

Trabajo de fin de grado

Aplicación móvil y backend para gestión de rutas de senderismo

Estudiante: Hernández Hernández, Nauzet Aduen

Tutor: Hernández Figueroa, Zenón



UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática



SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Nauzet Aduen Hernández Hernández, autor del Trabajo de Fin de Título Aplicación móvil y backend para gestión de rutas de senderismo, correspondiente a la titulación Grado en Ingeniería Informática.

SOLICITA

que se inicie el procedimiento de defensa de este, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

☐ Se ha iniciado o hay intención de iniciarlo (defensa no pública).

☒ No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 12 de julio de 2019.

El estudiante

HERNANDEZ
HERNANDEZ NAUZET
ADUEN - 78521856V

Fdo.: _____


A rellenar y firmar **obligatoriamente** por el/los tutor/es, en relación a la presente solicitud, se informa:

☒ Positivamente

☐ Negativamente

Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Universidad de Las Palmas de Gran Canaria			
Página 1 / 1	ID. Documento	Mw4TW7Iba3XXh4ZClxoWvA\$\$	
Este documento ha sido firmado electrónicamente por		Fecha de firma	
ZENÓN JOSÉ HERNÁNDEZ FIGUEROA		12/07/2019 08:47:22	

Contenido

Resumen	1
Introducción.....	2
Contexto.....	2
Uso del móvil.....	2
Uso de aplicaciones móviles vs navegadores	3
Aplicación multiplataforma	4
Frameworks multiplataforma	5
Aplicaciones y mapas	7
Objetivos	8
Objetivos funcionales.....	8
Objetivos personales	9
Objetivos extra.....	9
Metodología	10
Introducción.....	10
Scrum en uso.....	10
Herramientas.....	11
Competencias específicas cubiertas.....	14
Competencias adquiridas	15
Aportaciones al entorno	16
Estructura de la memoria	16
Análisis previo	18
Estado del arte	18
AllTrails	18

Maps3D.....	19
Spyglass.....	21
MapMyHike.....	22
Wikiloc.....	22
Requisitos	24
Requisitos funcionales.....	24
Requisitos no funcionales.....	25
Mockups.....	26
Paleta de colores	27
Modelo de negocio.....	28
Publicidad, In-app advertising	28
Modelo Freemium	29
Porcentajes	30
Normativa y legislación	30
Ley Orgánica de Protección de Datos	30
Uso que da el usuario	31
Persistencia de datos.....	31
Desarrollo	32
Alcance de la implementación.....	32
Interfaz	32
Google+ vs otros	32
Autenticación.....	33
Provider.....	33

Serialización.....	33
Componentes reactivos	35
Filtros	36
Ingeniería del software	37
Casos de uso	37
Diagramas de clase	39
Modelo de la base de datos	46
Arquitectura	46
Vanilla	47
Scoped Model y Redux	47
Provider.....	48
Provider y Streams	49
Streams	50
Tests.....	50
Tecnologías	52
Flutter	52
(García, https://guillermogarcia.es , s.f.).....	54
Cloud Firestore	54
Acceso y despliegue	55
Conclusiones	56
Trabajo futuro	56
Referencias	58
Anexos	59
Manual de usuario	59

Sesión	59
Rutas.....	61
Mapa de rutas	64
Eventos	64
Filtros	65

Índice de figuras

Ilustración 1 Reparto de tráfico por dispositivo	2
Ilustración 2 Participación navegador/apps	3
Ilustración 3 GoogleMaps vs otros	8
Ilustración 4 Ejemplo de correo electrónico del Servicio de Deportes.....	9
Ilustración 5 VsCode extensions.....	12
Ilustración 6 Usuarios en el backend	12
Ilustración 7 Protitipado en Figma	13
Ilustración 8 Repositorio github	14
Ilustración 9 Ejemplo formato GPX	20
Ilustración 10 Mockups 1	26
Ilustración 11 Mockups 2	27
Ilustración 12 Interfaz de usuario en Dribbble.....	28
Ilustración 13 Preciso "In-App Ad Targeting" (https://www.smaato.com/guide-to-in-app-advertising).....	29
Ilustración 14 Evento universitario de senderismo	30
Ilustración 15 jsonGeneration error	34
Ilustración 16 Método setToMap en MarkerHelper.....	34
Ilustración 17 Clase TrailListPage con Streambuilder.....	35
Ilustración 18 Ejemplo de filtro	36
Ilustración 19 setState	¡Error! Marcador no definido.
Ilustración 20 Casos de uso de sesión	37
Ilustración 21 Casos de uso Rutas y Eventos	38

Ilustración 22 Caso de uso de filtrado.....	39
Ilustración 23 Diagramas de modelo.....	39
Ilustración 24 Diagramas de componentes.....	40
Ilustración 25 Vistas de acceso.....	41
Ilustración 26 StatefulWidget.....	41
Ilustración 27 Vistas creacionales.....	42
Ilustración 28 Vistas detalladas.....	42
Ilustración 29 Vista de errores.....	43
Ilustración 30 Estilos.....	44
Ilustración 31 Utilidades.....	44
Ilustración 32 Diagramas con las vistas principales.....	45
Ilustración 33 Modelo de la base de datos.....	46
Ilustración 34 Ejemplo de ScopedModel.....	47
Ilustración 35 Ejemplo de Redux.....	48
Ilustración 36 Provider y Streams.....	49
Ilustración 37 Ejemplo de ejecución de tests.....	50
Ilustración 38 Widget Test.....	51
Ilustración 39 Flutter Framework y Engine.....	53

Resumen

Este trabajo de fin de grado plantea una aplicación móvil multiplataforma que permita a sus usuarios administrar rutas de senderismo, posibilitando, a su vez, administrar eventos sobre dichas rutas. A su vez, se ha implementado un backend funcional en usando la plataforma Cloud Firestore.

El proyecto ha sido realizado en Flutter, un framework para desarrollo de aplicaciones móviles.

This final degree Project propose a cross-platform mobile app which allow users to manage hiking trails, enabling, also, to manage events in those trails. We also implemented a working backend with the Cloud Firestore platform.

This Project has been developed with Flutter, a framework to create mobile apps.

Introducción

Contexto

Nos encontramos en una sociedad en la que el uso del móvil es ya innato, parte de nuestra vida, una necesidad. Según el (Informe Mobile en España y en el Mundo, 2018),

“En 2018, el número de usuarios de móviles en el mundo asciende a 5.135 mil millones, lo que significa que el 68% de la población mundial ya cuenta con un móvil. El mundo es eminentemente móvil y la tendencia continúa.”

Ilustración 1 Reparto de tráfico por dispositivo

Uso del móvil

Reparto del tráfico web por dispositivo

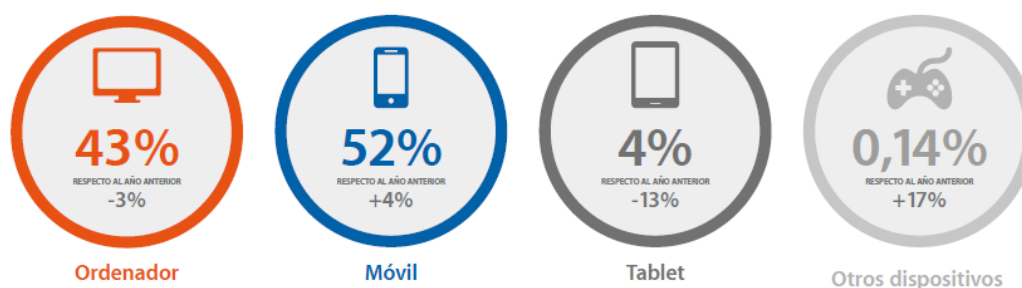


Gráfico elaborado por ditrendia a partir de datos de We are Social

ditrendia
digital marketing trends

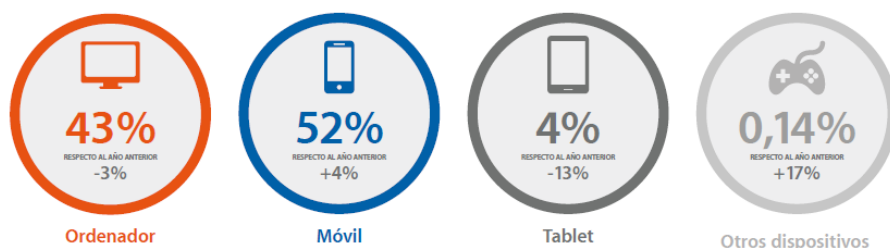


Gráfico elaborado por ditrendia a partir de datos de We are Social

ditrendia
digital marketing trends

En este mismo informe, se destaca que el uso del móvil para visitar páginas web sigue creciendo frente a otros dispositivos, como el ordenador o una tableta.

Uso de aplicaciones móviles vs navegadores

Si nos fijamos en el uso que le dan los usuarios a las aplicaciones, encontramos que el 90% del tiempo los usuarios se lo dedican a las aplicaciones y solo un 10% se lo dedican al navegador.

Participación de minutos móviles totales por navegador/aplicación

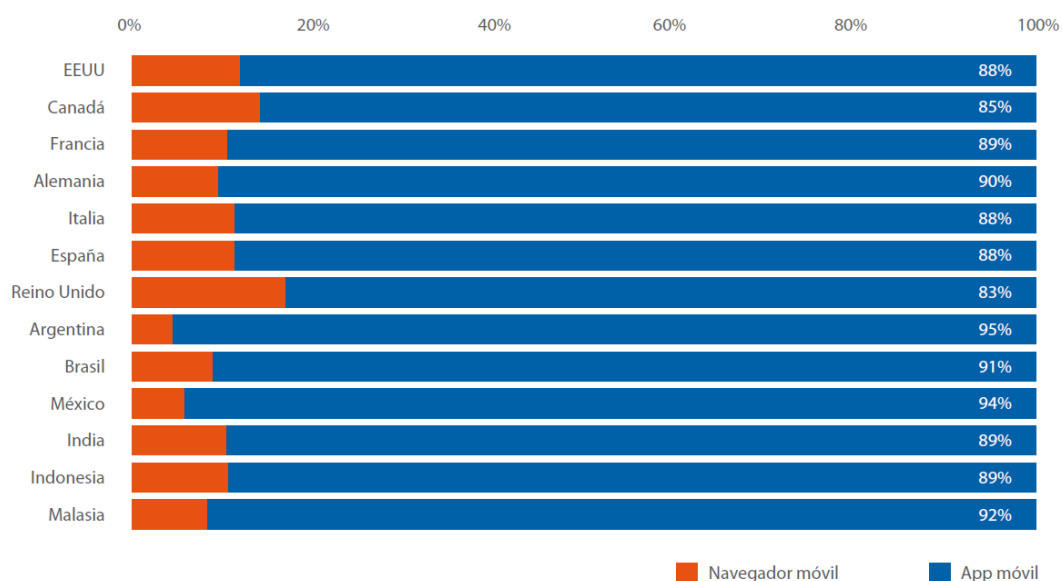


Gráfico elaborado por ditrendia a partir de datos de Comscore

Ilustración 2 Participación navegador/apps

ditrendia
digital marketing trends

Un dato interesante de este informe, es que los usuarios españoles mayoritariamente descargan aplicaciones gratuitas, aunque la descargas de aplicaciones de pago aumentará en los próximos años.

“Cada día se descargan en España 4 millones de aplicaciones y el 82% de los españoles solo descarga aplicaciones gratuitas.”

“Gastaremos más en Apps de pago: En 2018 se espera que los usuarios se gasten un 30% más en aplicaciones de nivel mundial de lo que lo hicieron en 2017, superando los 110 miles de millones de dólares a nivel global.”

Aplicación multiplataforma

A la hora de desarrollar aplicaciones móviles, nos encontramos con dos formas muy diferenciadas: desarrollo nativo y desarrollo multiplataforma.

El desarrollo nativo se centra en implementar una aplicación específica para una plataforma concreta, como son Android e iOS. Esta aplicación solo funcionaría en dicho sistema operativo, por lo que para abarcar un porcentaje mayor de usuarios, sería necesario realizar una aplicación nativa por plataforma que se quiera usar. Por el contrario, una única aplicación multiplataforma, funcionará para diferentes plataformas.

Estos dos tipos de desarrollo tienen algunas diferencias importantes que no se centran en la plataforma:

- En el caso de querer abarcar Android e iOS, necesitamos dos aplicaciones diferentes que hagan lo mismo, es decir, dos códigos fuentes diferentes para desarrollo nativo, mientras que en multiplataforma, tendríamos un solo código base.
- Necesitamos dos equipos diferentes de trabajo, ya que exigir a un solo equipo que domine dos tecnologías diferentes, es un objetivo irrealista, mientras que en multiplataforma, con un solo equipo tenemos.
- Inconsistencia entre iOS y Android, ya que cada uno tiene sus propios componentes que funcionan de manera diferente, pudiendo unificarlos en desarrollo multiplataforma.
- Funcionalidades limitadas, no integradas en los frameworks multiplataforma, dependiendo de plugins externos también llamados “third-parties”, haciendo que el desarrollador deba esperar por dichos plugins o realizarlos él mismo. Por el contrario, en desarrollo nativo, todas las herramientas y funcionalidades vienen por defecto.

- El rendimiento de las aplicaciones multiplataforma suele ser peor que las aplicaciones nativas, debido a las traducciones que se suelen realizar en estos frameworks para simular o copiar elementos nativos, **aunque no ocurre esto en todos los casos.**
- Las diferentes experiencias de usuario se pueden ver empeoradas al intentar centralizar o unificar una aplicación, debido a que los usuarios se acostumbran a la plataforma del móvil que disponen. Con un framework multiplataforma, la aplicación tiende a ser neutra, haciendo difícil la utilización de la aplicación, cuando un usuario está muy acostumbrado a dicha plataforma.

Frameworks multiplataforma

Aquí se plantea una lista con los frameworks multiplataforma más importantes del mercado actual, entre ellos Flutter, el elegido para realizar nuestro trabajo de fin de grado. La información ha sido sacada del blog [netsolutions](https://netsolutions.es).

Xamarin, lanzado en 2011 de forma independiente, comprado por Microsoft en 2016, es un framework de código libre que surgió para resolver los problemas del desarrollo nativo explicados anteriormente. La mayor ventaja que aporta es que permite acceder a las APIs nativas de forma directa, cosa que muchos otros framework



no lo permiten. Los expertos lo suelen calificar como el más competitivo de los existentes, pero no es gratuito, limitando entre otras cosas el uso de ciertas librerías, lo que hace que no sea muy elegido por pequeñas empresas o por startups.

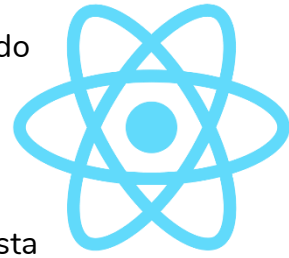
React Native, lanzado por Facebook en 2015, es quizá el más

popular por su utilización de tecnologías web, permitiendo

programar en JavaScript. A diferencia de Xamarin, este

es gratuito, lo que acentúa su popularidad en pequeñas

empresas con equipos de desarrollo web. Al usar esta



tecnología, tiene multitud de componentes listos para ser usados, lo que hace

más rápido su implementación. Su principal problema es el rendimiento, nada

comparable con Xamarin o Flutter, ya que lo que hace React Native es

transformar sus componentes en componentes nativos mediante “bridges” o

puentes, ralentizando las aplicaciones.

