **Visual studio code** y probandolo en una máquina virtual Android.

2.1.9. Modelio

Modelio es una solución para el diseño de arquitecturas software. Soporta el estándar UML2, es multiplataforma y de código abierto. [13]

Se ha usado esta solución debido a que es gratuita y es posible diseñar todo lo necesario para el proyecto. En fases más avanzadas del proyecto se consiguió una licencia de **Visual paradigm** de parte de la UMA y la versión final de los diagramas de clase están hechas con esta herramienta.

2.1.10. Visual paradigm

Visual paradigm es un paquete de herramientas para el diseño y análisis de software. [14]

Se ha usado este programa para los últimos diseños de diagramas de clase debido a que se consiguió de forma tardía en el progreso del proyecto.

2.1.11. Overleaf

Overleaf es un editor de texto colaborativo en LaTex. [15, 16]

Se ha usado este editor para redactar el documento de requisitos y la especificación de casos de uso para mantener el formato con el documento de la memoria.

2.1.12. Krita

Krita es un programa de dibujo gratuito y de código abierto. [17]

Se ha usado esta herramienta para hacer las maquetas de software iniciales.

2.1.13. Pencil project

Pencil es un programa gratuito y de código abierto diseñado para crear maquetas de software para programas tanto de escritorio como para dispositivos móviles. [18]

Se ha usado esta herramienta para hacer las maquetas finales de la aplicación a desarrollar y tener un diseño más estandarizado.

2.1.14. Trello

Trello es un software web para la gestión del trabajo. Permite usar tablas y listas con tareas.

Se ha usado en este proyecto durante la fase de desarrollo para gestionar las tareas a realizar. [19]

2.1.15. Oracle datamodeler

Datamodeler es una herramienta gráfica de diseño de modelos de datos lógicos y relacionales perteneciente a Oracle [22].

Se ha usado esta herramienta para realizar el diagrama entidad-relación de la base de datos.

2.2. Metodología de trabajo

El proyecto se ha llevado a cabo en dos fases distintas cada una con una forma de trabajar.

La primera fase, de diseño, ha consistido en crear toda la documentación previa de diseño del proyecto. En esta primera fase se han ido generando los documentos y recogiendo la opinión del cliente con cada documento generado. Antes de pasar a un nuevo documento se han hecho todo los cambios solicitados por el cliente hasta su validación. No ha habido un límite de tiempo para entregar cada documento, cada vez que se termina un documento se ha entregado y empezado con el siguiente si no ha habido ningún inconveniente.

En la segunda fase, el desarrollo, se ha seguido una metodología ágil basada en Scrum. Una metodología ágil es un enfoque de gestión de proyectos basada en iteraciones [20]. Scrum es una metodología ágil que divide el trabajo en "Sprints" en los que se tienen distintas tareas a realizar. Suelen durar alrededor de dos semanas y las tareas a realizar se escogen de un "backlog" que son todas las tareas que se han establecido que hay que realizar para que el proyecto esté finalizado. Durante los sprints se suelen hacer reuniones cortas para conocer el desarrollo de este y al final se hace una reunión más larga y se cierra el sprint con las tareas que hayan sido hechas. [21]

En este proyecto se han realizado sprints de dos semanas donde las tareas de cada uno se han seleccionado al inicio de este junto al cliente. Al final de cada sprint se ha mostrado la funcionalidad realizada para validar que lo implementado es lo pedido y que las tareas están completadas. Para gestionar los sprints se ha usado **Trello**.

3

Diseño

En este apartado se muestran y explican los documentos de diseño finales del proyecto.

3.1. Requisitos

Para generar los requisitos se ha tenido principalmente en cuenta cómo usan los alumnos de la universidad de Málaga el grupo para compartir coche para conseguir que la aplicación resulte intuitiva y fácil de usar para aquellos que ya estuviesen usando el grupo. Junto a esto, se han intentado encontrar nuevas funcionalidades que puedan facilitar o mejorar la experiencia aunque su forma de funcionar sea ligeramente distinta y finalmente se han validado todos los requisitos propuestos con el cliente. Los requisitos mostrados a continuación son los finales, los cambios desde la primera versión son los siguientes:

- Los requisitos relacionados con Google han sido cambiados a Discord. Esto se debe a que durante el desarrollo se ha descubierto que para su uso es necesario introducir datos bancarios y tras consultarlos con el cliente y para evitar problemas se ha cambiado al uso de Discord.
- Los requisitos **RF-7.7. Viajes periódicos** y **RF-7.8. Eliminar usuarios apuntados** han sido añadidos posteriormente por petición del cliente.
- En **RF-7. Crear oferta de viaje** se ha añadido a la descripción que se puede seleccionar una fecha junto a una hora y que opcionalmente se puede añadir un título y descripción.
- En **RF-7.2. Borrar oferta de viaje** se ha cambiado la descripción a que los usuarios puedan borrar un viaje en cualquier momento y se ha cambiado la localización del botón para acceder a la funcionalidad.

- En el proceso de diseño se ha decidido con el cliente que la descripción de un viaje se vea solo desde el viaje y no desde el chat (**RF-7.5. Dar detalles de viaje**)
- A petición del cliente se ha cambiado el requisito **RF-8. Iniciar chat entre usuarios** para que los usuarios puedan empezar un chat sin estar apuntados al viaje que ofrezca el usuario.
- Se ha arreglado la descripción de **RF-6. Selección de posición**, se usa un radio y no un zoom.

A continuación se listan los requisitos definidos finalmente.

Requisitos funcionales

3.1.1. RF-1. Iniciar sesión

Los usuarios podrán usar la aplicación introduciendo sus credenciales de cuenta. Para ello tendrán que introducir dichas credenciales en los campos de texto correspondientes para correo y contraseña.

3.1.2. RF-1.1. Iniciar sesión con Discord

Los usuarios podrán pulsar un botón e introducir sus credenciales de Discord para usar la aplicación.

3.1.3. RF-2. Registro de usuario

Los usuarios podrán proporcionar un correo y una contraseña para crear un usuario en la aplicación. Para ello tendrán que pulsar en un botón que les llevará a una página donde poner un correo y contraseña. La contraseña se pedirá dos veces.

3.1.4. RF-2.1. Registro automático con Discord

El usuario solo tendrá que poner sus credenciales de Discord como si iniciara sesión y se creará su usuario automáticamente. La primera vez que alguien inicie sesión con Discord habrá que guardar el ID proporcionado en la base de datos para que pueda empezar a gestionar sus viajes.

3.1.5. RF-3. Borrar cuenta de usuario

El usuario podrá borrar su cuenta y todos sus datos desde la pestaña de configuración. No se podrá hacer esto si tiene algún viaje en curso y se preguntará por confirmación antes de borrarlo todo.

3.1.6. RF-4. Cerrar sesión

El usuario podrá pulsar un botón en la interfaz de ajustes para cerrar sesión. Al pulsar dicho botón la aplicación deberá borrar el token de sesión activo del usuario y mandarlo

a la interfaz de inicio de sesión.

3.1.7. RF-4.1. Cerrar sesión automáticamente

Si al comprobar el token se encuentra que este está caducado se cerrará la sesión. Esto puede ocurrir al no abrir la aplicación en mucho tiempo ya que en otro caso se aplica el requisito **RF-5.1. Renovar token automáticamente**.

3.1.8. RF-5. Mantener sesión abierta

Si el usuario inició sesión anteriormente entonces su sesión seguirá abierta mientras no cierre sesión o pase un tiempo determinado. Para ello se deberá guardar el token de sesión en el dispositivo y cada vez que se abra la aplicación será rescatada y se comprobará si todavía es válido.

3.1.9. RF-5.1. Renovar token automáticamente

Sí el usuario inicia sesión automáticamente entonces se renovará también el tiempo de expiración del token.

3.1.10. RF-6. Selección de posición

Se podrá seleccionar en un mapa la posición en la que el usuario quiere que lo recojan o la posición donde un usuario recoge a otro usuario. La posición inicial será la posición geográfica actual del usuario y se tendrá un radio de 100 metros.

3.1.11. RF-6.1. Mirar posición geográfica

El usuario podrá ver su posición geográfica actual en el mapa y la posición de destino del viaje. Esto se mostrará en un mapa en la descripción del viaje una vez seleccionado y al seleccionar una posición (**RF-6. Selección de posición**)

3.1.12. RF-6.2. Selección de zonas concretas

El usuario podrá seleccionar una zona geográfica concreta (como una ciudad o pueblo) en vez de una zona personalizada por el usuario para ser recogido o recoger. Las zonas

geográficas concretas se mostrarán en una lista con las posibilidades que hay y se dará un buscador para facilitar la búsqueda. Algunos ejemplos de zonas serían: Torremolinos, Fuengirola, Teatinos, Málaga Centro, Churriana, Alhaurín de la Torre.

3.1.13. RF-7. Crear oferta de viaje

Todos los usuarios podrán crear una oferta para llevar a otros usuarios en su coche. Esta oferta se plantea del lado del usuario que disponga de un coche y se ofrezca a llevar a otro usuario sin coche. Para ello deberán llenar en qué zona geográfica recogen, a qué hora y en qué fecha comienza el viaje o se llega al destino, hasta qué zona geográfica se ofrecen a transportar y el número de plazas disponibles en su coche. Opcionalmente el usuario podrá dar un título y una descripción al viaje.

3.1.14. RF-7.1. Ver oferta de viaje

Los usuarios podrán ver ofertas de viajes. Podrán ver sus detalles haciendo clic en las ofertas, las hayan aceptado o no.

3.1.15. RF-7.2. Borrar oferta de viaje

El usuario que haya creado una oferta podrá borrarla. Para ello deberá ir a su lista de viajes pendientes, pulsar en el viaje y en la vista de ver viaje ([RF-7.1. Ver oferta de viaje](#)) pulsar en el botón de borrar. Una vez pasado 15 minutos de la hora de comienzo del viaje la oferta no será visible en la búsqueda de ofertas ([RF-7.3. Editar oferta de viaje](#)).

3.1.16. RF-7.3. Editar oferta de viaje

El usuario que haya creado una oferta podrá editar información de esta antes de que comience el viaje. Para ello deberá ir a su lista de viajes pendientes y pulsar en un botón junto a la oferta para editar. Al pulsar se abrirá una nueva pantalla donde se podrá cambiar la descripción, el número de asientos de coche disponible (No puede ser menor que el número de personas que ya han aceptado la oferta), la hora del viaje y la zona de recogida y destino.

3.1.17. RF-7.4. Buscar oferta de viaje

Los usuarios podrán buscar ofertas de viaje por zonas de origen, destino y horas. Este se ofrecerá en forma de filtro en la interfaz de búsqueda de viajes. El usuario podrá pulsar un botón que le enseñará todos los filtros disponibles que aplicar ([RF-9. Aplicar filtro de búsqueda de viajes](#)).

3.1.18. RF-7.5. Dar detalles de viaje

Los usuarios podrán añadir opcionalmente información extra como el modelo de coche, el color, o una descripción para que sea más fácil identificar el coche o persona. Esta descripción será accesible desde la interfaz de un viaje y se podrá establecer detalles por defecto desde la página de usuario.

3.1.19. RF-7.6. Aceptar viaje

Los usuarios podrán aceptar una oferta de viaje disponible. Se aceptarán pulsando un botón desde la vista del viaje y una vez aceptado el número de plazas disponibles del viaje seleccionado disminuirá.

3.1.20. RF-7.7. Viajes periódicos

Los usuarios que ofrezcan viajes podrán establecer un viaje como periódico. Esto podrán hacerlo en la creación de la oferta de viaje ([RF-7. Crear oferta de viaje](#)) y una vez seleccionen que el viaje es periódico este se publicará automáticamente cada semana hasta que el usuario creador lo cancele .

3.1.21. RF-7.8. Eliminar usuarios apuntados

El creador de un viaje podrá cancelar la reserva de una persona que haya reservado una plaza. Esto será posible hacerlo desde la vista de un viaje.

3.1.22. RF-8. Iniciar chat entre usuarios

Los usuarios que ofertan un viaje y los que buscan un viaje podrán hablar por chat para concretar detalles del viaje. Este chat se ofrecerá con un botón en la descripción del

viaje haya sido aceptado o no.

3.1.23. RF-8.1. Ver mensajes de chat

Los usuarios con un chat iniciado podrán ver los mensajes de chat haciendo clic en él. Haciendo clic se abrirá una pantalla con todos los mensajes del chat.

3.1.24. RF-8.2. Enviar mensajes

Los usuarios que hayan iniciado un chat podrán enviar mensajes entre ellos desde la pantalla de chat. Esta pantalla tendrá una caja de texto donde escribir el mensaje, un botón para mandar el mensaje y todos los mensajes enviados por ambos usuarios hasta el momento.

3.1.25. RF-8.3. Recibir mensajes

Un usuario en un chat podrá recibir mensajes del otro usuario y estos se mostrarán en el historial de mensajes en la pantalla de chat. Los mensajes se recibirán una vez se ha abierto la aplicación. Si un mensaje es enviado por el otro usuario ([RF-8.1. Ver mensajes de chat](#)) y no se está en la aplicación ese mensaje se verá una vez se entre. Esto será así durante un plazo de un mes, una vez pasado este plazo el mensaje se borrará.

3.1.26. RF-8.4. Borrado de mensajes automático

Los mensajes de chat enviados se borrarán automáticamente de la base de datos después de un mes. Si los mensajes están guardados en el dispositivo, estos todavía se podrán ver pero una vez borrados de aquí ya no serán recuperables.

3.1.27. RF-9. Aplicar filtro de búsqueda de viajes

En la pantalla de búsqueda de viajes habrá un botón que al pulsar llevará a una pantalla donde se podrán aplicar filtros para ver los viajes disponibles.

3.1.28. RF-9.1. Filtrar por hora

En la pantalla de filtro se podrá establecer una hora de comienzo del viaje, de fin del viaje y un número de minutos de margen. Este número de minutos de margen por defecto estará puesto en 15 y lo que indica es que si un viaje comienza a las 13:30, se muestren los viajes que empiezan entre las 13:15 y a las 13:45.

3.1.29. RF-9.2. Filtrar por posición

El usuario podrá seleccionar una zona desde la que salen viajes o a la que llegan viajes para que solo salgan esos. Para ello habrá un mapa para elegir la posición de salida y uno para la de llegada ([RF-6. Selección de posición](#))

3.1.30. RF-9.3. Eliminar filtro

Una vez un filtro se ha aplicado será posible eliminar filtros individuales o quitar todos los filtros a la vez pulsando un botón.

3.1.31. RF-10. Cambiar tema de color

Los usuarios podrán cambiar entre un “modo claro” de colores y uno “oscuro” en la aplicación. Para hacerlo deberá ir a la interfaz de ajustes y pulsar un botón. El “modo claro” consistirá en usar como tema de color de la aplicación principalmente colores claros como el blanco, azul celeste y un color de resalte más oscuro como el morado. El “modo oscuro” usará colores más oscuros como el negro y el azul marino.

Requisitos no funcionales

3.1.32. RNF-1. Registro de usuario con Discord sencillo

Registrarse con Discord debe ser fácil. El usuario debe poder ver esta opción nada más abrir la aplicación. Para lograr esto, lo primero que se debe ver al iniciar la aplicación es la pantalla de inicio de sesión con Discord indicando que con una cuenta de Discord es posible directamente usar la app y una vez puestas las credenciales no debe pedirse nada extra para que el usuario puede empezar a usar ya la aplicación.

3.1.33. RNF-2. Intuitividad

La aplicación debe ser intuitiva y fácil de usar, sobre todo para usuarios que vengan del grupo de whatsapp. Ninguna opción debe estar a más de cuatro menús, botones o pulsaciones de distancia del menú de inicio.

3.1.34. RNF-3. Eficiencia

La aplicación no deberá tardar en cargar ninguna página más de 10 segundos.

3.1.35. RNF-4. Fluidez

La navegación por las distintas páginas debe ser fluida. La tasa de refresco de las animaciones debe ser igual a la tasa de refresco del dispositivo para lograr esto.

3.1.36. RNF-5. Interfaz agradable a la vista

La aplicación tendrá colores agradables para la vista y una opción para usar un “modo claro” o un “modo oscuro”. Se usará un modelo de color con tres colores principales para cada uno. De ser necesario se usarán hasta 3 colores extra puntualmente.

3.1.37. RNF-6. Fácil adaptación del inicio de sesión a nuevos procedimientos

El diseño debe estar hecho de forma que sea fácil añadir nuevos métodos para iniciar sesión. Para ello los nuevos métodos se podrán debajo de los anteriores y se podrá deslizar

la página para verlos. Además estos métodos nuevos implementarán una interfaz definida que todos los métodos de inicio de sesión seguirán.

3.1.38. RNF-7. Modularidad

El diseño debe ser lo más modular posible para facilitar la reutilización de componentes. Los botones y otros elementos recurrentes se harán aparte para facilitar su reutilización. Elementos a reutilizar:

- Botón de editar
- Botón de eliminar
- Botón de aceptar
- Widget de selección de posición en mapa
- Widget de lista de viaje

3.1.39. RNF-8. Seguridad

Las contraseñas de los usuarios se guardarán cifradas de manera segura.

3.2. Casos de uso

A continuación, a partir de los requisitos se ha generado un diagrama de casos de uso y la especificación final para cada uno de los casos de uso. Cuando ha habido un cambio en los requisitos se han cambiado los casos de uso para mantener la coherencia. Para ver los diagramas de casos de uso anteriores, acuda al apéndice C, **Diagramas de casos de uso anteriores**

3.2.1. Diagrama

A continuación, en **Diagrama de casos de uso del proyecto** se muestra el diagrama realizado con la herramienta **Modelio** y en las siguientes secciones se muestran las especificaciones de los casos del diagrama.

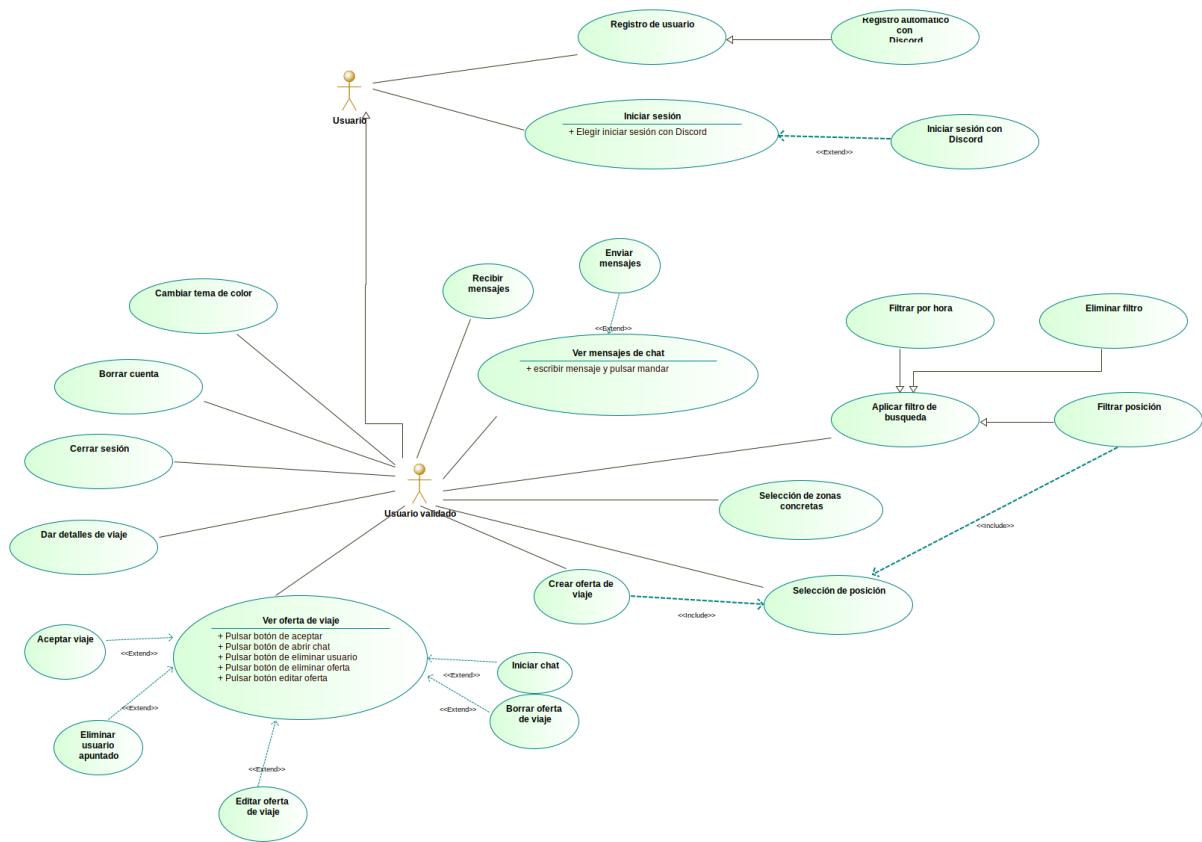


Figura 1: Diagrama de casos de uso del proyecto

3.2.2. Iniciar sesión

Descripción

Los usuarios pueden iniciar sesión en la aplicación poniendo un correo electrónico y una contraseña.

Precondición

El usuario debe tener instalada la aplicación en su teléfono móvil

Postcondición

El usuario se encuentra en la pantalla de inicio de la aplicación.

Escenario principal

1. El usuario abre la aplicación.
2. La aplicación muestra la pantalla de inicio de sesión
3. Extensión: El usuario elige iniciar sesión con Discord ([Iniciar sesión con Discord](#))
4. El usuario introduce correo electrónico y contraseña
5. El usuario pulsa aceptar
6. La aplicación manda las credenciales al back-end
7. El back-end valida las credenciales
8. Las credenciales son correctas y la aplicación muestra la pantalla de inicio de la aplicación.

Escenario alternativo

- 6B. Las claves de acceso introducidas no son correctas
- 7B. La aplicación muestra un mensaje de error
- 8B. El usuario puede volver a introducir credenciales de acceso.

3.2.3. Iniciar sesión con Discord

Descripción

Punto de extensión del caso de uso de iniciar sesión. Añade pasos específicos para el inicio de sesión con Discord.

Precondición

El usuario está en la pantalla de inicio de sesión.

Postcondición

El usuario ha sido devuelto a la aplicación

Escenario principal

1. El usuario hace clic en el botón para iniciar sesión con Discord.
2. El sistema abre el navegador o la aplicación de Discord para que el usuario pueda introducir sus credenciales

3.2.4. Registro de usuario

Descripción

El usuario se crea una cuenta en la aplicación. Para ello se introduce un correo electrónico y una contraseña (La contraseña es introducida dos veces).

Precondición

El usuario debe haber iniciado la aplicación.

Postcondición

El usuario tiene una cuenta creada.

Escenario principal

1. El usuario abre la aplicación.
2. La aplicación muestra la pantalla de inicio de sesión
3. El usuario hace clic en Regístrate
4. La aplicación muestra una pantalla donde introducir un correo electrónico y un campo para escribir la contraseña y otro para repetirla
5. El usuario rellena los campos solicitados y pulsa enviar
6. La aplicación crea un usuario con los datos recibidos y manda un correo de confirmación
7. La aplicación devuelve al usuario a la pantalla de inicio de sesión

Escenario alternativo

- 6B. La contraseña introducida por el usuario no es la misma que la contraseña introducida en el campo para repetirla.
- 7B. La aplicación marca el error y deja que el usuario modifique los campos.

- 6C. El correo electrónico introducido ya existe en la base de datos
- 7C. La aplicación muestra un mensaje informando de esto y deja que el usuario cambie el campo.

3.2.5. Registro automático con Discord

Descripción

El usuario tiene una cuenta de Discord y decide directamente iniciar sesión con ella

Precondición

El usuario debe haber iniciado la aplicación y tener una cuenta de Discord.

Postcondición

El usuario tiene una cuenta creada y accede a la aplicación.

Escenario principal

1. El usuario abre la aplicación.
2. La aplicación muestra la pantalla de inicio de sesión
3. El usuario hace clic en iniciar sesión con Discord
4. El usuario rellena los campos solicitados y pulsa enviar
5. La aplicación crea un usuario con los datos recibidos
6. La aplicación muestra al usuario la pantalla inicial de la aplicación

3.2.6. Borrar cuenta

Descripción

El usuario validado borra su cuenta creada en la aplicación.

Precondición

El usuario debe tener una cuenta creada y debe estar en una sesión.

Postcondición

El usuario deja de tener una cuenta en la aplicación.

Escenario principal

1. El usuario va a la pantalla de configuración
2. La aplicación muestra las opciones de configuración
3. El usuario hace clic en “Borrar mi cuenta definitivamente”
4. La aplicación muestra un mensaje de confirmación
5. El usuario acepta el mensaje
6. La aplicación borra la cuenta del usuario y lo lleva a la pantalla de inicio de sesión

Escenario alternativo

- 5B. El usuario no acepta el mensaje de borrado
- 6B. La aplicación cancela el proceso de borrado
- 7B. El usuario vuelve a pulsar el botón y acepta el mensaje
- 8B. La aplicación borra la cuenta del usuario

3.2.7. Cerrar sesión

Descripción

El usuario cierra su sesión activa en la aplicación.

Precondición

El usuario debe tener una sesión abierta en la aplicación.

Postcondición

El usuario no tiene una sesión abierta y esta en la pantalla de inicio de sesión

Escenario principal

1. El usuario va a la pantalla de configuración
2. La aplicación muestra las opciones de configuración
3. El usuario pulsa en cerrar sesión
4. La aplicación cierra la sesión y muestra al usuario la pantalla de inicio de sesión

3.2.8. Ver oferta de viaje

Descripción

El usuario ve los datos de una oferta de viaje.

Precondición

Existe una oferta.

Postcondición

El usuario puede ver los datos de la oferta

Escenario principal

1. El usuario hace clic en una oferta
2. La aplicación muestra la pantalla del viaje
3. La aplicación muestra un mapa con la posición del usuario y la ruta del viaje, las horas, las plazas, el usuario que lo ofrece, el título y la descripción si la tiene
4. Extensión: El usuario pulsa el botón de aceptar viaje ([Aceptar viaje](#))
5. Extensión: El usuario pulsa el botón de abrir chat ([Iniciar chat entre usuarios](#))
6. Extensión: El usuario pulsa el botón de eliminar usuario ([Eliminar usuarios apuntados](#))
7. Extensión: El usuario pulsa el botón de eliminar oferta ([Borrar oferta de viaje](#))
8. Extensión: El usuario pulsa el botón de editar oferta ([Editar oferta de viaje](#))

3.2.9. Crear oferta de viaje

Descripción

El usuario crea una nueva oferta de viaje que otros usuarios pueden aceptar.

Precondición

Tener una sesión abierta.

Postcondición

La aplicación ha creado una nueva oferta en el sistema.

Escenario principal

1. El usuario se va a la pantalla de ofertas de viaje y pulsa el botón de crear oferta
2. La aplicación muestra una pantalla donde se puede introducir la fecha y hora de llegada al destino o de salida, el destino del viaje, la zona de salida, el número de plazas disponibles, un título y una descripción.
3. El usuario introduce la fecha y hora y corresponda
4. **Selección de posición**
5. El usuario introduce el número de plazas
6. El usuario pulsa en aceptar
7. La aplicación crea el viaje con los datos dados

Escenario alternativo

- 3B. El usuario rellena posición y plazas y pulsa aceptar
- 4B. La aplicación muestra un mensaje de error y deja que el usuario modifique los campos
- 5B. El usuario rellena la fecha y hora de llegada y pulsa aceptar
- 6B. La aplicación crea el viaje

3C. El usuario rellena los datos y selecciona que el viaje sea periódico.

4C. La aplicación crea un viaje periódico.

3.2.10. Editar oferta de viaje

Descripción

El usuario cambia información de una oferta que haya creado previamente.

Precondición

El usuario ha creado una oferta anteriormente y todavía es válida.

Postcondición

El usuario ha podido modificar los datos de su oferta o dejarlos igual.

Escenario principal

1. El usuario hace clic en el botón de modificar
2. La aplicación muestra una pantalla con los datos previos del viaje en campos que se pueden modificar
3. El usuario cambia el valor de cualquiera de los campos a otro válido
4. El usuario pulsa en aceptar
5. La aplicación actualiza los datos

Escenario alternativo

- 3B. El usuario establece un número de plazas invalido o una hora invalida
 - 4B. El usuario pulsa aceptar
 - 5B. La aplicación muestra un mensaje de error y deja que el usuario modifique los campos inválidos
 - 6B. El usuario pone valores válidos y pulsa aceptar
 - 7B. La aplicación actualiza los datos
-
- 3C. El usuario no cambia ninguno dato y pulsa aceptar.
 - 4C. La aplicación no actualiza los datos y cierra la pantalla de edición

3.2.11. Borrar oferta de viaje

Descripción

El usuario borra uno de sus viajes creados.

Precondición

El viaje no debe haber comenzado.

Postcondición

El viaje ya no se encuentra en el sistema.

Escenario principal

1. El usuario pulsa en el botón de borrar
2. La aplicación muestra un mensaje de confirmación para borrarlo
3. El usuario pulsa aceptar.
4. La aplicación borra el viaje

Escenario alternativo

- 4B. El usuario pulsa cancelar
- 5B. La aplicación cierra el mensaje y no borra el viaje
- 6B. El usuario vuelve a hacer clic en el botón y pulsa aceptar en el mensaje
- 7B. La aplicación borra el viaje

3.2.12. Aceptar viaje

Descripción

El usuario acepta un viaje de otro usuario que lo ha creado.

Precondición

Hay viajes disponibles y las plazas disponibles son mayores que uno.

Postcondición

El número de plazas del viaje a disminuido en uno y el usuario ahora esta relacionado con el viaje.

Escenario principal

1. El usuario hace clic en el botón de aceptar viaje.
2. La aplicación disminuye en uno el número de plazas y asigna al usuario al viaje

3.2.13. Dar detalles de viaje

Descripción

EL usuario añade información extra (una descripción) para un viaje o en general para todos sus viajes

Precondición

El usuario tiene un viaje creado donde añadir la información.

Postcondición

La descripción es añadida.

Escenario principal

1. El usuario va a la pantalla de configuración
2. La aplicación muestra las opciones disponibles
3. El usuario va al cuadro de texto de descripción y escribe algo y pulsa aceptar.
4. La aplicación guarda esa información y la muestra en los viajes del usuario.

Escenario alternativo

- 3B. El usuario no escribe nada en la caja de texto y pulsa aceptar
- 4B. La aplicación no hace nada
- 5B. El usuario escribe algo y pulsa aceptar
- 6B. La aplicación guarda la información

3.2.14. Eliminar usuarios apuntados

Descripción

El usuario creador de un viaje puede desde la vista de un viaje cancelar la reserva de otro usuario que haya reservado su viaje.

Precondición

El usuario tiene un viaje creado y otro usuario ha reservado una plaza.

Postcondición

El viaje tiene una plaza disponible extra y hay un usuario menos apuntado.

Escenario principal

1. El usuario pulsa un botón para eliminar uno de los usuarios que ha reservado
2. La aplicación elimina al usuario del viaje

3.2.15. Selección de zonas concretas

Descripción

El usuario hace clic en un desplegable y selecciona una zona geográfica predefinida

Precondición

El usuario está en una de las pantallas con un mapa.

Postcondición

El usuario ha seleccionado una posición.

Escenario principal

1. La aplicación muestra un desplegable
2. El usuario abre el desplegable de zonas y selecciona una opción
3. El usuario pulsa aceptar
4. La aplicación guarda la posición seleccionada

Escenario alternativo

- 2B. El usuario pulsa el desplegable pero no selecciona nada y pulsa aceptar
- 3B. La aplicación muestra un mensaje de error y deja que el usuario introduzca una posición
- 4B. El usuario selecciona una posición y pulsa aceptar
- 5B. La aplicación guarda los datos

3.2.16. Selección de posición

Descripción

El usuario puede seleccionar una posición en un mapa.

Precondición

El usuario está en una de las pantallas con un mapa.

Postcondición

El usuario ha seleccionado una posición.

Escenario principal

1. La aplicación muestra un mapa
2. El usuario toca una posición en el mapa
3. El usuario selecciona un radio de rango para esa posición
4. El usuario pulsa aceptar

Escenario alternativo

- 3B. El usuario deja el radio por defecto

3.2.17. Aplicar filtro de búsqueda

Descripción

El usuario aplica un filtro para ver los viajes disponibles.

Precondición

El usuario esta en la pantalla de viajes.

Postcondición

Se están mostrando los viajes siguiendo el filtro aplicado

Escenario principal

1. El usuario pulsa el botón de filtros
2. La aplicación muestra la pantalla de filtros
3. El usuario selecciona un filtro o ninguno y pulsa el botón “Aplicar filtros”
4. La aplicación cierra la pantalla y muestra los viajes siguiendo el filtro aplicado.

Escenario alternativo

- 3B. El usuario pulsa en cancelar
- 4B. La aplicación cierra la pantalla de filtros y no aplica los filtros específicos que se hayan marcado

3.2.18. Filtrar por posición

Descripción

El usuario aplica un filtro posición

Precondición

El usuario esta en la pantalla de viajes.

Postcondición

Se están mostrando los viajes siguiendo un filtro de posición

Escenario principal

1. El usuario pulsa el botón de filtros
2. La aplicación muestra la pantalla de filtros
3. **Selección de posición**
4. El usuario pulsa el botón “Aplicar filtros”
5. La aplicación cierra la pantalla y muestra los viajes siguiendo el filtro aplicado.

3.2.19. Filtrar por hora

Descripción

El usuario aplica un filtro para que solo salgan viajes respecto de una hora indicada

Precondición

El usuario esta en la pantalla de viajes.

Postcondición

Se están mostrando los viajes siguiendo un filtro de hora

Escenario principal

1. El usuario pulsa el botón de filtros
2. La aplicación muestra la pantalla de filtros
3. El usuario selecciona una hora de llegada para el viaje y pulsa el botón “Aplicar filtros”
4. La aplicación cierra la pantalla y muestra los viajes siguiendo el filtro aplicado.

3.2.20. Eliminar filtro

Descripción

El usuario elimina todos los filtros aplicados.

Precondición

El usuario esta en la pantalla de viajes.

Postcondición

Se están mostrando los viajes siguiendo el filtro aplicado

Escenario principal

1. El usuario pulsa el botón de filtros
2. La aplicación muestra la pantalla de filtros
3. El usuario pulsa el botón de “Eliminar filtros”
4. La aplicación elimina los filtros aplicados
5. El usuario cierra la pantalla de filtros

Escenario alternativo

- 4B. No hay filtros aplicados y la aplicación no hace nada
- 5B. El usuario cierra la pantalla de filtros

3.2.21. Iniciar chat entre usuarios

Descripción

Un usuario va a la pantalla de un viaje e inicia un chat con el usuario que lo ofrece.

Precondición

Existe un viaje disponible por el que se puede iniciar el chat.

Postcondición

Existe un chat entre los dos usuarios.

Escenario principal

1. El usuario hace clic en el botón de abrir chat
2. La aplicación crea un chat entre ambos usuarios y muestra la pantalla de chat

3.2.22. Ver mensajes de chat

Descripción

Un usuario va a la pantalla de chats y abre uno para ver los mensajes.

Precondición

Existe un chat creado entre dos usuarios.

Postcondición

El usuario ha visto los mensajes del chat.

Escenario principal

1. El usuario va a la pantalla de chats
2. La aplicación muestra una lista con los chats creados
3. El usuario hace clic en uno de los viajes
4. La aplicación muestra los mensajes que tenga el chat
5. Extensión: Usuario escribe y manda un mensaje ([Enviar mensajes](#))

3.2.23. Enviar mensajes

Descripción

El usuario podrá mandar mensajes por un chat que haya iniciado con otro usuario.

Precondición

El usuario tiene que haber creado un chat

Postcondición

Un mensaje ha sido enviado desde el dispositivo de un usuario a otro

Escenario principal

1. La aplicación muestra la pantalla de chat con un campo donde escribir y un botón de enviar
2. El usuario escribe un mensaje en el campo proporcionado
3. El usuario pulsa el botón de enviar
4. La aplicación manda el mensaje y lo muestra en la pantalla de chat

Escenario alternativo

- 3B. El usuario no escribe nada y pulsa enviar
- 4B. La aplicación muestra un error que indica que hace falta escribir algo
- 5B. El usuario escribe algo y pulsa enviar
- 6B. La aplicación manda el mensaje

3.2.24. Recibir mensajes

Descripción

Un usuario que tenga un chat iniciado con otro recibe mensajes de este.

Precondición

El usuario debe tener un chat iniciado con otro y el otro debe haber mandado un mensaje.

Postcondición

Se ha recibido un mensaje del otro usuario.

Escenario principal

1. El usuario abre la aplicación
2. La aplicación se abre y mira si hay mensajes nuevos
3. La aplicación pone los nuevos mensajes para el usuario
4. El usuario abre el chat y mira los mensajes recibidos

Escenario alternativo

- 3B. La aplicación no encuentra mensajes nuevos al abrir la aplicación
- 4B. Mientras el usuario usa la aplicación otro usuario manda un mensaje
- 5B. La aplicación actualiza la caja de chat y muestra el mensaje al usuario
- 6B. El usuario abre la caja de chat y mira el mensaje

3.2.25. Cambiar tema de color

Descripción

El usuario pulsa un botón y cambia el tema de color al otro que tenga. Si está en claro pasa a oscuro y si está en oscuro pasa a claro

Precondición

No hay precondición para que un usuario validado realice este caso de uso

Postcondición

El tema de color ha cambiado.

Escenario principal

1. El usuario hace clic en el botón de configuración
2. La aplicación muestra la pantalla de configuración
3. EL usuario hace clic en el botón de cambio de tema
4. La aplicación cambia el tema de color

3.3. Diagrama de base de datos

Para satisfacer los requisitos y los casos de uso anteriormente expuestos, se ha diseñado el siguiente diagrama entidad-relación con **Oracle datamodeler**. Todos los tipos de los atributos son tipos de **Postgres**. Este es el diagrama final, para ver las versiones anteriores acuda al apéndice D, **Diagramas de bases de datos anteriores**

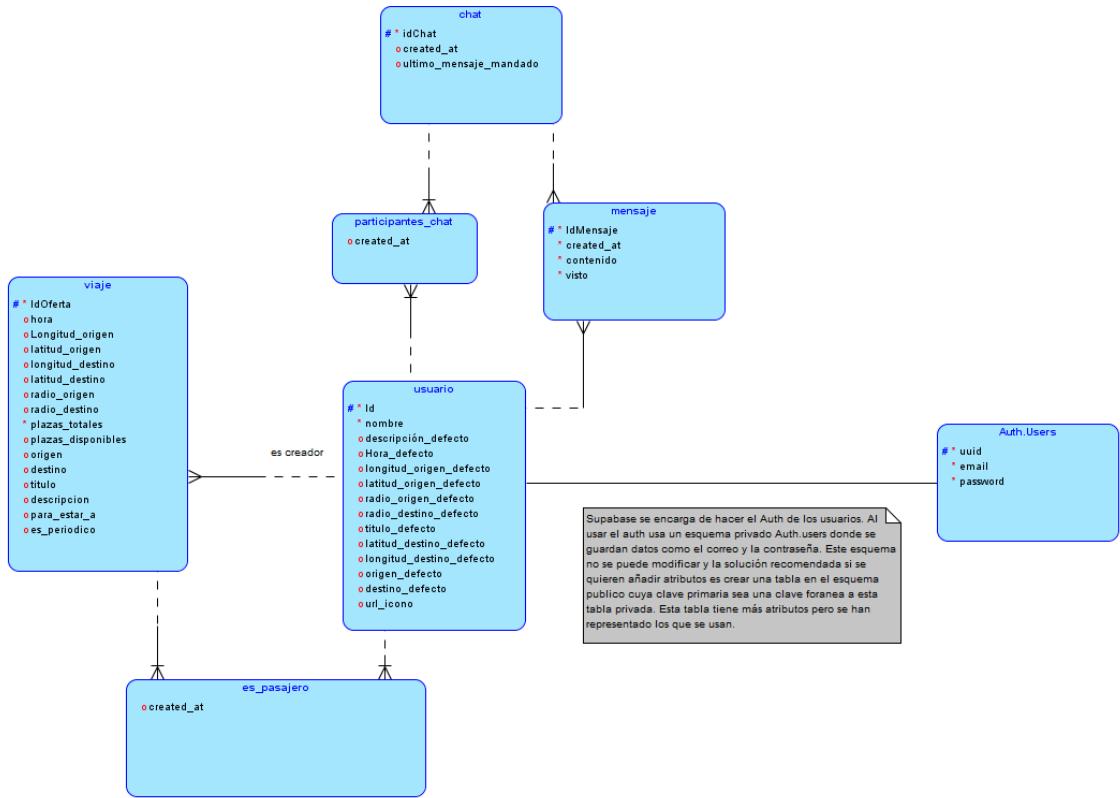


Figura 2: Diagrama entidad relación de la base de datos

3.3.1. Tabla viaje

Esta tabla representa un viaje u oferta que un usuario ha creado. Tiene los siguientes atributos:

- **idOferta**: Identificador único de la tabla viaje, de tipo int8.
- **hora**: Representa la fecha y hora de comienzo o de llegada de un viaje en base al atributo para_estar_a. Es de tipo timestamptz.

- **longitud_origen:** Cuando se usa una posición personalizada, este atributo almacena la longitud de origen del viaje. Es de tipo float8.
- **latitud_origen:** Cuando se usa una posición personalizada, este atributo almacena la latitud de origen del viaje. Es de tipo float8.
- **longitud_destino:** Cuando se usa una posición personalizada, este atributo almacena la longitud de destino del viaje. Es de tipo float8.
- **latitud_destino:** Cuando se usa una posición personalizada, este atributo almacena la latitud de destino del viaje. Es de tipo float8.
- **radio_origen:** Cuando se usa una posición personalizada, este atributo almacena el radio desde la posición origen seleccionada. Se usa para determinar como de lejos esta el usuario dispuesto a desplazarse para comenzar el viaje y después poder filtrar en base a ello. Es de tipo int8.
- **radio_destino:** Cuando se usa una posición personalizada, este atributo almacena el radio desde la posición destino seleccionada. Se usa para determinar como de lejos esta el usuario dispuesto a desplazarse para llegar a su destino una vez se llega al punto de destino del viaje y después poder filtrar en base a ello. Es de tipo int8.
- **plazas_totales:** Indica el total de plazas disponibles que el usuario que ofrece el viaje quiere dar como disponibles. Es de tipo int2.
- **plazas_disponibles:** Indica las plazas que tiene disponible un viaje. Este atributo empieza con el valor de las plazas totales, disminuye en uno cuando alguien se apunta al viaje y aumenta en uno cuando alguien cancela la reserva. Este atributo puede ser calculado usando la tabla es_pasajero pero se ha hecho de esta forma para tener mayor rendimiento. Es de tipo int2.
- **origen:** Este atributo almacena el nombre de la localización de origen. Cuando el viaje es simple (usando una zona concreta) se almacena dicha zona concreta y cuando es un viaje personalizado se almacena el nombre del municipio o ciudad de la posición seleccionada. Es de tipo text.

- **destino:** Este atributo almacena el nombre de la localización de destino. Cuando el viaje es simple (usando una zona concreta) se almacena dicha zona concreta y cuando es un viaje personalizado se almacena el nombre del municipio o ciudad de la posición seleccionada. Es de tipo text.
- **título:** Guarda un título que el usuario haya establecido para el viaje. En el caso de que no se establezca, la aplicación mostrará el texto "Viaje a destino" donde destino es el valor del atributo destino. Es de tipo text.
- **descripción:** Guarda una descripción que el usuario haya establecido para el viaje. En el caso de que no se establezca, la aplicación mostrará el texto "*Sin descripción*". Es de tipo text.
- **para_estar_a:** Este atributo indica si la hora de un viaje es una hora de salida (empezar el viaje) o una hora de llegada (finalización aproximada). Es de tipo bool.
- **es_periodico:** Este atributo indica si un viaje es periódico. En caso de que tenga el valor "True", el viaje no será borrado automáticamente al terminar y su fecha de comienzo se incrementará en una semana. Es de tipo bool.

3.3.2. Tabla usuario

Esta tabla almacena la información de perfil de un usuario. Tiene los siguientes atributos:

- **id:** Identificador único de un usuario. Este atributo es a su vez una clave foranea a [Tabla Auth.Users](#). Es de tipo uuid.
- **nombre:** Nombre que ha introducido el usuario. En caso de hacer registro con Discord se usa inicialmente el nombre de usuario de Discord. Es de tipo text.
- **descripción_defecto:** Descripción que el usuario ha establecido para los viajes por defecto. El valor de esta variable se pondrá en el campo de descripción al crear un viaje en caso de que no este vacío. Es de tipo text.

- **hora_defecto:** Guarda una hora por defecto para los viajes. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo timestamptz.
- **longitud_origen_defecto:** Valor de longitud de origen que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo float8.
- **latitud_origen_defecto:** Valor de latitud de origen que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo float8.
- **longitud_destino_defecto:** Valor de longitud de destino que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo float8.
- **latitud_destino_defecto:** Valor de latitud de destino que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo float8.
- **radio_origen_defecto:** Valor de radio de origen que el usuario establece por defecto para los viajes. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo int8.
- **radio_destino_defecto:** Valor de radio de destino que el usuario establece por defecto para los viajes. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo int8.
- **titulo_defecto:** Titulo que el usuario ha establecido para los viajes por defecto. El valor de esta variable se pondrá en el campo de titulo al crear un viaje en caso de que no este vacío. Es de tipo text.
- **origen_defecto:** Valor de origen que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo text.

- **destino_defecto:** Valor de destino que el usuario establece por defecto. En la versión actual del programa este atributo no ha sido usado pero se ha dejado para futuras mejoras. Es de tipo text.
- **url_icono:** Este atributo guarda una URL que hace referencia al icono que haya establecido el usuario. Actualmente los usuarios no pueden establecer una imagen de perfil. Para los usuarios que inicien sesión con Discord su imagen de perfil será usada directamente. Es de tipo text.

3.3.3. Tabla chat

En esta tabla se almacenan los datos propios de un chat entre usuarios.

- **idChat:** Identificador único para un chat. Es de tipo int8.
- **creado_en:** Fecha y hora en la que se creo el chat. Es de tipo timestamptz.
- **ultimo_mensaje_mandado:** Este atributo almacena la fecha y hora del último mensaje mandado por este chat. Es usado en la aplicación para ordenar los chats del usuario. Es de tipo timestamptz.

3.3.4. Tabla mensaje

En esta tabla se guardan los mensajes de un chat de dos usuarios. Esta tabla tiene activada la opción de tiempo real de **Supabase** para que cada vez que hay un cambio este se notifique a los usuarios.

- **idMensaje:** Identificador único para un mensaje. Es de tipo int8.
- **created_at:** Fecha y hora en la que se mando el mensaje. Se usa para ordenar los mensajes del chat de un usuario. Es de tipo timestamptz.
- **contenido:** Atributo que almacena el texto del mensaje mandado por el usuario. Es de tipo text.
- **visto:** Este atributo indica si el mensaje ha sido visto por el otro usuario perteneciente al chat. Es de tipo bool.

3.3.5. Tabla Auth.Users

Esta tabla almacena los datos de la cuenta del usuario. Se encuentra en el esquema privado Auth. Esto se debe a que al usar el sistema de autenticación de Supabase los datos se almacenan aquí y no son accesibles desde fuera por seguridad. Es por ello que la solución recomendada por Supabase [23] es lo realizado aquí que es crear una relación uno a uno entre esta tabla y una tabla que haga de perfil ([Tabla usuario](#)).

- **uuid:** Identificador único para un usuario. Es de tipo uuid.
- **email:** Correo electrónico del usuario. En el caso de iniciar sesión con Discord y después crearse una cuenta con el mismo correo, la cuenta será la misma.
- **password:** Contraseña del usuario. En caso de iniciar sesión con Discord este campo no es usado.

3.3.6. Relaciones

- **Tabla es_pasajero:** Esta relación que es representada con una tabla ya que tiene un atributo, representa que usuarios son pasajeros de un viaje. El atributo created_at es de tipo timestampz y es usado para ordenar.
- **Tabla participantes_chat:** Esta relación representada con una tabla guarda la información de que usuarios pertenecen a un chat. El atributo created_at es de tipo timestampz y sirve para ordenar las filas.
- **Relación es_creador:** Esta relación indica que un usuario puede haber creado ningun viaje o varios viajes.
- **Relación uno a muchos entre chat y mensaje:** Esta relación se usa para saber que mensajes pertenecen a que chat, un chat puede no tener mensajes y por ello esa parte de la relación es opcional.
- **Relación uno a muchos entre usuario y mensaje:** Esta relación sirve para saber que mensajes ha escrito que usuario. Un usuario puede no haber escrito ningún mensaje.

- **Relación uno a uno entre usuario y Auth.Users:** Como se ha explicado en [Tabla Auth.Users](#) esta relación uno a uno es la forma recomendada por Supabase de guardar información de los usuarios.

3.3.7. Funciones creadas

Para el funcionamiento de la aplicación se han creado diversas funciones en la base de datos durante el desarrollo. Las funciones se muestran y explican a continuación:

- **deleteUser**

Esta función dado un id de usuario se encarga de borrar el usuario de la base de datos. Esto solo lo puede hacer el propio usuario. La función usa la seguridad por filas de Supabase con la palabra clave *Security definer* [24]. Además para realizar estas funciones se han consultado [25, 26]

```
CREATE or replace function "deleteUser"(idUser uuid)
| returns void
LANGUAGE SQL SECURITY DEFINER
AS $$|
| delete from usuario where usuario.id = idUser;
| delete from auth.users where id = idUser;
$$;
```

Figura 3: Función para borrar un usuario

Se ha encontrado otra forma de hacer esta función sin el parámetro. Inicialmente esa forma no ha funcionado porque no se estaba borrando en cascada en la tabla usuario. Las políticas de seguridad definidas y otras configuraciones pueden verse en [Otras configuraciones en Supabase](#). La función sin parámetro es la que se puede ver a continuación en la [Función para borrar un usuario sin parámetro](#).

```

CREATE or replace function deleteUser()
returns void
LANGUAGE SQL SECURITY DEFINER
AS $$ 
    delete from auth.users where id = auth.uid();
$$;

```

Figura 4: Función para borrar un usuario sin parámetro

▪ Crear nuevo viaje

Esta función crea un nuevo viaje dados todos los parámetros necesarios. La función devuelve el id del viaje creado.

```

create or replace function crear_viaje(hora timestamp, latitud_origen float, longitud_origen float, latitud_destino float, longitud_destino float, plazas_totales smallint, plazas_disponibles smallint, descripcion
text, titulo text, origen text, destino text, radio_origen bigint, radio_destino bigint, para_estar_a boolean, es_periodico boolean) returns bigint as $$ 
declare
    new_viaje_id bigint;
begin
    insert into public.viaje (hora, latitud_origen, longitud_origen, latitud_destino, longitud_destino, plazas_totales, plazas_disponibles, descripcion, titulo, origen, destino, radio_origen, radio_destino,
para_estar_a, es_periodico, creado_por) values (hora, latitud_origen, longitud_origen, latitud_destino, longitud_destino, plazas_totales, plazas_disponibles, descripcion, titulo, origen, destino, radio_origen,
radio_destino, para_estar_a, es_periodico, auth.uid()) returning id into new_viaje_id;

    return new_viaje_id;
end;
$$ language plpgsql security definer;

```

Figura 5: Función para crear un nuevo viaje

▪ Actualizar fecha último mensaje mandado

Esta función actualiza la fecha y hora en la que se ha mandado el último mensaje mandado a través de un chat concreto. Esta función es usada dentro de Triggers.

```

create or replace function public.actualizar_ultimo_mensaje()
returns trigger
language plpgsql
security definer set search_path = public
as $$ 
begin
    update chat set ultimo_mensaje_mandado = current_timestamp where new.chat_id = chat.id;
    return new;
end;
$$;

```

Figura 6: Función para actualizar la fecha del último mensaje mandado del chat

- **Crear nuevo chat**

Esta función recibe el id de otro usuario y con ello primero agrupa los usuarios de la tabla participantes_chat usando el id de chat para saber si el chat con ambos usuarios ya existe. Si no existe se crea y se devuelve el id y si ya existía se devuelve el id encontrado. Esta función se ha conseguido del tutorial [27].

```
create or replace function crear_chat(other_user_id uuid) returns bigint as $$  
declare  
    new_chat_id bigint;  
begin  
    --Aqui si cambio la tabla al diseño nuevo puedo buscar directamente en la tabla chats si el chat existe  
    -- Check if room with both participants already exist  
    with chats_with_usuarios as (  
        select chat_id, array_agg(usuario_id) as participants  
        from participantes_chat  
        group by chat_id  
    )  
    select chat_id  
    into new_chat_id  
    from chats_with_usuarios  
    where crear_chat.other_user_id=any(participants)  
    and auth.uid()=any(participants);  
  
    --Aqui si no existe puedo meter una fila nueva con los dos usuarios y lo demas dejarlo default  
    if not found then  
        -- Create a new room  
        insert into public.chat default values  
        returning id into new_chat_id;  
  
        -- Insert the caller user into the new room  
        insert into public.participantes_chat (usuario_id, chat_id)  
        values (auth.uid(), new_chat_id);  
  
        -- Insert the other_user user into the new room  
        insert into public.participantes_chat (usuario_id, chat_id)  
        values (other_user_id, new_chat_id);  
    end if;  
  
    return new_chat_id;  
end  
$$ language plpgsql security definer;
```

Figura 7: Función para crear un nuevo chat

3.3.8. Triggers

- **Actualizar fecha y hora del último mensaje mandado**

Este trigger se ejecuta cada vez que se inserta una nueva fila en la Tabla mensaje y llama a la función actualizar fecha del último mensaje mandado para que esta variable este siempre actualizada.

```
create trigger on_mensaje_nuevo
  after insert on public.mensaje
  for each row execute procedure public.actualizar_ultimo_mensaje();
```

Figura 8: Trigger para actualizar la fecha del último mensaje mandado

- **Crear nuevo usuario**

Este trigger se ejecuta cada vez que se inserta una nueva fila en la tabla auth.users y se encarga de insertar una nueva fila en la Tabla usuario con el id de la tabla auth.users. Este trigger ha sido conseguido de [23].

```
create function public.handle_new_user()
returns trigger
language plpgsql
security definer set search_path = public
as $$
begin
  insert into public.usuario (id)
  values (new.id);
  return new;
end;
$$;

-- trigger the function every time a user is created
create trigger on_auth_user_created
  after insert on auth.users
  for each row execute procedure public.handle_new_user();
```

Figura 9: Trigger para crear un nuevo usuario

3.3.9. Otras configuraciones en Supabase

▪ Funciones programadas

Para borrar los datos de los usuarios cuando ya no son necesarios y para cumplir los requisitos del proyecto se han programado dos funciones con la extensión de Supabase *Cron*. Cron es una herramienta para programar trabajos y junto a la extensión de Supabase es posible hacer esto mismo en la base de datos [28].

