Universitat Oberta de Catalunya

uoc.edu

Desarrollo de aplicación móvil para facilitar el desplazamiento urbano de personas con movilidad reducida: EasyMov

Álvaro Castillo Cabero
Máster en Ingeniería Informática
Desarrollo de Aplicaciones sobre Dispositivos Móvil.

Jordi Ceballos Villach
Robert Clarisó Viladrosa

09/06/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de aplicación para facilitar el desplazamiento urbano de personas con movilidad reducida: EasyMov*
Nombre del autor:	*Álvaro Castillo Cabero*
Nombre del consultor/a:	*Jordi Ceballos Villach*
Nombre del PRA:	*Robert Clarisó Viladrosa*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Máster en Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo de Aplicaciones sobre Dispositivos Móvil*
Idioma del trabajo:	*Español*
Palabras clave	*Flutter, movilidad, mapa*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

En la actualidad, las principales aplicaciones móviles de mapas no ofrecen la posibilidad de marcar obstáculos para personas con problemas de movilidad en las rutas a pie. Esto es un gran problema, ya que más de 2,5 millones de personas en nuestro país tienen problemas de movilidad. Por ello, he decidido usar el framework de Flutter para crear una aplicación móvil en la que los usuarios puedan marcar barreras arquitectónicas en el mapa, consultables por el resto de los usuarios y que se permita resolverlas, además de poder crear rutas desde la propia aplicación y gestionar el perfil de usuario.

En el proyecto se ha usado Firebase como backend, integrando diferentes APIs de mapas y rutas (como Google Maps o Mapbox) y APIs de gestión de contenido multimedia (Cloudinary). En cuanto al diseño, uno de los patrones de diseño ha sido el patrón BLoC, altamente usado en Flutter.

El resultado del proyecto ha sido una aplicación completamente funcional, desarrollada para Android pero con el código fuente preparado para generar la aplicación para iOS, que cumple con todos los requisitos que se han definido al inicio del proyecto y que, además, ha permitido conocer la tecnología y profundizar en el desarrollo de aplicaciones móvil.

Abstract (in English, 250 words or less):

Currently, the main mobile map applications do not offer the possibility to mark obstacles for people with mobility problems on walking routes. This is a big problem because more than 2.5 million people in our country have mobility affected. For this reason, I have decided to use the Flutter framework to create a mobile application in which users can add architectural barriers on the map, which can be consulted by other users and can be resolved. Furthermore, the users are able to create routes from the application itself and manage the user profile.

In the project, Firebase has been used as a backend, integrating different maps and routes APIs (such as Google Maps or Mapbox) and multimedia content management APIs (Cloudinary). Regarding the design, one of the design patterns used is the BLoC pattern, highly used in Flutter.

Finally, the result of the project is a fully functional application, developed for Android but with the source code prepared to generate the application for iOS. The project has achieved all the requirements that have been defined at the beginning of it and also, it has allowed me to know the technology and deepen the development of mobile applications.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

El siguiente proyecto se trata de un **Trabajo Final de Máster** del área de Desarrollo de Aplicaciones sobre Dispositivos Móvil de la Universitat Oberta de Catalunya (UOC).

Desplazarse por muchas de nuestras ciudades para personas con problemas de movilidad puede suponer toda una odisea, ya que existen multitud de barreras arquitectónicas que impiden que se pueda circular correctamente con una silla de ruedas. Allí nació mi interés por elegir este tema como Trabajo Final de Máster, ya que la gran mayoría de aplicaciones de movilidad no tienen en cuenta estos obstáculos.

A través de este proyecto, además, pretendo entender el framework[1] de Flutter para crear aplicaciones para iOS y Android, profundizar en él para ver cómo se crean y descubrir principales problemas con los que los desarrolladores de aplicaciones se encuentran a la hora de crear dichas aplicaciones.

1.1.1 Descripción general

EasyMov es una aplicación pensada para gente con problemas de movilidad, la aplicación tiene la función de ayudar a estas personas a realizar un recorrido a pie por cualquier ciudad que tenga sus calles definidas en Google Maps. Para ello, en la aplicación se podrá marcar puntos negros en los cuales se encuentre una barrera arquitectónica y las personas con movilidad reducida no puedan pasar o lo tengan excesivamente difícil.

Se ha elegido usar Google Maps ya que tal y como se indica en [1], más de 1.000 millones de usuarios usan mensualmente la aplicación, convirtiéndola en la aplicación de mapas más usada.

1.1.2 Problema a resolver

Actualmente cuando una persona busca cómo realizar un recorrido a pie en Google Maps, no puede ver qué puntos del recorrido son de difícil acceso para personas de movilidad reducida hasta que está desplazándose por la ruta y se lo encuentra. El INE[2] presento este estudio [2] en el que se mostraba que en España hay 2,5 millones de personas con problemas de movilidad, de ellos, el 50% detectan problemas para desplazarse en la calle. El 25% de ellos reporta problemas al subir o bajar aceras y el 20% al cruzar la calle.

[1] Marco de trabajo que facilita la programación.
[2] Instituto Nacional de Estadística.

Por último y a nivel personal tengo muy poco conocimiento en el desarrollo de aplicaciones móviles, por lo que creo que es buen momento para aprender con un proyecto tan interesante.

1.1.3 Solución

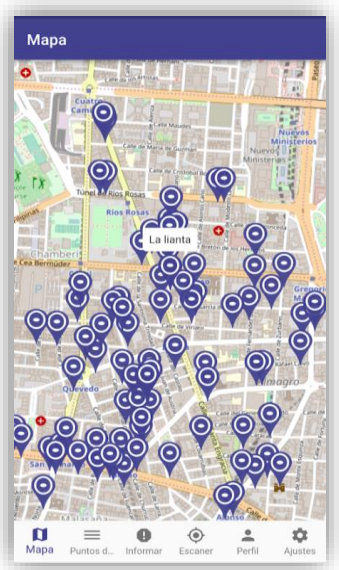
EasyMov propone un sistema de navegación en los usuarios pueden ver en el recorrido que deben realizar los puntos negros de difícil acceso, además, por cada punto que aparece en el mapa pueden ver los detalles del mismo.
La aplicación permite que los usuarios puedan marcar en el mapa puntos negros. Estos puntos negros contienen una breve descripción de la incidencia, la alternativa para evitarlo propuesta por el usuario que reporta el problema, el tipo de obstáculo que se trata así como una fotografía donde se puede ver en detalle el problema.
Los puntos negros son compartidos por todos los usuarios y tienen un sistema de eliminación de puntos negros que hayan sido solucionados para que desaparezcan de la aplicación.

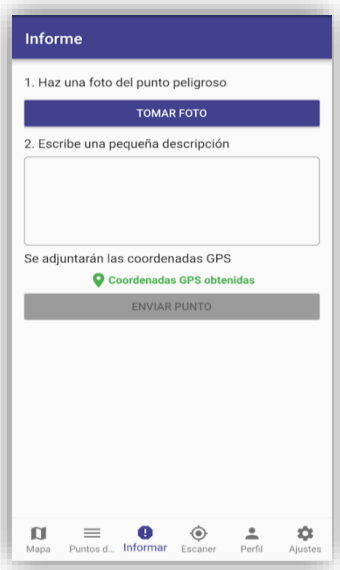
1.1.4 Análisis de mercado

El método de análisis ha sido el uso de Google Play para buscar aplicaciones que cumplieran características semejantes a la mía. A continuación muestro los resultados y una tabla comparativa.

1.1.4.1 Outbarriers

Outbarriers es una app que **proporciona información sobre el entorno a personas con problemas de visión**. Está enfocada hacia puntos de interés y del día a día, tales como restaurantes, escuelas y aseos públicos. Tiene la funcionalidad de poder ver sobre puntos peligrosos para personas ciegas en el mapa de la zona. Además, también permite informar sobre puntos peligrosos, escribiendo un texto y/o colgando una imagen.





1.1.4.2 Disco

DISCO es una aplicación que permite **ver los aparcamientos de coche accesibles para personas con discapacidad motriz**. También es posible ver los aparcamientos reservados para este tipo de discapacidad. También es posible añadir puntos de aparcamientos nuevos a la aplicación que sean aptos para personas con discapacidad.

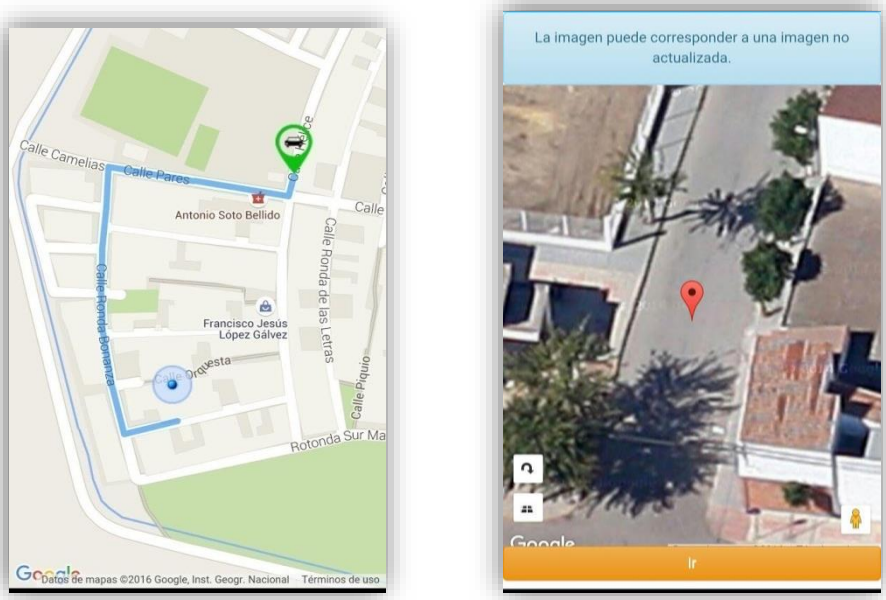


Figura 2. Interfaz Disco

1.1.4.3 Ciudades Patrimonio Accesibles

Ciudades Patrimonio Accesibles es una app creada para que **las personas con discapacidad de diferentes tipos puedan visitar lugares turísticos**. Incorpora una función de búsqueda (por ciudades, por lugar de interés) y también tiene la posibilidad de visitar los lugares más cercanos a la ubicación actual.

Figura 3. Interfaz Ciudades Patrimonio Accesibles

1.1.4.4 Wheelmap

Wheelmap es una app que permite a los usuarios **ver si diferentes puntos de interés son accesibles para personas con movilidad reducida**. La app permite a los usuarios valorar si un punto es accesible en tres niveles: completamente accesible, parcialmente accesible o no accesible. También permite añadir comentarios a las valoraciones.

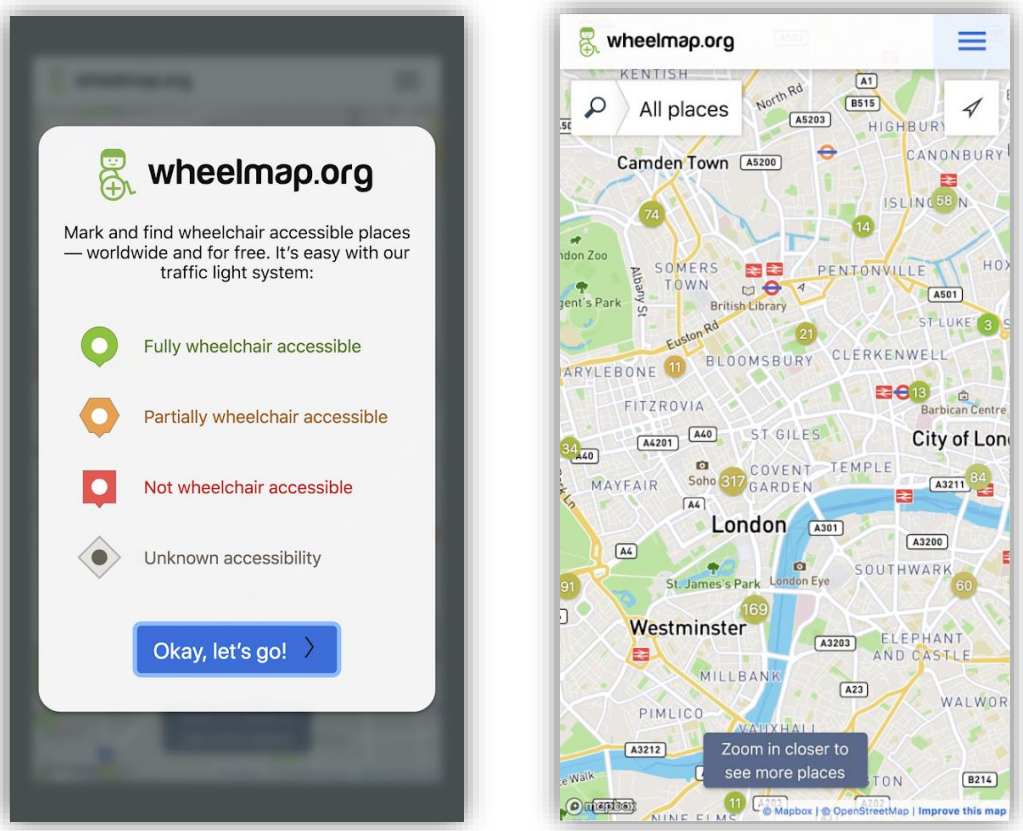


Figura 4. Interfaz Wheelmap

1.1.4.5 Tabla comparativa

	Outbarriers	Disco	CiudPatrAcc	Wheelmap	EasyMov
Permite ver puntos de interés en el mapa	✓	✓	✓	✓	✓
Muestra Información detallada de los puntos	✗	✗	✓	✓	✓
Permite añadir diferente tipo de puntos	✗	✗	✗	✓	✓
Permite colgar nuevos puntos (texto)	✓	✓	✗	✓	✓
Permite colgar nuevos puntos (foto)	✓	✗	✗	✓	✓
Permite borrar puntos (o solicitar el borrado)	✗	✗	✗	✗	✓
Permite buscar por dirección	✓	✗	✓	✗	✓
Permite ver parkings accesibles	✗	✓	✗	✗	✗
Permite realizar rutas	✗	✓	✗	✗	✓

Tabla 1. Comparativa de las Apps del Mercado.

1.2 Objetivos del Trabajo

La funcionalidad de la aplicación es ofrecer un mapa con los obstáculos que se pueden encontrar en la ciudad las personas con dificultades de movilidad, permitiendo añadir nuevos y crear rutas dentro de la propia app. Por otra parte, se permitirá al usuario marcar como resueltos los puntos registrados por otros usuarios y por él mismo.

1.2.1 Objetivos de la aplicación

Se han definido diversos objetivos que deberá cumplir nuestra aplicación, que son los siguientes:

- O1: Hacer Registro / Login.
- O2: Modificar perfil de usuario y ajustes.
- O3: Geolocalizar al usuario.
- O4: Gestionar creación y resolución de obstáculos.
- O5: Mostrar mapa con información útil.
- O6: Crear ruta en el mapa.

1.2.1 Requisitos funcionales

Los requisitos funcionales de nuestra aplicación los definiremos en la siguiente tabla:

Requisito Funcional	Objetivo relacionado
RF1. Gestión de usuarios y perfiles	
RF1.1 – Registrar cuenta de usuario	O1
RF1.2 – Login con usuario existente	
RF1.3 – Cerrar sesión del usuario	
RF1.4 – Crear perfil de usuario	O2
RF1.5 – Modificar perfil de usuario (Nombre y foto)	
RF1.6 – Modificar ajustes de la aplicación	
RF2. Geolocalización	
RF2.1 – Gestionar los permisos de ubicación de la app	O3
RF2.2 – Geolocalizar la posición actual del usuario en el mapa	
RF3. Gestión de obstáculos	
RF3.1 – Permitir al usuario crear un obstáculo	O4
RF3.2 – Permitir resolver un obstáculo propio o de otro usuario	
RF3.3 – Mostrar obstáculos del usuario	
RF4. Gestión del mapa	
RF4.1 – Mostrar mapa con obstáculos	O5
RF4.2 – Mostrar información del obstáculo al hacer click	
RF4.3 – Permitir al usuario indicar un destino.	O6
RF4.4 – Guiar al usuario con una ruta hasta el destino.	
RF4.5 – Mostrar ruta que está haciendo el usuario al moverse	

Tabla 2. Requisitos funcionales

1.2.2 Requisitos no funcionales

También es importante detallar los requisitos no funcionales, es decir, las propiedades y características que a pesar de no aportar una utilidad, hacen la aplicación más usable, estética o disminuyen la curva de aprendizaje, entre otros, a continuación los detallamos:

1.2.2.1 Rendimiento

Cualquier funcionalidad y transacción del sistema que interactúe con el usuario no debe tardar en responder más de 5 segundos.
Los datos de la base de datos, no pueden tardar más de 5 segundos en actualizarse.
La aplicación debe poder correr en cualquier dispositivo Android con versión 7.1 o posterior.

1.2.2.2 Disponibilidad

El sistema debe estar disponible en el 99% de los días del año.
La base de datos debe estar disponible en el 99% de las ocasiones.

1.2.2.3 Estabilidad

El contexto del sistema se debe poder mantener al cerrar y abrir la aplicación.
El sistema no se puede desconectar o cerrar de manera repentina por un error.

1.2.2.4 Funcionalidad

Un usuario estándar, no puede tardar más de una hora en aprender a usar la aplicación.
El sistema debe informar al usuario cada vez que esté realizando una tarea que pueda tardar unos segundos.
Las interfaz debe estar diseñada de tal manera que cualquier funcionalidad sea entendida en menos de 10 minutos para el usuario estándar.

1.2.2.5 Estética

El sistema no puede tener colores antagónicos en la escala cromática contiguos.
Los colores en sí no pueden tener un significado fundamental para entender la aplicación, por la gente que sufre daltonismo[3].
El diseño debe ser atractivo visualmente, en un 90% de los casos, el usuario estándar no debe estar desalentado o limitado por el diseño visual.

1.2.2.6 Dispositivos a los que se dirige

La aplicación está pensada para ser usada en **iOS y Android**, aunque en solo se genera el paquete de Android está preparada para ser exportada en iOS.

[3] Alteración de origen genético que afecta a la capacidad de distinguir los colores.

1.2 Enfoque y método seguido

Se ha decidido **realizar una aplicación desde cero**, el principal motivo es que debido a mi poca experiencia en general en el desarrollo de aplicaciones móviles y en particular en el desarrollo con Flutter, creí oportuno plantear este proyecto como un aprendizaje integro tanto del proceso de definición de un proyecto como en el proceso de desarrollo.

Tras darle vueltas al tema sobre el cual hacerla, me decanté por **indicar problemas de movilidad** ya que me parecía interesante debido a que es una realidad en nuestras ciudades y me permitiría cubrir muchos de los aspectos que usan la gran mayoría de aplicaciones, como la conexión con bases de datos, el uso de APIs como la de Google Maps y que el propio usuario sea el que genere el contenido del sistema.

1.4 Planificación del Trabajo

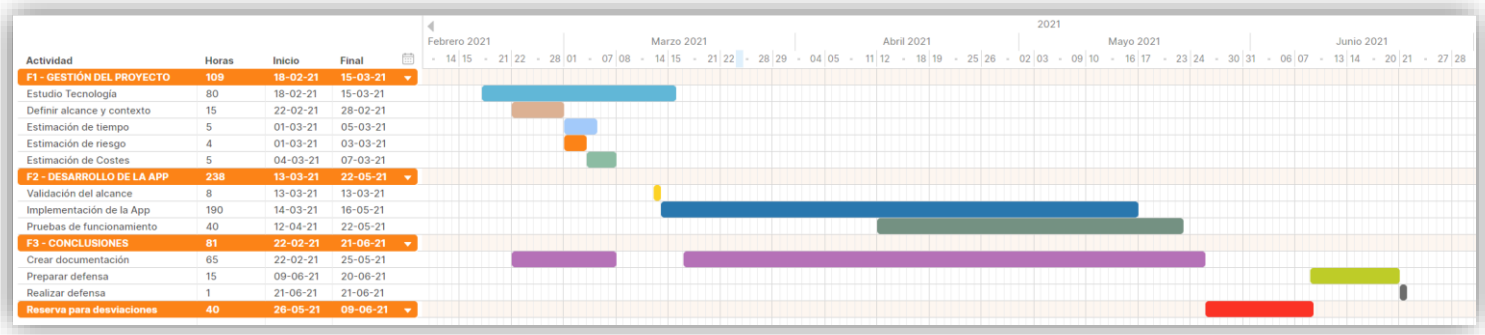


Figura 5. Diagrama de Gantt con la planificación.

Desviaciones producidas

Debido a problemas en el desarrollo de la APP, debidas al desconocimiento de la tecnología, **se ha tenido que consumir horas de la reserva para desviaciones**, en total 15 horas, por lo que la fase de implementación y Pruebas de funcionamiento han finalizado el 30/05/2020.

Finalmente la fecha de entrega defensa es el día 9, por lo que **el tiempo para preparar documentación y defensa en vídeo se ha modificado** y comprende del día 31/05 al 09/06, con un total de 32 horas disponibles, consumiendo las horas restantes de la reserva para desviaciones.

Observaciones

- La dedicación a este trabajo es de 3 horas en días laborables y 4 los días de fin de semana.
- Se establece la ronda de preguntas (defensa) como día orientativo el 21/06/21, aunque a fecha de entrega de este documento no está definida.
- Las tareas en paralelo se reparten las horas del día, estableciendo como prioritaria la tarea con más horas totales.

1.5 Metodología

Este trabajo final de máster es un proyecto de casi cinco meses de duración y con una gran carga de trabajo detrás, por lo que creo que se deben marcar las bases que sirvan como guía para poder finalizar con éxito el proyecto dentro de los márgenes de tiempo establecidos.

He valorado utilizar diversas metodologías que se usan en proyectos actualmente, pero finalmente me he decidido por una **metodología ágil**.

Esta metodología fue definida en [3] por un grupo de ingenieros de software, su origen es fruto de intentar buscar una alternativa a las metodologías tradicionales, que destacan por ser muy estructuradas y estrictas.

El problema que se encontró a este tipo de metodologías es que al no haber feedback[4] del cliente en las diversas fases del proyecto, podría no entregarse una solución óptima para sus intereses, que pueden variar durante el desarrollo.

En nuestro caso, al trabajar con una metodología ágil con desarrollo iterativo, nos permite ir realizando cambios e ir obteniendo opinión del cliente, en este caso el profesor consultor asignado al proyecto. Otro aspecto en el que beneficia al proyecto esta metodología se trata de poder modificar el alcance o los requisitos del proyecto en las fases iniciales, ya que debido a mi falta de experiencia con la tecnología pueden haberse definido de forma incorrecta.

1.5.1 Scrum

Una vez definida la metodología que se va a usar llega el momento de definir el marco de trabajo que va a implementarse para este proyecto. Para este trabajo, considero que el mejor marco de trabajo agile que se puede aplicar es Scrum, ya que es el que he usado en algún proyecto en el ámbito laboral y considero que sus características encajan perfectamente con el proyecto que se va a realizar.