Flutter, lanzado por Google en 2017, es un framework libre,

basado en Dart, un lenguaje orientado a objetos creado

también por Google. Permite un desarrollo muy rápido de

las aplicaciones gracias a multitud de librerías y

herramientas. Una de sus mejores características es que el



código resultante es “ahead of time”, lo que hace que las aplicaciones estén

“precompiladas” antes de ser utilizadas, mejorando los tiempos de carga

enormemente, haciendo que el rendimiento de las aplicaciones se pueda

equiparar al código nativo, mientras que muchas son “Just in time”, compiladas

en tiempo de ejecución, como es el caso de Java y la máquina virtual en Android.

Este framework carece de la madurez necesaria para competir con otros, pero

está creciendo a pasos agigantados. Una de las razones por las que se debería

apostar por Flutter es Fuchsia, un nuevo sistema operativo diseñado por Google

para sustituir a Android, donde la programación se realizará en Flutter y Dart.

Adobe PhoneGap / Apache Cordova (versión inicial, de código abierto), es un framework muy sencillo que usa HTML5, CSS y JavaScript para desarrollar, lo que hace que tenga una fácil barrera de entrada. Fue creado en 2011, siendo muy popular en el pasado, perdiendo peso respecto a otros frameworks en



la actualidad. Tiene multitud de herramientas externas que lo hacen muy atractivo, dándole mucha variedad. Sus mayores desventajas son que no soporta muchas funcionalidades nativas y como dijimos de React Native, al ser web, su rendimiento no es óptimo.

ionic, lanzado en 2013, es otro framework que utiliza tecnologías web para el desarrollo de aplicaciones. Tiene mayor cantidad de componentes ya creados para usarse que otros como Phonegap, multitud de librerías y la capacidad de combinarse con otros frameworks como AngularJS



(framework para desarrollo web), haciendo que muchos equipos web lo prefieran respecto a otros. Ionic presenta como mayor ventaja mejor rendimiento entre los que usan tecnologías web y como mayor inconveniente que se ha hecho muy dependiente de frameworks externos como el anteriormente dicho AngularJS.

Aplicaciones y mapas

La principal característica de un dispositivo móvil, como su nombre indica, es la movilidad. Por ello, queremos destacar la accesibilidad y el uso de mapas en aplicaciones móviles. En un artículo de Riley Panko para TheTerminal:

“El 77% de las personas con smartphones usan aplicaciones con mapas regularmente.” (Panko, s.f.)

Antes de usar el plugin de Google Maps para nuestro proyecto, dudábamos si elegirlo frente a otros plugins más desarrollados y maduros. Pensamos que usar un plugin que usara el servicio de Google Maps aportaría mayor calidad a nuestros mapas. Como vemos en el mismo artículo, los usuarios prefieren Google Maps frente a otros, lo que nos confirma que la elección es la correcta.

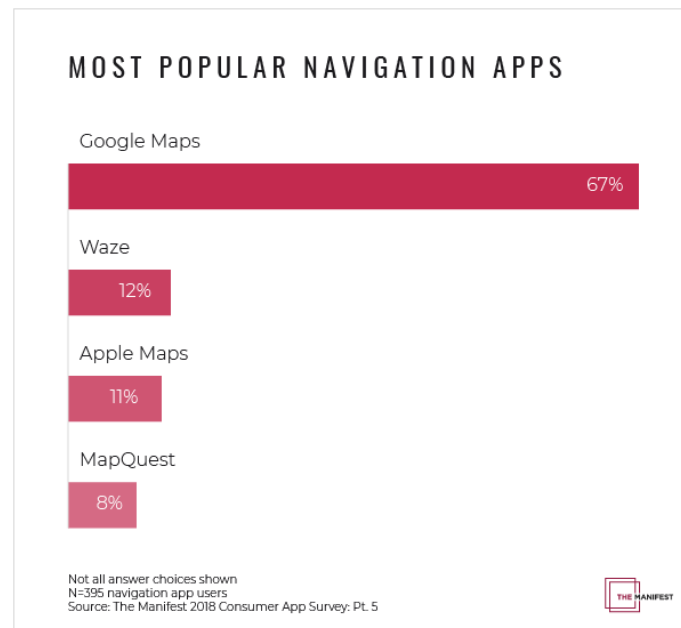


Ilustración 3 GoogleMaps vs otros

Objetivos

Dentro de los objetivos que teníamos para desarrollar este proyecto, encontramos dos categorías, los objetivos personales y los objetivos funcionales.

Objetivos funcionales

Como objetivo funcional del proyecto de fin de grado, propusimos la creación de una aplicación funcional de senderismo que permita a usuarios de distintos niveles gestionar y compartir sus rutas.

Objetivos personales

El principal objetivo personal era aprender a desarrollar con el framework Flutter, un framework moderno para el desarrollo de aplicaciones móviles multiplataforma, más concretamente Android e iOS. Para ello, nos marcamos como objetivo personal indirecto el aprendizaje de Dart como lenguaje de programación, necesario para desarrollar en Flutter.

Otro objetivo personal fue aprender programación reactiva, o “reactive programming”, una de las razones por las que elegimos Flutter como framework de desarrollo. En este tipo de paradigma, las aplicaciones se actualizan “solas”, teniendo en todo momento los datos reales de nuestra base de datos.

Objetivos extra

Durante las primeras semanas, mientras dábamos forma al prototipo de la aplicación, leyendo los correos universitarios desde el servicio de deportes de la universidad, se nos ocurrió poder gestionar eventos de ese estilo.

De esta forma, se decidió contactar con dicho servicio universitario, pero después de múltiples intentos fallidos, no recibimos ninguna respuesta..



Ilustración 4 Ejemplo de correo electrónico del Servicio de Deportes

A su vez, mejoraría el servicio de senderismo, que actualmente se maneja mediante correos electrónicos y una [web](#), que no maneja usuarios ni rutas, cosa que sí haría nuestro trabajo de fin de grado.

Metodología

Introducción

La metodología seguida fue parecida a Scrum, una metodología de desarrollo ágil, aunque no se siguió al pie de la letra por diferentes razones.

- Scrum destaca por las buenas prácticas para trabajar colaborativamente, pero el proyecto es de realización personal, lo que dificulta su adaptación.
- Al ser un lenguaje (Dart) y un framework (Flutter) desconocidos, hubo que dedicar mucho tiempo al aprendizaje y exploración de posibilidades. La inexperiencia en el ámbito del desarrollo de aplicaciones móviles acentuó la inversión de tiempo en aprendizaje de conocimientos nuevos.
- A pesar de cumplir con los objetivos marcados, sabíamos de ante mano que muchas características habría que investigarlas y tomar decisiones sobre si incluirlas o no, debido al carácter desconocido anteriormente explicado.

Scrum en uso

Una de las características más importantes de scrum son las entregas parciales, generando un mejor control sobre el producto final por parte del cliente. Generalmente estas entregas se realizan en iteraciones que duran varias semanas, denominada *sprint*. Antes de realizar un sprint, el equipo se reúne y marca las pautas y los objetivos de dicho sprint, limitando enormemente las sorpresas o desviaciones que puedan surgir. En estas reuniones, el equipo debe seleccionar los requisitos necesarios a realizar en el sprint como planificar la iteración con una lista de tareas basada en dichos requisitos. Al finalizar el sprint, hay que reunirse y revisar lo realizado, valorando los resultados. A su vez, hay que analizar cómo ha ido el sprint, para intentar mejorar de manera continua el trabajo a realizar.

En nuestro contexto, los “sprints” duraban una semana y en algunas ocasiones dos, realizando dichas reuniones con el tutor. En estas reuniones constantes, se mostraba el trabajo realizado, se hablaba de los cambios necesarios, de los puntos positivos, de los negativos y de los pasos a seguir en el siguiente “sprint”. Aunque no seguíamos a rajatabla la definición de scrum, lo adaptamos a nuestras necesidades.

En scrum el producto debe ser una aplicación funcional, por lo que en cada reunión mostrábamos la aplicación funcionando y demostrábamos su correcto funcionamiento. La mayoría de las veces usábamos el emulador de Android para hacer demostraciones en directo y en alguna ocasión construíamos la “apk” o **Android Application Package**, que es el formato de las aplicaciones Android, y se la enviábamos al tutor.

En nuestra pila de producto, lista con todas las tareas a implementar, declaramos todas las funcionalidades básicas especificadas en el TF01. Esta pila tiende a cambiar mucho durante el proceso de desarrollo, como fue nuestro caso, a medida que surgían nuevas funcionalidades o propuestas.

Herramientas

Para la realización del proyecto, como hemos dicho anteriormente, hemos usado **Flutter**. Al ser código abierto, Flutter no te obliga a usar ningún IDE (Integrated Development Enviroment) concreto, pudiéndose usar Android Studio, IntelliJ o **Vistual Studio Code**, por ejemplo. Hemos elegido este último por su ligereza frente a otros IDE's, haciendo la programación y el desarrollo más agradables.

Además tiene una gran cantidad de plugins o “extensions” que mejoran enormemente el entorno.

Usamos **Cloud Firestore** como backend de nuestra aplicación. Cloud Firestore es

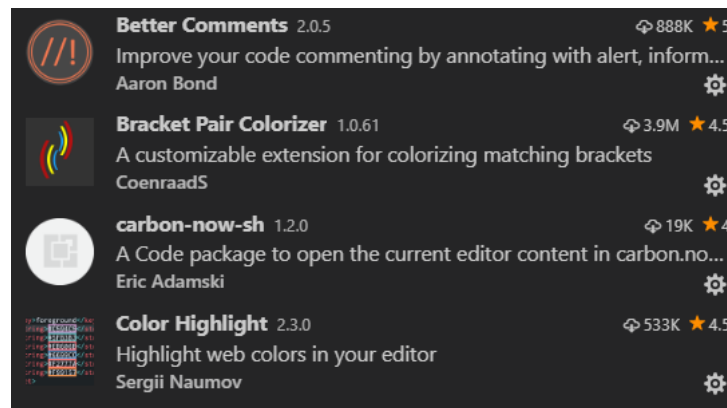


Ilustración 5 VsCode extensions

una base de datos NoSQL flexible y escalable para la programación en servidores, dispositivos móviles y la Web desde Firebase y Google Cloud Platform. Al igual que Firebase Realtime Database, mantiene tus datos sincronizados entre apps cliente a través de **agentes de escucha en tiempo real**, principal razón por la que la elegimos, haciendo la app reactiva a cambios.

Identificador	Proveedores	Creado	Accediste a tu cuenta	UID de usuario ↑
prueba@prueba.com	✉	31 may. 2019	20 jun. 2019	1gELdpe7xtNqIPSqxmXCw3t3Yof1
parapa@parapa.com	✉	4 jun. 2019	4 jun. 2019	H5MCwTn5xNbix22ckwiz5qdg4gd2
nauzet.adien@gmail.com	🌐	30 may. 2019	20 jun. 2019	v8QT3Pn3bLbLVgNIduS5GLDPNp...
nauzet.hernandez102@alu.ul...	✉	31 may. 2019		xY92Nf6AHKgpFCyRtNE71c4iit2
abc@abc.com	✉	2 jun. 2019	2 jun. 2019	z8NCqq25rvNVXUhqN61SBJD1e7...

Ilustración 6 Usuarios en el backend

Figma fue nuestra herramienta elegida para prototipar la aplicación. Funciona tanto en web como en escritorio, haciéndola muy simple para compartir los diseños.

Trello, fue nuestra herramienta para organizar el proyecto, permitiendo tener tareas y sus diferentes estados en diferentes listas, de la forma que lo hace scrum, como son por ejemplo “pendientes”, “en progreso” y “terminadas”, dentro de un sprint. Esta herramienta se usa en multitud de asignaturas, por lo que su elección fue instantánea.

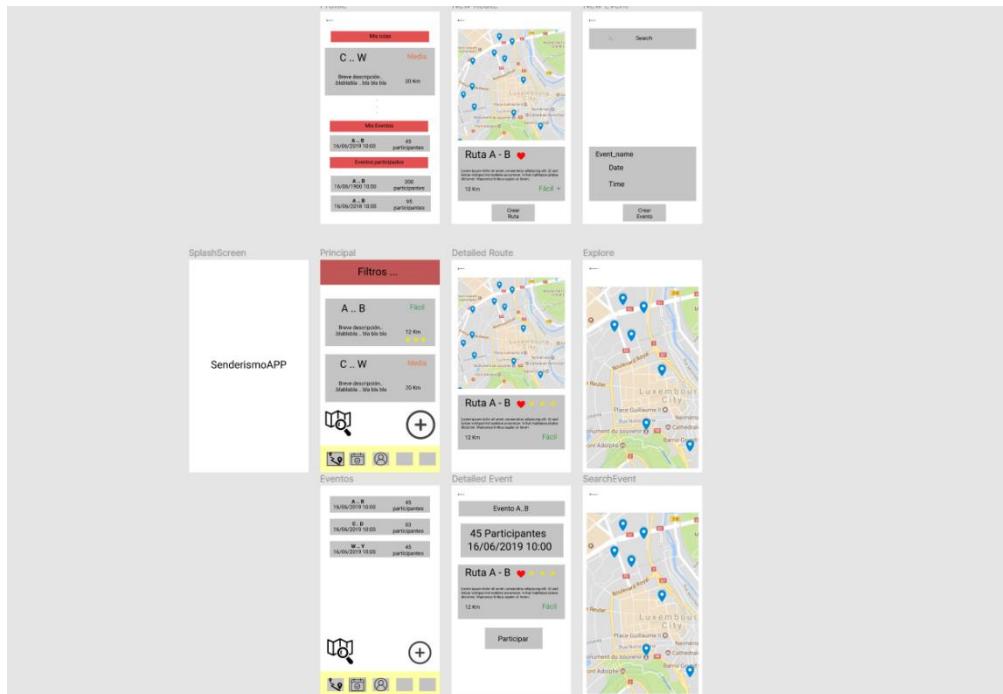


Ilustración 7 Protitipado en Figma

Usamos **Github** como repositorio, sirviéndonos para desarrollar sin miedo a cambios en el código. Github nos permite guardar el código en forma de “commits” o pequeños cambios que van actualizando el repositorio. A su vez, nos permite crear ramas para funcionalidades, generando un mayor control de lo que hace nuestra aplicación en cada etapa.

Nuestro proyecto se encuentra alojado en un [repositorio privado](#), ya que usamos claves de Google maps y ficheros de configuración que no deben ser públicos como el Google-services.json para conectarnos al backend.

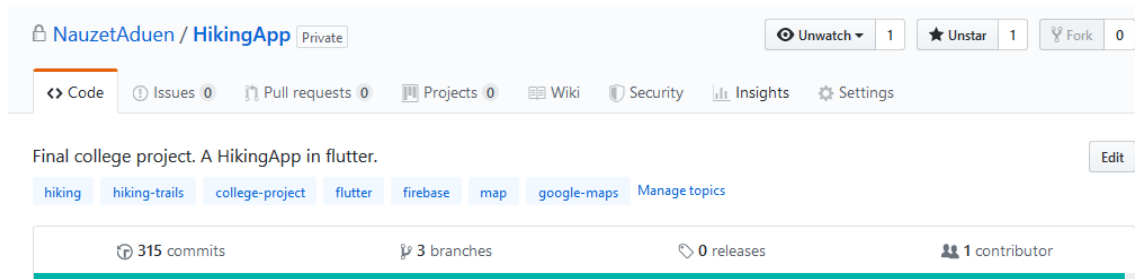


Ilustración 8 Repositorio github

Competencias específicas cubiertas

IS01

“Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software. “

IS02

“Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones. “

IS03

“Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles. “

IS04

“Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales. “

IS05

“Capacidad de identificar, evaluar y gestionar los riesgos potenciales asociados que pudieran presentarse. “

Competencias adquiridas

Nuestra aplicación ha cumplido con todos los requisitos funcionales propuestos, aplicando estándares de calidad del software, usando patrones de diseño actuales y paradigmas actuales de alta complejidad, para obtener un resultado satisfactorio.