En [Trabajo Cron para borrar viajes y actualizar los periódicos](#) se muestra como cada día a las 2:30AM GMT la base de datos borra los viajes que tengan más de dos días de antigüedad desde que finalizaron y no son viajes periódicos y a los viajes periódicos se les suma una semana a su fecha de inicio siempre que haya pasado al menos un día desde que el viaje terminó por última vez.

En [Trabajo Cron para borrar mensajes](#) se muestra como cada día a las 3:30AM GMT se borran los mensajes que sean más antiguos que treinta días.

```
select cron.schedule (
    'borrar-viajes-actualizar',
    '30 2 * * *', -- Borrar cada dia a las 2:30am (GMT)
    $$ 
        delete from public.viaje where (current_date - date(public.viaje.hora)) >= 2) and not (public.viaje.es_periodico);
        update viaje set hora = hora + interval '7 days' where (current_date - date(hora)) >= 1) and es_periodico;
    $$ 
);
```

Figura 10: Trabajo Cron para borrar viajes y actualizar los periódicos

```
select cron.schedule (
    'borrar-mensajes',
    '30 3 * * *', -- Borrar cada dia a las 3:30am (GMT)
    $$ delete from public.mensaje where current_date - date(public.mensaje.created_at) >= 30 $$ 
);
```

Figura 11: Trabajo Cron para borrar mensajes

- **Políticas de seguridad definidas**

Supabase usa por defecto la seguridad por fila de Postgres por lo que para poder hacer cualquier operación es necesario definir que tipo de usuarios o bajo que condiciones se puede hacer cada operación [24]. Para establecer estas políticas se ha usado la interfaz gráfica que ofrece Supabase. En las figuras siguientes pueden verse las políticas definidas sobre las tablas.

La tabla chat tiene la política de que todos los usuarios autenticados pueden hacer todas las operaciones disponibles.

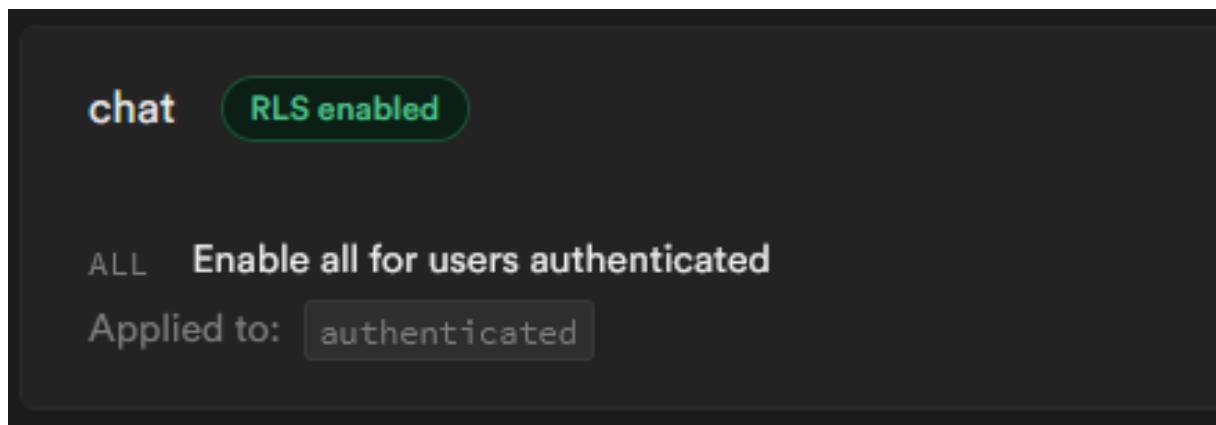


Figura 12: Políticas de la tabla chat

En la tabla viaje solo los usuarios cuyo id coincide con el de la fila pueden borrar y actualizar dicha fila. La operación de inserción de una fila y de seleccionar una fila las pueden hacer todos los usuarios autenticados.

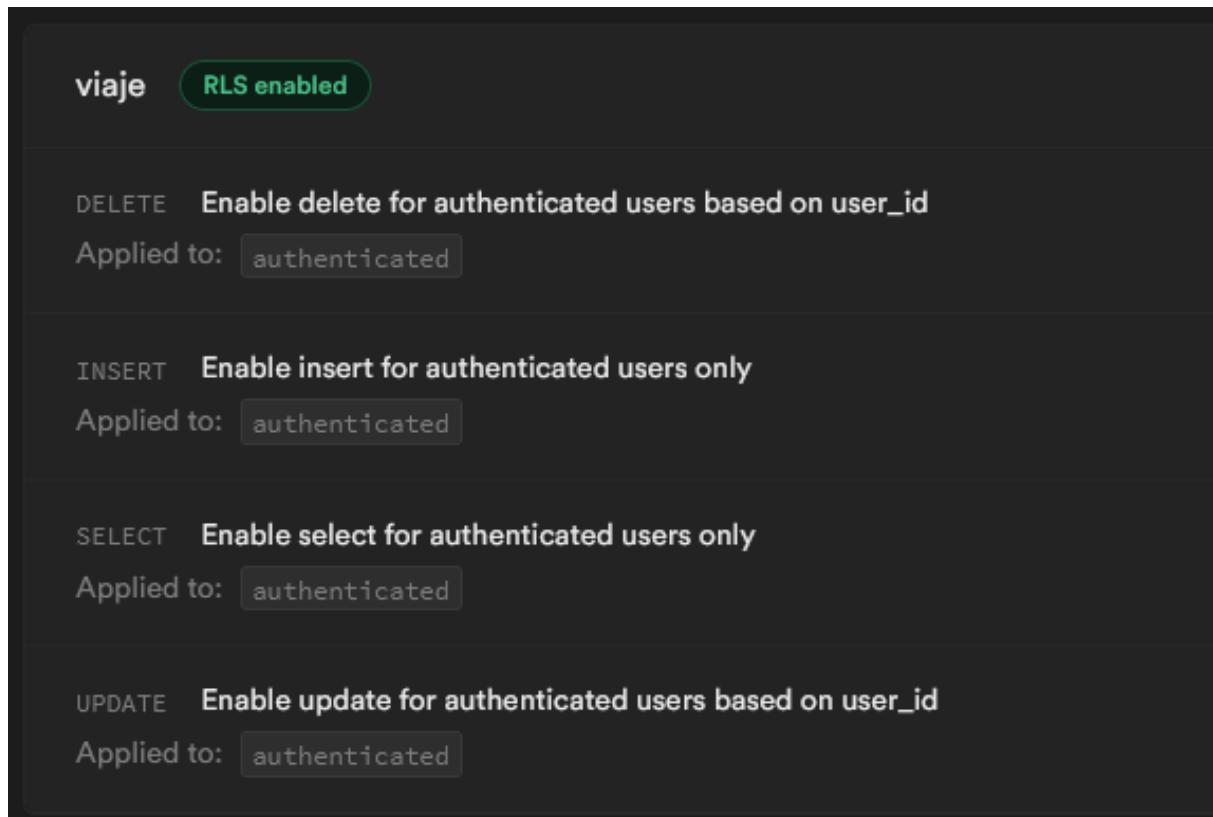


Figura 13: Políticas de la tabla viaje

En la tabla mensaje todos los usuarios autenticados pueden hacer todas las operaciones. Esto se ha hecho así para evitar problemas con el borrado automático de mensajes. Aunque estas operaciones están permitidas para los usuarios autenticados, no se dan opciones para realizar operaciones como la de borrar y en el caso de que se de, se haría una comprobación de id previamente.

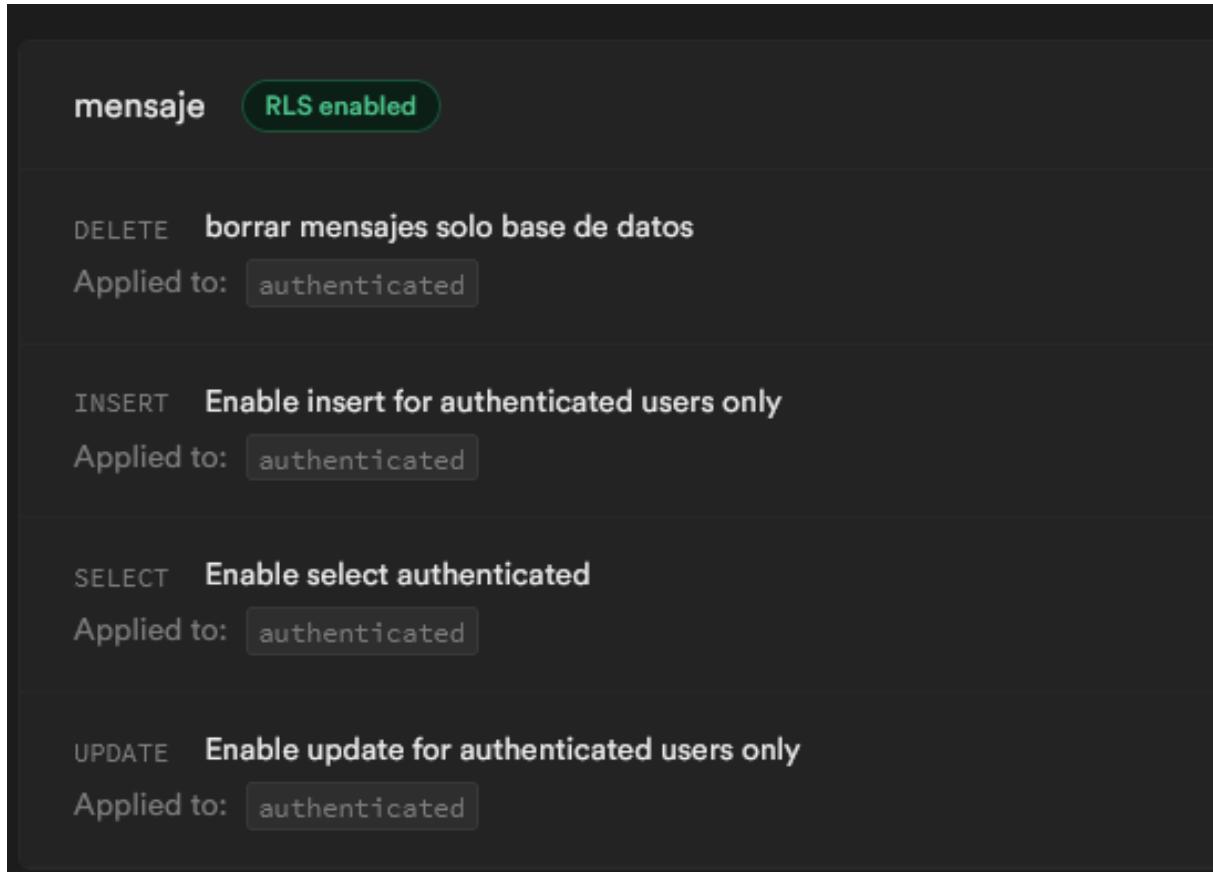


Figura 14: Políticas de la tabla mensaje

En la tabla usuario solo los usuarios cuyo id coincide con el de la fila pueden borrar y actualizar dicha fila. La operación de inserción de una fila y de seleccionar una fila las pueden hacer todos los usuarios autenticados. La selección de una fila es de esta forma para acceder a datos como el nombre de un usuario ajeno y la inserción para hacer la inserción inicial de la fila de usuario.

usuario	
DELETE	Enable delete for users based on user_id
Applied to:	authenticated
INSERT	Enable insert for authenticated users only
Applied to:	authenticated
SELECT	Enable select for users authenticated
Applied to:	authenticated
UPDATE	Enable update for users based on user_id
Applied to:	authenticated

Figura 15: Políticas de la tabla usuario

La tabla es_pasajero tiene la política de que todos los usuarios autenticados pueden hacer todas las operaciones disponibles.

es_pasajero	
ALL	Enable all for users authenticated
Applied to:	authenticated

Figura 16: Políticas de la tabla es_pasajero

En la tabla participantes_chat solo los usuarios cuyo id coincide con el de la fila pueden borrar y actualizar dicha fila. La operación de inserción de una fila y de seleccionar una fila las pueden hacer todos los usuarios autenticados.

The screenshot shows the AWS CloudTrail RLS (Row-Level Security) configuration for the 'participantes_chat' table. At the top, it says 'participantes_chat' and 'RLS enabled'. Below that, there are four policy entries:

- DELETE** **Enable delete for users based on user_id**
Applied to: **authenticated**
- INSERT** **Enable insert for authenticated users only**
Applied to: **authenticated**
- SELECT** **Enable select for users authenticated**
Applied to: **authenticated**
- UPDATE** **Enable update for users based on user_id**
Applied to: **authenticated**

Figura 17: Políticas de la tabla participantes_chat

- Restricciones sobre tablas

Todas las tablas han sido creadas con la interfaz gráfica de Supabase que no soporta el poder definir que una fila se borre si una fila de otra tabla a la que referencia es borrado. Por tanto ha sido necesario cambiar las tablas para poder tener esta funcionalidad. En las figuras siguientes pueden verse las instrucciones usadas para ello.

```
ALTER TABLE public.viaje
DROP CONSTRAINT viaje_creado_por_fkey,
ADD CONSTRAINT viaje_creado_por_fkey
    FOREIGN KEY (creado_por)
    REFERENCES usuario(id)
    ON DELETE CASCADE;
```

Figura 18: Al borrar un usuario, se borran los viajes que haya creado

```
ALTER TABLE public.es_pasajero
DROP CONSTRAINT es_pasajero_id_viaje_fkey,
ADD CONSTRAINT es_pasajero_id_viaje_fkey
    FOREIGN KEY (id_viaje)
    REFERENCES viaje(id)
    ON DELETE CASCADE;
```

Figura 19: Si un viaje es borrado, también se borran todos los pasajeros del viaje

```
ALTER TABLE public.es_pasajero
DROP CONSTRAINT es_pasajero_id_usuario_fkey,
ADD CONSTRAINT es_pasajero_id_usuario_fkey
    FOREIGN KEY (id_usuario)
    REFERENCES usuario(id)
    ON DELETE CASCADE;
```

Figura 20: Si un usuario es borrado, también se borra su participación en los viajes donde se haya apuntado

```
ALTER TABLE public.mensaje
DROP CONSTRAINT mensaje_creador_fkey,
ADD CONSTRAINT mensaje_creador_fkey
    FOREIGN KEY (creador)
    REFERENCES usuario(id)
    ON DELETE CASCADE;

ALTER TABLE public.mensaje
DROP CONSTRAINT mensaje_chat_id_fkey,
ADD CONSTRAINT mensaje_chat_id_fkey
    FOREIGN KEY (chat_id)
    REFERENCES chat(id)
    ON DELETE CASCADE;
```

Figura 21: Si el usuario creador es eliminado, se borran los mensajes relacionados y si un chat es borrado, se borran todos los mensajes relacionados respectivamente

```
ALTER TABLE public.participantes_chat
DROP CONSTRAINT participantes_chat_chat_id_fkey,
ADD CONSTRAINT participantes_chat_chat_id_fkey
    FOREIGN KEY (chat_id)
    REFERENCES chat(id)
    ON DELETE CASCADE;
```

Figura 22: Si un chat se elimina, se borran las entradas de participante de chat

```
ALTER TABLE public.participantes_chat
DROP CONSTRAINT participantes_chat_usuario_id_fkey,
ADD CONSTRAINT participantes_chat_usuario_id_fkey
    FOREIGN KEY (usuario_id)
    REFERENCES usuario(id)
    ON DELETE CASCADE;
```

Figura 23: si se borra un usuario, se borra su participación de los chats donde se encuentre

3.4. Diagramas de clase

Para la aplicación se ha seguido una arquitectura tipo MVC (modelo-vista-controlador). Por ello se han separado los diagramas para cada una de esas partes. Para simplificar la comprensión de los diagramas cuando una clase en un diagrama tiene una relación con una clase que esta en otro diagrama, en vez de representar la clase entera, se ha representado como un atributo con el tipo de la clase externa.

La aplicación ha sido programada en **Dart** por lo que los tipos son los pertenecientes a este. Además este lenguaje tiene una función llamada *Null safety* [29] que hace que las variables no puedan ser null por defecto y avisa en caso de que haya algún problema relacionado. En el caso de querer que una variable pueda ser null, es necesario añadir el símbolo "?" al final del tipo de la variable. Un ejemplo de esto último puede ser, si queremos tener una variable que es un número entero que también puede ser null, este sería llamado "int? nombreVariable".

Los diagramas siguientes son muy detallados debido a que se han completado después de crear la aplicación. Para ver los diagramas inicialmente desarrollados e intermedios, consulte el apéndice E, [Diagramas de clases anteriores](#). Se ha usado **Visual paradigm** para crear los diagramas siguientes.

3.4.1. Modelo

El modelo de la aplicación esta formado por tres clases, una clase Usuario, una clase Oferta y una clase Chat. Estas clases principalmente representan las entidades de la base de datos usuario, viaje y chat respectivamente.

Como se puede observar en [Diagrama de clases del modelo de datos](#), las clases aparte de los constructores por defecto tienen un método *copyWith()*. Este método recibe uno o muchos parámetros y devuelve un objeto en los que solo los parámetros dados han cambiado y el resto son iguales. Este método es necesario porque en Dart se considera una buena practica que los objetos sean inmutables, es decir que una vez el objeto ha sido creado con unos valores, estos no se pueden cambiar.

Otros métodos que tienen como los llamados *fromKeyValue()* o *fromList()* sirven para crear un objeto a partir de un objeto con pares llave-valor como puede ser un JSON [30]

y para crear una lista de objetos a partir de una lista respectivamente.

Por otro lado, se puede observar que la relación entre la clase Usuario y las clases Oferta y Chat es de composición. Esto se debe a que ni una oferta puede existir sin un usuario que lo haya creado ni un chat puede existir sin tener dos usuarios.

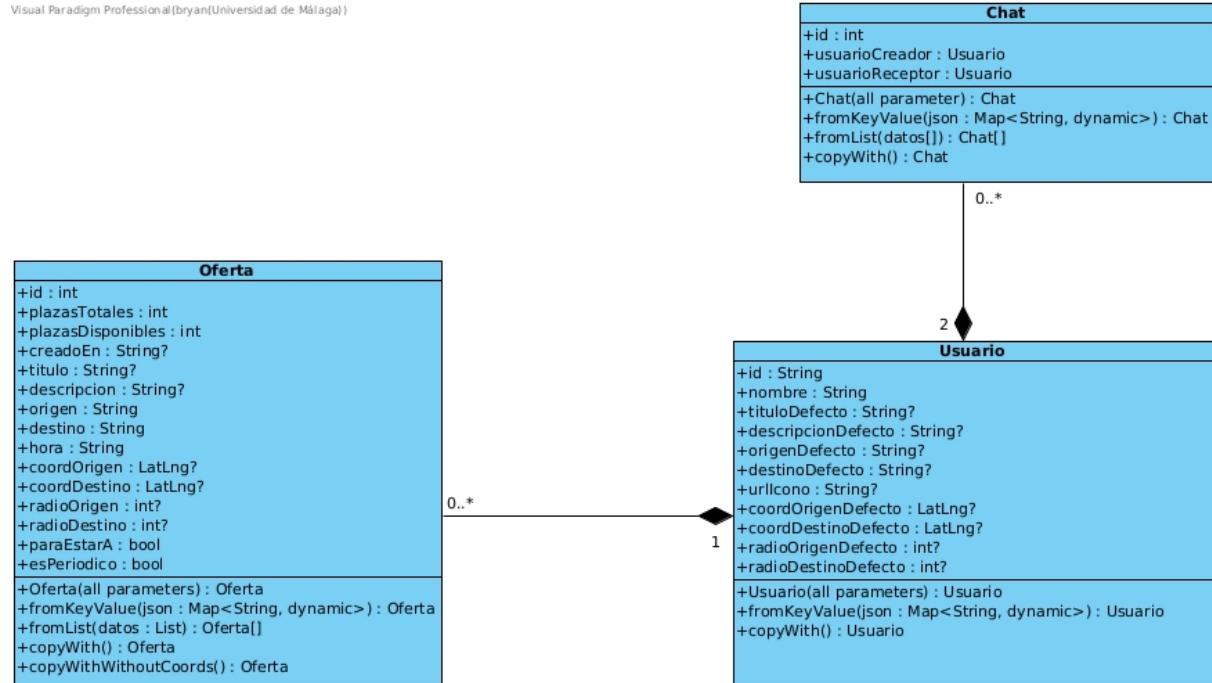


Figura 24: Diagrama de clases del modelo de datos

3.4.2. Vista

Para la vista, ya que se ha usado **Flutter** todas las clases de la vista se han hecho siguiendo sus especificaciones. Más concretamente, en **Flutter** las clases de interfaz de usuario se agrupan en forma de árbol, es decir, que existe una clase padre de toda la aplicación y esa clase tiene un hijo que a su vez tiene más hijos dentro. De esta forma consigue que sea sencillo crear interfaces gráficas complejas y adaptables y de manera eficiente. Esta forma de funcionar se puede observar en [Diagrama de clases de la vista de la aplicación](#) donde hay una alta cohesión de las clases debido a que hay clases que no existen en tiempo de ejecución si no son hijas de otras clases y si las clases padres desaparecen también lo hacen las hijas.

Todas las clases representadas heredan de *StatelessWidget*, *StatefulWidget* o una implementación que hereda de estas de Riverpod que es un marco de trabajo de enlace de datos [31] que se ha usado. Esto hace que todas estas clases tengan un método *build()* para crear su interfaz y llamar a sus hijos y también que tengan un atributo de tipo *Key?*.

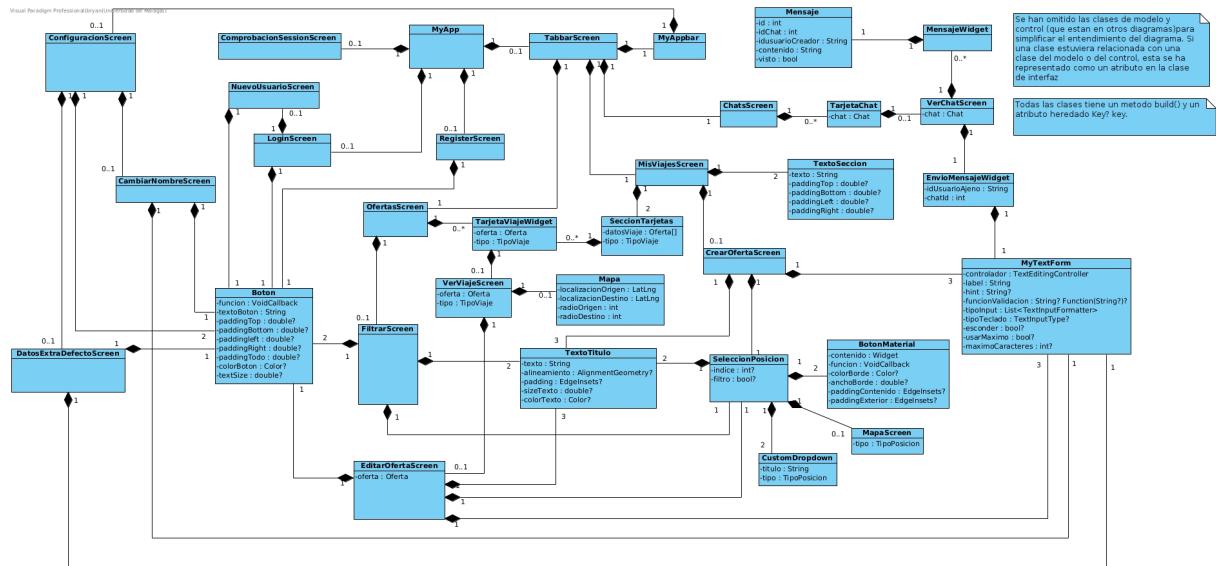


Figura 25: Diagrama de clases de la vista de la aplicación

3.4.3. Control

Para el control, la lógica de la aplicación se ha hecho usando un paquete llamado Riverpod que se encarga de enlazar los datos entre distintas partes de la aplicación [31].

La forma de usar este paquete es creando proveedores de información. Esta información puede ser una sola variable o una clase de control. Estos proveedores se definen como variables globales pero solo son accesibles desde otras clases que hereden de clases de Riverpod a través de una variable de tipo *Ref*. Que sea necesario usar esta variable es lo que hace que en la figura [Diagrama de clases de control de la aplicación](#) se pueda ver la alta cohesión con una clase en el centro del diagrama.

Por simplificación, las clases relacionadas con el modelo en vez de volver a aparecer en este diagrama, cuando una clase tuviese una relación con una clase del modelo esta ha sido representada como una variable en la clase de control. Un ejemplo de esto sería en la clase *OfertasController* donde en vez de encontrarse una relación uno a muchos con la clase del Modelo Oferta, se puede ver como tiene una variable *Oferta[]* que es el estado del proveedor.

Para facilitar cambios de tecnología en un futuro de ser necesario, como puede ser cambiar de [Supabase](#) a Firebase o cambiar el servicio de localización usado, se ha creado una interfaz para la base de datos que una clase debe implementar al completo para funcionar. En caso de que se cambie de servicio, solo es necesario volver a implementar esta clase con el servicio nuevo y el resto de la aplicación seguirá funcionando de manera correcta. Para la localización se ha seguido el mismo proceso.

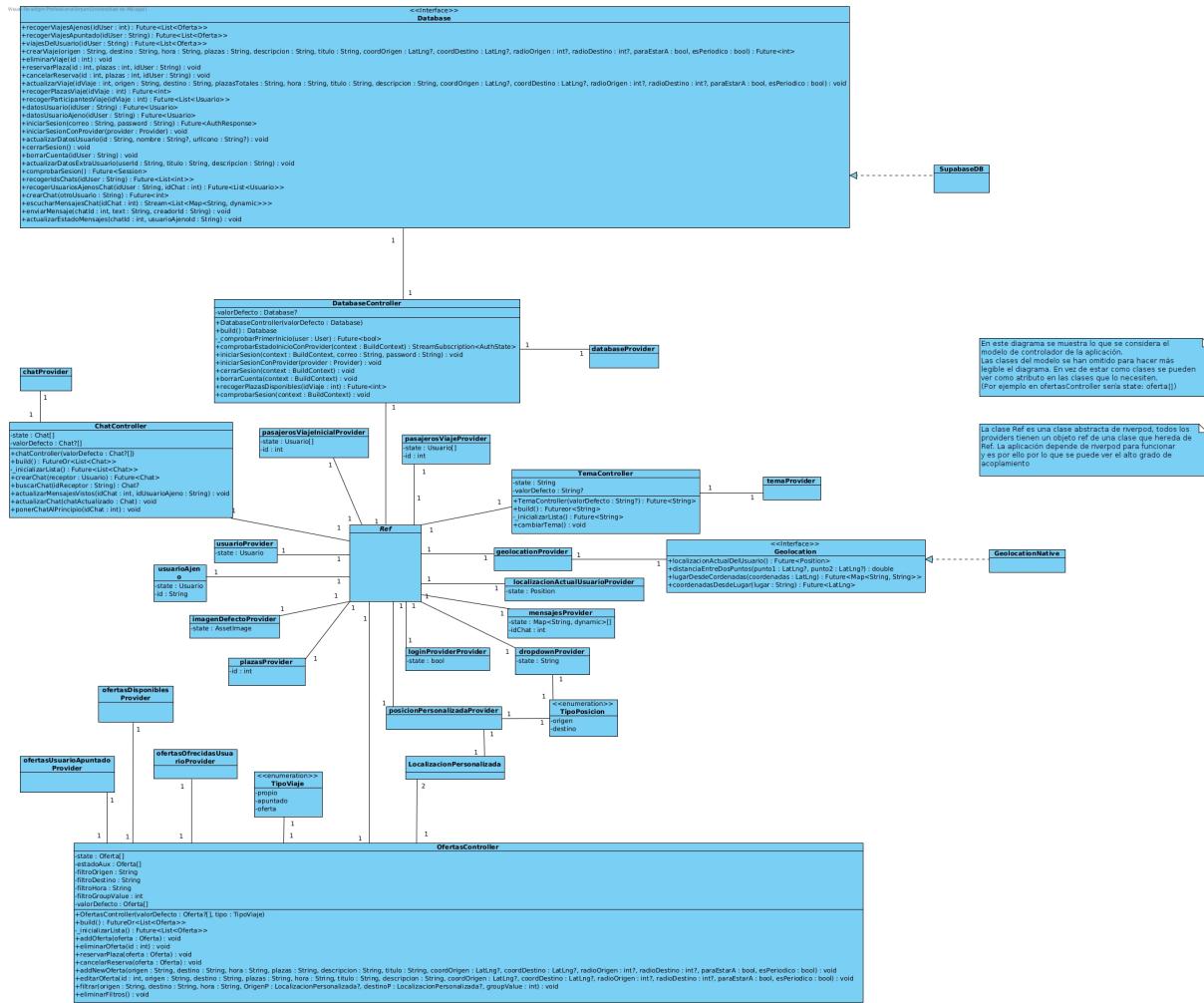


Figura 26: Diagrama de clases de control de la aplicación

3.5. Diagramas de secuencia

Una vez definida la estructura de la aplicación se han generado diagramas de secuencia para especificar el funcionamiento lógico aproximado y las interacciones que se deben seguir. Se ha intentado especificar cada parte de la aplicación con al menos un diagrama. Los diagramas de esta sección son la versión final generada, para ver versiones anteriores acuda al apéndice F, [Diagramas de secuencia anteriores](#)

Los diagramas siguientes han sido desarrollados con la herramienta [Modelio](#).

3.5.1. Secuencia de inicio de sesión

Este diagrama esta relacionado con el requisito [RF-1. Iniciar sesión](#).

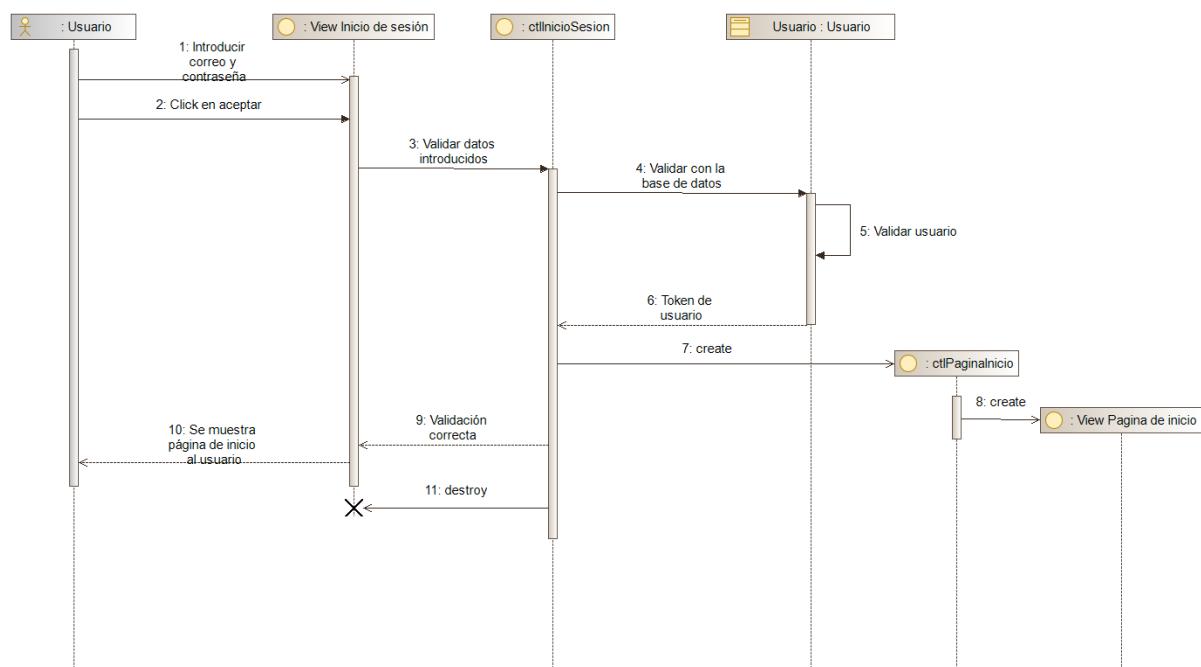


Figura 27: Diagrama de secuencia de iniciar sesión

Aquí un usuario introduce unas credenciales correctas y el sistema valida que el correo electrónico es válido y posteriormente que el par correo y contraseña son válidos. Hecho esto se devuelve un token de sesión y se crean y muestran las pantallas de la aplicación necesarias.

3.5.2. Secuencia de registro de usuario

Este diagrama esta relacionado con el requisito RF-2. Registro de usuario.

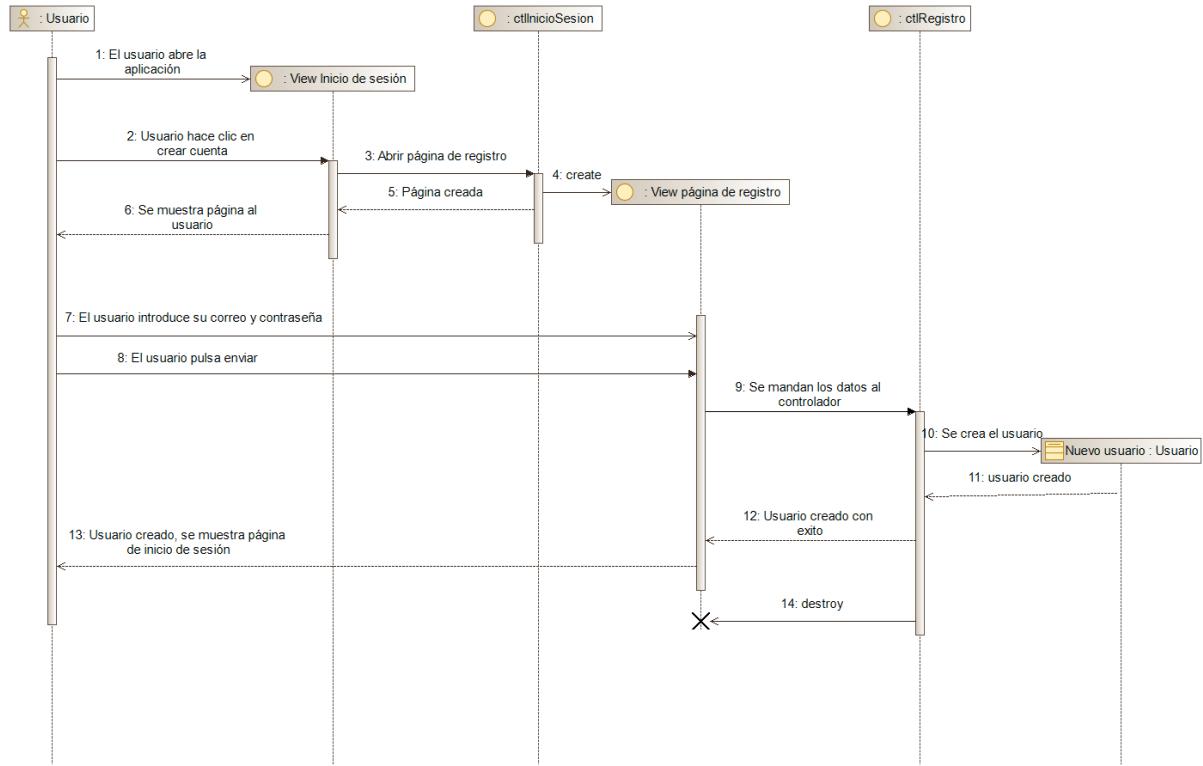


Figura 28: Diagrama de secuencia de registro de usuario

En esta interacción el usuario abre la aplicación, hace clic en el botón de crear cuenta, la aplicación le muestra la pantalla y el usuario introduce los datos necesarios para crear una cuenta. Una vez el usuario pulsa el botón de enviar, el sistema crea un nuevo usuario con los datos dados y devuelve al usuario a la pantalla de inicio de sesión.

3.5.3. Secuencia de crear oferta de viaje

Este diagrama esta relacionado con el requisito RF-7. Crear oferta de viaje.

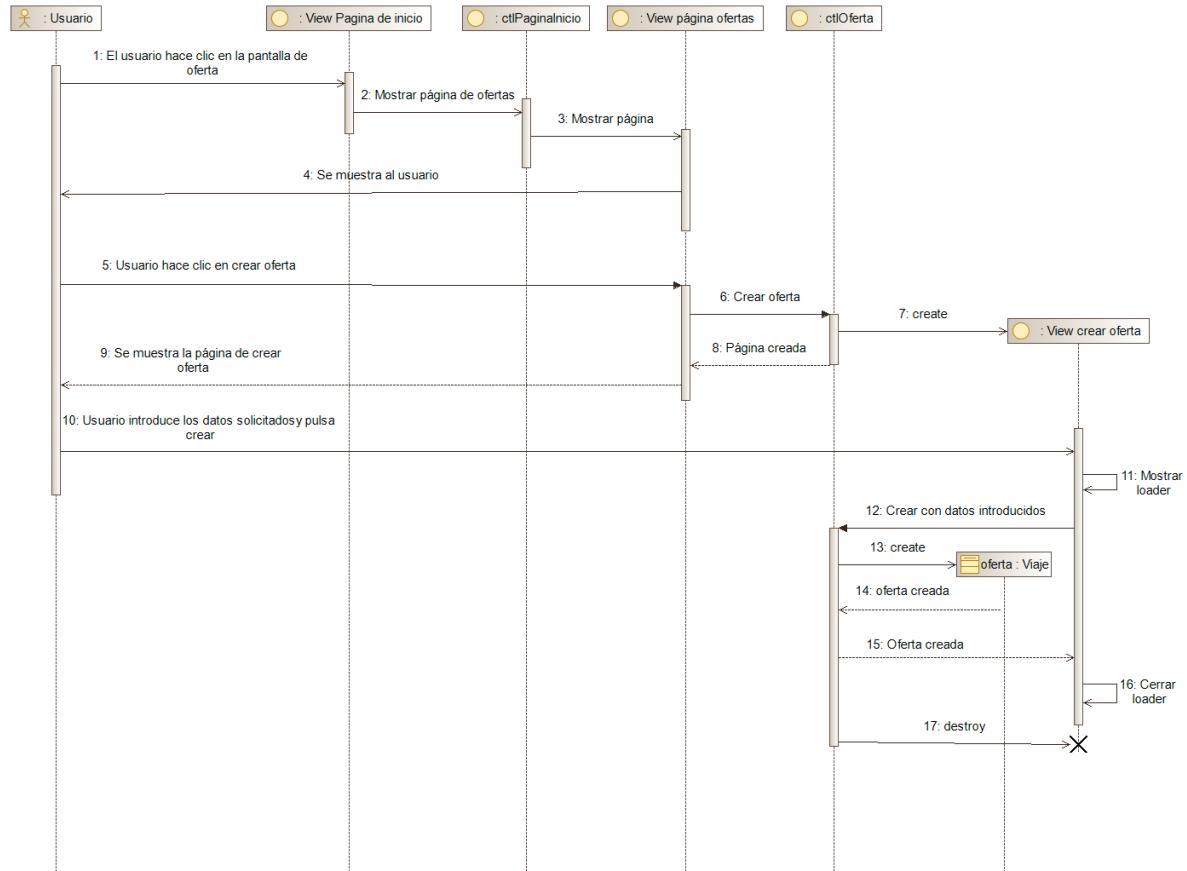


Figura 29: Diagrama de secuencia de crear un viaje nuevo

En esta interacción, el usuario se dirige a la pantalla de ofertas y pulsa en el botón de crear una nueva oferta y se muestra una pantalla donde introducir los datos necesarios. El usuario rellena los campos y pulsa en el botón de crear. Después el sistema crear un nuevo objeto oferta y cierra la pantalla.

3.5.4. Secuencia de editar oferta

Este diagrama esta relacionado con el requisito RF-7.3. Editar oferta de viaje.

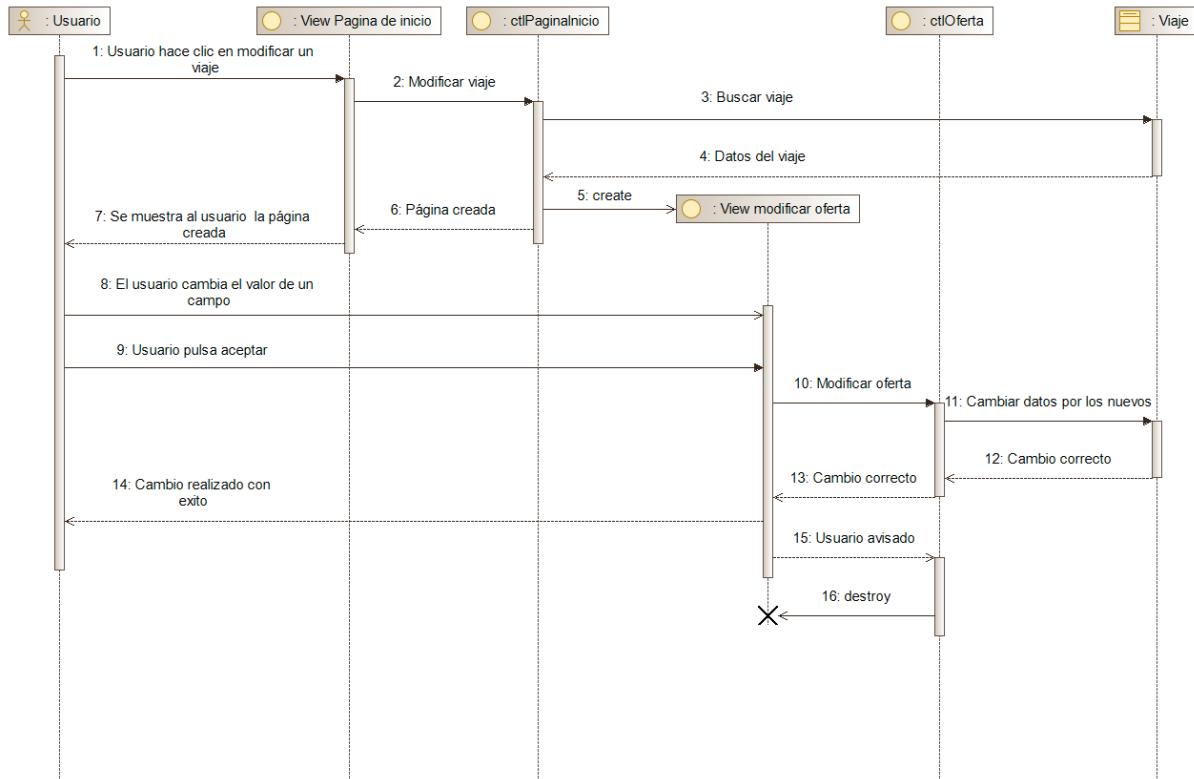


Figura 30: Diagrama de secuencia de editar oferta

En esta interacción el usuario se dirige a un viaje concreto que haya creado y pulsa el botón de editar. Una vez en la pantalla de editar un viaje cambia el valor que quiera que este disponible y pulsa aceptar. Después el sistema cambia los datos antiguos por los nuevos que hayan sido cambiados y devuelve al usuario a la pantalla anterior.

3.5.5. Secuencia de filtrar por posición

Este diagrama esta relacionado con el requisito RF-9.2. Filtrar por posición.

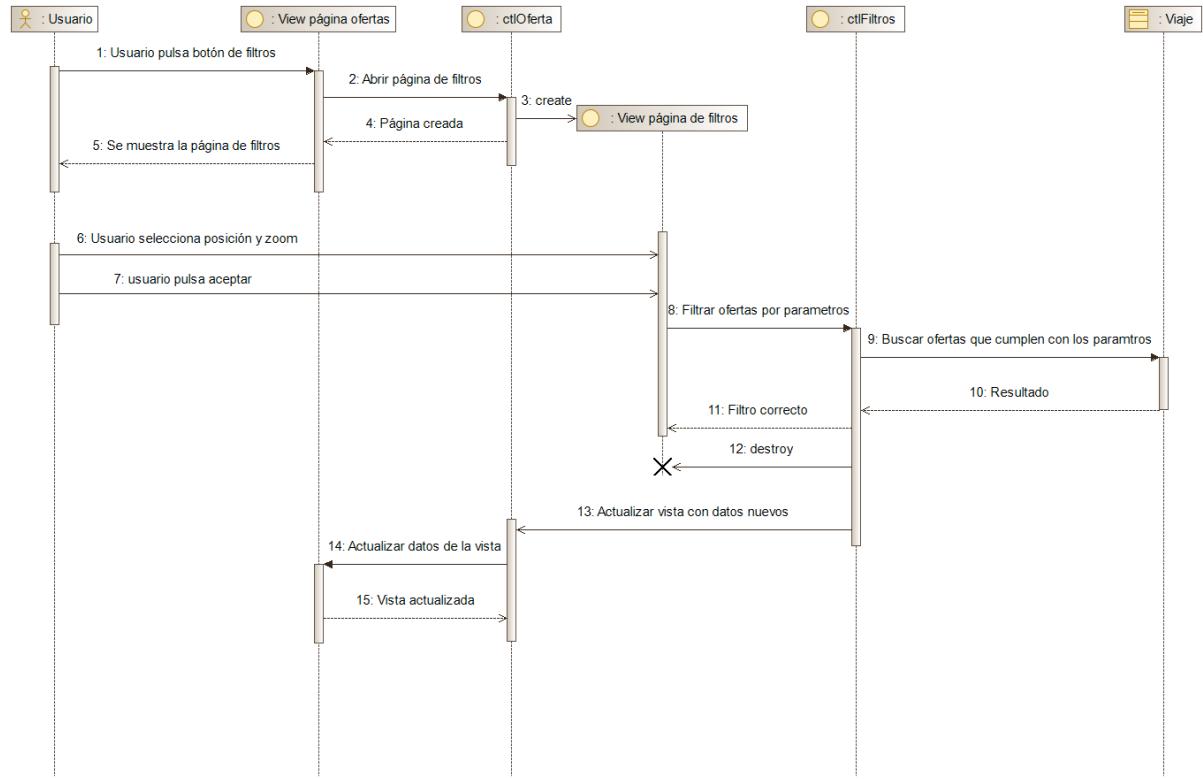


Figura 31: Diagrama de secuencia de filtrar por posición

En esta interacción, el usuario se dirige a la pantalla de ofertas y pulsa en el botón de filtrar. El sistema muestra la pantalla con los filtros disponibles y el usuario selecciona un filtro de posición. Una vez el usuario aplica el filtro, el sistema busca entre las ofertas disponibles las que coinciden con el filtro seleccionado y las muestra al usuario en la página de ofertas.

3.5.6. Secuencia de enviar mensaje

Este diagrama esta relacionado con el requisito RF-8.2. Enviar mensajes.

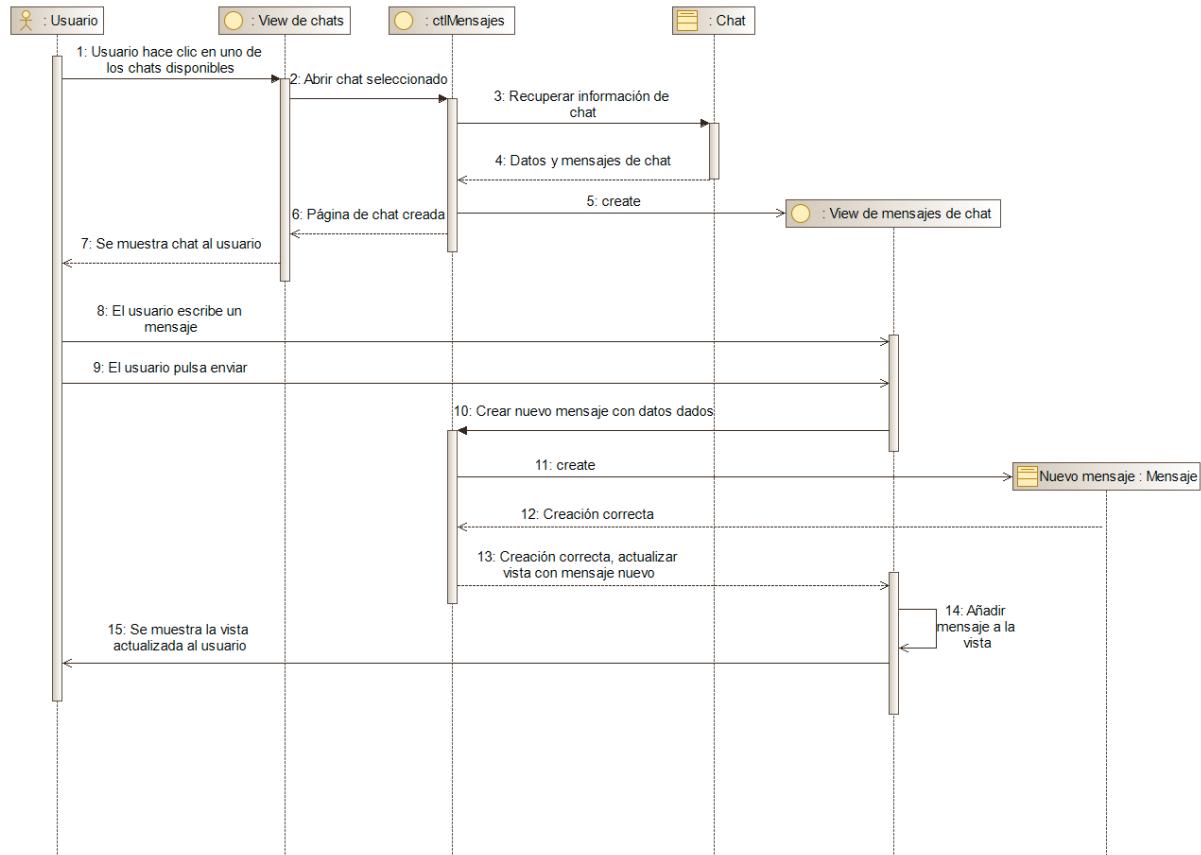


Figura 32: Diagrama de secuencia de enviar mensaje

En esta interacción, el usuario se abre uno de los chats que ha creado anteriormente. Al pulsar en el chat, el sistema carga todo los mensajes y los muestra. Después el usuario escribe un mensaje en la campo de texto disponible y pulsa enviar. Al hacer esto, el sistema crea un nuevo mensaje y actualiza la vista con el mismo.

3.5.7. Secuencia de cambiar tema de color

Este diagrama esta relacionado con el requisito RF-10. Cambiar tema de color.