El significado de scrum es “melé” y apareció definida por primera vez en [4]. Como ventaja principal se obtiene la generación de resultados de forma rápida y la fácil modificación de los elementos del proyecto.

El proyecto se divide en unas divisiones llamadas sprints[5], con duración de 1 a 4 semanas y en cada uno de esos sprints se obtiene como resultado un producto funcional. La lista de tareas se conoce como product backlog[6], está ordenado por prioridad y en cada sprint se decide qué tareas se van a realizar en función del esfuerzo de ellas, si una tarea no se ha podido completar en un sprint se revisará en la siguiente iteración.

[4] Reacción, respuesta u opinión que nos da un interlocutor como retorno sobre un asunto determinado.
[5] Intervalo prefijado durante el cual se crea un incremento de producto.
[6] Lista con todos los requerimientos iniciales del producto que se va a desarrollar

En nuestro caso no hemos tenido dailies, debido a que solo estoy yo miembro del equipo pero serían reuniones con todos los participantes del equipo para analizar qué tareas está desarrollando cada miembro y ver si aparecen dificultades.
Para este proyecto se han definido sprints con una duración de 4 semanas, por lo que se va a ir notificando al consultor de los cambios que se producen en el proyecto y recibir el feedback que considere oportuno.

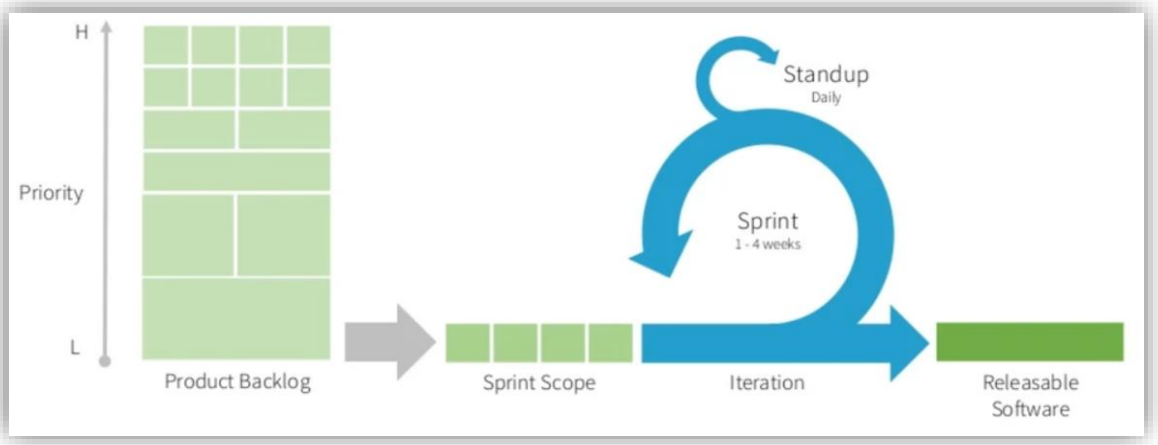


Figura 6. Esquema conceptual Scrum

1.6 Análisis de riesgos y medidas de contingencia

En este apartado vamos a analizar los posibles riesgos que pueden poner en peligro al proyecto, para cada uno se han definido planes de acción a desarrollar para reducirlo o eliminarlo, si es posible.

1.6.1 Framework

El framework nos puede suponer un problema, no tanto por las limitaciones del mismo sino por mi desconocimiento sobre su uso. Las partes que más problemas pueden generar son:

- La creación de la interfaz con los elementos que se han definido en los diseños iniciales: Si nos encontramos con este problema, tendremos que modificar la interfaz usando los elementos que podamos crear y reduciendo un poco la calidad visual de la aplicación.
- La gestión de rutas del usuario al hacer la integración con Google Maps: Si nos encontramos con algún problema en esta parte, tendremos que mostrar la ruta que debe hacer el usuario desde la aplicación externa de maps.

1.6.2 Tiempo

Puede ser que debido a cambios del alcance o retraso en alguna de las fases de desarrollo respecto a la planificación, aparezcan problemas de tiempo. Si se diese el caso lo podríamos solucionar utilizando las 40 horas que se han reservado para posibles desviaciones de

planificación. En caso de necesitar más de 40 horas tendremos que priorizar las funcionalidades críticas que nos quedan por implementar y modificar el alcance para eliminar las que no podremos llegar a tener acabadas en la fecha de entrega.

1.6.3 Presupuesto

Una de las premisas de este proyecto es **desarrollar con coste 0.** Por lo que si nos encontramos que alguna de las funcionalidades definidas se realiza con herramientas de pago, primero intentaremos encontrar herramientas gratuitas y, en caso de no ser posible encontrar alternativa gratis, deberemos eliminar esa funcionalidad y ampliar las que se puedan desarrollar con herramientas gratuitas.

1.7 Breve sumario de productos obtenidos

Los entregables finales serán:

1. Memoria final del proyecto.
2. Código fuente[7] de la aplicación, comprimido en archivo ZIP con las carpetas del proyecto para ser importadas en VS Code.
3. Ejecutable de la aplicación para Android, en formato .apk .
4. Presentación en vídeo y PowerPoint del trabajo con una breve demo.
5. Manual de usuario e información sobre la compilación y ejecución.

1.8 Sostenibilidad

1.8.1 Autoevaluación

Hoy en día la sostenibilidad aparece en todos los proyectos, pero se debe tener cuidado ya que tal y como se explica en [5], el alcance de la sostenibilidad debe enfocarse de forma semejante en las tres dimensiones que la comprenden (económica, ambiental y social).
Antes de entrar en dichas dimensiones, voy a hacer un análisis de las debilidades y fortalezas que he observado sobre este tema durante el transcurso del máster. El principal **punto fuerte** es que en las diferentes asignaturas se ha trabajado para que los alumnos entiendan la importancia que tiene la sostenibilidad en cualquier proyecto, aunque sea de software. Pese a ello, no me considero experto en el tema, ya que no hemos trabajado en valoraciones de sostenibilidad ni tampoco en herramientas para llevar un control de la misma. Como **punto débil**, también me gustaría añadir mi desconocimiento sobre los principios deontológicos[8] que tienen relación con la sostenibilidad.

[7] Código legible por un humano y redactado en un lenguaje de programación.
[8] Principios generales de las normas éticas de conducta del profesional.

De todas formas, soy plenamente consciente de las consecuencias en términos de sostenibilidad (sobre todo de la dimensión ambiental) que producen los productos tecnológicos, por lo que para este proyecto se ha intentado generar un producto sostenible.
Finalmente, pienso que pese a que mi conocimiento sobre las herramientas para trazar la sostenibilidad no es muy amplio, soy consciente de las consecuencias que puede producir una mala gestión de la misma, por lo que el proyecto ha servido para aprender en esos aspectos.

1.8.2 Dimensión Económica

En la dimensión económica se ha partido de una **premisa[9] muy clara: coste 0**. Aunque no ha sido del todo así ya que pese a no tenemos costes relacionados con el software, sí que tenemos costes de recursos humanos. Otros proyectos del mercado, las empresas destinan parte del presupuesto del proyecto a contratar versiones empresariales o de pago del software, por lo que en este caso, al haber usado únicamente software gratis, estamos mejorando respecto a las demás soluciones.

1.8.3 Dimensión Ambiental

La principal variable de nuestro proyecto que puede afectar negativamente a la sostenibilidad ambiental se trata del **consumo de electricidad de los recursos hardware** destinados a la creación de la aplicación.
Solo he decidido usar un único ordenador, en lugar de realizar el trabajo desde el ordenador de sobremesa y el portátil del trabajo, ya que esta acción ayudará a reducir el impacto ambiental. Por otra parte, considero **un acierto el no tener que imprimir la memoria** para ser entregada, ya que de esta forma se evita imprimir el gran número de hojas de este documento.

1.8.4 Dimensión Social

En cuanto a esta dimensión de la sostenibilidad creo que **sí que ha existido una necesidad real de hacer el proyecto**, ya que me ha servido para aprender sobre la tecnología y también para mejorar un problema actual en nuestras ciudades con un alto compromiso social.
También, se pretende que este proyecto **sirva como base** para que otras personas puedan adoptar esta solución a su caso concreto, ahorrando esfuerzo y tiempo en su implementación.

[9] Idea que sirve de base

2. Análisis, Diseño y Arquitectura

2.1 Experiencia de Usuario

2.1.1 Perfil de usuario

En esta sección vamos a indicar las características del perfil de usuario, incluyendo las características demográficas, los intereses, las motivaciones, la experiencia con el uso de la tecnología móvil, etc. De esta forma podremos identificar el tipo de consumidor más atractivo para nuestra aplicación agrupándolo en un nicho y definiendo las características de su perfil, para así reducir los segmentos a los que sean similares en cuanto a comportamiento o motivaciones.

- **Edad comprendida entre 15 y 60 años**, ya que considero que las funcionales de la aplicación no serán sencillas.
- **Ambos sexos** usarán nuestra aplicación de igual forma.
- Las necesidades de nuestros usuarios será encontrar una **alternativa a los obstáculos** que se encuentran las personas con problemas movilidad al desplazarse a pie.
- Nuestros usuarios **tendrán algún problema de movilidad o serán** acompañantes de alguien con problemas de movilidad.
- Los usuarios necesitarán **una cierta experiencia en el uso de la tecnología móvil** ya que algunas de las funcionalidades que integra nuestra aplicación, como por ejemplo hacer uso de la cámara para capturar los problemas, son complejas.

2.1.2 Contextos de uso

En esta sección vamos a definir dónde, cuándo y en qué contextos los usuarios harán uso de la aplicación. Esto nos ayudará a mejorar la eficiencia y eficacia de la interacción con nuestra aplicación.

- Usuarios con problemas de movilidad o los acompañantes de estos, tratando de encontrar una ruta en la que pueda **llegar a un destino observando los obstáculos** que le impidan seguir. Normalmente harán uso de la aplicación desde la ubicación que marcan como origen, por lo tanto para esta primera versión solo es posible hacerlo de esa forma.

- Usuarios que al conocer la zona, se encuentren que uno de los puntos con dificultad de paso se ha solucionado, por lo que desea **corregirlo en la aplicación para que se abra al paso de los demás usuarios**. En este caso no harán uso de la aplicación mientras están en ruta, ya que no pasarán por esos puntos.

- Usuarios que desea saber qué barreras de movilidad hay en una zona concreta, **buscando en el mapa alguna dirección postal** de la zona que quiere observar. Estos usuarios puede que se encuentren en la zona que van a buscar (no haría falta introducir la dirección) o también puede ser que estén buscando otras ubicaciones.

2.1.3 Fichas de usuario y escenarios

2.1.3.1 Mujer en silla de ruedas que va al médico

Atributo	Detalles
Nombre	Laura
Edad	36
Profesión	Administrativa
Familia	Soltera y sin hijos
Ciudad	Barcelona

Biografía: Laura vive en Barcelona, es soltera y no tiene hijos. Trabaja de administrativa para una empresa tecnológica en Barcelona, por lo que en su trabajo está utilizando siempre el ordenador y el correo electrónico. Al trabajo siempre acude en moto, pero en estos momentos se encuentra de baja ya que hace unas semanas paseando a su perro, se le cruzó la correa mientras ella bajaba unas escaleras y al caerse se rompió la cadera. Laura tiene que estar un mes en silla de ruedas y todavía no acaba de acostumbrarse.

Escenario de uso: Ha llegado el día en el que Laura debe acudir a su revisión médica, esta vez le toca ir a ella sola, ya que la vez anterior fue en ambulancia. Se acuerda de que su amiga le comentó que ahora que va en silla de ruedas necesitaría usar la aplicación de EasyMov, ya que de lo contrario tardaría el doble en llegar a los sitios.
Laura decide hacerle caso y se instala la aplicación, de repente se encuentra un mapa con multitud de puntos negros con un icono tachado de silla de ruedas, se da cuenta que eso significa problemas, así que decide buscar en la barra de búsqueda la dirección de la consulta del médico.
Tras un segundo, le aparece una ruta dibujada que atraviesa algunos de esos puntos, así que decide pulsar encima de ellos para ver más detalles y se memoriza las alternativas que describe la gente, a cada imagen de obstáculo que ve, descubre que realmente hubiese tardado el doble en llegar a su destino. Finalmente, sale de casa con un poco de margen y consigue evitar todos los obstáculos llegando a su revisión unos minutos antes de la hora de la cita.

2.1.3.2 Hombre que acompaña a su mujer a comprar

Atributo	Detalles
Nombre	Carlos
Edad	55
Profesión	Policía Local
Familia	Casado con un hijo
Ciudad	Tarragona

Biografía: Carlos vive en Tarragona, está casado con Marta desde hace 25 años, tienen un hijo de 20 años llamado Pablo, que está estudiando en Berlin la carrera de Negocio Internacional. Carlos trabaja como policía local en Altafulla, un pueblo con mucho encanto situado a 10 minutos de la ciudad. Por suerte, su trabajo le permite tener mucho tiempo libre así que destina gran parte de él a sus dos aficiones: diseñar maquetas de parques de atracciones con movimiento y la fotografía, que combinadas dan un resultado sorprendente. Pero no todo el tiempo lo dedica a ello, sino que siempre que puede sale a dar un paseo junto con su mujer, que hace 10 años, sufrió un grave accidente de coche que la dejó postrada en una silla de ruedas. Marta se ha podido adaptar muy bien a su situación y es autosuficiente.

Escenario de uso: Es sábado y como cada sábado los payeses de la región venden sus productos en un mercado que está situado a 15 minutos de casa de Carlos. Carlos, que es un fiel defensor del comercio de proximidad, no perdona ni un sábado para ir a hacer la compra de la semana, así que él y su mujer se levantan bien pronto para ir a comprar.
Marta siempre utiliza la aplicación EasyMov cuando tiene que desplazarse por la ciudad, ya que no la conoce tanto como Altafulla y sabe que a veces hay bastantes barreras arquitectónicas en el camino, por lo que Carlos, también la usa cuando va con ella para añadir obstáculos que se encuentran o corregir cuando se hacen modificaciones en las rutas que ellos conocen. Así que mientras van camino al mercado y observando los diferentes obstáculos que hay registrados en la aplicación se dan cuenta que la calle que llevaba un mes en obras por fin vuelve a estar accesible, por lo que el obstáculo que reportó Carlos del tipo “Obras”, ya no tiene sentido, así que lo resuelve e inmediatamente desaparece de la aplicación mostrando un mensaje de agradecimiento. Pero cuando están circulando por esa calle, se dan cuenta que el paso de peatones no tiene rampa, así que Carlos, selecciona el punto dónde se encuentra ese paso de peatones, toma una imagen y crea el obstáculo, que genera un marcador de color negro en el mapa. Los demás usuarios ahora podrán saber qué obstáculo es y cómo evitarlo, que Carlos ha indicado usando el siguiente paso de peatones.

2.1.4 Análisis de tareas

Tarea	Detalles
Registrarse	Para ello, al entrar los usuarios deberán darle al texto de “Crear cuenta nueva” de la pantalla de inicio de sesión. Deberán introducir el nombre, correo electrónico y contraseña.
Iniciar sesión	Los usuarios deberán introducir sus datos de registro que son correo electrónico y contraseña.
Editar perfil	Tras iniciar sesión, los usuarios podrán acceder a su perfil en el menú desplegable de la parte superior izquierda.
Buscar ubicación	Una vez iniciada la sesión, en el mapa el usuario podrá buscar una ubicación o usar su geolocalización del dispositivo para definirla.
Añadir obstáculo	Tras haber hecho login y buscado una ubicación, podrá añadir un nuevo obstáculo en el mapa rellenando su información. También lo podrá hacer sin tener que buscar una localización en concreto.
Crear ruta	Para ello, el usuario tendrá que buscar una ubicación o seleccionarla manualmente. Se creará la ruta con el origen actual.
Mostrar movimiento	Mostrar la ruta que ha ido siguiendo el usuario mientras tenía la app abierta, en futuras versiones se podrán guardar rutas frecuentes.
Resolver obstáculo	Cuando el usuario quiera resolver un obstáculo, tendrá que ir a la lista de obstáculos del menú lateral o tendrá que pulsar sobre un obstáculo. Una vez allí, deberá pulsar sobre el botón de “Resolver”. Si el obstáculo es de otro usuario se necesitarán 4 resoluciones, si el obstáculo es propio se eliminará.
Cerrar sesión	Para cerrar sesión, el usuario tendrá que tener sesión iniciada y darle al botón del menú lateral de “Cerrar Sesión”

Tabla 3. Análisis de tareas.

2.1.5 Listado de elementos de la interfaz

Elemento	Detalles
Pantalla Login / Registro	Pantalla principal al NO estar con sesión iniciada, solicitará al usuario que inicie sesión o cree una cuenta.
Mapa	Pantalla principal de la aplicación al estar con sesión iniciada, con el mapa de fondo.
Menú lateral	Menú lateral con icono de hamburguesa. Con la foto de perfil y las diversas funciones del menú.
Perfil	Pantalla accesible desde el menú lateral, con información del usuario.
Ajustes	Pantalla que permite configurar el tipo de mapa y el modo de color.
Sección de mis obstáculos	Pantalla accesible desde el menú lateral, con una lista de obstáculos creados por el usuario y la posibilidad de resolverlos.
Búsqueda de obstáculos	Barra accesible desde la parte superior del mapa, permite buscar una ubicación.
Modal de obstáculo	Una vez seleccionado un obstáculo, se mostrará esta modal que incluirá los detalles y la opción de resolver.

Tabla 4. Elementos de la interfaz.

2.2 Árbol de navegación

Después de analizar los requisitos funcionales de nuestra aplicación y los diferentes contextos de uso, es momento de definir la navegación de nuestra aplicación, **una buena navegación ayudará a nuestros usuarios a lograr sus objetivos de uso**, por lo que este punto es un buen punto de partida para el diseño de nuestra aplicación.

A continuación se muestra la figura con la navegación personalizada para este caso, en secciones posteriores del documento se verá cada pantalla con más detalle.

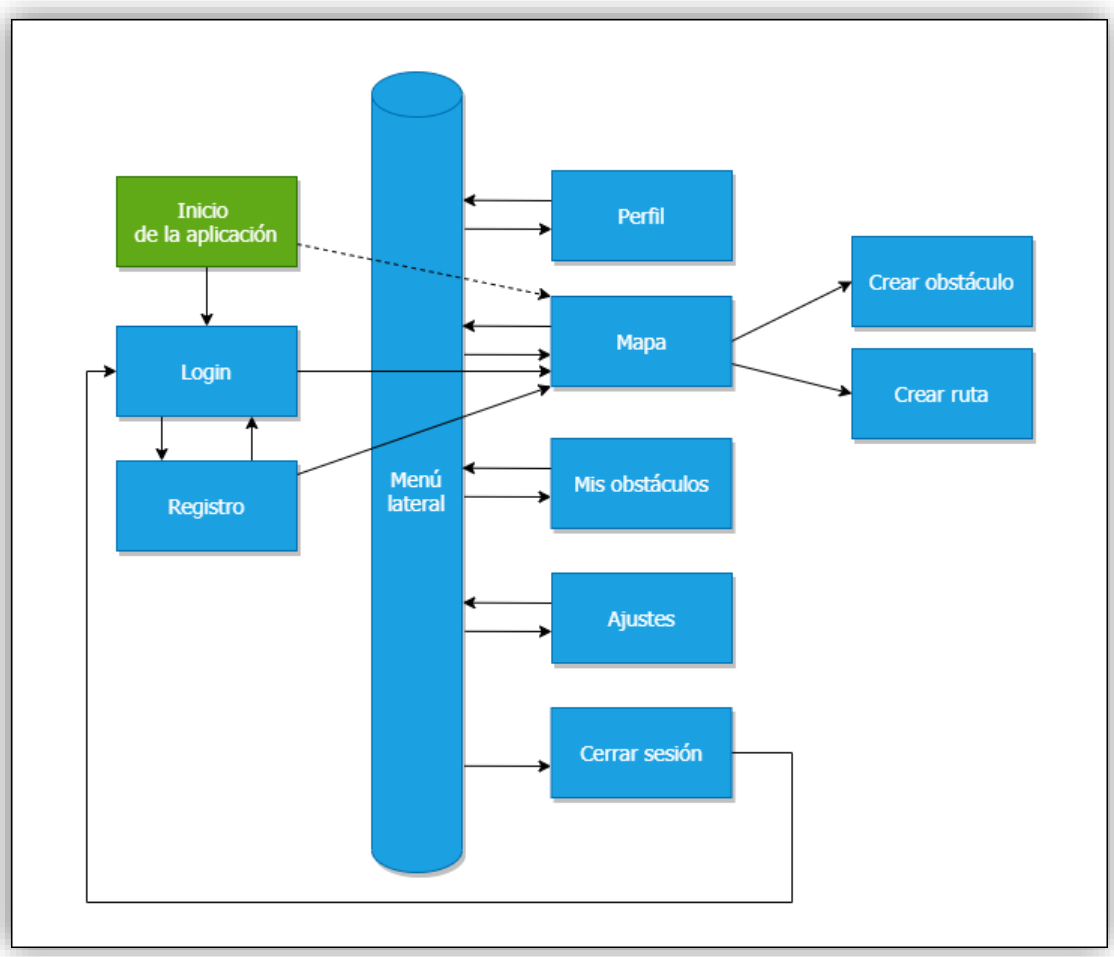


Figura 7. Árbol de navegación de EasyMov

Observaciones

- El comportamiento de inicio de la aplicación es acceder al login, pero en caso de ya tener sesión iniciada se accederá al mapa.
- Tras cerrar sesión se redireccionará al login y se borrará cualquier información de la sesión.
- Desde el menú lateral no se puede acceder al login ni al registro, salvo que se pase por el cierre de sesión.