Hemos dado solución a un problema real, reduciendo el coste relativo a realizar dos aplicaciones, a solo una.

Hemos trabajado en todos los aspectos posibles dentro del desarrollo de una aplicación móvil. Generalmente, se dividen en frontend y backend, el frontend es la aplicación móvil en sí, incluyendo la interfaz y las funcionalidades básicas; el backend, por el contrario, se refiere al diseño y desarrollo del lado del servidor. Normalmente, los equipos se dividen en dos grupos ya que la tecnología y el desarrollo son muy diferentes, casi polos opuestos. Cuando un desarrollador implementa ambas posibilidades, se le llama “Full-stack”, siendo muy poco frecuente en el mercado actual.

Aportaciones al entorno

Un proyecto como el nuestro tiene un evidente carácter social que puede ayudar en muchos ámbitos diferentes. Podría, por ejemplo, mejorar el turismo en la isla, incrementando el valor entre los amantes del senderismo, o incluso entre los más noveles, que nunca hayan realizado actividades como esta. Nuestra aplicación permite diferenciar entre rutas más complejas y rutas para principiantes.

A su vez, nuestra aplicación podría generar un mayor y mejor servicio de rutas en clubs de senderismo, permitiendo a sus socios generar eventos y tener rutas en el móvil, pudiendo compartirlas.

Como hablamos en los objetivos, durante las primeras etapas del proyecto, intentamos ponernos en contacto con el servicio de deportes de la ULPGC, pero no recibimos respuesta, después de múltiples intentos. Este sería el caso más claro de aportación al entorno, ya que la universidad podría usar la aplicación como medio para gestionar todo el sistema de eventos del servicio de senderismo, haciéndolo más rentable, más usado y moderno. La aplicación se moldeó pensando en una posible respuesta, por lo que cumple muchas de sus necesidades.

Estructura de la memoria

Una vez introducido el TFG, pasamos a detallar las diferentes partes que componen este documento.

En el capítulo siguiente, denominado análisis previo, hacemos un resumen del estado del arte del ámbito de nuestro proyecto, incluyen las aplicaciones más importantes que encontramos sobre senderismo, ayudando a ubicar nuestro trabajo dentro de su ámbito y relevancia. Seguidamente hablamos de los requisitos funcionales y no funcionales de nuestra aplicación, detallando sus

principales características, sirviendo de base para el desarrollo de la misma. Dentro del mismo capítulo, hablaremos del modelo de negocio de nuestro proyecto, su viabilidad y posibilidades económicas. Para finalizar el capítulo, hablaremos de los puntos más importantes de la normativa y legislación que atañen a nuestra aplicación, valorando factores como las responsabilidades de los usuarios o la ley de protección de datos.

En el siguiente capítulo, denominado “Desarrollo”, hablamos en una primera instancia, de las diferentes etapas de implementación de nuestro trabajo, de las decisiones importantes tomadas y del alcance de la implementación.

A posteriori, hablamos de todo lo relativo a la ingeniería del software de nuestro proyecto, incluyendo casos de uso, historias de usuario y diagramas de clase.

Finalizaremos este capítulo con una descripción global de la arquitectura de nuestra aplicación, explicando el porqué de su elección y posibles alternativas. A su vez, hablaremos del framework que trae Flutter para la realización de tests, ya sean “unit test”, “integration test” o “widget tests”.

En el capítulo “Tecnologías” desgranaremos más a fondo las diferentes tecnologías usadas para el desarrollo de nuestro trabajo de fin de grado, explicando sus puntos fuertes frente a otras herramientas desechadas.

En el capítulo “Acceso y despliegue” explicaremos como poder revisar el código de nuestra aplicación, dónde está alojado y los motivos de ello.

Para finalizar, hablaremos de las conclusiones obtenidas en nuestro proyecto, así como la opinión personal, las referencias usadas en él y el anexo, con el manual de usuario.

Análisis previo

Estado del arte

Con el objetivo de definir el contexto de nuestra aplicación, se presenta en esta sección un estudio del estado del arte realizado a partir de una recopilación y análisis de algunas de las aplicaciones más relevantes sobre senderismo.

Hemos realizado una búsqueda analítica de las aplicaciones más importantes, tanto en buscadores de internet como en la tienda de aplicaciones Android “PlayStore” o en la tienda de aplicaciones iOS “Apple Store”, centrándonos en sus funcionalidades, analizando aquellas que comparten con las inicialmente propuestas para nuestra aplicación, y estudiando sus posibles carencias y puntos fuertes.

Nuestra búsqueda se centra en dos funcionalidades principales: en las que destaca nuestro proyecto: la **administración de rutas**, encapsulando la creación, la edición o mantenimiento y la eliminación de estas; y la **administración de eventos** de senderismo, con las mismas características que las anteriormente citadas para las rutas.

Otra funcionalidad en la que se ha centrado nuestra búsqueda ha sido la posibilidad de que estas aplicaciones tengan tanto versión android como versión iOS, ya que nuestro proyecto es ‘**crossPlatform**’, funcionando para ambas plataformas.

AllTrails

<https://www.alltrails.com>

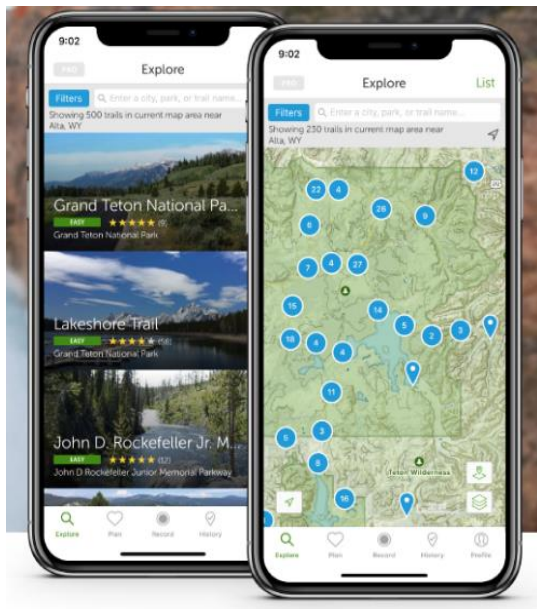
Esta aplicación se caracteriza por tener una interfaz mucho más simple que otras aplicaciones estudiadas aquí.

Su versión gratuita solamente ofrece la posibilidad de buscar, crear y compartir rutas. En su versión de pago, permite a los usuarios descargarse mapas y usarlos cuando no dispongan de conexión a internet.

Tiene una base de datos muy extensa, con múltiples filtros, como, por ejemplo: por países, por parques, más populares, etc.

Dispone de filtros poco frecuentes como son kid-friendly y dog-friendly. A su vez dispone de un modo “historia” donde el usuario podrá ver las rutas realizadas.

Como muchas otras, permite tener un listado de rutas favoritas por cada usuario.



Alltrails usa Mapbox, una librería de código abierto para mapas embebidos en aplicaciones tanto web, móvil o de escritorio. Destaca por la alta personalización de sus mapas no dependiendo de un único servidor de mapas, como puede ser GoogleMaps.

A su vez, usa OpenStreetMap como servidor gratuito de mapas, en forma de imágenes, sin entorno tridimensional, permitiendo combinarlo con Mapbox, dando este funcionalidad extra a OpenStreetMap.

Maps3D

<http://www.movingworld.de>

Destaca por una vista 3D muy trabajada de todos los mapas disponibles, facilitando la orientación del usuario durante un recorrido, sobre todo si no se encuentra en un terreno familiar.

Usa OpenStreetMap como servidor de mapas y los combina con scans 3D de la NASA que han sido previamente optimizados.



No tiene un modo premium, sino que tiene “In-App purchases” para comprar rutas offline y funcionalidades extra en los mapas.

Otra función a destacar es que usa el formato GPX, GPS exchange Format; permitiendo guardar las rutas en este formato. Se suele utilizar para compartir datos entre distintas aplicaciones, usándose para describir puntos, recorridos o rutas. Se basa en el formato XML teniendo una estructura similar. Algunos tags son el <trk> para tracks, el <trkseg> para segmentos de rutas y <wpt> para waypoints, puntos o markers.

```
<wpt lat="39.921055008" lon="3.054223107">
  <ele>12.863281</ele>
  <time>2005-05-16T11:49:06Z</time>
  <name>Cala Sant Vicenç - Mallorca</name>
  <sym>City</sym>
</wpt>
```

Ilustración 9 Ejemplo formato GPX

Una desventaja importante de esta aplicación es que está disponible únicamente para dispositivos iOS.

Spyglass

<http://happymagenta.com/spyglass>

Esta aplicación es quizás la que más se diferencia del resto, transformando el móvil en una brújula, añadiendo funcionalidades únicas como la realidad aumentada, que permite, por ejemplo, usar la posición del sol y las estrellas para guiarte.

Una funcionalidad muy destacada es generar “waypoints” o puntos destacados y, por ejemplo, volver al lugar donde aparcaste el coche, habiéndolo marcado previamente.

Tiene 3 modos, uno gratuito y dos de pago. De ellos, el primero te permite únicamente personalizar los colores de la aplicación; el segundo, permite tener mapas offline, realidad aumentada, uso de las estrellas como guía, etc.



Dispone de una versión para iOS y otra para Android, como la mayoría de este tipo de aplicaciones.

Esta aplicación no dispone de ninguna funcionalidad social como podría ser compartir rutas, pero se destaca del resto por sus funcionalidades únicas con brújulas.

MapMyHike

<https://www.mapmyhike.com>

Esta aplicación combina el senderismo con el mundo fitness, estando centrada en entrenamientos.

Como otras, permite crear y manejar rutas. Permite buscar y revisar. Se combina con otra app, myFitnessPal, para controlar la nutrición del usuario.

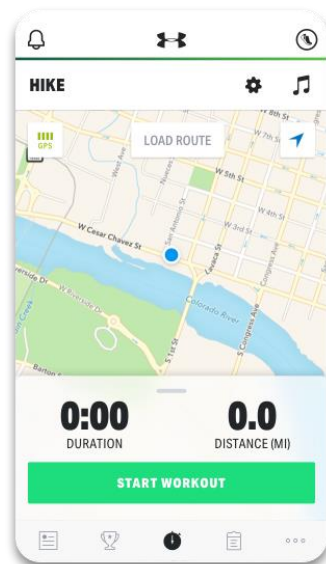
Esta app tampoco tiene un modo premium, pero tiene “in-app products” que varían de los 4 a los 30€. También tiene un modelo de suscripción, con pagos mensuales o anuales.

Esta aplicación permite compartir y realizar rutas, pero no tiene un factor realmente social que permita a los usuarios conectar.

Permite sincronizar con las zapatillas “Under Armour Connected Footwear”, para medir pasos, la cadencia, longitud de zancada, etc.

Tiene un modo desafío que propone marcas a batir en cierto tiempo, por ejemplo, 10 kilómetros en 1 semana.

Tiene versión para iOS y versión para Android.



Wikiloc

<https://es.wikiloc.com/outdoor-navigation-app>

Esta aplicación es la más completa que hemos encontrado en el mercado. Tiene rutas de todo tipo, bici, senderismo, todoterreno, kayak y sus filtros correspondientes. Es la más usada, con 5 millones de miembros y 11 millones de rutas.

Tiene multitud de filtros interesantes como kilometraje, desnivel acumulado o también acotando a la zona que te interesa con el mapa. Da la posibilidad de elegir entre distintos servidores de mapas como son Google maps y OpenStreetMaps.

Permite grabar rutas en vivo para autogenerarlas. También permite seguir al usuario y corregir su posición en caso de desvío. Genera estadísticas como velocidad, distancia recorrida y gráficas de elevación.

Tiene modo premium, con indicador de rumbo, brújula y alertas cuando te alejas de la ruta. Una funcionalidad centrada en la seguridad es el “seguimiento en vivo”, permitiendo que los usuarios que tu elijas vean dónde te encuentras, requiriendo el modo premium.

Tiene un modo sincronización con smartwatches Garmin, permitiendo ver las rutas en el reloj.

El único aspecto social de la aplicación, es compartir con tus amigos que has realizado una ruta, además podrás compartirlas en redes sociales como facebook o twitter.



Requisitos

En este apartado hablaremos de los requisitos funcionales y no funcionales principales de nuestra aplicación. Luego mostraremos los mockups iniciales y finalizaremos justificando la paleta de colores.

Requisitos funcionales

Nuestros requisitos funcionales representan los servicios que ofrece la aplicación:

Respecto al login

- El sistema debe permitir crear cuentas de usuario, usando un correo electrónico válido.
- El sistema validará los datos al crear cuentas.
- El sistema validará que los correos electrónicos tienen un patrón coherente.
- El sistema debe permitir usar la cuenta de Google de un usuario para autenticarse e iniciar sesión.
- El sistema debe permitir resetear contraseñas a los usuarios.
- El sistema debe permitir terminar sesión al usuario.

Respecto al perfil

- El sistema debe permitir ver al usuario sus rutas y sus eventos y desde ahí acceder a ellos.

Respecto a las rutas

- El sistema permitirá crear rutas personalizadas.
- El sistema permitirá editar y/o borrar una ruta propia.

Respecto a eventos

- El sistema permitirá crear eventos en las rutas disponibles.
- El sistema permitirá eliminar y/o editar las rutas propias.

Respecto a visualización de eventos/rutas

- El sistema permitirá ver las rutas disponibles en una lista y/o en un mapa.
- El sistema permitirá ver los eventos disponibles en una lista, en un mapa y/o en un calendario.

Requisitos no funcionales

Requisitos de percepción y estilo

- La aplicación debe tener tonos oscuros y ligeros para una mejor lectura.
- La aplicación debe tener títulos resaltados para mayor claridad.

Requisitos de facilidad de uso

- La aplicación debe ser fácil de usar para cualquier tipo de usuario.
- La aplicación no debe atosigar al usuario con información.
- No debe ser necesaria la retroalimentación.

Requisitos de internacionalización

- La aplicación debe poder tener diferencias opciones de idiomas.
 - Este apartado no se llegó a implementar por falta de tiempo, explicándose en trabajo futuro.

Requisitos de aprendizaje

- La aplicación no requiere de conocimientos previos y no será necesario ningún tipo de entrenamiento.
- La inclinación de la curva de aprendizaje de la aplicación debe ser por tanto pequeña.

Requisitos de velocidad y latencia

- La respuesta debe ser suficientemente rápida para evitar interrumpir el flujo de pensamiento del usuario.

Requisitos de confiabilidad y disponibilidad

- La aplicación estará disponible todo el tiempo.

Requisitos de soporte

- La aplicación funcionará en las dos plataformas más usadas, Android e iOS.

Requisitos de seguridad

- Los usuarios no registrados no podrán usar la aplicación.

Mockups

Los mockups que presentamos en este apartado se realizaron en los momentos iniciales de los análisis de requisitos de nuestra aplicación. Utilizamos la aplicación web Figma para ello. Dejamos el enlace [aquí](#).