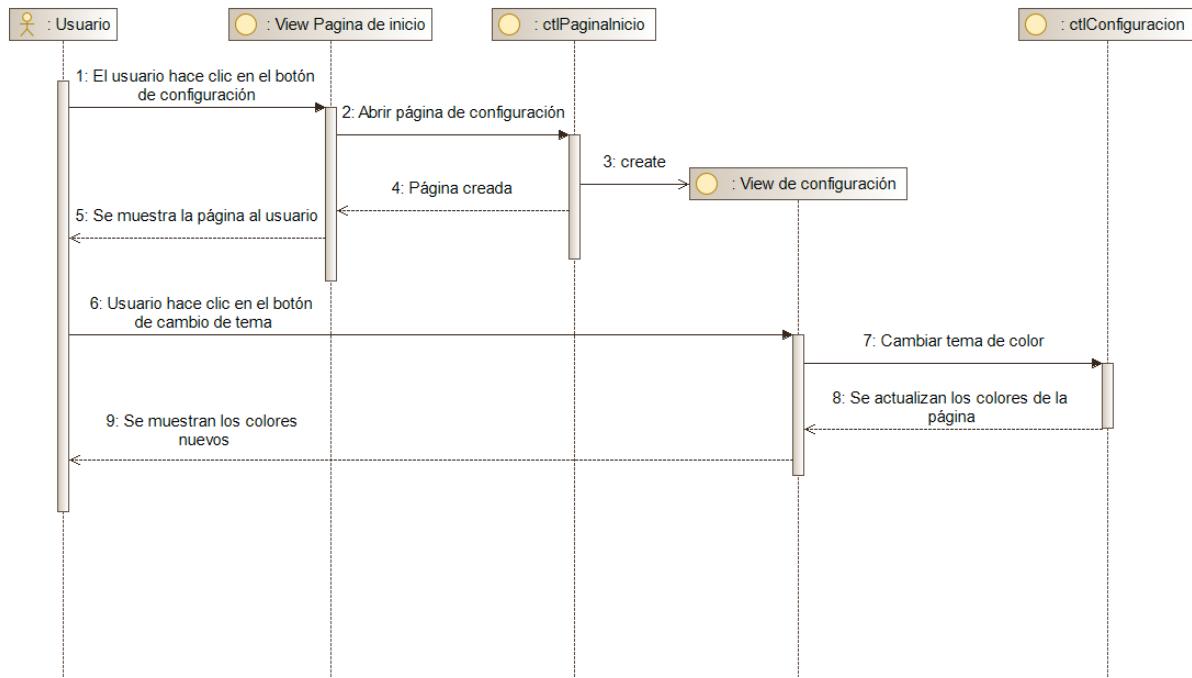


Figura 33: Diagrama de secuencia de cambiar tema de color

En esta interacción, el usuario se dirige a la pantalla de configuración y hace clic en el botón de cambio de tema. El sistema después actualiza todos los colores de las pantallas de la aplicación.

3.6. Diagramas de estado

Para acompañar a los Diagramas de secuencia y completar aun más el diseño del funcionamiento de la aplicación se han creado los siguientes diagramas de estado. Estos son los diagramas finales generados, para ver los anteriores acuda al apéndice G, [Diagramas de estado anteriores](#). Estos diagramas se han desarrollado con la herramienta [Modelio](#)

3.6.1. Diagrama de estado de iniciar sesión

Este diagrama esta relacionado con el requisito RF-1. Iniciar sesión.

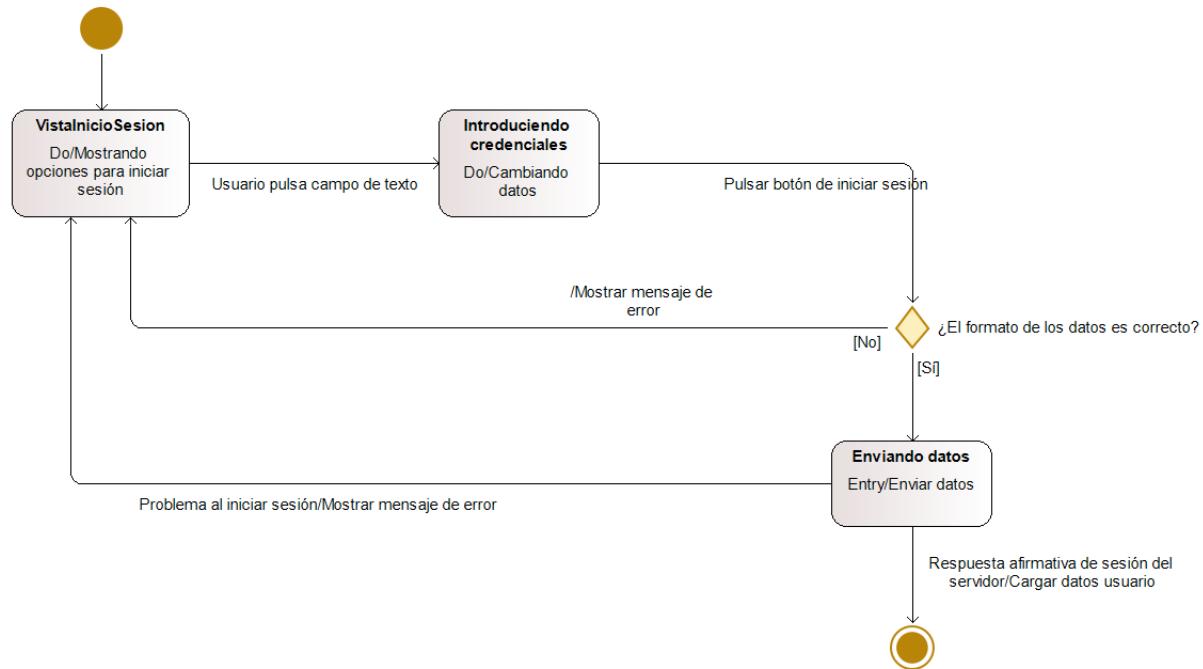


Figura 34: Diagrama de estado de iniciar sesión

El diagrama comienza cuando el usuario se encuentra en la pantalla de inicio de sesión. Aquí, si pulsa en el campo de texto disponible pasa al estado de “introducir credenciales” donde el usuario esta cambiando los datos del campo. Una pulsa el botón para iniciar sesión si los datos que ah introducido son correctos se pasará al estado “Enviando datos”. En el caso de que los datos no sean correctos se vuelve al estado inicial mostrando un mensaje de error. Desde el estado “Enviando datos”se mandan los datos al servidor y si se recibe una respuesta afirmativa se llega al estado final y si hay algún problema se vuelve al estado inicial mostrando un error.

3.6.2. Diagrama de estado de editar oferta

Este diagrama esta relacionado con el requisito RF-7.3. Editar oferta de viaje.

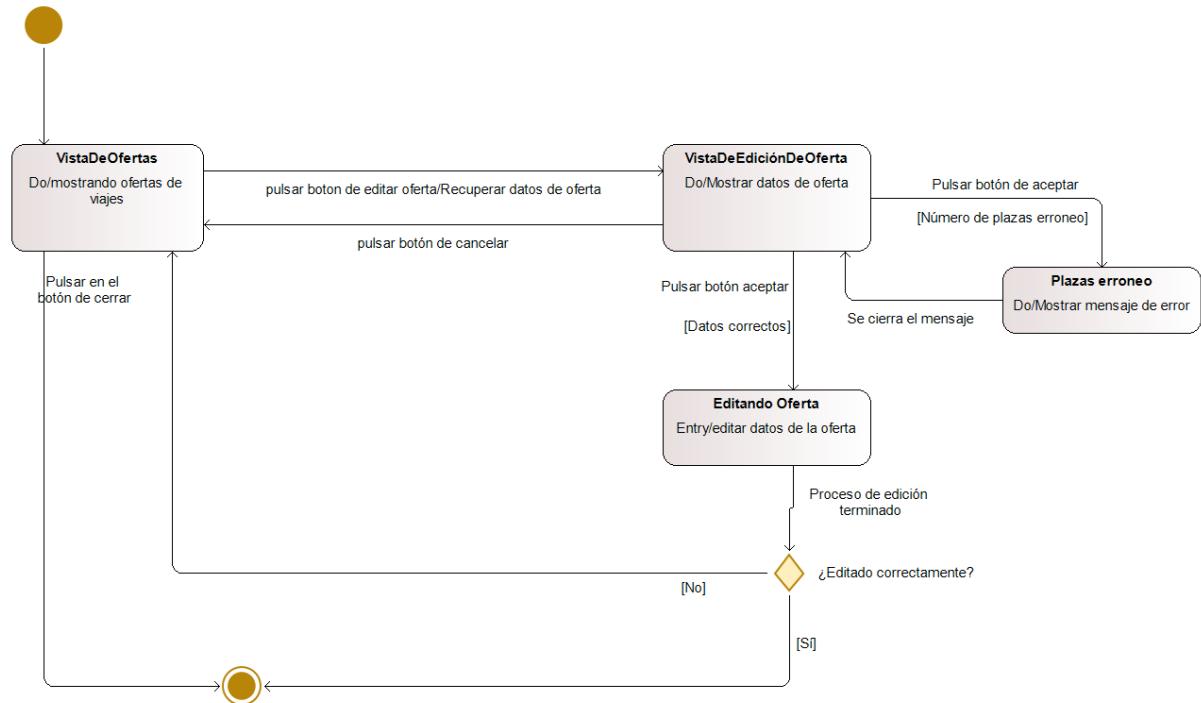


Figura 35: Diagrama de estado de editar oferta

Se comienza en el estado “VistaDeOfertas” donde se están mostrando las ofertas disponibles. Si el usuario pulsa el botón de cerrar se pasa al estado final y si pulsa el botón para editar una oferta se pasa al estado “VistaDeEdicionOferta” y se recuperan los datos de la oferta. En este estado se muestran los datos de la oferta, si el usuario pulsa el botón de aceptar y hay un número de plazas erróneo se pasa al estado “Plazas erróneo” y se muestra un mensaje de error. Una vez se cierra el mensaje se vuelve al estado anterior. Por el contrario si los datos son correctos, se pasa al estado “Editando oferta” y cuando se han cambiado los datos de la oferta correctamente se pasa al estado final y cuando ha habido algún problema se pasa al estado inicial.

3.6.3. Diagrama de estado de borrar oferta

Este diagrama esta relacionado con el requisito RF-7.2. Borrar oferta de viaje.

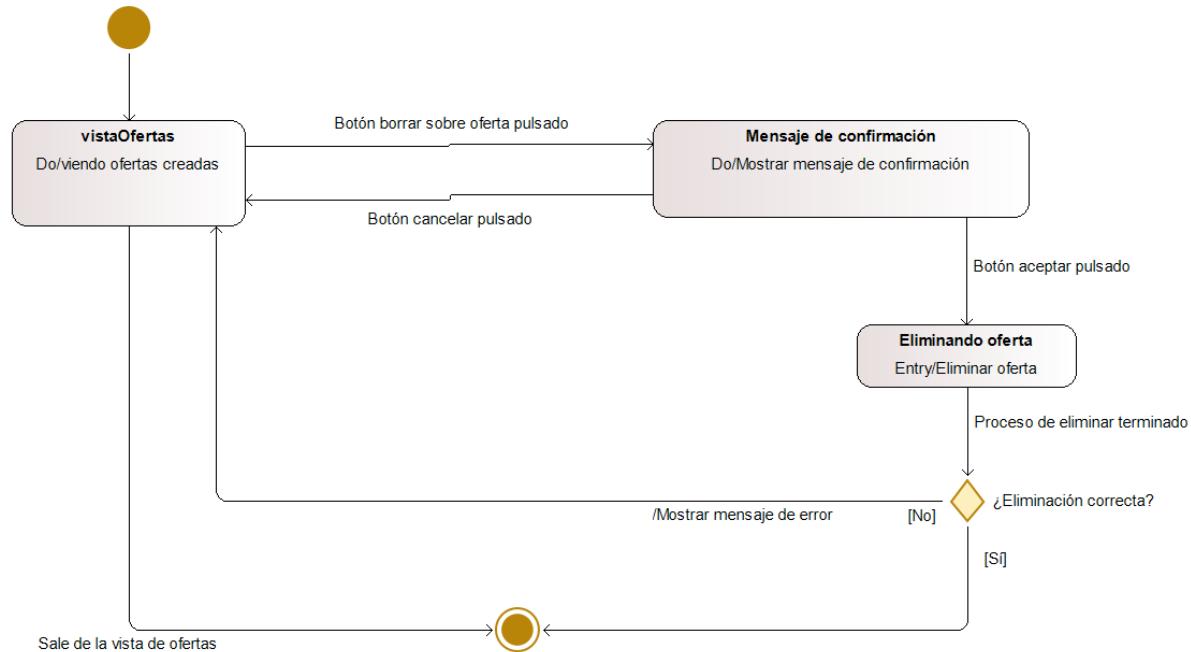


Figura 36: Diagrama de estado de borrar oferta

Se comienza en el estado “vistaOfertas” mostrando las ofertas disponibles. Desde aquí, si el usuario sale de la vista de ofertas se pasa al estado final y si se pulsa el botón de borrar se pasa al estado “Mensaje de confirmación”. En este estado se muestra un mensaje para confirmar que se quiere borrar el viaje. Si el usuario pulsa el botón de cancelar vuelve al estado inicial y si pulsa aceptar se pasa al estado “Eliminando oferta” donde se elimina la oferta y si hay algún problema se pasa al estado inicial mostrando un mensaje de error y si se elimina de forma correcta se pasa al estado final.

3.6.4. Diagrama de estado de filtrar por hora

Este diagrama esta relacionado con el requisito RF-9.1. Filtrar por hora.

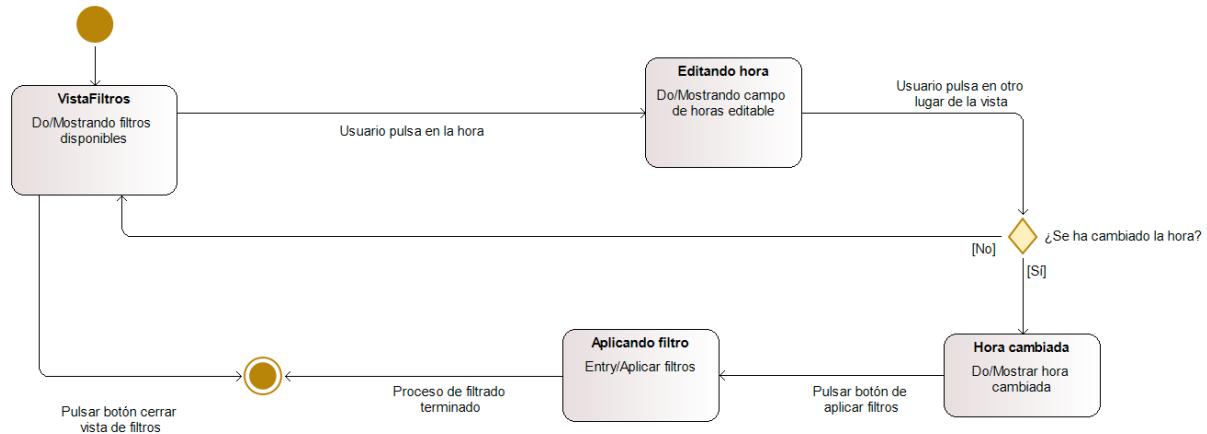


Figura 37: Diagrama de estado de filtrar por hora

Se comienza en el estado “VistaFiltros” donde se muestran los filtros aplicables. Si el usuario pulsa en el botón de cerrar se pasa al estado final y si pulsa en la hora pasa al estado “Editando hora” donde se muestra el campo de editar hora. Si el usuario pulsa en cualquier otro lado y la hora no ha sido cambiada se vuelve al estado inicial. En cambio, si ha sido cambiado se pasa al estado “Hora cambiada” y se muestra la hora nueva. Una vez el usuario pulsa el botón de aplicar filtros y pasa al estado “Aplicando filtro” donde al terminar se pasa al estado final.

3.6.5. Diagrama de estado de enviar mensaje

Este diagrama esta relacionado con el requisito RF-8.2. Enviar mensajes.

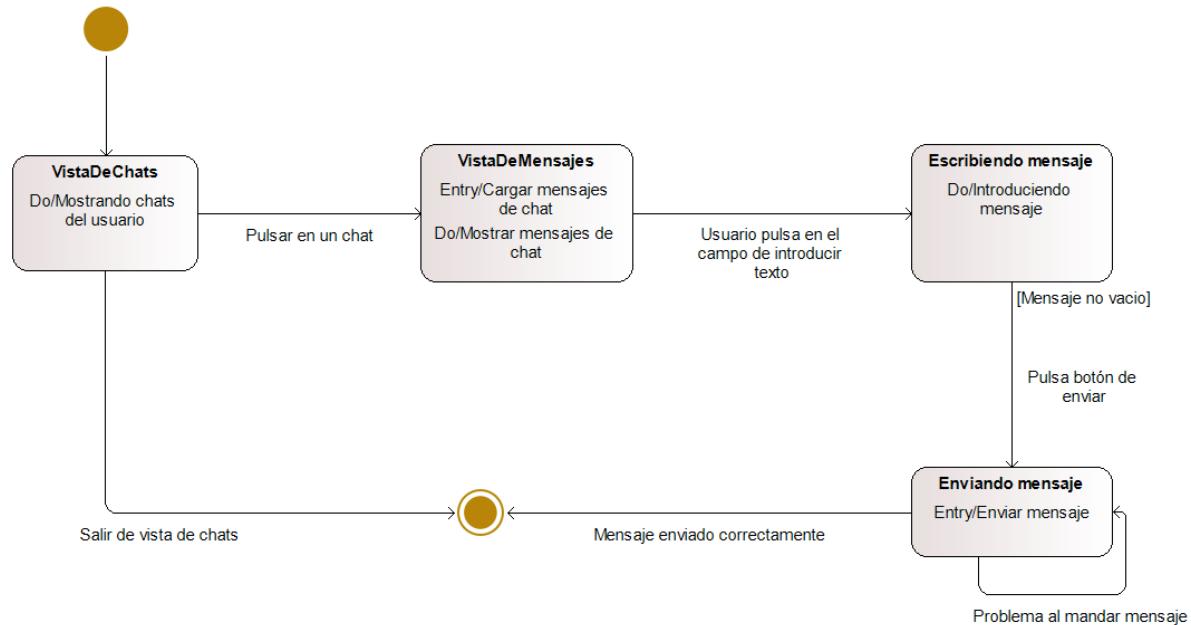


Figura 38: Diagrama de estado de enviar mensaje

Se comienza en el estado “VistaDeChats” donde se ven todos los chats del usuario. Si se sale de la pantalla de chats se pasa al estado final y si se pulsa en un chat se pasa al estado “VistaDeMensajes”. En este estado se cargan los mensajes y se muestran. Si el usuario pulsa en el campo para introducir texto se pasa al estado “Escribiendo mensaje”. Si este estado se pulsa el botón de enviar y el mensaje no esta vacío se pasa al estado “Enviando mensaje” donde se envía el mensaje hasta que se haya hecho de manera correcta y pasa al estado final.

3.7. Maquetas de software

Por último se han creado maquetas de software para dejar que el cliente pueda ver de manera sencilla y rápida que es lo que se va a hacer y cambiar cosas de ser necesario. Los mockups de esta sección son los finales, para ver versiones anteriores acuda al apéndice H, [Maquetas iniciales de software realizadas](#). Las maquetas han sido diseñadas con la herramienta [Pencil project](#).

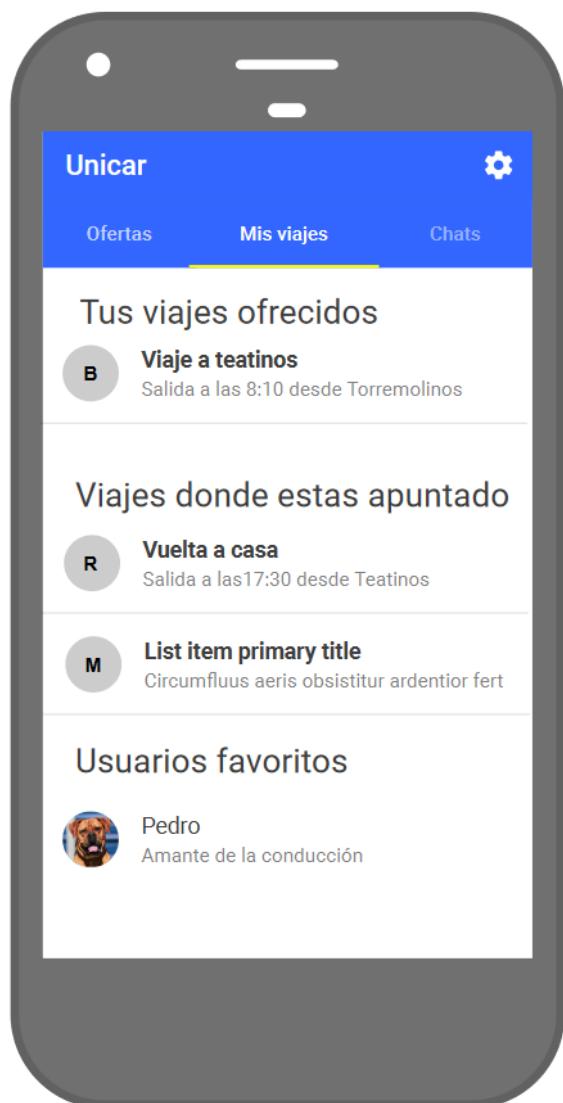


Figura 39: Maqueta de pantalla de viajes del usuario

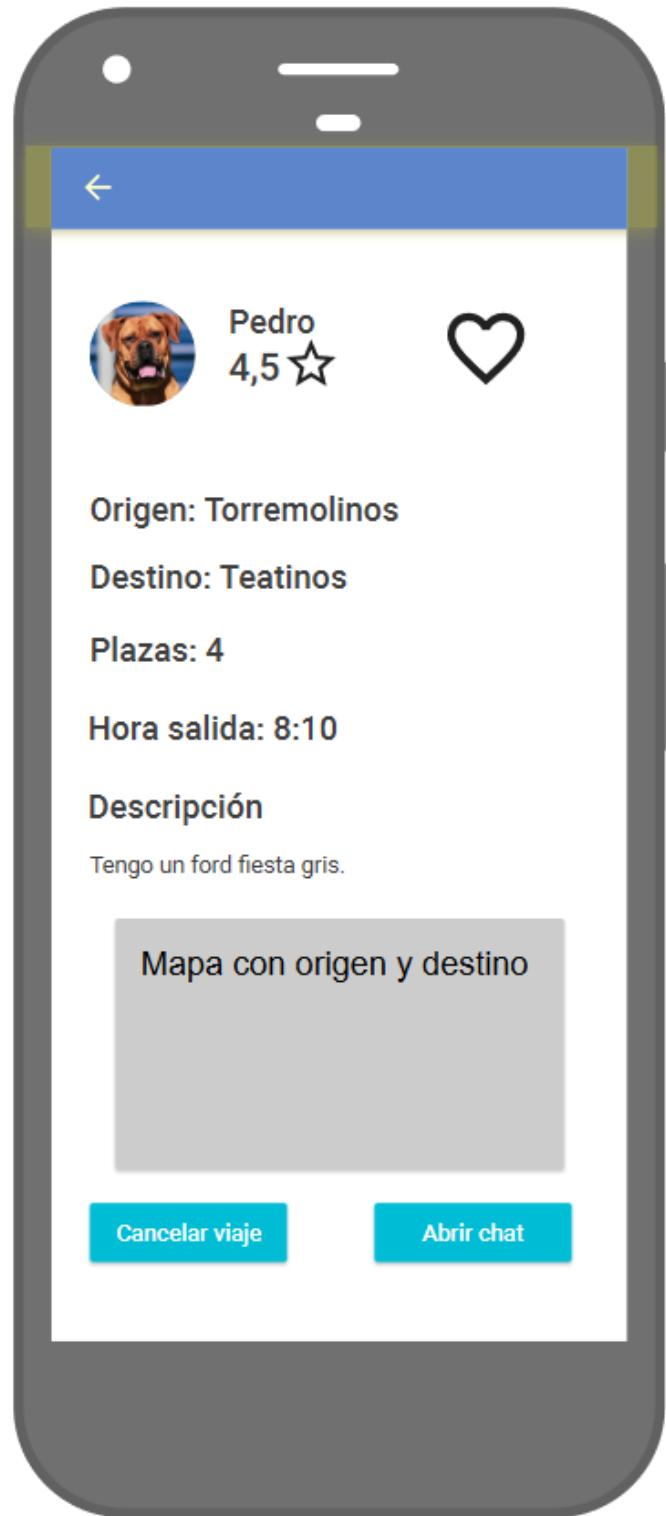


Figura 40: Maqueta de pantalla de viaje concreto en el que el usuario ha reservado plaza

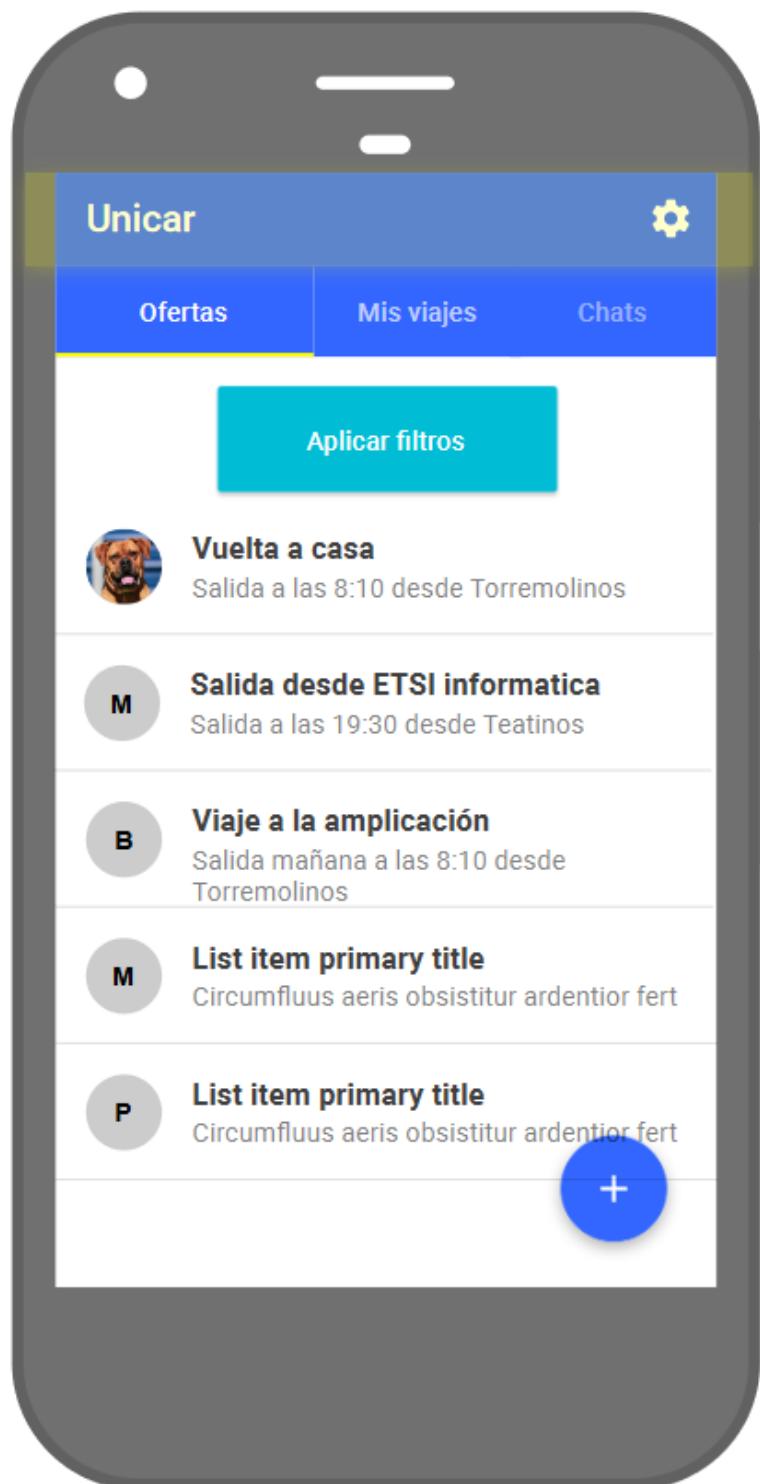


Figura 41: Maqueta de pantalla de ofertas disponibles

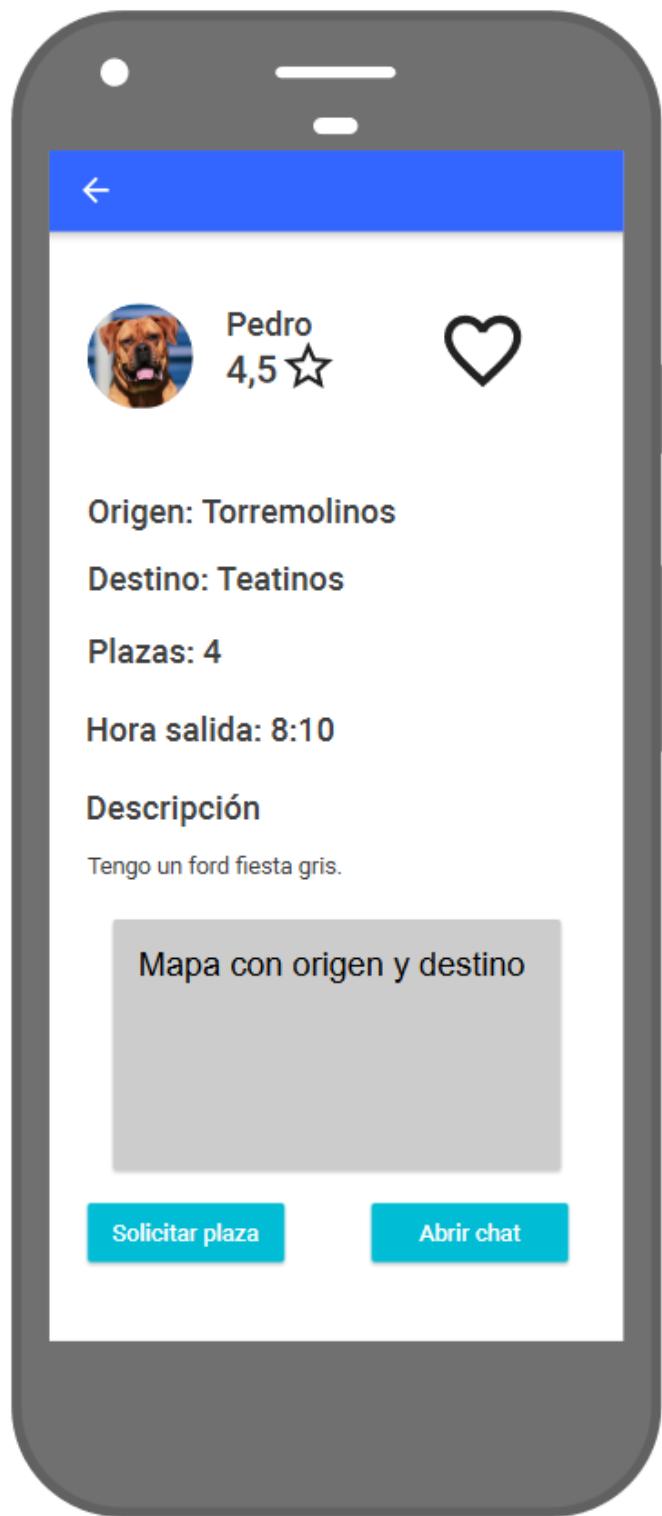


Figura 42: Maqueta de pantalla de datos de oferta disponible

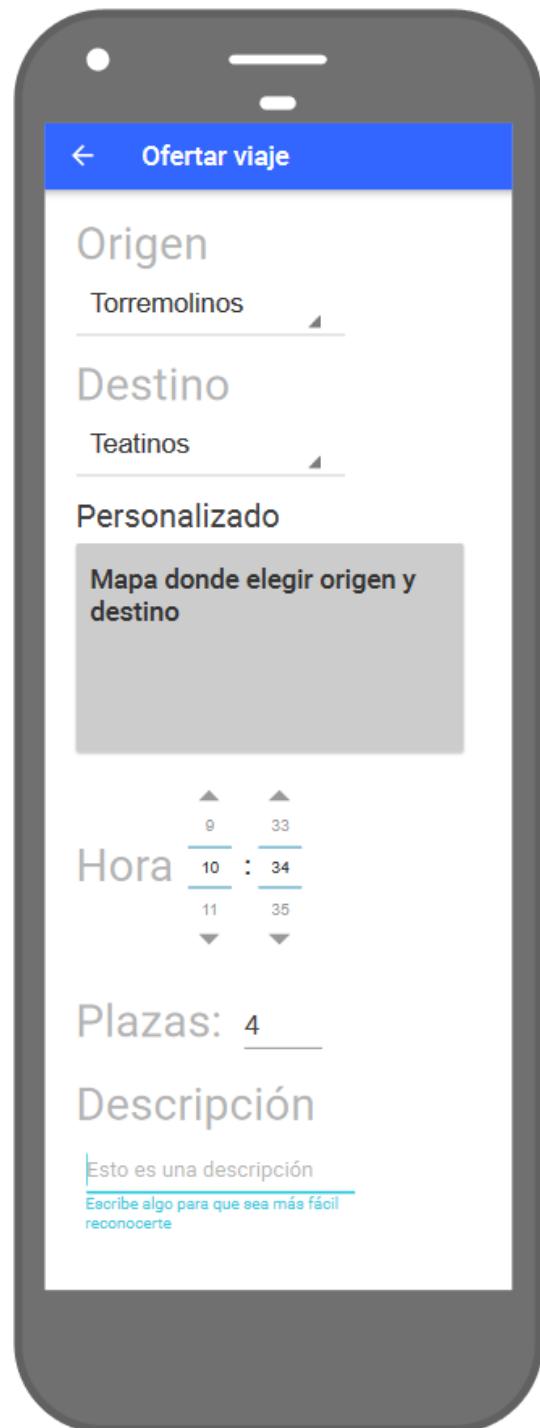


Figura 43: Maqueta de pantalla de crear una nueva oferta



Figura 44: Maqueta de pantalla de filtros disponibles

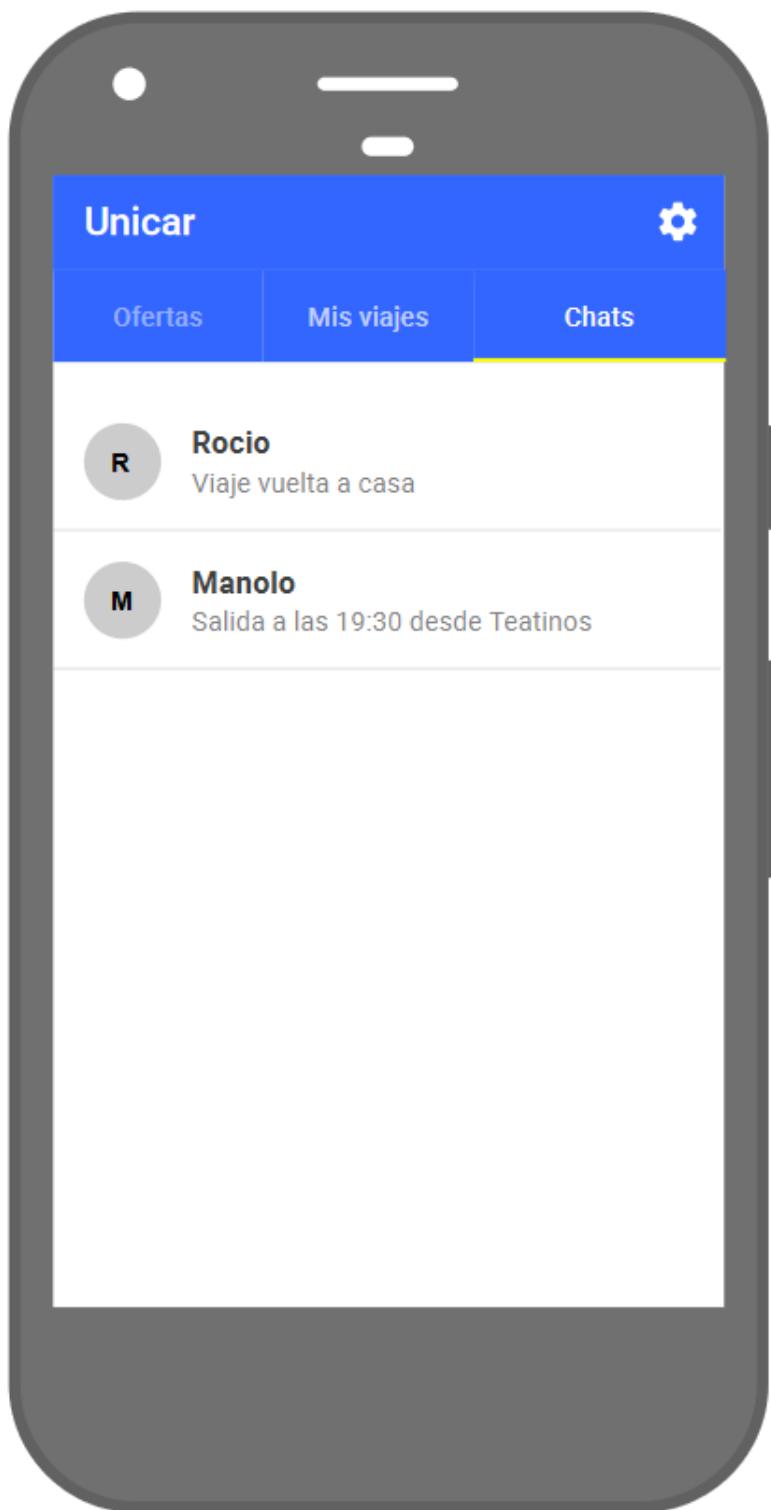


Figura 45: Maqueta de pantalla de chats abiertos

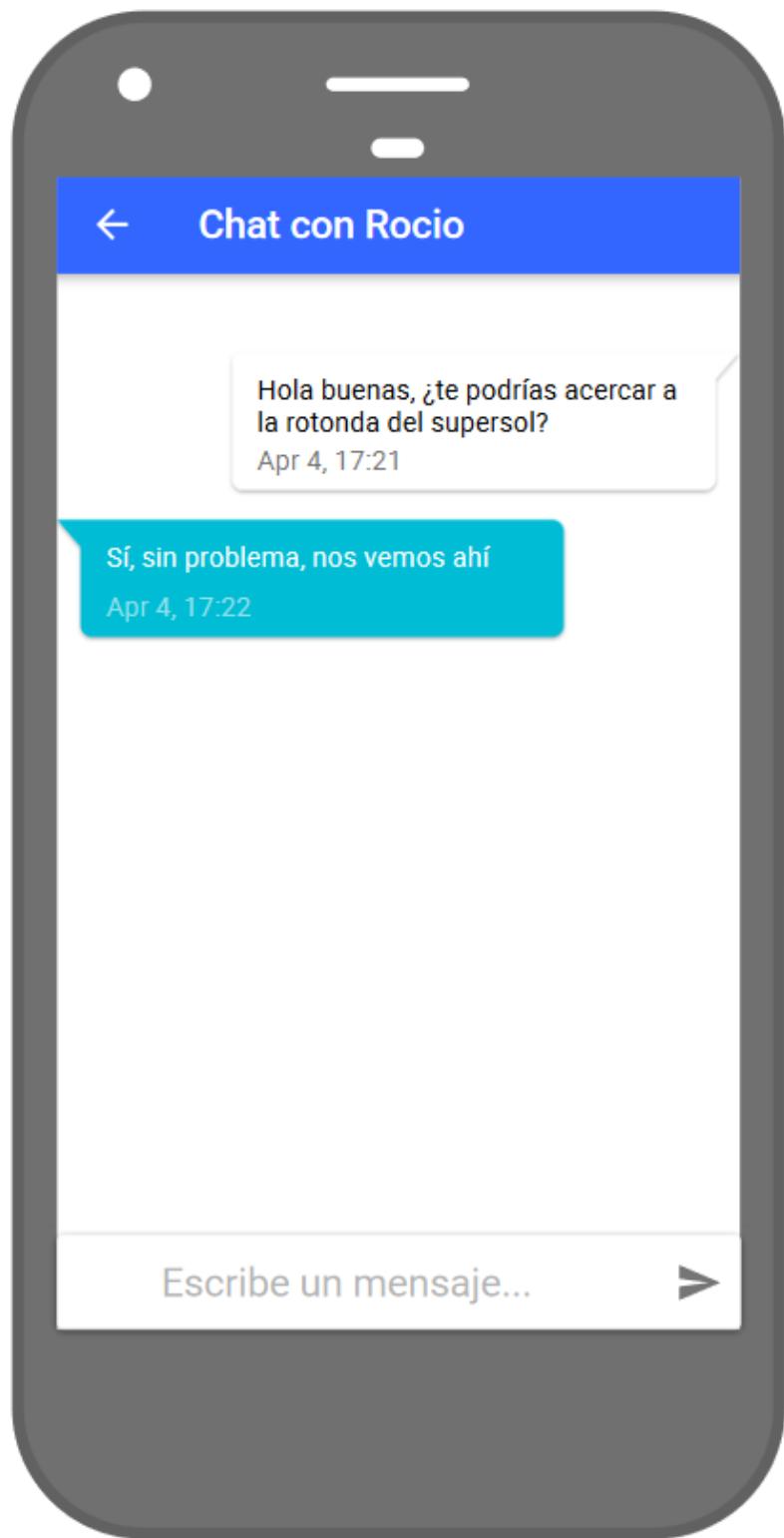


Figura 46: Maqueta de pantalla de chat concreto

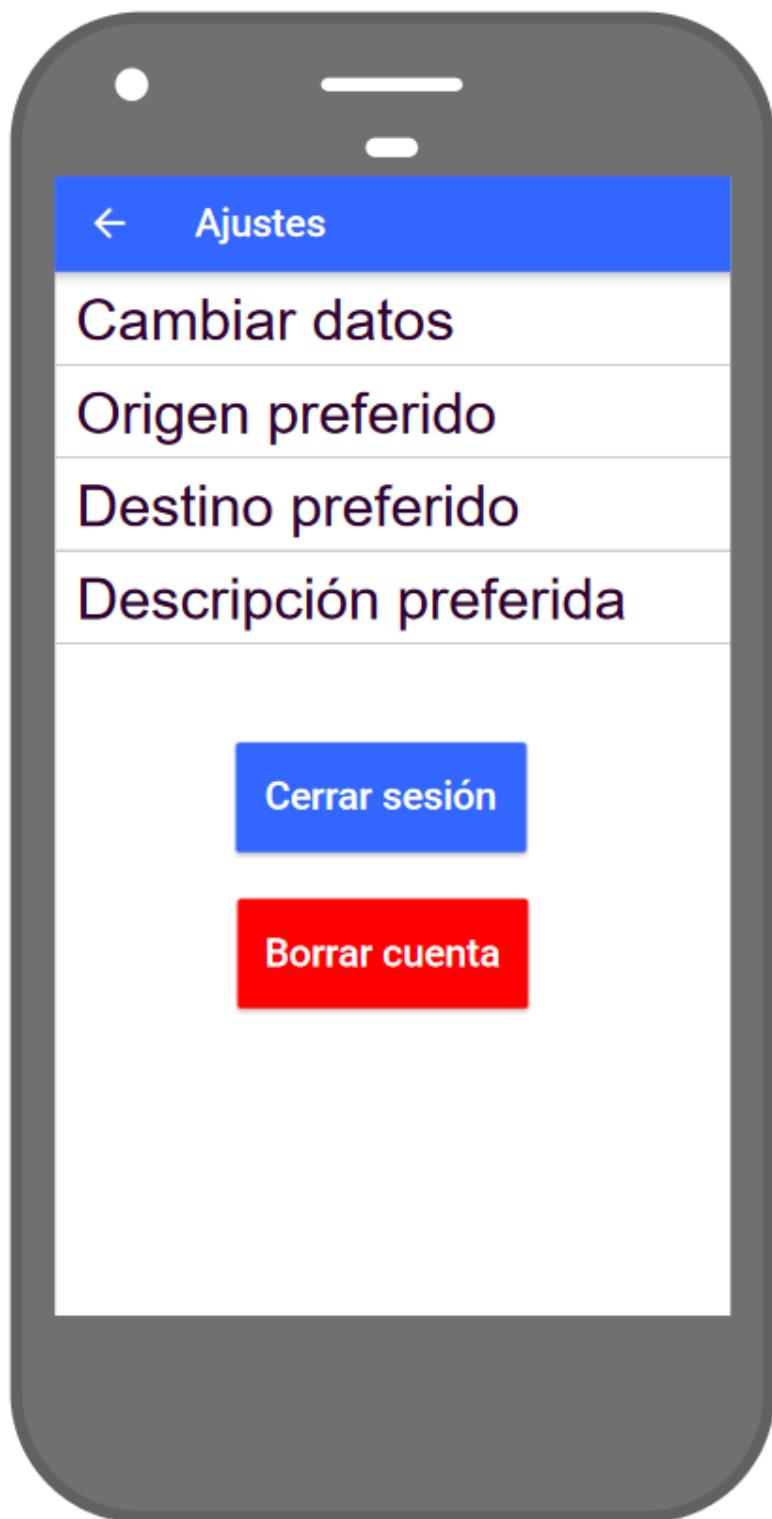


Figura 47: Maqueta de pantalla de configuración

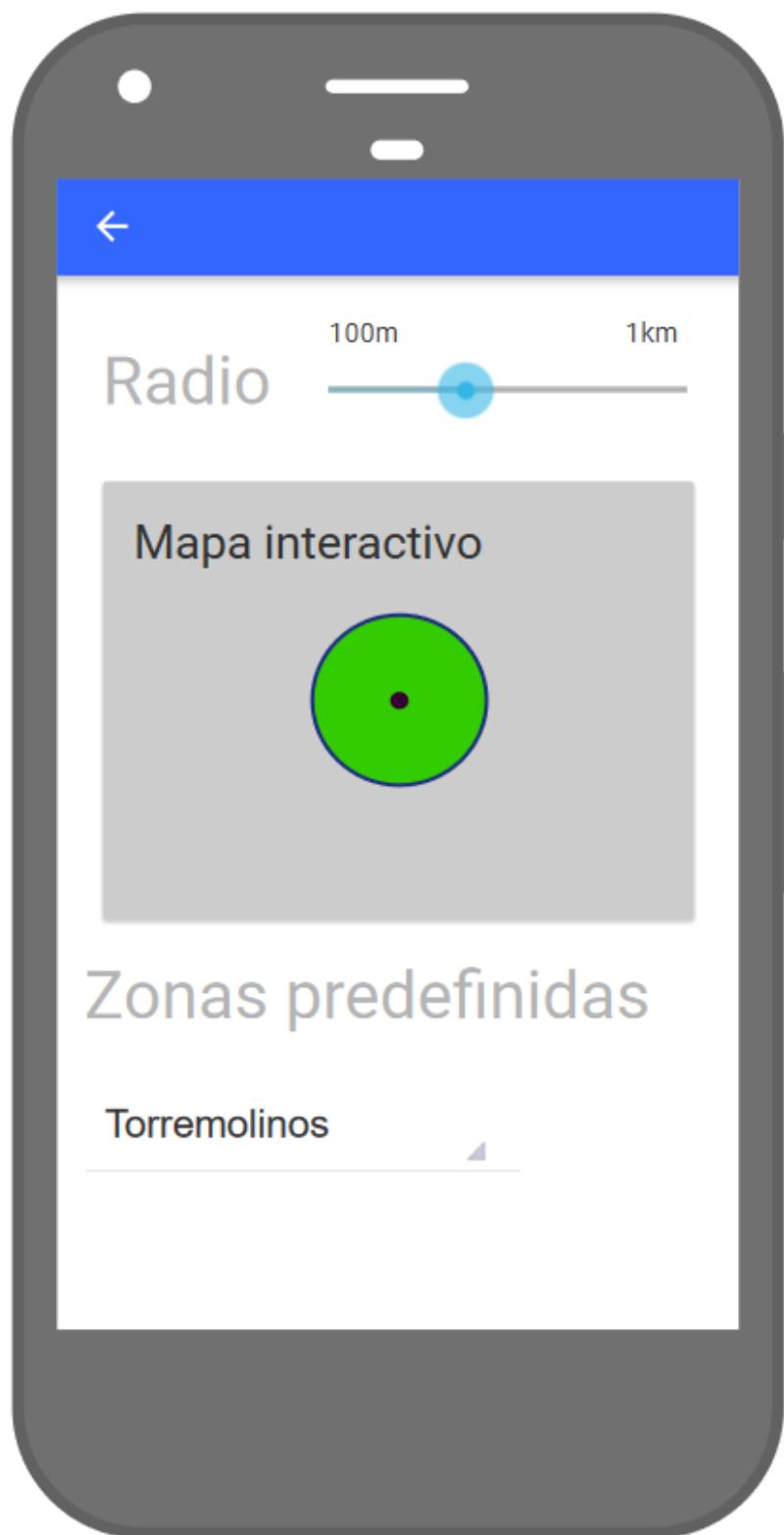


Figura 48: Maqueta de pantalla de posición preferida

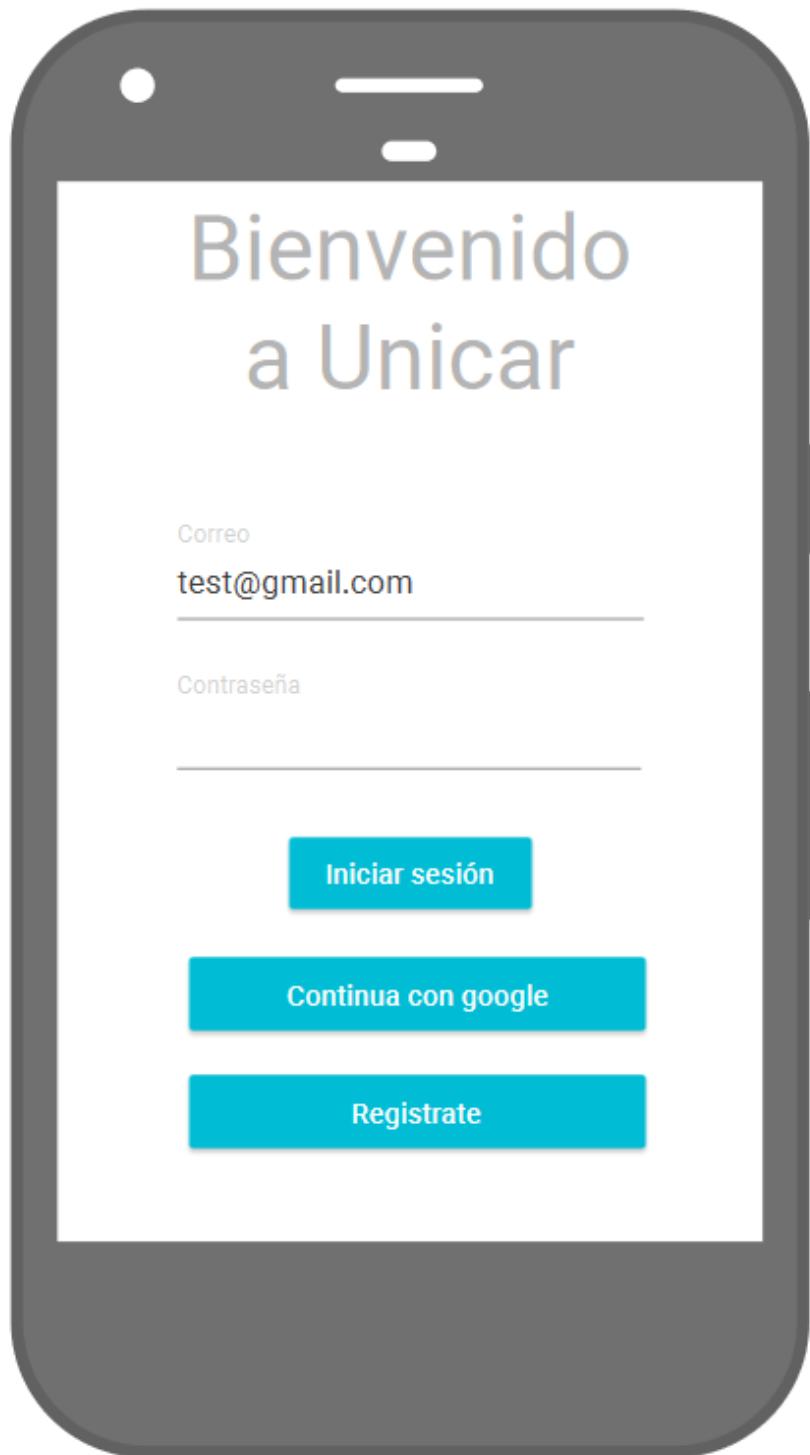


Figura 49: Maqueta de pantalla de iniciar sesión en la aplicación

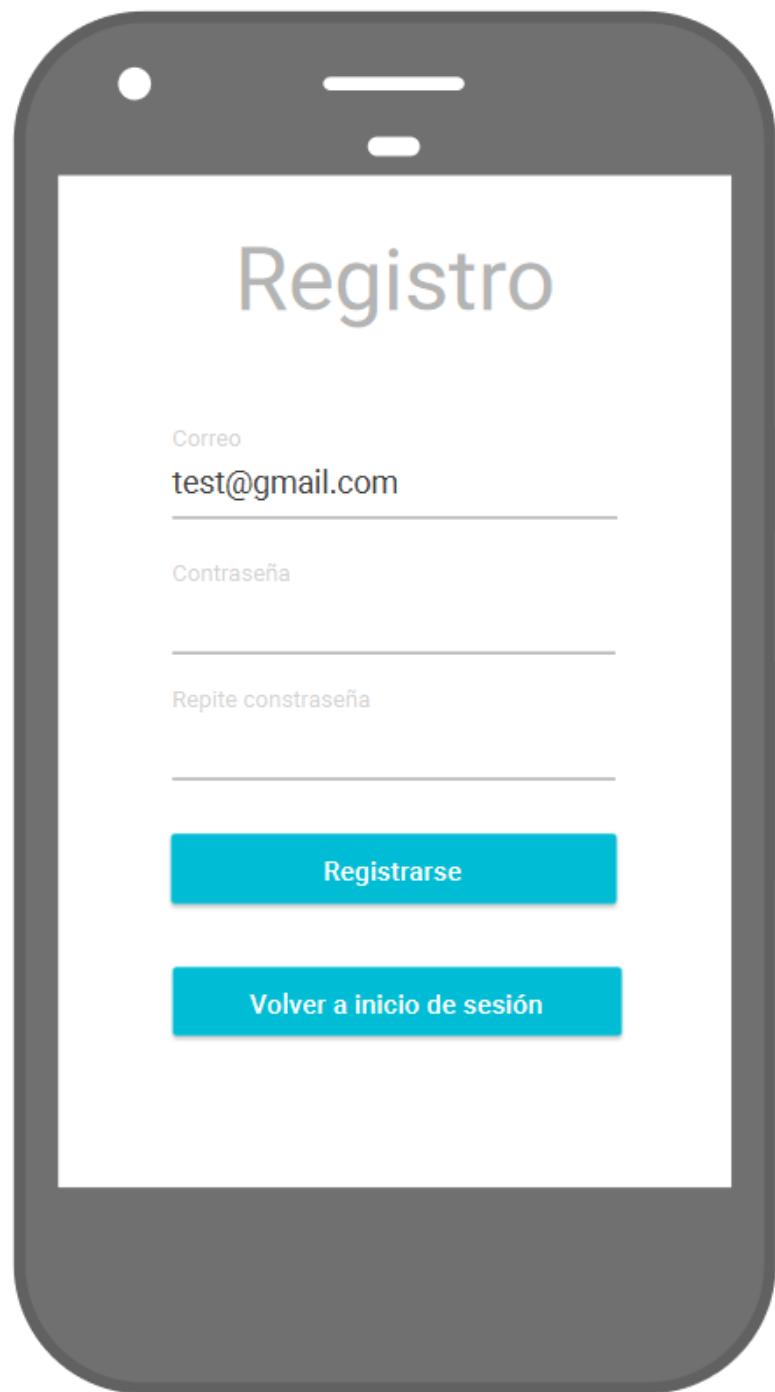


Figura 50: Maqueta de pantalla de registro en la aplicación

4

Desarrollo

En esta sección se explica el proceso de desarrollo de la aplicación seguido. Como se explica en [Metodología de trabajo](#) se ha seguido una metodología ágil basada en Scrum, donde se han completado todos los requisitos propuestos y las pruebas unitarias en cuatro Sprints de dos semanas. La herramienta usada para gestionar el trabajo ha sido [Trello](#) y para el desarrollo se ha usado [Visual studio code](#). Se ha dado una prioridad a todos los requisitos que en los tableros de Trello se representa con los colores rojo, naranja y amarillo para prioridad alta, media y baja respectivamente.