2.3 Prototipo

2.3.1 Sketches

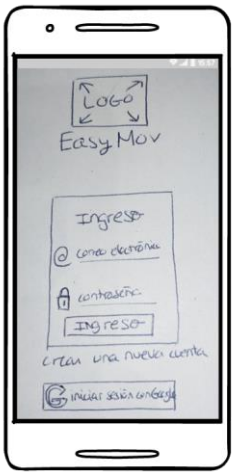


Ilustración 1. Login Sketch

Ilustración 2. Pág. Principal Sketch

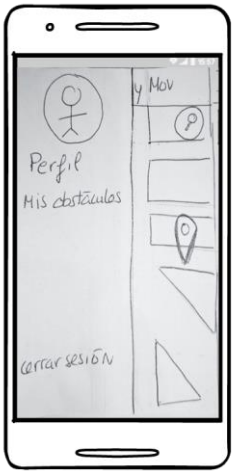


Ilustración 3. Menú lateral Sketch

Ilustración 4. Perfil Sketch

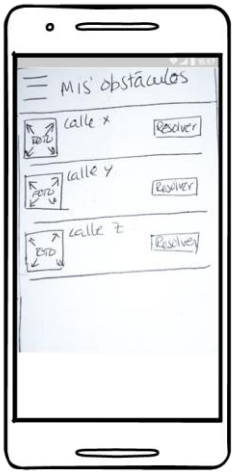


Ilustración 5. Mis obstáculos Sketch

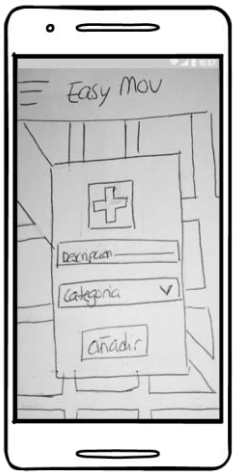


Ilustración 6. Crear obstáculo Sketch

Ilustración 7. Buscar en mapa Sketch

Ilustración 8. Abrir obstáculo Sketch

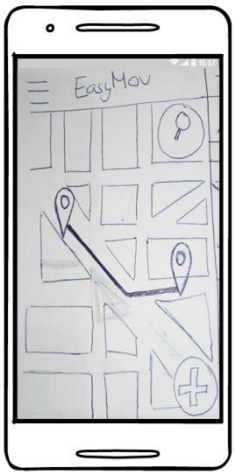


Ilustración 9. Crear ruta Sketch

2.3.2 Prototipo horizontal de alta fidelidad

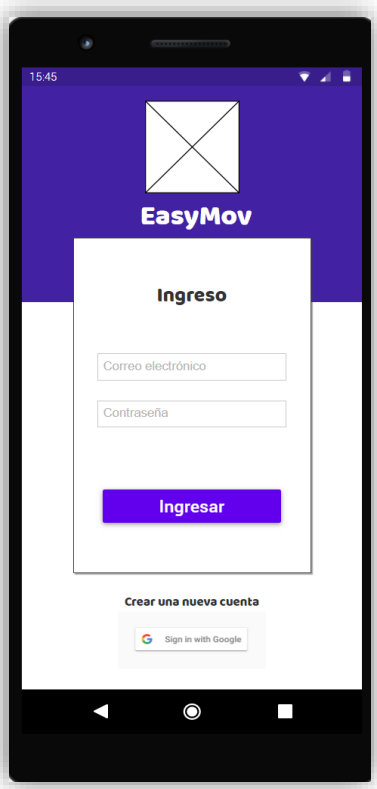


Ilustración 10. Login prot.

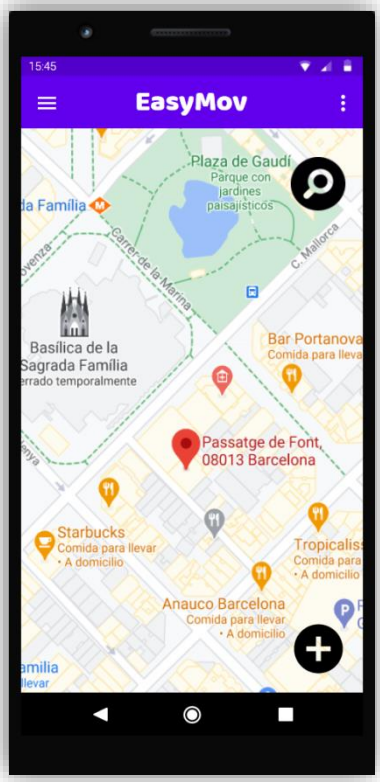


Ilustración 11. Pág. Principal prot.

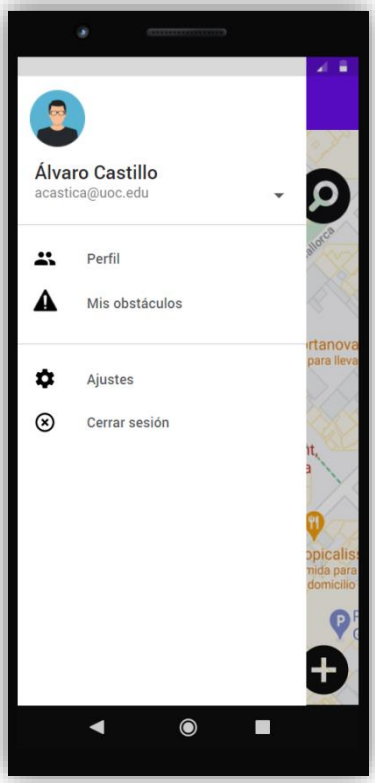


Ilustración 12. Menú lateral prot.

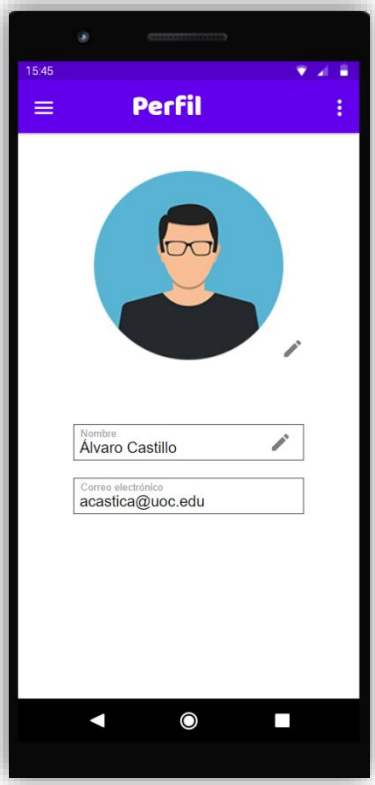


Ilustración 13. Perfil prot.

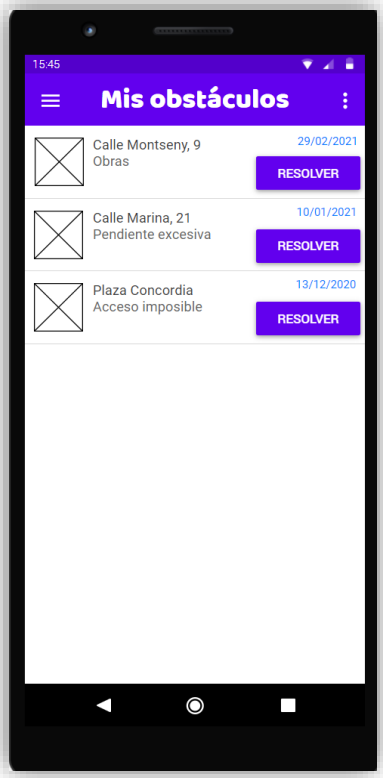


Ilustración 14. Mis obstáculos prot.

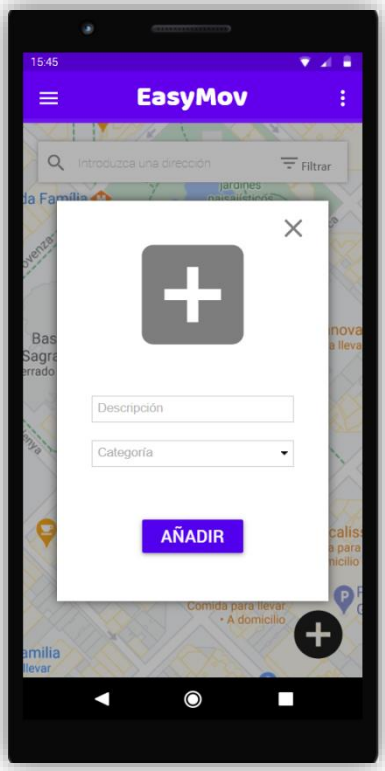


Ilustración 15. Crear obstáculo prot.

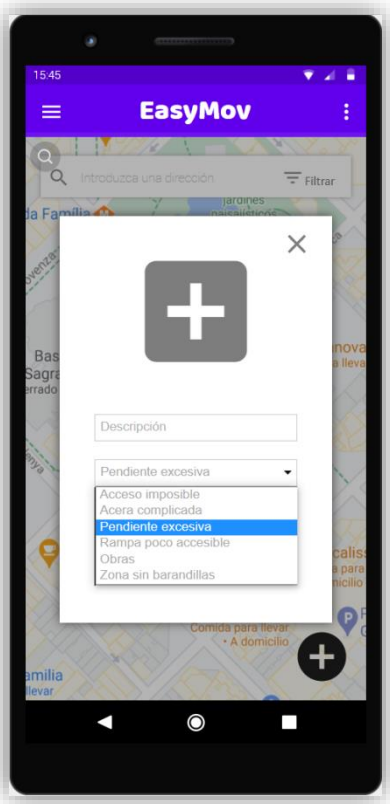


Ilustración 16. Crear obstáculo (2) prot.

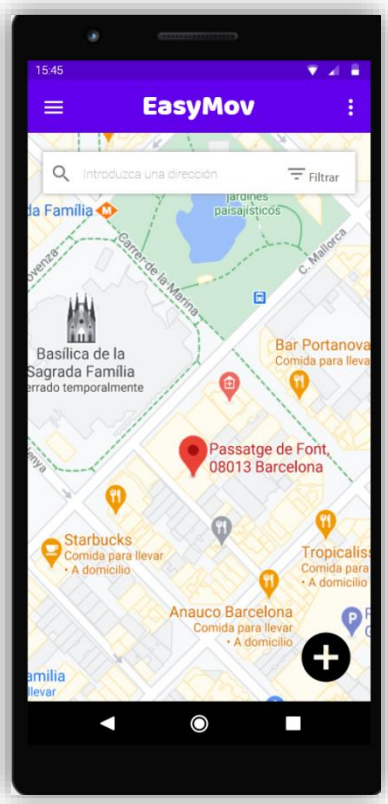


Ilustración 17. Buscar en mapa prot.

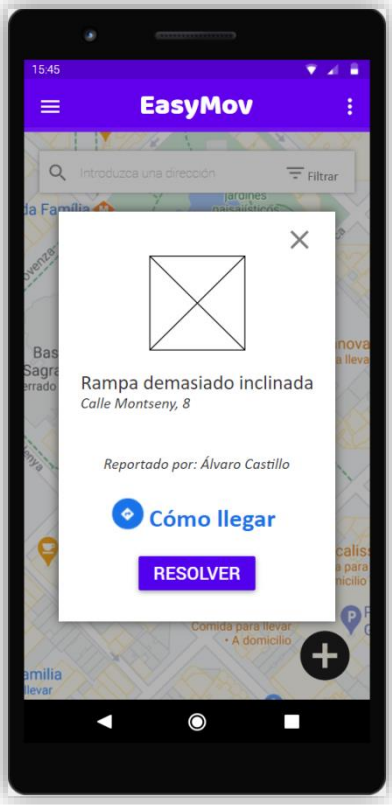


Ilustración 18. Abrir obstáculo prot.

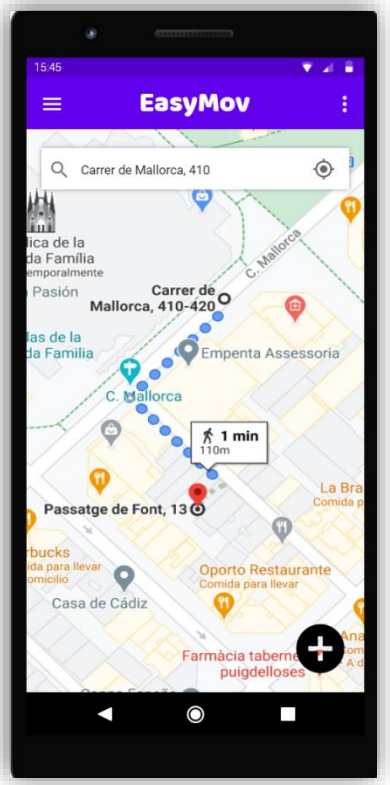


Ilustración 19. Crear ruta prot.

2.4 Casos de uso

Es fundamental definir diagramas que permitan **representar que hará el sistema**, sin especificar cómo funciona. Estos diagramas sirven para poder definir con el cliente la funcionalidad de la aplicación sin que esté acabada, para evitar errores en la definición de los requerimientos y provocar que se plantee algo diferente a lo que espera el cliente, por lo que en los siguientes puntos presentaremos al cliente una perspectiva de cómo será la aplicación una vez terminada.

2.4.1 Diagrama UML de actores y acciones

Los diagramas de esta sección solo tienen un actor, que es el usuario que utiliza la App, además hemos dividido los diagramas en dos paquetes: Las acciones con la gestión del usuario de la aplicación y las acciones relacionadas con la gestión de obstáculos y rutas.

Hay un administrador que únicamente tiene como función modificar o eliminar registros de la base de datos de usuarios y obstáculos, en caso de que fuera necesario. He creído correcto ocultar la actividad del administrador ya que se usará de forma muy excepcional.

2.4.1.1 Listado casos de uso.

------GESTIÓN DE USUARIO------
CU-001: Registrarse.
CU-002: Iniciar sesión.
CU-003: Cerrar sesión.
CU-004: Editar perfil de usuario.
CU-005: Ver mis obstáculos.
CU-006: Modificar ajustes.

------GESTIÓN DE OBSTACULOS Y RUTAS------
CU-007: Marcar ubicación como obstáculo.
CU-008: Añadir imagen del obstáculo.
CU-009: Añadir descripción del obstáculo.
CU-010: Añadir alternativa para evitarlo.
CU-011: Añadir categoría del obstáculo.
CU-012: Resolver obstáculo.
CU-013: Elegir ubicación destino (Se extiende con buscar por dirección postal y buscar desplazándose por el mapa).
CU-014: Crear ruta.

2.4.1.2 Diagrama de gestión de usuario.

El siguiente diagrama, muestra las acciones que puede realizar el usuario dentro de la aplicación.

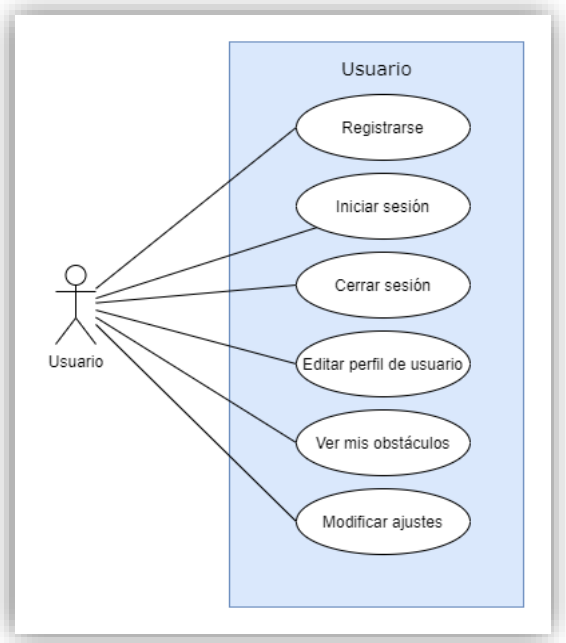


Figura 8. Diagrama UML acciones gestión de usuario.

2.4.1.3 Diagrama de gestión de obstáculos y rutas.

A continuación se muestran las acciones que puede realizar el usuario y hacen referencia con la gestión de obstáculos (creación / eliminación) y la gestión de rutas.

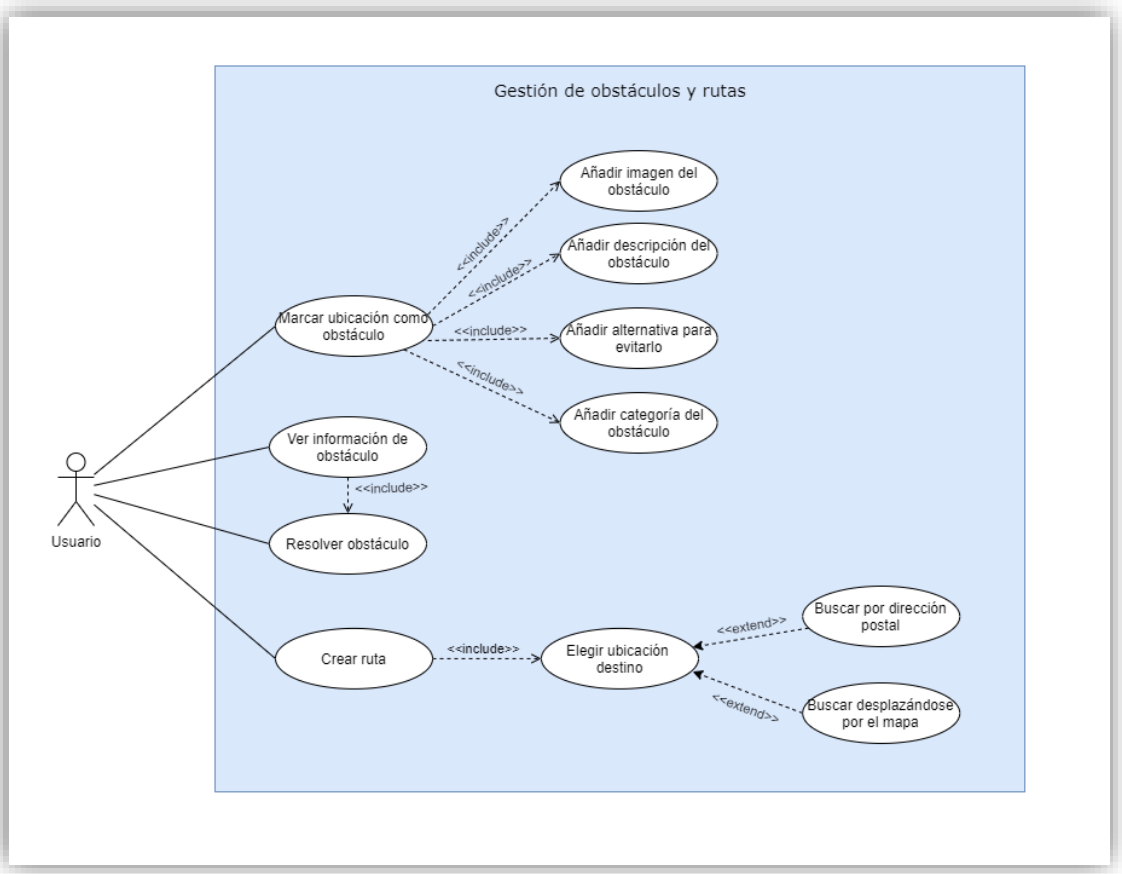


Figura 9. Diagrama UML acciones gestión de obstáculos y rutas.

2.4.2 Listado de casos de uso

Identificador	CU-001
Nombre	Registrarse
Prioridad	Normal
Descripción	Registro mediante correo/contraseña o Google
Actores	Usuario
Pre-Condiciones	No existir un usuario/contraseña con ese correo
Iniciado por	Usuario
Post-Condiciones	Se registrará un nuevo usuario en la BBDD
Notas	-

Identificador	CU-002
Nombre	Iniciar sesión
Prioridad	Normal
Descripción	Los usuarios con cuenta podrán iniciar sesión en la app
Actores	Usuario
Pre-Condiciones	*CU-001 – Registrarse* y acceder con esos datos
Iniciado por	Usuario
Post-Condiciones	El usuario inicia la sesión en la app
Notas	Se mostrará mensaje de error en caso de no ser correctas las credenciales de inicio de sesión

Identificador	CU-003
Nombre	Cerrar sesión
Prioridad	Normal
Descripción	Permite al usuario cerrar la sesión activa
Actores	Usuario
Pre-Condiciones	*CU-002 – Iniciar sesión*
Iniciado por	Usuario
Post-Condiciones	La sesión activa del usuario en la aplicación será cerrada
Notas	-

Identificador	CU-004
Nombre	Editar perfil de usuario
Prioridad	Bajo
Descripción	Permite al usuario modificar sus datos de perfil
Actores	Usuario
Pre-Condiciones	*CU-002 – Iniciar sesión*
Iniciado por	Usuario
Post-Condiciones	Se actualizarán los datos del usuario en su perfil
Notas	-

Identificador	CU-005
Nombre	Ver mis obstáculos
Prioridad	Alta
Descripción	Permite al usuario ver sus obstáculos reportados.
Actores	Usuario
Pre-Condiciones	*CU-002 – Iniciar sesión* y *[OPT] CU-007 – Marcar ubicación como obstáculo*
Iniciado por	Usuario
Post-Condiciones	El usuario verá un listado de sus obstáculos y se le permitirá resolverlos
Notas	Si el usuario no tiene ninguno, se le mostrará vacío con un mensaje

Identificador	CU-006
Nombre	Modificar ajustes
Prioridad	Baja
Descripción	Permite al usuario modificar ajustes de la aplicación.
Actores	Usuario
Pre-Condiciones	*CU-002 – Iniciar sesión*
Iniciado por	Usuario
Post-Condiciones	Se modificará la característica seleccionada para futuras ejecuciones.
Notas	Algunas características requieren reiniciar la aplicación

Identificador	CU-007
Nombre	Marcar ubicación como obstáculo
Prioridad	Alta
Descripción	Permite indicar que en la ubicación hay un obstáculo
Actores	Usuario
Pre-Condiciones	*CU-006 – Ver ubicación en el mapa*
Iniciado por	Usuario
Post-Condiciones	Se añadirá el obstáculo a la BBDD tras añadir la información
Notas	-

Identificador	CU-008
Nombre	Añadir imagen del obstáculo
Prioridad	Normal
Descripción	Permite al usuario subir una imagen del obstáculo
Actores	Usuario
Pre-Condiciones	*CU-008 – Marcar ubicación como obstáculo*
Iniciado por	Usuario
Post-Condiciones	Se guardará en la información del obstáculo
Notas	La imagen se puede seleccionar de la galería o hacerla

Identificador	CU-009
Nombre	Añadir descripción del obstáculo
Prioridad	Normal
Descripción	Permite al usuario añadir una descripción del obstáculo
Actores	Usuario
Pre-Condiciones	*CU-008 – Marcar ubicación como obstáculo*
Iniciado por	Usuario
Post-Condiciones	Se guardará en la información del obstáculo
Notas	Se limitará a 60 caracteres

Identificador	CU-010
Nombre	Añadir alternativa del obstáculo
Prioridad	Normal
Descripción	Permite al usuario añadir una alternativa para evitar el obstáculo
Actores	Usuario
Pre-Condiciones	*CU-008 – Marcar ubicación como obstáculo*
Iniciado por	Usuario
Post-Condiciones	Se guardará en la información del obstáculo
Notas	Se limitará a 100 caracteres y mínimo 10

Identificador	CU-011
Nombre	Añadir categoría del obstáculo
Prioridad	Baja
Descripción	Permite al usuario indicar la categoría del obstáculo
Actores	Usuario
Pre-Condiciones	CU-008 – Marcar ubicación como obstáculo
Iniciado por	Usuario
Post-Condiciones	Se guardará en la información del obstáculo
Notas	La categoría se seleccionará de un desplegable

Identificador	CU-012
Nombre	Resolver obstáculo
Prioridad	Alta
Descripción	Permite a los usuarios marcar si un obstáculo se ha resuelto
Actores	Usuario
Pre-Condiciones	*CU-006 – Ver ubicación en el mapa* o *CU-005 – Ver mis obstáculos*
Iniciado por	Usuario
Post-Condiciones	Al llegar al número mínimo de resoluciones, se eliminará
Notas	Se necesitarán 4 resoluciones de usuarios para que sea efectivo, si el obstáculo es propio solo 1

Identificador	CU-013
Nombre	Elegir ubicación de destino
Prioridad	Alta
Descripción	Permite al usuario indicar el destino de la ruta
Actores	Usuario
Pre-Condiciones	*CU-002 – Iniciar sesión*
Iniciado por	Usuario
Post-Condiciones	Se marcará el punto seleccionado como destino
Notas	-

Identificador	CU-014
Nombre	Crear ruta
Prioridad	Alta
Descripción	Permite al usuario crear una ruta con ese destino
Actores	Usuario
Pre-Condiciones	*CU-013 – Elegir ubicación de destino*
Iniciado por	Usuario
Post-Condiciones	Se mostrará la ruta en el mapa con ese destino
Notas	Se puede eliminar esa ruta pulsando en el botón lateral

2.5 Framework elegido

Después de analizar las diferentes opciones que hay en el mercado, me ha parecido interesante desarrollar una aplicación con el framework de Flutter para poder descubrirlo en profundidad y poder usarlo en el futuro en otros proyectos personales.