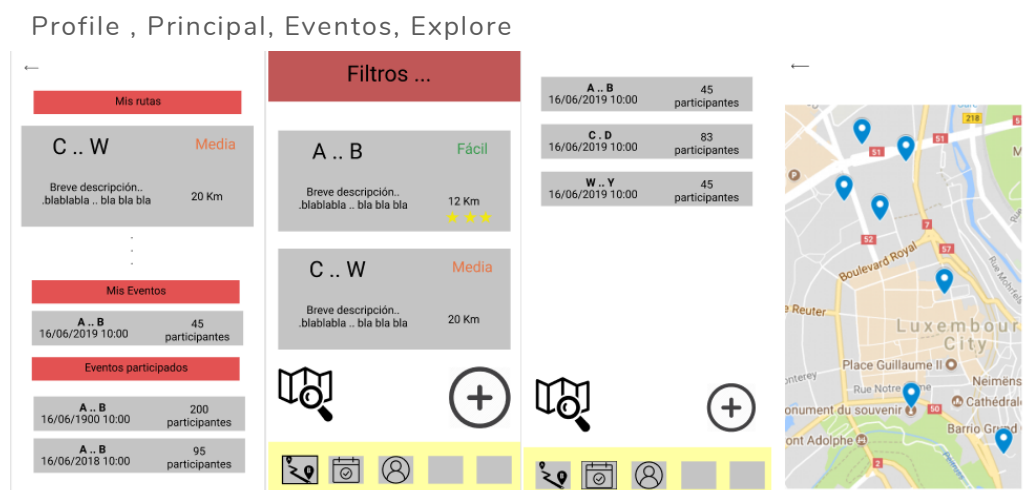


Ilustración 10 Mockups 1

Aquí vemos 4 pantallas principales, **Profile**, donde el usuario ve sus datos y sus rutas y/o eventos, **Principal**, donde el usuario ve una lista de rutas, **Eventos**, donde el usuario ve una lista de eventos y **Explore**, donde ve eventos y/o rutas en un mapa.

New Trail, New Event, Detailed Route, Detailed Event

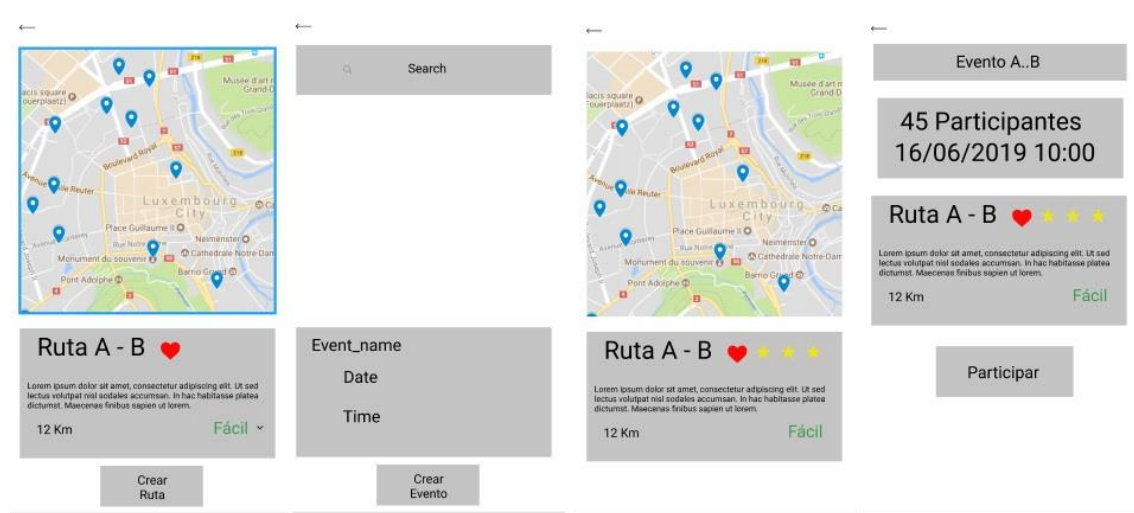


Ilustración 11 Mockups 2

Aquí podemos ver los mockups correspondientes a nuevas rutas, nuevos eventos y las vistas detalladas tanto de rutas como de eventos.

Paleta de colores

A la hora de diseñar la aplicación, investigamos en el portal web [dribbble](https://dribbble.com), un portal web para que los diseñadores puedan compartir sus ideas y diseños. Una vez allí, buscamos interfaces móvil atractivas, encontrando [esta](#), realizada por el usuario “uixNinja”.



Ilustración 12 Interfaz de usuario en Dribbble

Una vez descartadas otras interfaces, decidimos utilizar esta como el modelo a seguir para nuestra aplicación, intentando imitar la paleta de colores y algún que otro matiz como las sombras.

Modelo de negocio

Publicidad, In-app advertising

Una de las formas en las que nuestra aplicación podría ser rentable es añadiendo publicidad en ella, siendo una estrategia de monetización muy efectiva para desarrolladores móviles, donde estos son pagados por mostrar publicidad dentro de las aplicaciones.

Las aplicaciones móviles tienen un entorno mucho más dinámico y atractivo que las aplicaciones web, convirtiendo esta estrategia en un medio efectivo para anunciantes.

A su vez un preciso “In-App Ad Targeting” hace que los anunciantes puedan elegir con mayor certeza el público al que van dirigidos sus anuncios.

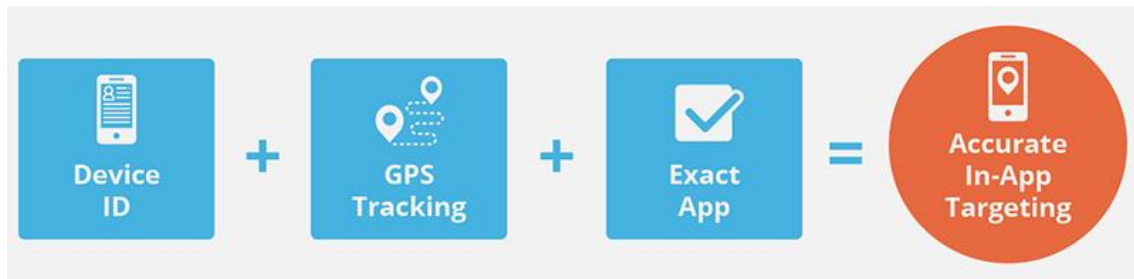


Ilustración 13 Preciso "In-App Ad Targeting" (<https://www.smaato.com/guide-to-in-app-advertising>)

Como discutimos en el contexto de nuestro proyecto, los usuarios de aplicaciones móviles prefieren las aplicaciones gratuitas a aplicaciones de pago, por lo que la publicidad dentro de las aplicaciones se convierte en el modelo preferido de las aplicaciones gratuitas, siendo, en muchos casos, su única forma de monetización.

Modelo Freemium

Freemium es un modelo de negocio cuyo nombre proviene de la fusión de dos palabras en inglés “Free” y “Premium”, este sistema consiste en que una empresa otorgue a los usuarios un servicio completamente funcional de manera gratuita, además de ofrecer una opción en la que el usuario siempre pueda adquirir mayores beneficios, es decir, adquirir una licencia Premium la cual le permitirá obtener herramientas avanzadas y otros beneficios del mismo software en relación a la versión gratuita.

Muchas de las aplicaciones estudiadas en el estado del arte usan este modelo como fuente de ingresos, dando dos versiones de su aplicación, una, gratuita y con funcionalidades mínimas y otra, premium, con la totalidad de ellas.

Si quisiéramos implementar este modelo en nuestra aplicación, podríamos añadir funcionalidades como un modo offline en el modelo premium, dando incentivos a los usuarios para descargarla.

Nuestra aplicación es reactiva y en todo momento muestra los datos reales, pudiendo dejar esa funcionalidad en el modo premium, obligando a los usuarios a actualizar cada vez que lo necesiten.

Porcentajes

Si aplicásemos el modelo que utiliza el servicio universitario de senderismo a la hora de cobrar por sus eventos de senderismo, podríamos mejorar nuestra aplicación generando un servicio de pagos “in-app” donde los usuarios tendrían que pagar para poder inscribirse en eventos.

PRECIO:
Comunidad Universitaria: 6€
Externos: 10€
LUGAR DE SALIDA:
Fuente Luminosa: 8:30 horas
Campus Universitario de Tafira: 8:45 horas
(Parada junto a la cafetería "Las Casitas")

Ilustración 14 Evento universitario de senderismo

Dado este servicio, podríamos descontar un porcentaje de dicha inscripción para nosotros.

Este método de pago implicaría cambios importantes en nuestra aplicación y en la ley de protección de datos, que habría que considerar antes de implementar.

Normativa y legislación

Ley Orgánica de Protección de Datos

Nuestra aplicación, respecto a la **Ley Orgánica de Protección de Datos:**

- Cumple con las obligaciones como responsables cuando tratamos datos de usuarios.
- Solicitamos consentimiento con carácter previo y consentimiento específico.
- Permitimos a los usuarios rescindir su consentimiento y desinstalar la app.
- Facilitamos una política de privacidad inteligible y fácilmente accesible.
- Facilitamos a los usuarios el ejercicio de sus derechos ARCO (Acceso, Rectificación, Cancelación y Oposición).
- No conservamos datos de usuarios.

Uso que da el usuario

Nuestra aplicación, respecto al **uso que le dé el usuario**:

- No se hace responsable de la veracidad del contenido de las rutas y/o eventos.
- No se hace responsable de la legalidad y el acceso a las rutas.
- No se hace responsable del contenido y descripciones de las rutas y/o eventos.

Persistencia de datos

Nuestra aplicación, respecto a la **persistencia de datos**:

- Entiende que el usuario comparte sus rutas y/o eventos de manera intencional.
- Entiende que el usuario permite la visualización de dicha información.
- Permite que el servicio Cloud Firestore maneje la autenticación de usuario según sus propias cláusulas.

Desarrollo

Alcance de la implementación

En este apartado explicaremos como ha transcurrido el desarrollo e implementación de nuestra aplicación de un modo **cronológico**, explicando las dificultades y las decisiones tomadas.

Interfaz

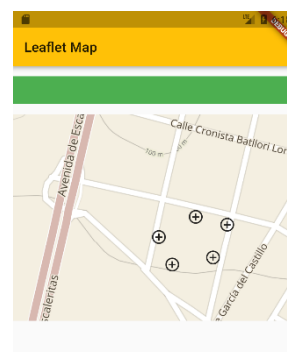
La primera incógnita a la hora de implementar el proyecto fue qué hacer primero. Después de debatir con el profesor qué era lo mejor, decidimos hacer la interfaz primero con un modelo simulado. Decidimos que la funcionalidad y el backend se implementara más tarde.

Google+ vs otros

La primera decisión importante que tuvimos que tomar fue qué plugin utilizar para gestionar los mapas. Teníamos muchas opciones pero las reducimos a dos:

Flutter_map

Flutter_map es un plugin que utiliza [Leaflet](#) para dibujar mapas. Una de sus mayores ventajas era la posibilidad de configurar el mapa, pudiendo cambiar, por ejemplo, los colores usados en el mapa.



Google_maps_flutter

Este [plugin](#), que todavía esta en “modo beta”, presenta todas las características disponibles del servicio de Google Maps, lo que nos hizo decantarnos por él. Su

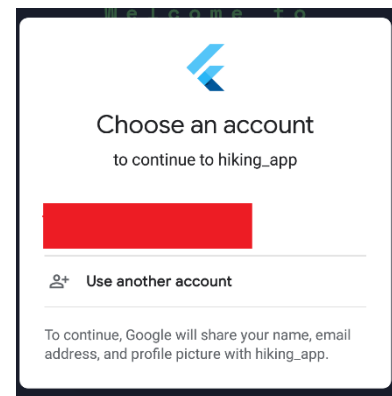
mayor punto negativo, es que no es una vista compilada en dart, sino una vista nativa embebida, lo que hace que su rendimiento no sea óptimo.

Autenticación

Uno de los pasos que requirió más esfuerzo fue desarrollar un sistema que se autentificara con el backend. La dificultad no se encontraba en el desarrollo en sí, si no en la complejidad y el comportamiento del sistema.

Para su implementación, utilizamos el plugin [firebase_auth](#), que aporta la funcionalidad necesaria para ello.

Una de las ideas que se nos ocurrió mientras implementábamos esta funcionalidad, era poder iniciar sesión con nuestra cuenta de Google+. Para ello utilizamos el plugin [Google_sign_in](#), que combinado con el plugin `firebase_auth`, generan una funcionalidad de sesión completa para el usuario, mejorando la calidad del sistema.



Provider

A

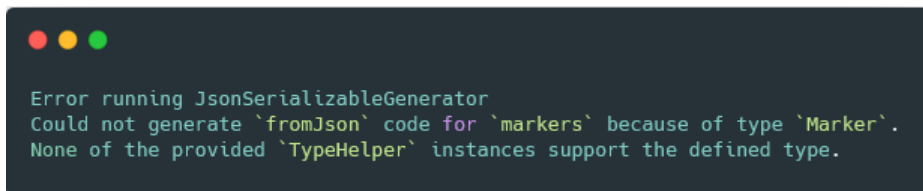
Serialización

La serialización – transformar los objetos de nuestro modelo en formato json y viceversa – fue la funcionalidad que más tiempo nos ocupó. La serialización es necesaria para tener un backend funcional y seguro, ya que, usando una base de datos NoSQL, los objetos pueden estar compuestos por cualquier dato, siendo aceptados por la base de datos. Esto implica que nuestro sistema debe generar de forma segura y consistente los objetos que van a ser serializados y enviados

al backend, para ello utilizamos tres plugins: `json_annotation`, `build_runner` y `json_serializable`.

Cuando utilizamos las herramientas que Flutter propone para serializar un objeto, generamos unos ficheros con el sufijo “.g.” que aportan los métodos para serializar y deserializar. La primera vez que utilizamos la herramienta con el modelo “evento”, todo funcionó a la perfección.

El problema lo tuvimos con el modelo “ruta”. Las herramientas propuestas, tienen la capacidad de manejar los tipos conocidos, pero ¿qué ocurre si en nuestro modelo tenemos un objeto no primitivo?.



```
Error running JsonSerializerGenerator
Could not generate `fromJson` code for `markers` because of type `Marker`.
None of the provided `TypeHelper` instances support the defined type.
```

Ilustración 15 jsonGeneration error

El plugin de Google maps tiene una clase “Marker” que usábamos en nuestro modelo para generar la lista de markers que necesitamos para poder visualizar puntos en un mapa, pero al ser un objeto al que nuestro sistema de generación no tiene acceso, no podemos utilizarlo.

La solución tomada fue tener un mapa de objetos del tipo `<String, Dynamic>`, siendo Dynamic cualquier cosa, para guardar los markers y luego transformarlos manualmente con los métodos que generamos en la clase `MarkerHelper`.



```
static Map<String,dynamic> setToMap(Set<Marker> markersList){
  Map<String, dynamic> map = {};
  int index = 0;
  markersList.forEach((marker){
    map['$index'] = {
      'latitude': marker.position.latitude,
      'longitude': marker.position.longitude
    };
    index++;
  });
  return map;
}
```

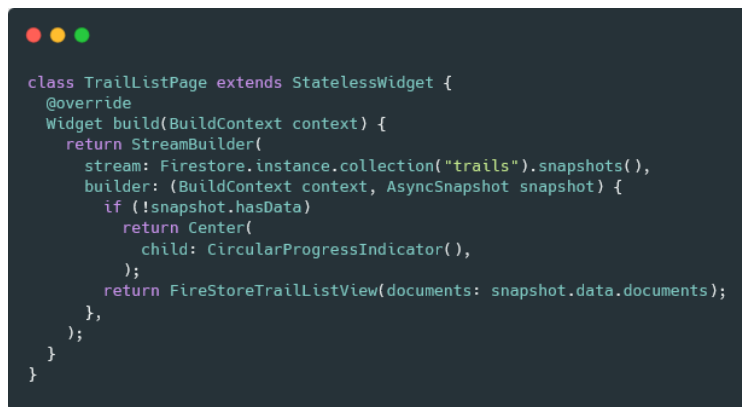
Ilustración 16 Método setToMap en MarkerHelper

Componentes reactivos

Una vez corregimos todos los problemas que tuvimos con nuestros modelos y el backend, decidimos aplicar este paradigma en nuestra aplicación. Para ello, necesitamos entender StreamBuilders.