4.1. Sprint 1

En este primer Sprint en acuerdo con el cliente se ha desarrollado una primera versión de las funcionalidades relacionadas con los viajes y una primera versión de la interfaz. En la figura 51 pueden verse los requisitos a desarrollar inicialmente.

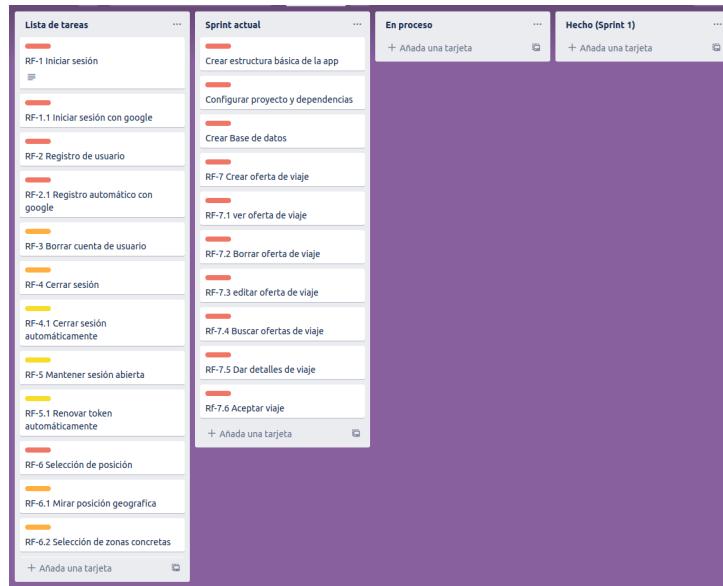


Figura 51: Captura de Trello al inicio del primer Sprint.

Para comenzar el desarrollo como tal lo primero que ha sido necesario es configurar el entorno de desarrollo y crear la base de datos.

Primero se han descargado e instalado las herramientas y las extensiones necesarias para trabajar con **Dart** y **Flutter**. Después se ha creado el proyecto base usando el comando “flutter create unicar” siendo “unicar” el nombre del proyecto. Con el proyecto creado se ha abierto el archivo pubspec.yaml para añadir algunos paquetes externos para facilitar y acelerar el desarrollo. Los paquetes usados son los siguientes:

- **supabase_flutter:** librería de cliente para facilitar el uso de las funciones de Supabase [32].
- **flutter_riverpod:** librería de gestión de estado de objetos y enlace de datos [33].
- **riverpod_annotation:** anotaciones de riverpod [34].

- **image_picker:** librería para facilitar la implementación de selección de imágenes [35].
- **date_time_picker:** librería para facilitar la selección de una fecha y una hora [36].
- **string_validator:** librería para facilitar la validación de cadenas de texto [37] .
- **intl:** librería para internacionalización, localización y formateo de fechas y horas [38]. Se ha usado el formateo de fecha y hora.
- **geocoding:** librería para facilitar el uso de los servicios gratuitos de geocoding de Android e IOS [39].
- **geolocator:** librería para facilitar el uso de los servicios de localización de Android e IOS. [40].
- **flutter_map:** librería para crear mapas con Flutter [41].
- **latlong2:** librería que usa flutter_map para colocar puntos en los mapas[42].
- **toggle_switch:** librería que implementa diversos botones y permite gran personalización [43].
- **shared_preferences:** librería que simplifica guardar datos en memoria persistente [44].
- **permission_handler:** librería para facilitar la comprobación de permisos dados por el usuario y pedirlos [45].

Por último, se ha descargado **Android Studio** y configurado el servicio de dispositivos virtuales para poder compilar la aplicación y probarla rápidamente. Con esto hecho el entorno de desarrollo está listo.

Por otro lado, es necesario también crear la base de datos. Para ello ha sido necesario crear una cuenta de Supabase y con ella ha sido posible desde la interfaz crear las tablas según el diseño del **Diagrama de base de datos**. Ha sido necesario realizar algunas configuraciones más, para consultarlas acuda a la sección **Otras configuraciones en Supabase**.

Las funcionalidades relacionadas con las ofertas se han implementado con proveedores con clases de controlador. Inicialmente se ha hecho una implementación simple que solo recoge los datos de la base de datos y filtra para mostrar mostrar cada viaje en la sección en la que debe aparecer. El problema de esta implementación es que si se hace una reserva de plaza u otra operación esta no se ve reflejada en la interfaz de usuario. Para solucionar esto en vez de solo recoger los datos y mostrarlos se ha hecho un controlador para cada lista de ofertas. Estas listas se inicializan de manera asíncrona y una vez esta todo inicializado si el usuario hace cualquier operación la información se actualiza en la base de datos y localmente en la interfaz de usuario.

En una reunión con el cliente se ha llegado al acuerdo de añadir a este Sprint las funcionalidades relacionadas con los filtros debido a su alta relación con los viajes. Los filtros se han implementado como una función en el controlador de las ofertas de viaje.

Finalmente se pudieron implementar todos los requisitos propuestos este Sprint. En la figura [Captura de Trello al final del primer Sprint](#). puede verse el tablero de Trello al finalizar. También se han incluido capturas de la aplicación para ver su estado al final de este Sprint. Para ver el código desarrollado hasta este punto puede dirigirse al commit con el SHA a6570f1a7bac9fb039cec3ffde55d9bc2d3cece3.

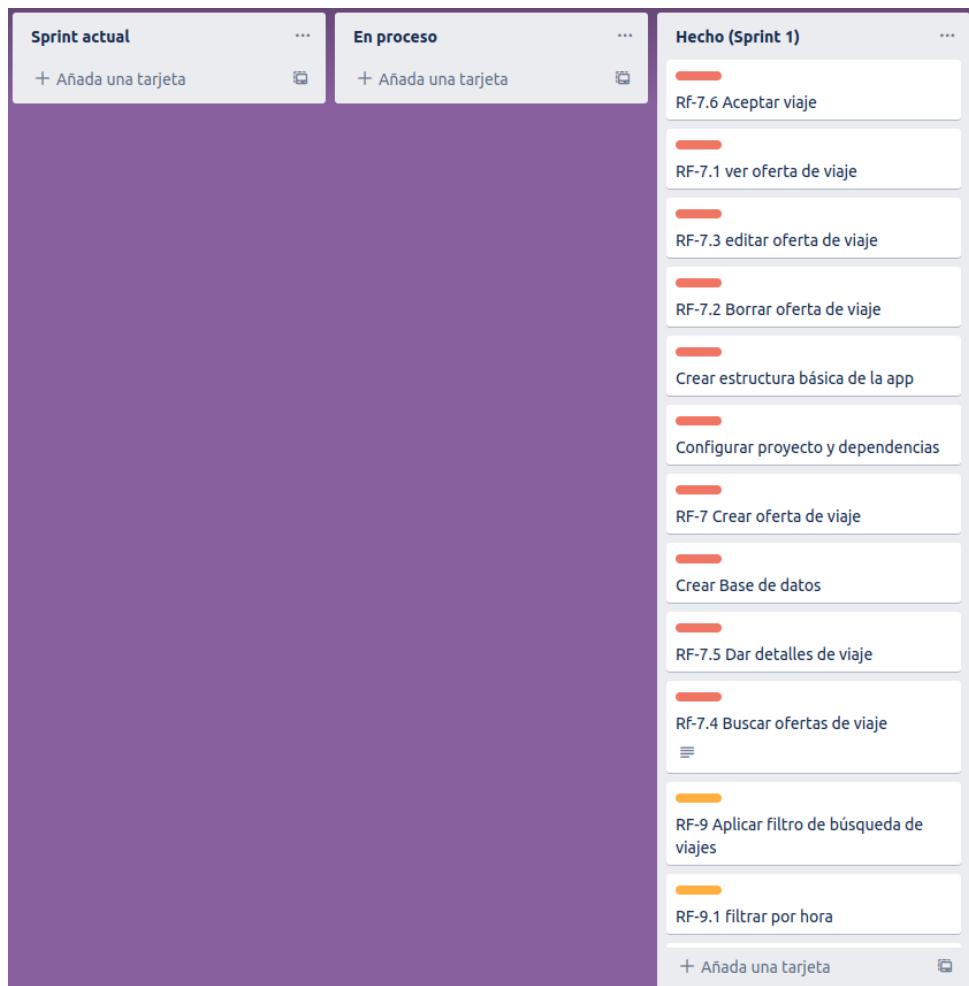
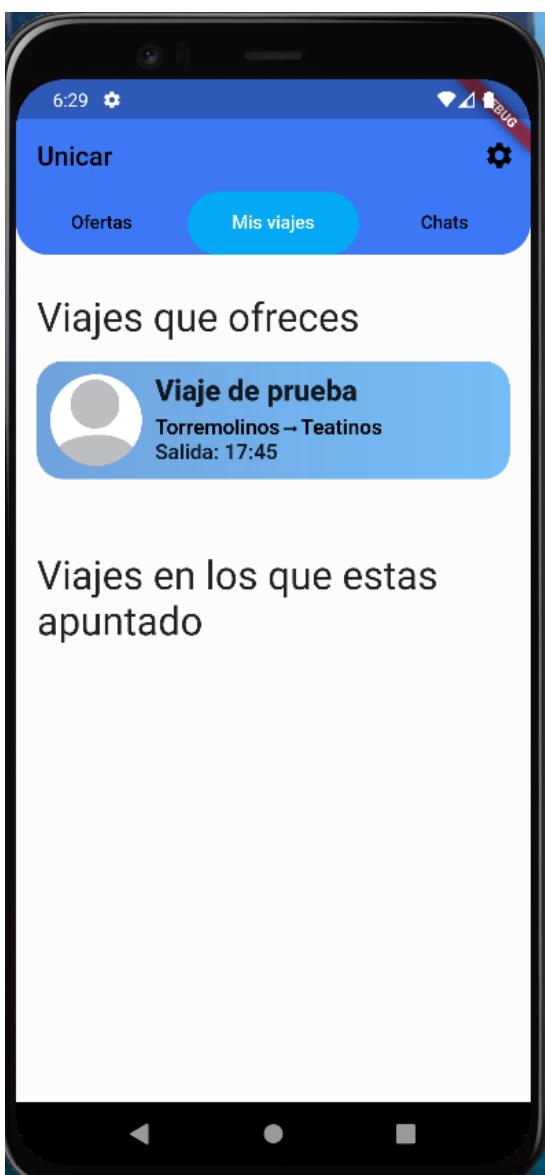


Figura 52: Captura de Trello al final del primer Sprint.



(a)



(b)

Figura 53: En (a) se puede ver la pantalla de “Mis viajes“ y en (b) la de Ofertas.

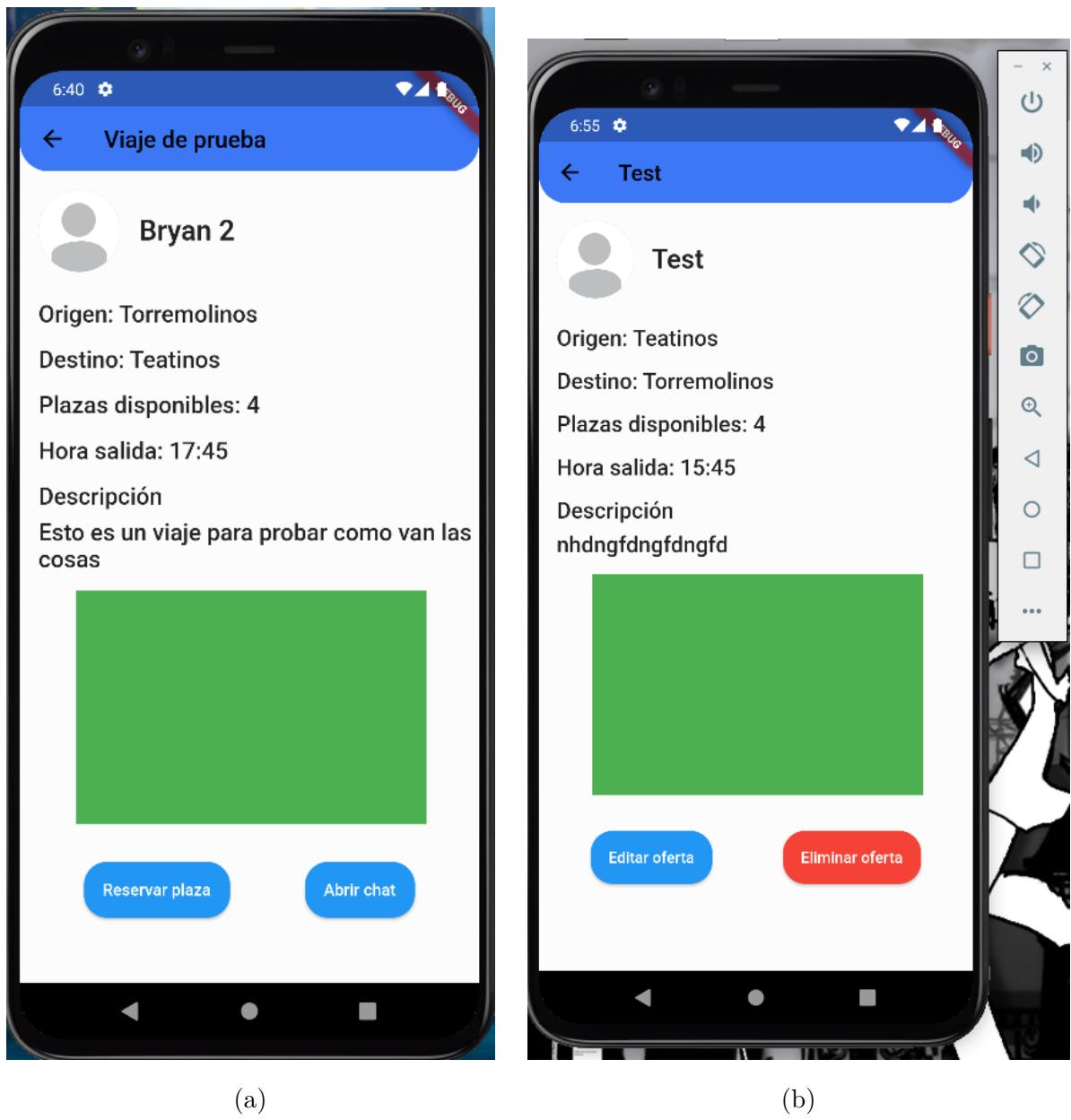
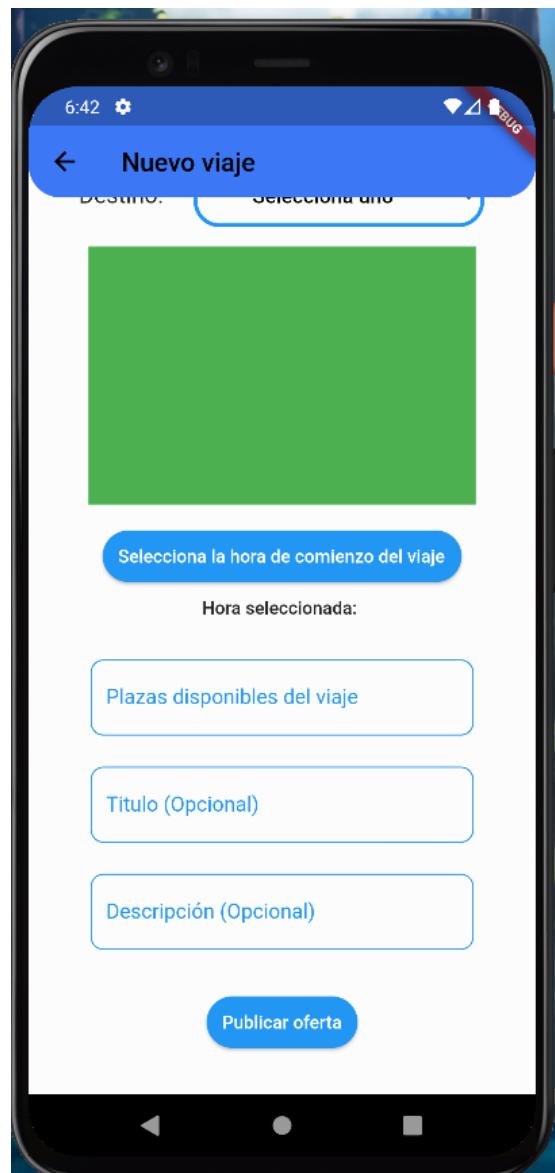


Figura 54: En (a) se puede ver la pantalla de un viaje que no ha creado el usuario que la esta viendo y en (b) la pantalla de un viaje que ha creado el usuario. El cuadro verde es un objeto temporal hasta la implementación de los mapas.



(a)



(b)

Figura 55: Pantalla de crear una nueva oferta. El cuadro verde es un objeto temporal hasta la implementación de los mapas.



Figura 56: Pantalla de filtros. El cuadro verde es un objeto temporal hasta la implementación de los mapas.

4.2. Sprint 2

En este Sprint se ha llegado al acuerdo de implementar las funcionalidades relacionadas con la gestión de usuarios. Se ha hecho esto porque con ello la aplicación ya podría ser usada, aunque con funcionalidad limitada. En la figura [Captura de Trello al inicio del segundo Sprint](#). pueden verse el tablero de Trello al principio del Sprint.

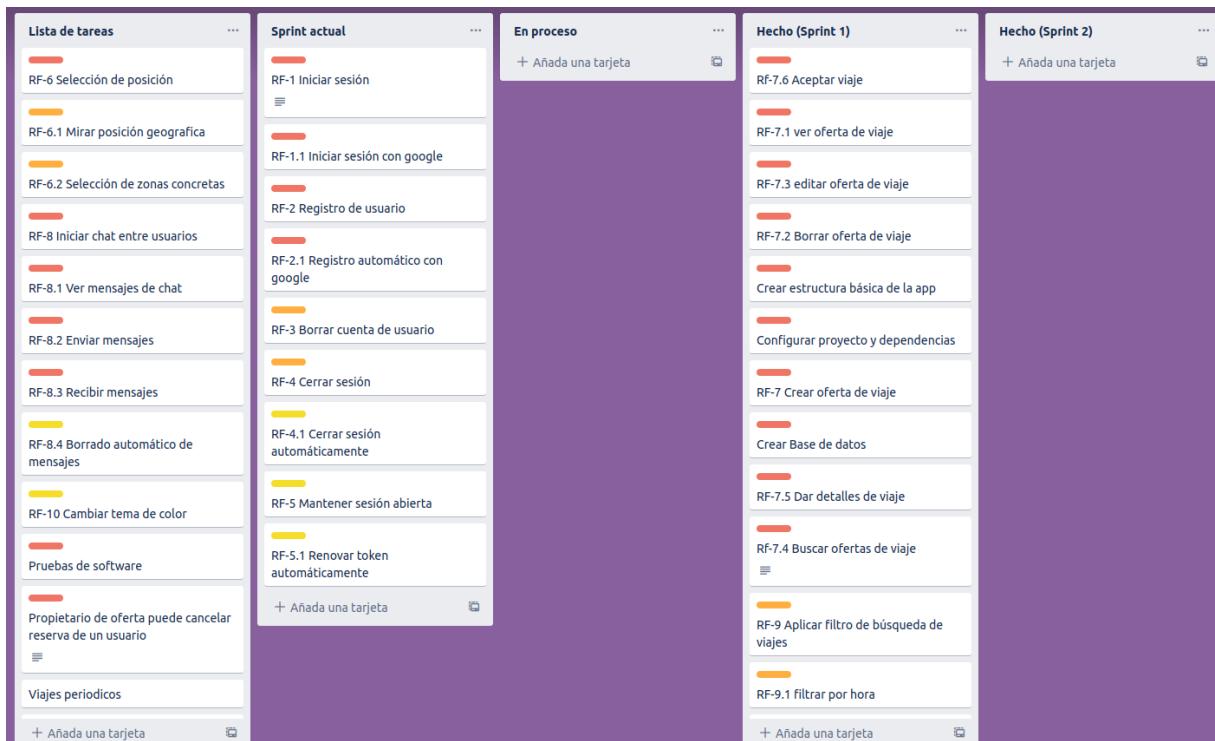


Figura 57: Captura de Trello al inicio del segundo Sprint.

Antes de empezar con los requisitos del Sprint se han refactorizado partes del código de la interfaz de usuario y de partes de la lógica de la aplicación para hacer que sea más sencillo hacer futuros cambios.

Una vez hecho esto se ha comenzado a implementar el inicio de sesión y el registro. Para ello se ha usado el Auth de [Supabase](#). Para esto ha habido que configurar un trigger para añadir los datos a la [Tabla usuario](#) desde la [Tabla Auth.Users](#). Para ver los triggers desarrollados puede dirigirse a la sección [Triggers](#). Por otro lado para poder mantener la sesión abierta durante el mayor tiempo posible se ha actualizado el valor del tiempo de caducidad de un día a una semana.

En la parte de la aplicación la implementación con la librería de Supabase no ha

provocado grandes problemas ya que dispone de funciones para realizar la mayoría de acciones necesarias sobre una sesión de usuario. Lo único que no es proporcionado es una función para borrar una cuenta de usuario. Esto no es ofrecido por un tema de seguridad. Como no es ofrecido por la librería se ha implementado como una función en la base de datos. Para ver su definición puede ir a la sección [Funciones creadas](#).

En este Sprint se ha decidido cambiar los requisitos relacionados con Google por Discord. Esto se debe a que para usar el inicio de sesión con Google era necesario introducir datos bancarios y se ha decidido con el cliente no tomar dicho riesgo en este proyecto. Se ha usado Discord como sustituto por ser gratuito y de fácil uso. Para ver el código desarrollado hasta este punto puede dirigirse al commit con el SHA 0875adbf2c72222c77044b6c963c0ec87e673322.

En las siguientes figuras pueden verse capturas de la aplicación al final del Sprint y el tablero de Trello.

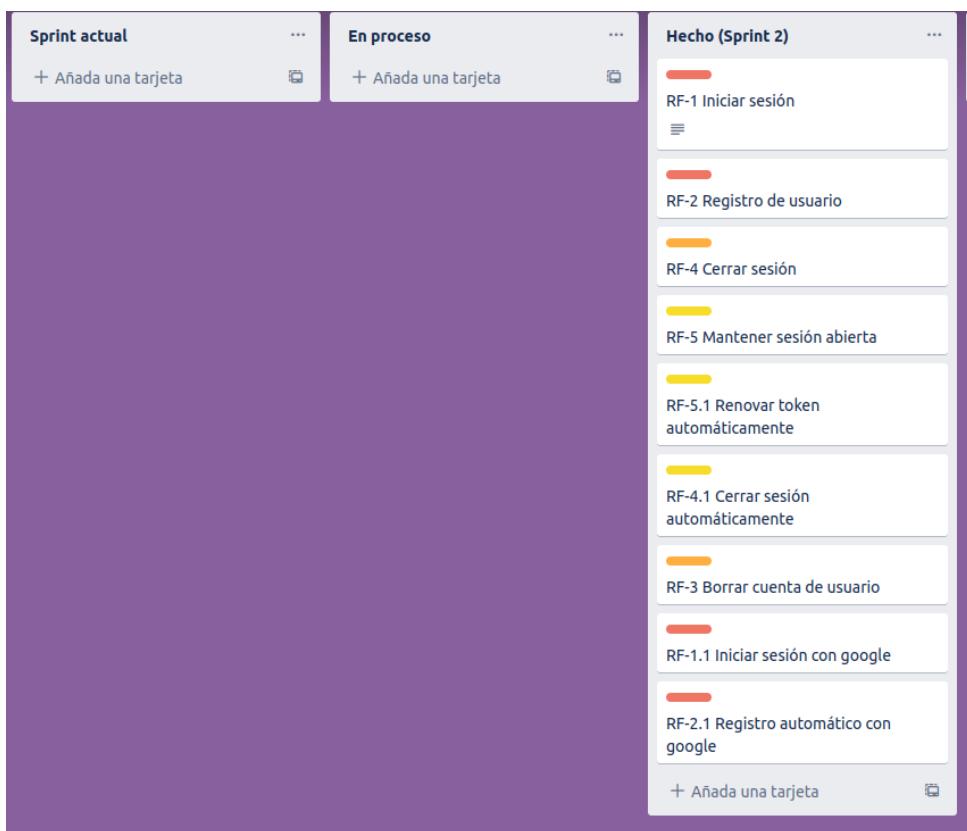
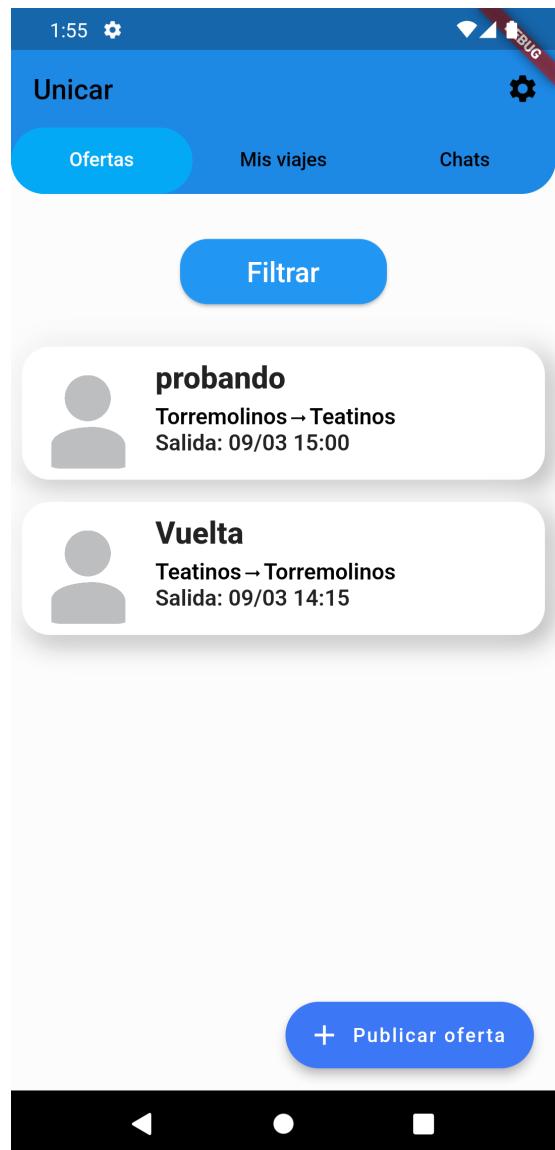


Figura 58: Captura de Trello al final del segundo Sprint.

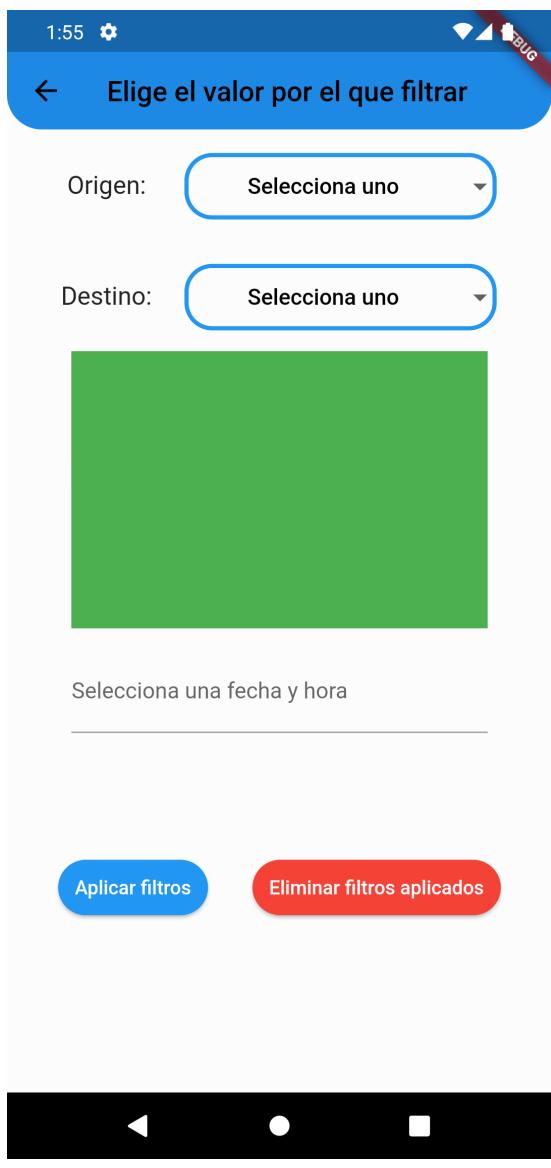


(a)

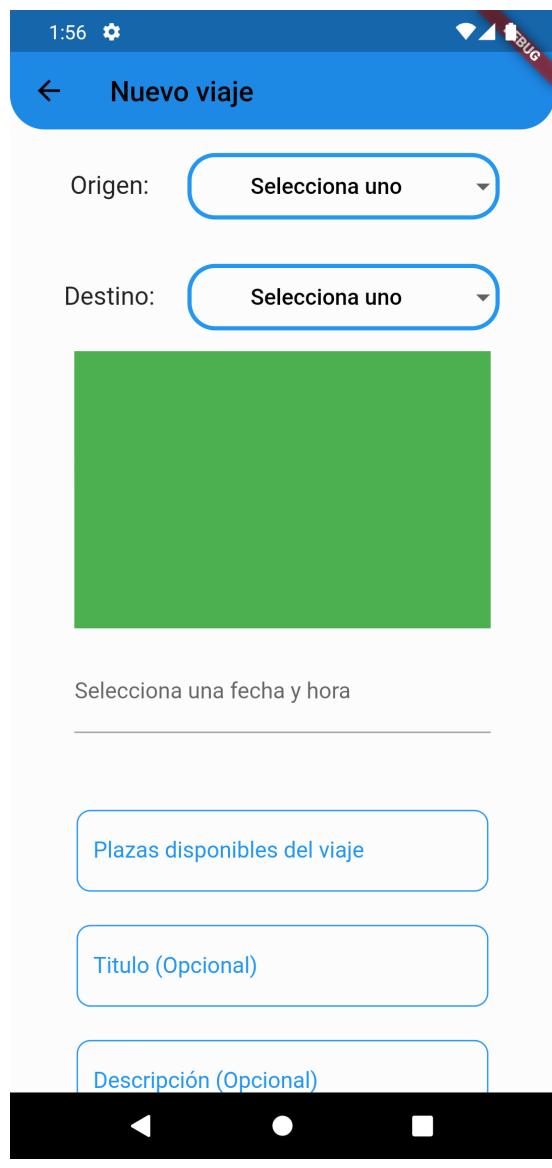


(b)

Figura 59: En (a) pantalla de “Mis viajes”, en (b) pantalla de ofertas.

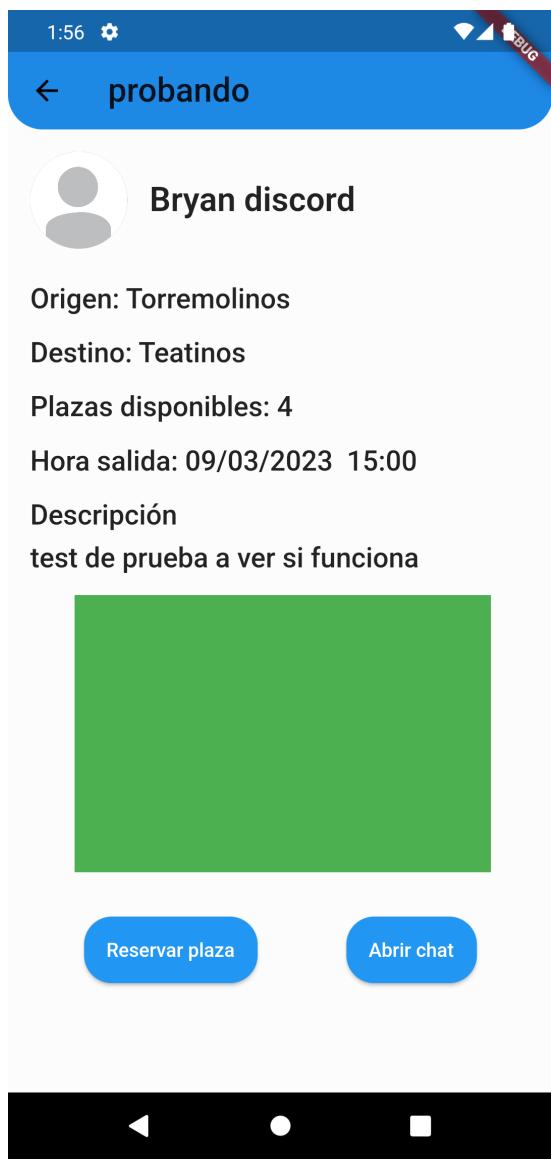


(a)



(b)

Figura 60: En (a) pantalla de filtros, en (b) pantalla de crear oferta.

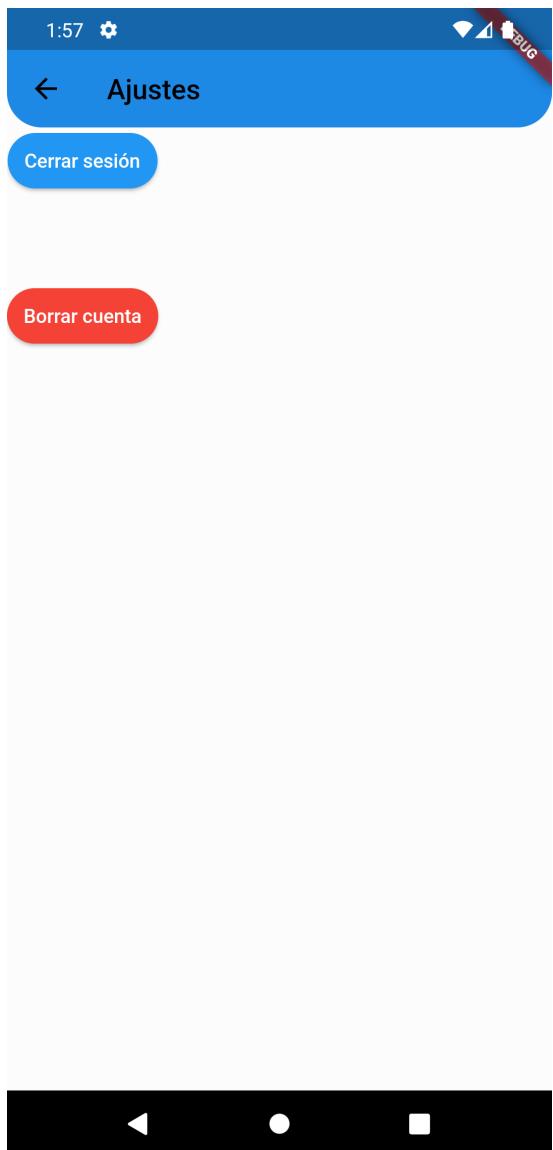


(a)

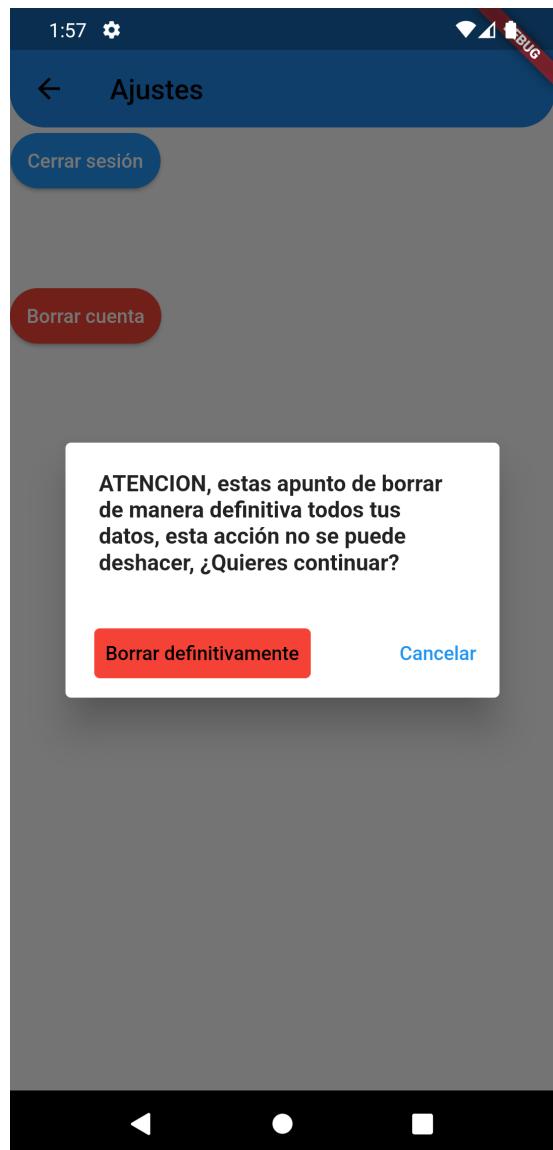


(b)

Figura 61: En (a) pantalla de viaje que el usuario no ha creado, en (b) pantalla de viaje que el usuario ha creado.



(a)



(b)

Figura 62: En (a) pantalla de configuración, en (b) mensaje que aparece al pulsar el botón de borrar cuenta.

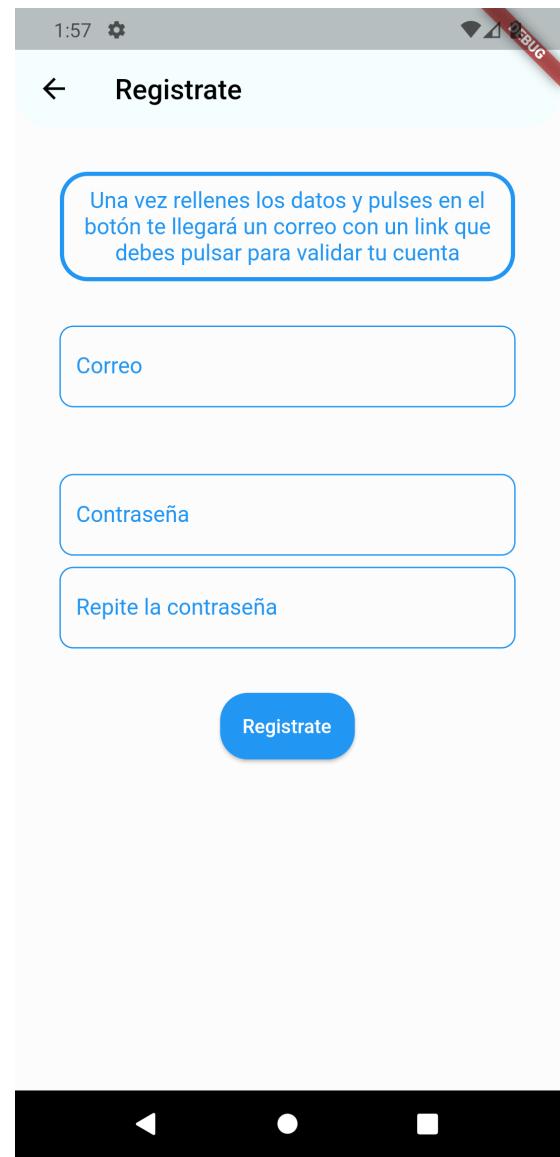


Figura 63: En (a) pantalla de inicio de sesión, en (b) pantalla de registro.

4.3. Sprint 3

En este Sprint se ha desarrollado las funciones relacionadas con el Chat y los mapas.

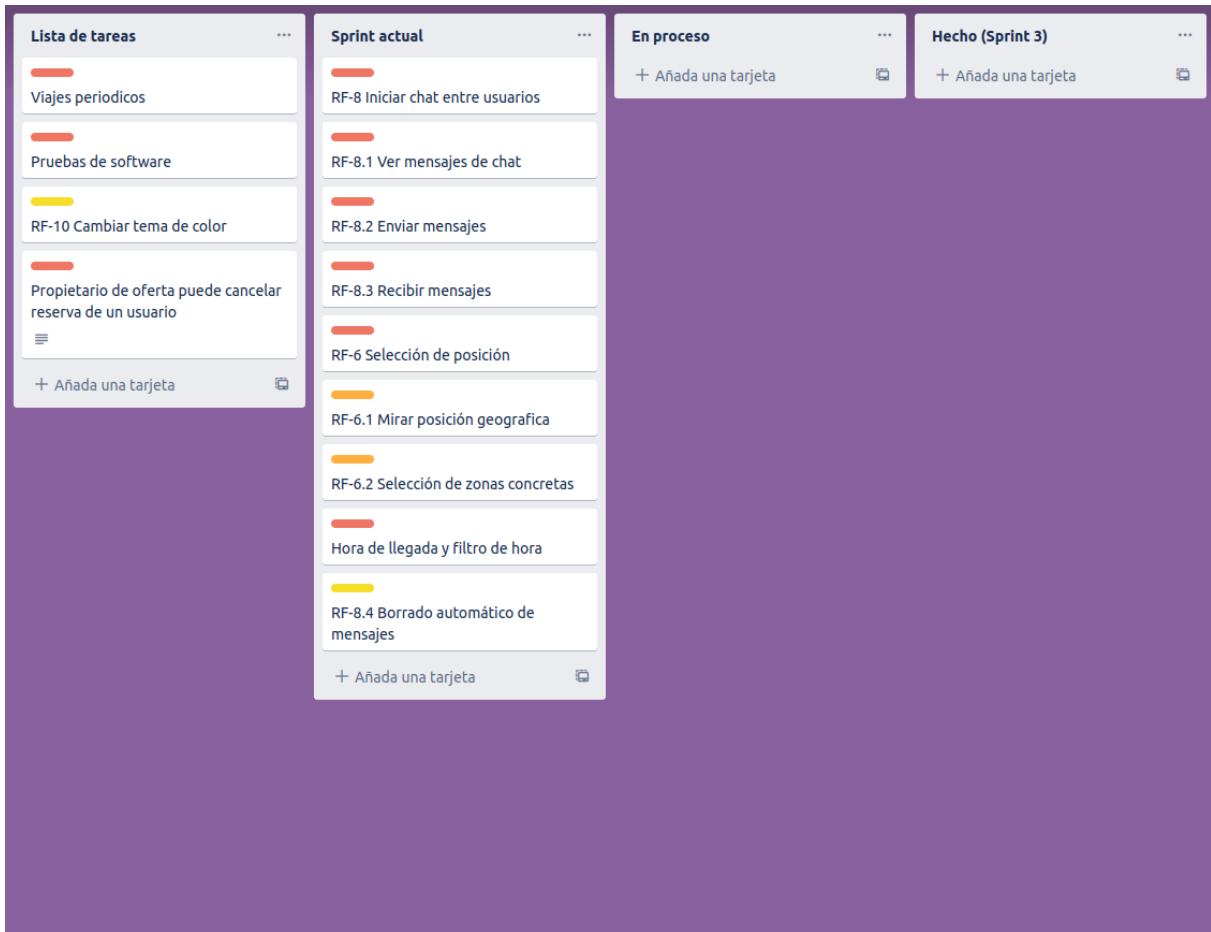


Figura 64: Captura de Trello al inicio del tercer Sprint.

Al igual que en el Sprint anterior, se ha refactorizado código para solucionar errores encontrados y facilitar cambios en futuras versiones.

Para implementar los chats se ha usado la funcionalidad de base de datos en tiempo real de Supabase. Dicha funcionalidad tiene que ser activada para las tablas que hagan uso de ella. En este caso solo es necesaria para la **Tabla mensaje** ya que se quiere que cada vez que haya un cambio en esta tabla los cambios se notifiquen a los usuarios que sean necesarios. Otra cosa que ha sido necesaria hacer en la base de datos es crear una función para crear un chat. Se ha hecho esta función en la base de datos para poder tener el id único que se crea y evitar problemas de inconsistencia. La función para crear una oferta de viaje también ha sido vuelta a implementar pero en la base de datos para evitar

inconsistencias y mejorar el rendimiento. Para ver estas funciones, acuda a [Funciones creadas](#).

Los mapas ya que se han implementado con la librería flutter_maps no han supuesto problemas. Se ha usado el servidor gratuito de OpenStreetMaps [46, 47]. Esta es una solución temporal con el propósito de demostrar que los mapas funcionan, en el caso de sacar la aplicación a producción sería conveniente usar un servicio de pago más confiable y sin grandes restricciones de políticas de uso. Para ver el código desarrollado hasta este punto puede dirigirse al commit con el SHA 382c25cdeb2fd410f8fd67d5e963270beaf045da.

A continuación se muestran algunas capturas del tablero de Trello y de la aplicación al final de este Sprint.

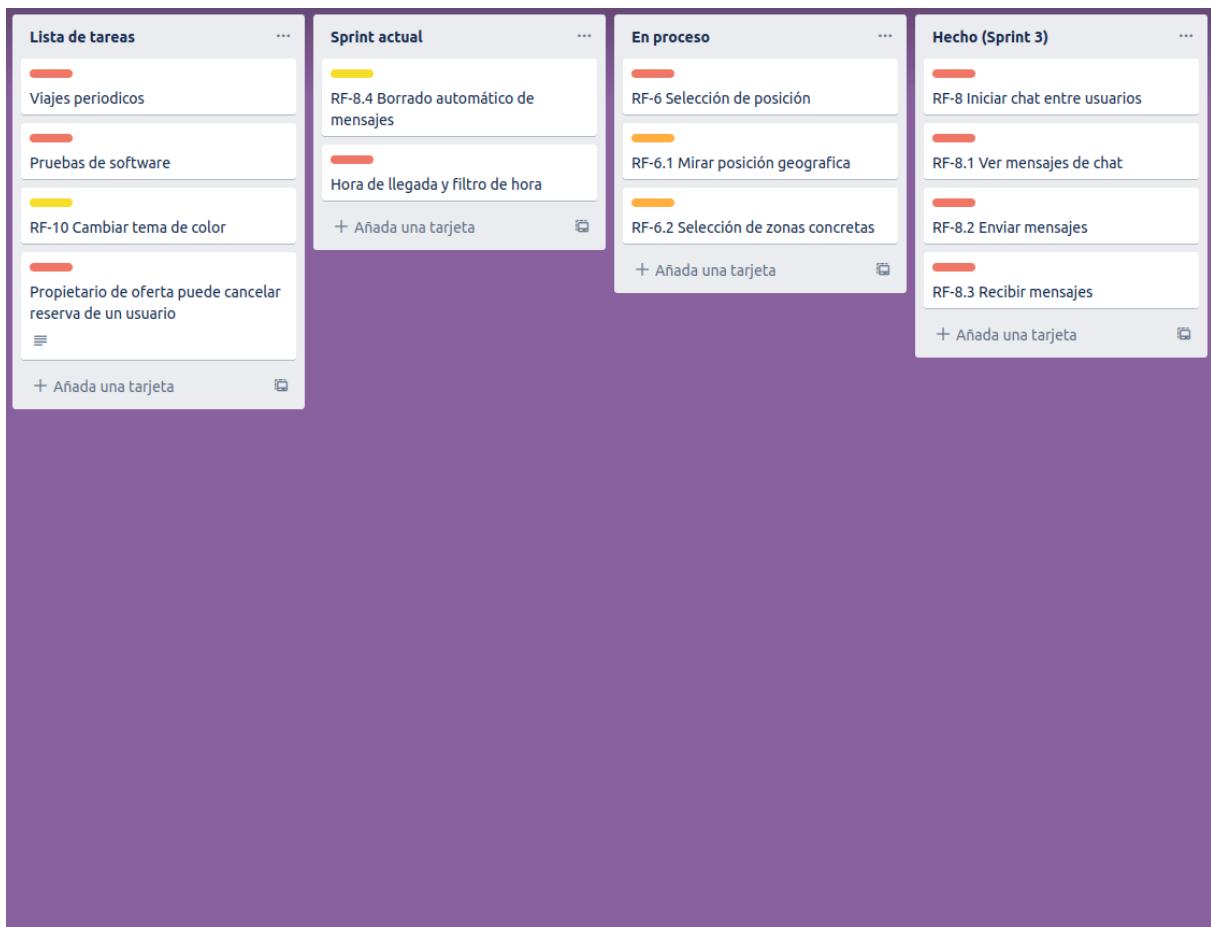


Figura 65: Captura de Trello al finalizar las funcionalidades de chat.