2.5.1 Flutter

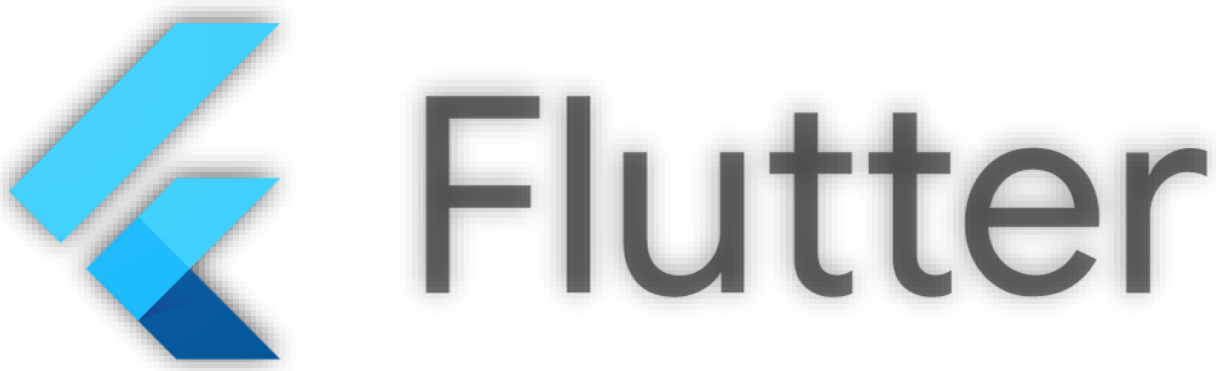


Figura 10. Logo de Flutter

Flutter es un framework elaborado por Google y lanzado por primera vez en diciembre de 2018 en [6] que permite **desarrollar aplicaciones para plataformas Android, iOS y web**. Fue desarrollado como un software para uso interno dentro de la compañía pero vieron el potencial que tenia y decidieron lanzarlo como proyecto de código libre. La ventaja de este framework es que no es necesario escribir un código base propio de cada sistema, ya que antes de su publicación se compilan para la plataforma correspondiente, dando resultado a auténticas aplicaciones nativas.
Su lenguaje de programación es DART, que explicaremos con detalle en el siguiente apartado.
Los principales IDEs para desarrollar aplicaciones en Flutter son VS Code (el elegido para este proyecto), Android Studio, IntelliJ IDEA y Xcode. Todos ellos con sus plugins oficiales tanto de Flutter como de Dart para facilitar el desarrollo y evitar errores.
Con Flutter se puede desarrollar de forma realmente cómoda, ya que posee una ventaja que otros frameworks no tienen, que es el llamado “Hot-Reload”, que permite ir visualizando en tiempo real el estado de la aplicación y los cambios sin parar la ejecución.
Internamente el framework utiliza el motor gráfico Skia, que se encarga de renderizar en 2D los elementos gráficos de la aplicación.

Para programar en Flutter solo es necesario un ordenador con uno de estos sistemas operativos: Windows, Linux o Mac. Simplemente hay que descargar Flutter desde su página web y seguir la instalación que indican. Se debe instalar el SDK de Android, que normalmente se hace mediante Android Studio. Para compilar la app en iOS solo se puede hacer con Xcode. Además, flutter permite importar librerías de funcionalidades desarrolladas por otra gente, por lo que las posibilidades crecen exponencialmente.

2.5.1.1 Widgets

Flutter utiliza las bases de la programación orientada a objectos y utiliza **bloques de código reusables llamados widgets**, inspirados en React.
Cada elemento de la interfaz de Flutter es un widget, que a su vez pueden estar contenidos dentro de widgets más complejos. Por ejemplo, cada botón, columna, imagen… se construye a partir de un widget y posee diversas propiedades que se pueden modificar, las propiedades varían según el tipo de widget. Otro aspecto interesante de los widgets es que pueden interactuar entre ellos y reaccionar a cambios de estado externos.
Según el dispositivo en el que vayan a ser usados existen dos paquetes de interfaces de usuario: Material para Android, basado en el diseño de Google y Cupertino basado en el diseño de Apple. En las últimas versiones y para algunos widgets es posible definir ambos paquetes y que se muestre un estilo u otro en función del dispositivo.

Existen dos tipos de widgets, los Stateless Widgets (sin estado) y los Stateful Widgets (con estado). En el primer de los casos, se trata de widgets que no guardan valores que puedan cambiar, por ejemplo una imagen. En el segundo tipo, los widgets con estado sí que tienen seguimiento de los cambios que se producen en ellos, por lo que pueden modificar la interfaz en función de dichos cambios, por ejemplo un botón.

También existen widgets que implementan mecanismos de layout[10] para poder generar composiciones de widgets más complejas, estos widgets contienen un vector llamado children con los widgets internos. Los principales layout son:

- **Row**: Este widgets permite añadir widgets en paralelo en el sentido horizontal de la pantalla (eje x). El Main Axis para alinear, correspondería al eje x y el Cross axis al eje-y.

Figura 11. Row layout esquema (fuente: javatpoint.com)

- **Column**: Este widgets permite añadir widgets en paralelo en el sentido vertical de la pantalla (eje x). El Main Axis para alinear, correspondería al eje y y el Cross axis al eje-x.

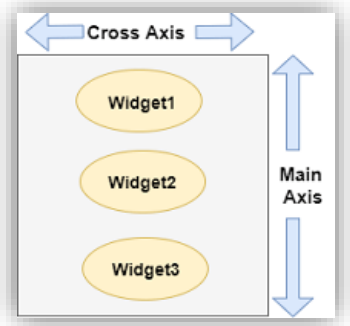


Figura 12. Column layout esquema (fuente: javatpoint.com)

[10] Disposición de los elementos en la pantalla.

- **Stack**: En este caso los widgets se colocan uno encima del otro, teniendo como referencia las distancias con los bordes.

Los widgets se van anidando unos dentro de otros, usando los layout anteriores y muchos más, dando como resultado el llamado **árbol de widgets**, que contiene la estructura padre-hijo de nuestra pantalla.

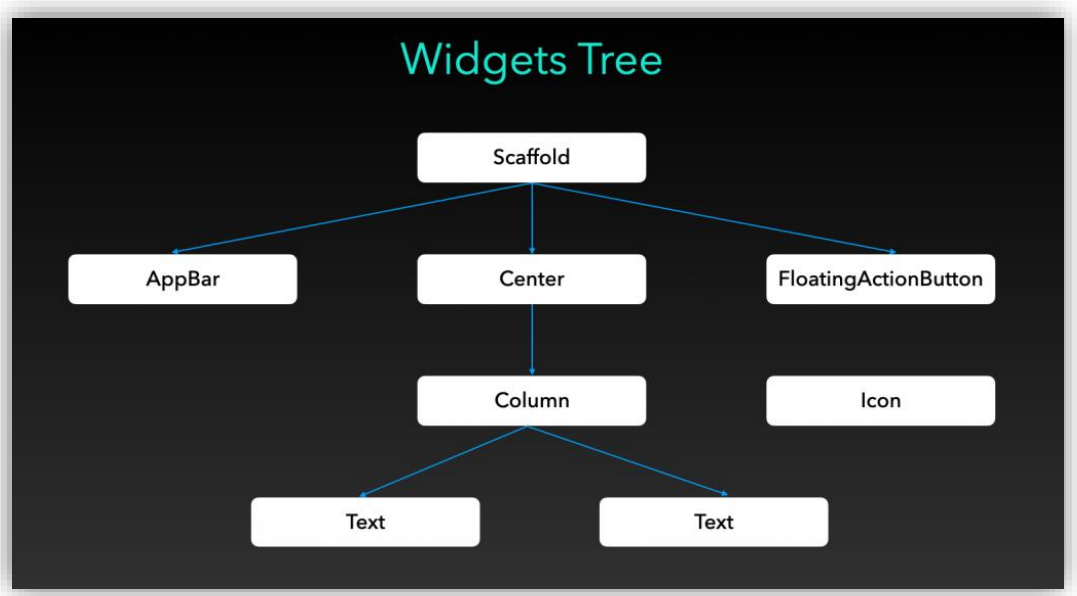


Figura 13. Ejemplo árbol de widgets (fuente www.raulferrergarcia.com)

Para crear cualquier widget, se utiliza el método build, que únicamente acepta un argumento de tipo BuildContext, normalmente llamado context.
Este **contexto es una descripción de la posición del widget en el árbol de widgets** y también aporta mucha más informacion sobre el elemento que se está renderizando en esa clase, como el estado en el que se encuentra, el tipo de widget que es, etc.

Es esta clase Buildcontext la que guarda información la relación padre-hijos y permite implementar funciones muy útiles que permiten utilizar el contexto del widget padre como showDialog, Theme.of y algunas otras como su tamaño y ubicación. Además de todo esto, el contexto se utiliza para evitar la manipulación directa del widget principal del árbol y para obtener la información necesaria que está contenida en el contexto principal. Por lo que es este el motivo que hace que en Flutter los widgets puedan comunicarse con otros widgets ubicados tanto debajo como encima del árbol de widgets

2.5.1.2 Ventajas y desventajas

Después de explicar un poco Flutter y sus Widgets, vamos a analizar las ventajas e inconvenientes, porque aunque es verdad que este framework en general tiene ventajas muy positivas, también hay algunas desventajas, que realmente se quedan pequeñas a su lado.

Ventajas de Flutter:

- **Un único código fuente para diversas plataformas de salida**: Este punto es sumamente positivo, ya que como hemos comentado anteriormente, solo hace falta desarrollar un único código para poder exportarlo en las plataformas de Android, iOS y Web.
- **Características nativas en las aplicaciones generadas**: Flutter permite mostrar características nativas, como por ejemplo los paquetes de UI[11] propios de cada plataforma (Material y Cupertino), el tipo de scroll, fuentes, etc.
- **Widgets predefinidos con configuración sencilla**: Resulta muy sencillo configurar los widgets ya que vienen prefabricados y con valores por defecto que se pueden modificar en caso de ser necesario.
- **Hot Reload para ver cambios**: Esta característica permite que los cambios que se produzcan en el IDE, se actualicen al momento en el emulador o dispositivo dónde se esté probando la aplicación. Esta característica acelera las pruebas y es realmente rápida.
- **Mucha documentación**: Hay una gran cantidad de información disponible tanto desde la propia web del framework como generada por la comunidad, por lo que cualquier duda o problema seguramente tendrá respuesta. También existen multitud de ejemplos y tutoriales de los propios desarrolladores de Flutter.

Desventajas de Flutter:

- **Código confuso con muchos widgets:** Esto es debido a que a medida que van creciendo los widgets anidadados dentro de otros, el código también lo hace y eso hace que el código sea un poco confuso y si no se organiza muy bien sea complicado de leer.
- **Los módulos están integrados de forma fija:** Eso produce que en caso de una actualización en el sistema operativo del dispositivo, sea necesario volver a generar las aplicaciones e instalarlas nuevamente.
- **Exige aprender Dart:** Al tener que programar únicamente con el lenguaje Dart, es necesario aprenderlo para poder usar Flutter, ya que no se podrá usar otro lenguaje de programación.

[11] Interfaz de Usuario (User Interface)

2.5.2 Dart

Figura 14. Logo de Dart.

Dart es un lenguaje de programación desarrollado por Google, empezó a desarrollarse en 2010 y fue presentado en 2013 en la conferencia [7]. Este lenguaje se podría catalogar como lenguaje orientado a objetos (C#, Java, etc.), aunque la sintaxis es parecida a C.
Este lenguaje surgió como alternativa a JavaScript, ya que según los desarolladores de Dart, ya no es posible arreglar las deficiencias de JavaScript desarrollándolo todavía más. Mediante el compilador Dart2js se puede ejecutar Dart en todos los navegadores actuales, ya que de forma nativa no es posible hacerlo. Además, Dart utiliza una filosofía parecida a Java en cuanto a que tiene una máquina virtual propia llamada DartVM, por lo que al usarla para ejecutarse lo convierte en multiplataforma y hace que pueda ser usado tanto en Mac, como en Windows o Linux. Es questión de tiempo que esta máquina virtual sea incorporada también en los navegadores para poderlo ejecutar de forma nativa.

El lenguaje Dart dispone de todos los tipos de datos y herramientas de los demás lenguajes: variables, operadores, condiciones, bucles, funciones, clases, objetos, listas, etc. Además, ofrece programación genérica y herencia, que considero que son importantes para un lenguaje orientado a objetos como pretende ser. También dispone de un tipo de datos llamado dynamic (como el que tiene C#), que permite adoptar cualquier tipo de datos y se “tranformará” en el tipo del valor que se le asigne, empezando en null al estar vacío. También se puede modificar el tipo asignándole otro dato de tipo diferente.

Una parte muy positiva de este lenguaje es que es de código abierto, por lo tanto es gratuito y recibe actualizaciones constantemente. Aunque sea de código abierto no impide que tenga el soporte de Google por lo que parece que pese a ser un lenguaje nuevo ha venido para quedarse. Además, como Flutter es también de Google están diseñados en la misma dirección para poder trabajar juntos.
En [8], se explica que se ha que Dart es realmente fácil de aprender debido a que su sintaxis tiene características que son familiares para los usuarios de los lenguajes estáticos y dinámico, por lo que aprender su uso si se tiene una cierta base de programación no es para nada complicado.
Hay una herramienta muy interesante para este lenguaje que se trata de DartPad, en la que te permite hacer pruebas del lenguaje Dart e incluso ver algunos ejemplos realizados en ese lenguaje. Sin duda es una muy buena opción para probar el lenguaje.

2.5 Diseño de la Arquitectura

Para EasyMov he usado un principalmente un **diseño MVC** (Modelo – Vista - Controlador), que permite separar en capas de componentes diferentes los datos de la aplicación (Modelo), la interfaz (Vista) y la lógica (Controlador). Cada uno de los componentes tendrá una responsabilidad dentro de la aplicación, y al separarlo en tres capas nos aseguramos que cualquier modificación no afecte a las otras capas.

Pero no en todos los componentes de la aplicación se ha seguido este diseño, ya que en algunos casos, sobre todo en la gestión del mapa, he preferido optar por el **patrón BLoC**.
Este patrón se basa en BLoCs, que son componentes intermedios entre las vistas y el modelo. Esto es muy útil ya que ayuda a acceder a los datos y a gestionar estados desde un punto central del proyecto. En teoría, si nos basásemos en este patrón para diseñar toda la aplicación, nos haría falta usarlo en todos los widgets pero sí en los más importantes.

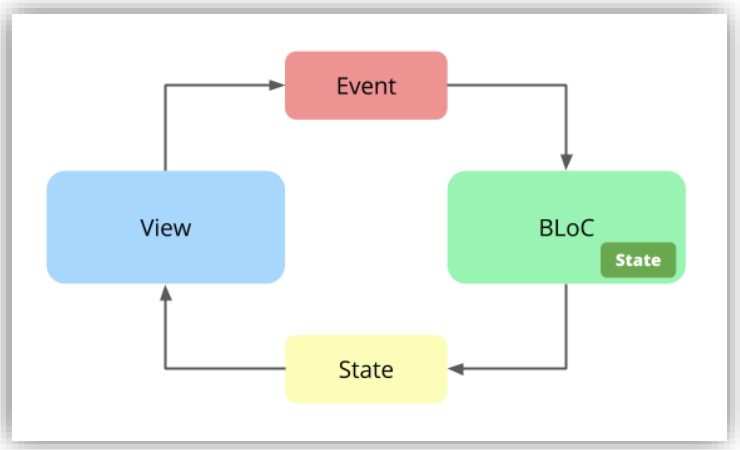


Figura 15. Esquema patrón BLoC (fuente xurxodev.com)

El funcionamiento se basa en eventos y estados, ya que hay un flujo unidireccional en el que cuando la vista produce un evento concreto (por ejemplo, cambiar la ubicación actual del usuario en el mapa), el BLoC lo captura y notifica a todos los widgets que están pendientes de este estado las novedades observadas.
El BLoC a efectos prácticos es como una caja negra que recibe eventos como inputs y tiene un estado observable como outputs, aunque como se ha hecho en nuestro caso, no es necesario que solo haya una entrada y una salida, sino que un mismo bloc podemos tener varias entradas para varias salidas, accediendo a las diferentes variables de estado según sea necesario.

Es realmente útil usar el patrón bloc ya que la vista estará en todo momento pendiente de cambios de estado, por lo que cuando se provoque el cambio se podrá lanzar una acción sin tener que redibujar expresamente el widget que lo reciba, o enviar datos únicamente en ese momento.

2.4.1 UML de entidades y clases

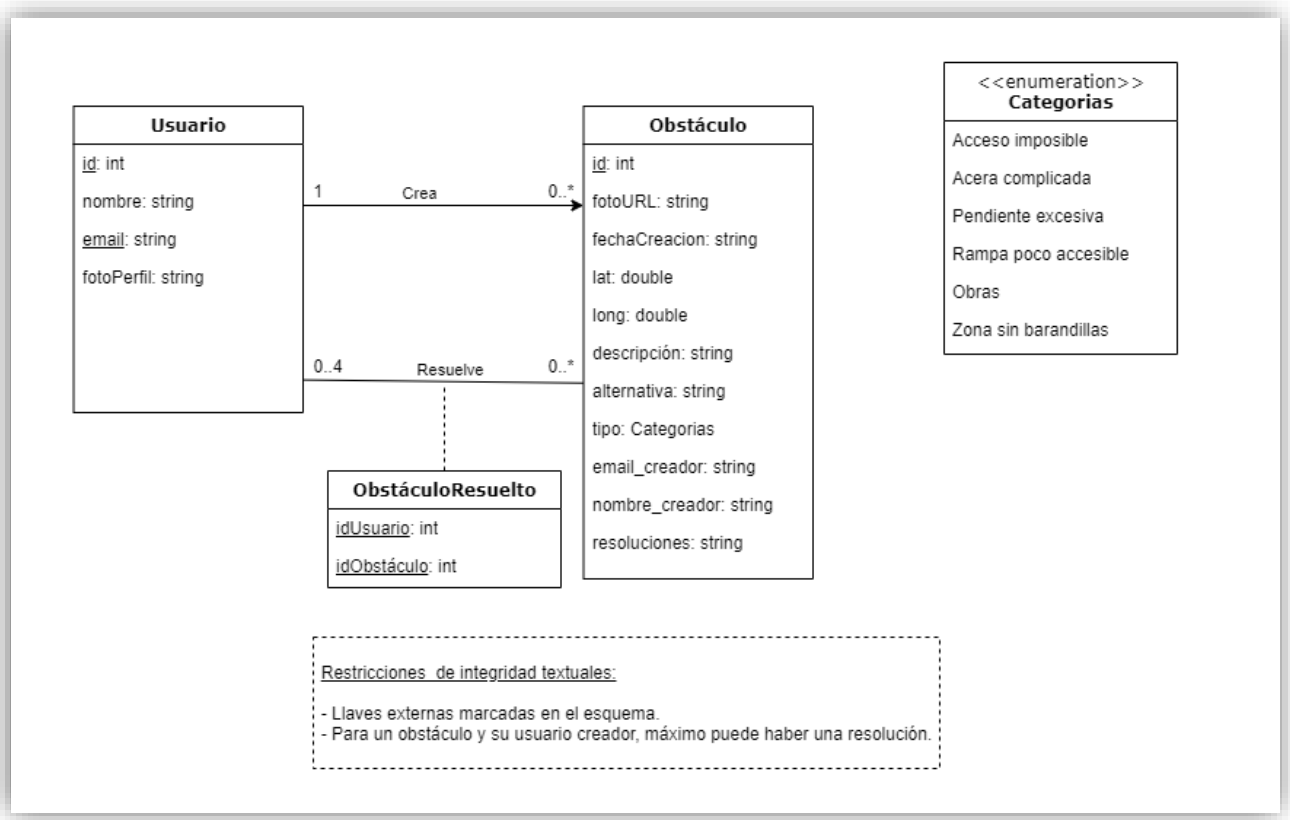


Figura 16. Diagrama de clases y entidades

USUARIO	
Atributo	**Descripción**
id	Identificador del usuario.
nombre	Nombre público del usuario.
email	Email del usuario.
fotoPerfil	Foto de perfil del usuario.

Tabla 5. Detalles de atributos de la clase Usuario.

OBSTÁCULO	
Atributo	**Descripción**
id	Identificador del obstáculo.
fotoURL	Foto del obstáculo.
fechaCreacion	Fecha en la que se ha creado el obstáculo.
longitud	Coordenada longitud del obstáculo.
latitud	Coordenada latitud del obstáculo.
descripción	Descripción extensa del obstáculo.
alternativa	Alternativa para evitar el obstáculo.
tipo	Categoría a la que pertenece el obstáculo.
email_creador	Email del usuario que ha creado el obstáculo.
nombre_creador	Nombre del usuario que ha creado el obstáculo.
resoluciones	Lista con el email de los usuarios que han resuelto el obstáculo.

Tabla 6. Detalles de atributos de la clase Obstáculo.

2.4.2 Diagrama arquitectura

En cuanto a los elementos de la arquitectura, he decidido usar **Firebase** para gestionar las cuentas de usuario y los datos de los obstáculos, los motivos de la elección han sido la facilidad de su uso, su rápidez en las operaciones y el haberlo usado previamente en asignaturas del máster.

Para el mapa se usará la **API de Maps**, que nos ofrece un gran rendimiento y poca latencia para mostrar los datos, que obtendremos de la **API de Mapbox**, a través de las herramientas para generar rutas entre dos puntos (*Directions*) y la API para obtener información de las direcciones a través de las coordenadas del mapa (*Geocoding*). He decidido usar esta solución ya que la API de maps que ofrece estas características (*Places*), es de pago y para realizar el trabajo con el plan gratuito que ofrece Mapbox es suficiente.

Para las imágenes se usará el almacenamiento de imágenes de **Cloudinary**, que permiten ser consultadas mediante API. Se había valorado usar el propio *Firebase Storage*, pero por facilidad de implementación y buenos resultados en proyectos anteriores, me he decantado por Cloudinary.