StreamBuilders

Este widget propuesto por Flutter, escucha eventos del stream al que está suscrito. Por cada nuevo evento, redibujará sus hijos, aportándole la información actualizada.



```
class TrailListPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return StreamBuilder(
      stream: Firestore.instance.collection("trails").snapshots(),
      builder: (BuildContext context, AsyncSnapshot snapshot) {
        if (!snapshot.hasData)
          return Center(
            child: CircularProgressIndicator(),
          );
        return FirestoreTrailListView(documents: snapshot.data.documents);
      },
    );
  }
}
```

Ilustración 17 Clase TrailListPage con Streambuilder

Esta clase muestra la utilización del objeto StreamBuilder, teniendo como “hijo” a la clase FirestoreTrailListView, que será dibujado cuando el objeto “snapshot” tenga datos, como demuestra la condición.

Nuestra clase FirestoreTrailListView no es más que una lista de objetos “ListTile”, que se generan a través de los documentos que se les pasan por parámetro. Cuando el stream se actualiza con nuevos datos enviados por la base de datos, esta clase se redibujará con los datos actualizados.

Esta arquitectura es así para toda nuestra aplicación, por lo que cada pantalla es reactiva.

Filtros

Los filtros fue lo último que implementamos, siendo más fácil de lo esperado. Para ello simplemente preguntamos por las condiciones, que serán fijadas por botones.

En este ejemplo queremos filtrar los eventos por texto. Comprobamos que el texto está en el nombre del evento, en la descripción o en el nombre del creador, si no es así, no mostramos ese objeto y mostramos un SizedBox vacío, ya que flutter no permite dibujar widgets como null.

A screenshot of a code editor with a dark background and light-colored text. The code is a Dart function that filters events based on a search message. It starts with an 'if' statement checking if the message length is greater than 0. Inside, there's a comment '//if is not in any of those' followed by another 'if' statement that checks if the message is not contained in the event's name, description, or creator name. If it's not contained, it returns an empty 'SizedBox()' widget. The code is as follows:

```
if (message.length > 0) {  
  //if is not in any of those  
  if (!event.eventName.contains(message) &&  
      !event.description.contains(message) &&  
      !event.creatorName.contains(message)) {  
    return SizedBox();  
  }  
}
```

Ilustración 18 Ejemplo de filtro

Sabemos que la lista se actualizará con nuevos datos gracias al StreamBuilder, pero ¿cómo actualizamos nosotros una vez pulsamos un botón de filtrado?, con setState, obligando a la lista de eventos a redibujarse.

Experiencia personal

Durante la búsqueda de prácticas de empresas, en el primer cuatrimestre, todas las ofertas pedían desarrollo multiplataforma, especialmente react-native. Acabé realizando las prácticas en Inerza, donde utilizan [Codename One](#).

Ingeniería del software

Casos de uso

Hemos dividido los casos de uso en estados más pequeños por claridad.

Casos de uso de sesión

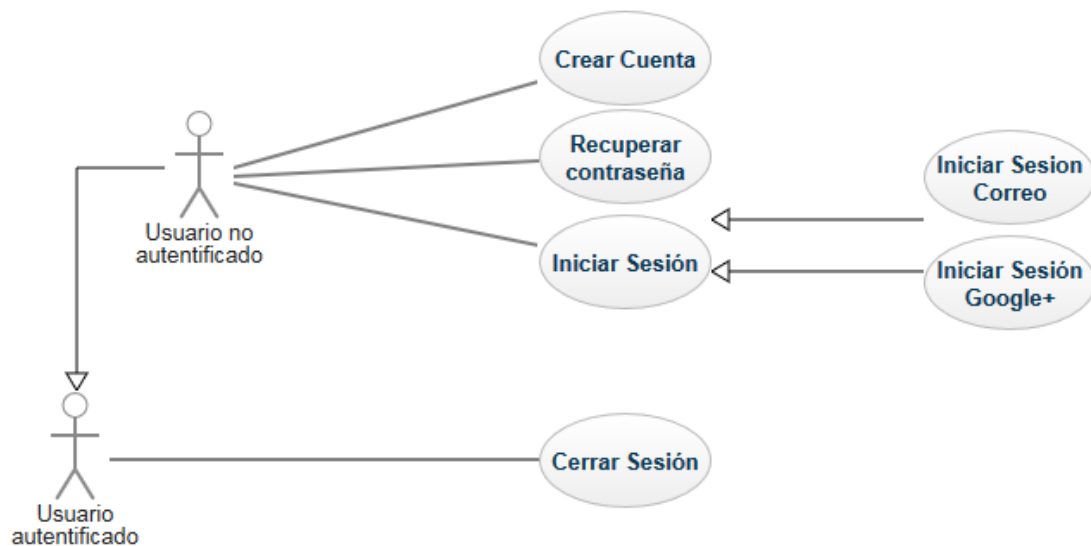


Ilustración 19 Casos de uso de sesión

En lo que respecta al uso de sesión en nuestra aplicación, podemos diferenciar dos actores, dependiendo del estado de la autenticación.

El usuario no autenticado se encontrará con la posibilidad de iniciar sesión de la forma clásica, con un correo y una contraseña, o usando su cuenta de Google+. A su vez, el sistema le permite crear una cuenta y recuperar una contraseña en caso de no acordarse.

Casos de uso rutas y eventos

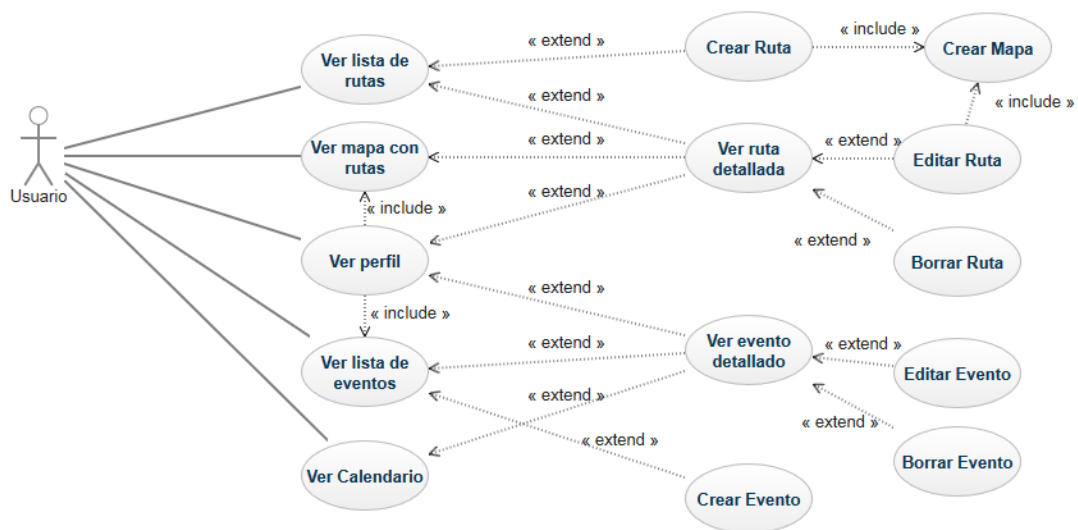


Ilustración 20 Casos de uso Rutas y Eventos

Una vez un usuario se autentifica en nuestra aplicación, dispone de varias funcionalidades para elegir.

La aplicación carga por defecto el perfil del usuario, mostrando información personal y las rutas y eventos creadas por el usuario. Desde esta vista, podemos ir a la vista detallada tanto de rutas como de eventos.

El usuario ahora puede elegir ver la lista de todas las rutas disponibles, pudiendo acceder a una vista detallada de cada una. Lo mismo ocurriría con los eventos, mostrados en otra lista y su correspondiente vista detallada.

El usuario dispone de la posibilidad de ver un mapa con todas las rutas disponibles y un calendario con los eventos ordenados por fecha. Desde dichas vistas podremos acceder a las vistas detalladas.

Casos específicos: Filtrado

El usuario podrá filtrar, en las vistas de listas, tanto rutas como eventos. Las rutas las podrá filtrar por dificultad, si son parte de favoritos o no y por texto.

Los eventos los podrá filtrar por participación, por fecha y por texto.

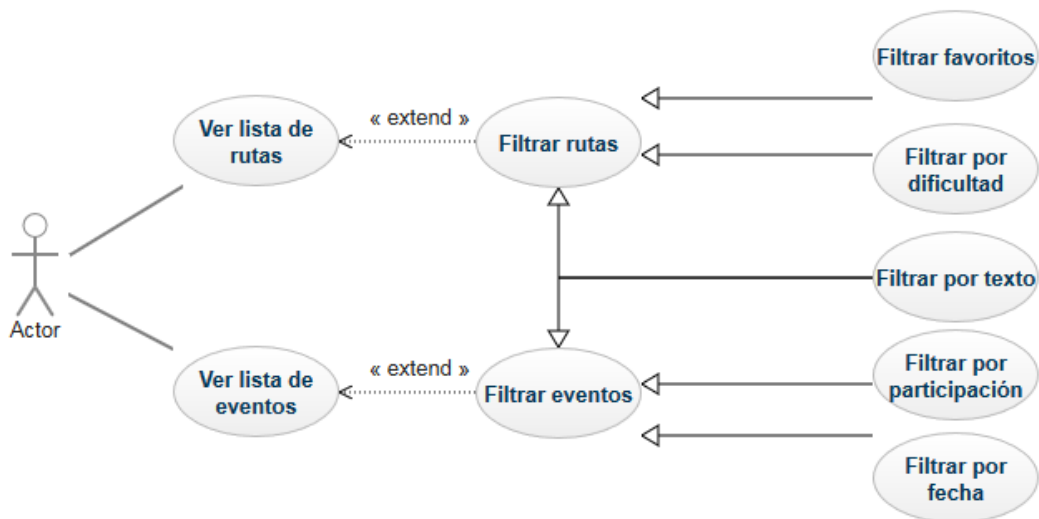


Ilustración 21 Caso de uso de filtrado

Diagramas de clase

En este apartado explicaremos con diagramas de clase la arquitectura y el diseño de nuestro proyecto. Para la realización de dichos diagramas, hemos utilizado la herramienta [dcdg](#) (Dart Class Diagram Generator), que nos ha generado dichos diagramas de forma automática. Hemos dividido las clases en pequeños grupos por mayor claridad.

Modelos

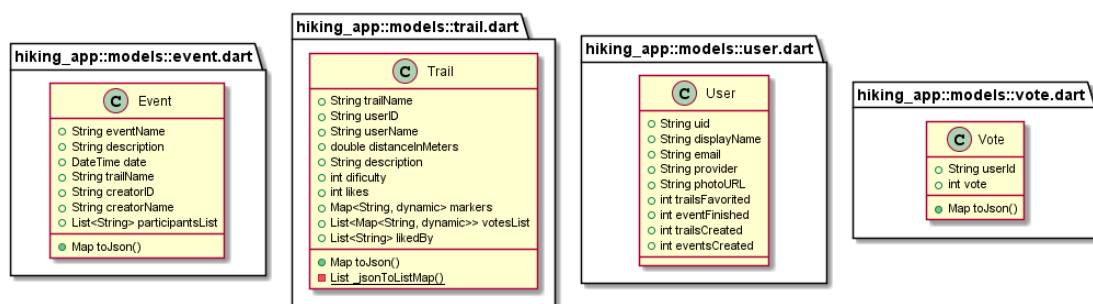


Ilustración 22 Diagramas de modelo

Nuestras clases de modelo son bastante simples. Tenemos la clase Event, la clase Trail, la clase User y la clave Vote. La mayor complejidad la encontramos en los métodos para transformar a json y viceversa. Estos métodos son

FirestoreEventListView, pero con una colección de Eventos. Aunque por el momento estos objetos no son reutilizados, podrían serlo añadiendo nuevas funcionalidades a la aplicación.

Login_pages

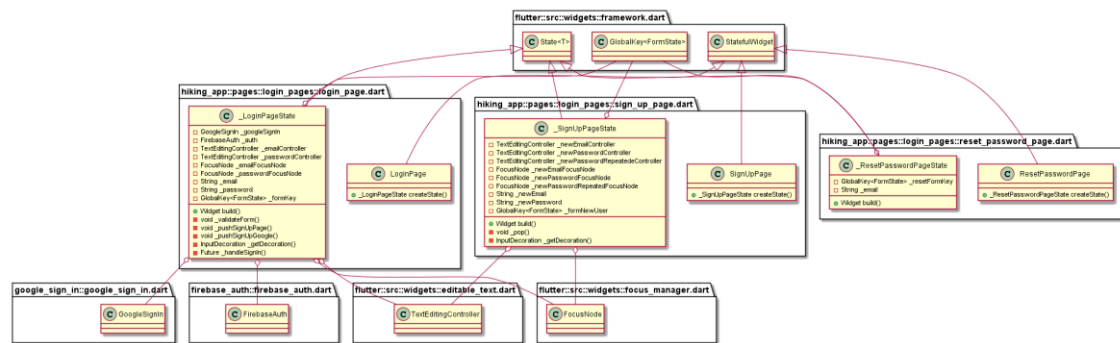


Ilustración 24 Vistas de acceso

Aquí vemos por primera vez un StatefulWidget. Declaramos las vistas con formularios así ya que cambiarán de forma dinámica cuando los usuarios escriban datos. Cada vista que extiende un StatefulWidget tiene un método “createState” que devuelve el objeto “state” de la vista. Este objeto state tiene el método build que dibuja la vista.

```

class DetailedTrail extends StatefulWidget {
  final String documentID;
  DetailedTrail(this.documentID);
  @override
  _DetailedTrailState createState() => _DetailedTrailState();
}
  
```

Ilustración 25 StatefulWidget

La clase LoginPage hace uso de dos librerías muy importantes: GoogleSignIn y FirebaseAuth. Estas clases traen la funcionalidad necesaria para poder autenticarnos en el backend tanto con correo como con Google+.

New_views



Detailed_views



42

principio, parece innecesario, ya que tenemos la colección con todas en la pantalla anterior, pero si hacemos una llamada nueva aquí, conseguimos que la aplicación sea reactiva. Esto se consigue ya que mientras estamos viendo una vista detallada, estamos suscritos a los cambios que ocurran en dicho documento.

Errors

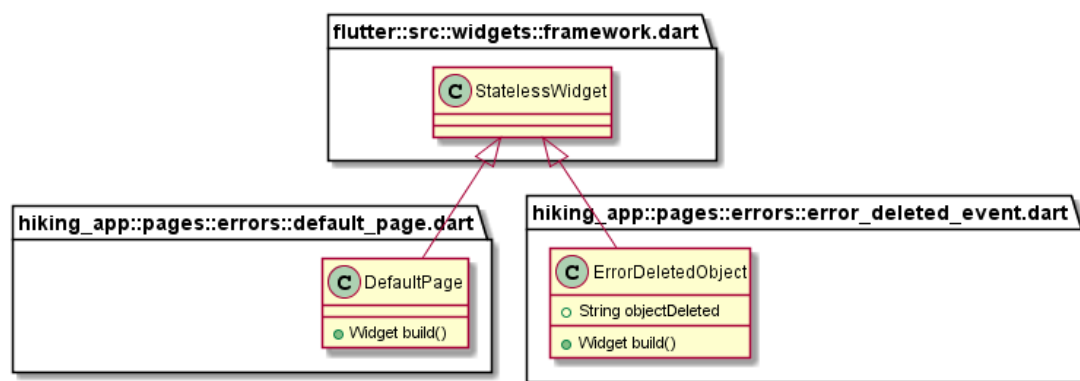


Ilustración 28 Vista de errores

Ahora mostramos las alternativas que hemos creado para cuando nuestra aplicación se encuentra cargando algún dato o un objeto ha sido borrado. La vista `DefaultPage` es una vista con un `CircularProgressIndicator` que es actualizada con una vista apropiada cuando los datos llegan. La vista `ErrorDeletedObject` nos ayuda a avisar a un usuario cuando está viendo una vista detallada y el documento es eliminado del backend.