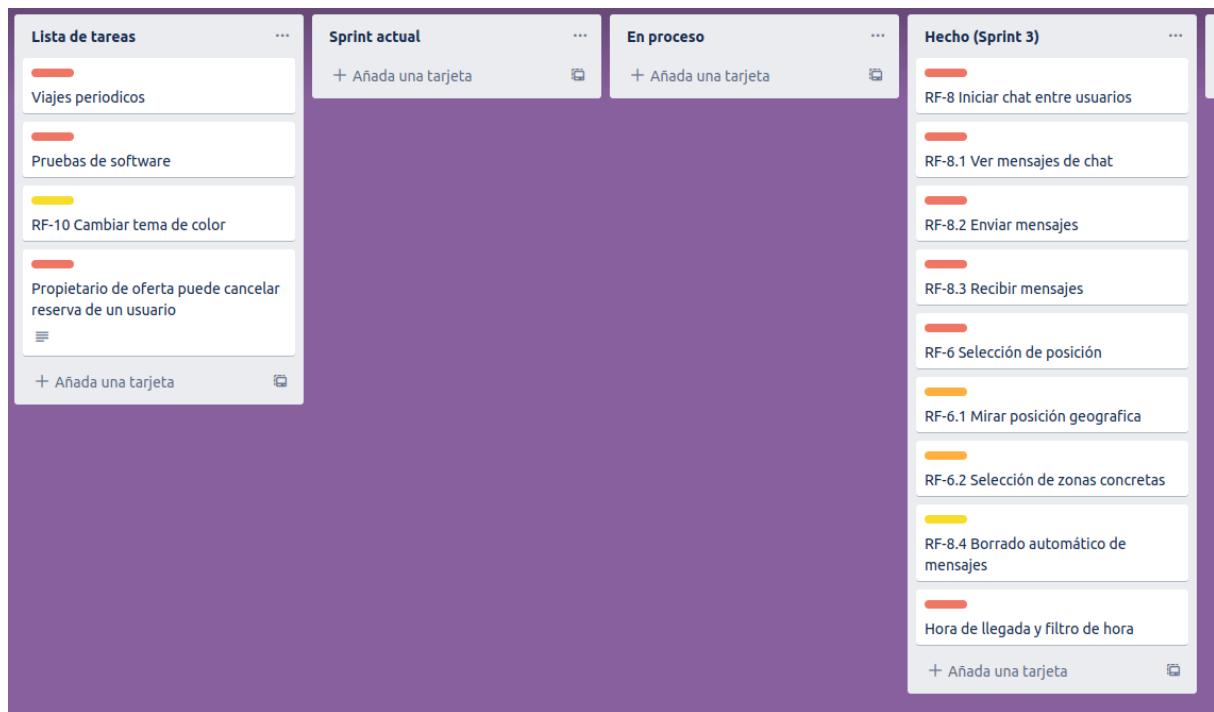
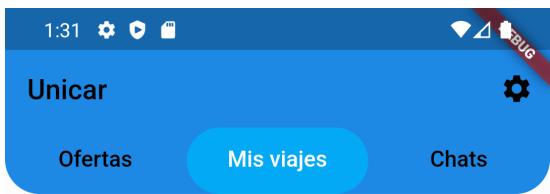


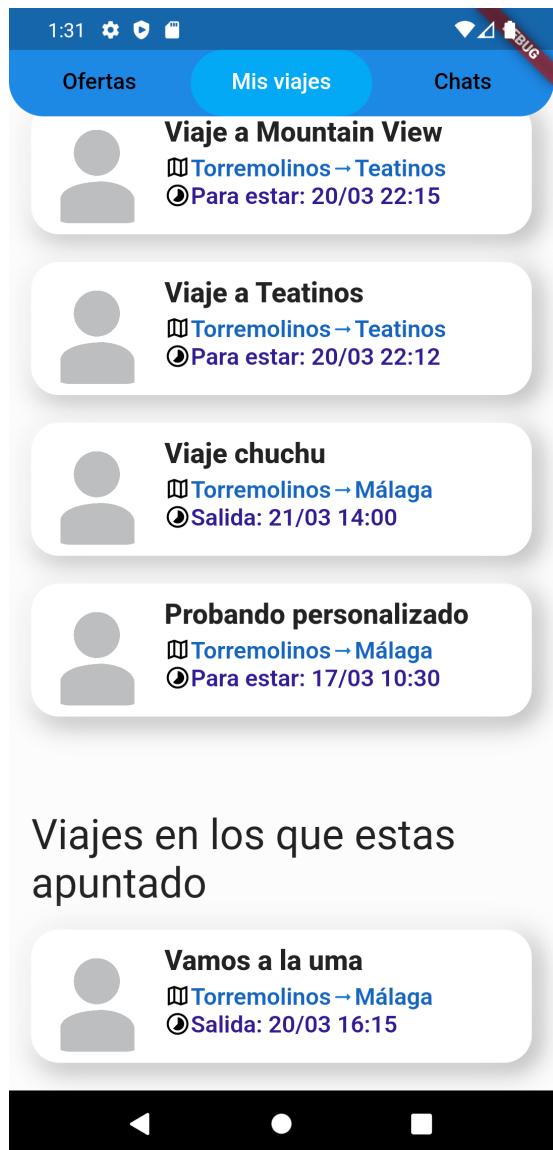
Figura 66: Captura de Trello al finalizar el Sprint 3.



Viajes que ofreces

- Viaje a Mountain View**
Torremolinos → Teatinos
Para estar: 20/03 22:15
- Viaje a Teatinos**
Torremolinos → Teatinos
Para estar: 20/03 22:12
- Viaje chuchu**
Torremolinos → Málaga
Salida: 21/03 14:00
- Probando personalizado**
Torremolinos → Málaga
Para estar: 17/03 10:30

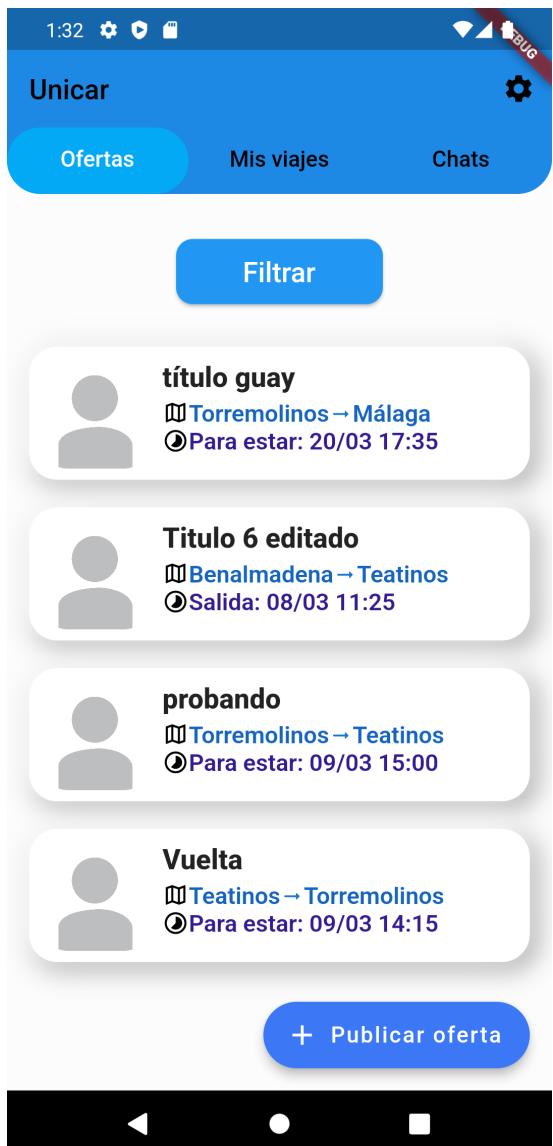
Viajes en los que estas apuntado



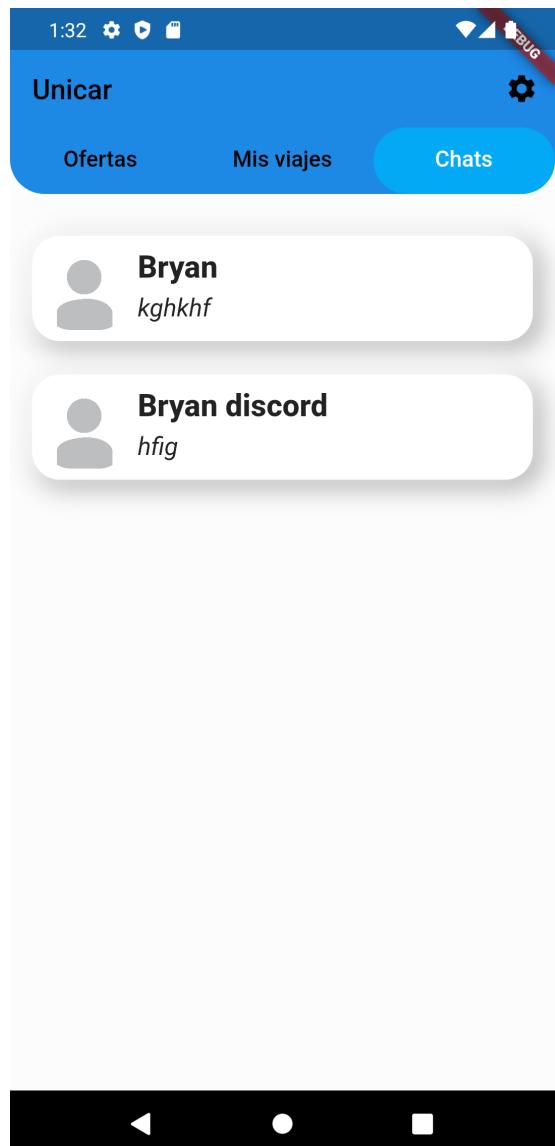
(a)

(b)

Figura 67: Pantalla de “Mis viajes”.

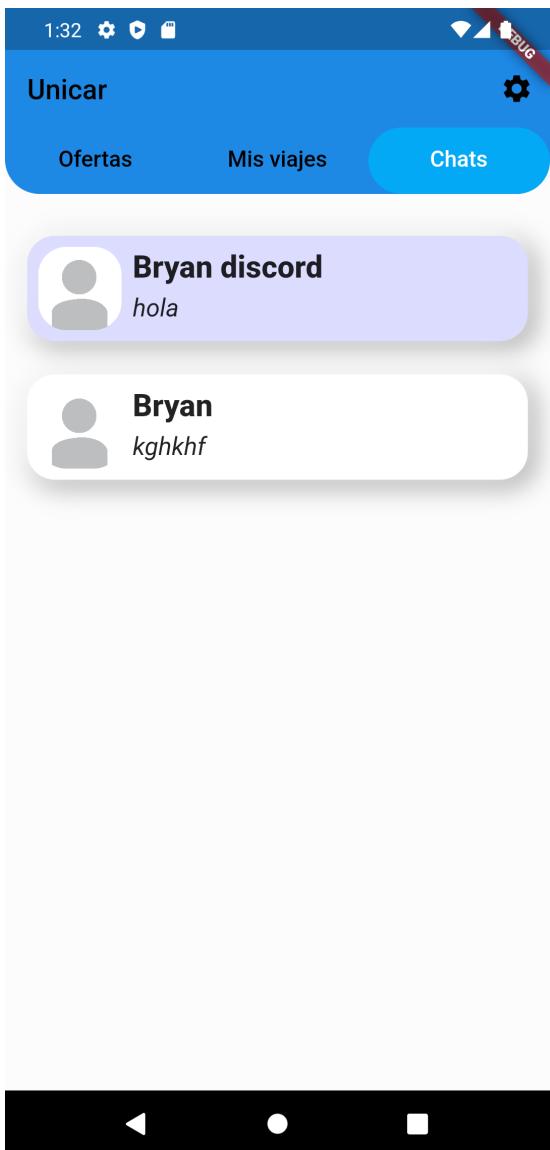


(a)

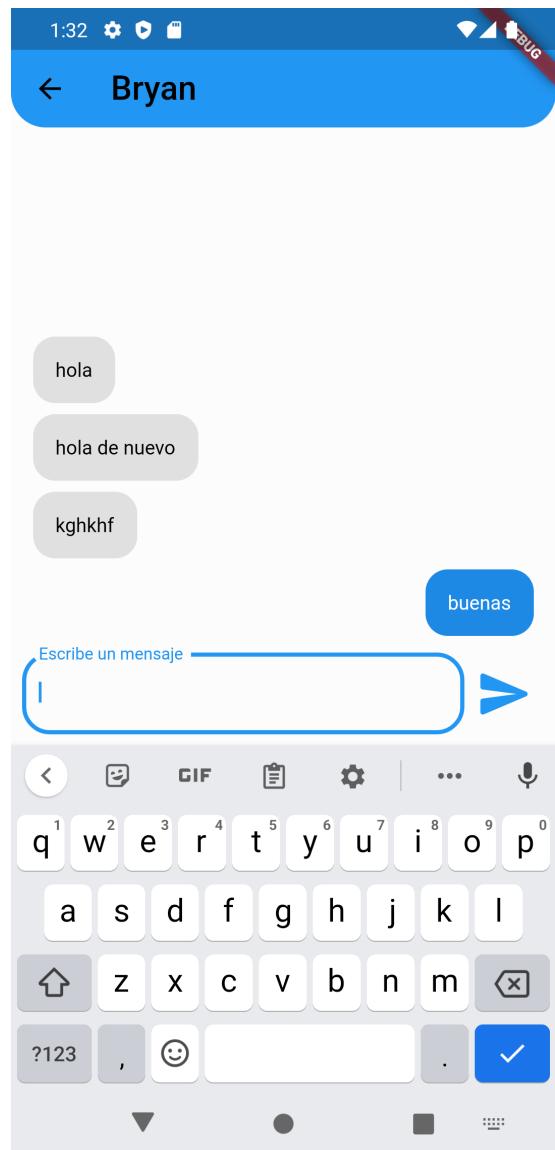


(b)

Figura 68: En (a) pantalla de Ofertas, en (b) pantalla de Chats.

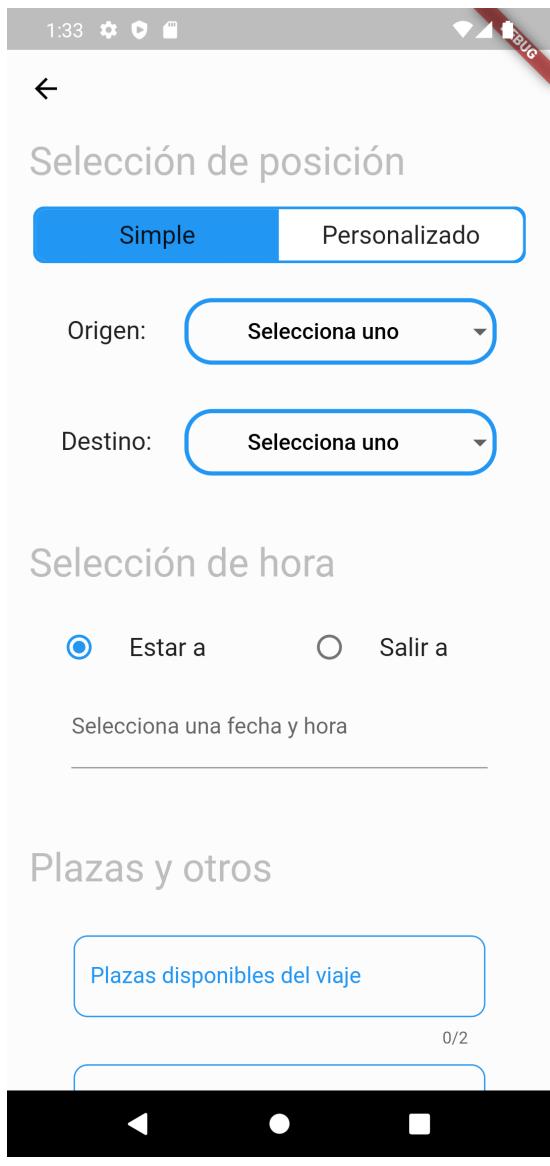


(a)

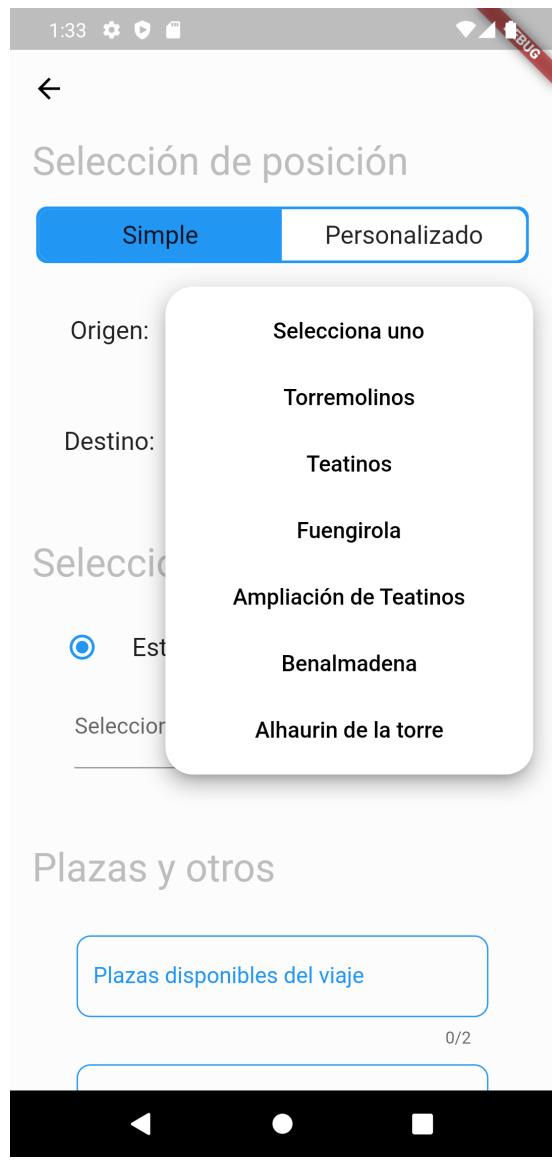


(b)

Figura 69: En (a) pantalla de Chat cuando hay un mensaje no leído, en (b) pantalla de chat concreto con mensajes.

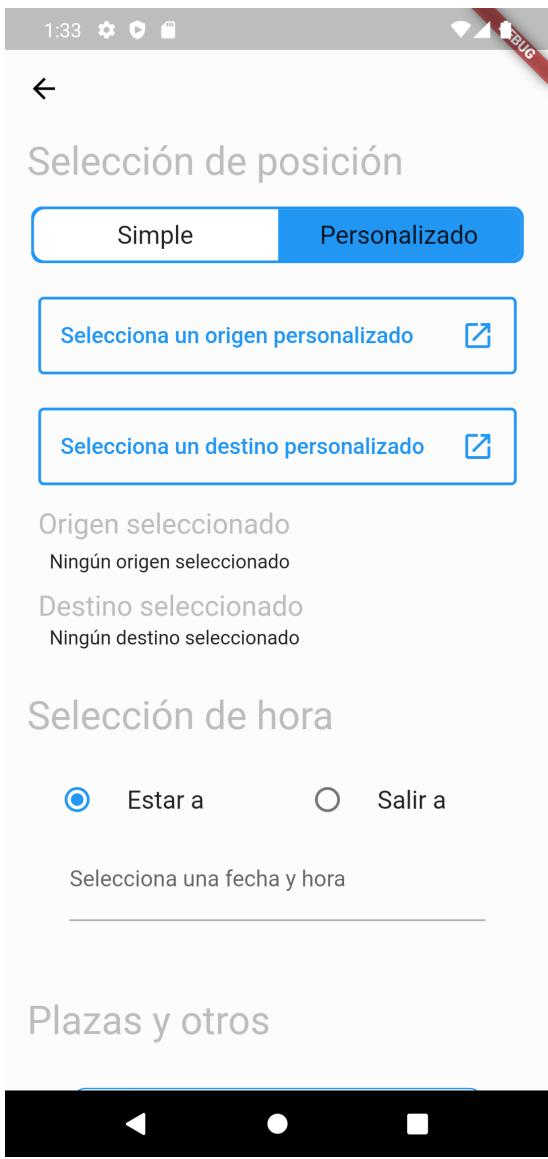


(a)



(b)

Figura 70: En (a) pantalla de crear oferta, en (b) pantalla de crear oferta con menú desplegable abierto.

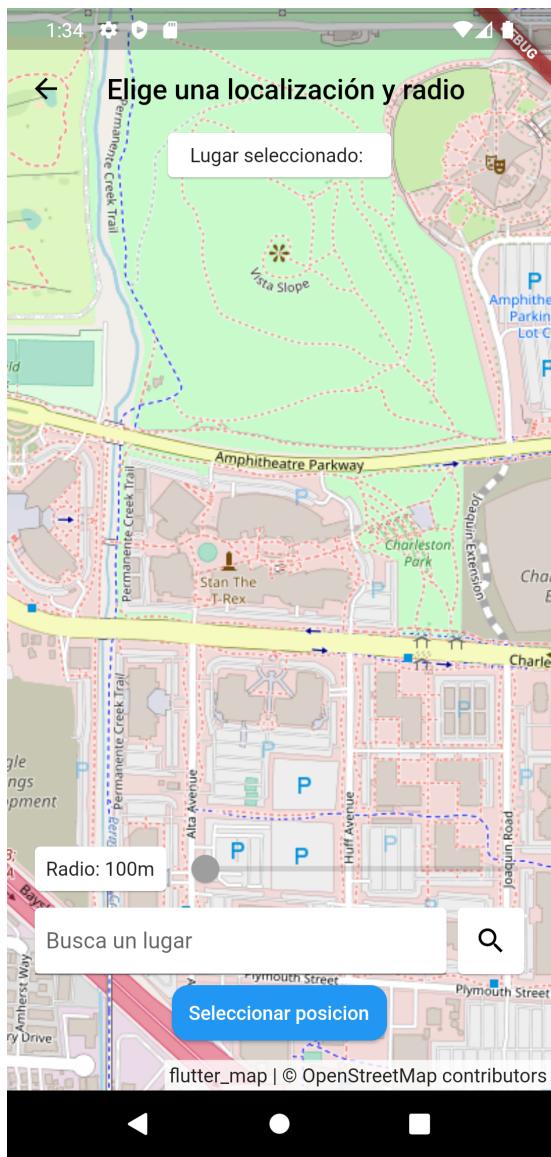


(a)

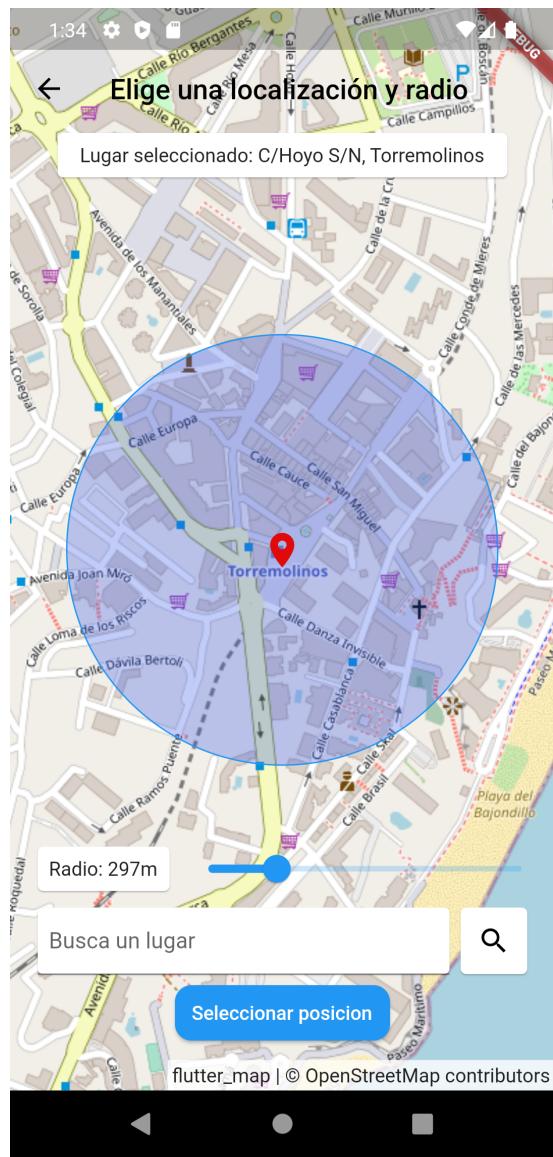


(b)

Figura 71: En (a) pantalla de crear oferta en selección de posición personalizada, en (b) pantalla de crear oferta, selección de hora y otros.

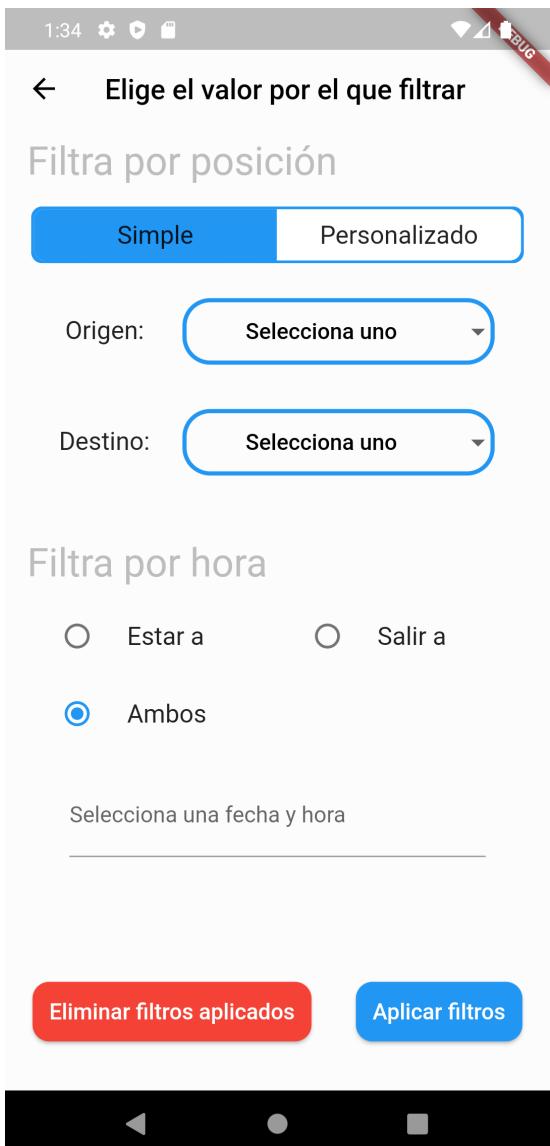


(a)



(b)

Figura 72: En (a) pantalla de mapa para seleccionar posición sin haber pulsado, en (b) pantalla de mapa para seleccionar posición donde el usuario ha pulsado.

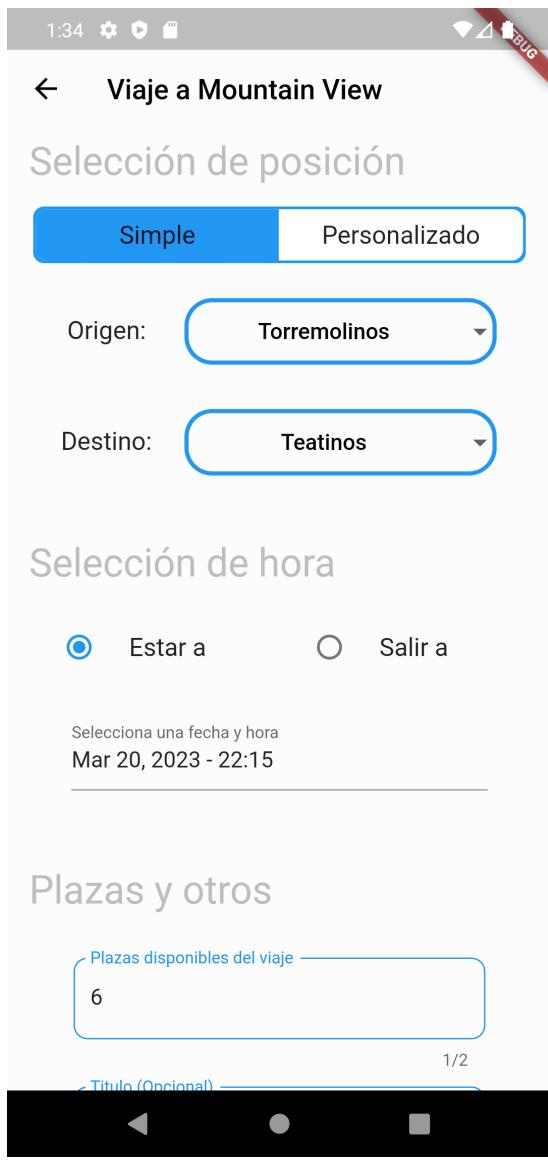


(a)

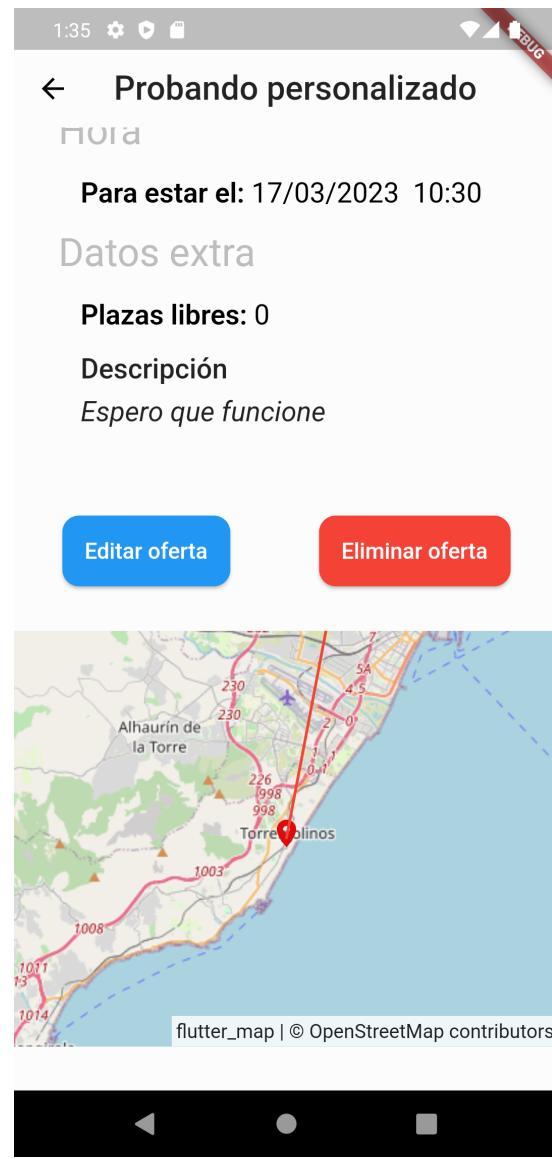


(b)

Figura 73: En (a) pantalla de filtros, en (b) pantalla de ver un viaje que el usuario ha creado.



(a)



(b)

Figura 74: En (a) pantalla de edición de una oferta de viaje, en (b) pantalla de oferta de viaje con una posición personalizada.

4.4. Sprint 4

En este último Sprint se han hecho los requisitos restantes para finalizar el proyecto. Se han dejado el requisito **RF-7.7. Viajes periódicos** y el requisito **RF-7.8. Eliminar usuarios apuntados** para este sprint porque fueron añadidos durante el desarrollo en acuerdo con el cliente. Las pruebas unitarias también han sido desarrolladas en este Sprint, pero son explicadas en la sección siguiente, **Validación y verificación**.

Por último al igual que en anteriores Sprints, se ha refactorizado código y arreglado errores encontrados. Un cambio importante es que se ha cambiado el controlador de las ofertas para que sea una sola clase que se usa para tres listas distintas en vez de tres controladores distintos uno para cada lista de ofertas. Para ver el código desarrollado hasta este punto puede dirigirse al commit con el SHA `6e25620d317994496c4e0824e035ca549b21cceb`.

A continuación se muestran capturas del tablero de Trello durante este Sprint y capturas del diseño final de la aplicación.

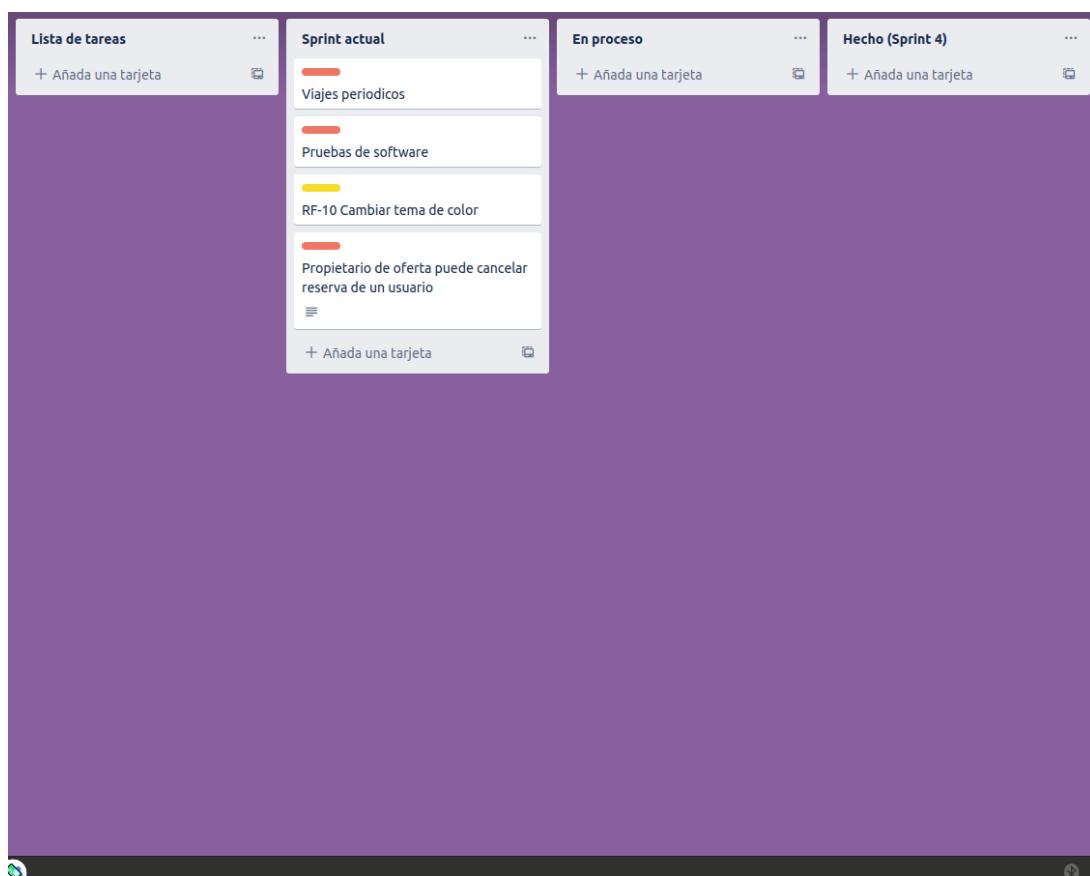


Figura 75: Captura de Trello al inicio del cuarto Sprint.

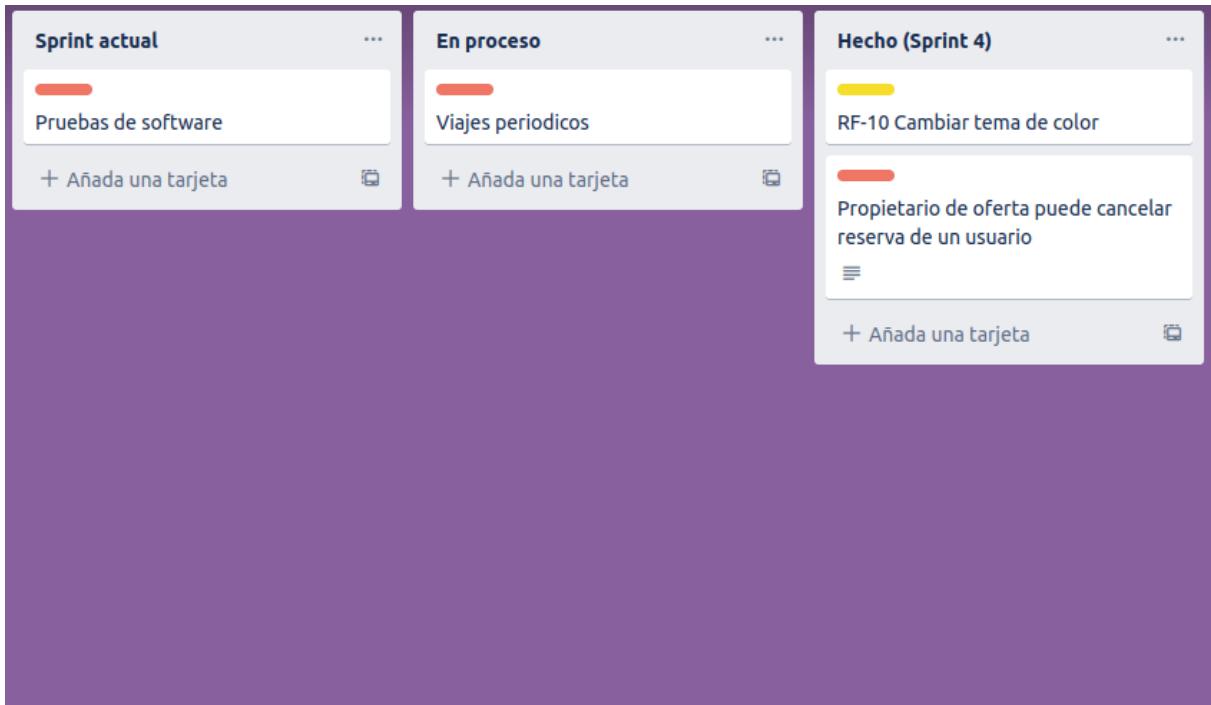


Figura 76: Captura de Trello durante el desarrollo de los viajes periódicos.

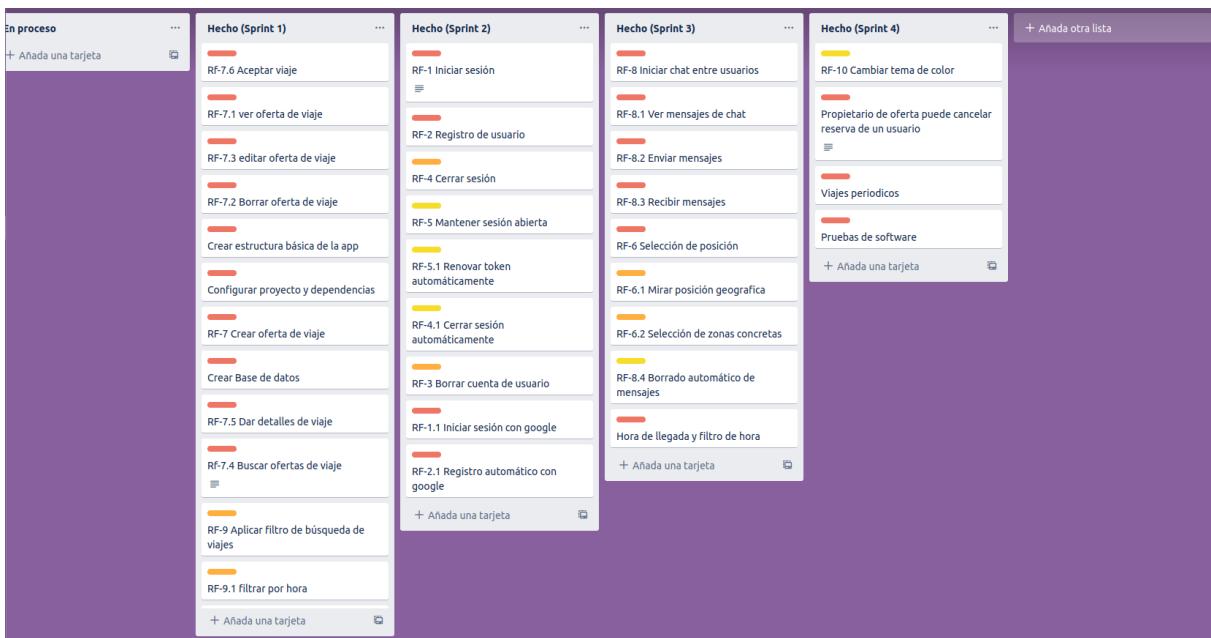
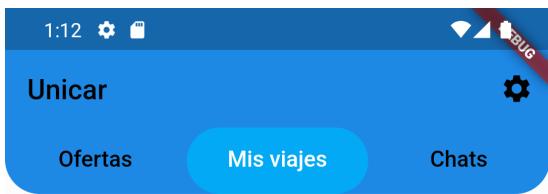


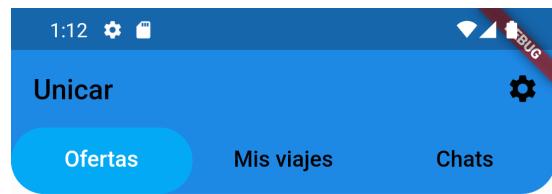
Figura 77: Captura de Trello al final del cuarto Sprint.



Viajes que ofreces

Viaje a la uma 2
Fuengirola → Ampliación de Teatinos
Para estar: 09/04 12:30

Viaje a la uma
Torremolinos → Teatinos
Salida: 12/04 09:45



viaje discord

Torremolinos → Teatinos
Para estar: 07/04 18:40

Viajes en los que estas apuntado

Un viaje recurrente
Benalmadena → Teatinos
Salida: 10/04 16:15

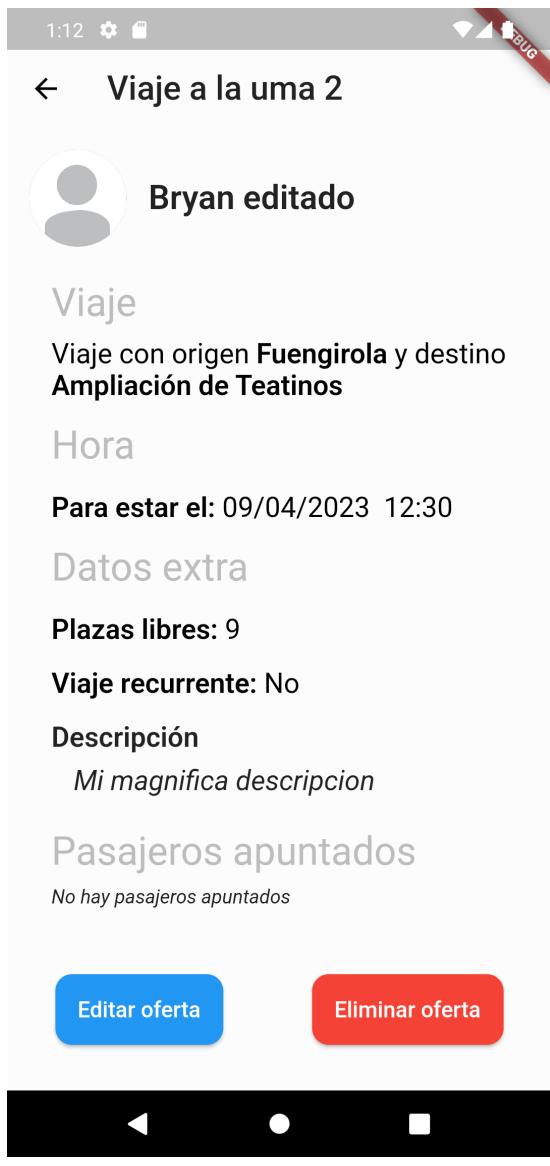
+ Publicar oferta

Filtrar

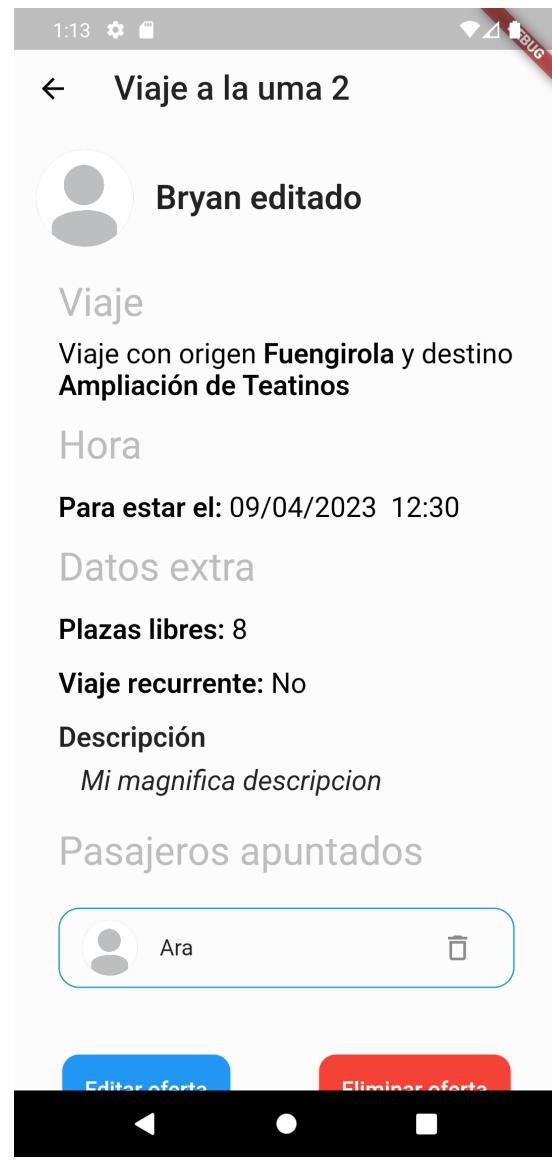
(a)

(b)

Figura 78: En (a) pantalla de “Mis viajes”, en (b) pantalla de Ofertas con una oferta de un usuario con que ha iniciado sesión con Discord con foto.

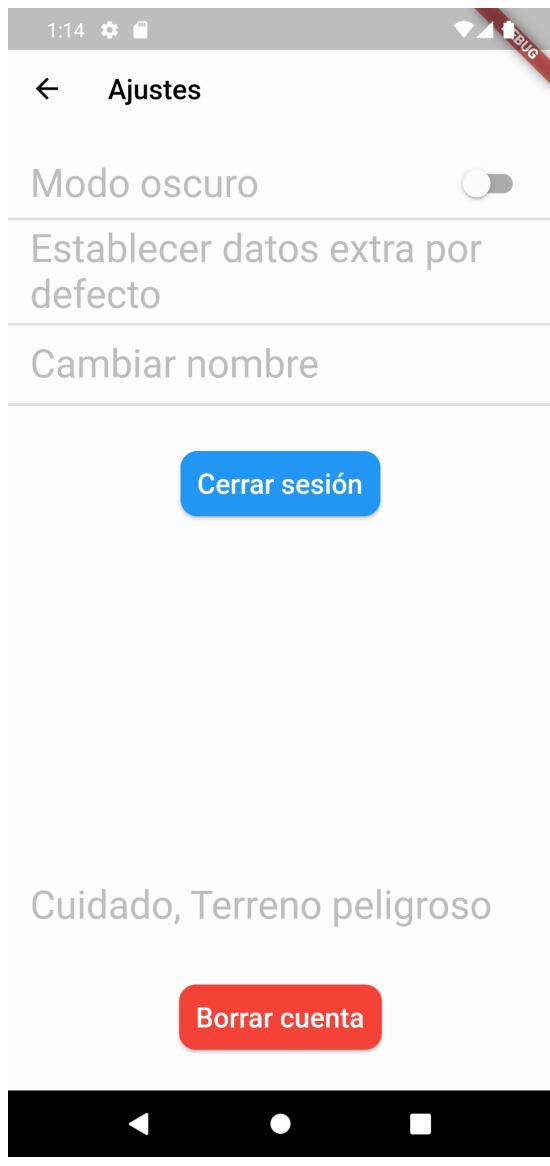


(a)



(b)

Figura 79: En (a) pantalla de viaje que ha creado el usuario que no tiene usuarios apuntados, en (b) pantalla de viaje que ha creado el usuario que tiene un usuario apuntado.

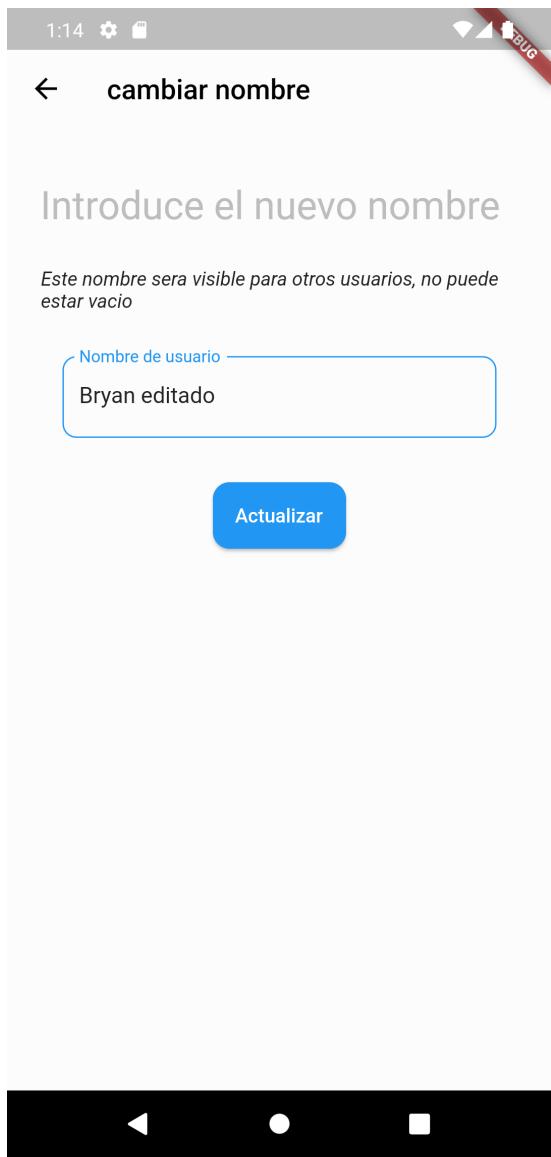


(a)

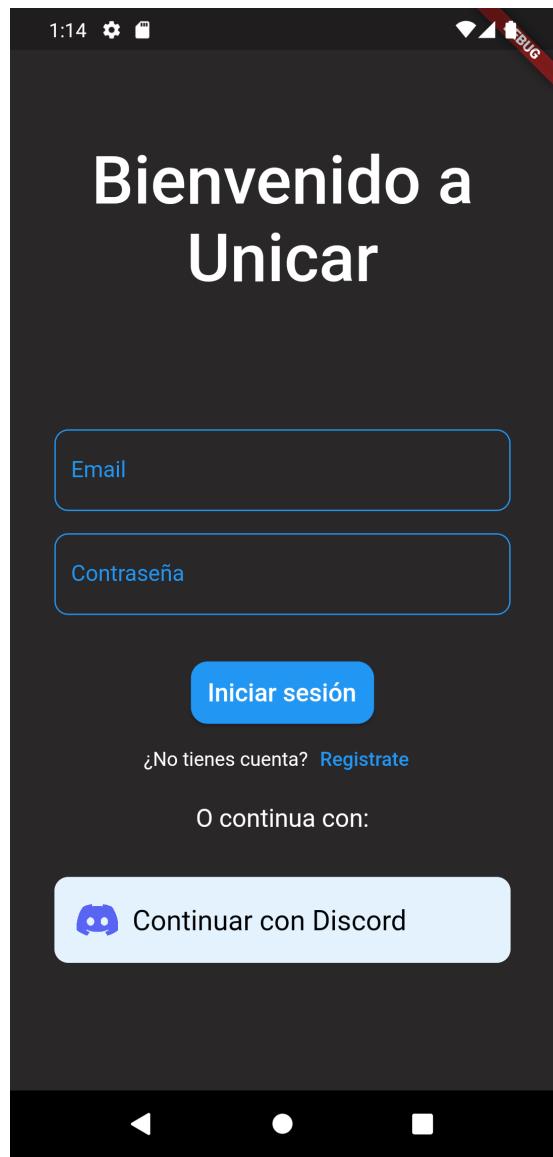


(b)

Figura 80: En (a) pantalla de configuración, en (b) pantalla de establecer datos extra por defecto.