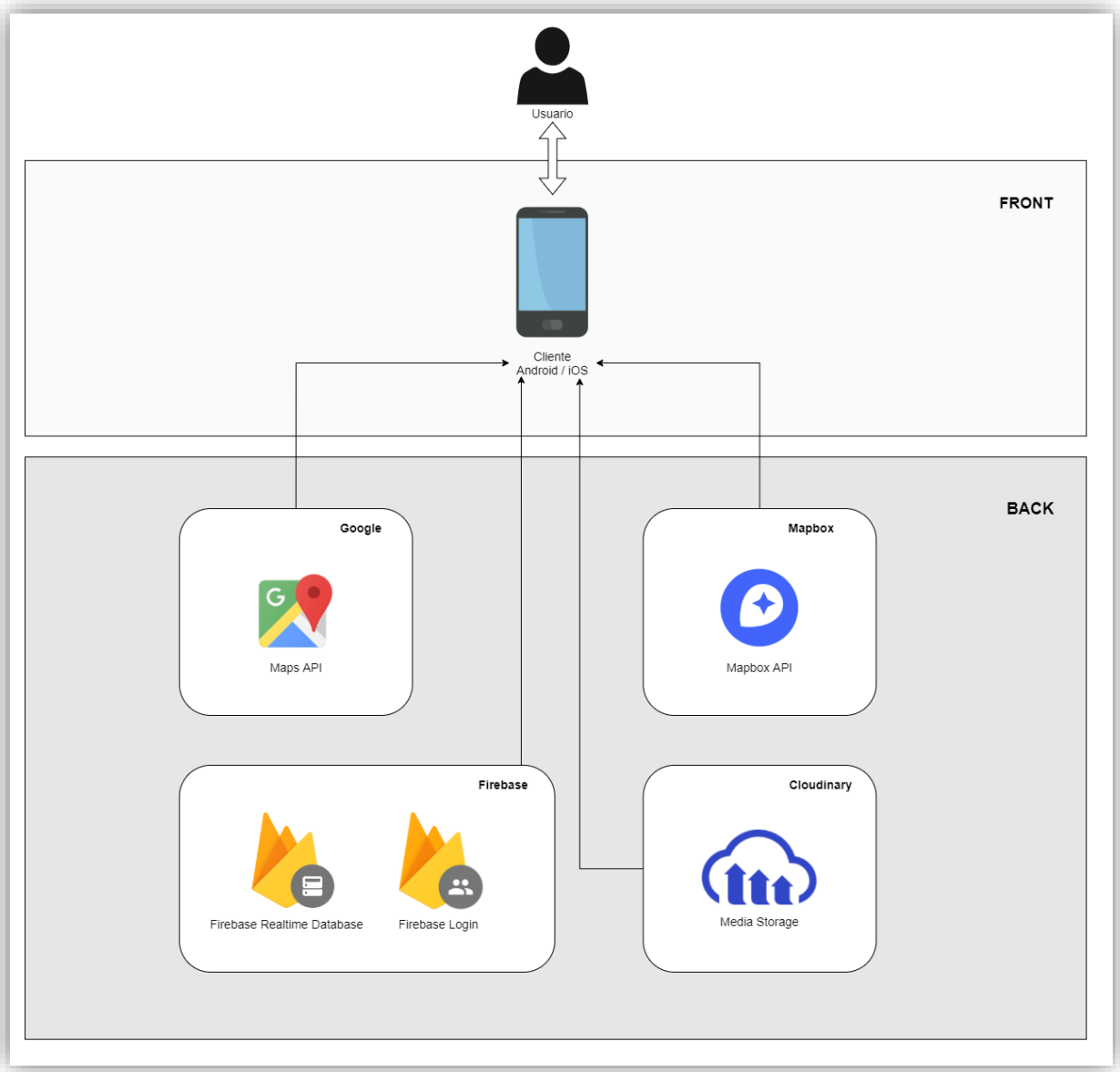


Figura 17. Diagrama de Arquitectura.

2.4.2 Diagrama de componentes

Gracias a generar un diagrama de componentes, podemos visualizar **la organización de los componentes del sistema y también ver las relaciones de dependencia** que hay entre ellos. Este tipo de diagramas tienen un nivel más alto de abstracción que un diagrama de clase, ya que un componente usualmente se suele implementar por más de una clase.
Para nuestro proyecto, el diagrama de componentes generado es el siguiente.

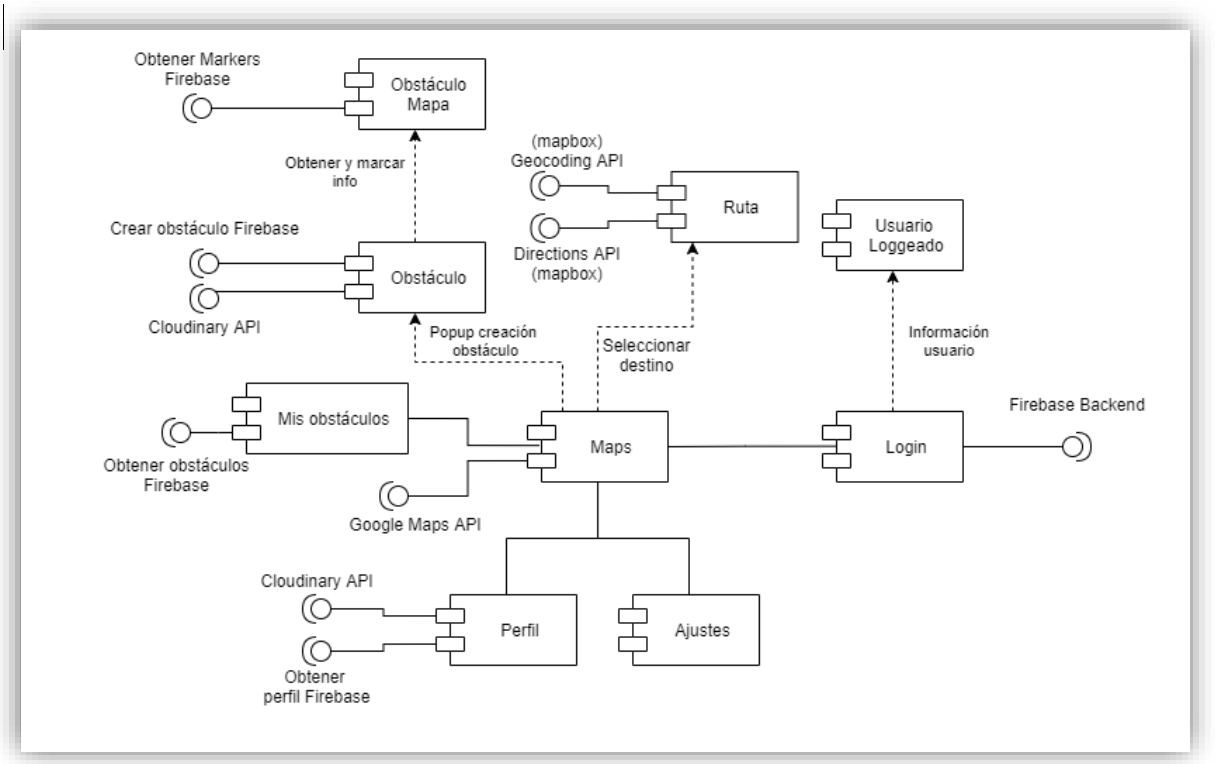


Figura 18. Diagrama de componentes.

La navegación en el diagrama de componentes se iniciaría con el login / registro de usuarios, conectándose internamente con el backend de Firebase y dando como resultado un usuario logueado con toda la información de usuario cargada en las preferencias.
A continuación, se encuentra Maps como componente central de nuestra aplicación, este utiliza el paquete de flutter de maps que internamente realiza llamadas a la API.
Desde el mapa (y en general desde cualquier pantalla a través del menú lateral) se puede acceder al perfil, que también tiene conexión con Firebase para actualizar los detalles del usuario, si es necesario y también se puede acceder a los ajustes, que no tienen conexión con elementos externos pero sí que modifican elementos internos que se almacenan en las preferencias del usuario. También podemos acceder a mis obstáculos, que carga a través de Firebase el conjunto de obstáculos del usuario.
Al crear un obstáculo nuevo desde el popup de creación se sube la imagen en Cloudinary y finalmente se crea el obstáculo en Firebase con los datos introducidos. Una vez se ha creado el obstáculo, se obtienen los obstáculos de firebase y se pintan en el mapa.
Al seleccionar destino en el mapa, se obtiene la ruta desde Directions de mapbox y también se obtiene la dirección que hay en esas coordenadas.

3. Implementación

En este apartado de la memoria, se explicará las principales decisiones de diseño que se han tenido en cuenta para el desarrollo de la aplicación.

3.1 El proyecto Android e iOS

El proyecto se ha desarrollado utilizando el IDE **Visual Studio Code** en su versión 1.56, la más actual. El uso de VS Code nos ha permitido trabajar con las siguientes extensiones oficiales:

- **Awesome Flutter Snippets:** Plugin que permite añadir fragmentos y accesos directos para funciones y clases de Flutter de uso común.
- **Bracket Pair Colorizer 2**: Permite añadir colores a los corchetes para mostrar cuando están abiertos e identificar qué sección de código comprende cada uno.
- **Dart:** Añade el soporte para dart permitiendo debugar ese lenguaje.
- **Flutter:** Permite ejecutar Flutter desde VS Code, permitiendo abrir el emulador de Android desde el IDE.
- **Material Icon Theme:** Tema de iconos para los directorios del proyecto, basados en el tema de Material.
- **Paste JSON as Code:** Permite pegar JSON y tranformarlo a definiciones de clase de forma automática.
- **Prettier Formatter:** Aplica automáticamente estilo al código al guardar cambios, teniendo en cuenta la longitud máxima de línea e identaciones, ajustando el código cuando sea necesario.
- **Terminal:** Permite ejecutar la terminal bash directamente desde VS Code.

Para iOS, no se ha definido versión mínima, ya que no se ha hecho la creación de paquetes necesarios porque, aunque el código fuente sería el mismo, **no se ha desarrollado la aplicación para esa plataforma debido a que no se dispone de un dispositivo macOS** y solo se puede desarrollar en ese dispositivo. En caso de desarrollar para iOS se tendría que usar el XCode, ya que es el único que permite emular dispositivos iOS.

En cuanto a la versión de Android que se ha especificado para este proyecto, se ha intentado buscar un equilibrio entre la versión que nos permita llegar al máximo de usuarios y a su vez permita ejecutar correctamente todas las funcionalidades que incorpora nuestra aplicación. Tras analizar las versiones mínimas para ejecutar todas las funcionalidades, se ha decidido usar como **versión mínima la versión 21 del SDK de Android**. Esta versión es conocida como Android Lollipop y equivale a la versión 5.0. Con esta versión nos aseguramos llegar a muchos usuarios ya que salió en noviembre de 2014 y por lo tanto, todos los dispositivos actualizados con fecha posterior a esa, podrán usar la aplicación. Actualmente está próxima a lanzarse la versión 31 del SDK, llamada Android 12

3.2 Principales paquetes usados

Para importar paquetes externos en Flutter se debe añadir el paquete en el apartado de depdencias del fichero *pubspec.yaml* junto con la versión (si se deja en blanco se usará la versión más reciente del paquete).

```yaml
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  rxdart: ^0.26.0
  http:
  image_picker: ^0.7.4
  mime_type: ^1.0.0
  shared_preferences: ^2.0.5
  google_maps_flutter:
  rflutter_alert: ^2.0.2
  path_provider: ^2.0.1
  animate_do: ^2.0.0
  dio: ^4.0.0
  flutter_bloc: ^7.0.0
  permission_handler: ^7.1.0
  geolocator: ^7.0.3
  polyline: ^1.0.2
```

Figura 19. Dependencias del proyecto.

- **cupertino_icons:** Paquete de iconos para añadir el estilo de Apple.
- **rxdart:** Implementación de reactiveX para programación asíncrona y streams.
- **http:** Paquete para realizar consumir recursos http de forma sencilla.
- **image_picker:** Paquete para añadir el selector de imágenes desde cámara o galería.
- **mime_type:** Paquete para obtener la extensión y tipo de un archivo.
- **shared_preferences:** Paquete para guardar pares de clave-valor que almacenen información no crítica.
- **google_maps_flutter:** Paquete para integrar los mapas de Google en forma de widget.
- **rflutter_alert:** Paquete para generar popups o alertas de forma sencilla.
- **path_provider:** Paquete para encontrar rutas de ubicaciones como los directorios temp o app data.
- **animate_do:** Paquete de animaciones para widgets inspirado en anímate.css.
- **dio:** Paquete que añade un cliente http, que permite añadir FormData o descarga de ficheros.
- **flutter_bloc:** Paquete que permite implementar el patrón de diseño BLoC (Business Logic Component) de manera sencilla.
- **permission_handler:** Paquete que proporciona una API para solicitar y validar que se han aceptado permisos para usar la aplicación.

- **geolocator:** Paquete que añade funciones de geolocalización, desde la localización del usuario hasta calcular distancia en metros entre dos puntos.
- **polyline:** Paquete que permite generar una polyline (ruta de maps) a partir de una lista de coordenadas.

3.3 APIs utilizadas

Aunque las APIs utilizadas en el proyecto se han nombrado en el *apartado 2.4.2 Diagrama de Arquitectura*, a continuación, las explicaremos con un poco más de detalle. Pese a que se han usado muchos paquetes que incluyen sus propias APIs, estas son las principales del proyecto:

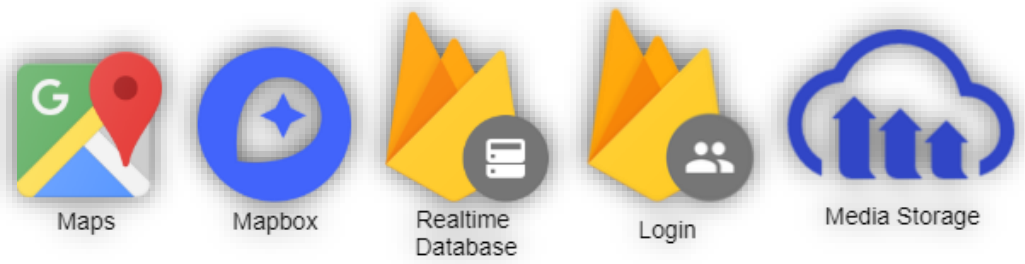


Figura 20. Conjunto de APIs utilizadas

- **Maps API:** Mediante el paquete de Google maps, podemos generar un widget que nos pinta el mapa de Google y añadirle información.

```
return GoogleMap(
  mapToolbarEnabled: false,
  mapType: tipoMapa(),
  myLocationButtonEnabled: false, //Muestra el botón de ubicación
  zoomControlsEnabled: false, // Botones de Zoom
  myLocationEnabled: true,
  initialCameraPosition: cameraPosition,
  onMapCreated: (GoogleMapController controller) {
    mapaBloc.initMapa(controller);
  },
  polylines: mapaBloc.state.polylines.values.toSet(),
  markers: mapaBloc.state.markers.values.toSet(),
  onCameraMove: (cameraPosition) {
    mapaBloc.add(OnMovioMapa(cameraPosition.target));
  },
); // GoogleMap
```

Figura 21. Código para dibujar mapa.

En este widget, se le pueden configurar multitud de parámetros, en nuestro caso hemos desactivado la toolbar, que es una barra para redireccionar a la app de maps al pulsar sobre un marcador, también hemos configurado el tipo de mapa, en función de lo que se seleccione en los ajustes, se ha desactivado el botón de la localización del usuario, que evita que se muestre el icono para centrar el mapa dónde está el usuario, ya que hemos creado nosotros uno personalizado. También hemos ocultado

los botones para hacer zoom, hemos configurado la posición inicial de la cámara en dónde se encuentra el usuario, hemos lanzado un evento cuando el mapa se crea, hemos cargado el conjunto de obstáculos y ruta que hay en el estado del BLoC del mapa y por último hemos lanzado un evento cada vez que se mueve el mapa.
Como se ha comentado, este widget ofrece muchas más opciones de personalización, incluso el diseño del mapa se puede cambiar como se ha hecho y se hablará de ello en la sección de Temas.

- **Mapbox API:** Esta API ha sido utilizada como alternativa a Google Maps, para la gestión de puntos en el mapa. Ha sido realmente útil para poder usar herramientas que también ofrece Google, pero que debido a su coste no hubiese sido posible utilizar en este proyecto.
  Mapbox tiene diferentes productos disponibles, desde mapas y rutas hasta gestión con VR de los elementos que rodean a un conductor o grandes conjuntos de datos para ser usados en otras aplicaciones.
  Por otra parte, gracias a esta API nos permite mostrar la información en todo momento desde nuestra app, evitando llevar a los usuarios a una app externa y pudiéndolos perder por el camino.

  - **Directions:** Este producto de mapbox permite obtener un conjunto de puntos (ruta) al enviar unas coordenadas de origen y destino en la url donde se hace la petición.
    Como parámetros de query se puede enviar si se desea que la respuesta contenga rutas alternativas, el tipo de ruta que devuelve, las indicaciones de la ruta y el lenguaje de la respuesta.

```dart
final _baseUrlDir = 'https://api.mapbox.com/directions/v5';
final _baseUrlGeo = 'https://api.mapbox.com/geocoding/v5';
final _apiKey =
    'pk.eyJ1IjoiZW9tZXIwMSIsImEiOiJja29sOWdlbDAwYnV0MnZwZnFlMzRlbGI4In0.gTub6qtVVaX1M7Xjm81-0A';

Future<DrivingResponse> getCoordsInicioYDestino(
    LatLng inicio, LatLng destino) async {
  final coordString =
      '${inicio.longitude},${inicio.latitude};${destino.longitude},${destino.latitude}';
  final url = '${this._baseUrlDir}/mapbox/walking/$coordString';
  final resp = await this._dio.get(url, queryParameters: {
    'alternatives': 'true',
    'geometries': 'polyline6',
    'steps': 'false',
    'access_token': this._apiKey,
    'language': 'es,ca',
  });
  final data = DrivingResponse.fromJson(resp.data);

  return data;
}
```

Figura 22. Ejemplo uso Directions API.

- **Geocoding**: Este producto de mapbox, permite obtener tanto las coordenadas de una búsqueda como información sobre unas coordenadas, entre otras cosas.
  En nuestro caso, hemos realizado de esta forma la búsqueda de coordenadas a partir de texto:

```dart
Future<SearchResponse> getResultadosPorQuery(
    String busqueda, LatLng proximidad) async {
  final url = '${this._baseUrlGeo}/mapbox.places/$busqueda.json';
  try {
    final resp = await this._dio.get(url, queryParameters: {
      'access_token': this._apiKey,
      'autocomplete': 'true',
      'proximity': '${proximidad.longitude},${proximidad.latitude}',
      'language': 'es',
    });

    final searchResponse = searchResponseFromJson(resp.data);

    return searchResponse;
  } catch (e) {
    return SearchResponse(features: []);
  }
}
```

Figura 23. Ejemplo búsqueda dirección.

Se envía en la url el string que ha buscado el usuario y como query parameters de la llamada get, el autocomplete, que envía sugerencias al usuario, la proximidad, que se trata de una coordenada desde dónde se busca para mostrar resultados más cercanos y el lenguaje.

Por otra parte, también se usa para obtener información a partir del envío de una coordenada por url, a través de la funcionalidad reverse geocoding.

```dart
Future<ReverseQueryResponse> getCoordenadasInfo(LatLng destinoCoords) async {
  final url =
      '${this._baseUrlGeo}/mapbox.places/${destinoCoords.longitude},${destinoCoords.latitude}.json';

  final resp = await this._dio.get(url, queryParameters: {
    'access_token': this._apiKey,
    'language': 'es',
  });

  final data = reverseQueryResponseFromJson(resp.data);

  return data;
}
```

Figura 24. Ejemplo reverse geocoding.

En este caso, se envía por url el punto y como query parameter el lenguaje y el servicio nos devuelve un json con toda la información disponible del punto, con más o menos detalle, según necesitemos. En nuestro caso se usa para mostrar la dirección de inicio y final en las rutas que genera el usuario.

- **Firebase:** Se trata de una plataforma en la nube que permite simplificar el backend de una aplicación web o móvil, lo bueno es que sus herramientas están agrupadas en una misma plataforma y con un funcionamiento sencillo y muy parecido entre ellas.
  Para usar sus funcionalidades desde nuestra aplicación, debemos guardar el token de firebase, que se obtiene desde el sitio web.

  - **Login:** En nuestro caso, los usuarios deben estar identificados para acceder a las características de la aplicación. Después de valorar diferentes servicios de login se vió que Firebase era el más simple y el que mejor cumplía las necesidades del proyecto. Firebase ofrece el sistema de autenticación que permite tanto el registro con usuario y contraseña como el inicio de sesión. También se podría implementar el acceso mediante otras plataformas como Facebook o Google, pero no ha formado parte del alcance de este proyecto.
    La implementación de ambas funcionalidades es muy parecida y únicamente dista del endpoint al que se lanza la petición post.

    Para el registro de un nuevo usuario, el código es este:

```dart
Future<Map<String, dynamic>> nuevoUsuario(
    String email, String password) async {
  final authData = {
    'email': email,
    'password': password,
    'returnSecureToken': true
  };

  final resp = await http.post(
      Uri.parse(
          'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=$_firebaseToken'),
      body: json.encode(authData));

  Map<String, dynamic> decodedResp = json.decode(resp.body);

  print(decodedResp);

  if (decodedResp.containsKey("idToken")) {
    _prefs.token = decodedResp['idToken'];
    _prefs.refreshToken = decodedResp['refreshToken'];

    return {'ok': true, 'token': decodedResp['idToken']};
  } else {
    return {'ok': false, 'mensaje': decodedResp['error']['message']};
  }
}
```

    Figura 25. Código para el registro de usuarios

    En el caso de que el usuario ya esté registrado, la respuesta no tendrá un idToken, por lo que podemos retornar un “ok” en estado false, que capturaremos desde la pantalla de registro para mostrar a los usuarios que ese correo y contraseña ya está registrado en el sistema.
    Para el inicio de sesión en la aplicación el código es muy parecido, cambiando el endpoint al que hacemos la llamada post por el correspondiente al login.
    También debemos enviar el email y el password como cuerpo de la petición.

```dart
Future<Map<String, dynamic>> login(String email, String password) async {
  final authData = {
    'email': email,
    'password': password,
    'returnSecureToken': true
  };

  final resp = await http.post(
      Uri.parse(
          'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=$_firebaseToken'),
      body: json.encode(authData));

  Map<String, dynamic> decodedResp = json.decode(resp.body);

  print(decodedResp);

  if (decodedResp.containsKey("idToken")) {
    //Guardamos el token en el storage
    _prefs.token = decodedResp['idToken'];
    _prefs.refreshToken = decodedResp['refreshToken'];
    return {'ok': true, 'token': decodedResp['idToken']};
  } else {
    return {'ok': false, 'mensaje': decodedResp['error']['message']};
  }
}
```

Figura 26. Código para el login de usuarios.

De la misma forma que para el registro, en caso de que la respuesta no contenga el idToken, devolveremos un “ok” a false, junto con el mensaje de error para luego mostrar en la pantalla de inicio de sesión que el usuario y contraseña no es correcto.

Como se puede observar, las tareas de login y registro de usuarios se hacen de forma muy simple, ya que simplemente debemos enviar a Firebase el usuario y contraseña y él se encarga de lo demás. Además, debemos tener en cuenta que desde allí se gestionan los accesos con total seguridad, garantizando una mayor seguridad y protección de datos de lo que seguramente podríamos ofrecer implementando un sistema de login y registro en la propia aplicación.

- **Realtime Database:** Esta es una de las herramientas más destacadas de Firebase, se trata de una base de dato en tiempo real, que permite alojar y disponer de los datos en tiempo real, manteniendo estos actualizados sin necesidad de que el usuario realice ninguna acción.
  Estas bases de datos se alojan en la nube y son del tipo No SQL, por lo que almacenan los datos en formato JSON.
  En este caso, se ha usado esta base de datos para gestionar información de los obstáculos de la aplicación y de la información de usuarios. Para ellos, se han realizado las funciones CRUD (en inglés Crear, Leer, Actualizar y Borrar).
  Un ejemplo de código de estas funciones, en este caso de creación de obstáculo en la base de datos sería:

```dart
//crear obstaculo en BBDD
Future<bool> crearObstaculo(ObstaculoModel obstaculo) async {
  await idToken(_prefs.refreshToken);

  final url = Uri.parse("$_url/obstaculos.json?auth=${_prefs.token}");

  final resp = await http.post(url, body: obstaculoModelToJson(obstaculo));

  final decodedData = json.decode(resp.body);

  print(decodedData);

  return true;
}
```

Figura 27. Ejemplo CRUD de Firebase

Un ejemplo de vista de la BD desde la consola de Firebase, mostrando los dos conjuntos, el de obstáculos y el de usuarios. En cada una de las entradas Firebase le asigna un ID único, que lo identifica y mediante el cual podemos acceder a modificar los datos. El ejemplo sería el siguiente:

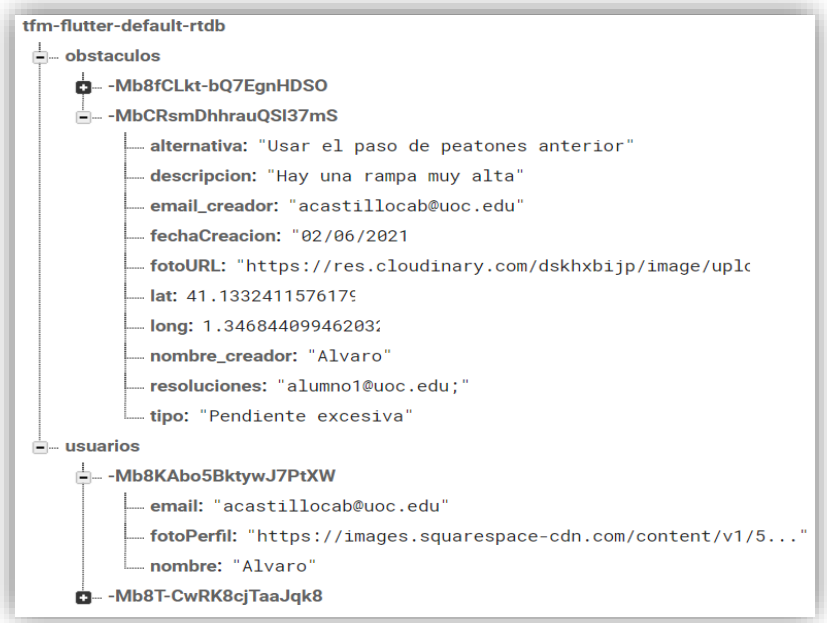


Figura 28. Ejemplo de la vista de BD

- **Cloudinary Media Storage:** Cloudinary es un servicio en la nube que permite almacenar imágenes en su servidor y posteriormente poder disponer de ellas mediante una url.

  Este servicio utiliza una red de distribución que permite que el usuario obtenga los recursos de el servidor más cercano de donde se encuentra, reduciendo así el tiempo de entrega en comparación de un servicio que tenga un único servidor. Por otra parte, también añade un cacheo de los recursos que evita tener que buscarlos en el servidor en cada petición, reduciendo también el tiempo de entrega.
  Por último, podemos jugar con la URL para modificar características de calidad o tamaño de las imágenes.

  Un ejemplo de subida de imágenes, en el que se envía a una url la imagen con información de su tipo, sería el siguiente.

```dart
Future<String> subirImagen(File imagen) async {
  final url = Uri.parse(
      'https://api.cloudinary.com/v1_1/dskhxbijp/image/upload?upload_preset=nt7byz24');

  final mimeType = mime(imagen.path).split('/');

  final imageUploadRequest = http.MultipartRequest('POST', url);

  final file = await http.MultipartFile.fromPath('file', imagen.path,
      contentType: MediaType(mimeType[0], mimeType[1]));

  imageUploadRequest.files.add(file);
  final streamResponse = await imageUploadRequest.send();
  final resp = await http.Response.fromStream(streamResponse);

  if (resp.statusCode != 200 && resp.statusCode != 201) {
    print("Algo salió mal");
    print(resp.body);
    return null;
  }

  final respData = json.decode(resp.body);

  print(respData);

  return respData['secure_url'];
}
```

Figura 29. Ejemplo código subida imagen

El uso de APIs es la única barrera que puede impedir que logremos alcanzar el **requisito no funcional de disponibilidad,** ya que lo demás está ejecutándose sobre el dispositivo. La disponibilidad se ha fijado en un 99%. Por suerte, analizando cada disponibilidad por separado no hemos detectado que ninguna tenga un valor inferior, por lo que parece que no habrá problema para alcanzar esa disponibilidad.

3.4 Pantallas creadas

Se han generado 8 páginas o pantallas de nuestra aplicación, que a continuación detallaremos. Para navegar entre las pantallas se puede hacer a través de eventos de sesión iniciada y a través del menú lateral desplegable.

Pantallas	
Pantallas	**Descripción**
Login	Pantalla inicial de la aplicación que se muestra si el usuario no ha iniciado sesión previamente, permite validar que al introducir un email y contraseña, está registrado en nuestro conjunto de usuarios. En caso de que no sean correctos los datos, se muestra un error y en caso de que sean correctos, se redirige al usuario hacia la Home.
Registro	Pantalla que permite registrar a usuarios, se puede acceder a ella desde el Login. Se debe introducir un nombre de usuario, el email y la contraseña. En caso de que ese email y contraseña ya exista se mostrará un aviso, en caso contrario se redirige al usuario a la Home.
Acceso_gps	Pantalla que permite al usuario activar los permisos de GPS, tiene un botón para acceder a la configuración y activarlos.
Loading	Pantalla que valida que se han concedido los permisos de GPS y en caso de que no se hayan concedido se muestra un mensaje y se le indica al usuario que los conceda. Hasta que no se haya permitido usar el GPS no se permite continuar a la siguiente pantalla, que es la Home.
Home	Pantalla principal de la aplicación, en ella se carga el mapa con el conjunto de obstáculos de todos los usuarios y se permite generar nuevos. Este mapa es el que se puede cambiar de tipo a través de los ajustes y además se ha añadido un tema personalizado con otros colores. En el caso de que el usuario tenga ya sesión iniciada, se enviará a esta página, previamente validando a través del Loading que el usuario tiene los permisos GPS habilitados.
Perfil	Pantalla del perfil de usuario, en ella se puede modificar la foto de perfil y también el nombre de usuario. Se puede acceder a ella desde el menú lateral pulsando tanto en la foto de perfil del usuario como en la sección de Perfil.
Mis_obstaculos	Pantalla que muestra los obstáculos creados por el usuario que tiene la sesión iniciada. En ella se puede eliminar los obstáculos desplazando hacia la izquierda el obstáculo o pulsando sobre el botón de eliminar.
Ajustes	Pantalla que permite configurar un par de ajustes de la aplicación, como es el color principal para mejorar la visibilidad, cambiando el color verde por un color negro, requiriendo un reinicio de la aplicación. Por otra parte, también se permite modificar el tipo de mapa que dibuja Maps, pudiendo cambiar entre el normal, el de satélite y el hibrido, que combina detalles de los dos.

Tabla 7. Resumen de pantallas de la app

3.5 Providers

Los providers nos permiten establecer conexión con la base de datos de Flutter, para manejar información de los obstáculos y de los usuarios.
En ambos casos las operaciones que se han realizado son las mismas, accediendo a diferentes endpoints de la base de datos y obteniendo otro objeto como respuesta. Las operaciones son las siguientes:

Providers para obstáculos y usuarios	
Operación	**Descripción**
Crear	Permite crear un registro en la base de datos, en la lógica actual se podría crear el mismo obstáculo y volverlo a guardar en la base de datos, para los usuarios no sería posible ya que se comprueba si existe. Para crearlo, hay que enviar como parámetro el objeto en la función y en caso de que se haya podido realizar la operación se devuelve true.
Editar	Permite editar un registro de la base de datos, para ello, hay que enviar un objeto como parámetro y lo que se hace es buscar su id en la base de datos y actualizar los valores con los del objeto que se ha enviado.
Cargar	Permite obtener TODOS los registros de la base de datos de obstáculos o usuarios, una vez ha leído todos los valores, devuelve una lista con todos los objetos de esa base de datos.
Cargar por Email	Para cargar los registros de una de las bases de datos por email, se debe pasar como parámetro el email del que se quiera leer y se hará una búsqueda de todos los registros que tienen ese email como campo email. Posteriormente, se devolverá una lista con los objetos que cumplen la condición.
Borrar	Para eliminar de la base de datos un registro, se debe enviar como parámetro el ID de ese registro. En este caso solo está implementado para los obstáculos, ya que la eliminación de usuarios no está contemplada en el alcance. Se devuelve 1 si el usuario se ha borrado correctamente.
Subir Imagen	Permite subir una imagen a cloudinary, pasándole como parámetro el archivo que se quiere subir. Si se ha subido correctamente, devuelve la url de la imagen subida, en caso contrario devuelve null. Esto nos sirve para subir imágenes de obstáculos y también de los perfiles de usuario. Almacenando luego las url en la base de datos.

Tabla 8. Providers de BD usados

3.6 Uso de BLoC

En el apartado *2.5 Diseño de la Arquitectura*, se ha explicado el patrón BLoC como concepto de diseño, pero en este apartado vamos a explicar en concreto en qué puntos se ha usado.

Se pueden hacer dos implementaciones de patrón bloc, en una de ellas se ha gestionado todo en el mismo fichero y en la otra se ha definido por separado el bloc, los eventos y los estados, haciendo uso del paquete de bloc propio de flutter.
Estos son los BLoCs que se han definido en el mismo fichero pero haciendo uso de un provider para notificar los cambios definido en *provider.dart*:

- **Login bloc:** Este BLoC se usa en el login y en el registro, en este caso para validar que los datos introducidos son correctos. Para ello, se usa el concepto de sink, que se trata de una tubería en la que se introducen datos y se leen desde el otro extremo, pudiéndolos transformar al entrar en ella y finalmente hacer las validaciones oportunas combinando valores cuando son ciertos. Las fases serían las siguientes:

Introducir valores al stream

```
// Insertar valores al Stream
Function(String) get changeEmail => _emailController.sink.add;
Function(String) get changePassword => _passwordController.sink.add;
Function(String) get changeName => _nameController.sink.add;
```

Figura 30. Código insertar valores stream BLoC

Leer datos del stream

```
//Recuperar los datos del Stream
Stream<String> get emailStream =>
    _emailController.stream.transform(validarEmail);

Stream<String> get passwordStream =>
    _passwordController.stream.transform(validarPassword);

Stream<String> get nameStream =>
    _nameController.stream.transform(validarName);
```

Figura 31. Código leer datos stream BLoC

Validar datos del stream

```
//Validación del formulario login
Stream<bool> get formValidStream =>
    Rx.combineLatest2(emailStream, passwordStream, (e, p) => true);

//Validación del formulario registro
Stream<bool> get formValidStreamReg => Rx.combineLatest3(
    nameStream, emailStream, passwordStream, (n, e, p) => true);
```

Figura 32. Código validar datos stream BLoC

- **Obstáculos bloc**: En este caso también se ha unificado todo en un solo fichero completando la lógica con el *providers.dart.* Este bloc se ha destinado a la gestión de obstáculos, para cargar todos los obstáculos, cargar los obstáculos de un usuario por email, crear obstáculo, subir foto del obstáculo, editar un obstáculo o borrar un obstáculo.

  En los casos de cargar obstáculos, se han obtenido de firebase a través del provider que conecta con la BD y luego se han añadido a la tubería por la que "*fluyen*" los obstáculos que son cargados en la página de "*Mis Obstáculos*".
  Cuando se trata de las otras operaciones, se ha usado una tubería de carga que si se envía el valor true significa que está realizando una operación en BD y tras finalizar esa carga, se envía el valor false. Esto nos sirve por ejemplo para modificar colores de botones mientras se está cargando información en BD.

```
void crearObstaculo(ObstaculoModel obstaculo) async {
  _cargandoController.sink.add(true);
  await _obstaculosProvider.crearObstaculo(obstaculo);
  _cargandoController.sink.add(false);
}
```

Figura 33. Ejemplo de tubería de carga en BD

Por otra parte, tenemos los blocs en los que se ha separado el contenido en ficheros de bloc, state y event. En el fichero de state, tenemos definidas las variables que formarán parte del estado y que por lo tanto se irán actualizando desde los eventos y podrán ser consultadas. El fichero de events, se crean las clases abstractas[12] que definirán los eventos que se lanzarán. Por último, el fichero bloc, contiene la lógica asociada a los eventos y mediante un copyWith[13] del estado se puede modificar la variable del estado que queramos.

Los blocs de este tipo son los siguientes:

- **Búsqueda:** Este bloc corresponde a la búsqueda de ubicaciones para seleccionar como destino, buscando por dirección o seleccionando en el mapa, el agregar al historial la búsqueda por dirección y también la selección en el mapa de un nuevo obstáculo. Esto permite controlar variables de estado para mostrar iconos determinados o actualizar valores de variables.

[12] Clases que no tienen código asociado.
[13] Método que copia el contenido del objeto modificando algunos valores.

Eventos del BLoC de búsqueda	
Evento	**Detalles**
OnActivarMarcadorManual	Se lanza al seleccionar ubicación en el mapa para la ruta, ocultando la barra de búsqueda y mostrando un icono para regresar atrás.
OnDesactivarMarcadorManual	Se lanza al dejar de seleccionar destino en el mapa, mostrando la barra de búsqueda nuevamente.
OnAgregarHistorial	Se lanza al realizar una búsqueda añadiendo la dirección al historial.
OnAgregarObstaculo	Se lanza al seleccionar la ubicación del obstáculo en el mapa, ocultando la barra de búsqueda y mostrando una fecha para regresar, pero añadiendo el marcador de diferente color que en la búsqueda de destino.
OnDesactivarAgregarObstaculo	Se vuelve a mostrar la barra de búsqueda y se oculta la flecha para regresar.

Tabla 9. Eventos del BLoC de búsqueda

- **Mapa:** Este bloc contiene eventos del mapa, en el se tratan todas las acciones que puede realizar el usuario al interactuar con el mapa, además de lanzarse algunos eventos al cargar el mapa o al modificar la ubicación del usuario.

Eventos del BLoC de mapa	
Evento	**Detalles**
OnMapaListo	Se lanza al cargar el mapa, en él se carga el conjunto de obstáculos que aparecerán en el mapa.
OnMarcarRecorrido	Se lanza cuando el usuario pulsa el botón de marcar el recorrido que ha hecho con la aplicación abierta. Mostrando una línea con el camino que ha ido haciendo el usuario.
OnSeguirUbicacion	Se lanza cuando el usuario se va desplazando y el mapa tiene que cambiar su centro al nuevo punto donde se encuentra el usuario.
OnCrearRutaInicioDestino	Se lanza cuando el usuario crea una ruta indicando un destino y se dibuja en el mapa los marcadores y la ruta entre el punto actual y el destino.
OnMovioMapa	Se lanza al mover el mapa, actualizando la ubicación central con la nueva ubicación.
OnNuevaUbicacion	Se lanza cuando el usuario se cambia de ubicación, añadiendo ese punto a la ruta que ha realizado en la aplicación.
OnLimpiarMapa	Se lanza al pulsar el botón de resetear el mapa, eliminando todas las rutas dibujadas y también la que ha realizado el usuario.

Tabla 10. Eventos del BLoC de mapa

- **Mi ubicación:** Este BLoC simplemente contiene un evento para gestionar la ubicación del usuario, mediante este bloc se actualiza el punto del mapa donde se encuentra el usuario, obtenido con la información del GPS.

3.7 Widgets personalizados

Además de los widgets que ofrece flutter, se pueden crear widgets personalizados. Estos widgets pueden ser importados y usados en otras partes del código sin tener que volver a definirlos.

Widgets personalizados	
Nombre	**Detalles**
BtnLimpiarMapa	Botón con icono “sync” que lanza el evento OnLimpiarMapa del mapa bloc, para borrar rutas del mapa.
BtnMiRuta	Botón con icono “linear_scale” que lanza el evento OnMarcarRecorrido del mapa bloc, para pintar el recorrido que ha hecho el usuario.
BtnUbicacion	Botón con icono “my_location” que lanza la función moverCamara del mapa bloc, permitiendo centrar la cámara en la ubicación del usuario.
MarcadorManual	Stack de widgets que se carga al estar buscando una nueva ubicación seleccionando en el mapa, en él, se añade una flecha para volver y se muestra un marcador de color azul en la parte central central del mapa junto con un botón de “Confirmar destino”.
MarcadorNuevoObs	Stack de widgets que se carga al crear un nuevo obstáculo, primero mostrando un marcador de color rojo con el icono de problema de movilidad junto con el texto “Añadir obstáculo” y posteriormente mostrando el popup para introducir toda la información del obstáculo.
MenuWidget	Widget que carga un Drawer (menú hamburguesa) con todas las navegaciones a las diferentes pantallas y por último la opción de cerrar sesión.
SearchBar	Widget que crea la barra de búsqueda y carga el SearchDelegate para buscar direcciones en concreto.

Tabla 11. Widgets personalizados

3.8 Utils

En este directorio del código fuente se han definido diversos archivos con funcionalidades que son candidatas para ser usadas en otras partes del código. De esta forma nos aseguramos tener estas funcionalidades definidas en un sitio concreto para poder hacer cambios si fuesen necesarios.

- **Alerta de calcular destino:** En el fichero *calculando_alerta.dart* se valida si la plataforma es Android o iOS y en función de eso mostrar un tipo de alerta u otra, optimizada con el paquete de iconos de cada plataforma. La configuración de la alerta es parecida, simplemente se debe modificar el tipo de alerta y algún detalle más.

```dart
void calculandoAlerta(BuildContext context) {
  if (Platform.isAndroid) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: Text('Calculando Ruta'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            CircularProgressIndicator(),
          ],
        ), // Column
      ), // AlertDialog
    );
  } else {
    //Alerta para iOS
    showCupertinoDialog(
      context: context,
      builder: (context) => CupertinoAlertDialog(
        title: Text('Calculando Ruta'),
        content: CupertinoActivityIndicator(),
      ), // CupertinoAlertDialog
    );
  }
}
```

Figura 34. Alerta de cálculo de destino

- **Utilizar marcadores personalizados:** En el fichero *custom_image_markers.dart* se define la función para obtener una imagen de marcador personalizada para nuestra aplicación, que será usada para representar los obstáculos en el mapa.

Figura 35. Marcador personalizado para obstáculos

- **Aplazar búsquedas con debouncer:** Cuando el usuario está buscando direcciones en el mapa, hay que añadir un control para que no se estén lanzando búsquedas en todo momento, sino que se tarde unos milisegundos entre búsqueda y búsqueda.
  Para ello, se ha definido el fichero *debouncer.dart*, en el que se regula este comportamiento mediante un timer, teniendo en cuenta el tiempo y el valor introducido en ese momento.

```dart
class Debouncer<T> {
  Debouncer({this.duration, this.onValue});

  final Duration duration;
  void Function(T value) onValue;
  T _value;
  Timer _timer;

  T get value => _value;

  set value(T val) {
    _value = val;
    _timer?.cancel();
    _timer = Timer(duration, () => onValue(_value));
  }
}
```

Figura 36. Código debouncer búsquedas

- **Animación navegación:** En el fichero *navegar_fadein.dart* se ha definido una animación de estilo *fade in* que se ha usado al cambiar de pantalla en alguno de los casos. En este fichero se configura la duración de la animación y la opacidad de la transición.

- **Resto de utilidades:** Por último se ha definido el fichero *utils.dart* en el que se registran las demás funciones que serán usadas en la aplicación, a continuación se muestra una tabla con detalles.

Funciones utils.dart	
Función	**Descripción**
isNumeric	Pasando por parámetro un string, devuelve un booleano si se trata de un número convertido a String.
mostrarAlerta	Permite mostrar un popup de alerta con un mensaje personalizado, que se le pasa como parámetro.
today	Función que devuelve un String con la feca actual, en formato “DD-MM-AAAA”.
mostrarSnackbar	Permite mostrar un Snackbar en la parte inferior de la pantalla durante 1,75 segundos y mostrando el texto que se le pasa por parámetro.
actualizarPrefsDB	Permite actualizar valores de un mapa que guarda información temporal del usuario, esta información se guarda en el dispositivo.
getPerfilFoto	Permite obtener una foto predefinida para el perfil del usuario.

Tabla 12. Funciones utils.dart

3.9 Temas

Para los temas se han realizado dos configuraciones, uno para los temas relacionados con el menú y botones y otra configuración para definir un tema especial para el mapa.

Para la configuración de colores de menú se ha definido un tema personalizado dentro del main, en el que se ha especificado el color del AppBar y el modo oscuro y también se ha definido el color principal de la aplicación. Para ello se ha usado el color de la configuración de usuario (por defecto color Teal 0x008080) pero modificable a negro.

```
theme: ThemeData(
  appBarTheme: AppBarTheme(
      color: utils.colorPrincipal, brightness: Brightness.dark), // AppBarTheme
  primaryColor: utils.colorPrincipal,
), // ThemeData
```

Figura 37. Cambio de tema de la aplicación

Por otra parte, para definir un tema personalizado de mapa, hemos accedido a la web de https://snazzymaps.com, en la que la comunidad genera estilos de mapa que pueden ser usados al desarrollar una aplicación, copiando el JSON de configuración en un fichero (en este caso easymovMapTheme) y cargándolo a través del controlador de mapa con la función .setMapStyle(jsonEncode(easymovMapTheme));

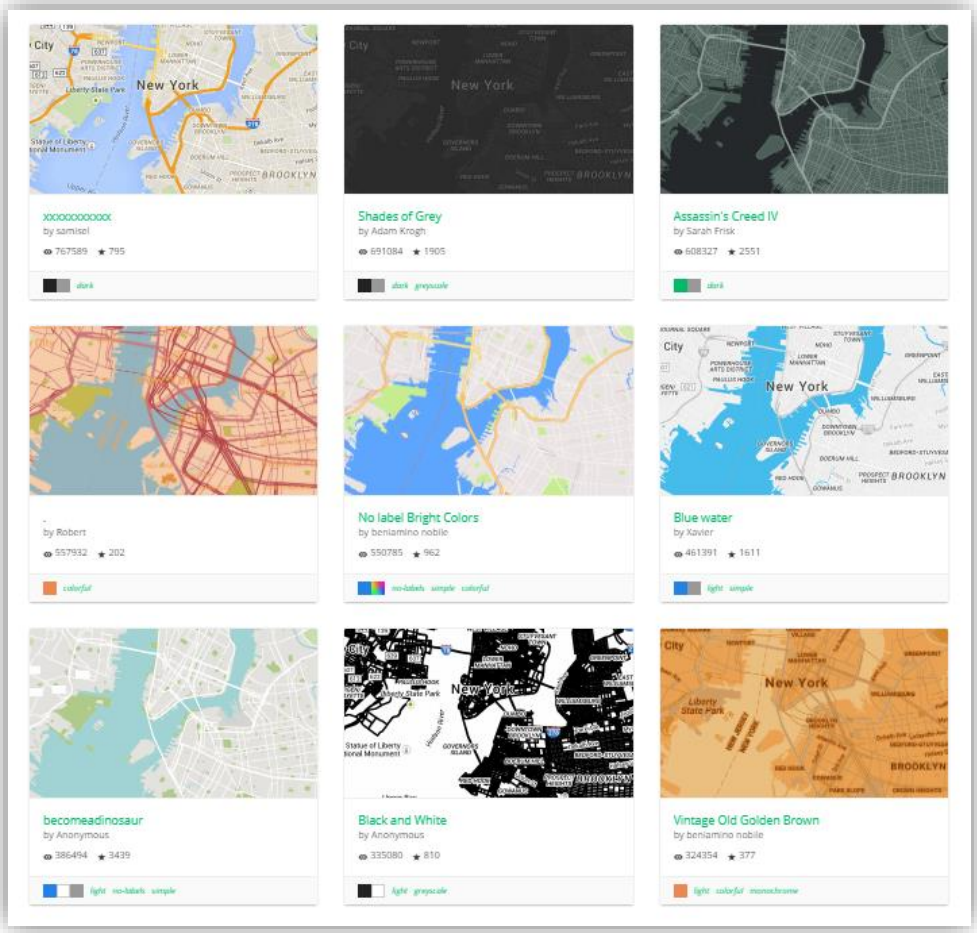


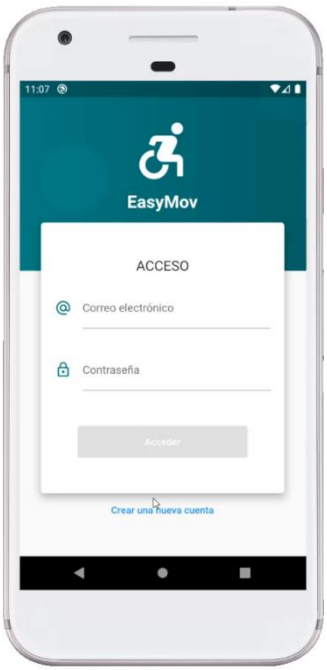
Figura 38. Listado de temas para mapas (snazzymaps.com)

3.10 Funcionalidades

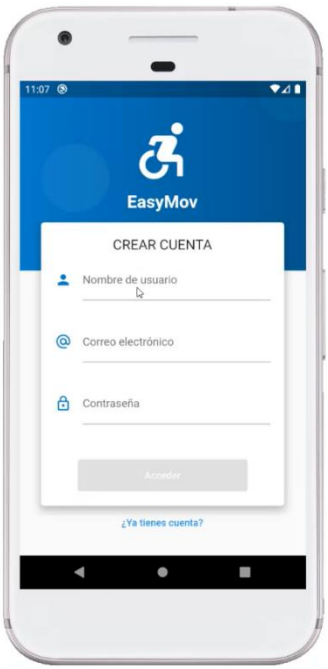
Vamos a detallar las diferentes funcionalidades que tiene nuestra aplicación, ahora que ya se ha explicado las decisiones de diseño que se han tenido en cuenta, nos centraremos en las decisiones de interfaz.

3.10.1 Login y Registro

Estas dos pantallas tienen una interfaz parecida, aunque en la parte de registro se solicita al usuario que introduzca el nombre de usuario. Ambas opciones tienen un control de los inputs y validaciones con Firebase Auth, por lo que si los datos no son correctos se muestra un mensaje de error. Se puede navegar de una pantalla a otra mediante el texto en azul de debajo de la caja de formulario.
Se ha decidido usar colores diferentes para indicar el login y el registro, para facilitar que el usuario distinga estas dos pantallas ya que son parecidas en cuanto a contenido.



Login



Registro

Error en usuario

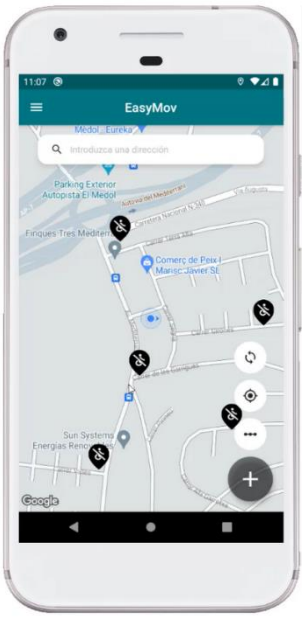
3.10.2 Pantalla principal

La pantalla principal es la pantalla de mapa, a ella se accede al abrir la aplicación si se ha iniciado sesión previamente o tras iniciar sesión o registrarte.

3.10.2.1 Interacción con el mapa

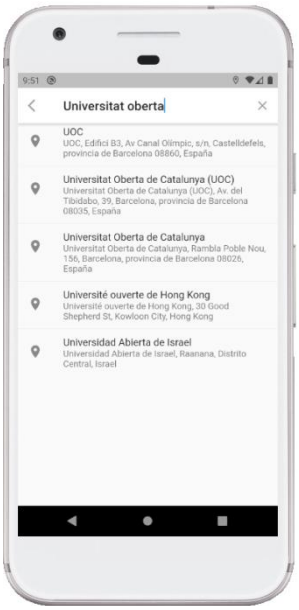
En esta pantalla se muestra el mapa, además se pueden trazar rutas y buscar ubicaciones concretas. También se puede trazar una ruta del movimiento del usuario al tener la app abierta. Los botones blancos de la derecha sirven para limpiar el mapa de rutas, que el mapa se centre a la ubicación del usuario o para mostrar la ruta que vas realizando al tener la app abierta.

Al abrir el buscador, se le da la opción al usuario de seleccionar el destino en el mapa o introducir la dirección, que cada 200ms refresca las sugerencias.

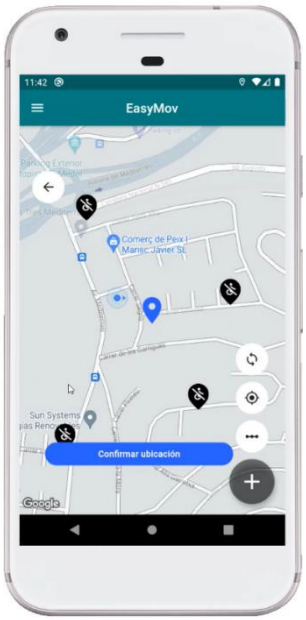


Mapa

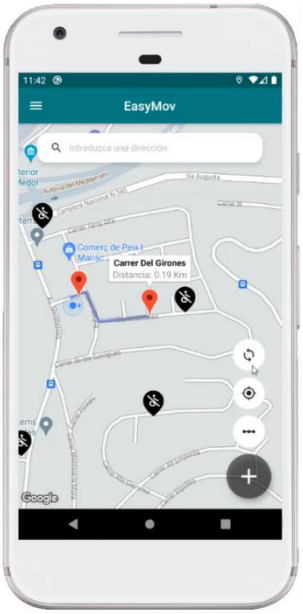
Búsqueda



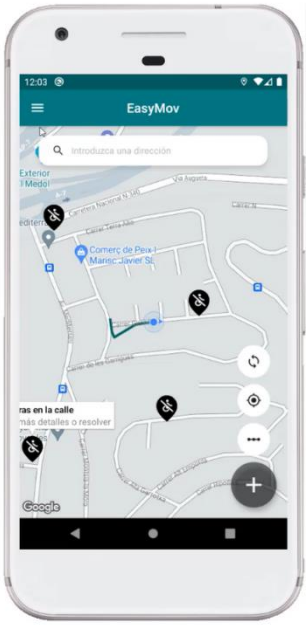
Sugerencias e historial



Seleccionar destino manualmente



Creación de ruta

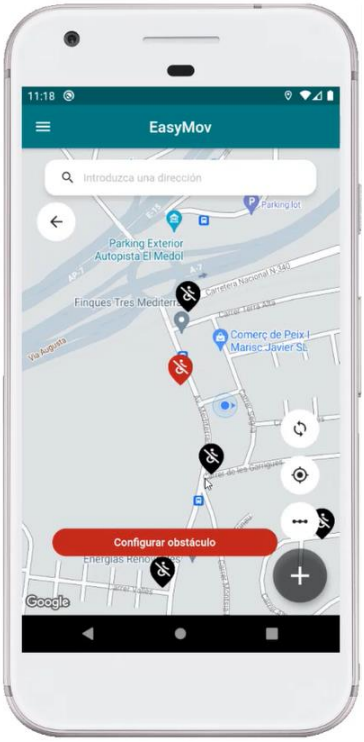


Ruta del usuario con la app abierta

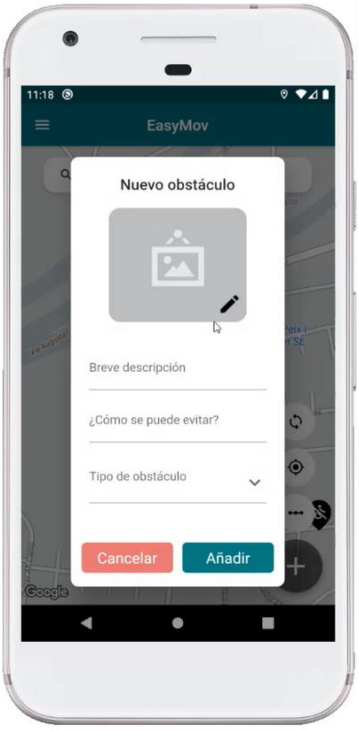
3.10.2.2 Creación de obstáculos

Al pulsar el botón de “+”, se permite al usuario seleccionar una ubicación en el mapa, en la que se encontrará el obstáculo. En este momento se guardan las coordenadas dónde se ubicará y se abre un popup para añadir los detalles del obstáculo. Para añadir la imagen desde la galería o la cámara, se muestra el mismo Picker que en la Imagen de perfil, que se muestra unos puntos más adelante. También se debe añadir una breve descripción y posteriormente añadir una forma de evitarlo y el tipo de obstáculo que se trata.

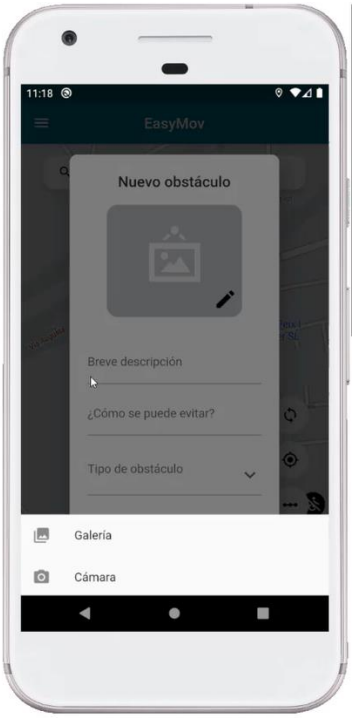
Tras pulsar añadir, se muestra un mensaje de que el obstáculo se ha creado correctamente.



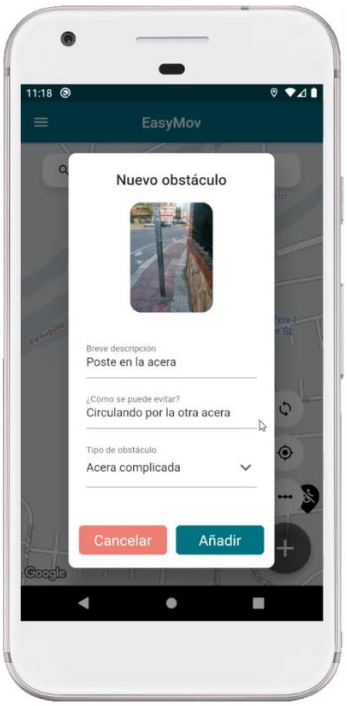
Seleccionar ubicación obstáculo



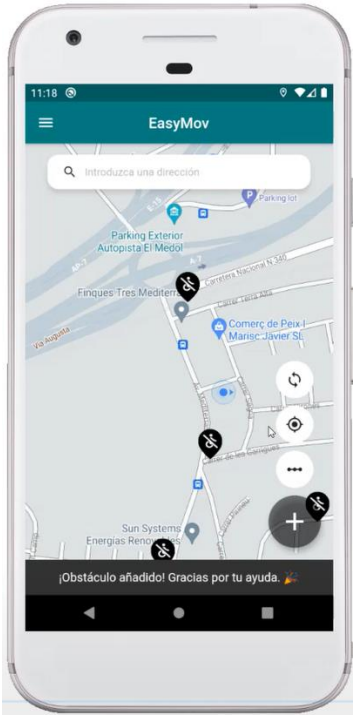
Popup creación obstáculo



Picker para seleccionar foto



Obstáculo con información completa

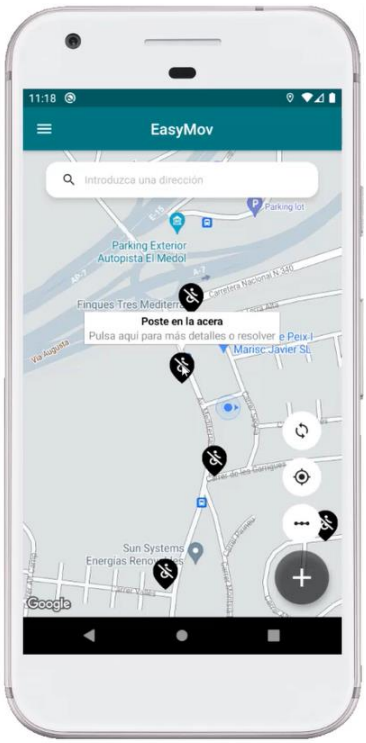


Snack de confirmación de creación.

3.10.2.3 Ver información de obstáculo y resolver

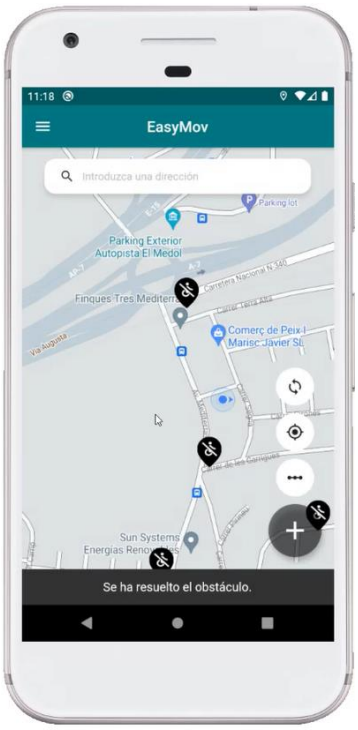
Al pulsar sobre uno de los marcadores del obstáculo, se nos muestra una ventana de información que nos muestra la descripción y nos permite pulsar para ver más detalles. En los detalles, se nos muestra la imagen, la descripción, la forma de evitarlo y el usuario que lo ha reportado.

Nos aparece la opción de resolver el obstáculo, si el obstáculo es nuestro y lo resolvemos se eliminará. Si es de otro usuario se necesitarán 4 resoluciones para eliminarlo, indicando las restantes al resolverlo. En el caso de que ya lo hayamos marcado como resuelto se nos indicará que ya lo hemos marcado como resuelto previamente y no hará nada.

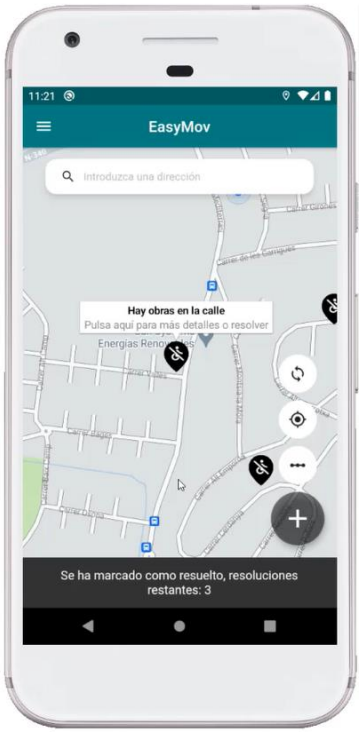


Ventana de información

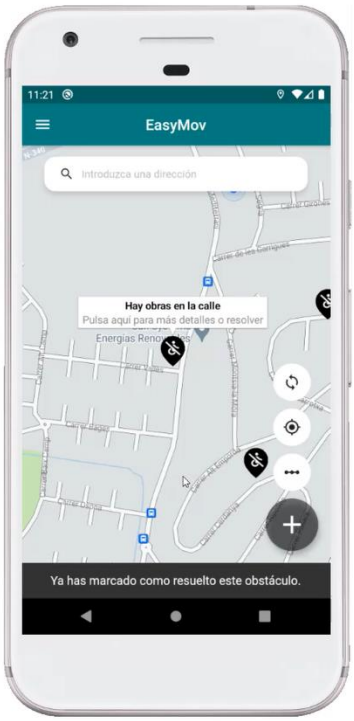
Información de obstáculo



Resolución de obstáculo



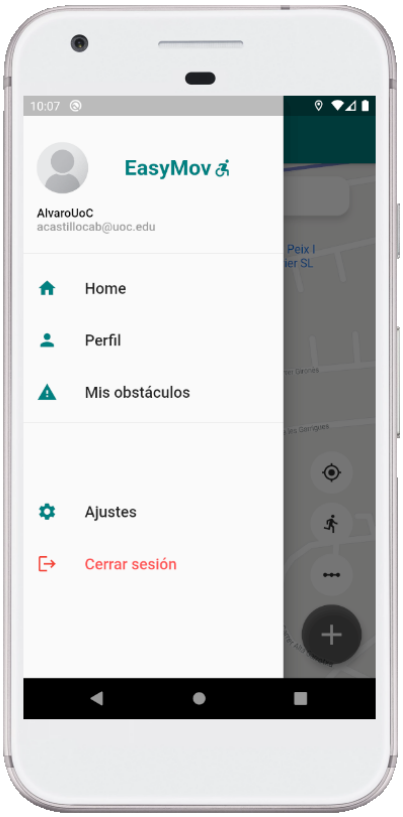
Resoluciones restantes



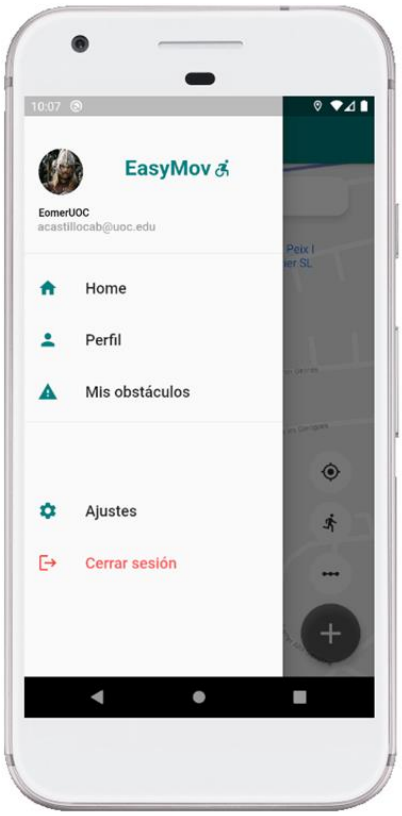
Obstáculo ya resuelto previamente

3.10.3 Menú lateral

Este menú lateral tiene una pequeña imagen del usuario (por defecto se muestra un avatar vacío) y luego incluye el nombre de usuario y el correo.
Pulsando en la imagen de perfil también se puede acceder a la pantalla de perfil.
Desde este menú se puede acceder a la Home, el Perfil, Mis obstáculos, Ajustes y finalmente Cerrar sesión. El botón de cerrar sesión te devuelve al login.



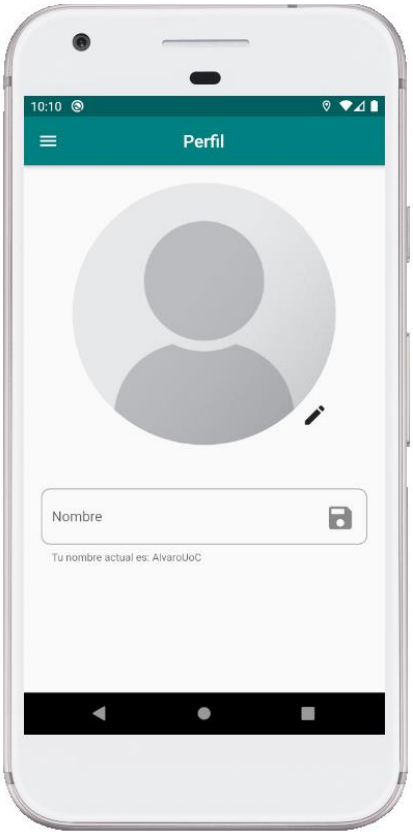
Menú lateral sin foto



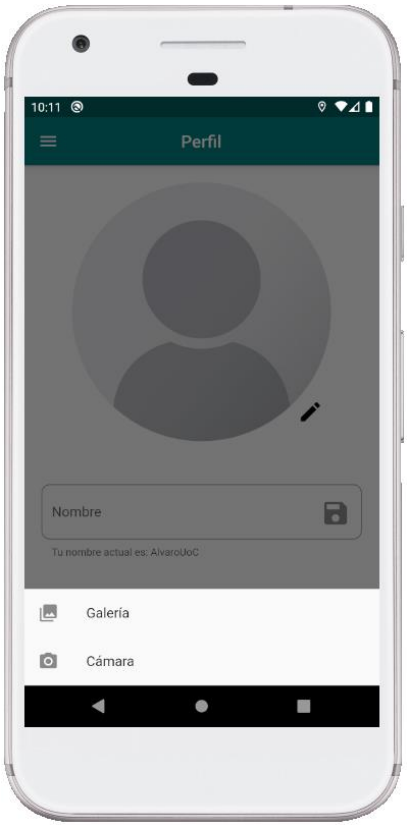
Menú lateral con foto y nombre actualizado

3.10.4 Perfil

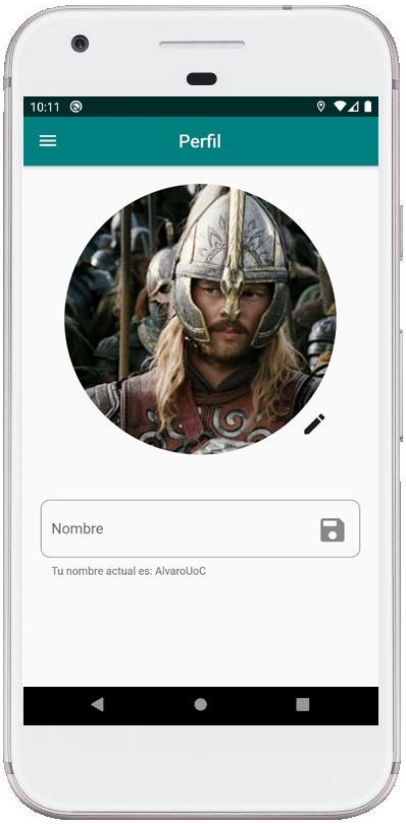
Dentro del perfil podemos actualizar tanto la imagen como el nombre de usuario. La imagen se actualiza automáticamente mientras que el nombre de usuario se debe guardar pulsando el icono de guardar.



Perfil antes de modificar.



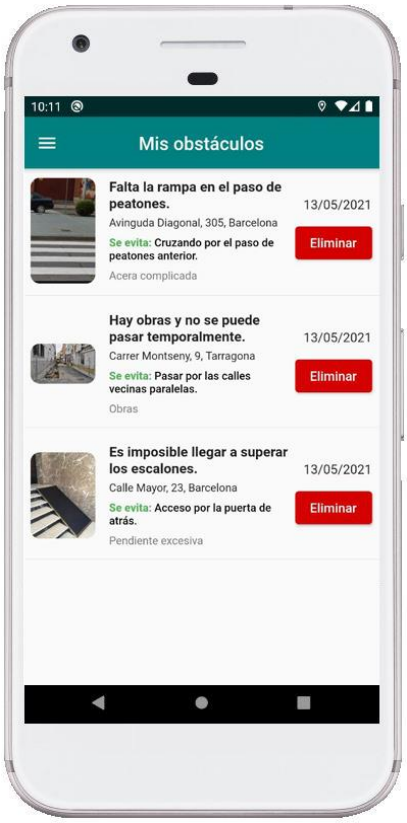
Picker para fuente de la foto.



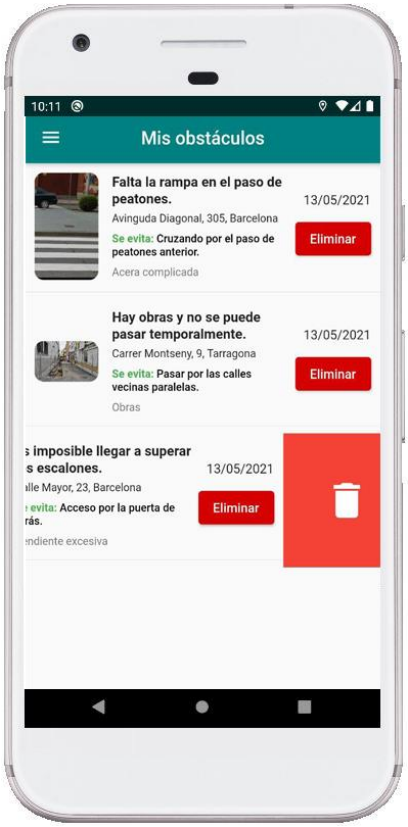
Perfil con foto cambiada.

3.10.5 Mis obstáculos

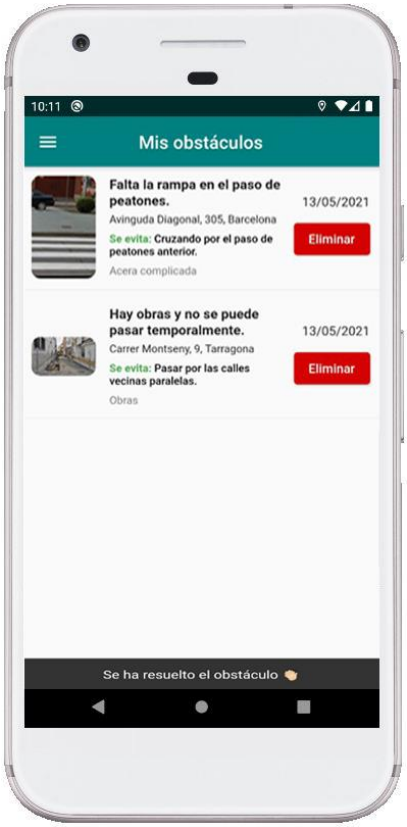
En la parte de obstáculos se pueden observar todos los obstáculos reportados por el usuario, si lo desea, puede eliminar alguno de ellos pulsando el botón de eliminar o haciendo swipe hacia la izquierda. Tras la eliminación se muestra un mensaje en la parte inferior.



Lista de obstáculos del usuario.



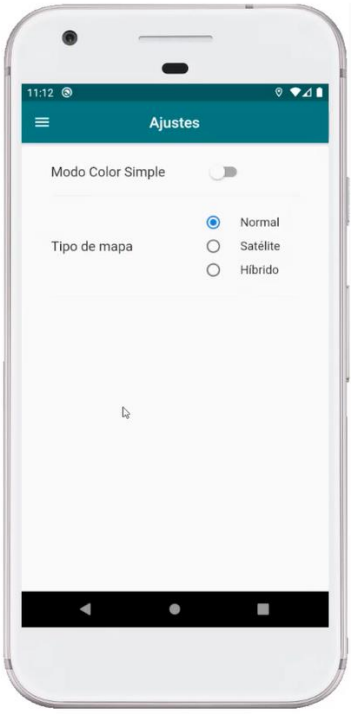
Eliminar obstáculo deslizando.



Mensaje de eliminación de obstáculo.

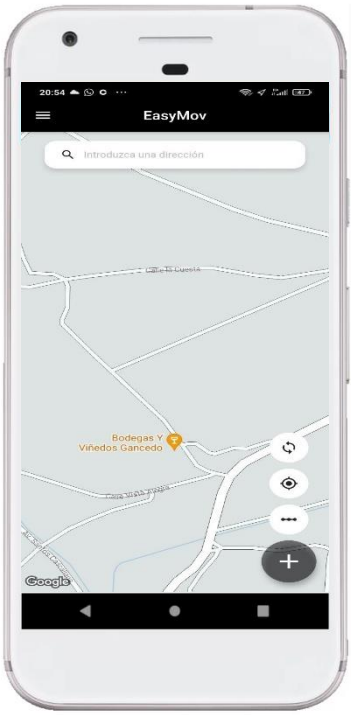
3.10.6 Ajustes

En esta pantalla se pueden modificar algunos ajustes de la aplicación, en este caso disponemos de dos ajustes personalizables. Se puede activar el modo de color simple, que modifica el color verde principal por color negro. Aunque requiere reinicio de aplicación para que tenga efecto. También se puede modificar el tipo de mapa, que esta configuración sí que se puede observar al momento.

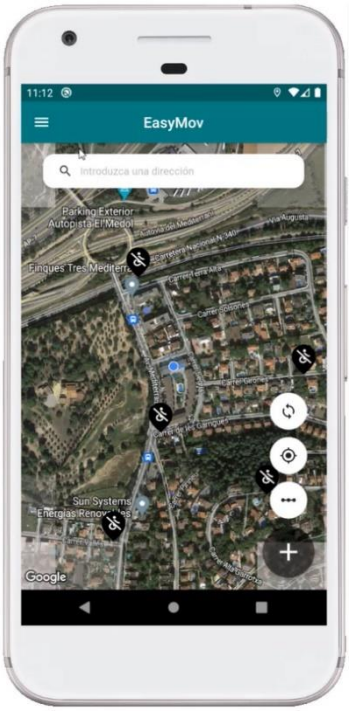


Pantalla de ajustes

Aviso al activar color simple



Modo color simple activo



Diferente tipo de mapa

4. Pruebas

No se han realizado pruebas sobre iOS ya que para ello se necesitaría usar un ordenador macOS, que no dispongo de él. Por eso solo se ha desarrollado el código para Android.
Para problar la aplicación se ha hecho mayormente sobre dispositivo virtual, un Google Pixel, pero posteriormente se ha probado la aplicación en un dispositivo real, un Xiaomi Mi9. También, se han realizado pruebas sobre un terminal con pantalla más pequeña, aunque para ello ha sido necesario utilizar un emulador de Nexus 5.

Para esta primera versión de las pruebas, se han realizado pruebas unitarias a través de el paquete test_flutter, que añade todas las funciones para realizar tests unitarios. En este caso hemos realizado tests unitarios a los obstáculos y su integración con Firebase.

A continuación detallamos el proceso de testing que se ha realizado con esta clase, de cara a además de esta clase se podrían realizar tests sobre los usuarios, sobre las ubicaciones en el mapa y sobre algún widget como por ejemplo la lista del conjunto de obstáculos, ya que también se pueden realizar tests sobre widgets de la aplicación.

Grupo de tests	Tests realizados	Descripción
1	Test de crear obstáculo en la APP.	En este test se crea una instancia del Obstáculo y se revisa que sus datos corresponden con los parámetros que se han pasado en la construcción.
	Test para validar atributos predefinidos.	En este test se revisa que los atributos que no se han pasado en la constructora se inicializan con unos valores predefinidos.
2	Test de subir obstáculo a Firebase.	En este test se revisa si la conexión con Firebase es correcta y se pueden subir Obstáculos, para ello se sube el obstáculo y se mira si devuelve true.
	Test de leer obstáculos de Firebase por email.	En este test se quiere validar que al leer los obstáculos de Firebase en función del email, solo devuelve el obstáculo que se ha creado y subido previamente.
	Test para validar que se leen los datos correctos de Firebase.	En este test se revisa que el obstáculo que se ha leído con el email, corresponde al obstáculo creado previamente, en este caso a través de la descripción del mismo.
	Test de eliminar obstáculos en Firebase.	En este test se revisa que la eliminación de obstáculos funciona correctamente, para ello se elimina el obstáculo y se vuelven a leer los datos, obteniendo 0 obstáculos por ese email.
	Test de leer todos los obstáculos de Firebase.	En este test se revisa que se pueden leer los obstáculos subidos por los demás usuarios a Firebase, para ello, se leen todos los obstáculos y se valida que se obtienen más de 0 resultados (en este caso 9)

Tabla 13. Conjunto de tests realizados.

4.1 Detalle de los tests.

```dart
import 'package:easymov_app/src/models/obstaculo_model.dart';
import 'package:easymov_app/src/providers/obstaculos_provider.dart';
import 'package:flutter_test/flutter_test.dart';

Run | Debug
void main() {
  //Usamos un email que no existe en la base de datos
  final email = "unit@test.com";

  //Creamos un obstáculo
  final obstaculo = ObstaculoModel(
    emailCreador: email,
    ubicacion: "Avinguda Diagonal, 20, 08019 Barcelona",
    descripcion: "Es una calle con mucha pendiente.",
    alternativa:
        "Se puede ir por la calle de al lado que no tiene tanta pendiente.",
    tipo: "Pendiente excesiva",
  );

  //Creamos el objeto que nos conectará con Firebase
  final _obstaculosProvider = new ObstaculosProvider();
```

Figura 39. Inicio tests

Primero de todo usaremos un email que no está registrado en nuestra base de datos, luego, crearemos un objeto de la clase Obstaculo y le definiremos unos valores iniciales. También crearemos un objeto provider que nos servirá para tener centralizadas las llamadas a firebase.

```dart
group('Tests para la creación de obstáculos en la app', () {
  test('El obstáculo se crea correctamente en la APP', () {
    expect(obstaculo.emailCreador, email);
    expect(obstaculo.descripcion, "Es una calle con mucha pendiente.");
  });

  test('Los atributos con valor predefinido se crean correctamente', () {
    expect(obstaculo.fechaCreacion, "12/05/2021");
    expect(obstaculo.numResoluciones, 0);
  });
});
```

Figura 40. Grupo 1 tests (app).

El primer grupo de tests corresponden a los valores de los obstáculos antes de llamar a Firebase, en ellos validamos que el objeto se ha creado correctamente con los valores

definidos, tanto los que hemos indicado en la construcción del objeto como los que no hemos indicado valor y por lo tanto se han creado con valores por defecto.

```
group('Tests de Firebase', () {
  test('El obstáculo se sube correctamente a Firebase', () async {
    bool resultado = await _obstaculosProvider.crearObstaculo(obstaculo);
    expect(resultado, true);
  });

  List<ObstaculoModel> obstaculos = [];

  test('Solo hay un obstaculo en Firebase con este email', () async {
    obstaculos = await _obstaculosProvider.cargarObstaculosPorEmail(email);
    expect(obstaculos.length, 1);
  });

  test('El obstáculo leído corresponde con el creado', () {
    expect(obstaculos[0].descripcion, obstaculo.descripcion);
  });

  test('El obstáculo de firebase se elimina correctamente', () async {
    String id = obstaculos[0].id;
    await _obstaculosProvider.borrarObstaculo(id); //Borramos el obstáculo
    obstaculos = await _obstaculosProvider
        .cargarObstaculosPorEmail(email); //Volvemos a cargarlos

    expect(obstaculos.length, 0);
  });

  test('Se pueden leer todos los obstáculos correctamente', () async {
    obstaculos = await _obstaculosProvider.cargarObstaculos();
    print("Obstáculos totales de la BBDD: ${obstaculos.length}");
    expect(obstaculos.length > 0, true);
  });
});
}
```

Figura 41. Grupo 2 de tests (Firebase).

El segundo grupo está compuesto por los tests de Firebase, en él validaremos que las funciones definidas en el provider funcionan correctamente. Validaremos la subida de obstáculos a la Base de Datos, la lectura de obstáculos por email, la eliminación de obstáculos (eliminando el obstáculo creado en este test) y por último, la lectura de todos los obstáculos de la BD.

4.2 Resultado de los tests

```
Alvaro@DESKTOP-DLO8N3D MINGW64 ~/Desktop/TFM/APP/easymov_app
$ flutter test test/test_obstaculos.dart
00:03 +6: Tests de Firebase Se pueden leer todos los obstáculos correctamente
Obstáculos totales de la BBDD: 9
00:03 +7: All tests passed!
```

Figura 42. Resultado de los tests.

Se ha pasado correctamente los 7 tests definidos para la clase Obstaculo, antes del último test se ha mostrado el resultado de los obstáculos totales que hay en la Base de Datos.
Por falta de tiempo no se ha decidido hacer tests de otras clases o widgets de la aplicación, pero las siguientes pruebas sería ampliar este conjunto de tests añadiendo las demás clases definidas en la aplicación o sino las más importantes y también hacer algún test sobre widget, como por ejemplo el Widget del mapa de Google, que tiene gran peso en la aplicación.

5. Conclusiones

Tras finalizar el proyecto, ha llegado el momento de analizar qué se ha aprendido y si la metodología que se ha seguido ha sido la correcta. Dividiré las conclusiones en cuatro grupos: Análisis de la planificación, conclusiones sobre la tecnología, las conclusiones personales y finalmente las oportunidades de mejora o líneas de trabajo futuras.

- **Planificación:** La planificación inicial se vio afectada por algunas desviaciones, en primer lugar **se modificó el alcance del proyecto** ya que inicialmente se planteó crear rutas evitando los obstáculos, pero en el momento de desarrollo era demasiado complejo para poder hacerlo y mantenerlo en el alcance podría suponer un riesgo para todo el proyecto, ya que supondría un bloqueo importante hasta encontrar (si se llegase a encontrar) la forma de hacerlo. El motivo es que las APIs que incluyen esa funcionalidad son de pago. Pese a todo, no ha supuesto un problema ya que se había reservado 40 horas para posibles desviaciones, entonces hemos podido alargar la implementación 27 horas más (9 días). Desarrollando con éxito las demás funcionalidades.
  Otra modificación ha sido que se **modificó la preparación de la defensa** ya que inicialmente lo había planteado para el día 21 de junio pero realmente se debía entregar el vídeo en la entrega actual, a día 9 de junio. También hemos podido destinar horas de reserva para poder avanzarlo y que finalice el día 9 de junio.

- **Tecnología:** Tras usar este framework durante cuatro meses, puedo concluir que *Flutter tiene un gran potencial y es realmente cómodo desarrollar con ello.* En cuanto a las APIs usadas, vemos que la API de Google Maps no es una solución para que particulares realicen proyectos mínimamente complejos, ya que dejando de lado las funcionalidades más básicas, enseguida se añaden costes que impiden que los particulares puedan usarlo. Para ello, el combinar las funcionalidades básicas de Google Maps con las APIs de Mapbox, permiten desarrollar soluciones más complejas sin tener costes asociados.

- **Personales:** Flutter me ha parecido sumamente interesante, ya que mi conocimiento previo sobre desarrollo de aplicaciones móviles era muy limitado y este proyecto me ha servido para aprender sobre el framework y poder descubrir gran parte de sus características, como el Hot Reload, que agiliza el proceso de desarrollo al poder ver los cambios en tiempo real sobre el emulador. Gracias al proyecto he podido conocer más a fondo cómo funcionan las aplicaciones móviles y a usar herramientas de desarrollo de estas que me serán útiles para un futuro.

También, gracias a este proyecto, he aprendido a llevar a cabo la actividad de planificación de un proyecto de software íntegramente, también a trabajar con roles diferentes y luego combinar los resultados, y a gestionar un proyecto en temas de planificación, plazos, presentaciones orales y documentación.

- **Oportunidades de mejora:**
    - **Crear rutas que eviten puntos:** Sería realmente interesante poder crear rutas entre dos ubicaciones que eviten los puntos negros que puedan haber por el camino, de esta forma el usuario podría seguir la ruta sin tener que entrar a la información de los obstáculos para saber cómo evitarlo.

    - **Guardar rutas habituales:** Aprovechando el seguimiento del usuario al tener la aplicación abierta, se podría desarrollar la funcionalidad para guardar rutas y que posteriormente pudiesen ser pintadas en el mapa.

    - **Añadir notificaciones:** Se podría añadir notificaciones a la app para informar, por ejemplo, cuando un obstáculo de tus rutas habituales ha sido resuelto o cuando otro usuario nos ha resuelto un obstáculo de los que han sido reportados por mí.

    - **Añadir diferentes idiomas:** Para que la aplicación pueda ser usada por usuarios de diferentes países y no solo de España o países de habla hispana.

    - **Añadir funciones de descripción por voz:** Ya que hay usuarios que además de problema de movilidad pueden tener problemas de visión, por lo que al añadir descripción por voz se podría llegar a más usuarios.

6. Glosario

- App = Término usado para referirse a las aplicaciones de dispositivos móviles.
- API = Interfaz de programación de aplicaciones (Application Programming Interface) o el conjunto de herramientas que ofrece una biblioteca para ser usada como una capa de abstracción en otro software.
- Barrera arquitectónica = Obstáculos que impiden o dificultan a las personas moverse o acceder a un espacio.
- Dart = Lenguaje de programación de Google orientado a objetos que utiliza Flutter.
- Feedback = Acción de ofrecer información a una persona sobre un resultado.
- Firebase = Plataforma de desarrollo de aplicaciones en la nube, propiedad de Google.
- Flutter = Conjunto de herramientas (Framework) de Google para crear Apps.
- Framework = Estructura conceptual que facilita el desarrollo y organización de otro proyecto de software. En español se llama marco de desarrollo.
- IDE = Entorno de Desarrollo Integrado (Integrated Development Environment).
- Librería = Conjunto de código externo que puede aprovecharse para conseguir determinados comportamientos.
- Marcador = Elemento visual que permite identificar una localización en el mapa.
- Modelo = Definición de características y método que tendrán un tipo de objeto.
- SDK = Kit de desarrollo de software (Software Development Kit), conjunto de herramientas necesarias para desarrollar el código de una aplicación.
- TFM = Trabajo Final de Máster.
- UML = Lenguaje Unificado de Modelado (Unified Modeling Language) es un estándar internacional que sirve para especificar o para describir procesos o sistemas.
- UI = Interfaz de Usuario (User Interface)
- Widget = Bloque de código reusable que se usa en Flutter para elementos de UI.

7. Bibliografía

[1] E. Russell, “9 things to know about Google’s maps data: Beyond the Map | Google Cloud Blog,” 2019. https://cloud.google.com/blog/products/maps-platform/9-things-know-about-googles-maps-data-beyond-map (accessed Jun. 01, 2021).

[2] Instituto Nacional de Estadística, “Panorámica de la discapacidad en España,” *Boletín Inf. del Inst. Nac. Estadística*, vol. 10, pp. 1–12, 2009, Accessed: Jun. 02, 2021. [Online]. Available: www.ine.es.

[3] K. Beck *et al.*, “Manifesto for Agile Software Development,” *The Agile Alliance*, 2001. http://agilemanifesto.org/ (accessed Jun. 01, 2021).

[4] H. Takeuchi and I. Nonaka, “The new new product development game,” *J. Prod. Innov. Manag.*, vol. 3, no. 3, pp. 205–206, 1986, doi: 10.1016/0737-6782(86)90053-6.

[5] B. Penzenstadler, “Towards a definition of sustainability in and for software engineering,” *Proc. ACM Symp. Appl. Comput.*, no. May, pp. 1183–1185, 2013, doi: 10.1145/2480362.2480585.

[6] T. Sneath, “Google Developers Blog: Flutter 1.0: Google’s Portable UI Toolkit,” 2018. https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html (accessed Jun. 02, 2021).

[7] L. Kasper and B. Lars, *Web Languages and VMs: Fast Code is Always in Fashion. (V8, Dart) - Google I/O 2013*. 2013.

[8] G. Bracha, “The Dart Programming Language - Gilad Bracha - Google Books,” 2016. https://books.google.es/books?hl=es&lr=&id=UHAlCwAAQBAJ&oi=fnd&pg=PT13&dq=dart+language&ots=Pp-MhZDdCL&sig=xKPQYCAwZSG8gjP8WZNfbQpFeZI#v=onepage&q=dart language&f=false (accessed Jun. 02, 2021).

8. Anexos

Manual de usuario

No se ha considerado oportuno crear un manual de usuario con las funcionalidades de la aplicación ya que todas las pantallas de la aplicación se han definido en la sección *3.10 Funcionalidades.*
Para ejecutar la aplicación simplemente hace falta instalar el archivo .apk adjunto con este proyecto y conceder permisos de GPS.

En caso de querer ejecutar el código fuente, se debe instalar flutter desde su sitio web: https://flutter.dev y si se desea usar Visual Studio Code, se debe instalar el plugin de Dart y Flutter, además de instalar el SDK de Android con un emulador.
En caso de no querer instalar un emulador, se puede usar un dispositivo físico para las pruebas, pero para ello se debe activar la depuración por USB en las opciones de desarrollador del teléfono y conectarlo por usb.
Antes de lanzar la aplicación, se debe ejecutar en consola *flutter pub get* en el directorio del proyecto para instalar el conjunto de paquetes de la aplicación y sus dependencias. Y posteriormente *flutter run* para lanzar la aplicación.