Styles

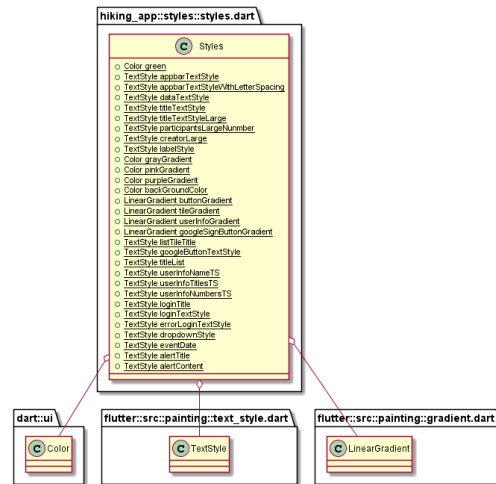


Ilustración 29 Estilos

La clase styles es una clase con objetos estáticos que nos sirve para agrupar configuraciones visuales. En Flutter es frecuente este tipo de diseño, imitando un fichero CSS. Esta clase devuelve colores, estilos de texto y gradientes para widget como pueden ser los botones.

Utils

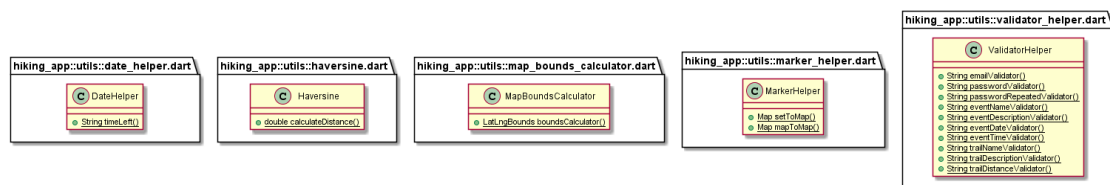


Ilustración 30 Utilidades

Estas clases son helpers que nos ayudan con diferentes tipos de problemas. DateHelper nos ayuda a saber si un evento ha pasado o no con su método timeLeft.

[Haversine](#), hace uso de una [librería](#) para calcular la distancia entre dos puntos y así poder calcular la distancia total de una ruta. MapBoundsCalculator nos ayuda a centrar los mapas al cargarlos. MarkerHelper hace transformaciones entre objetos, generalmente a de lista de Markers a mapas. Por último ValidatorHelper

lo usamos en los formularios a lo largo de nuestra app, para validar los campos escritos por los usuarios.

Vistas principales

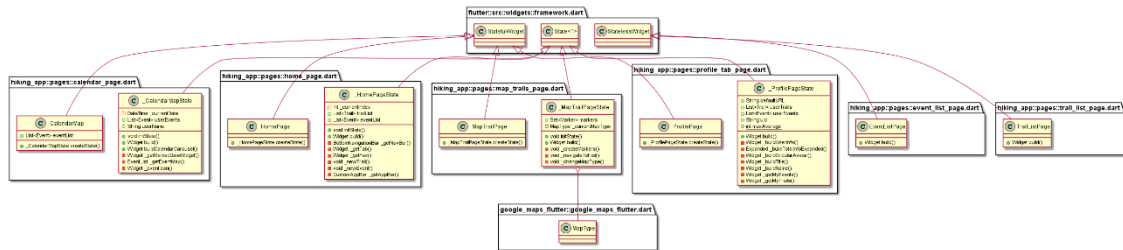


Ilustración 31 Diagramas con las vistas principales

Estas son las cinco vistas principales de nuestra aplicación, siendo la clase `HomePage` la encargada de controlar qué vista es cargada mediante un `BottomNavigationBar`, haciendo de contenedor de las vistas posibles.

`ProfileTabPage` usa las vistas descritas en componentes para cargar las vistas de los eventos y las rutas creadas por el usuario. A su vez, muestra información general del usuario.

`TrailListPage` usa también las vistas descritas en componentes, suscribiéndose a los cambios que puedan ocurrir en la colección de rutas, construyendo la vista de rutas de forma estática. Así, se reconstruirá de manera automática cuando ocurra algún cambio. De la misma manera funciona la clase `EventListPage`.

`MapTrailListPage` es una vista personalizada de las posibles rutas. Cargamos dichas rutas de la misma manera que otras vistas, para mantener el paradigma reactivo. Usamos un widget de [GoogleMaps](https://pub.dev/packages/google_maps_flutter) para implementar dicha funcionalidad. Disponemos de algunas funcionalidades extra como puede ser cambiar el tipo de mapa.

Por último mostramos la clase CalendarPage, haciendo uso de la librería [Flutter Calendar Carousel](#), en la cual mostramos los eventos ordenados por fecha.

Modelo de la base de datos

Nuestra base de datos es una base de datos no relacional o NoSQL. Esto quiere decir que no existen tablas “fijas” y que la información almacenada puede ser variable. Aún así, intentamos emular una tabla de datos relacional con dos tablas, rutas y eventos. Nuestra base de datos permite a un usuario crear tantas rutas como eventos desee. Nuestros eventos tienen una ruta asociada a este.

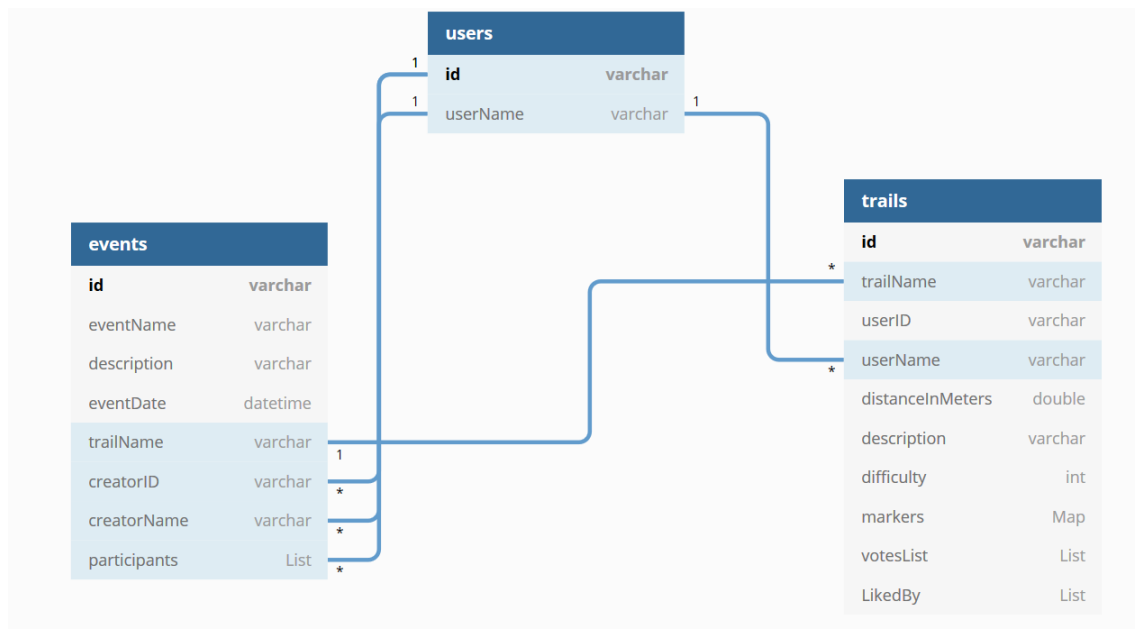


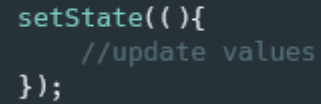
Ilustración 32 Modelo de la base de datos

Arquitectura

En lo que se refiere a Flutter, la arquitectura es un tema muy delicado. Flutter es un framework que permite diseñar tu proyecto de la manera que más convenga dadas las necesidades de tu aplicación. De manera nativa, Flutter no fuerza ninguna arquitectura, dando una funcionalidad muy básica para diseñarlas. A esa funcionalidad la llamaremos “**Vanilla**” de aquí en adelante.

Vanilla

El modo Vanilla, consiste en mezclar la “User Interface” declarativa que promueve Flutter, con la lógica y el control de estados. Para cambiar valores, se usa una función dada por el framework llamada



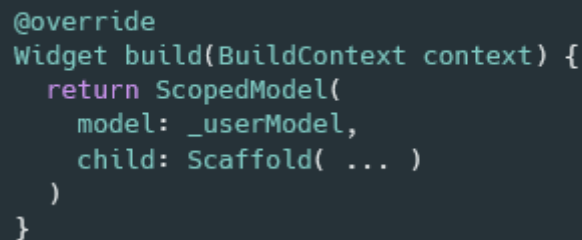
```
setState((){  
    //update values  
});
```

“**setState**”, que obliga a reconstruir el árbol de widgets existente. Esto es, en otras palabras, la pantalla que estamos viendo en ese momento. Así, actualizaríamos un valor en la pantalla, por ejemplo.

Con esta aproximación, podríamos utilizar las arquitecturas más conocidas, como pueden ser el modelo-vista-controlador, el model-view-viewmodel u otras por el estilo.

Scoped Model y Redux

Existen otras aproximaciones, frecuentes en otros frameworks, como pueden ser “scoped model” o “redux”. Scoped model consiste en especificar que objeto o modelo va a ser actualizado de antemano, haciéndolo, como su nombre indica, “alcanzable”.



```
@override  
Widget build(BuildContext context) {  
    return ScopedModel(  
        model: _userModel,  
        child: Scaffold( ... )  
    )  
}
```

Ilustración 33 Ejemplo de ScopedModel

Redux, es un paradigma que se encarga de “desacoplar” el estado global de una aplicación de su interfaz y sus componentes.

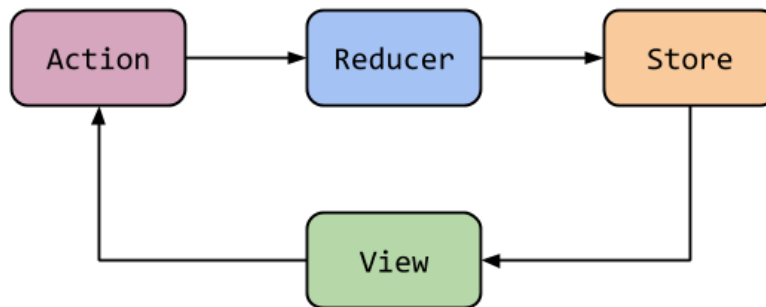


Ilustración 34 Ejemplo de Redux

Y nosotros usamos **Provider**, una combinación de todo lo anterior.

Provider

Esta librería tiene una historia muy interesante. Mientras Google diseñaba sus propios métodos para conseguir el mismo objetivo, un desarrollador, [Remi Rousselet](#), creó [esta librería](#). Google la vio, la analizó, canceló la suya y propuso a la comunidad de Flutter que utilizáramos Provider. Lo hizo ni más ni menos que en el [Google I/O'19](#).

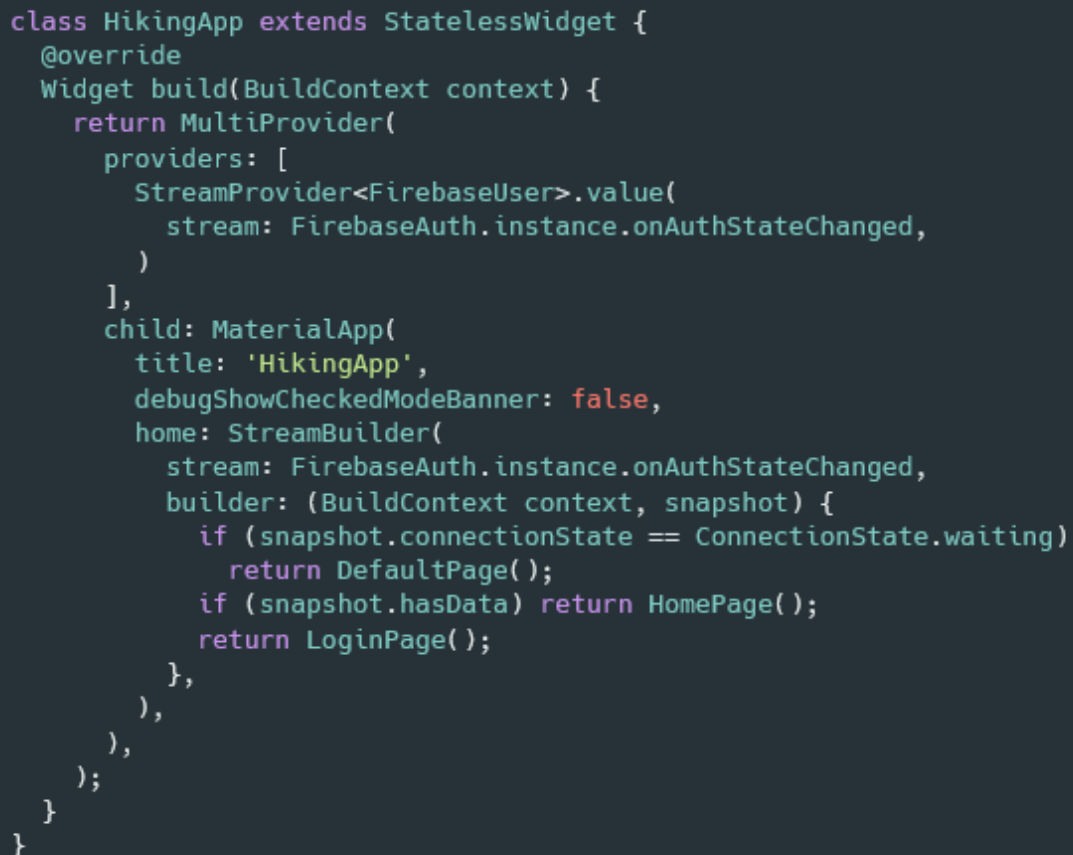
Provider es una librería que se autodefine como:

*“A dependency injection system built with widgets for widgets. **provider** is mostly syntax sugar for **InheritedWidget**, to make common use-cases straightforward.”*

O en otras palabras, provider te permite exponer el modelo que desees e inyectarlo en la vista cuando lo necesites. Esta construido sobre la idea de “scoped model”, teniendo que definir el modelo de antemano y te desacopla el modelo de la vista.

Provider y Streams

El paradigma “streams” se define como una secuencia de datos que están disponibles a lo largo del tiempo. Nosotros combinamos este paradigma con la librería Provider para conseguir una aplicación completamente reactiva.



```
class HikingApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        StreamProvider<FirebaseUser>.value(
          stream: FirebaseAuth.instance.onAuthStateChanged,
        ),
      ],
      child: MaterialApp(
        title: 'HikingApp',
        debugShowCheckedModeBanner: false,
        home: StreamBuilder(
          stream: FirebaseAuth.instance.onAuthStateChanged,
          builder: (BuildContext context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting)
              return DefaultPage();
            if (snapshot.hasData) return HomePage();
            return LoginPage();
          },
        ),
      ),
    );
  }
}
```

Ilustración 35 Provider y Streams

En este ejemplo real de nuestra aplicación, vemos como podemos exponer, gracias a Provider y al objeto StreamBuilder de Flutter, el estado de la sesión de un usuario. De esta manera, cuando la sesión cambie (abra o cierre sesión), la aplicación mostrará una HomePage() o una LoginPage(), dependiendo del stream de datos denominado **FirebaseAuth.instance.onAuthStateChanged**. Este stream, gracias a Provider, podrá ser accedido en cualquier momento, en

cualquier vista de nuestra aplicación. Eso quiere decir que la funcionalidad de cerrar sesión, simplemente debe “cerrar sesión”, sin tener que avisar o notificar a nada ni nadie.