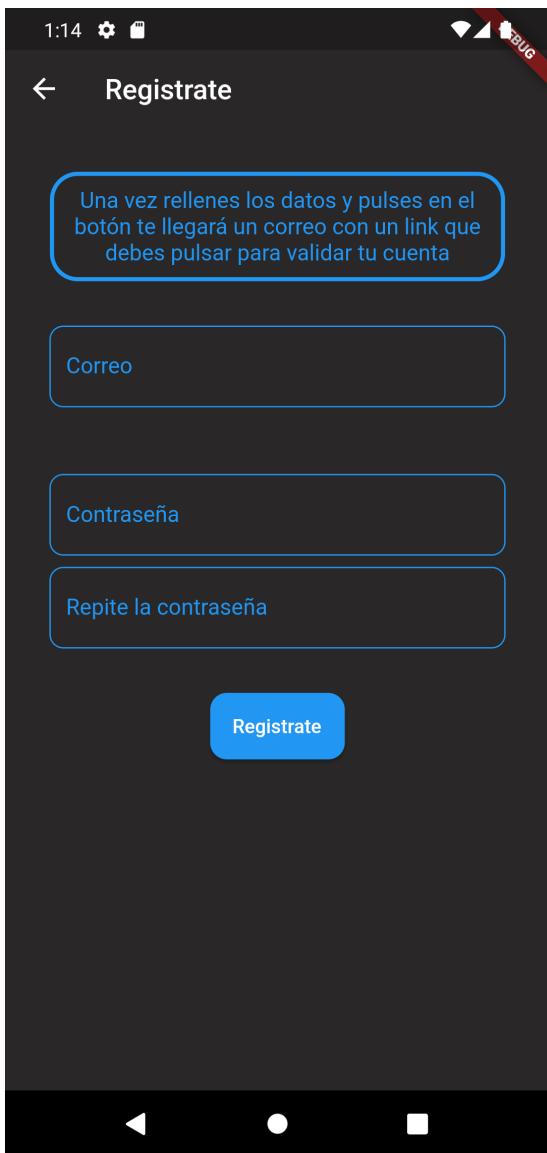


(a)

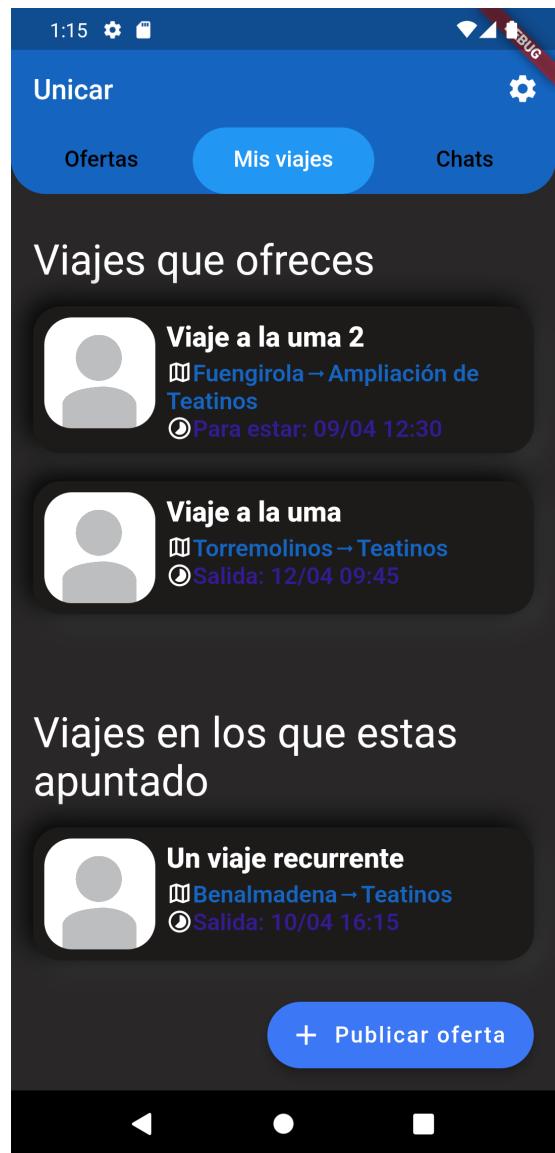


(b)

Figura 81: En (a) pantalla de cambiar nombre, en (b) pantalla de iniciar sesión en modo oscuro.

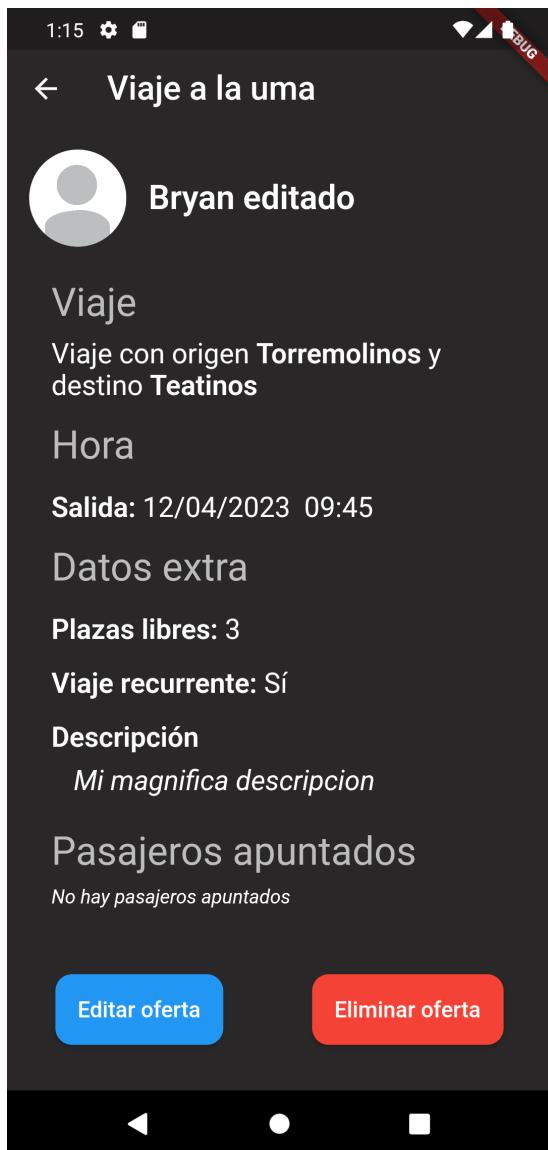


(a)

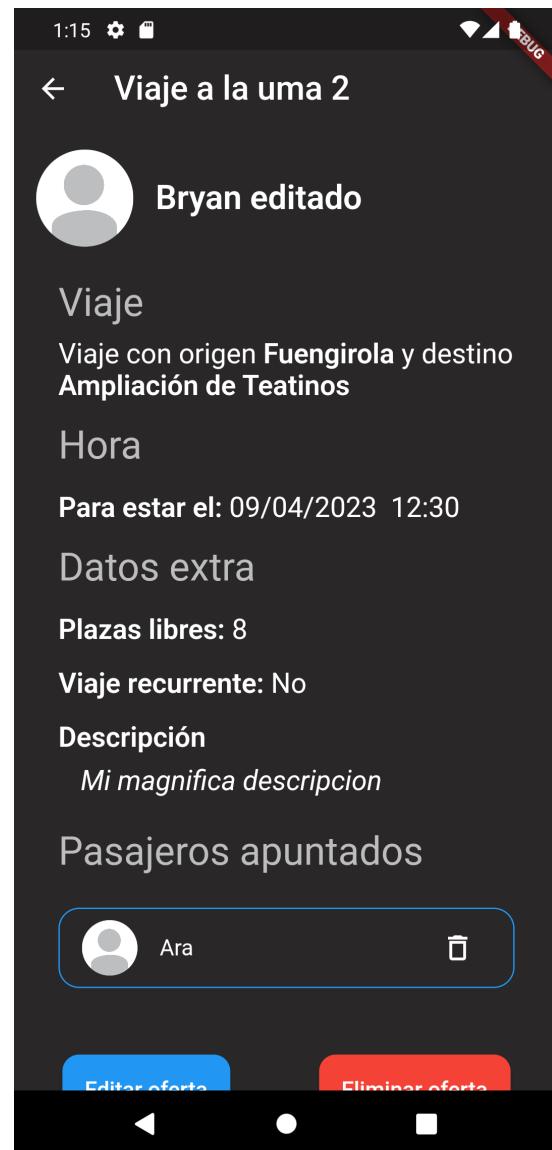


(b)

Figura 82: En (a) pantalla de registro en modo oscuro, en (b) pantalla de “Mis viajes” en modo oscuro.

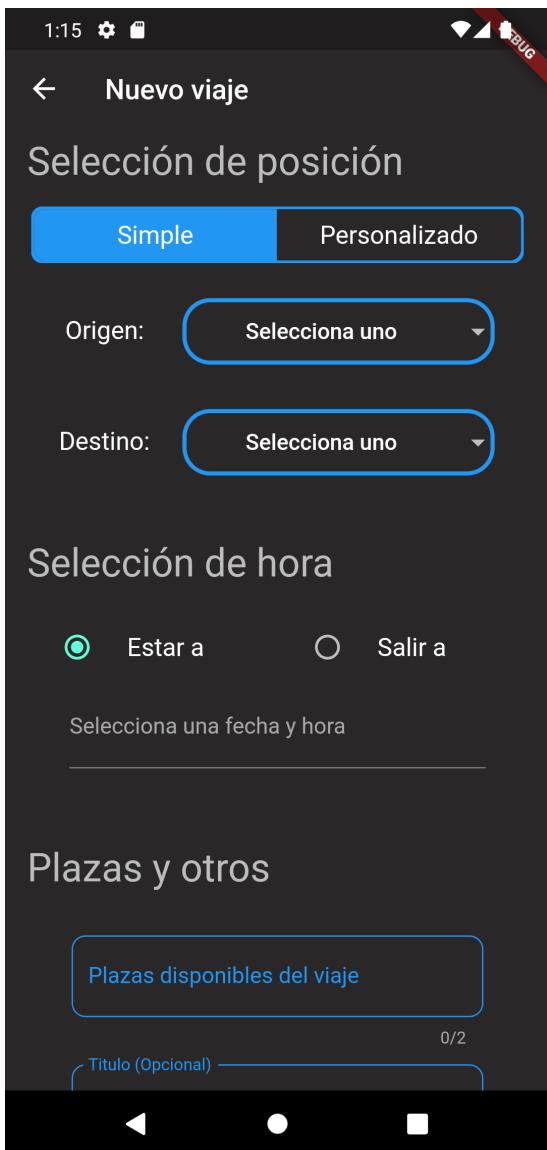


(a)

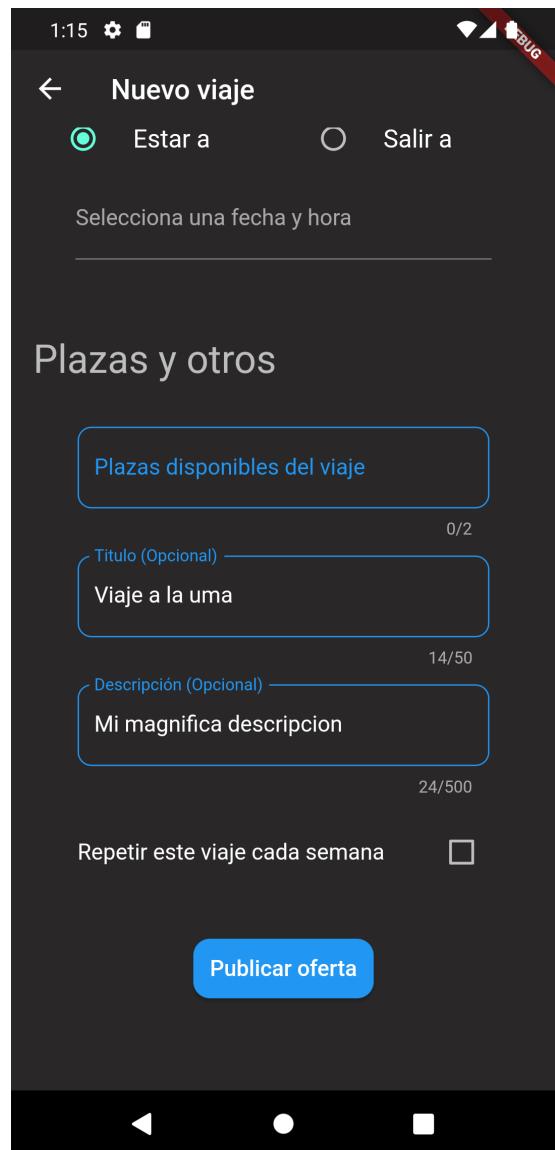


(b)

Figura 83: En (a) pantalla de viaje creado por el usuario sin pasajeros en modo oscuro, en (b) pantalla de viaje creado por el usuario con un pasajero en modo oscuro.

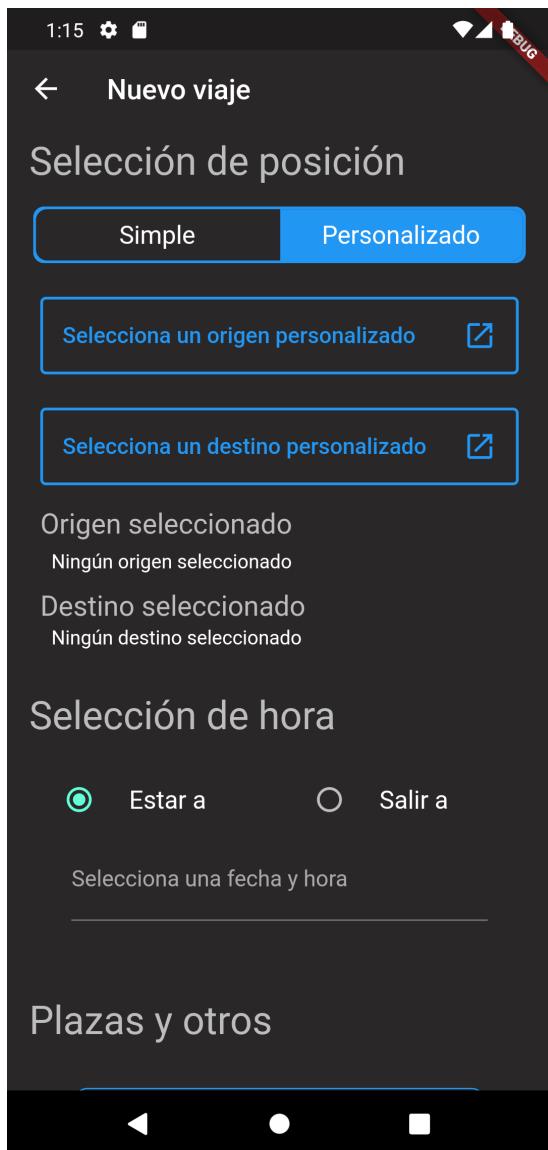


(a)

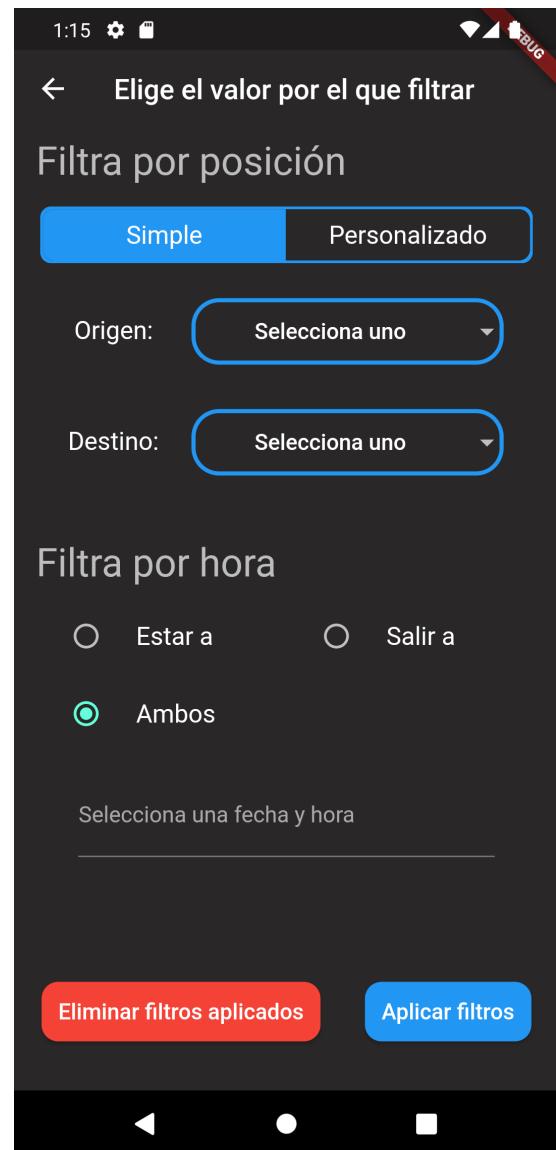


(b)

Figura 84: En (a) pantalla de crear una oferta de viaje en modo oscuro, en (b) pantalla de crear una oferta de viaje en modo oscuro al final.

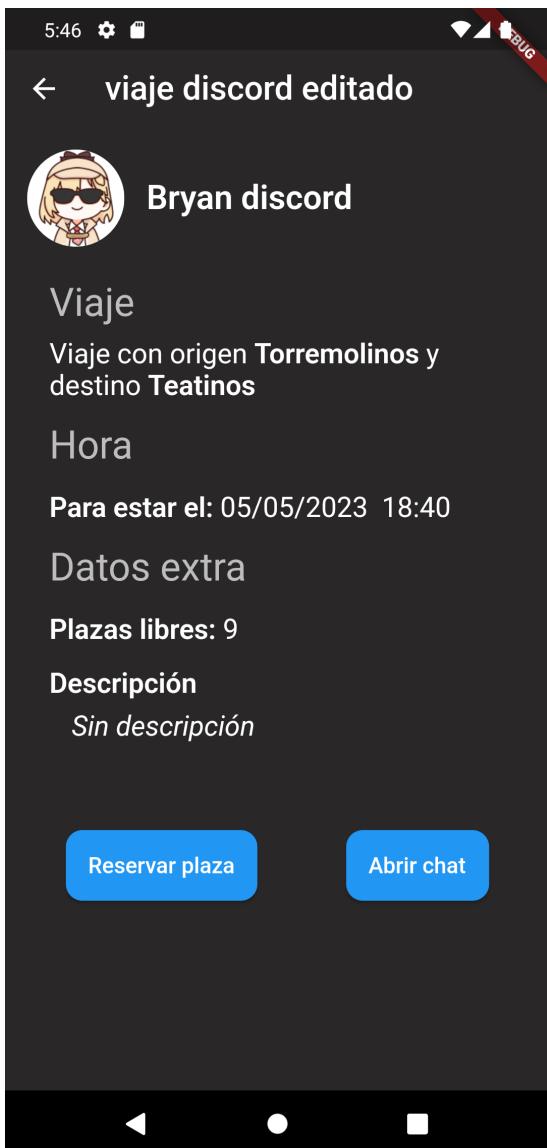


(a)

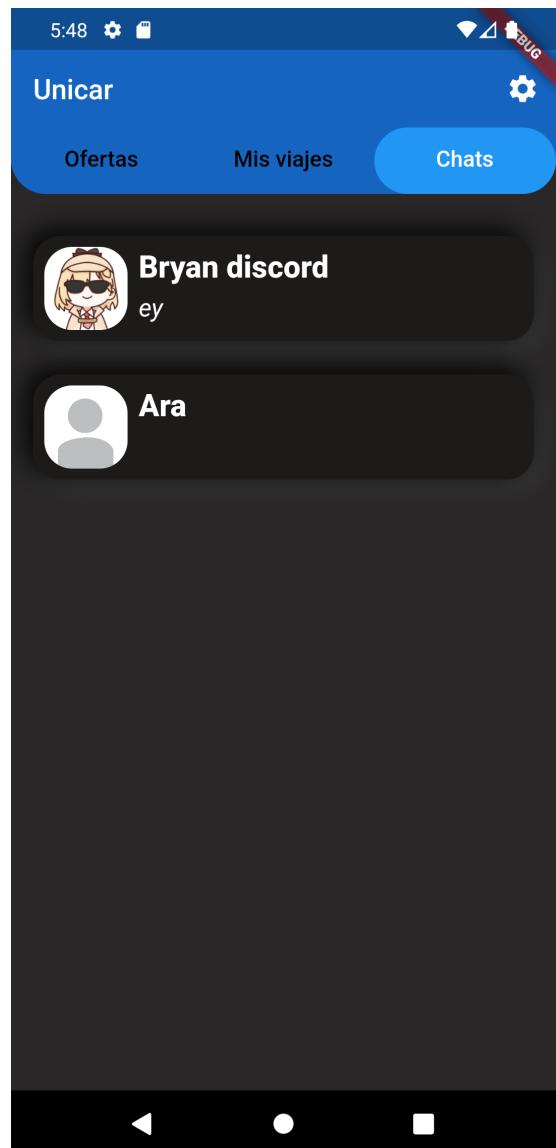


(b)

Figura 85: En (a) pantalla de crear una oferta de viaje con posición personalizada en modo oscuro, en (b) pantalla de filtros en modo oscuro.

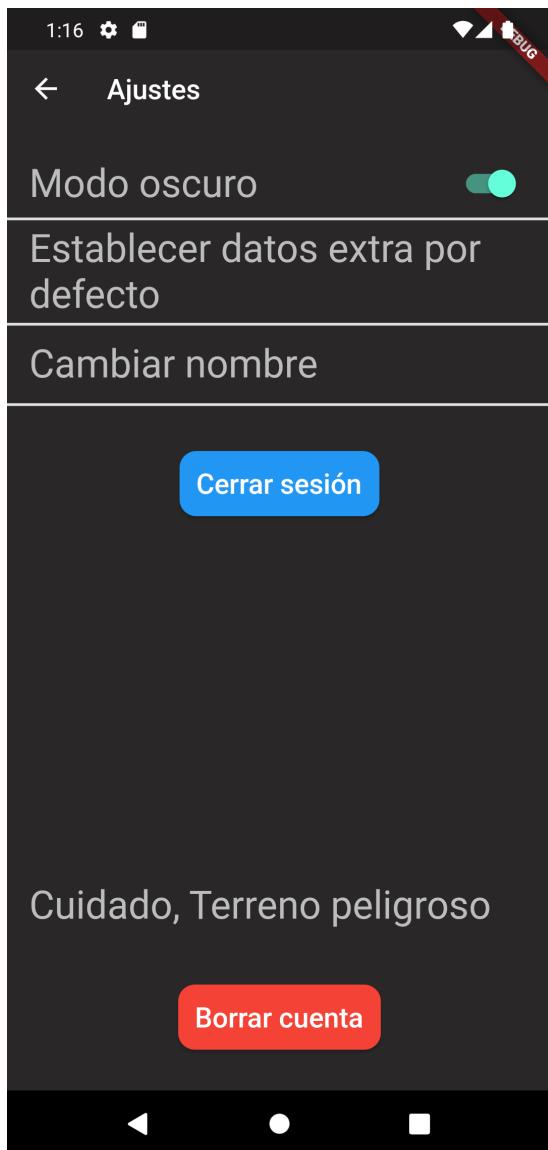


(a)

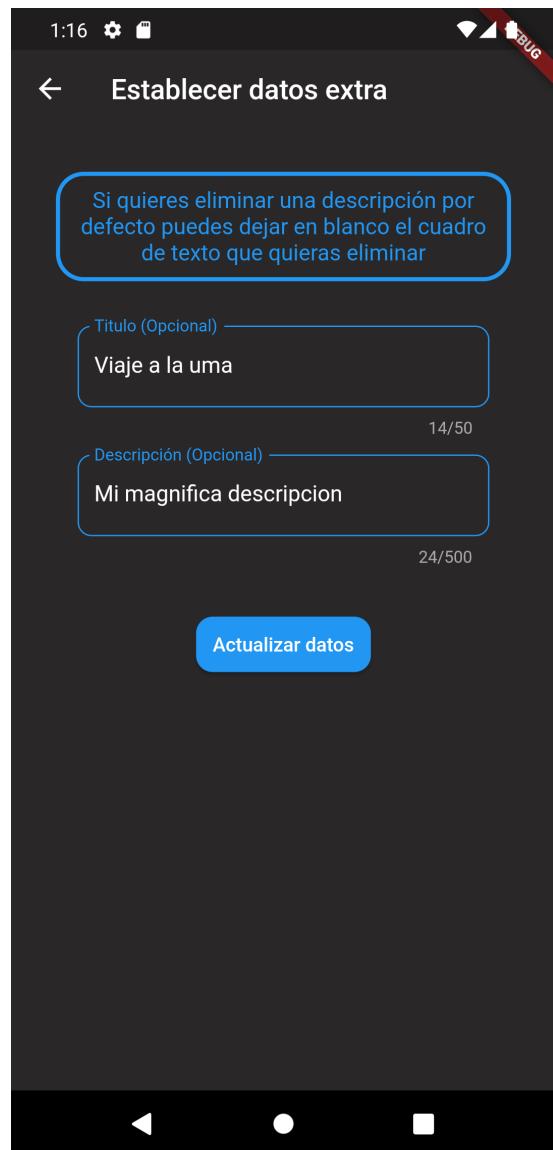


(b)

Figura 86: En (a) pantalla de una oferta de viaje en modo oscuro, en (b) pantalla de Chats en modo oscuro.

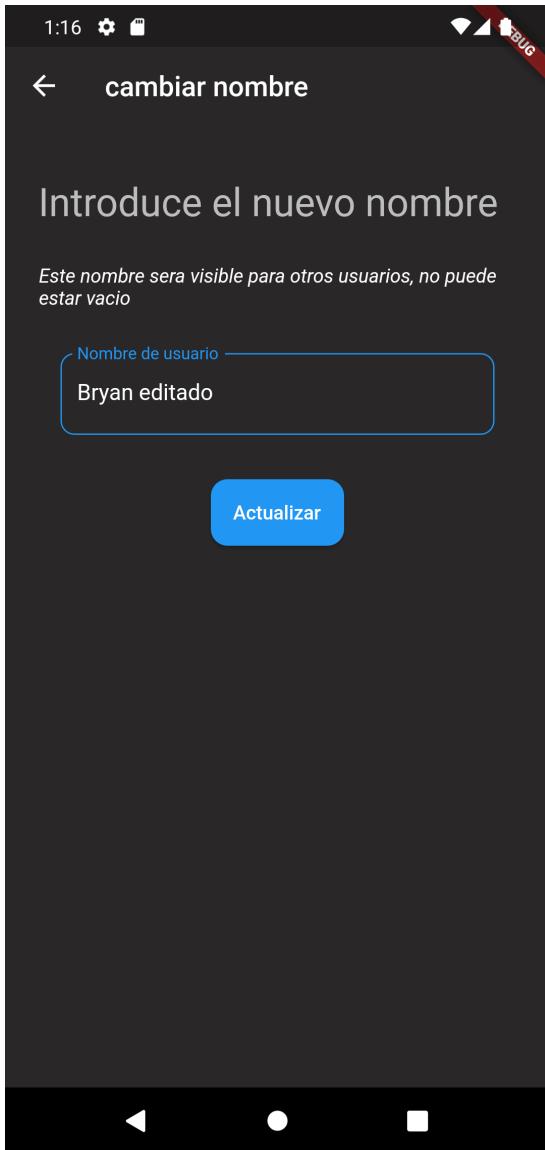


(a)

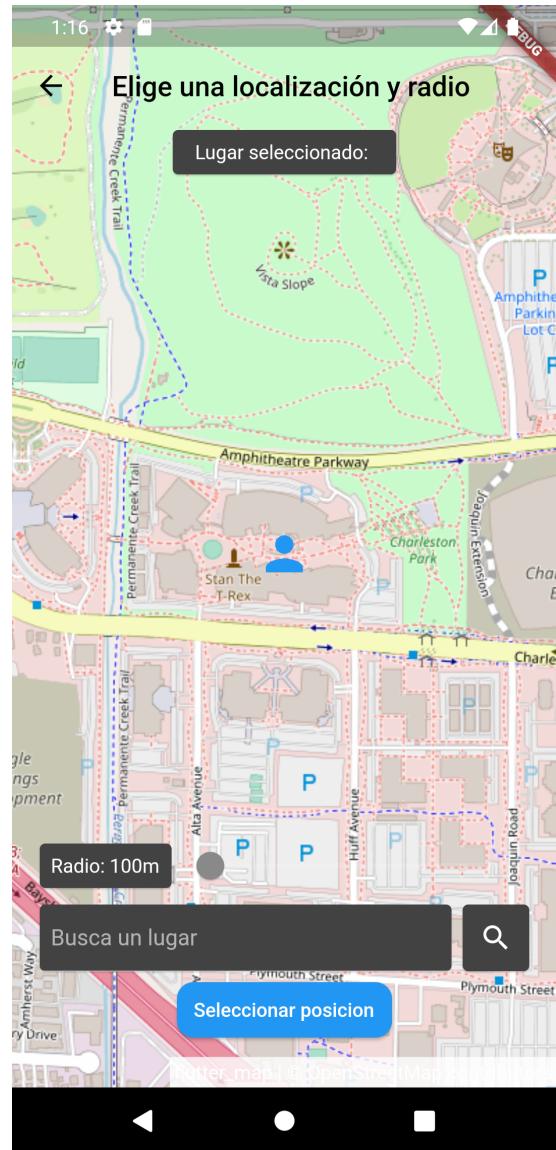


(b)

Figura 87: En (a) pantalla de configuración en modo oscuro, en (b) pantalla de establecer datos extra en modo oscuro.



(a)



(b)

Figura 88: En (a) pantalla de cambiar nombre en modo oscuro, en (b) pantalla de mapa para seleccionar posición en modo oscuro.

5

Validación y verificación

En esta sección se explica lo realizado para validar que el trabajo realizado se ajusta a los requisitos y verificar el correcto funcionamiento de la aplicación sometiéndola a una batería de pruebas.

5.1. Validación

Para validar que lo realizado era lo que el cliente quería y lo que estaba especificado en los requisitos, al final de cada Sprint se ha hecho una reunión donde se ha enseñado todo lo realizado haciendo una demo y a que requisito corresponde cada cosa. Si ha habido algo que no correspondía a lo que se pedía se ha arreglado al principio del siguiente Sprint. Además al final del desarrollo se ha hecho una revisión final comprobando que todo lo especificado ha sido cumplido. Esta comprobación ha sido también hecha para los requisitos no funcionales. Se ha comprobado lo siguiente:

- **RNF-1. Registro de usuario con Discord sencillo:** Se ha comprobado que cuando un usuario quiere iniciar sesión con Discord solo necesita pulsar el botón correspondiente e introducir sus credenciales y automáticamente entra la aplicación.
- **RNF-2. Intuitividad:** Se han contado las pulsaciones necesarias para acceder a todas las funcionalidades y en el peor de los casos se han encontrado 4 que es el límite.
- **RNF-3. Eficiencia:** Se ha medido el tiempo medio que la aplicación tarda en cargar y en ningún caso de funcionamiento normal y de no error se ha encontrado que tarde más de 6 segundos.

- **RNF-4. Fluidez:** Gracias a Flutter, las animaciones por defecto intentan adaptarse a la tasa de refresco de la pantalla del dispositivo.
- **RNF-5. Interfaz agradable a la vista:** Se han contado los colores usados, y en el modo claro se usan el blanco, azul y rojo de manera principal, con el extra de usar el color negro y el morado puntualmente y en el modo oscuro se ha usado principalmente el color negro, azul y rojo con el extra de usar el verde, blanco y morado. Además de esto se ha comprobado con el cliente que la aplicación es agradable a la vista.
- **RNF-6. Fácil adaptación del inicio de sesión a nuevos procedimientos:** Para lograr esto se ha hecho una clase de interfaz llamada Database y un controlador de modo que para añadir un nuevo método de inicio de sesión solo sería necesario añadir los métodos necesarios que se requieran implementar por estas clases.
- **RNF-7. Modularidad:** Se ha creado una clase “Boton” y otra “BotonMaterial” para reutilizar en todos los botones de la aplicación. También se tiene un Widget de mapa para poder usarse en varios sitios llamado “seleccion_posicion” y un widget que es una lista de viajes llamado “seccion_tarjetas”.
- **RNF-8. Seguridad:** Supabase cifra las contraseñas por defecto.

5.2. Verificación

Para asegurar el correcto funcionamiento de todo lo implementado, aparte de probar manualmente todo lo implementado, se han hecho pruebas de caja blanca a través de test unitarios. Las pruebas de caja blanca consisten en hacer pruebas teniendo el código disponible para revisar, mientras que los test de caja negra son test donde no se conoce la implementación de lo que se prueba.

Con los test unitarios realizados se ha buscado probar dos cosas concretas. Primero se han hecho pruebas sobre los widgets para probar que todos los elementos gráficos y datos se muestran de manera correcta. Lo segundo que se ha hecho es hacer pruebas sobre las clases de controlador que contienen toda la lógica de la aplicación. La excepción a esto son las funciones que llaman a la base de datos debido que sería necesario crear una base de datos de prueba y debido a restricciones de tiempo y complejidad no ha sido posible.

Widgets probados:

- **Pantalla de crear oferta:** Se ha comprobado que se muestre la selección de posición, que se muestre una selección de origen y una de destino, que la selección personalizada muestra una opción de elegir un origen y un destino, que la selección de hora se muestra y que la sección de plazas y otros se muestra.
- **Pantalla de editar oferta:** Se ha comprobado que se muestren los valores correctos en los menús despegables, que la hora original antes de editar se muestran y que los valores de datos extra se muestran.
- **Pantalla de filtros:** Se ha comprobado que se muestren todos los tipos de filtro correctamente y el botón para aplicarlos y eliminarlos.
- **Pantalla de mapa:** Se ha comprobado que se muestra correctamente la interfaz del mapa al seleccionar una posición y que se muestran los datos si se esta editando una posición personalizada. No se ha podido probar que se muestran los cuadros del mapa debido a limitaciones en las librerías de pruebas de Dart y Flutter.
- **Pantalla principal:** Se ha comprobado que se muestran los viajes propios donde se espera y de manera correcta, los viajes en los que se esta apuntado aparecen donde se esperan correctamente y las ofertas se muestran correctamente y donde se espera. También se ha comprobado que los chats se muestran correctamente y donde es esperado. Por último se ha comprobado que si no hay datos no aparece nada.
- **Pantalla de vista de chat:** Se ha comprobado que se muestra el campo para escribir un mensaje y el botón para enviarlo y también que se muestran los mensajes si los hay de manera correcta y también el nombre del usuario.
- **Pantalla de vista de viaje:** Se ha comprobado que se muestren correctamente los datos de un viaje, que se muestran las tarjetas de pasajeros, se muestran los botones correctos dependiendo del tipo de oferta que es y también que aparecen o no las tarjetas de pasajeros dependiendo del tipo de la oferta.

Funciones de controladores probadas:

- **Funciones de gestión de ofertas:** Se ha probado que se añaden ofertas y eliminan correctamente al controlador. Si se intenta borrar una oferta que no existe, no hace nada. Reservar plaza disminuye el numero de plazas de la oferta y si se usa sobre una oferta del usuario o en la que esta apuntado no hace nada. Cancelar plaza aumenta el número de plazas disponibles y si el usuario no esta apuntado no hace nada. Editar oferta cambia solo los valores que se le pasa nuevos y no cambia coordenadas si no se le pasan. Los viajes se crean correctamente.
- **Función de filtro:** Se ha comprobado que todos los filtros se aplican de manera correcta individualmente y combinando varios de ellos. Tanto para valores con los que existen ofertas como para valore para los que no lo hay.
- **Funciones de chat:** Se ha probado que los chats se buscan correctamente, si un chat no existe la función “buscarChat“ devuelve el valor “null“, los chats se crean correctamente y los chats se ponen al principio cuando un mensaje nuevo llega de manera correcta.

Por último en la figura Captura donde se ve que las pruebas se han pasado satisfactoriamente, se muestra una captura de como se pasan todas las pruebas.

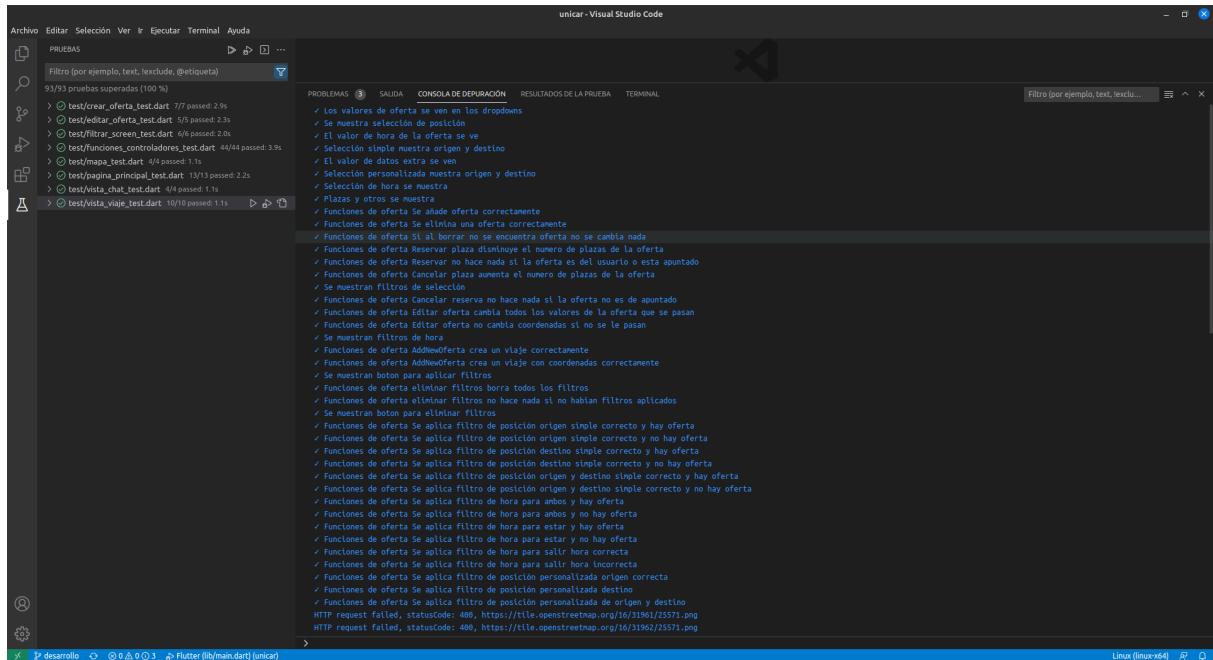


Figura 89: Captura donde se ve que las pruebas se han pasado satisfactoriamente

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

En conclusión a este proyecto, se ha conseguido hacer el diseño completo, desde los requisitos hasta los diagramas de interacción, maquetas de software y pruebas y además llevar el diseño a la práctica con una implementación usando Flutter y Dart.

El proyecto se ha abordado desde un punto de vista ingenieril, donde se han sopesado todas las posibles decisiones de diseño y generando la documentación necesaria en cada paso, con el fin de poder continuar con el proyecto en un futuro y que sea eficiente la incorporación de terceros que pudieran trabajar en el mismo. Por otro lado, se ha usado una metodología ágil lo cual ha resultado beneficioso ya que con su flexibilidad ha facilitado el cambio o adición de requisitos en etapas tempranas del proyecto.

El objetivo inicial de este proyecto era conseguir una solución mejor que un grupo de mensajería instantánea para compartir coche para los alumnos de la universidad de Málaga. Y creo que con la aplicación que se ha diseñado y desarrollado no solo se ha conseguido el objetivo sino que se ha conseguido hacer algo que es fácil y cómodo de usar, eficiente y bastante adaptable a nuevos requisitos que puedan surgir en un futuro. La buena calidad de los documentos y del seguimiento de los Sprint no solo ha sido gracias a mi esfuerzo sino también gracias al de mi tutor que en cada fase del proyecto ha revisado mi trabajo y me ha dado retroalimentación e ideas para hacerlo todo lo mejor posible.

Y ya que se tiene tal extensa documentación cualquier persona con ciertos conocimientos podría continuar el proyecto. Este proyecto podría ser continuado por la propia universidad de Málaga para poder ofrecérselo a los alumnos como una opción medio am-

bientalmente sensible y mejor que un grupo de mensajería o un formulario.

6.2. Líneas Futuras

Aunque la aplicación ya podría ser usada por los alumnos de la universidad, todavía se pueden implementar mejoras que hacen la experiencia más cómoda y dan más posibilidades de uso.

Algunas de las mejoras que se podrían realizar en un futuro serían:

- **Solicitud de viajes:** Actualmente los usuarios solo pueden ofrecer viajes, pero no pueden pedir un viaje como podrían hacerlo en el grupo de mensajería instantánea. Al igual que con el ofrecimiento de viajes, los usuarios podrán crear, borrar, editar, escoger y filtrar solicitudes. Estas solicitudes aparecerán en la misma lista que las ofertas pero tendrán un color distinto para ser fácilmente distinguibles. También se podrá filtrar para ver solo ofertas o solo solicitudes. Además, las solicitudes dejarán de mostrarse en la lista una vez alguien la ha aceptado.
- **Gestión de lista de usuarios favoritos:** Los usuarios podrán añadir a una lista de favoritos a otros usuarios. Con esto los usuarios podrán comunicarse más rápidamente y ver directamente si un usuario concreto ofrece un viaje. También será posible eliminar a un usuario de esta lista.
- **Puntuar a usuarios:** Los usuarios podrán puntuar a otros usuarios con puntuaciones de 0 a 5. De esta forma los usuarios podrán saber si una persona que ofrece un viaje es de fiar o si ha habido algún problema y se le ha evaluado negativamente. Esta puntuación podrá ser editada una vez establecida.
- **Recibir notificaciones push:** Los usuarios recibirán notificaciones sobre los mensajes de chats que reciban y también avisos en forma de notificación sobre viajes en los que están apuntados y que empiezan pronto. No será necesario tener la aplicación abierta para recibir estas notificaciones. Esta es quizás la mejora más importante que se podría hacer, ya que actualmente para un usuario comprobar si tiene mensajes nuevos tiene que entrar en la aplicación y comprobarlo.

- **Filtrado automático:** Los usuarios podrán establecer una hora y zona preferida y las ofertas disponibles se ordenarán y mostrarán en base a esas preferencias.
- **Desplazar para recargar:** Ahora mismo la única forma de cargar nuevas ofertas en la pantalla de ofertas es cerrando la aplicación y volviéndola a abrir. Por ello sería necesario incluir una opción para recargar las ofertas. Una forma bastante común de hacer esto es que en la pantalla de ofertas al desplazar hacia abajo salga un ícono y se recarguen.
- **Paginación:** Ahora mismo la aplicación carga todas las ofertas que hay disponibles. Para ahorrar recursos en el caso de que hayan muchas ofertas, sería conveniente cargar solo las últimas 20 y cuando el usuario llega al final de la lista que se carguen otras 20.
- **Filtros:** Ahora mismo los filtros filtran sobre las ofertas que hay localmente. Es decir que si se implementa la paginación solo se filtraría sobre lo que haya. Por ello sería una buena idea implementar los filtros de manera que se consulte la base de datos para asegurar que se muestra todo.

Referencias

- [1] “Flutter - Build apps for any screen.” <https://flutter.dev/> (Accedido el 16 de abril de 2023).
- [2] “Dart programming language” Dart. <https://dart.dev/> (Accedido el 18 de abril de 2023).
- [3] “The Open Source Firebase Alternative | Supabase” Supabase. <https://supabase.com/> (Accedido el 18 de abril de 2023).
- [4] Firebase, “Firebase” Firebase. <https://firebase.google.com/?hl=es-419> (Accedido el 18 de abril de 2023).
- [5] MongoDB, “Explicación Sobre Las Bases De Datos NoSQL” MongoDB. <https://www.mongodb.com/es/nosql-explained> (Accedido el 18 de abril de 2023).
- [6] “PostgreSQL” PostgreSQL, Apr. 20, 2023. <https://www.postgresql.org/> (Accedido el 20 de abril de 2023).
- [7] “PL/SQL para desarrolladores | Oracle España.” <https://www.oracle.com/es/database/technologies/appdev/plsql.html> (Accedido el 20 de abril de 2023).
- [8] “Git.” <https://git-scm.com/> (Accedido el 20 de abril de 2023).
- [9] “GitHub: Let’s build from here” GitHub. <https://github.com/> (Accedido el 20 de abril de 2023).
- [10] “Documentation for Visual Studio Code” Nov. 03, 2021. <https://code.visualstudio.com/docs> (Accedido el 21 de abril de 2023).
- [11] “Introducción a Android Studio” Android Developers, [Online]. Available: <https://developer.android.com/studio/intro?hl=es-419> (Accedido el 21 de abril de 2023).
- [12] “Cómo crear y administrar dispositivos virtuales” Android Developers, [Online]. Available: <https://developer.android.com/studio/run/managing-avds?hl=es-419> (Accedido el 21 de abril de 2023).

- [13] ModelioOpenSource, “GitHub - ModelioOpenSource/Modelio at v5.3.1” GitHub. <https://github.com/ModelioOpenSource/Modelio/tree/v5.3.1> (Accedido el 21 de abril de 2023).
- [14] “Ideal Modeling and Diagramming Tool for Agile Team Collaboration.” <https://www.visual-paradigm.com/> (Accedido el 21 de abril de 2023).
- [15] “Overleaf, Online LaTeX Editor.” <https://www.overleaf.com/> (Accedido el 21 de abril de 2023).
- [16] “LaTeX - A document preparation system.” <https://www.latex-project.org/> (Accedido el 21 de abril de 2023).
- [17] Krita Foundation, “Krita” Krita, Aug. 31, 2020. <https://krita.org/> (Accedido el 21 de abril de 2023).
- [18] “Home - Pencil Project.” <https://pencil.evolus.vn/> (Accedido el 21 de abril de 2023).
- [19] “Manage Your Team’s Projects From Anywhere | Trello.” <https://trello.com/> (Accedido el 22 de abril de 2023).
- [20] Atlassian, “¿Qué es ágil? | Atlassian” Atlassian. <https://www.atlassian.com/es/agile> (Accedido el 22 de abril de 2023).
- [21] Atlassian, “Scrum: qué es, cómo funciona y por qué es excelente” Atlassian. <https://www.atlassian.com/es/agile/scrum> (Accedido el 22 de abril de 2023).
- [22] “Data Modeling Tools | Oracle SQL Developer Data Modeler.” <https://www.oracle.com/database/sqldeveloper/technologies/sql-data-modeler/> (Accedido el 22 de abril de 2023).
- [23] “Managing User Data | Supabase Docs” Apr. 22, 2023. <https://supabase.com/docs/guides/auth/managing-user-data> (Accedido el 22 de abril de 2023).
- [24] “Row Level Security | Supabase Docs” Apr. 25, 2023. <https://supabase.com/docs/guides/auth/row-level-security> (Accedido el 25 de abril de 2023).

- [25] “How to let a user delete their own account? · supabase · Discussion 1066” GitHub. <https://github.com/orgs/supabase/discussions/1066> (Accedido el 25 de abril de 2023).
- [26] “How to delete users? · supabase · Discussion 250” GitHub. <https://github.com/orgs/supabase/discussions/250> (Accedido el 25 de abril de 2023).
- [27] “Flutter Tutorial: building a Flutter chat app” Supabase, Jun. 30, 2022. <https://supabase.com/blog/flutter-tutorial-building-a-chat-app> (Accedido el 25 de abril de 2023).
- [28] “Postgres as a CRON Server” Supabase, Mar. 05, 2021. <https://supabase.com/blog/postgres-as-a-cron-server> (Accedido el 25 de abril de 2023).
- [29] “Sound null safety” Dart. <https://dart.dev/null-safety> (Accedido el 26 de abril de 2023).
- [30] “What is JSON.” https://www.w3schools.com/whatis/whatis_json.asp (Accedido el 26 de abril de 2023).
- [31] “Riverpod.” <https://riverpod.dev/es/> (Accedido el 26 de abril de 2023).
- [32] “supabase_flutter | Flutter Package” Dart Packages. https://pub.dev/packages/supabase_flutter (Accedido el 28 de abril de 2023).
- [33] “flutter_riverpod | Flutter Package” Dart Packages. https://pub.dev/packages/flutter_riverpod (Accedido el 28 de abril de 2023).
- [34] “riverpod_annotation | Dart Package” Dart Packages. https://pub.dev/packages/riverpod_annotation (Accedido el 28 de abril de 2023).
- [35] “image_picker | Flutter Package” Dart Packages. https://pub.dev/packages/image_picker (Accedido el 28 de abril de 2023).
- [36] “date_time_picker | Flutter Package” Dart Packages. https://pub.dev/packages/date_time_picker (Accedido el 28 de abril de 2023).

- [37] “string_validator | Dart Package” Dart Packages. https://pub.dev/packages/string_validator (Accedido el 28 de abril de 2023).
- [38] “intl | Dart Package” Dart Packages. <https://pub.dev/packages/intl> (Accedido el 28 de abril de 2023).
- [39] “geocoding | Flutter Package” Dart Packages. <https://pub.dev/packages/geocoding> (Accedido el 28 de abril de 2023).
- [40] “geolocator | Flutter Package” Dart Packages. <https://pub.dev/packages/geolocator> (Accedido el 28 de abril de 2023).
- [41] “flutter_map | Flutter Package” Dart Packages. https://pub.dev/packages/flutter_map (Accedido el 28 de abril de 2023).
- [42] “latlong2 | Dart Package” Dart Packages. <https://pub.dev/packages/latlong2> (Accedido el 28 de abril de 2023).
- [43] “toggle_switch | Flutter Package” Dart Packages. https://pub.dev/packages/toggle_switch (Accedido el 28 de abril de 2023).
- [44] “shared_preferences | Flutter Package” Dart Packages. https://pub.dev/packages/shared_preferences (Accedido el 28 de abril de 2023).
- [45] “permission_handler | Flutter Package” Dart Packages. https://pub.dev/packages/permission_handler (Accedido el 28 de abril de 2023).
- [46] “OpenStreetMap” OpenStreetMap. <https://www.openstreetmap.org/about> (Accedido el 29 de abril de 2023).
- [47] “Raster tile providers - OpenStreetMap Wiki.” https://wiki.openstreetmap.org/wiki/Raster_tile_providers (Accedido el 29 de abril de 2023).

Apéndice A

Manual de

Instalación

Para instalar la aplicación desarrollada solo es necesario descargar o mover el archivo “apk” al dispositivo Android donde se quiera instalar y pulsar sobre el archivo. una vez hecho esto se deberán seguir las instrucciones del dispositivo. Cabe destacar que el archivo no esta firmado actualmente por lo que saltarán varias alertas de seguridad.

En el caso de querer crear el archivo apk desde el código, será necesario ir a la raíz del proyecto con una terminal y ejecutar el comando “flutter build apk –split-per-abi” para tener una apk por cada conjunto de instrucciones disponibles o el comando “flutter build appbundle” para tener un conjunto con todas las apks en un solo archivo. Los archivos generados se pueden encontrar en el directorio del proyecto “/build/app/outputs/bundle/release”.

Apéndice B

Manual de Usuario

En esta sección se da una explicación breve de como usar de manera básica la aplicación desarrollada. Tiene el nombre temporal de “Unicar”.

B.1. Inicio de sesión y registro

Al abrir la aplicación por primera vez encontrarás una pantalla donde se te pide que introduzcas un correo electrónico y una contraseña o iniciar sesión con Discord. También se muestra un botón para crear una cuenta como se muestra en la siguiente figura.



Figura 90: Pantalla de inicio de sesión

Si es tu primera vez usando la aplicación deberás pulsar en el texto “Regístrate” que te llevará a una pantalla nueva para poner un correo y una contraseña o pulsar en el botón con el texto “Continuar con Discord” que abrirá tu navegador por defecto con una

página de Discord donde al poner tus credenciales serás devuelto a la aplicación y ya podrás usarla.

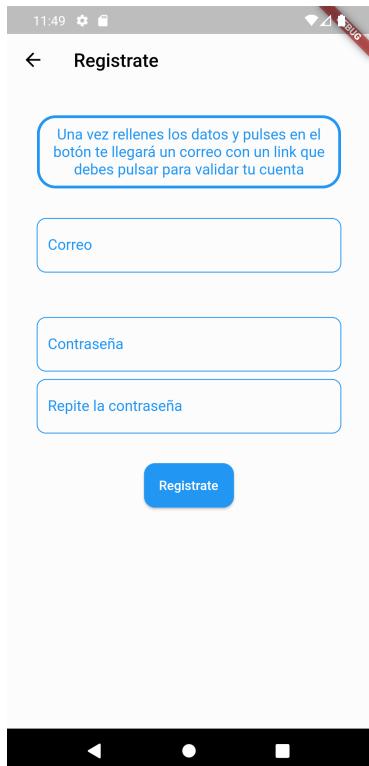


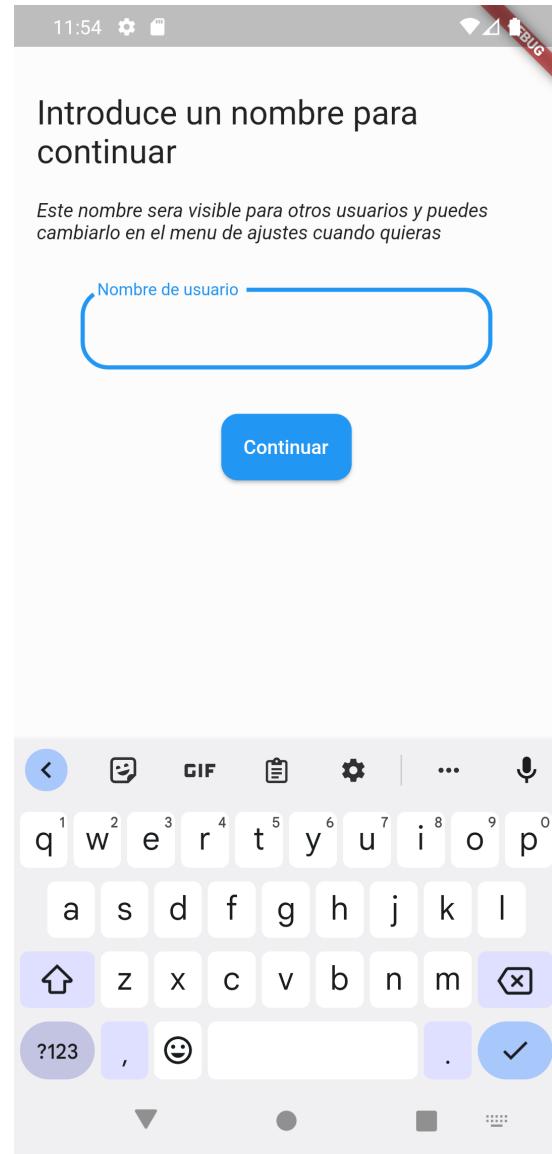
Figura 91: Pantalla de registro

En el caso de usar el registro, una vez hayas puesto el correo y contraseña y pulsas en el botón, te llegará un correo electrónico con un enlace que debes pulsar. No te preocupes si la página a la que te lleva muestra un error, esto es normal.

Una vez hayas validado tu cuenta ya puedes poner tu correo y contraseña en los campos correspondientes.



(a)



(b)

Figura 92: En (a) pantalla de iniciar sesión, en (b) pantalla de introducir nombre.

La primera vez que inicies sesión con correo electrónico y contraseña se te pedirá que introduzcas un nombre. Este nombre se puede cambiar después así que si no sabes qué poner no te preocupes.

B.2. Creando y editando un viaje

Una vez has entrado en la aplicación te encontrarás con una pantalla donde salen tus ofertas de viajes publicadas y las ofertas en las que te has apuntado (reservado una plaza). Para crear una nueva oferta pulsa en el botón con el texto “Publicar oferta”.

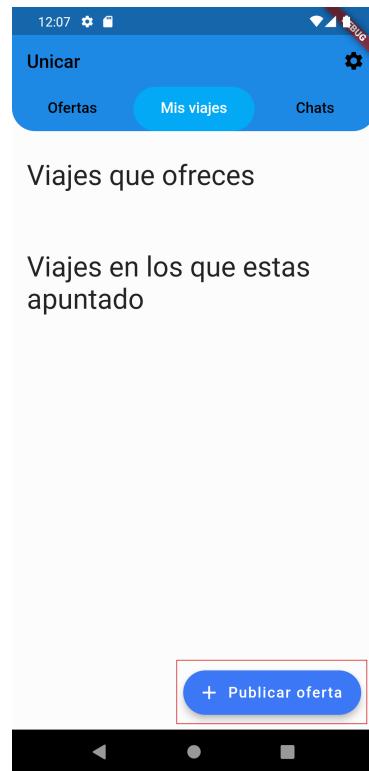


Figura 93: Pantalla de inicio

En la pantalla que te saldrá puedes elegir desde un menú desplegable un origen para el viaje que quieras crear y también un destino. Si la zona en la que estas no esta disponible en el menú puedes usar la selección personalizada pulsando en el texto “Personalizado”. Aquí se te presentan dos botones para seleccionar una posición. Al pulsarlos se abrirá una pantalla nueva donde al hacer clic seleccionas un punto. También puedes seleccionar un radio. Este radio expresa como de lejos estas dispuesto a desplazarte desde el punto que has elegido para poder empezar el viaje o para llegar a tu destino. El icono azul de una persona es la ubicación que la aplicación ha recuperado de ti. Si encuentras problemas con esta función debes dar permisos de localización a la aplicación para que todo funcione correctamente. Además de poder pulsar en el mapa, puedes buscar un lugar por su nombre

y la aplicación te llevará al lugar más cercano que encuentre con el nombre que le has dado.

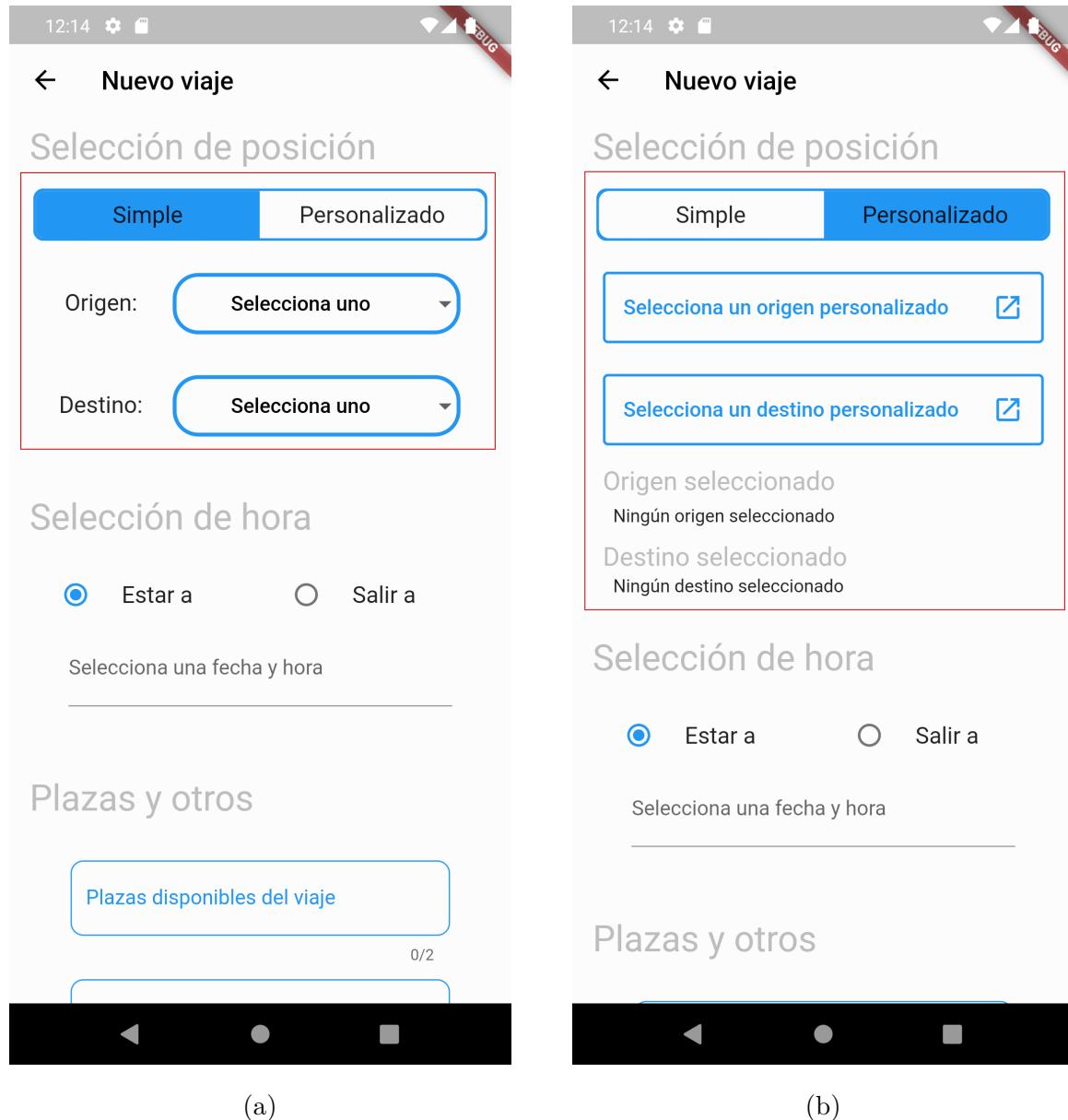


Figura 94: En (a) posición simple, en (b) posición personalizada.

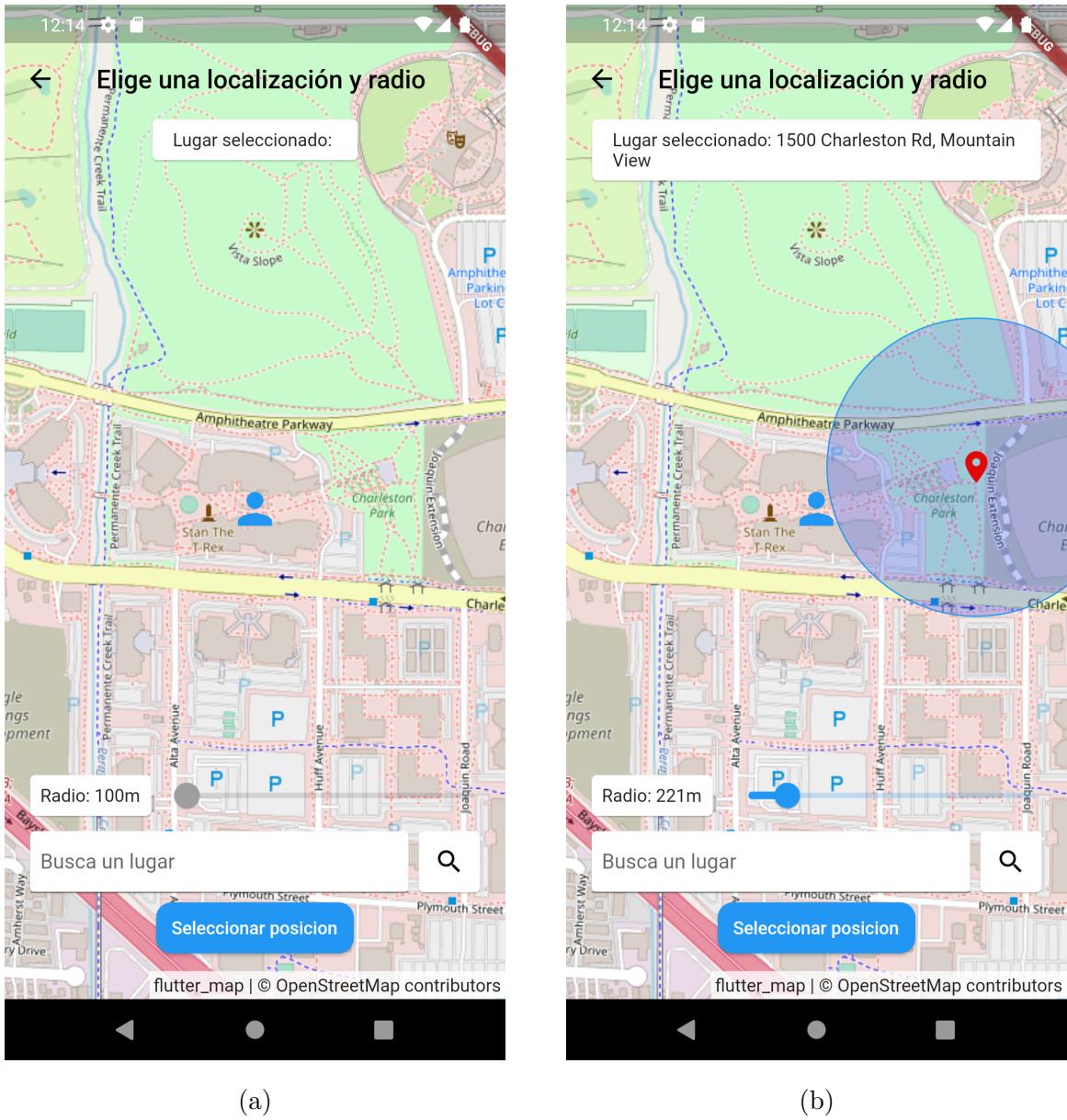


Figura 95: En (a) mapa sin seleccionar posición, en (b) mapa con posición elegida.

Una vez has elegido el origen y el destino tienes que elegir una hora para el viaje. Si seleccionas “Estar a” significa que esperas llegar a tu destino a la hora que especifiques y si seleccionas “Salir a” significa que quieres empezar el viaje a esa hora. Habiendo elegido esto tienes que elegir que día y a que hora quieres que sea el viaje. Puedes planificar viajes hasta dentro de una semana. Ten en cuenta que otros usuarios podrán ver tu viaje hasta 15 minutos después de que haya comenzado o terminado. Por otro lado tu viaje será visible para ti y para tus pasajeros durante las próximas 3 horas de la hora que tenga

el viaje.



Figura 96: Pantalla de crear oferta, selección de plazas y otros

Después en la sección de Plazas y otros tienes que especificar las plazas que tiene disponible tu vehículo para llevar a más personas y si quieres puedes ponerle un título al viaje y una descripción donde puedes especificar el precio del viaje o una descripción de tu coche si quieres. Si no pones un título se pondrá por defecto de título “Viaje a [destino seleccionado]” donde [destino seleccionado] es el nombre del destino que has elegido. Por último, si repites este viaje todas las semanas, puedes seleccionar que el viaje sea recurrente. De esta forma el viaje no se borrará al terminar y será publicado automáticamente de nuevo con la misma hora que has seleccionado pero para la semana siguiente. Con esto ya puedes publicar la oferta pulsando en el botón.

Una vez publicada la oferta puedes hacer clic en ella en la lista de viajes que ofreces para ver los datos con los que la has creado. Si vas al final de la pantalla podrás ver dos botones. Uno para editar la oferta y otro para borrarla. Al pulsa en el botón de editar se abrirá una nueva pantalla donde los campos aparecen con los valores que has establecido y donde puedes cambiar los valores. Ten e cuenta que no podrás borrar disminuir el número

de plazas por debajo del número de pasajeros que ya tiene el viaje. Para hacer esto tendrás que eliminar a algún pasajero desde la pantalla del viaje.

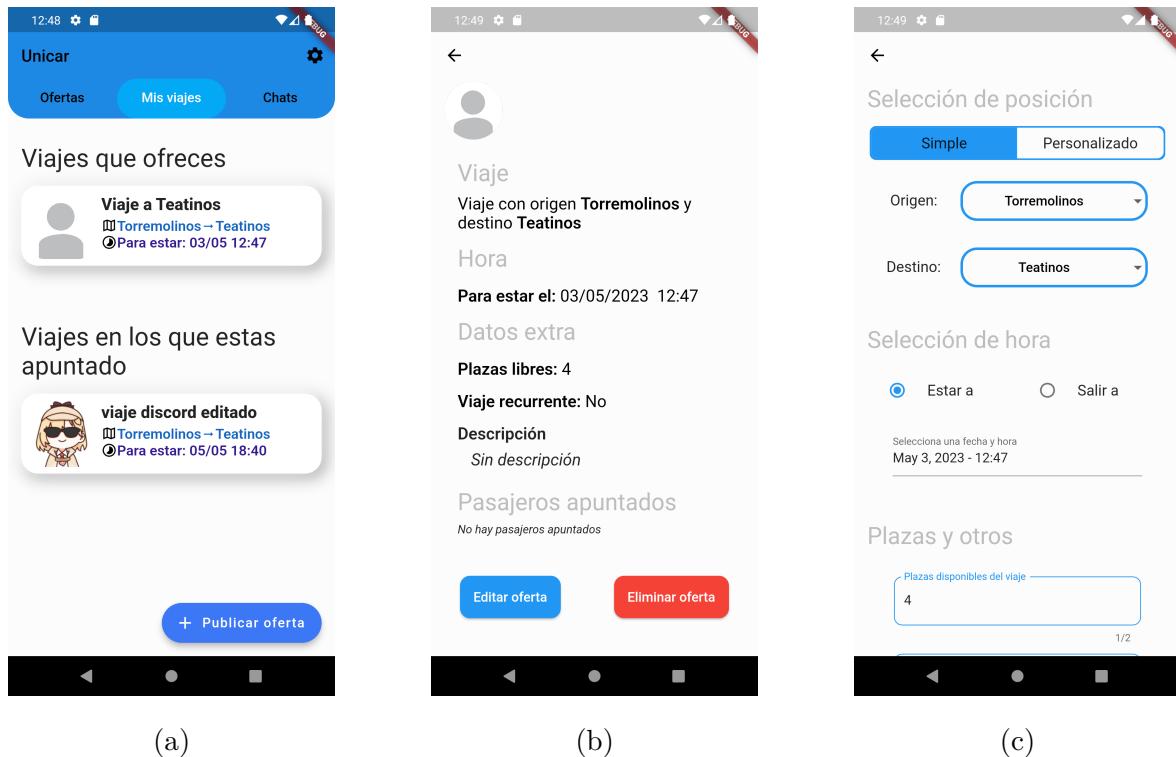


Figura 97: En (a) pantalla de inicio con viajes, en (b) pantalla de viaje que ha creado un usuario, en (c) pantalla de editar.

B.3. Buscar un viaje, verlo y reservarlo

Para poder reservar un viaje, primero hay que dirigirse a la pantalla de “Ofertas” que se encuentra a la izquierda de la pantalla de inicio. En esta pantalla aparecerán todas las ofertas de viaje disponibles. Si quieras buscar un viaje que tenga de salida Torremolinos y de destino Teatinos deberás pulsar en el botón con el texto “Filtrar”. Una vez lo pulses se abrirá una pantalla nueva donde puedes elegir una posición y una hora. Puedes elegir el filtro que quieras y pulsar en el botón con el texto “Aplicar filtros” para aplicar lo que has seleccionado. Si después vuelves a esta pantalla y seleccionas otro filtro más, este se aplica sobre lo que ya habías seleccionado por lo que si quieras hacer una búsqueda con el filtro nuevo tienes primero que pulsar el botón “Eliminar filtros aplicados”. En el caso de filtrar por una posición personalizada, se buscará que la posición elegida con el radio definido haga intersección con el de alguna oferta disponible en algún punto.

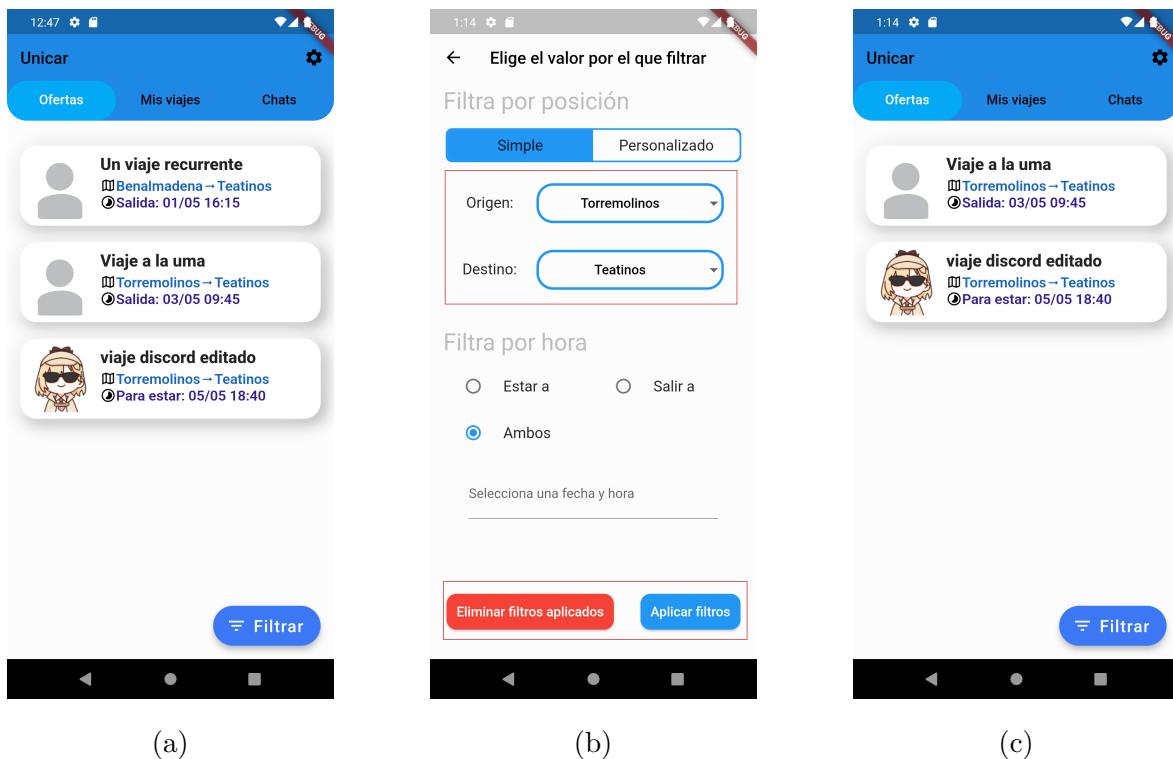


Figura 98: En (a) pantalla de Ofertas sin filtrar, en (b) pantalla filtros, en (c) pantalla de ofertas filtrada.

Una vez has encontrado el viaje que quieras, puedes pulsar sobre él para ver toda la información que tiene. Al final de la pantalla se pueden ver dos botones, un botón

reservar, y un botón para abrir chat. Para apuntarte al viaje deberás pulsar el botón de reservar y el viaje aparecerá entonces en tu lista de viajes en los que estas apuntado. Al hacer clic en el viaje una vez reservado aparecerá un nuevo botón con el que puedes cancelar la reserva siquieres.

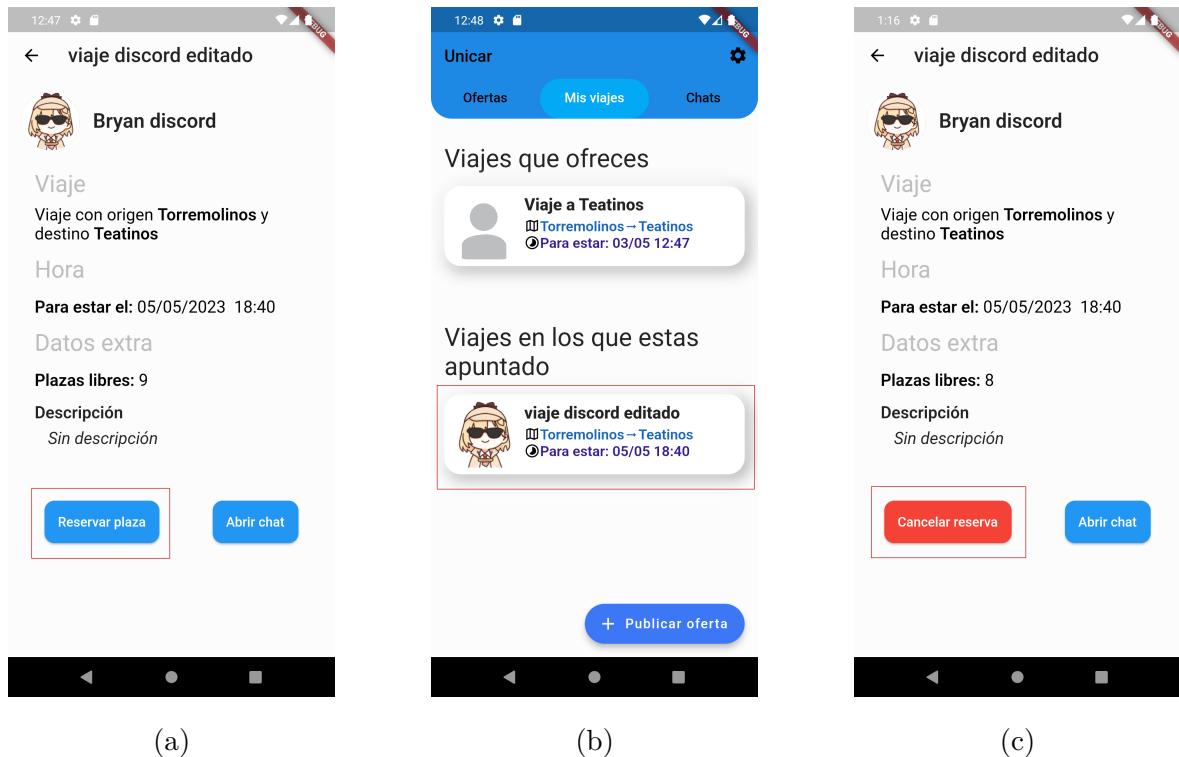


Figura 99: En (a) pantalla de oferta de viaje sin reservar, en (b) pantalla de inicio con un viaje reservado, en (c) pantalla de un viaje reservado.

B.4. Usando el chat

La aplicación tiene una función de chat. Puedes abrir un chat a través de una oferta que haya publicado. No es necesario reservar el viaje para abrir un chat con el usuario que lo publica. Una vez pulsas el botón abrir chat se te llevará directamente a la pantalla de chat donde puedes escribir un mensaje y el otro usuario lo recibirá. A partir de este momento puedes encontrar los chats que tienes en la pantalla de “Chats” a la derecha de la pantalla de inicio.

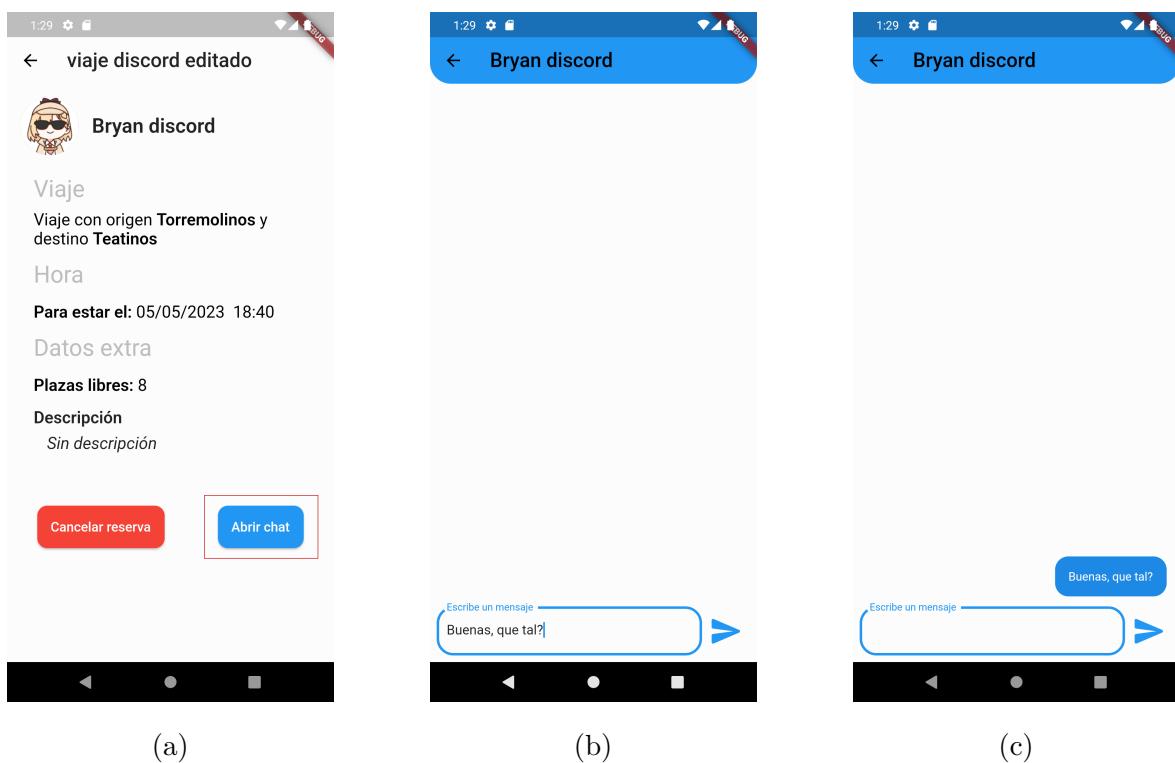


Figura 100: En (a) pantalla de oferta de viaje reservado, en (b) pantalla de un chat sin mensajes, en (c) pantalla de un chat con un mensaje mandado por el propio usuario.

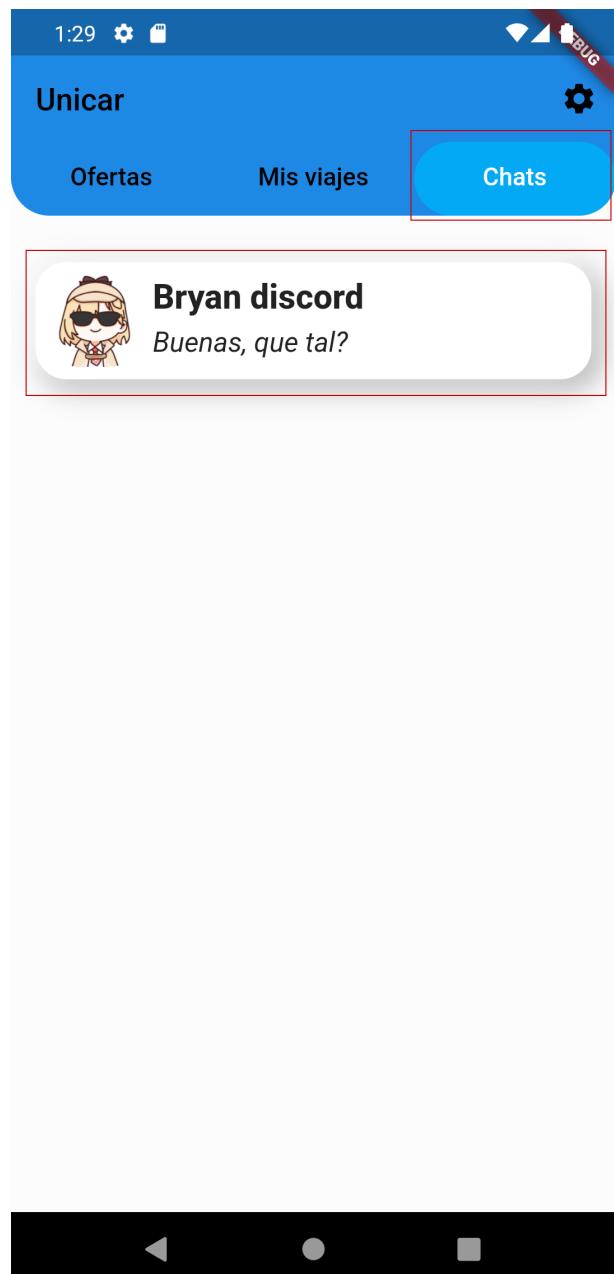


Figura 101: Pantalla de Chats

B.5. Configuraciones

Puedes cambiar tu nombre y establecer datos por defecto desde la pantalla de configuración. Para ello pulsa en el icono de una tuerca que puedes encontrar arriba a la derecha en la pantalla de inicio, Ofertas o Chat.

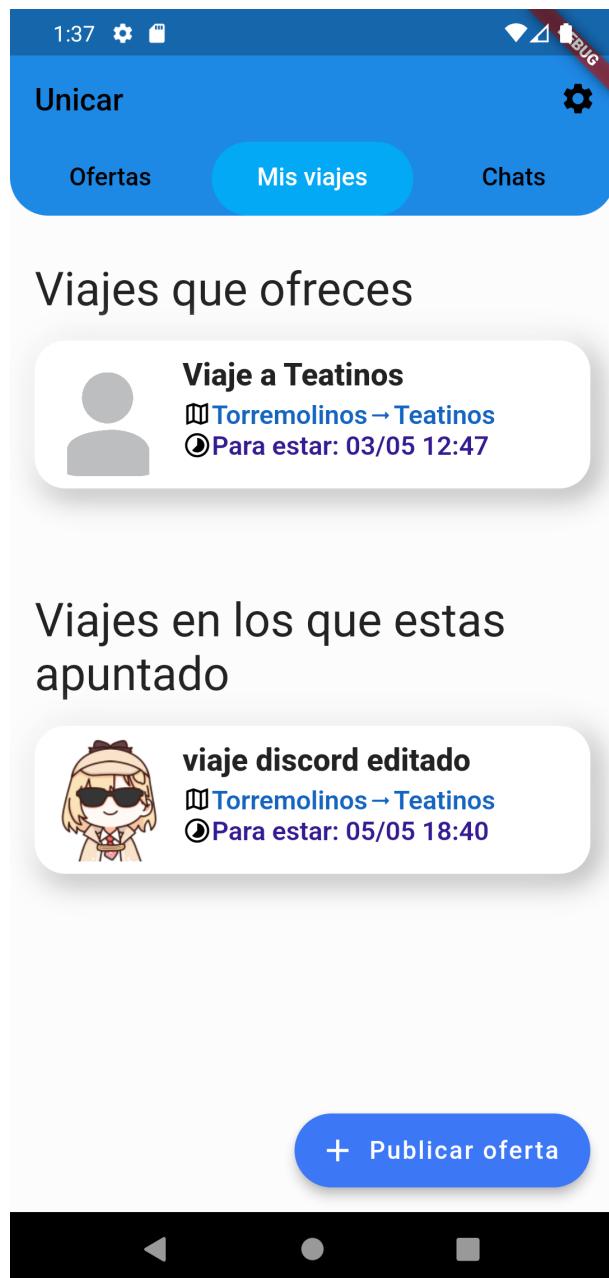


Figura 102: Pantalla de inicio

Una vez dentro verás varias opciones. Puedes cambiar la aplicación a modo oscuro si la estás usando de noche o te gusta más el aspecto, puedes cambiar tus datos por defecto

y también cambiar tu nombre. También encontrarás dos botones. Uno para cerrar sesión que te llevará a la pantalla de iniciar sesión donde podrás entrar con otra cuenta y otro botón para borrar tu cuenta. Ten mucho cuidado con este botón, ya que si borras tu cuenta perderás todos tus datos y no los podrás recuperar.

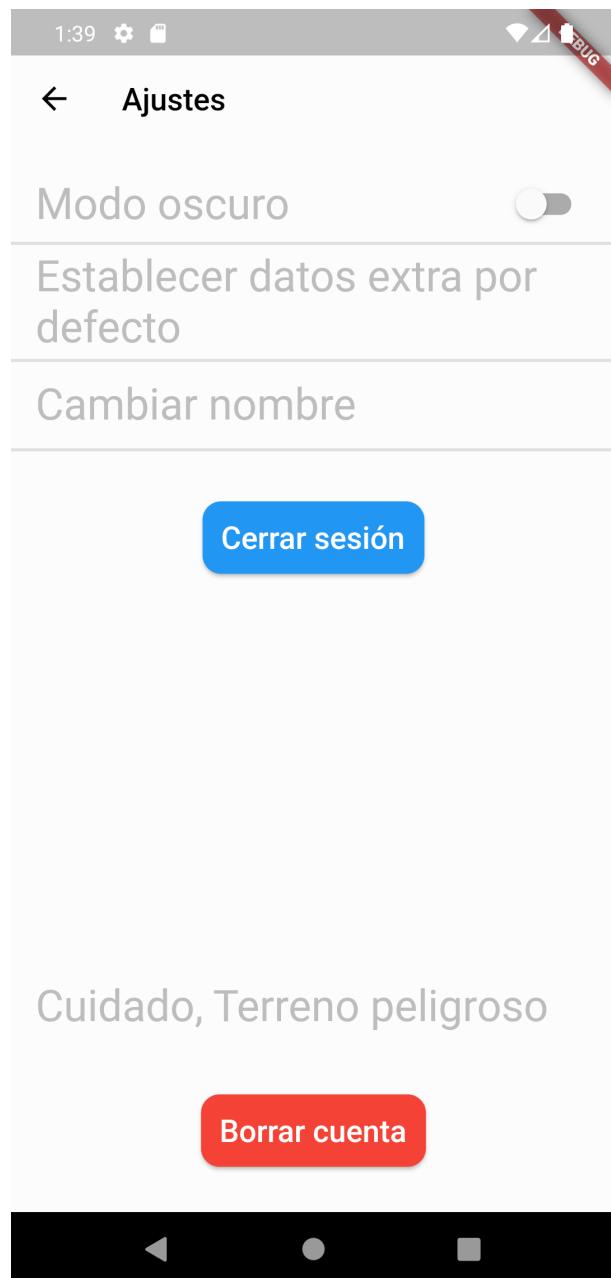


Figura 103: Pantalla de configuración

Apéndice C

Diagramas de casos de uso anteriores

Aquí se muestran algunas versiones anteriores de diagramas de casos de uso antes de llegar a la final.

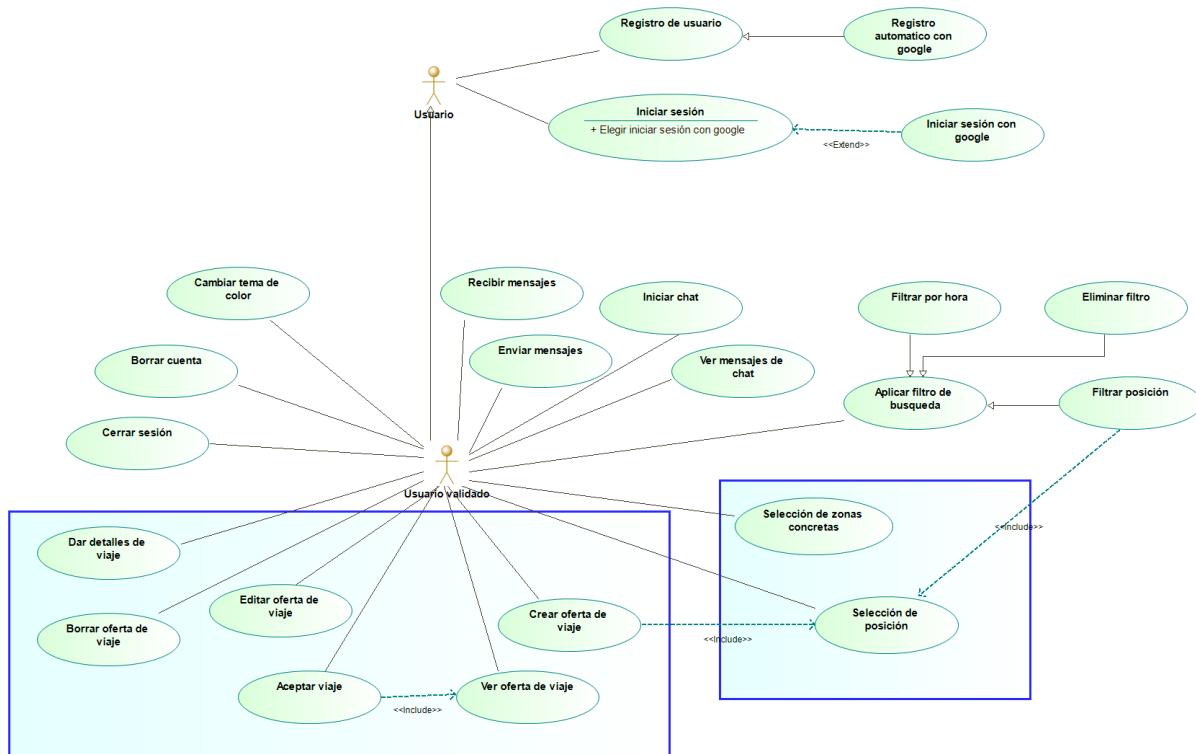


Figura 104: Primera versión del diagrama de casos de uso

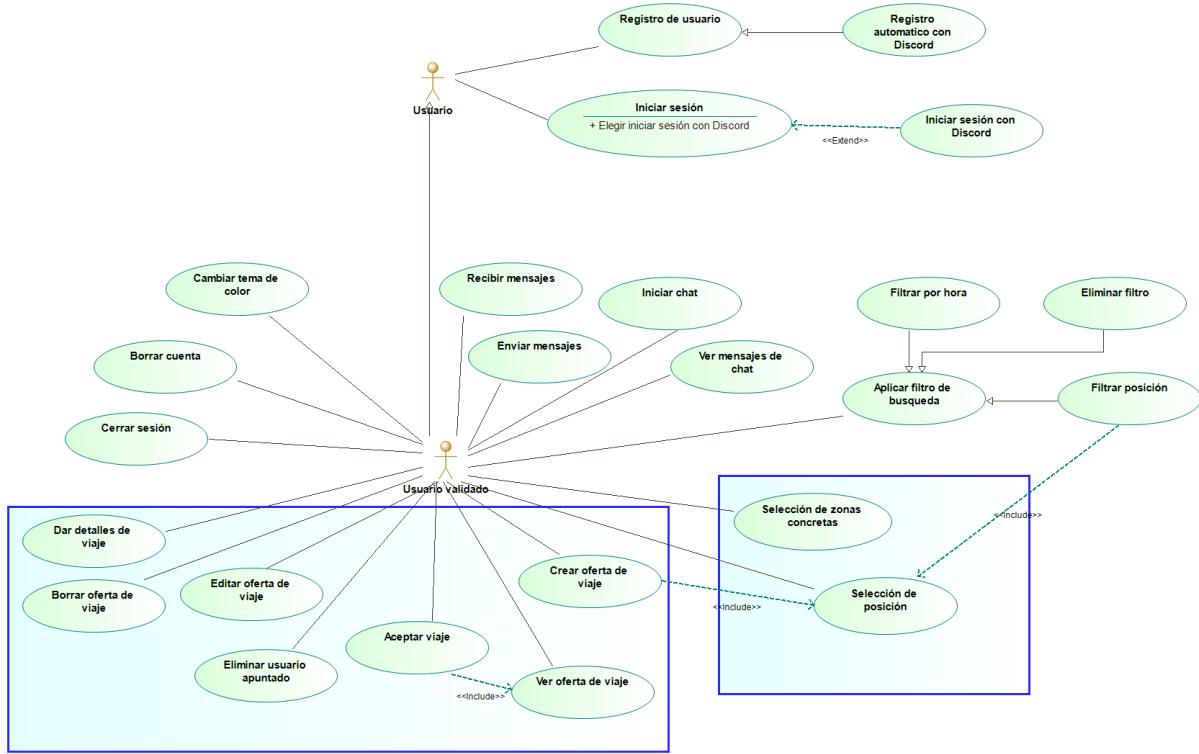


Figura 105: Segunda versión del diagrama de casos de uso

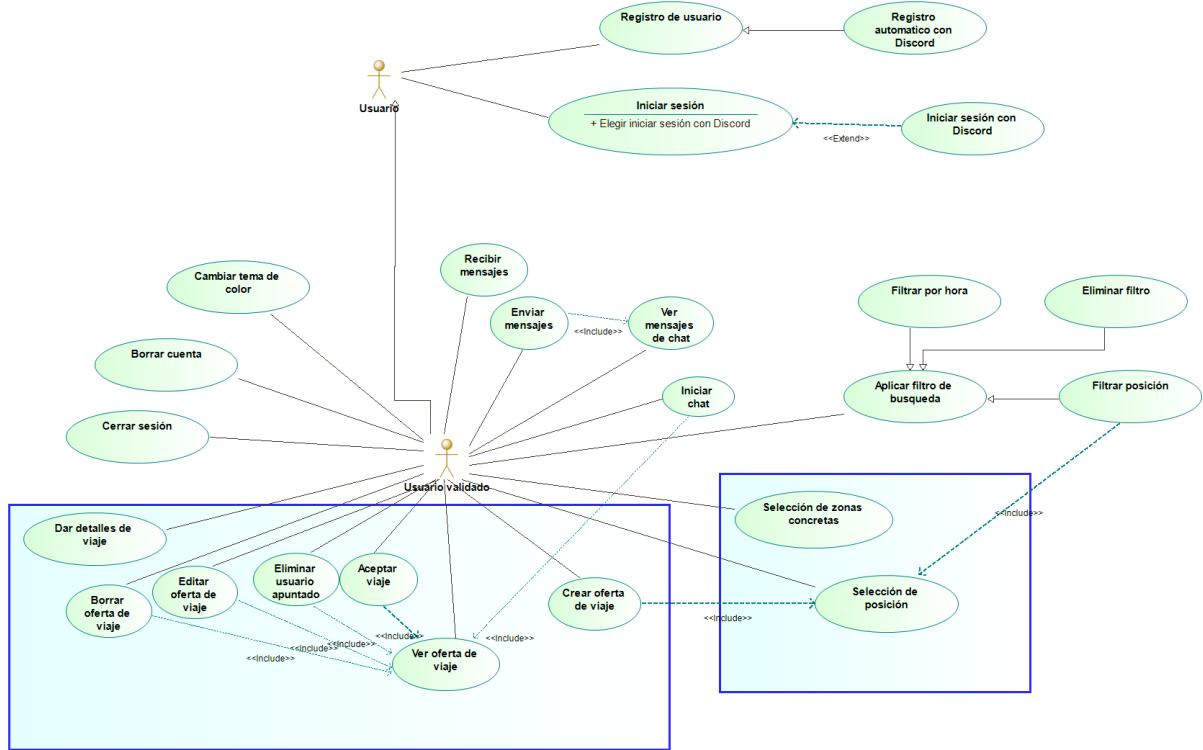


Figura 106: Tercera versión del diagrama de casos de uso

Apéndice D

Diagramas de

bases de datos

anteriores

Aquí se enseñan los diagramas entidad-relación que se han ido haciendo.

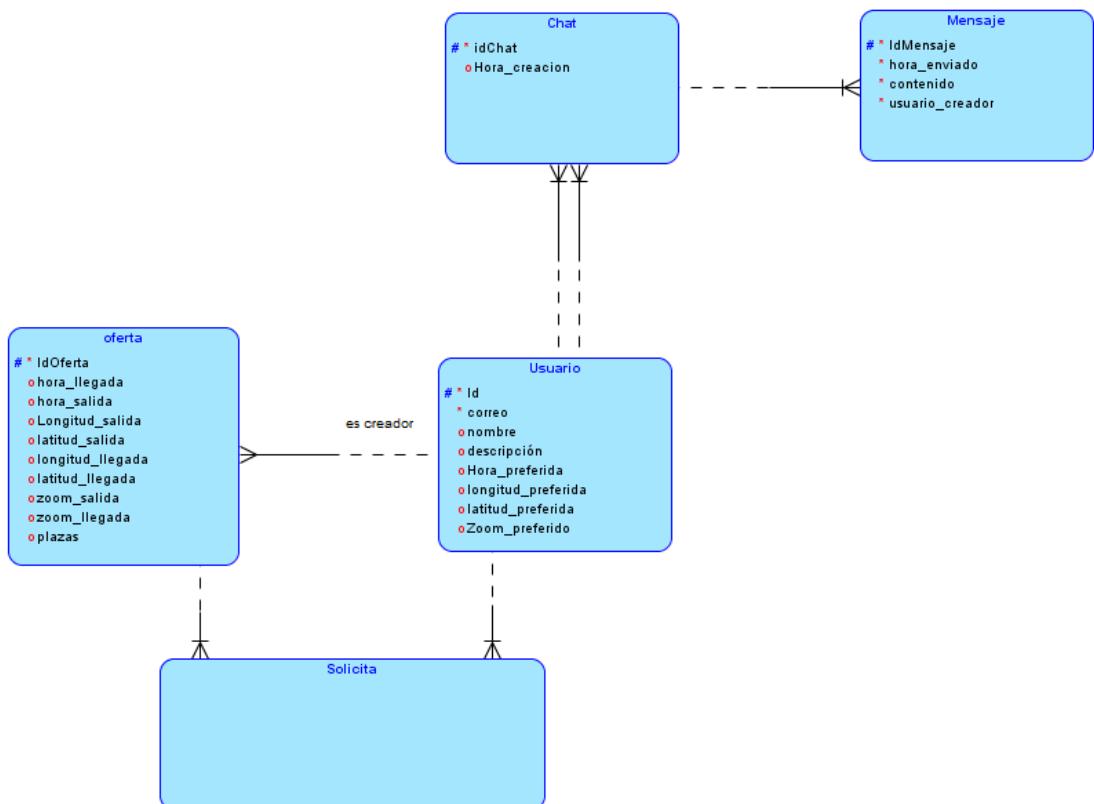


Figura 107: Primera versión del diagrama entidad-relación de bases de datos

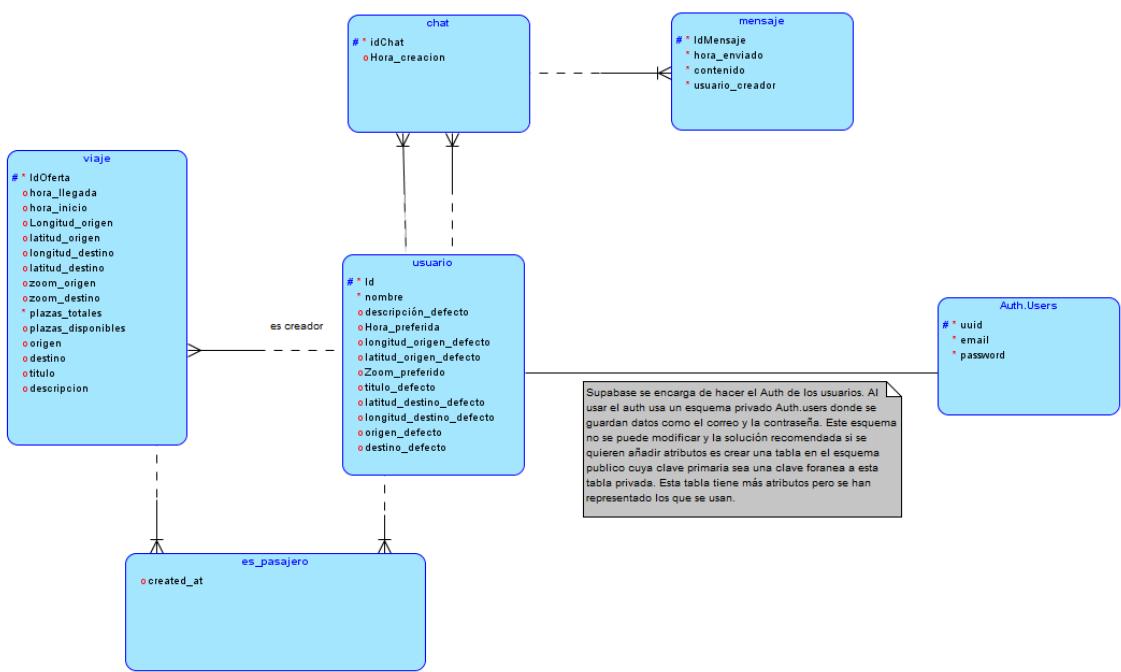


Figura 108: Segunda versión del diagrama entidad-relación de bases de datos

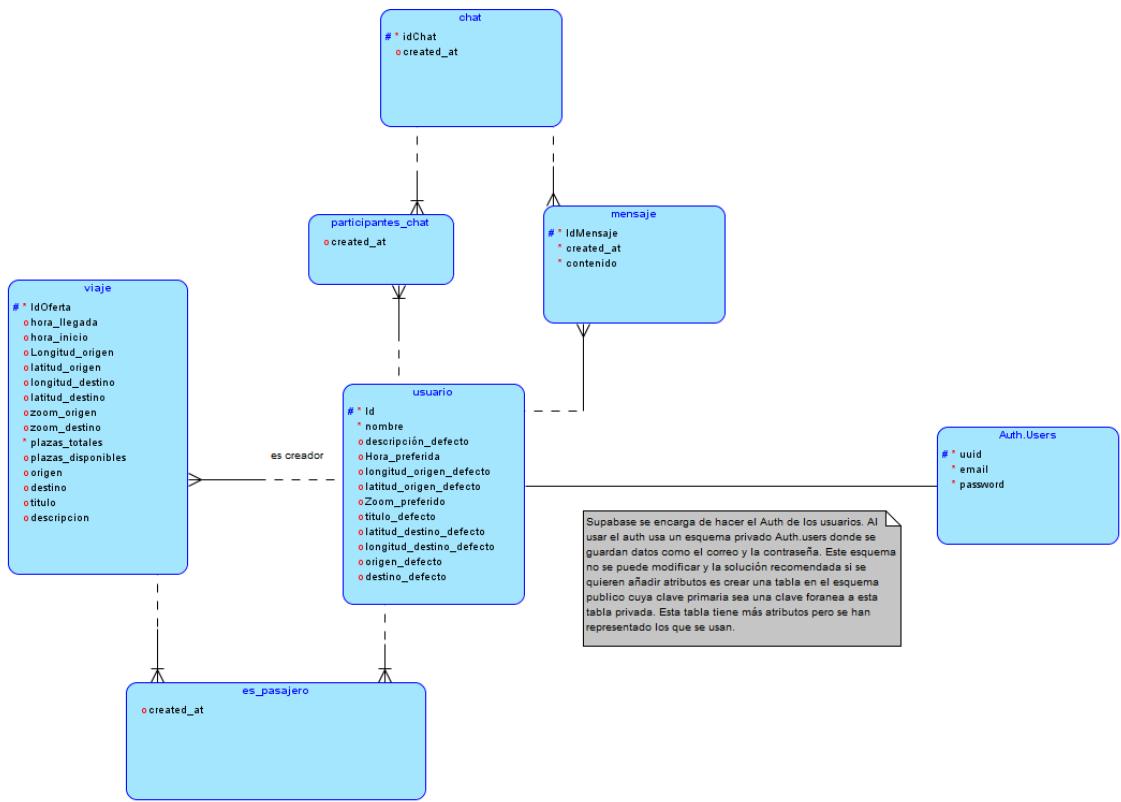


Figura 109: Tercera versión del diagrama entidad-relación de bases de datos

Apéndice E

Diagramas de

clases anteriores

E.1. Datos

Este diagrama es el que se hizo inicialmente. Posteriormente me di cuenta que no era adecuado ya que es más parecido a un diagrama de datos de una base de datos que un diagrama de clase.