Streams

Basándonos en este paradigma, conseguimos que toda nuestra aplicación sea reactiva. En ningún momento se aplica “Busy waiting”, que consiste en preguntar constantemente si un dato ha cambiado. El backend, cuando considera necesario, nos envía datos a los streams en los que estamos suscritos y la aplicación se redibuja sola, sin que el desarrollador deba controlar su estado y sin que el usuario deba realizar alguna acción para actualizar los datos.

Tests

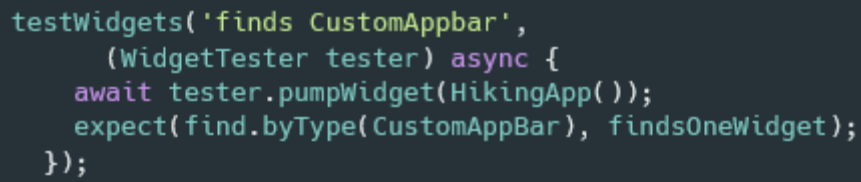
Flutter, según su [página web](#), nos permite hacer tres tipos de test: Unit test, Widget tests e Integration test.

En nuestra aplicación, bajo el directorio tests, realizamos dos de los tres tipos de tests disponibles. Debido a la falta de tiempo, no pudimos realizar todos los tests que hubiéramos querido.

```
C:\Users\nauze\FlutterProjects\hiking_app>flutter test
00:09 +43: All tests passed!
```

Ilustración 36 Ejemplo de ejecución de tests

Los más interesantes, diferentes a otros frameworks para realizar tests, son los **widget tests**.



```
testWidgets('finds CustomAppBar',
  (WidgetTester tester) async {
    await tester.pumpWidget(HikingApp());
    expect(find.byType(CustomAppBar), findsOneWidget);
  });
```

Ilustración 37 Widget Test

En este test que realizamos en nuestra aplicación, lanzamos el widget `HikingApp()` y luego buscamos **por tipo** un objeto “`CustomAppBar`”, esperando encontrar uno.

Flutter nos permite buscar por tipo, por texto y por muchos otros parámetros, permitiéndonos compararlo con lo encontrado.

Tecnologías

Flutter

Como ya hemos dicho anteriormente, **Flutter**, es un framework libre, basado en Dart, un lenguaje orientado a objetos creado también por Google. Permite un desarrollo muy rápido de las aplicaciones gracias a multitud de librerías y herramientas. Una de sus mejores características es que el código resultante es “ahead of time”, lo que hace que las aplicaciones estén “precompiladas” antes de ser utilizadas, mejorando los tiempos de carga enormemente), haciendo que el rendimiento de las aplicaciones se pueda equiparar al código nativo, mientras que muchas son “Just in time”, compiladas en tiempo de ejecución, como es el caso de Java y la máquina virtual en Android. Este framework carece de la madurez necesaria para competir con otros, pero está creciendo a pasos agigantados. Una de las razones por las que se debería apostar por Flutter es Fuchsia, un nuevo sistema operativo diseñado por Google para sustituir a Android, donde la programación se realizará en Flutter y Dart.

Flutter es diferente al resto ya que no usa “Webviews” o ni las vistas nativas que vienen con el sistema operativo, sino que usa su propio motor de renderizado para dibujar widgets en la pantalla. Este motor de renderizado es [Skia](#), una librería gráfica 2D más conocida por ser el motor de renderizado de Google Chrome, el explorador web más utilizado.

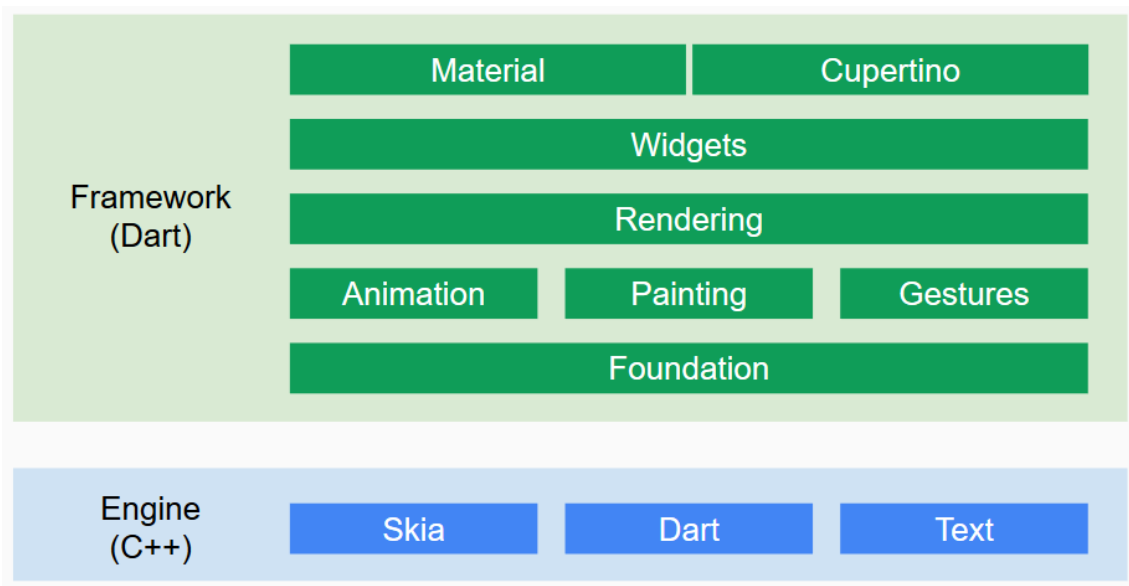


Ilustración 38 Flutter Framework y Engine

¿Cómo funciona en Android?

El código del motor escrito en C++ es compilado con el [NDK de Android](#). El código en Dart (el del desarrollador y el propio del framework) es compilado “ahead of time” en código nativo, código ARM y librerías x86. Este código está incluido en un proyecto Android que se construye dentro de una APK. Cuando lanzamos una aplicación Flutter en Android, la aplicación carga la librería de Flutter, haciendo que cualquier tipo de renderizado o control de eventos sea delegado al código Dart, tal y como ocurre en muchos motores gráficos para juegos.

¿Cómo funciona en iOS?

El código del motor ahora es compilador con [LLVM](#) (Low Level Virtual Machine). El código Dart es compilado “Ahead of time” en código nativo y código ARM, que es incluido en un proyecto iOS, que conforma el fichero “.ipa”, formato usado en aplicaciones para iOS. Cuando lanzamos la aplicación en iOS, funciona de la misma manera que en Android, siendo Flutter quien controla el renderizado, los eventos, etc.

El diseño de interfaces con Flutter se basa en widgets y los widgets se dividen en dos tipos, **stateful** y **stateless**.

“Decimos que un widget es un stateless widget si es estático, es decir, si no va a sufrir ningún cambio. Por ejemplo un texto, un icono... en principio son widgets que van a ser pintados en la pantalla y así se van a quedar, sin que ninguna acción del usuario los altere.

*Por otro lado tenemos a los stateful widgets, o widgets dinámicos. Éstos sí que pueden sufrir cambios debidos seguramente a una acción del usuario. Un claro ejemplo puede ser un checkbox, si el usuario pulsa sobre el su aspecto cambiará. Otro ejemplo puede ser un TextField, porque el usuario puede escribir sobre él y por tanto cambiarlo. Este tipo de widgets tienen un objeto «**State**» asociado para gestionar su estado (de ahí el nombre). Si el estado cambia el widget volverá a pintarse en la pantalla con la apariencia actualizada. Esto es muy importante porque cuando creamos nuestro propio stateful widget tendremos también que crear otra clase state.”*

(García, <https://guillermogarcia.es>, s.f.)

Cloud Firestore

Cloud Firestore es una base de datos NoSQL flexible, escalable y en la nube a fin de almacenar y sincronizar datos para la programación en el lado del cliente y del servidor.

El modelo de datos de Cloud Firestore admite estructuras de datos flexibles y jerárquicas. Almacenamos los datos en documentos, organizados en colecciones. Los documentos pueden contener objetos anidados complejos, además de subcolecciones.

En Cloud Firestore, puedes usar consultas para recuperar documentos individuales específicos o para recuperar todos los documentos de una colección que coinciden con los parámetros de la consulta. Las consultas pueden incluir varios filtros en cadena y combinar los filtros con criterios de orden. También se indexan de forma predeterminada, por lo que el rendimiento de las consultas es proporcional al tamaño del conjunto de resultados, no del conjunto de datos.

Cloud Firestore usa la **sincronización de datos** para actualizar los datos de cualquier dispositivo conectado. Sin embargo, también está diseñado para ejecutar consultas de recuperación únicas y sencillas de manera eficiente.

Almacena en caché datos que usa tu app de forma activa, por lo que la app puede escribir, leer, escuchar y consultar datos, aunque el dispositivo se encuentre sin conexión. Cuando el dispositivo vuelve a estar en línea, Cloud Firestore sincroniza todos los cambios locales de vuelta a Cloud Firestore.

Acceso y despliegue

Nuestro proyecto está alojado en [Github](#), en un repositorio privado. Los motivos de esta elección son la existencia de ciertos ficheros de configuración con datos sensibles a ser copiados. Uno es la API KEY de Google maps, estando en ficheros como el `AndroidManifest.xml`, fácilmente extraíble. Otro es el `Google-service.json`, con datos sensibles sobre nuestro backend y su conexión. Por ello, entregaremos la versión final de nuestro código al tribunal en formato zip o en su defecto generaremos un nuevo repositorio público sin los ficheros anteriormente explicados.

Conclusiones

“Un viaje de mil millas comienza con un primer paso” – Lao-Tse.

Y termina aquí.

Hemos alcanzado todos los objetivos propuestos en este proyecto, tanto los personales como los funcionales. El resultado es una aplicación funcional completamente reactiva y utilizable.

Desde un punto de vista más personal, el haber realizado una aplicación completa me ha ayudado a generar confianza para el futuro. Creo que puedo y quiero seguir aprendiendo Flutter y dedicarme a desarrollar aplicaciones móviles.

Aunque estamos contentos con el resultado y con el proceso de desarrollo, no recomendaría a otro alumno comenzar un proyecto sin un mínimo de experiencia en la tecnología a usar, ya que durante este proyecto tuvimos muchas complicaciones a la hora de tomar elecciones y resolver problemas. A su vez, al ser un framework nuevo, también era desconocido para el tutor. Por otro lado, la cantidad de tutoriales y manuales vas creciendo, pero sigue sin ser un entorno lo suficientemente maduro como para recomendarlo a estudiantes que quieran realizar su proyecto con él.

Trabajo futuro

Han habido muchas ideas que no hemos implementado por falta de tiempo. Algunas de ellas podrían ser:

- Almacenar información de usuario
 - Poder cambiar foto
 - Posición inicial del mapa de un usuario

- Generar eventos de pago (si aplicásemos ese modelo de negocio)
- Mejorar la información de las rutas
 - Clasificación de rutas
 - Información sobre animales, materiales, etc
- Añadir comentarios en las rutas y/o eventos
- Añadir fotos en las rutas y/o eventos
- Modo Offline, usando una base de datos local
- Buscar rutas y/o eventos por posición
- Aplicar [clustering](#) en la vista donde vemos las rutas en un mapa
- Añadir estadísticas sobre las rutas y/o eventos realizados
- Añadir animaciones

Referencias

(2018). Obtenido de Informe Mobile en España y en el Mundo:

<http://mktefa.ditrendia.es/informe-mobile-2018>

García, G. (s.f.). <https://guillermogarcia.es/stateful-y-stateless-widgets/>.

Panko, R. (s.f.). <https://themanifest.com/app-development/popularity-google-maps-trends-navigation-apps-2018>.

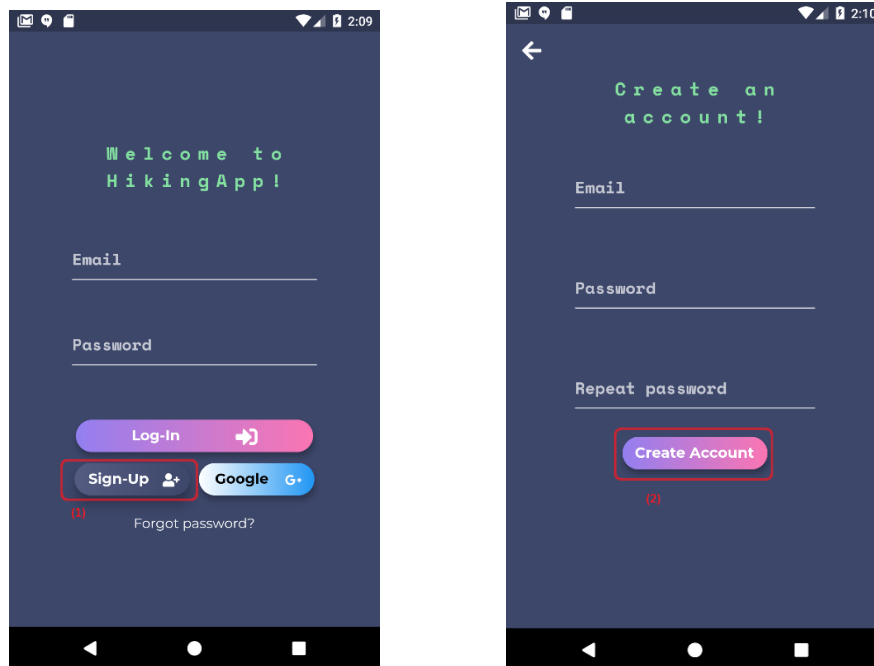
Anexos

Manual de usuario

Sesión

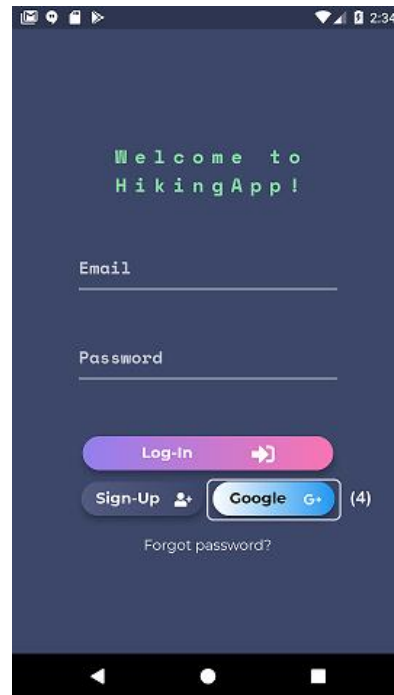
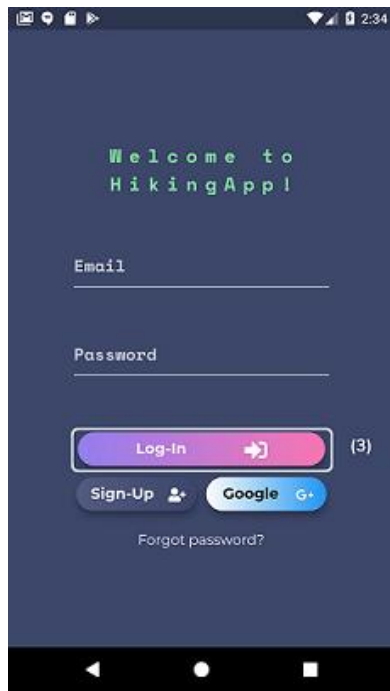
Crear cuenta

Cuando un usuario desee crear una cuenta, deberá pulsar el botón (1) específico, desplazándose a una pantalla nueva donde deberá escribir un correo válido y repetir una contraseña dos veces, que debe ser superior a 6 caracteres. Luego deberá pulsar (2) para validar la información. El usuario iniciará sesión automáticamente si todo ha sido correcto.



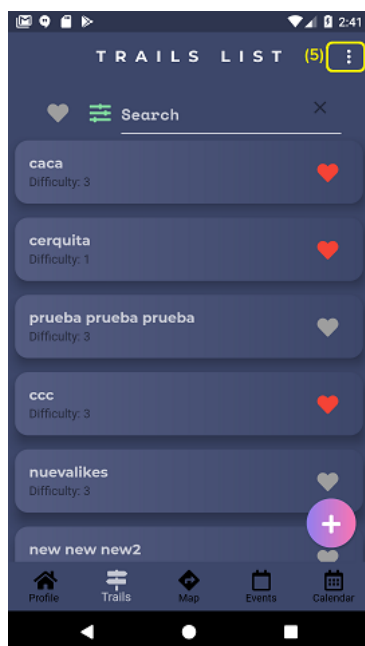
Iniciar sesión

Cuando un usuario desee iniciar sesión, deberá introducir sus datos y pulsar el botón (3). Si el usuario desea iniciar sesión, deberá pulsar (4), elegir su cuenta y finalmente acceder.



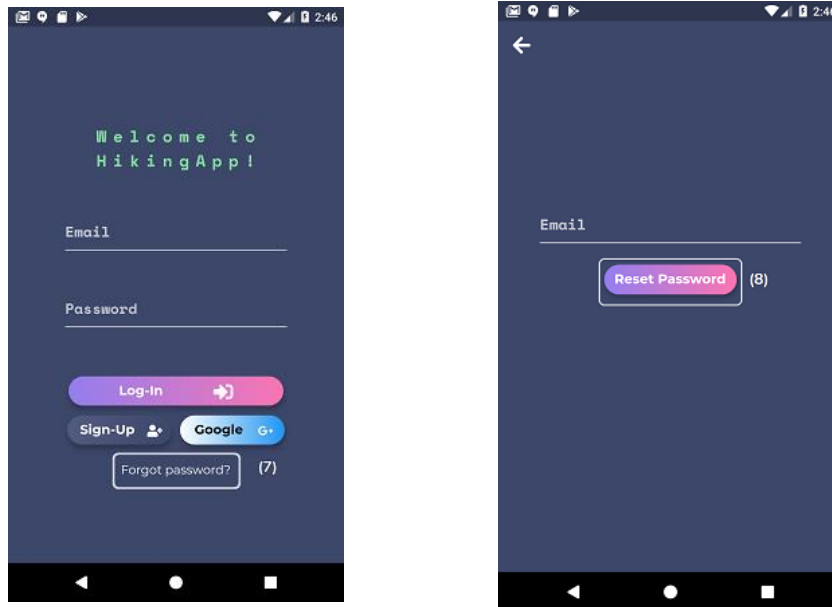
Cerrar sesión

Desde cualquier pantalla, el usuario podrá cerrar sesión, siempre que la haya iniciado anteriormente. Debemos pulsar (5) y a continuación el botón “Sign Out” (6).



Resetear password

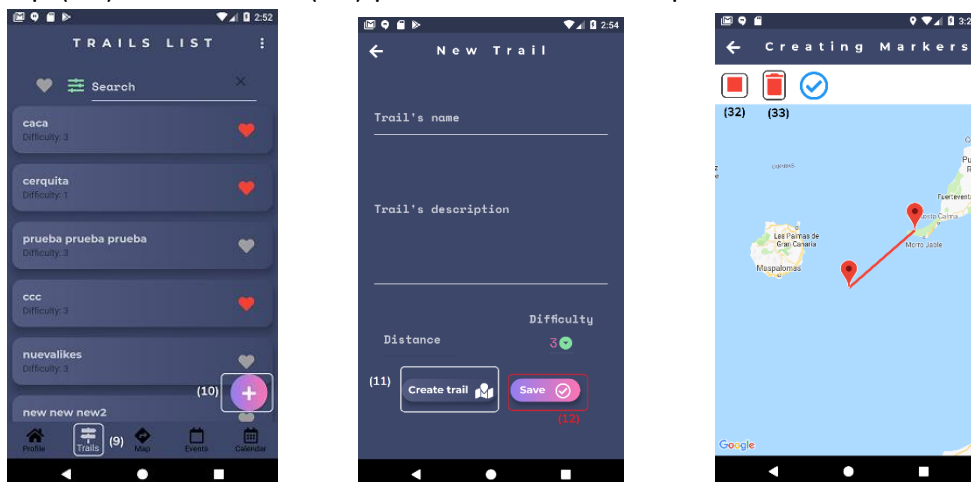
Cuando un usuario no recuerde su contraseña, o simplemente desee cambiarla, deberá usar el botón (7), escribir su correo y pulsar (8). El usuario recibirá un enlace al correo para poder gestionarlo todo.



Rutas

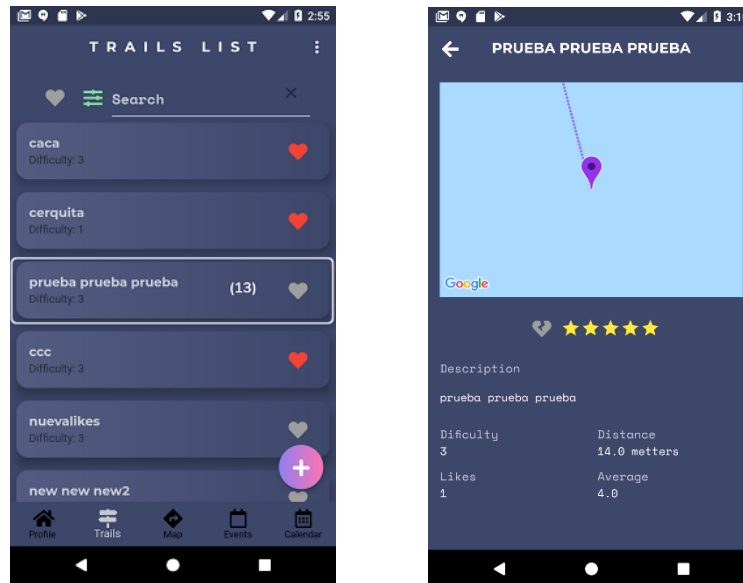
Crear ruta

Para crear una ruta el usuario deberá acceder a la lista de rutas pulsando (9) y posteriormente pulsar el botón (10). Ahora deberá rellenar la información necesaria, seleccionar la dificultad y pulsar (11) para generar la ruta. El usuario podrá crear rutas automáticamente con el modo record, pulsando el botón play o stop (32). Pulsaremos (33) para borrar el último punto añadido.



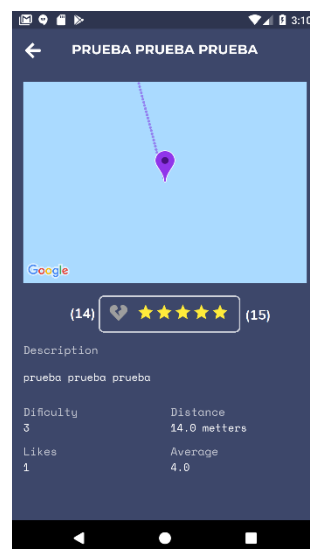
Ver ruta

El usuario que quiera ver la vista detallada de una ruta deberá pulsar en ella (13), en la lista que existe para ello. Esto abrirá la vista detallada de la ruta elegida.



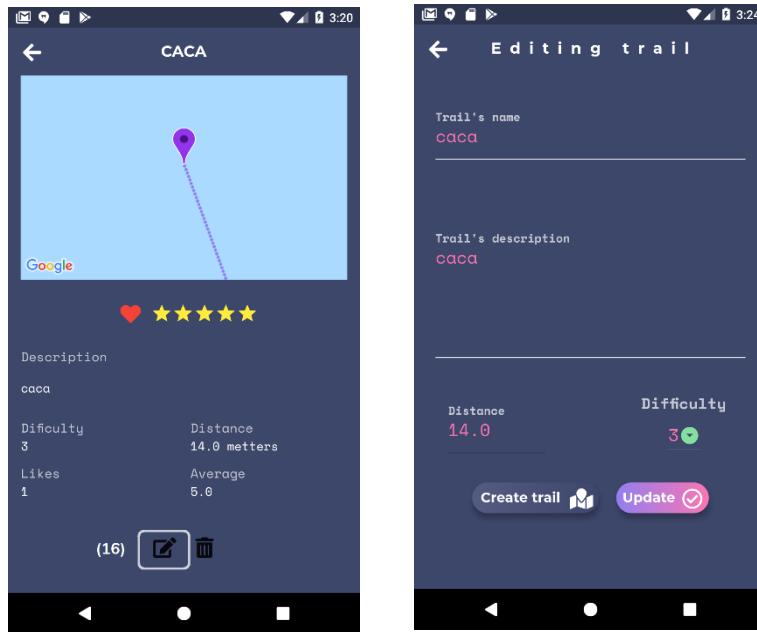
Valorar y favoritos

Cuando el usuario quiera usar estas dos características, deberá pulsar en (14) y/o (15). A su vez, podrá hacerlo desde las dos vistas de listas.



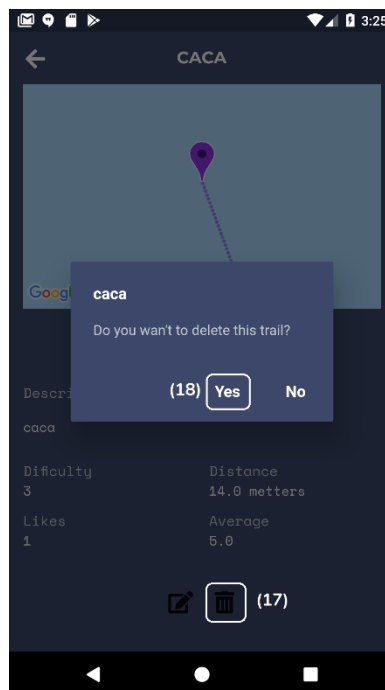
Editar ruta

Para editar una ruta, el usuario deberá acceder a la lista detallada de una ruta que haya creado él. Deberá pulsar el botón (16) y rellenar los datos de la misma manera que lo haríamos si fuera una ruta nueva.



Borrar ruta

Para borrar una ruta debemos pulsar (17) en una vista detallada y posteriormente aceptar la confirmación (18).



Mapa de rutas

Ver ruta

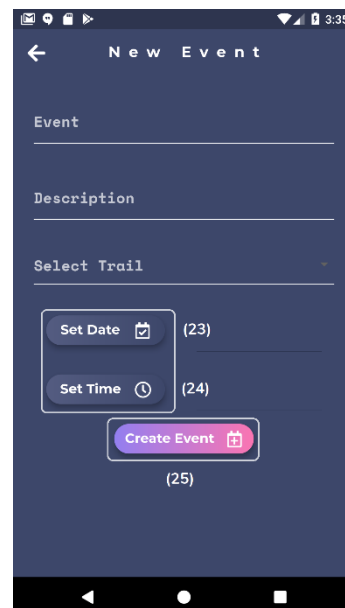
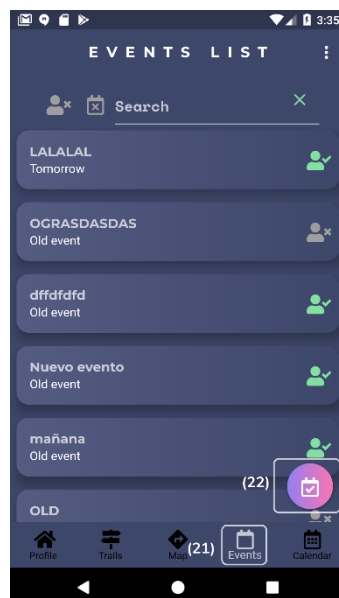
Para ver una ruta en esta vista, debemos pulsar en el marker (19) para que salga la información de la ruta, y posteriormente en (20) para acceder a ella.



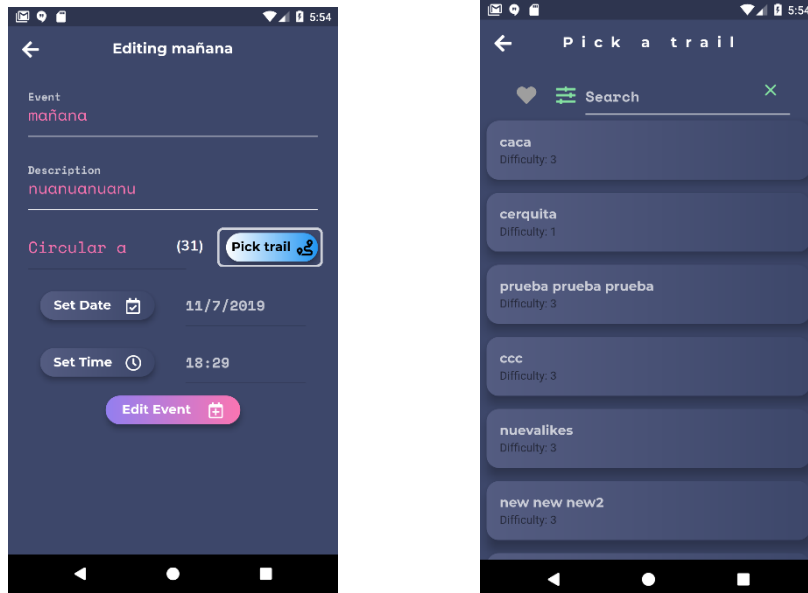
Eventos

Crear evento

Para crear un evento, vamos a la lista de eventos pulsando en (21) y posteriormente (22). Luego debemos rellenar los datos oportunos, pulsando en (23) y (24) para elegir la fecha y la hora. Para finalizar pulsamos en el botón (25).



En la creación de un evento, el usuario debe elegir una ruta para este evento. Eso lo conseguirá pulsando en el botón (31), apareciendo una pantalla con la lista de rutas disponibles, que se podrán filtrar. El usuario deberá ahora pulsar en una ruta para que se seleccione.



Ver evento

Para ver un evento, accedemos a la lista detallada y elegimos el evento a ver, de la misma forma que hicimos con las rutas.

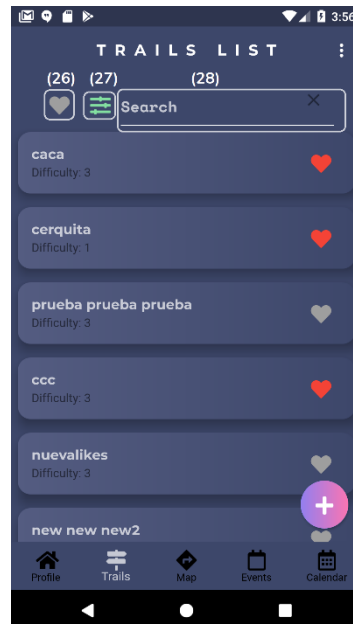
Editar evento y borrar evento

Usamos la misma metodología que con las rutas, ya que usamos un modelo similar en la implementación.

Filtros

Filtros de rutas

Para filtrar las rutas, debemos acceder a la lista detallada, disponiendo ahí de 3 posibilidades. El botón (26) filtra las rutas que tenemos en favoritos, el (27) nos permite elegir la dificultad y el (28) nos deja buscar por texto.



Filtros de eventos

Los filtros para eventos funcionan de forma similar, se sigue pudiendo filtrar por texto, pero ahora filtramos por fecha (29), pudiendo filtrar los eventos antiguos y por participación (30), es decir, si participamos o no en el evento.