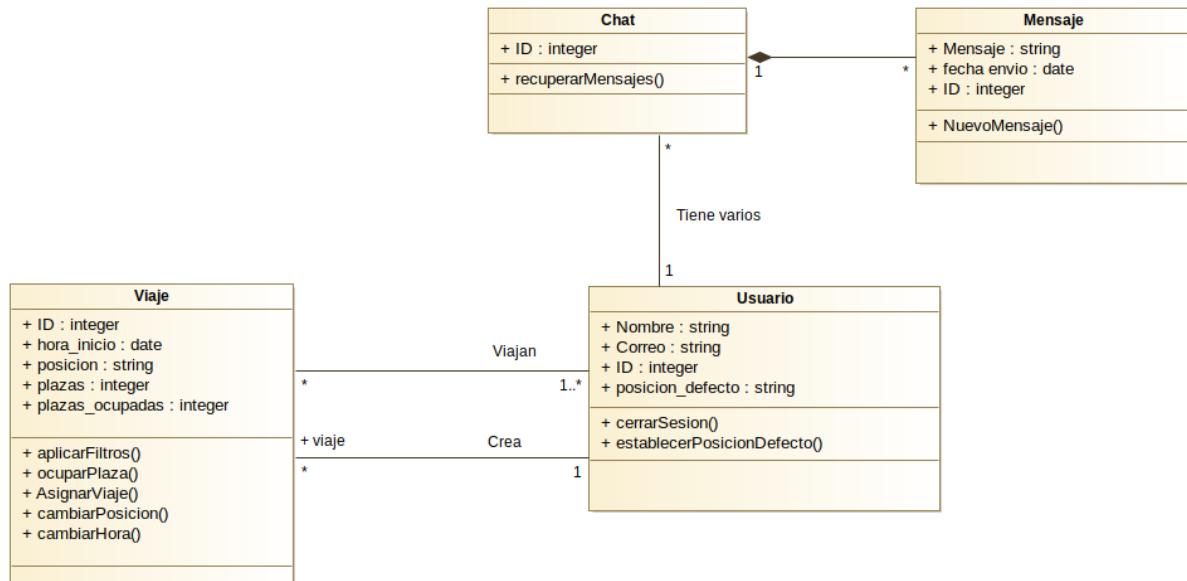


Figura 110: Primera versión del diagrama de clases

E.2. Vista

En mitad del desarrollo cuando ya se tenía cierta estructura para la aplicación se decidió hacer un modelo de vista construido hasta el momento. Ya que es la primera vez que se hacía un proyecto de este tipo, no se sabía como sería la estructura de la vista de la aplicación, por lo que se decidió no construir uno inicialmente.

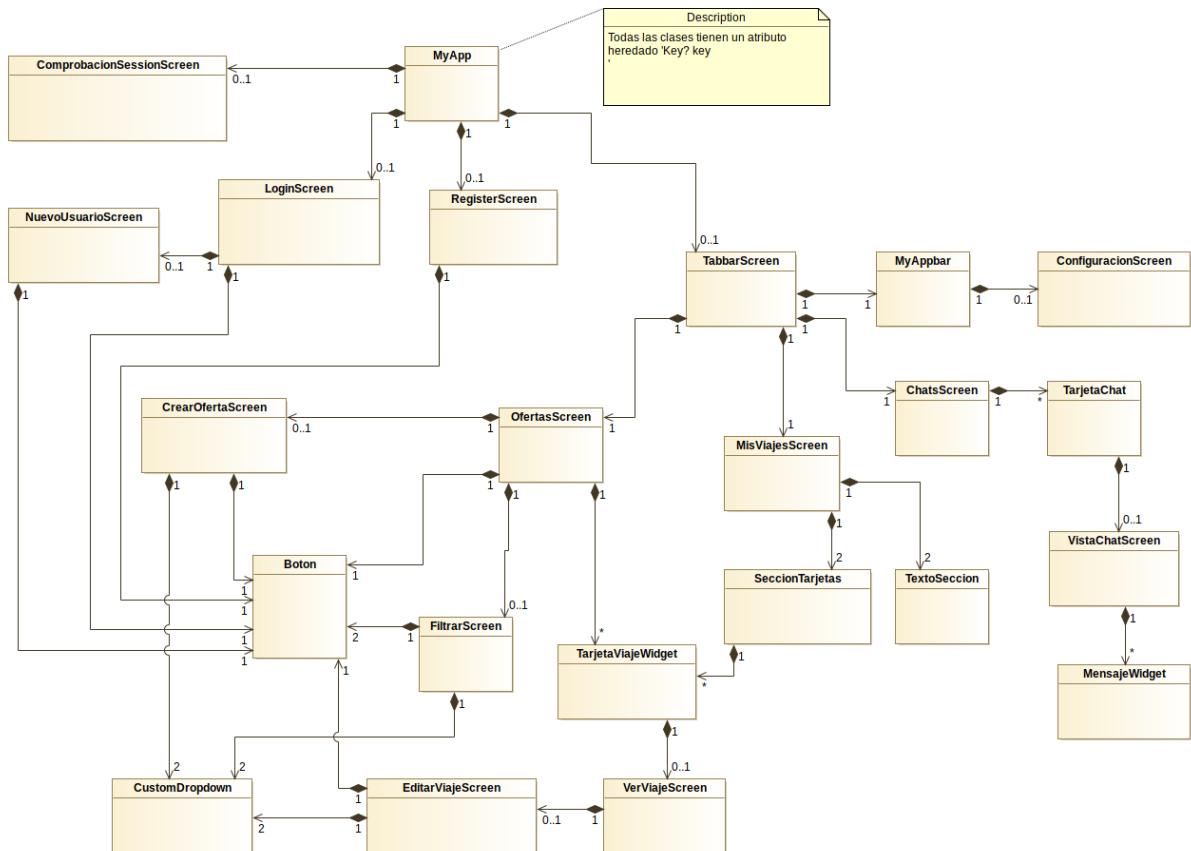


Figura 111: Primera versión del diagrama de la vista

E.3. Control

Al igual que con la vista, no se conocía como hacer el controlador de la aplicación antes de comenzar, por lo que se decidió no realizarlo hasta conocer su funcionamiento.

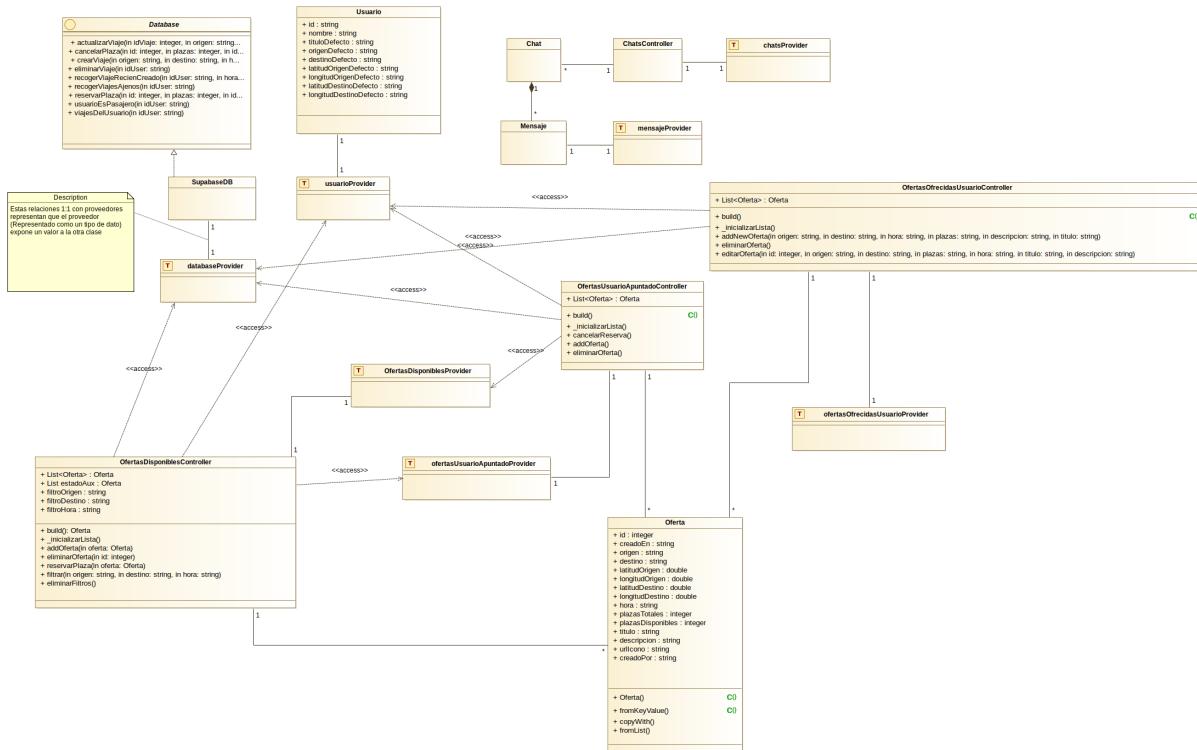


Figura 112: Primera versión del diagrama de control

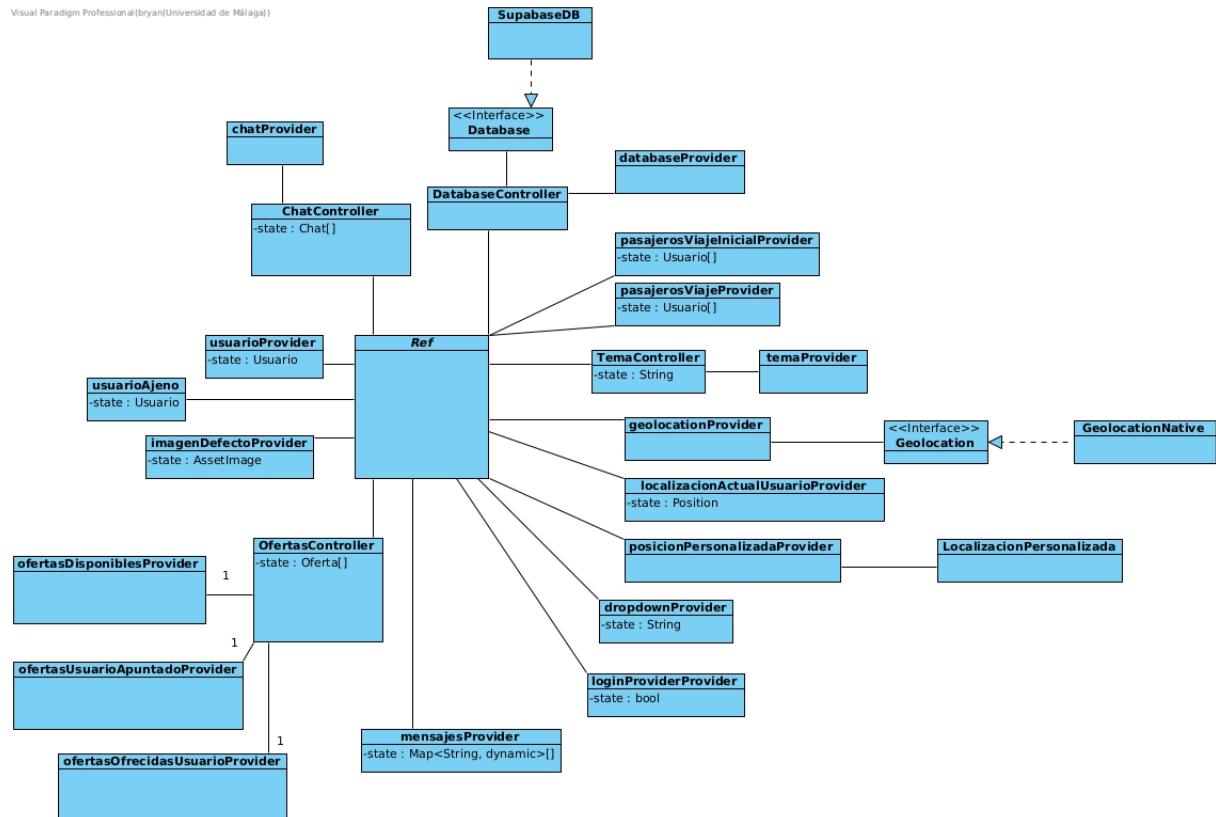


Figura 113: Segunda versión del diagrama de control

Apéndice F

Diagramas de secuencia anteriores

En esta sección se muestra la primera versión de los diagramas de secuencia. No han habido grandes cambios entre la primera versión y la última.

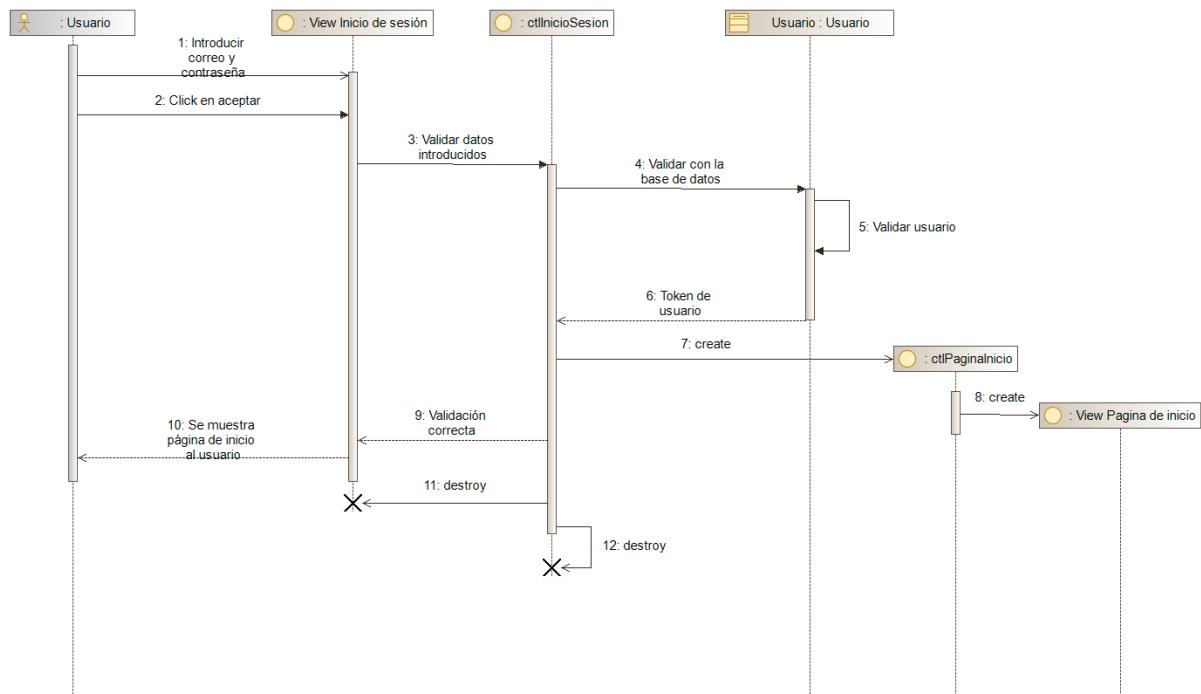


Figura 114: Primera versión del diagrama de secuencia de iniciar sesión

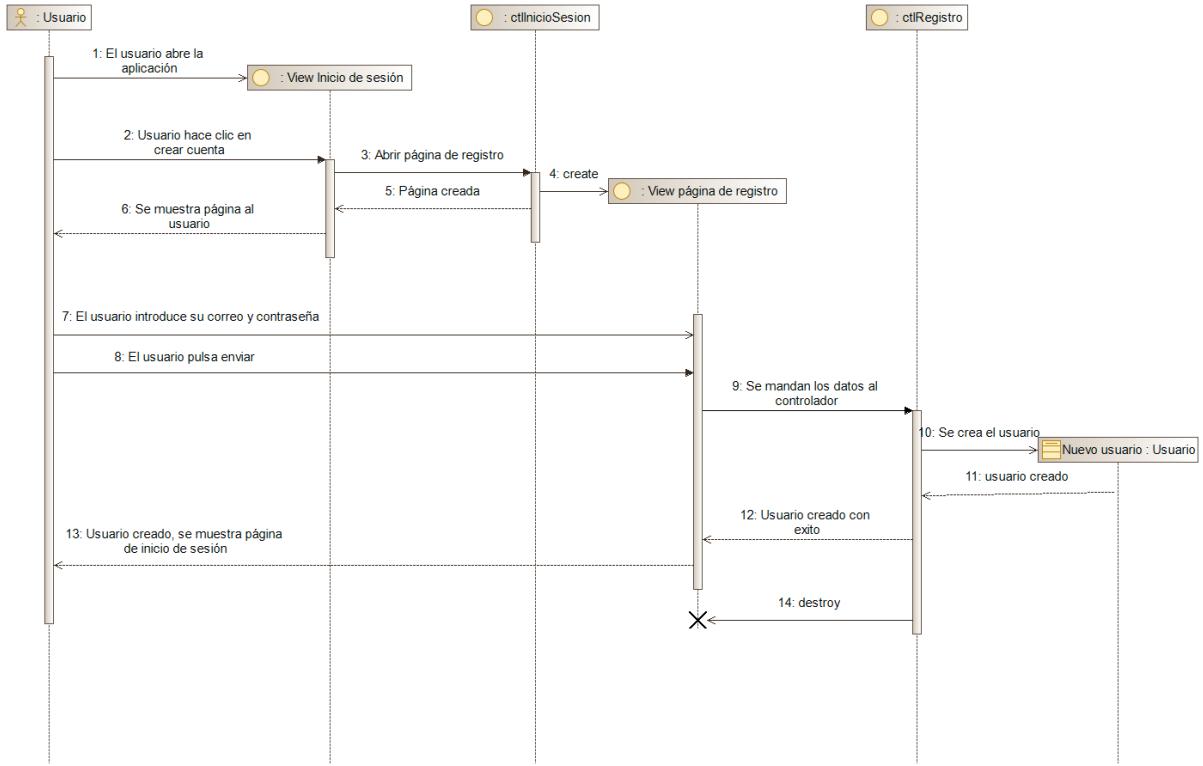


Figura 115: Primera versión del diagrama de secuencia de registro

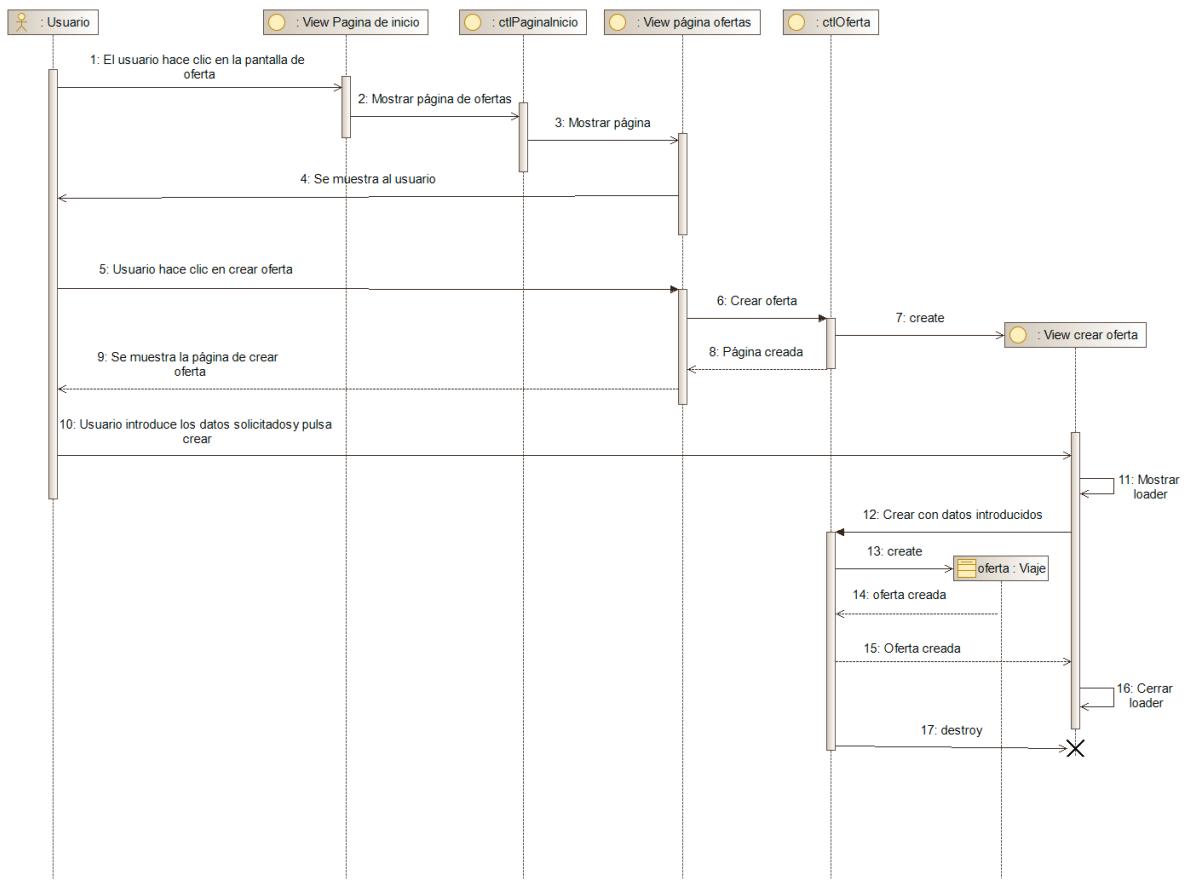


Figura 116: Primera versión del diagrama de secuencia de crear una oferta

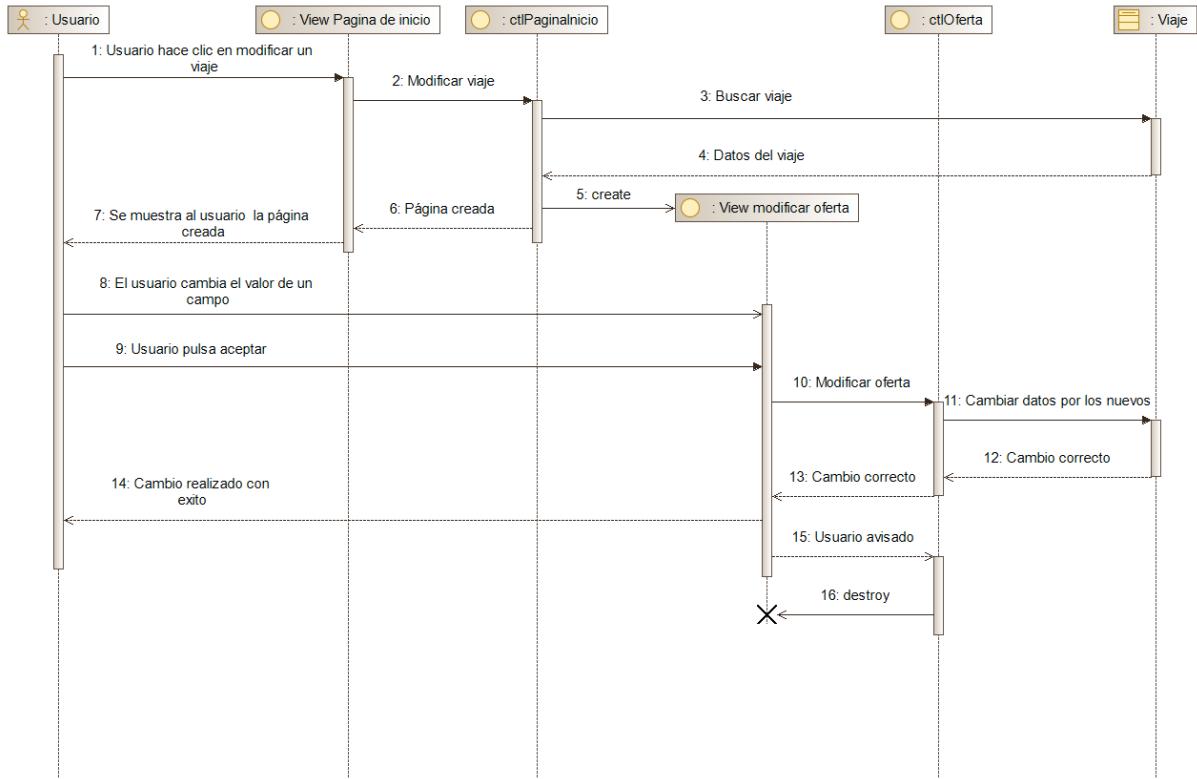


Figura 117: Primera versión del diagrama de secuencia de editar una oferta

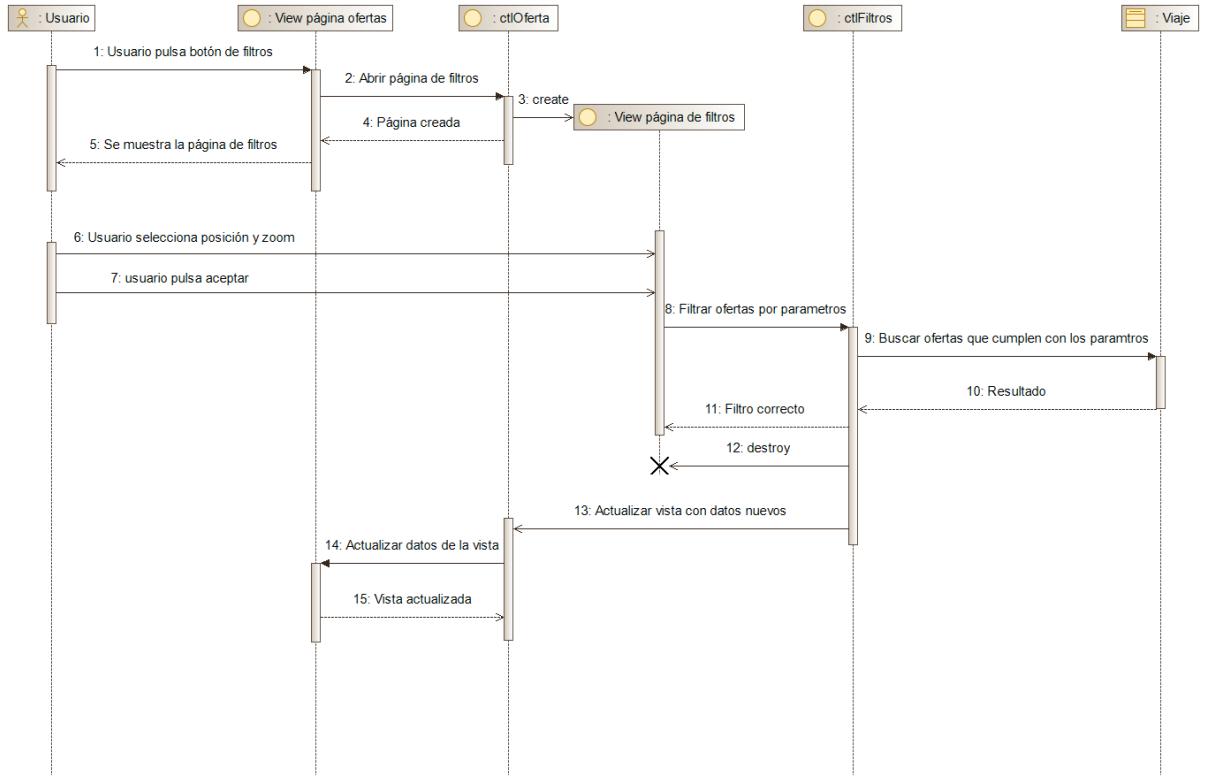


Figura 118: Primera versión del diagrama de secuencia de filtrar por posición

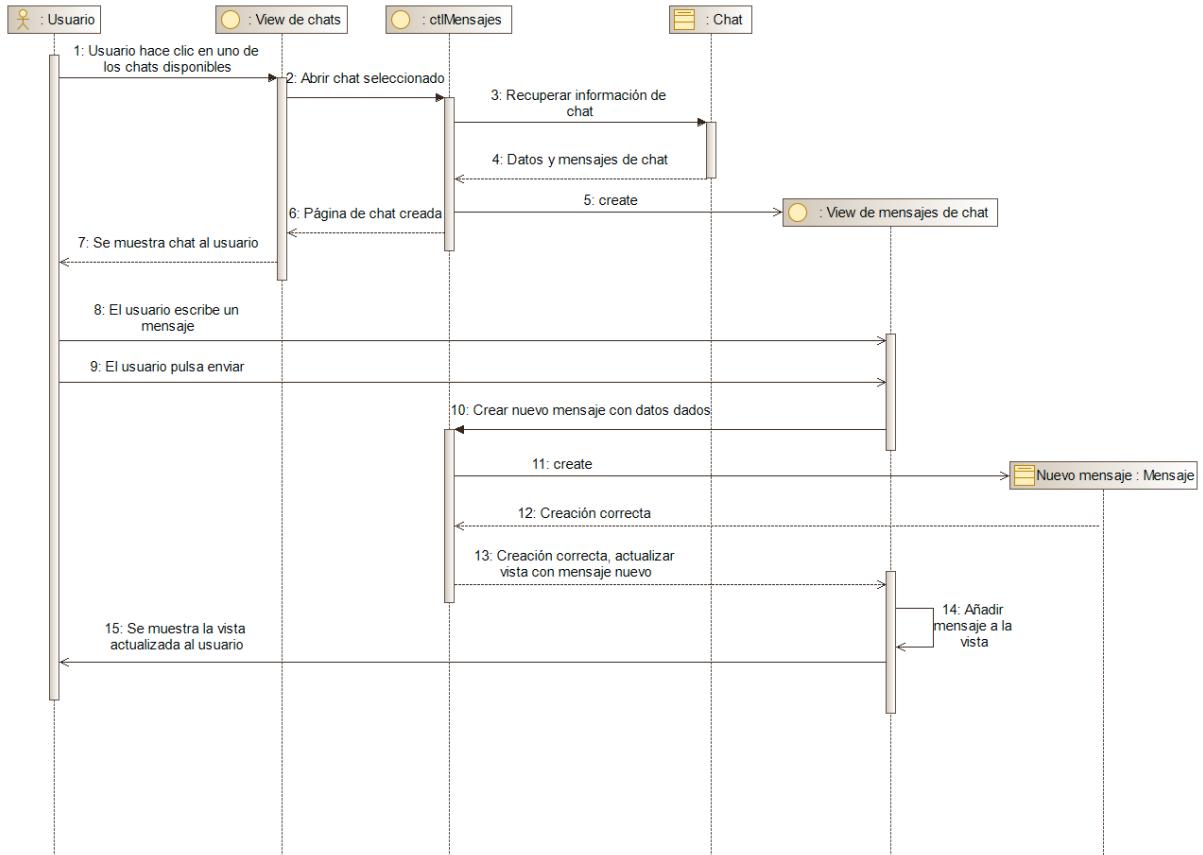


Figura 119: Primera versión del diagrama de secuencia de enviar mensaje

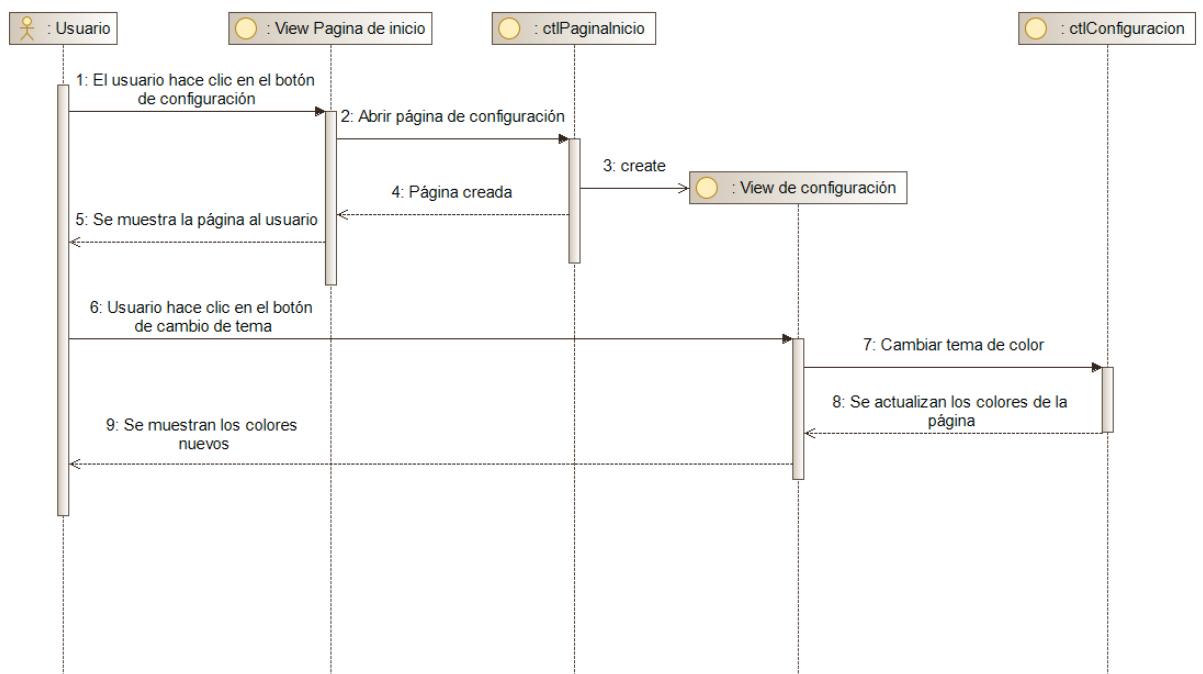


Figura 120: Primera versión del diagrama de secuencia de cambiar tema de color

Apéndice G

Diagramas de estado anteriores

En este apéndice se muestra la primera versión de los diagramas de estado realizados.

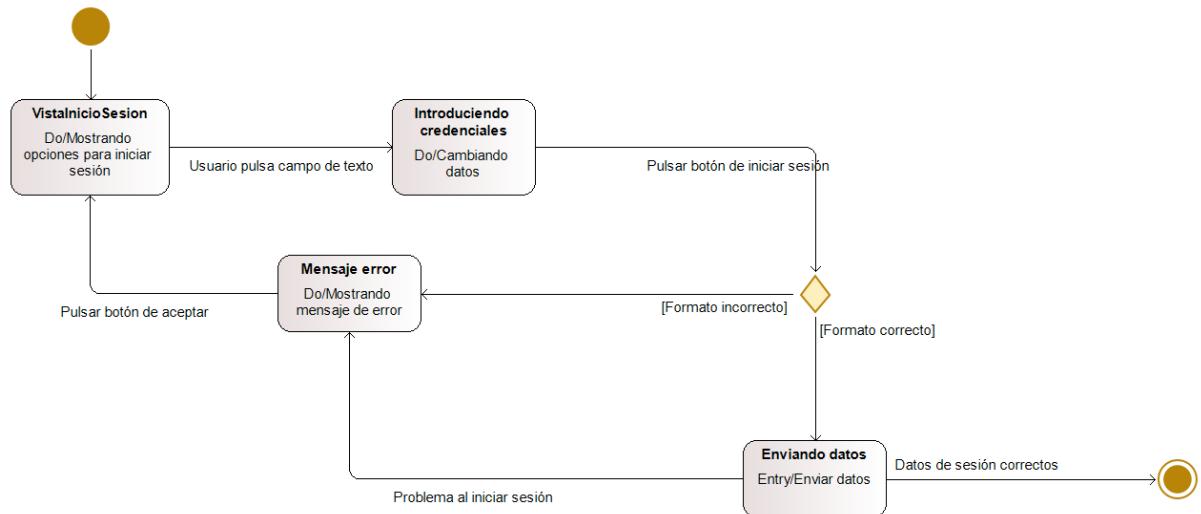


Figura 121: Primera versión del diagrama de estado de iniciar sesión

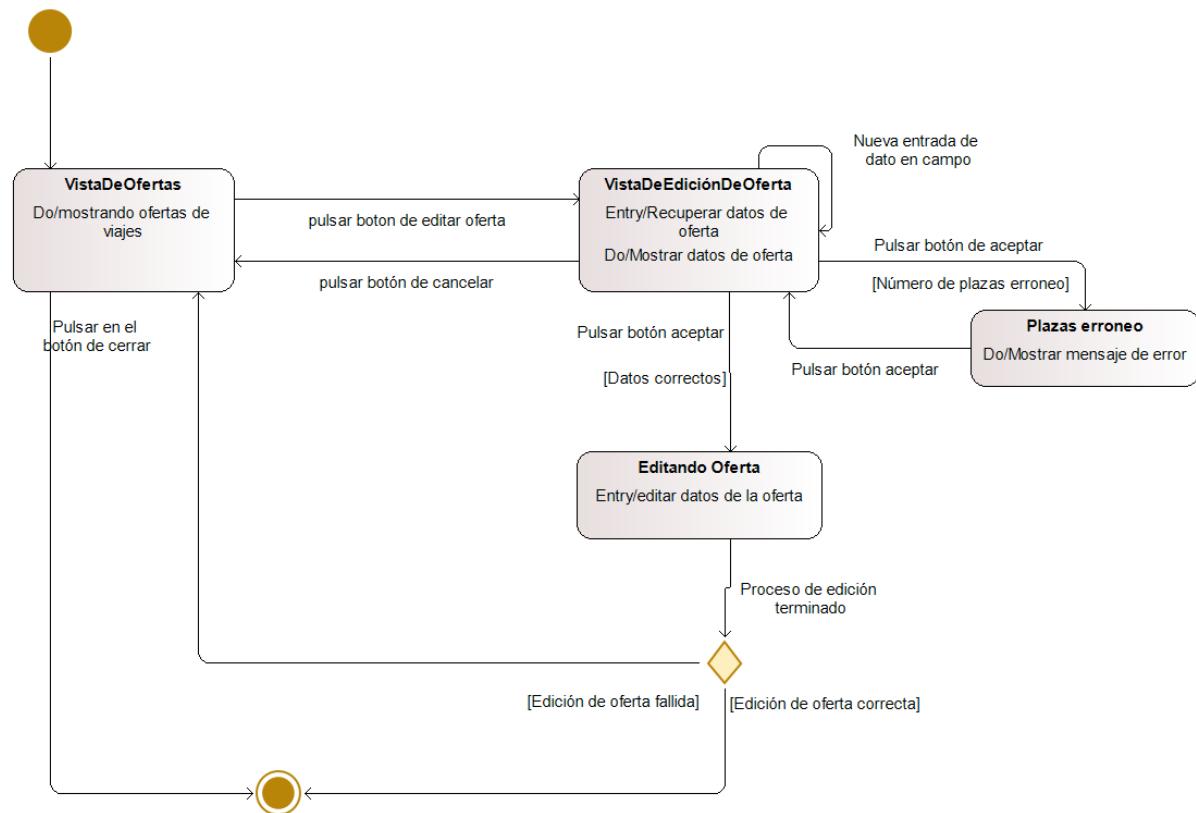


Figura 122: Primera versión del diagrama de estado de editar oferta

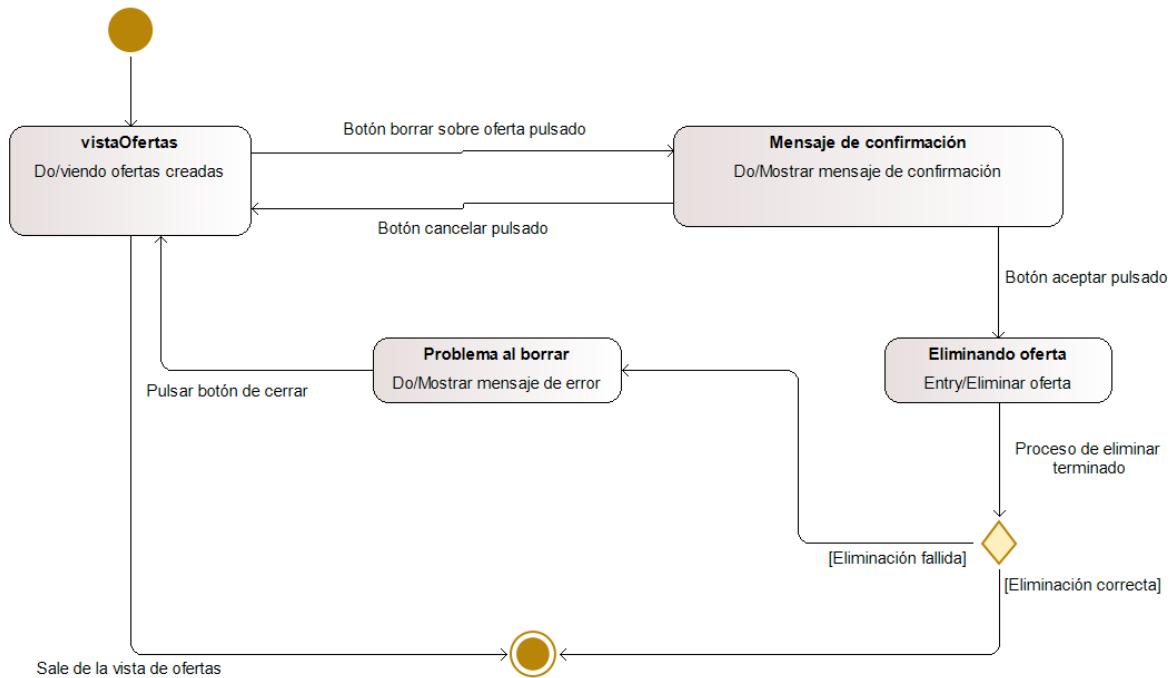


Figura 123: Primera versión del diagrama de estado de borrar oferta

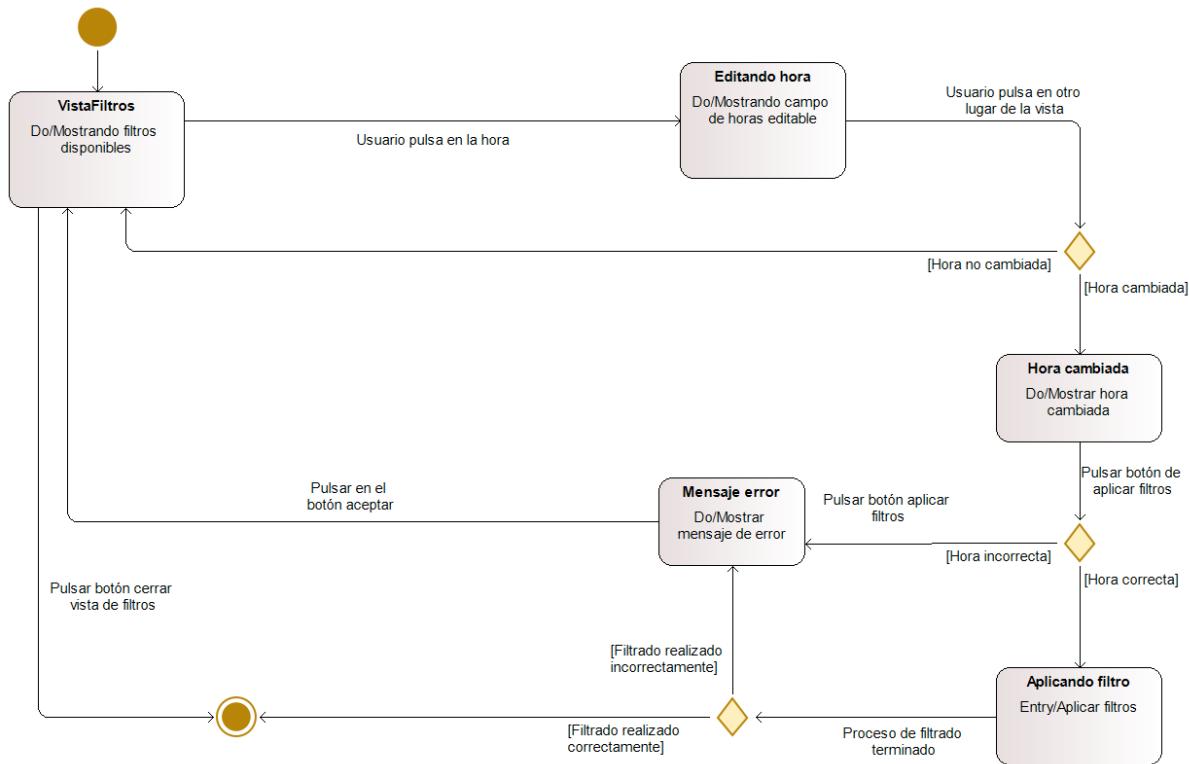


Figura 124: Primera versión del diagrama de estado de filtrar por hora

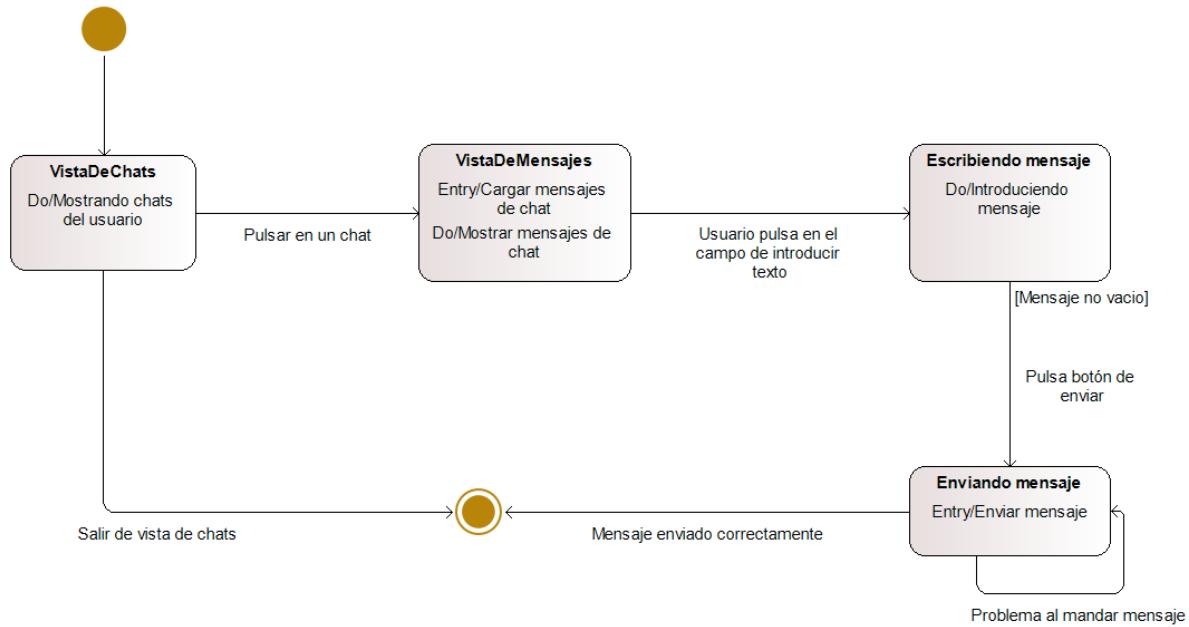


Figura 125: Primera versión del diagrama de estado de enviar mensaje

Apéndice H

Maquetas iniciales de software realizadas

Al principio del proyecto, antes de hacer las maquetas con **Pencil project** se hicieron unos borradores con **Krita** para dejar más claro la idea que se tenía.

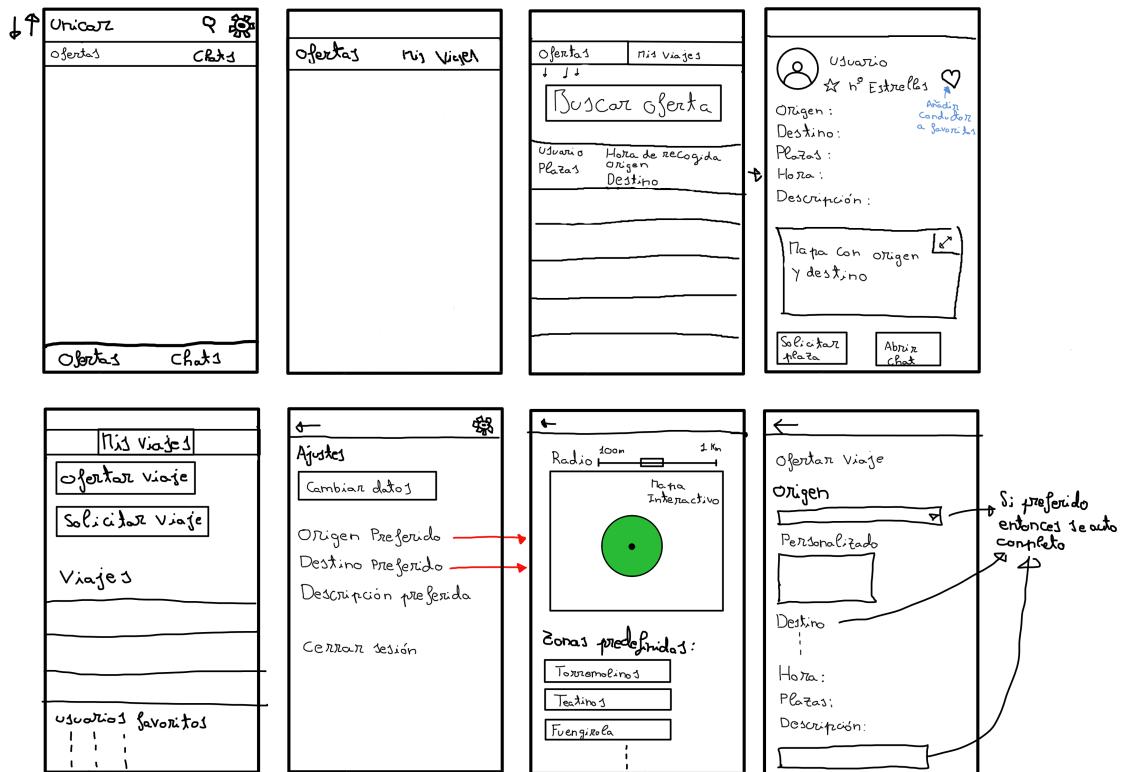


Figura 126: Primera hoja de maquetas

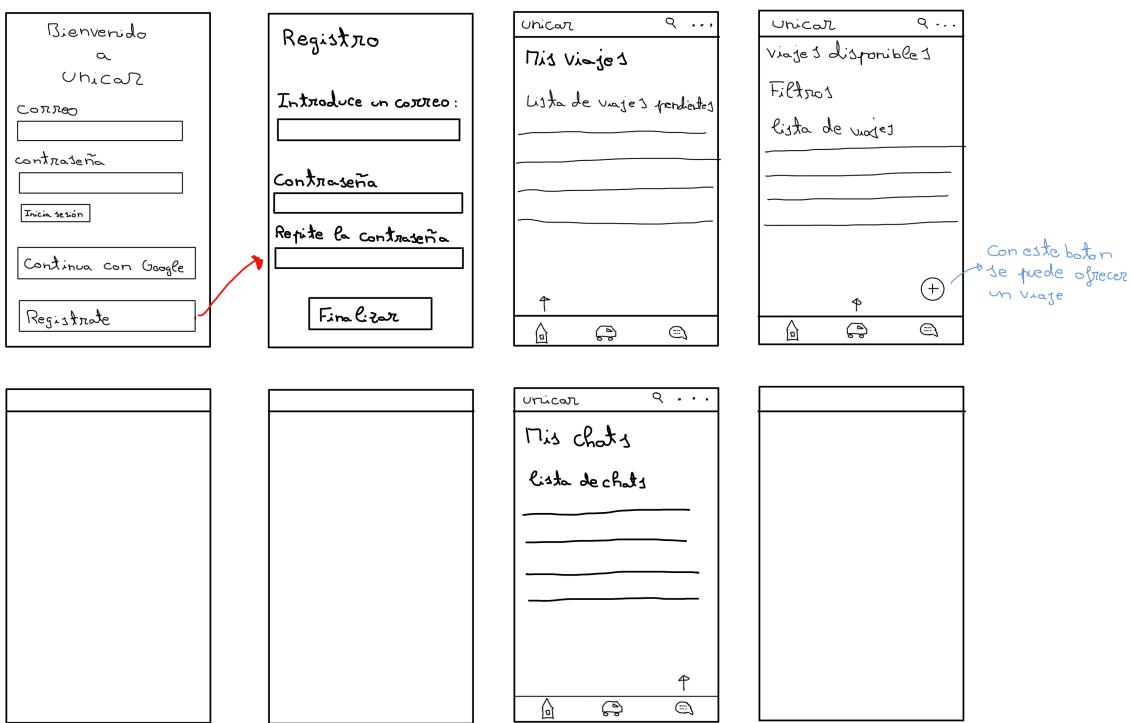


Figura 127: Segunda hoja de maquetas



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga