UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y desarrollo de una aplicación móvil para la gestión de actividades socioculturales en un área geográfica

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ponce Gómez, Javier

Tutor/a: García Escriva, José Ramón

CURSO ACADÉMICO: 2022/2023

Agradecimientos

En primer lugar quiero agradecer la gran ayuda y dedicación de tiempo a Jose Ramón García Escriva, quién me ha guiado para construir y dar forma a mi proyecto, y convertir mi idea ambiciosa en un trabajo de fin de grado. Sin ti no habría sido posible.

No puedo imaginar estos agradecimientos sin mencionar a mis padres y mi hermana que siempre me dieron todo lo que necesité y buscaban la forma de darme más. Siempre han confiado en mí y me han dado la oportunidad de desarrollarme profesional y personalmente. Ellos son verdaderamente responsables de que obtenga este título.

Gracias también a Rodrigo , mi amigo incondicional y antiguo compañero de trabajo. Nunca podré dejar de agradecerle todo lo que ha hecho por mi, la ayuda y el apoyo que me brindó fue indispensable para este proyecto. Mencionar a David Sancho y Sergio Martínez, que sin duda han sido el mejor descubrimiento que me podía ofrecer la universidad, y prácticamente mis mejores amigos. Sin ellos nunca habría crecido tanto personalmente, y estoy seguro de que este periodo universitario no hubiera sido tan fantástico. Ojalá poder compartir más años a su lado.

Gracias también a Liying, quién me ha ayudado a encontrar la motivación cuando más me costaba, y a seguir adelante y no rendirme.

Resumen

En este proyecto se pretende desarrollar una aplicación que permita la gestión de eventos: crear planes, consultar horarios, encontrar eventos a una distancia cercana, mostrar cuándo se acerca una actividad y la reserva de tickets para ésta. Además, contará con apartados para la gestión de usuarios y administración de preferencias en base a los gustos del usuario. Se trata de una app multiplataforma para dispositivos móviles utilizando un único *framework*. Todo ello conectado con un *backend* para la gestión de la base de datos y servicios de la misma. El desarrollo del proyecto lleva por tanto intrínseco el análisis del proceso de diseño de la aplicación, desde la aparición de su idea, hasta su planificación, desarrollo, diseño de la interfaz, pruebas, correcciones...

Abstract

This project aims to develop an application that allows event management: create plans, check schedules, find events at a close distance, show when an activity is coming and reserve tickets for it. In addition, it will have sections for user management and administration of preferences based on the user's preferences. It will be a multiplatform app for mobile devices using a single framework. Everything will be connected to a backend in order to manage all the services and the database of the application. The development of the project therefore carries an intrinsic analysis of the design process of the application, from the appearance of its idea, to its planning, development, interface design, tests, corrections...

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

En la actualidad, el mercado de las aplicaciones móviles se encuentra en un momento de auge. Gracias a la aparición y descubrimiento de nuevas tecnologías para el desarrollo de apps móviles como nuevos lenguajes, marcos de desarrollo, sistemas *No-code* o inteligencias artificiales (Apphive o ChatGPT entre otras), se ha producido una adaptación del mercado a esta oferta tan variada.

Por ello, multiples empresas dedicadas principalmente al desarrollo web, han comenzado a optar por alternativas para móviles de los principales sistemas (*Android* e *IOS*) para, de esta manera, poder llegar a más público mejorando la accesibilidad a sus servicios.

Con esto en cuenta, también se han creado numerosas herramientas de todo tipo. En este caso, el foco lo situamos en las que nos permiten fomentar el ocio. Véase redes sociales con todo tipo de intenciones o simplemente herramientas de creación y gestión de eventos socioculturales.

Así pues, manteniendo esta corriente, se presenta un proyecto de diseño y desarrollo de una aplicación móvil que permita la gestión de eventos en una determinada área geográfica en forma de *Minimun Valuable Product* (MVP) .Este MVP consta de un sitema que, posibilita tanto la creación de determinadas actividades llamadas "planes", gratuitas o de pago, como las inscripción y gestión de las mismas por parte de todos los tipos de usuarios de la app. Todas las actividades están pensadas para ser creadas y llevadas a cabo en cualquier área geográfica del mundo que se desee.

El nombre de la aplicación es ***FunWorld*** y en ella introducimos dos tipos de usuarios:

- Por un lado se aprecian los 'users' o usuarios normales . Estos pueden ver una serie de planes adaptados perfectamente a su area geográfica. Así podrán adquirir tickets para estos y verlos dentro de la propia aplicación. Podrán realizar diversas acciones, entre ellas crear perfiles y editarlos a su gusto.
- Por otro lado, para publicar estos planes existe el usuario del tipo 'creador'. Este usuario ha adquirido el rol mediante un permiso otorgado directamente por los propios desarrolladores de la aplicación. Estos usuarios pueden crear planes, modificarlos y realizar diversas acciones sin perder las propias de los usuarios normales. Incluso, se pretende que la aplicación procese directamente estos pagos y, tras una comisión, le envie las ganancias al creador del plan mediante cuenta bancaria.

Para poder esclarecer las diferencias entre estos tipos de usuarios, se muestra la Figura 1.1

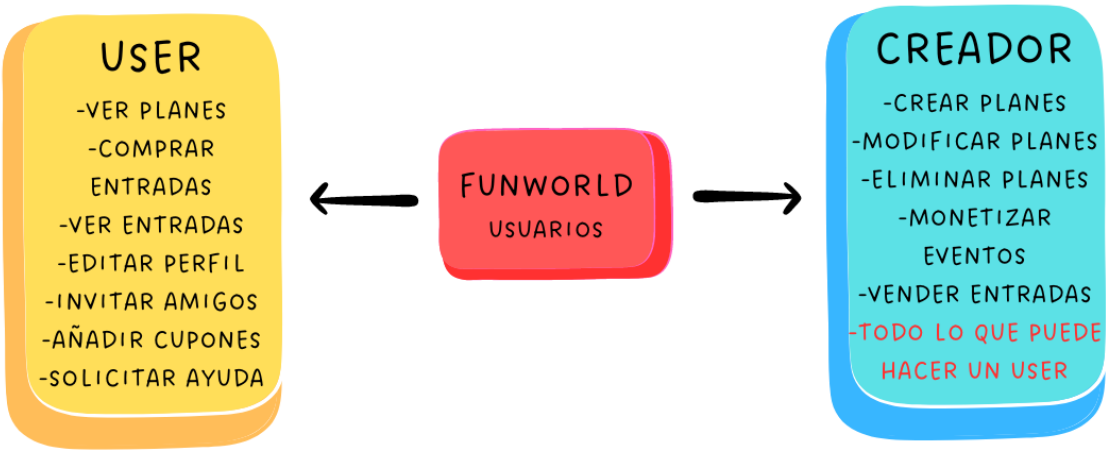


Figura 1.1: Tipos de usuarios.

Añadir que, esta aplicación trata de generar una mecánica nueva en el mercado de la gestión de eventos. Se permite para ello, la creación de los planes gratuitos. Estos tienen especial tratamiento debido a que, además de su atractivo natural por no tener coste, resultan muy interesantes en cuanto a *marketing* se refiere.

Se pretende, con la creación de estos, conseguir un tráfico amplio de creadores de planes y eventos para formar una red de usuarios orgánica y estable. Incluso, aumentar las intereacciones sociales. Además, son mencionados en futuras páginas los diversos avances que se plantean realizar en la aplicación próximamente: establecer una dinámica de red social, posibilidades de interacción entre usuarios.... Con ello se clarifica que este proyecto es todavía un prototipo.

Aun así, es necesario mencionar que, tras hablar con varias asociaciones ESN (*Erasmus Student Network*) se percibe como les interesaría además contar con una plataforma que les permitiera alojar sus planes gratuitos y fomentarlos mediante la aplicacion. Esta dinámica permitiría una entrada de tráfico significativa por esta via ya que los jóvenes son el público objetivo de la app.

1.1 Motivación

La motivación de este proyecto de fin de grado surge de una combinación entre gustos personales y una necesidad encontrada durante mi periodo de Erasmus en Finlandia.

En este país, el clima afecta negativamente a la población tanto de forma física como mental. Ello repercute en que, por lo general, sólo las organizaciones de estudiantes realizan algún tipo de actividad exclusiva para gente Erasmus. Esto implica que no se desarrolle prácticamente ninguna actividad social, véase espectáculos o conciertos.

Obviando los estudiantes, el resto de las personas pasa los inviernos en sus casas y su círculo social es bastante cerrado. De hecho, las actividades que más se realizan en Finlandia suelen tener una relación directa con el ejercicio físico. De esta manera se ve como uno de los países con mejor salud física, donde más se invierte en educación y en el fomento del deporte en general [11].

Indagando en los estudiantes de intercambio, se descubrió que era muy difícil conocer los detalles de estos pocos eventos sociales que se iban realizando, debido en gran parte al uso único del finlandés para comunicarlo y a la falta de una plataforma o grupo para transmitir la información. Todo esto dificultaba la integración en el propio páis. Pues era complicado conocer ciertas actividades si no era por mediación de un amigo que conocía a una persona X.

Posteriormente, se empezó a comprobar cómo el ser humano, un "ser social", se está alejando cada vez más de este concepto. Es más, la tecnología es vista por muchas personas como un gran enemigo de las interacciones humanas y como la principal responsable de la afirmación superior. Las personas se han adaptado tanto a las nuevas tecnologías que, en ciertas ocasiones prefieren estar en el "mundo digital.ªntes que en el mundo real [15].

Por ello, la motivación estaba creada. La idea de crear un sistema que permitiera incluir ambas ideas en una aplicación era magnífica. Se pretendía construir una aplicación de planes que fomentase la realización de actividades, que la gente descubra planes que le permitan desconectar, conocer a gente o simplemente salir a la calle. Una app que permita conectar para a su vez 'desconectar'.

El objetivo es conseguir una aplicación que ayude al ser humano a mejorar las relaciones interpersonales. Una aplicación que permita a las personas encontrarse en ambientes de todo tipo junto a personas de todo tipo, esten en el lugar que estén. En fin, que se le permita descubrir todo un mundo de posibilidades.

De ahí nació ***FunWorld***, este proyecto.

1.2 Objetivos

El objetivo general de este proyecto, como se ha comentado con anterioridad, es el diseño y desarrollo de una aplicación móvil para la gestión de actividades socioculturales en un área geográfica.

Por ello, esta aplicación propone una solución software para: permitir a usuarios acceder a todo tipo de planes en su zona, conocer a nuevas personas y compartir experiencias con amigos.

Esta aplicación contará con múltiples funcionalidades para poder cumplir con los requerimientos del proyecto. Para ello, se han dividido los objetivos en dos apartados: los principales y los secundarios. En cuanto a los objetivos principales, estos se reflejarán directamente en la aplicación y su propio funcionamiento. Por otro lado, los objetivos secundarios son más personales y están enfocados a tareas que se deben ir cumpliendo con el desarrollo de la app.

En consecuencia, para asegurar el adecuado progreso y funcionamiento de la aplicación, es necesario satisfacer una serie de requisitos técnicos distintos entre sí:

- Explorar y crear una funcionalidad integrada en la aplicación móvil que permita a los usuarios generar e inscribirse en planes utilizando dispositivos móviles y priorizando una experiencia de usuario óptima.
- Implementar un *backend* que complemente las funciones de la aplicación, utilizando servicios en la nube como una base de datos, sincronización, autenticación de usuarios, enlaces dinámicos y almacenamiento de archivos.

- Redactar las reglas de seguridad pertinentes dentro de ***Firebase*** con tal de crear un sistema seguro en el que los datos no puedan ser extraidos o manipulados sin consentimineto alguno.
- Aplicar los principios de *Clean Architecture* (Arquitectura limpia) durante el proceso de desarrollo, utilizando patrones de diseño y realizando pruebas exhaustivas para facilitar el mantenimiento a largo plazo.
- Definir y desarrollar un Producto Mínimo Viable (MVP, por sus siglas en inglés) que permita a los usuarios utilizar las principales funciones de la aplicación en su primera versión.

Por consiguiente, hay requisitos ciertamente personales que se van cumpliendo progresivamente durante todo el desarrollo del proyecto, como:

- Profundizar en el desarrollo de aplicaciones móviles, desde el inicio hasta la finalización, basándose en buenas prácticas y recomendaciones propuestas por artículos, documentación y comunidades de desarrolladores de renombre.
- Comprender y adentrarse en el vasto mundo del marketing de aplicaciones y las diferentes estrategias de negocio dentro de los mismos

1.3 Estructura de la memoria

La memoria cuenta con varios apartados que tratan de explicar con detalle el proceso seguido en la creación y desarrollo del proyecto.

El capítulo 2 se enfoca en el estado del arte, donde se realiza una labor de estudio de aquellas soluciones relacionadas con las aplicaciones de gestión de eventos dentro del mundo digital. Así, se profundiza en aplicaciones similares y se comparan con la propuesta.

En el capítulo 3 se trata el análisis del problema. En este se realiza un estudio exhaustivo sobre el proyecto y sus requisitos, detallando los aspectos más importantes del estado del arte y centrándose en el proyecto a realizar para comprender mejor su carácter diferenciador.

Posteriormente, en el capítulo 4, diseño de la solución, se proporciona información técnica, describiendo las tecnologías utilizadas a través de distintas formas como textos y diagramas de la solución. Así se muestran las decisiones tomadas a a nivel de arquitectura, patrones de diseño, y tecnologías empleadas.

En cuanto al capítulo 5, desarrollo de la solución propuesta, se abordan los diferentes componentes de la app, su comportamiento y funcionamiento interno de cada uno de ellos, adentrándose en los aspectos técnicos e ilustrándolos mediante el uso de *Clean Architecture*.

En lo que respecta a la fase de implantación (capítulo 6), se trata de uno de los últimos procedimientos del proyecto, su puesta en práctica. Este apartado detalla un ejemplo de uso correcto de la aplicación junto con la explicación de todos los procesos que se van realizando dentro de la misma. Incluso se anexan las partes más relevantes del código.

En el capítulo 7 de pruebas, se revisa la implementación y se detallan los procedimientos llevados a cabo para garantizar que todo se haya realizado de manera satisfactoria.

Así pues se identifican errores y posibles fallos de ejecución y se tratan de la forma correspondiente.

Como último de los principales capítulos, se muestra la conclusión (capítulo 8). Los aspectos y valoraciones derivados de la realización de este trabajo y las mejoras que se implementarán en futuras versiones del mismo se encuentran contempladas en este apartado.

Finalmente, se encuentra una sección de anexos. Esta sección se debe a que hay cierta información que no ha podido ser plasmada en el documento, ya que se tratan aspectos muy específicos acerca de algunas configuraciones del sistema que no encajan dentro de una estructura definida.

Por este motivo, se han dejado al final del documento estos anexos con información para hacer la lectura lo más interesante posible y para el posible interés de todo público.

CAPÍTULO 2
Estado del Arte

El mundo ha sufrido grandes cambios en los últimos años. En lo que a tecnología respecta, se ha podido ver un gran boom de las redes sociales y las plataformas para ligar, conocer a gente, vender entradas, etc.

Con ello la sociedad ha cambiado de forma muy drástica y se ha envuelto a las personas en un ambiente de digitalización que, si bien nos ha permitido evolucionar en algunos aspectos, nos ha sometido a ciertas circunstancias que podríamos calificar como negativas.

Por ello, se trata la creación de una aplicación que pueda ayudar a solucionar esta situación mediante la mejora de las relaciones sociales y la promoción de eventos que permitan a las personas mejorar sus calidades de vida.

Así pues, en este capítulo se analizarán diferentes aplicaciones relacionadas con la creación de planes y el entretenimiento. Con ello, no solo se valorarán las oportunidades que ofrecen sino que también se podrá comparar la propuesta con respecto a las demás del mercado.

Posteriormente se hará un análisis de las tecnologías utilizadas para realización de este proyecto incluyendo una justificación de su uso para comprender mejor porque se han tomado las decisiones pertinentes.

También se mostrará una proyección de futuro de la propia aplicación con los posteriores objetos de desarrollo y *sprints* de la misma.

2.1 Crítica al estado del arte

Para la investigación sobre aplicaciones similares, primero se buscan los títulos con más descargas de ambas tiendas de aplicaciones Android e iOS y que, de una u otra forma, tengan como idea principal enlazar a los usuarios con planes de todo tipo en una determinada ubicación. Esto permite cribar y descubrir en qué apartados estamos se está innovando y de qué manera se está abordando el problema.

Esta lista de aplicaciones se puede hacer muy larga. Sin embargo, se va a buscar aquellas con más éxito en la industria, para analizar el por qué de este éxito. Incluso, se intentará darle una vuelta más a la facilidad de acceso a estas. Además, es de notar que el problema principal de estas, de cara a la escalabilidad del número de clientes, es la baja afluencia de usuarios que tienen en ciertas zonas geográficas. Claro que es posible comparar la cantidad de usuarios que hacen planes en la calle si estos son de España o por lo

contrario son de Alaska. Por tanto, esto da lugar a que algunas aplicaciones no destaquen tanto como deberían.

Con todo este estudio previo, se pretende centrar los esfuerzos del desarrollo en una dirección concreta. Saber de antemano que características debe cumplir el MVP propuesto , y como se van a lograr. Es esencial hacer una inversión de tiempo óptima para avanzar desde el principio en la dirección correcta.

La primera aplicación que se puede venir a la cabeza residiendo en España es **Fever**. Esta aplicación es la más relacionada con nuestro propósito y la más descargada de Google Play de este país en la categoría Eventos. La mecánica de esta aplicación es bastante intuitiva y clara. Una vez el usuario se registra y da acceso a su ubicación aparecen en la ventana principal varios planes para hacer. Dentro se puede comprar tickets, unirse a grupos y buscar otros eventos. El inconveniente principal de esta aplicación es que lo planes se reducen exclusivamente a gente que paga para posicionarlos y son empresas quienes los crean. Es más, no permite crear planes gratuitos. Por tanto esto no permitiría escalarlo a situaciones diversas como las mencionadas arriba de estudiantes de intercambio, colectivos minoritarios, etc. Así pues, se ve que es una aplicación que sirve como medio publicitario y que además cobra altas comisiones para crear cierto tipo de planes 'exclusivos'. A pesar de ello, es la que más se asemeja a la idea establecida.

Posteriormente se destaca otra aplicación que se encuentra en el top 15 aplicaciones más descargadas en la categoría de "Citas" en App Store y Google Play. Se trata de **Bumble: Ligar, citas, amigos**. Una aplicación que parte de un concepto interesante, el de ofrecer una plataforma válida tanto para encontrar pareja, como para hacer amigos e incluso para hacer trabajos y hacerlos en un ambiente amable y respetuoso. La aplicación tiene un buscador con el que se puede encontrar el perfil de una persona afín a los objetivos que tengas (tanto a nivel romántico como a nivel puramente social). Es una aplicación que, si bien trata un concepto relacionado con el proyecto realizado, se adapta más al perfil de una aplicación de citas para conocer a personas.

Es necesario hacer un inciso y es que cabe destacar que la gran mayoría de aplicaciones de esta categoría van relativamente enfocadas a conocer a personas afines para tener citas con ellas y otro tipo de planes. Esto da la confianza de que se trata de un nicho todavía muy grande para explorar. Sin embargo, dentro de este universo, es necesario destacar la aplicación **Skout** . Esta nació para hacer frente a redes sociales de talla gigante como el mismísimo Facebook o Instagram. Su funcionamiento es prácticamente igual a la creación de Zuckerberg: existen personas afiliadas a la red social de todas las partes del mundo y puedes agregar a quien conozcas o quieras conocer, siendo posible incluso hacer eventos en *live* . Además permite crear horarios con tus amigos para hacer planes con estos. Esto puede ser interesante en el caso de que quieras aprovechar y conocer personas nuevas o simplemente utilizar una red social más. En todo caso tampoco tiene una semejanza tal a la idea que se tiene en mente.

Aunque, como ya se ha mencionado, en su mayoría, prácticamente ninguna aplicación cumple al cien por ciento con la idea que se tiene en mente. Todas derivan hacia el sector de las aplicaciones de citas o al de redes sociales más tradicionales. Esto supone una gran ventaja competitiva que se puede explorar y explotar. Pues, si bien es cierto, que otras aplicaciones como lo son **Happn** o **Meetup** empezaron con la misma idea, al final derivaron a apps de citas.

En la figura 2.1 se muestran todas los aplicaciones mencionadas y el lugar que ocupan en las tiendas de aplicaciones a fecha 5/05/2023.

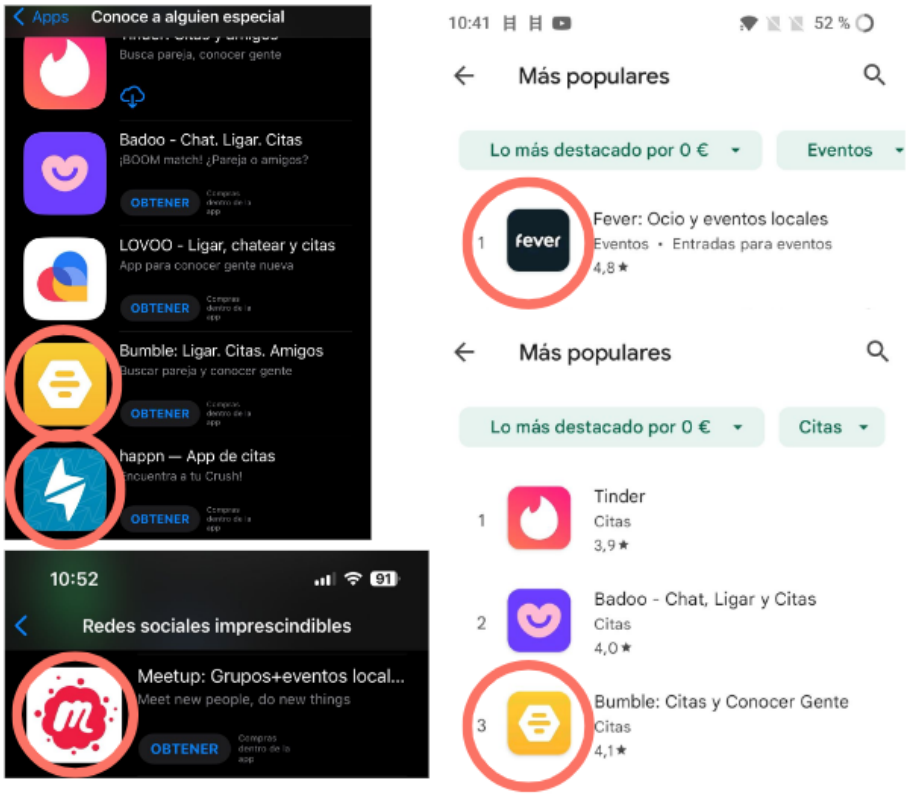


Figura 2.1: Apps analizadas con su respectiva posición en el ranking *Estado del Arte*.

El análisis de todas estas aplicaciones, así como las referencias que podemos extraer de otros diversas de contextos similares que incluyen una mecánica principal basada en la creación de eventos y la socialización, nos permiten crear un producto de calidad y que pueda generar un impacto real en las personas que se sumerjen en ella.

A continuación, se muestra una tabla representativa de cada una de las características estudiadas, donde se objetarán su final inclusión en la aplicación propuesta:

Característica encontrada	Fever	Bumble	Skout	Happn	Meetup
Aplicación multiplataforma	✓	✓	✓	✓	✓
Creación de planes	✓	X	X	X	X
Precio gratuito	✓*	✓*	✓*	✓*	✓*
Actividades gratuitas	X	✓	✓	✓	✓
Edición de perfil	✓	✓	X	✓	X
Eventos especiales	✓	X	✓	X	✓
Sistema de referidos	✓	X	X	X	X
Multilenguaje	✓	✓	✓	X	✓
Uso global	✓	✓	✓	✓	✓

Figura 2.2: Comparativa de características entre aplicaciones de interés y su inclusión como requisito.

La simbología siguiente se utiliza para indicar la inclusión de las características de la Tabla 2.2:

- ✓: Característica incluida en el aplicativo.
- X : Característica no incluida en el aplicativo.
- ✓* : Característica incluida en el aplicativo con ciertos matices.

Como conclusión, se pueden extraer varios puntos clave de cada una de las aplicaciones analizadas. Se considera, por tanto, que estén presentes en el resultado final del proyecto. La principal característica de todos ellos es que consiguen, cada uno con su mecánica propia, que conocer gente y salir de la zona de confort sea más fácil y tangible si se desea. Tratan de crear una mecánica de uso simple, y la explotan hasta el límite, poniendo al cliente ante un reto novedoso e interesante como es el de apuntarse a planes ya sea con amigos o con gente desconocida.

Aunque parezca algo obvio, hay muchas aplicaciones que, a pesar de tener un gran potencial, no consiguen captar la atención y retener al usuario debido a que, alomejor no hay suficientes creadores, o se encuentran en una zona donde no hay prácticamente ninguno. Todas estas aplicaciones tienen en común un intento de crear algo similar a esto que acaba derivando en un interés cada día más potenciado por la tecnología y la sociedad como es tener pareja.

Sin embargo esta aplicación es mucho más escalable. El poder salir de la zona de confort, crear una red que permita exponerse y experimentar esa adrenalina de conocer gente nueva, sin necesidad alguna de querer ligar o incluso de gastar dinero. Eso es lo que se busca. Esto es algo que se tendrá muy en cuenta en el diseño visual e interno de la app, pues se desea que sea tan fácil de usar y de entender que todo el mundo se sienta cómodo con ella. Esto fomentará que la gente quiera actuar y disfrutar de planes.

2.2 Propuesta

Una vez analizados los referentes de la industria para definir correctamente la dirección del proyecto, es la hora de elegir con que tecnologías se va a desarrollar el mismo. La elección del software, formatos de archivos, lenguaje de programación y demás herramientas de desarrollo es esencial para el correcto desarrollo del proyecto, que podría cambiar de forma radical dependiendo de lo que elijamos.

Cada día se lanzan al mercado cientos de aplicaciones con distintos propósitos y funcionalidades. Por tanto, es difícil hacer algo sorprendente e innovador en este campo sólo usando elementos cotidianos y conocidos en todo el mundo. Es por ello por lo que se plantea utilizar ciertas herramientas para crear algo verdaderamente práctico. Pues la importancia de este proyecto es resolver un problema. Si además se consigue que esta resolución se atractiva y de buenos resultados, el propósito se habrá alcanzado.

Para ello se necesita conocer el criterio de elección a valorar a la hora de elegir las herramientas con las que construir el proyecto.

Para la construcción de la Interfaz de Usuario (IU) de nuestra aplicación multiplataforma se ha decidido basar la elección en la búsqueda de frameworks gratuitos, que se encuentren en *momemtum* (cada vez son más usadas y con mayor popularidad), que permitan la implementación de widgets y que contengan documentación suficiente como para empezar a formarse al respecto una vez se haya realizado la elección.

En cuanto a los criterios de selección de la aplicación de gestión del backend. La elección se ha basado en la búsqueda de aplicaciones que tengan también una opción gratuita (pues no es necesaria realmente una gran cantidad de almacenamiento en la nube), que permitan una cierta escalabilidad a largo plazo (a la hora de aumentar el almacenamiento y el número de usuarios de esta), una integración cómoda y compatible con el framework que se ha seleccionado.

2.2.1. Framework de desarrollo

Actualmente, y a pesar de que existen muchos otros lenguajes de programación para desarrollo de aplicaciones multiplataforma, los más usados de la industria son React Native , Flutter y Xamarin según señalan los expertos de *TechnoStacks* , una de las empresas de desarrollo de software con más nombre en el mercado asiático.(artículo disponible en [18])

Tras la revisión de varios foros tecnológicos de renombre se vislumbra una comparativa realizada por *Nomtek* en el año 2021 y actualizada en 2022 entre React Native y *Flutter* (artículo disponible en [12]). En esta comparativa podemos ver claramente como *Flutter* se ha ido superponiendo a React y ganando peso como una de las tecnologías más utilizadas del mercado.

Además, según se aprecia en la siguiente imagen, existe una estadística global realizada por el portal *Statista* (portal estadístico de gran prestigio) que sitúa a *Flutter* junto a React Native como la plataforma de desarrollo multiplataforma más utilizada entre 2019 y 2021 (artículo disponible en [10]).

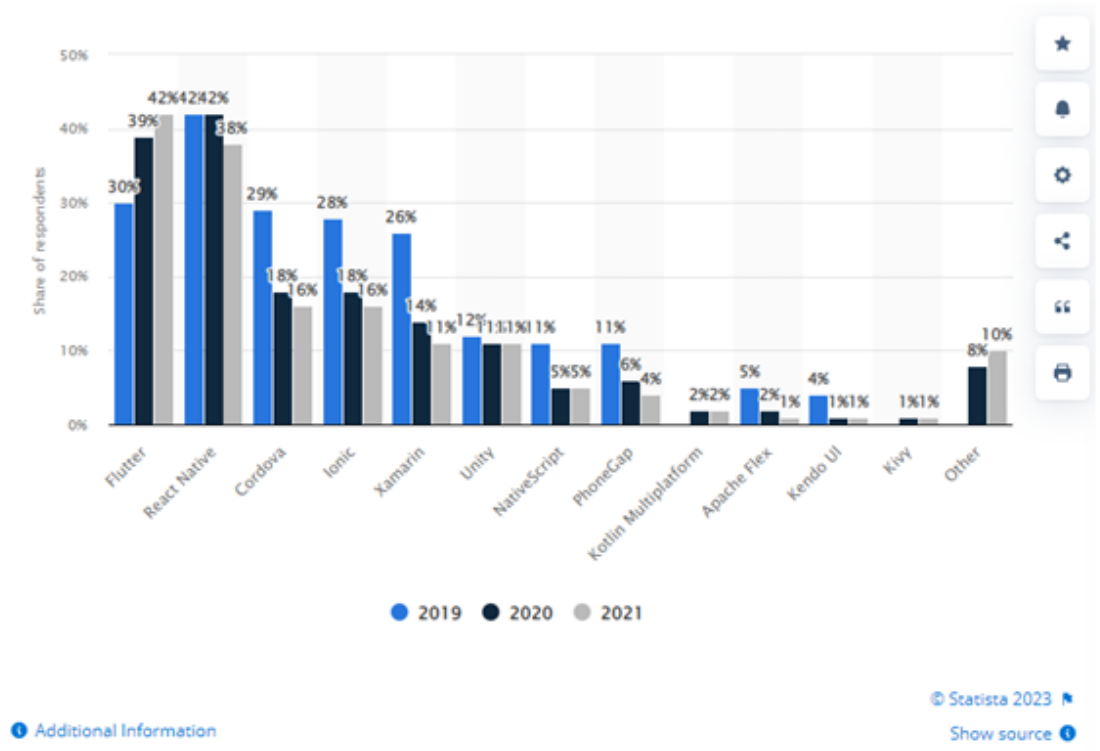


Figura 2.3: Estasítica realizada por el portal *Statista*.

Se puede ver como hay varias encuestas realizadas, de nuevo, por el portal *Statista* a miles de programadores senior que muestran como *Flutter* es el más preciado por los mismos (artículo disponible en [7]).

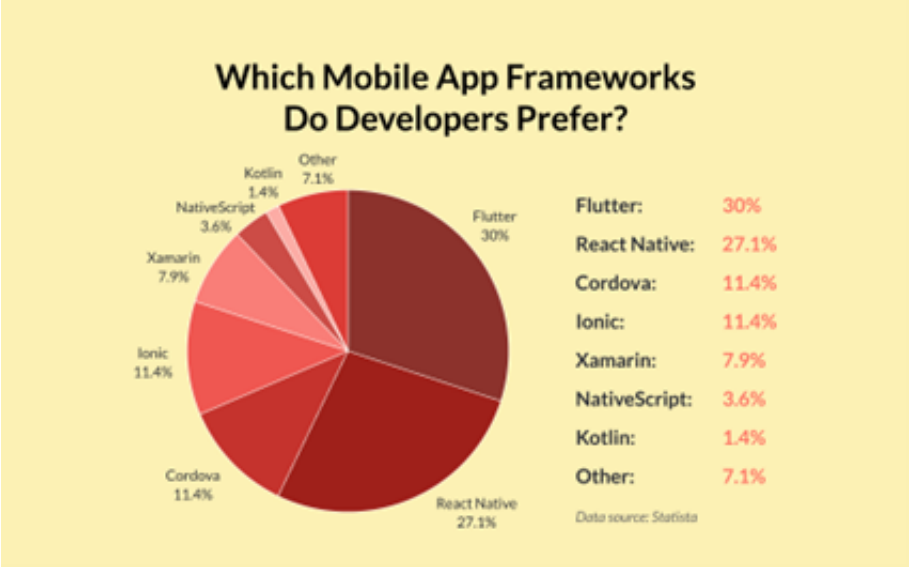


Figura 2.4: Estasítica realizada por el portal *Statista*.

Incluso se investigó en un post de los expertos arquitectos de software de *Kellton Tech* (un blog con muy buena reputación Norte América). Se puede ver como ellos mismos posicionan *Flutter* como el mejor framework de desarrollo para apps móviles en el año 2022 tras analizar varias encuestas y datos de *Github* y *Stack Overflow* (artículo disponible en [2]).

Por tanto, para llevar a cabo el desarrollo de esta aplicación multiplataforma se ha decidido utilizar *Flutter*. Esto se debe a que considero que es el framework que mejor se adapta a los requisitos establecidos previamente. *Flutter* es el framework multiplataforma para crear aplicaciones iOS y Android con una misma base de código mejor valorado en la actualidad (precio, momento, valoración y soporte).

2.2.2. Gestión del backend

En cuanto a la gestión del backend, el propósito es el de encontrar aplicaciones Backend-As-A-Service que permitan gestionar la App desde el punto de vista correcto y en base a las condiciones impuestas en la parte superior de este capítulo.

En base a esta premisa se ha realizado una exhaustiva investigación para descubrir cuales son las plataformas más utilizadas para crear un servicio de backend. Son muchos los servicios que se ofrecen para este tipo de necesidades. Por tanto, se ha indagado un poco más en lo que se puede ajustar mejor a la idea de la aplicación y a las condiciones.

Con ello, tras ver varios post como el de *Back4app* [3] o el de *Hitechnectar* [4] y analizar los pros y contras de las diversas opciones disponibles en el mercado, la favorita es Firebase .

Influyeron varias comparativas encontradas por Internet como las de *Capterra* [5]. En ella se descubrió como Firebase era una opción más completa que ofrecía un plan gratuito de 10GB que, si bien no era el mejor del mercado, permitía una integración muy cómoda con *Flutter* . Además se valoró que permitía una escalabilidad muy alta ya que, como se comentó arriba, se buscaba encontrar un servicio que prescindiera de servidor. En el post de *Rlogical* [1] se encuentran también mencionados muchos puntos positivos al respecto. Este blog es muy mencionado en diversos hilos de *Reddit* y *Stack Overflow* por programadores de renombre.

Finalmente se realizó una investigación a través de un post de *Techugo* que ejemplificaba porqué *Firebase* era una de las mejores opciones para el desarrollo de mi aplicación multiplataforma [25]. Esta permitió comprobar la gran cantidad de facilidades de uso de la misma y la gran cantidad de documentación que había en internet respecto a cómo enlazarla con *Flutter*.

Por último, destacar que aplicaciones como *Alibaba* , *Trivago* y *The New York Times* usan *Firebase* desde hace varios años y su crecimiento sigue siendo abrumador.

Figura 2.5: Resumen de las herramientas a utilizar.

2.2.3. Conclusión

Por tanto, tras todos los análisis realizados y los datos obtenidos tras las características reflejadas en la Tabla 2.2, se toman las siguientes conclusiones de cara a la creación de la primera versión de la aplicación:

- La aplicación estará disponible en ambas plataformas, Android e iOS; ya que todas la aplicaciones analizadas son multiplataforma.
- La aplicación tendrá como finalidad principal la creación de planes de todo tipo siempre que cumplan con ciertas condiciones.
- La aplicación no tendrá habilitadas compras dentro de la misma aplicación. Se pretende que todo el mundo la pueda utilizar de forma gratuita sin necesidad de hacer un gasto económico.
- Estos planes creados podrán ser totalmente gratuitos si el creador así lo considera. Esto será un diferenciador respecto de la competencia.
- La aplicación permitirá la edición del perfil a gusto del usuario con tal de crear una mayor fiabilidad entre los clientes de la misma.
- Se permitirán la creación de eventos especiales y promociones siempre que sean aceptadas por el equipo directivo de la aplicación.
- La aplicación incorporará un sistema de referidos de forma que se pueda invitar a ciertos usuarios y recibir una bonificación por ello.
- Si bien la aplicación en su versión de MVP se encuentra únicamente en castellano, se pretende que cuando se lance al público incorpore varios idiomas, entre ellos inglés y alemán.

- Se pretende que la aplicación se pueda utilizar en todo el mundo.
- Otras características estudiadas se han considerado no esenciales y quedarán descartadas para la primera versión del producto creado.

CAPÍTULO 3
Análisis del problema

Tras la realización de un análisis exhaustivo de aplicaciones similares a la que se quiere desarrollar, así como de las herramientas software usadas para su desarrollo, es el momento de realizar un análisis del problema desde distintos enfoques.

El proyecto que se pretende realizar es una aplicación multiplataforma que funcione tanto para Android como para iOS, donde los usuarios puedan crear planes y apuntarse a los de otras personas fomentando la realización de actividades sociales.

Se pretende, además, realizar unas interfaces cómodas que permitan acceder a ventanas sencillas para ver planes destacados, buscar planes, ver los tickets adquiridos y los ajustes de los usuarios, permitiendo elegir qué tipo de perfil desea el usuario además de diversas opciones como la introducción de cupones, la compartición de enlaces de referidos o la propia conversión a creador de planes.

3.1 Especificación de requisitos

Para una especificación completa de los requisitos, se seguirá la especificación tradicional formulada por el estándar internacional IEEE Std 830-1998 [16], el cual es ampliamente elegido por la mayoría de los jefes de departamento de software debido a su agilidad en la fase de gestión de los requisitos [17].

A continuación, se presentará el contenido según la norma ISO seleccionada, acompañado de diagramas y mejores prácticas.

3.1.1. Propósito

El propósito de la sección es la definición y formalización de los requerimientos que debe incorporar esta primera versión de la aplicación expuesta. Así se pretende facilitar y guiar el propio desarrollo de la misma.

3.1.2. Ámbito de la solución

El ámbito del presente proyecto, como ya se ha comentado con anterioridad, es el desarrollo de una aplicación móvil multiplataforma mediante el uso del *framework* ***Flutter***, con su lenguaje *Dart* y ***Firebase*** como *backend*.

La app tiene el nombre de ***FunWorld***. Este está compuesto por dos palabras en inglés: *Fun* que significa diversión o divertido en castellano, y *World* que significa mundo. Este juego de palabras con una traducción literal de 'Mundo divertido', nace de la idea de crear planes en cualquier ubicación del mundo con tal de divertirse y mejorar las vidas de los usuarios.

En esta, el usuario podrá además hacer uso de servicios propios y en la nube como serían inicios de sesión, interacción la base de datos, uso de enlaces dinámicos, sincronización automática y un sistema de almacenamiento de datos propios.

3.1.3. Modelo de dominio

Una vez introducido el ámbito y propósito de la solución, se procede a mostrar un diagrama de clases de acuerdo a las reglas básicas del modelado UML.

Así pues, el modelo de dominio de la aplicación se puede visualizar en el siguiente diagrama :

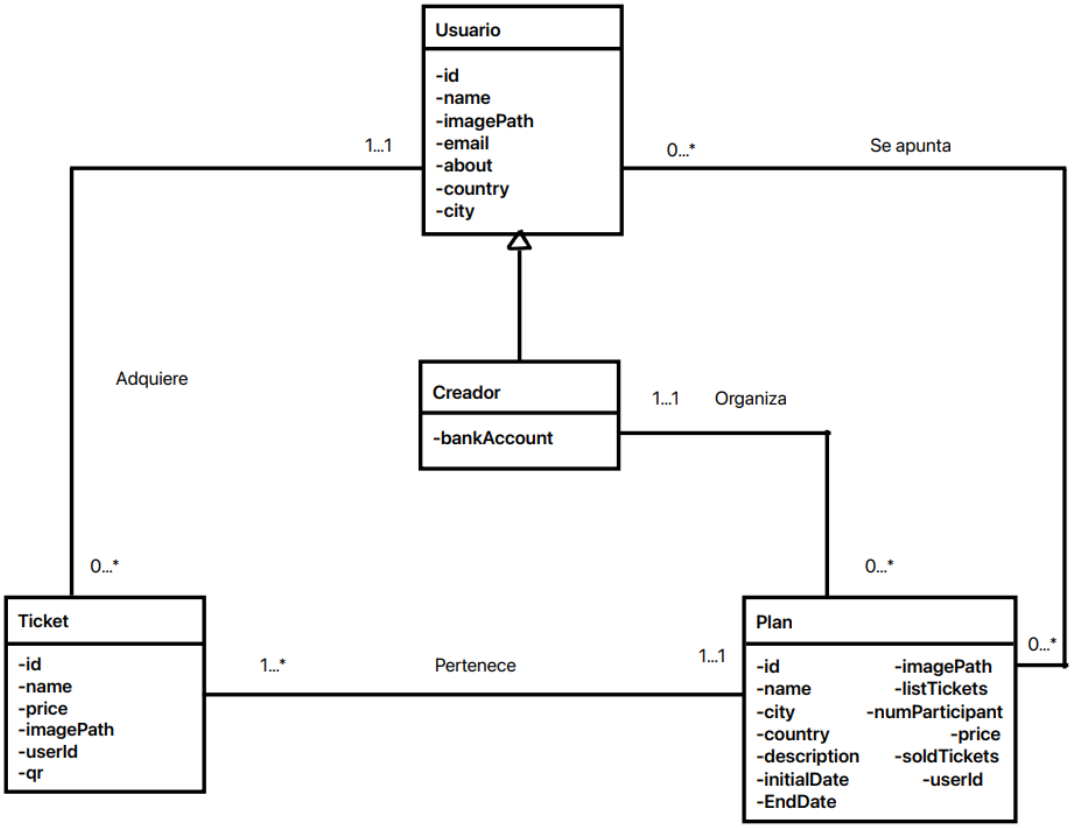


Figura 3.1: Diagrama UML.

A continuación, se anota la funcionalidad de cada clase de concepto asi como las propiedades que los componen y se anotan en la siguientes tablas. Es necesario mencionar que el símbolo ? en un tipo refleja su *nulabilidad* (que puedan contener valores nulos) en el sistema.

- **Clase Usuario:** Conjunto de datos del usuario relacionados con el sistema.

Atributos de Usuario	Descripción
id (*String*)	Cadena de caracteres único que identifica al usuario a nivel interno
name (*String*)	Pseudónimo del usuario en el sistema
imagePath (*String*)	Imagen de perfil del usuario en el sistema
email (*String*)	Cuenta de correo de registro única para el usuario
about (*String?*)	Información del usuario sobre él mismo
country (*String?*)	País donde se encuentra el usuario
city (*String?*)	Ciudad donde se encuentra el usuario
userListTickets (List?)	Lista donde se encuentran todos los tickets adquiridos por el usuario

Figura 3.2: Atributos de la clase usuario.

- **Clase Creador:** Tipo de usuario que puede crear y modificar planes.

Atributos de Creador	Descripción
Todos los atributos pertenecientes a Usuario	-
bankAccount	Cuenta de banco donde el creador recibirá los beneficios de los tickets vendidos

Figura 3.3: Atributos de la clase creador.

- **Clase Plan:** Evento sociocultural.

Atributos de Plan	Descripción
id (*String*)	Cadena de caracteres único que identifica al plan a nivel interno
name (*String*)	Pseudónimo del plan en el sistema
imagePath (*String*)	Imagen del plan en el sistema
listTickets (*List?*)	Lista de todos los tickets vendidos
listUsers (List?)	Lista de usuarios que han adquirido un ticket
numParticipant (Int)	Numero máximo de usuarios que pueden unirse al plan
country (*String*)	País donde se realiza el plan
city (*String*)	Ciudad donde se realiza el plan
Price (*Double*)	Precio del plan
Description (*String*)	Descripción del plan
SoldTickets (*int?*)	Número de tickets vendidos
initialDate (*DateTime*)	Cuando se estima que se realice el plan
endDate (*DateTime*)	Cuando se estima que finalice el plan
userId (*String*)	Identificador del usuario que ha creado el plan

Figura 3.4: Atributos de la clase plan.

- **Clase Ticket:** Entradas del plan al que se quiere acceder.

Atributos de Ticket	Descripción
id (*String*)	Cadena de caracteres único que identifica al ticket a nivel interno
name (*String*)	Pseudónimo del plan al que corresponde el ticket
imagePath (*String*)	Imagen del plan al que se accede con ese ticket
price (*Double*)	Precio que tiene el ticket
userId (*String*)	Id del usuario que adquiere ese ticket
qr (*String*?)	Código que permite acceder al plan se si es requerido

Figura 3.5: Atributos de la clase ticket.

3.1.4. Restricciones del Sistema

En este apartado se pretende mostrar algunas de las restricciones que pueden impedir que varias funcionalidades de la aplicación se ejecuten correctamente.

Se muestran a continuación:

Restricción	Descripción
Memoria del dispositivo	Si el dispositivo no tiene la suficiente memoria disponible para almacenar la aplicación no se podrá utilizar.
Conexión a internet	Se requiere conexión a internet para utilizar todas las funcionalidades en la nube y poder iniciar la aplicación.
Cuenta de Google	El usuario necesita una cuenta de Google para poder iniciar sesión/registrarse en la aplicación.

Figura 3.6: Atributos de la clase ticket.

3.1.5. Características del sistema

A continuación se realizará un estudio de los casos de uso del proyecto. Para ello se utilizarán actores y tablas que permitirán entender mejor la información y transmitir un conocimiento más exacto acerca del funcionamiento de la app.

Por tanto, se procede a mostrar las acciones de mayor importancia en este panel de control. Así pues, se tienen en cuenta el punto de vista del usuario, del creador y del propio sistema.

En la siguiente página se muestra el diagrama de casos de uso de la aplicación.

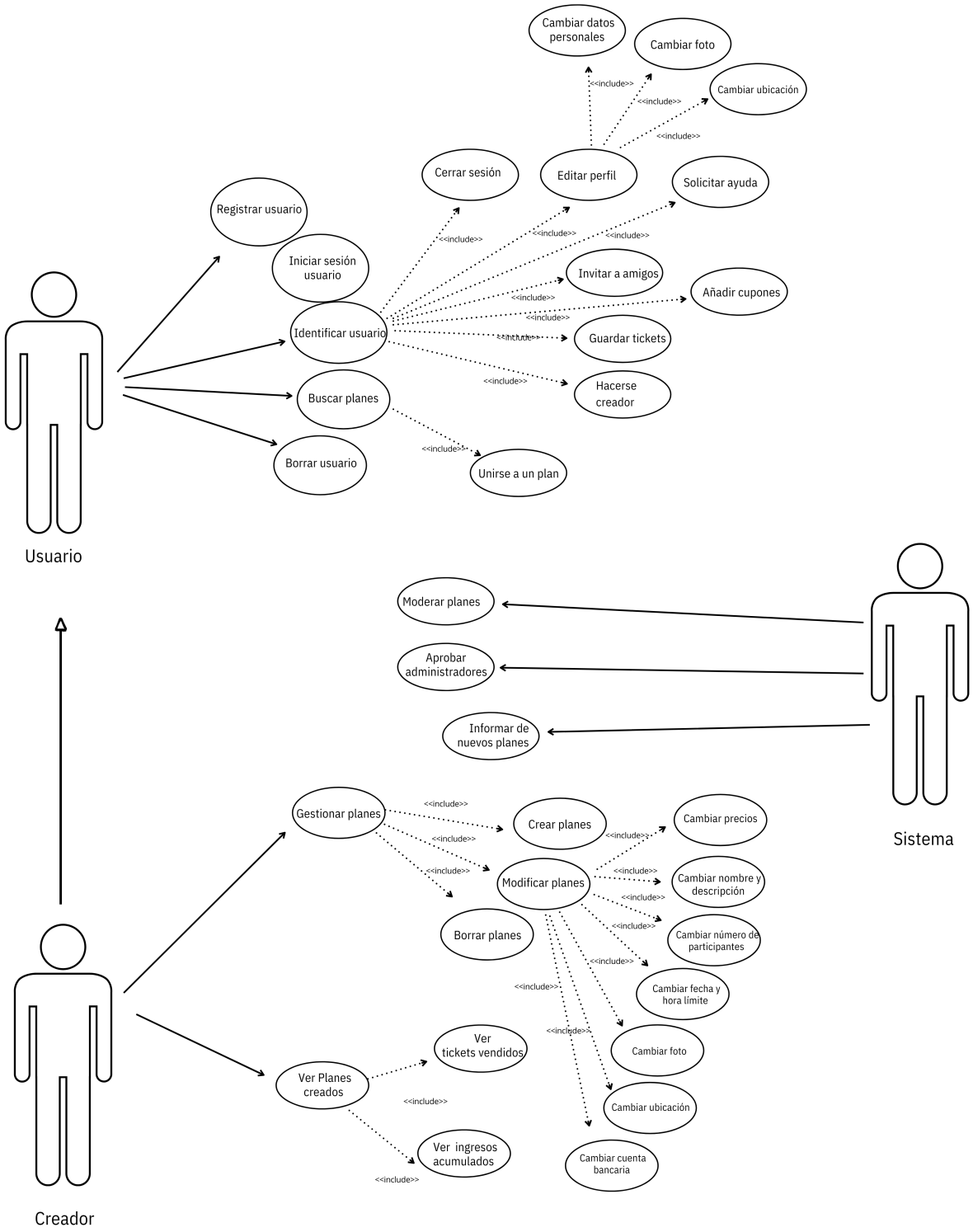


Figura 3.7: Casos de uso y actores.

3.1.6. Requisitos funcionales

En este apartado se pretenden especificar cada uno de lo requisitos funcionales con tal de obtener un mejor conocimiento acerca de la funcionalidad esperada por la aplicación.

Estas tablas incluyen los siguientes aspectos:

- Se encuentran divididos en secciones donde el actor se muestra en la parte inicial de la sucesión entre paréntesis.
- Caso de uso, se refiere al nombre de un proceso o tarea específica que un usuario puede realizar en una aplicación o sistema.
- Descripción, es un breve resumen que explica la acción que se realiza en ese proceso o tarea específica.
- Precondición, es una condición previa necesaria que debe cumplirse para que el usuario pueda llevar a cabo la acción del caso de uso.
- Secuencia, es una lista de pasos que el usuario debe seguir para completar la tarea o proceso específico del caso de uso.
- Postcondición, es una explicación de los efectos que la acción del caso de uso tendrá sobre el usuario o la aplicación.
- Excepciones, son situaciones en las que el caso de uso no puede ser llevado a cabo debido a algún error o problema, Estos problemas se describen para poder indentificar los motivos que los causaron.

Para facilitar la lectura y comprensión del trabajo, las imágenes de los casos de uso correspondientes a cada actor se encuentran en el Anexo 1 correspondiente al Capítulo 9.1.

Los casos de uso principales donde el actor es un **Usuario** son:

- Registrar usuario
- Iniciar sesión
- Buscar planes
- Cerrar sesión
- Editar perfil
- Solicitar ayuda
- Invitar amigos
- Añadir cupones
- Guardar tickets
- Convertirse en creador
- Cambiar datos de la cuenta
- Borrar cuenta

Los casos de uso principales donde el actor es el **Sistema** son:

- Moderar planes
- Aprobar creadores
- Informar de nuevos planes

Los casos de uso principales donde el actor es un **Creador** son:

- Gestionar planes
- Ver planes creados
- Crear planes
- Modificar planes
- Borrar planes
- Ver tickets vendidos
- Ver ingresos acumulados
- Ver tickets vendidos
- Cambiar cuenta bancaria

3.2 Identificación y análisis de posibles soluciones

Tras toda la información recogida durante el capítulo de *Estado del arte*, se concluye que existen diversas empresas en el mercado que realizan acciones relativas a la creación de eventos y el fomento de la socialización. No obstante, podemos observar como la gran mayoría de las propuestas nacen con un propósito similar al nuestro pero poco a poco se van transformando.

Así algunas se convierten en apps de citas que impulsan las relaciones más íntimas. Además, se puede apreciar como la principal fuente de ingresos de estas proviene de un modelo de negocio *Freemium*. Este consiste en la creación y prestación de un producto o servicio gratuito, pero que se convierte en pagado para tener algunas funcionalidades extra (artículo disponible en [20]) gracias a las compras dentro de la aplicación.

En el caso de las apps mencionadas en apartados anteriores, es necesario añadir que la gran mayoría utilizan anuncios dentro de la aplicación (*in-app*) con tal de monetizarla [19]. Por tanto, suelen depender del tipo de formula mencionada anteriormente.

Al realizar todo el análisis previo de una futurible 'competencia', se puede apreciar como la aplicación propuesta busca una corriente diferente a la seguida por sus rivales. Pretende crear una innovación que se enfoque expresamente en la creación de planes y eventos. Es más, para hacer una mayor distinción, se introducen los planes gratuitos. Estos surgen como una innovación ya que ningún competidor posee una opción tal en sus productos.

Con este añadido se pretende una captación mayor de usuarios en la aplicación, lo que hará que esta evolucione rápidamente mientras cumple el objetivo principal de ser útil para todo el mundo. Si valoramos las posibilidades de esta, se puede apreciar que,

si bien el público objetivo son los jóvenes y estudiantes, puede llegar mucho más lejos y escalarse eficientemente.

Dentro del modelo que se persigue, se busca una viabilidad alejada de los anuncios. Es obvio que estos no son cómodos para los usuarios de las aplicaciones. Se pretende, por tanto, tener una ganancia de las comisiones de la propia aplicación (en planes de pago). Pues cuando se establezca un plan, de cada entrada vendida se deducirá un dos por ciento de comisión que irá destinada al mantenimiento de la aplicación.

3.3 Solución propuesta

Durante el proceso de desarrollo de una aplicación, hay varias fases importantes a tener en cuenta.

Primero, se lleva a cabo una planificación previa que implica investigar y evaluar las tecnologías disponibles para determinar las mejores opciones para el proyecto. A continuación, se procede con el desarrollo de la aplicación, donde se detalla la creación de la misma. Una vez que se completa el desarrollo, se lleva a cabo la implementación en un entorno real, por ejemplo, un servidor. Finalmente, se perfeccionan diversos aspectos del proyecto y se prepara una presentación para el público y el tribunal.

CAPÍTULO 4
Diseño de la solución

4.1 Arquitectura del sistema

Para comenzar a entender el diseño de la solución propuesta, es muy importante definir una arquitectura clara de nuestro sistema. Así pues, se busca que esta encaje perfectamente dentro de la variedad de los requisitos que se desean cumplimentar en el proyecto.

Con tal de definir adecuadamente la arquitectura de nuestro sistema, es necesario tener en cuenta las características iniciales y las bases de las que parte este proyecto. Así pues, se sabe que el desarrollo y proceso industrial de la aplicación será realizado por una persona. Por ello, se reconoce que no se debe de tratar con una arquitectura compleja. Además, es necesario que la arquitectura esté bien establecida en el marco social actual con tal de mantener un sistema escalable, fácil de testear y mantener en el tiempo.

Por ello, tras un estudio exhaustivo de los tipos de arquitectura de software más famosos del mercado[6], se ha decidido que, para un desarrollo con *Flutter* como *framework* para desarrollo *front-end* y *Firebase* como infraestructura para gestión de servicios de *back-end*, se empleará la conocida como **Arquitectura hexagonal**. Esta arquitectura se reconoce también como *Clean Architecture* y es una de las más notorias del mercado y elegidas para el desarrollo de aplicaciones móviles multiplataforma [21].

4.1.1. Clean Architecture

Clean Architecture se basa principalmente en la separación de preocupaciones y la creación de capas de abstracción. Así pues, nuestros objetivos utilizándola son crear un software de alta calidad que, además sea fácil de entender, mantener y evolucionar aunque sea por una persona ajena al proyecto.

Para entender un poco cómo funciona se procede a explicar los siguientes conceptos:

- Separación de Preocupaciones: se centra en la separación de las preocupaciones relacionadas con el negocio y las preocupaciones técnicas. Esto implica que el código responsable de la lógica del negocio se mantiene aparte del código asociado con la infraestructura técnica, como la base de datos o la interfaz de usuario.
- Capas de Abstracción: busca establecer capas de abstracción en el software, donde cada capa tiene una responsabilidad clara y definida. Las capas internas se enfocan en la lógica del negocio, mientras que las capas externas se encargan de la infra-

estructura técnica. Esta estructura permite modificar o reemplazar cada capa de forma independiente, sin afectar las demás capas.

- Principios SOLID: se basa en los principios SOLID, un conjunto de principios de diseño de software que promueven una estructura clara y mantenible[23]. Estos principios incluyen: Responsabilidad Única, que establece que cada clase o módulo debe tener una única responsabilidad; el Principio Abierto/Cerrado, que fomenta la extensibilidad del software sin modificar su código fuente; la Sustitución de Liskov, que establece que los objetos deben ser reemplazables por instancias de sus subtipos sin alterar la corrección del programa; la Segregación de Interfaces, que propone interfaces cohesivas y específicas en lugar de interfaces generales; y la Inversión de Dependencias, que promueve la dependencia de abstracciones en lugar de implementaciones concretas.
- Patrones de Diseño: utiliza patrones de diseño para lograr la separación de preocupaciones y la creación de capas de abstracción. Algunos de estos patrones incluyen el Patrón de Capas, que organiza el software en capas con responsabilidades específicas, y el Patrón de Inversión de Dependencias, que establece que las dependencias deben ser invertidas para depender de abstracciones en lugar de implementaciones concretas. Además, se utiliza el Patrón de Inyección de Dependencias, que permite la inyección de dependencias en componentes, facilitando la prueba y el reemplazo de implementaciones.

Una vez comprendido el funcionamiento de esta, se espera que, utilizándola, se obtengan ciertos beneficios tanto a corto como medio plazo:

- Cada componente tiene asignada una única funcionalidad claramente definida.
- El sistema es escalable y favorece la inclusión de nuevos *widgets* y funcionalidades.
- Flexibilidad para que cada capa del software se pueda cambiar o reemplazar sin afectar las otras capas.
- Facilidad de búsqueda e identificación de posibles errores o *bugs* correspondientes a la ejecución de la aplicación.
- Inclusión sencilla de los llamados *Patrones de diseño*, como se ha mencionado unas líneas arriba.
- Código sencillo, simple y legible.

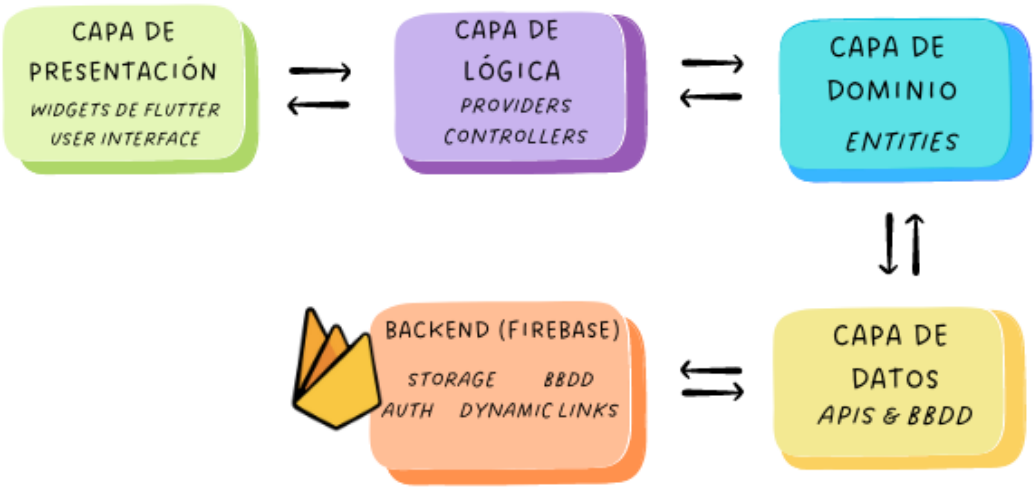


Figura 4.1: Diagrama de *Clean Architecture* en la aplicación propuesta.

A continuación, se comentará la funcionalidad de cada una de las capas mostradas en el diagrama anterior:

Capa	Funcionalidad
Capa de presentación	Usando los *widgets* de *Flutter* se diseñará la interfaz gráfica de usuario y los componentes comunes a diversas vistas de la aplicación.
Capa de lógica de negocio	Mediante el uso de controladores se manejarán todos los estados presentes que se puedan dar en la capa de presentación.
Capa de dominio	Nos encontramos las *entities*, que dan lugar a objetos de la aplicación y que servirán como esqueleto para la posterior obtención de esos datos.
Capa de datos	Gracias a la capa anterior podremos "rellenar" estos esqueletos con los datos que obtengamos de nuestro *backend* de *Firebase*.

Figura 4.2: Funciones de las capas de la arquitectura utilizada.

La capa de *backend* no ha sido incluida en esta tabla. Esto se debe a que es una capa externa a la que estamos accediendo en la nube. No obstante, esta capa es tan fundamental como el resto ya que, además de gestionar la base de datos y mandar estos datos a la capa de datos, realiza otro tipo de funciones. Se han implementado en ella las funcionalidades de *Storage*,*Authentication* y *Dynamic links* que serán explicadas posteriormente.

4.1.2. Patrones de diseño

Usar *Flutter* implica utilizar el manejador de estados de este. *Flutter* es capaz de manejar los diferentes *estados* que se van dando en la app en la que se está utilizando. Un ejemplo de ello es el famoso *Hot reload*. Esta función se basa en un botón que implica

poder realizar cambios en la app mientras esta se ejecuta. Así los cambios se actualizan automáticamente con un solo *click*.

Los *estados* de *Flutter* comprenden varios métodos implementados en *Dart* para su manejo y, son bien es muy útiles para proyectos más amplios, en la app propuesta resulta complejo entender todo su funcionamiento y aplicarlo de la forma correcta.

En cuanto a la comprensión del código, al principio resultó muy tediosa la adaptación a este lenguaje nunca antes estudiado. Así pues, se tomó la decisión de implementar varios patrones de diseño con los fines de comprender mejor el código, hacerlo más legible, controlar mejor los eventos y la modularidad propia de *Flutter* e incluso, mejorar la sincronización de los datos recibidos por la Capa de datos.

Dentro de la gran variedad de librerías e implementaciones que nos brinda *Flutter*, se ha optado por la combinación de los siguientes patrones:

- **Patrón Singleton**: Este patrón ha sido usado para asegurar que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia. Para ello se han usado clases para el *Tema* de la app, imágenes y textos. Así para aplicar un tema determinado, solicitar un texto o una imagen solo era necesario llamar a esa única instancia creada.

- **Patrón de inyección de dependencias**: Este patrón ha sido usado para insertar dependencias a otras clases en el constructor de una clase. En el código se ha implementado mediante el uso de *get.put* y *get.find*.

- **Patrón Observer**: Se usa cuando un cambio en un objeto requiere cambiar otros, pero no sabe cuántos objetos se deben cambiar y cómo. Por ello, se ha utilizado para crear código reactivo con uso de variables *.obx* . El patrón, además, permite suscribirse a tales eventos de objetos y cambiar el estado del objeto dependiente en consecuencia.

- **Patrón Provider**: Se ha usado para dotar y proveer al código de funcionalidades pertenecientes a la capa de lógica. En el caso del proyecto se han utilizado los *Providers* de *Firebase* para *Storage* y *FireStore*, el de Google para *Google Sign In*, e incluso se han creado algunos propios para un menú de la aplicación.

En el capítulo de *Desarrollo de la solución propuesta*, se podrán visualizar algunos ejemplos de dicha metodología en el sistema.

4.2 Tecnologías utilizadas

Como ya se ha mencionado con anterioridad, esta primera versión de la aplicación propuesta estará disponible tanto para *Android* como para *iOS*.

Es muy importante hacer una consideración, puesto que en primer lugar se planteó diseñar una app nativa para ambos sistemas (Kotlin para Android y Swift para iOS). Sin embargo, se acabó considerando que no era factible debido a que era necesario un esfuerzo inviable para una única persona en el tiempo correspondiente.

Así pues, se consideró el uso de un *framework* de desarrollo multiplataforma que diera un soporte a ambos sistemas operativos. Como ya se justificó en el segundo capítulo, el *framework* elegido fue *Flutter*, del que se habla a continuación.

4.2.1. Flutter y Dart

Si bien *Flutter* es el *framework* de desarrollo elegido, su lenguaje de programación es *Dart*. Este tiene el propósito de brindar al usuario una experiencia al más puro estilo *Google*. Para ello, se busca una apariencia similar al *Material Design* de *Google* y un rendimiento propio al de un desarrollo nativo. Así se pueden aprovechar todas las ventajas que nos brinda este *framework* multiplataforma[22].

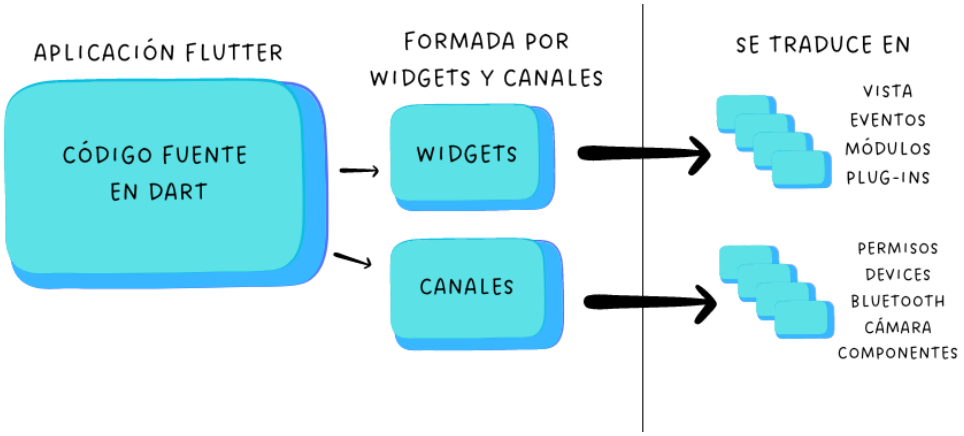


Figura 4.3: Diagrama de renderización de *Flutter* [24].

Es interesante explicar el diagrama superior puesto que los famosos *widgets* de *Flutter* toman un rol modular que permite que puedan ser utilizados de muchas formas posibles. Por ello, cuando se renderizan correctamente, pueden servir tanto como de estructura visuales para la capa de presentación como de métodos que permitan la gestión de eventos, como incluso de *plugins* creados por otros usuarios de internet que puedes importar a tu proyecto para *vitaminarlo*.

Los canales, por otro lado, mantienen una relación directa con lo anteriormente mencionado. Estos son aquellos *plugins* y librerías que ya vienen por defecto en todo proyecto de *Flutter* y que ya tienen unos servicios propios.

En el apartado de *Estado del arte* 2.2.1 ya se menciona por qué se ha elegido esta herramienta frente a sus rivales directos en el mercado. Añadir que , también ha sido necesario comparar Flutter con otras alternativas web diferentes. Se terminó valorando *Flutter* más positivamente debido a se considera prácticamente como la alternativa móvil a Angular, VueJS o JavaScript [13].

4.2.2. Firebase

Se ha utilizado *Firebase* como herramienta de gestión del *backend* del proyecto. La herramienta impulsada por *Google* ha facilitado sobremanera el uso de los servicios en la nube.

Esta herramienta presenta una base de datos *real - time*. Esto implica que los datos se sincronizan en tiempo real durante la ejecución de la aplicación. Este hecho permite que *Firebase* sea una *pareja de baile* más que adecuada para *Flutter*. Esto se debe a que, como ya se ha comentado, esta no sólo implementa el llamado *Hot reload* para realizar los cambios al instante mientras el proyecto se mantiene en ejecución, sino que, como *Flutter* es un *framework* declarativo no requiere de eventos para mantener los datos en sincronización.

Esto permite a los usuarios visualizar los datos en todo momento mientras los sincroniza a tiempo real.

La BBDD que implementa *Firebase* es de tipo *NoSQL* o *no relacional*. Esto no implica que no puedan usar el lenguaje SQL, pero no lo hacen como herramienta de consulta, sino como apoyo. Una característica fundamental de este sistema es que no utiliza estructuras rígidas. En otras palabras, los datos no se guardan en forma de tablas y la información no se organiza en registros o campos predefinidos.

Las funcionalidades o *add-ons* de *Firebase* que se han utilizado durante el desarrollo del proyecto son : Autenticación, Base de datos, Almacenamiento y Enlaces dinámicos. Procedemos a mostrarlos uno por uno.

4.2.3. Firebase Authentication

Firebase Authentication es el servicio de autenticación ofrecido por *Firebase*. Este permite al usuario el acceso a las funciones de la plataforma mediante inicio sesión con correo electrónico y contraseña o mediante el uso de otros proveedores como *Gmail*, *Cuenta de Apple* o *Facebook*.

Este además, te muestra un panel con todos lo usuarios que han iniciado sesión en alguna ocasión en la app:

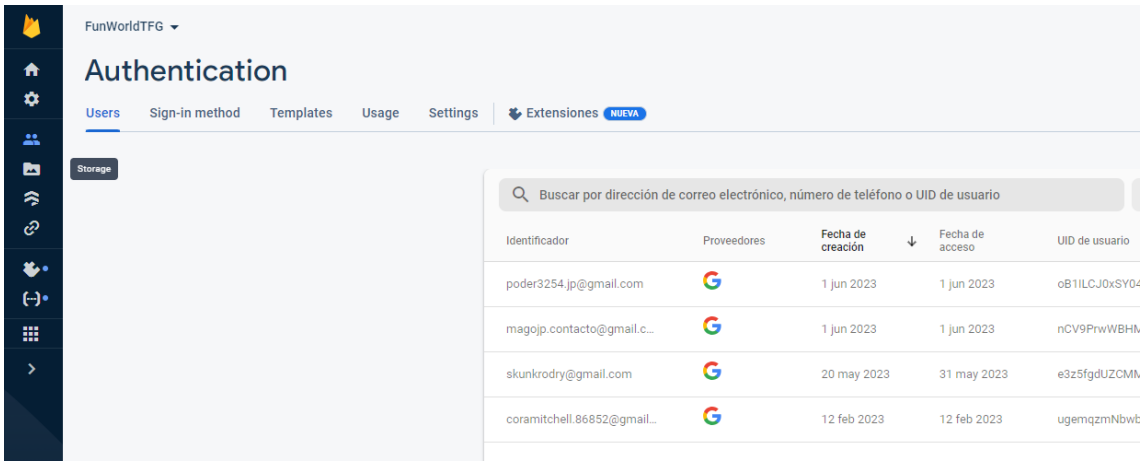


Figura 4.4: Panel de Autenticación de *Firebase*.

4.2.4. Firebase Firestore

Firebase Firestore es el servicio de gestión de Base de datos ofrecido por *Firebase*. Este permite al usuario interactuar mediante funciones CRUD (crear, leer, actualizar y borrar) de la base de datos del sistema.

Muestra un panel con todos los objetos y colecciones creados, así como sus atributos pertinentes:

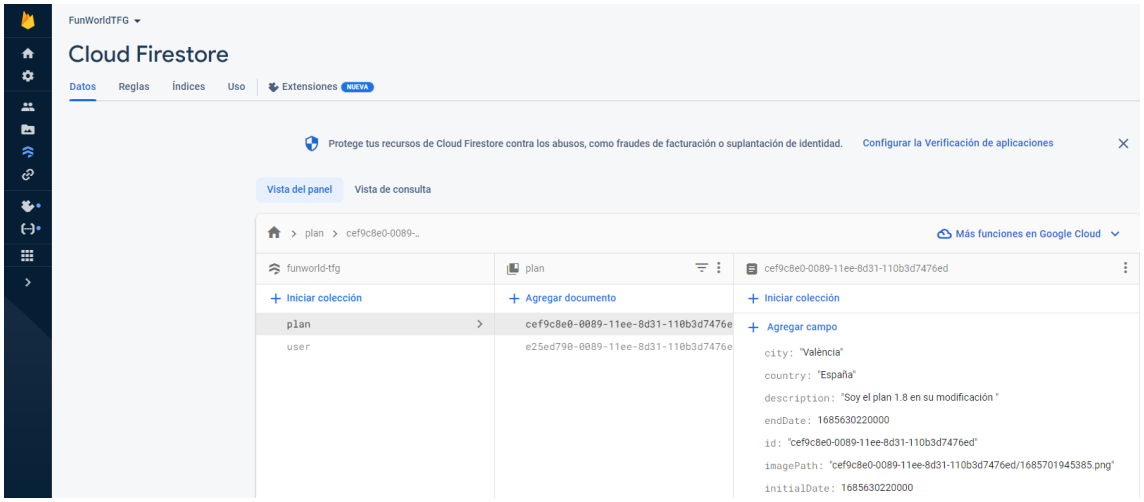


Figura 4.5: Panel de *Firestore*.

4.2.5. Reglas de Firestore

Las reglas de seguridad proporcionan control de acceso y validación de datos en un formato simple, pero claro y eficaz. Para crear sistemas de acceso según la función y el usuario que, además, mantengan protegidos los datos de los usuarios, es necesario utilizar *Firebase Authentication* con las reglas de seguridad de *Cloud Firestore.*

En este caso, es necesario mencionar que, al igual que se ha implementado *Git* en el proyecto para control de versiones (se muestra en páginas posteriores), *Firestore* tiene su propio control de versiones para las reglas de la misma base de datos.

A continuación se procede a mostrar una imagen de las reglas básicas que se han creado para el presente proyecto en *Firestore* y pequeña una explicación debajo de cada una de ellas.

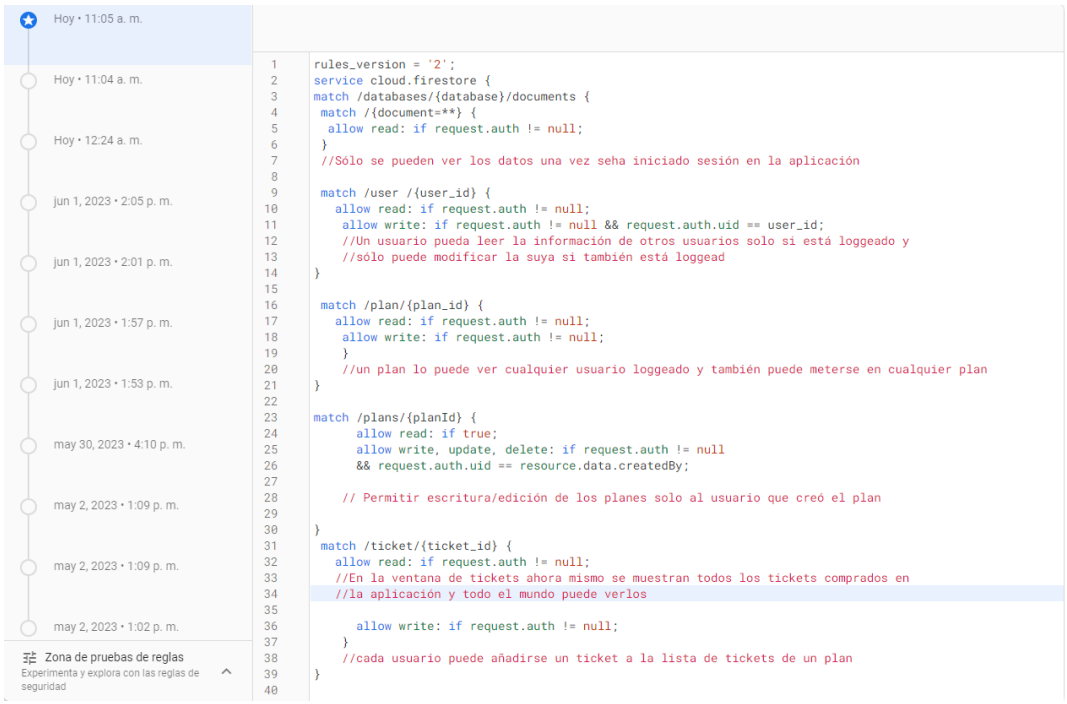


Figura 4.6: Panel de reglas de *Firestore*.

4.2.6. Firebase Storage

Otra plataforma que se encuentra disponible dentro de *Firebase* es *Storage*, un servicio de almacenamiento de elementos en la nube. Estos elementos son archivos inmutables y se almacenan en contenedores llamados *buckets*,los cuales están asociados con un proyecto específico.

Al utilizar *Storage*, tienes la posibilidad de acceder a tus archivos mediante referencias, cargar archivos de manera sencilla y supervisar el avance de las tareas.

Así pues, en la aplicación del proyecto se obtenía, a través de la cuenta de *Google*, el nombre y foto de perfil del usuario a registrar y se subía su enlace a *Storage* para, una vez cargado, crear una referencia local para no tener que cargar esa imagen cada vez que el usuario entrara a la aplicación. De esta forma se dinamizan también los tiempos de carga.

En la ventana de *Storage* se muestran todas estas imágenes y datos almacenados en carpetas con el *id* del usuario al que pertenecen:

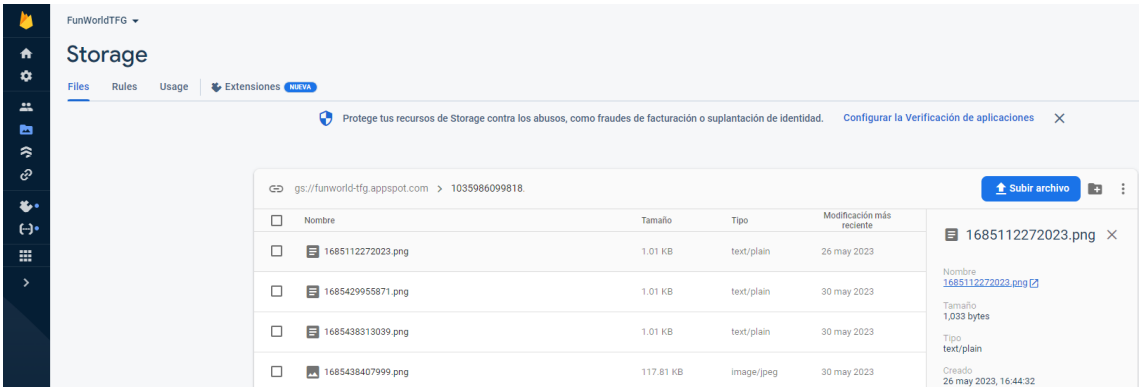


Figura 4.7: Panel de *Storage*.

4.2.7. Reglas de Storage

Para la funcionalidad mencionada en el anterior apartado también se han cambiado las reglas con total de mantener la protección sobre ciertos archivos de la aplicación. Las características de las reglas de *Storage* poseen una estructura exactamente igual a la vista en *Firestore*.

En este caso concreto solo ha sido necesaria una regla para garantizar la seguridad del *Storage* y se muestra en la siguiente imagen:

Figura 4.8: Panel de reglas de *Storage*.

4.2.8. Firebase Dynamic Links

Esta funcionalidad de *Firebase* permite crear enlaces dinámicos. Esto implica que se puede crear un enlace que vaya cambiando dependiendo del usuario que lo envíe.

El motivo principal de su implementación es el uso de enlaces de referidos para promocionar la aplicación. Este sistema de referidos consiste en que, cada usuario que se una a la plataforma tiene un enlace de referido que se puede compartir. Así, cada vez que tres usuarios descarguen la aplicación a través de ese enlace y realicen una compra, se le haga al administrador del enlace un descuento del diez por ciento en su próxima compra.

Se muestra, a continuación, una imagen del panel de *Firebase* correspondiente a esta funcionalidad:

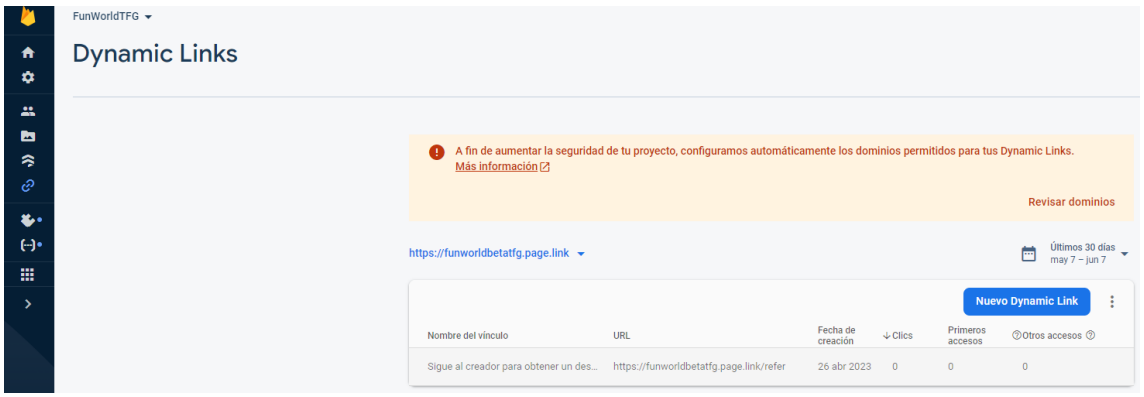


Figura 4.9: Panel de *Dynamic Links*.

A continuación, se muestra un diagrama de flujo con todos los serivicios de *Firebase* utilizados en la aplicación del proyecto:

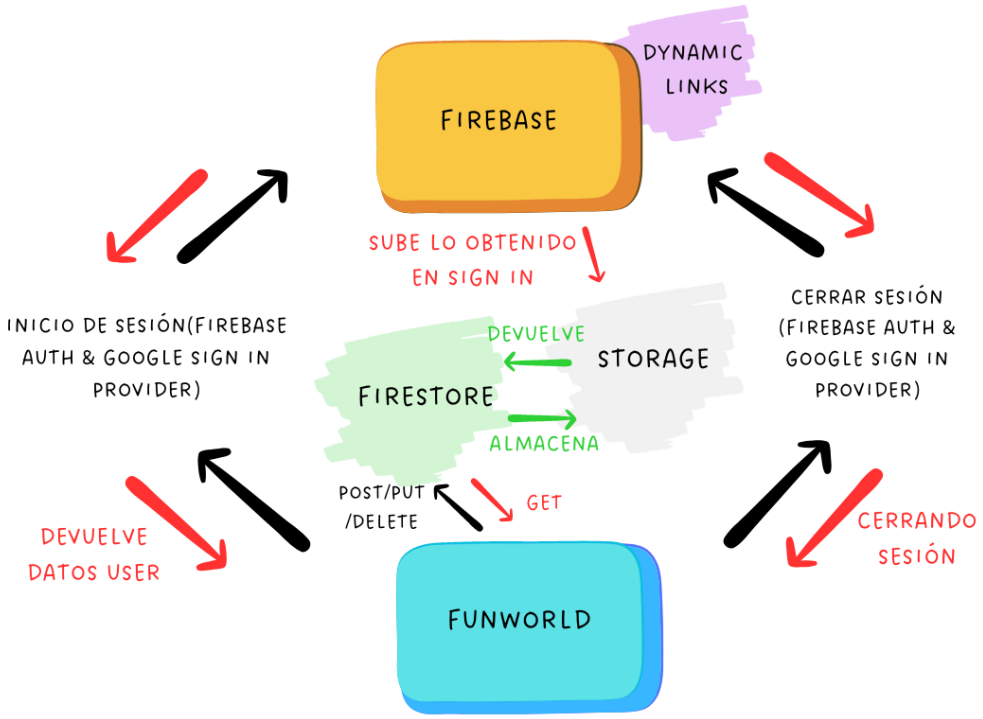


Figura 4.10: Diagrama de flujo de la app con *Firebase*.

En lo que respecta a la organización de los datos, considerando que se trata de una base de datos NoSQL, se empleará una estructura basada en documentos en lugar de tablas. Cada documento contará con una clave única de identificación y una serie de atributos. Estos documentos se almacenarán en colecciones específicas según su tipo, tales como usuarios, planes y tickets, en el caso de esta aplicación en particular.

A diferencia de los sistemas de bases de datos relacionales, este modelo de datos es altamente flexible y puede ser definido completamente por la aplicación en sí. No obstante, para garantizar una mayor consistencia, se ha establecido una estructura predefinida desde el portal web de *Firestore*.

4.2.9. Android Studio

Se ha decidido utilizar Android Studio como IDE de desarrollo. Esta herramienta creada e impulsada por el equipo de desarrollo de *Android* (que a su vez es el equipo de desarrollo de *Google*) está diseñada para trabajar al cien por cien con aplicaciones móviles Android. Es un IDE sencillo y aunque, si bien no posee tantos *plugins* como su competencia directa *Visual Studio Code*, es fácil de entender y el hecho de que sólo se use para aplicaciones móviles agradaba mucho.

Es interesante ver cómo ha mejorado su proceso de compilación y la interfaz gráfica de usuario, la cual es cada vez más intuitiva. Cabe destacar que la aplicación solo se puede ejecutar actualmente en *Android* debido a que el equipo utilizado para desarrollar el proyecto posee un sistema operativo de *Windows*. Sin embargo, si el proyecto se abriese en una computadora con *MacOS* y *Xcode* se podría ejecutar sin cambio alguno en un dispositivo *iOS*.

4.2.10. Git

Se ha decidido implementar *Git* para realizar el *control de versiones* alojando el proyecto en un repositorio privado de *GitHub*. Es necesario mencionar que se ha utilizado una única rama *main* para el desarrollo de este proyecto. Esto se debe a que, al tener un equipo de trabajo prácticamente unitario, y no tener que compartir el proyecto con otros compañeros, se valoró como la mejor opción.

A continuación se muestra una imagen de los últimos *commits* realizados en el proyecto:

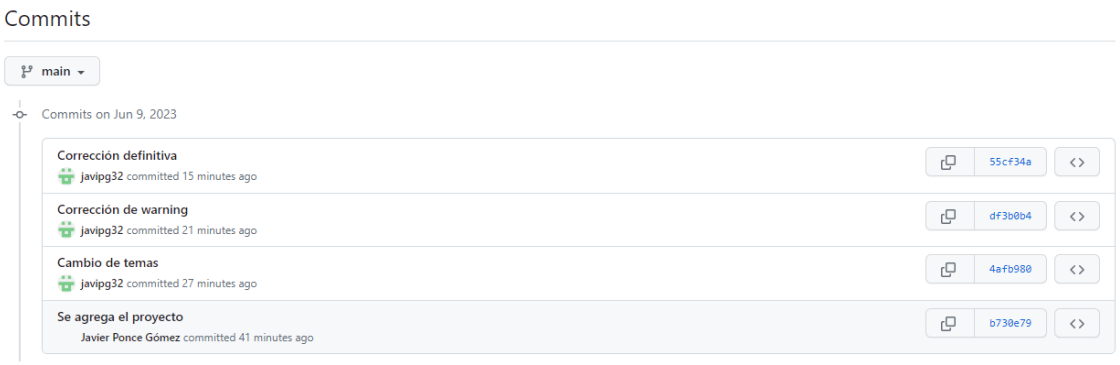


Figura 4.11: Últimos cambios realizados en el repositorio.

CAPÍTULO 5

Desarrollo de la solución propuesta

Tras el diseño de la solución realizado en el capítulo anterior. Se procede a realizar el desarrollo de la solución propuesta. Así pues, se mostrará código sencillo para comprender ciertos aspectos del proyecto y dotar de contexto, más aún a la situación.

También se comentará el proceso realizado y algunas de las decisiones tomadas para construir una propuesta sólida.

5.1 Modelo de datos

También llamado *Entities*, son los datos que hemos utilizado para este proyecto. Aquellas clases que, como se ha mencionado antes, sirven para construir una interfaz a partir de la cual, se irán creando los objetos protagonistas de esta aplicación.

Los tres tipos de colecciones que hemos creado para este proyecto son: User, Plan y Ticket. A continuación se muestra una imagen con los atributos de las mismas:

Figura 5.1: Interfaces de las clases de datos

Además procedo a destacar dos aspectos clave que se han dado durante el desarrollo del proyecto y que considero que han servido mucho para aprender a tener una visión más amplia y de optimización.

En primer lugar, se planteó crear una cuarta colección llamada Creador. Sin embargo, esto no se dio debido a las complicaciones técnicas que suponía tener un usuario normal que en algún momento sería necesario convertir en Creador. Así pues, la solución a la que se llegó fue añadir el atributo bankAccount a la clase User para facilitar la implementación. De esta manera, cuando un usuario normal se registrara, este campo se mantendría vacío hasta que se convirtiera en Creador, sólo ahí este atributo deja de ser nulo y se considera un Creador.

En segundo lugar, se percibió que, si bien *Dart* es un lenguaje más que correcto, definir modelos resultaba muy tedioso (constructor, propiedades, overrides, *copy-with*, serialización...) Entonces, tras mucho investigar descubrí un *plugin* llamado Freezed, el cual facilitaba increíblemente el código y permitía hacer todo el desarrollo con mucho menos código . Se añade un ejemplo de la página web muy representativo en el que se vislumbra la diferencia entre usarlo y no usarlo:

Before

```
@immutable
class Person {
  const Person({
    required this.firstName,
    required this.lastName,
    required this.age,
  });

  factory Person.fromJson(Map<String, Object?> json) {
    return Person(
      firstName: json['firstName'] as String,
      lastName: json['lastName'] as String,
      age: json['age'] as int,
    );
  }

  final String firstName;
  final String lastName;
  final int age;

  Person copyWith({
    String? firstName,
    String? lastName,
    int? age,
  }) {
    return Person(
      firstName: firstName,
      lastName: lastName,
      age: age,
    );
  }

  Map<String, Object?> toJson() {
    return {
      'firstName': firstName,
      'lastName': lastName,
      'age': age,
    };
  }

  @override
  String toString() {
    return 'Person('
      'firstName: $firstName, '
      'lastName: $lastName, '
      'age: $age'
    ')';
  }

  @override
  bool operator ==(Object other) {
    return other is Person &&
        person.runtimeType == runtimeType &&
        person.firstName == firstName &&
        person.lastName == lastName &&
        person.age == age;
  }

  @override
  int get hashCode {
    return Object.hash(
      runtimeType,
      firstName,
      lastName,
      age,
    );
  }
}
```

After

```
@freezed
class Person with _$Person {
  const factory Person({
    required String firstName,
    required String lastName,
    required int age,
  }) = _Person;

  factory Person.fromJson(Map<String, Object?> json)
      => _$PersonFromJson(json);
}
```

Figura 5.2: Código antes y después de *Freezed* [14]

1. **Secuencia de acciones**, que representan un posible comportamiento del sistema.

2. **Actores**, que son distintos roles que puede tomar el usuario a la hora de interactuar con el sistema. Un actor puede representar personas, dispositivos, otros sistemas, etc.

5.2 Estructura de ficheros

Como se ha mencionado anteriormente, la arquitectura de este proyecto es una arquitectura de tipo *Clean Architecture*. Así pues, se procede a mostrar una imagen de la disposición de los ficheros en el proyecto con la explicación pertinente de su utilidad:

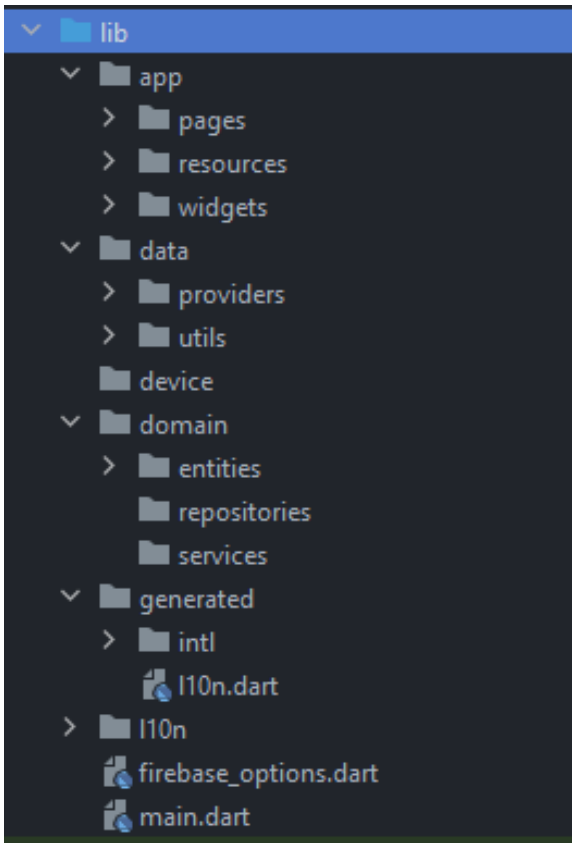


Figura 5.3: Estructura de los ficheros del proyecto

A continuación se procede a explicar la utilidad de todos los ficheros creados durante el proceso de la aplicación:

- ***App*** contiene todos los recursos referentes a la capa de presentación. Dentro de la misma podemos encontrar otros tres sub-ficheros:

  - ***Pages*** contiene todas las pantallas que se van a mostrar en la aplicación. Esta compuesta por cada una de las vistas de la interfaz de usuario y los *widgets* propios a cada vista.
  - ***Resources*** es la carpeta en la que se empieza a implementar el patrón *Singleton*, pues aquí se encuentran los recursos que se podrán instanciar en varios lugares de la aplicación. Se encuentran dentro los estilos de los *widgets*, el tema de la aplicación y las imágenes.
  - ***Widgets*** es la carpeta en la cual se encuentran todos los *widgets* comunes a varias pantallas de la aplicación, puesto que los *widgets* que solo se encuentran en una pantalla están dentro de la carpeta de esa pantalla.

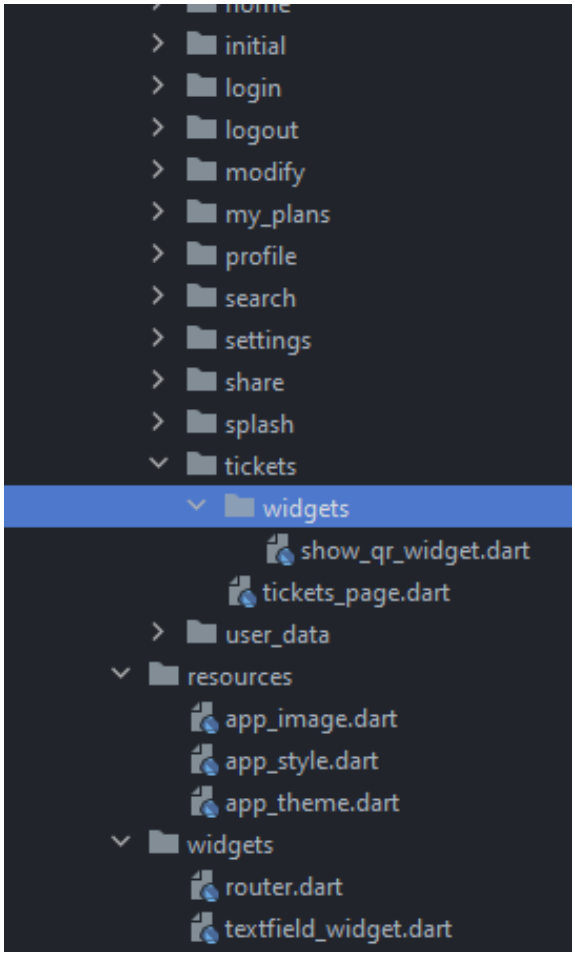


Figura 5.4: Estructura de widgets

- ***Data*** contiene todos los archivos referentes a la capa de lógica. Aquí se pueden visualizar dos carpetas diferentes.
  - ***Providers*** es la carpeta que, como su propio nombre indica, contiene todos los controladores y proveedores de servicios. Aquí destacan los servicios de *Firebase*, de *Google*, de localización, del mundo y las *Shared Preferences*.
  - ***Utils*** en esta carpeta se han añadido tipos de datos modificados para este programa. En este caso particular se han establecido logotipos para menús convertidos a *json*.
- ***Device*** es la carpeta donde se encuentran todos los métodos que tienen una relación directa con el dispositivo móvil. Ya sea para utilizar ciertas *APIS* de componentes *Bluetooth*, del giroscopio o así.
- ***Domain*** es el fichero representativo de la capa de dominio. En ella se encuentran, no solo los modelos de dominio presentados anteriormente, sino que también las carpetas de repositorios y servicios
  - ***Entities*** es la carpeta que contiene todo el modelo de datos. Así pues, dentro se encuentran las tres carpetas de las entidades correspondientes.
  - ***Repositories*** en esta carpeta se han añadido los repositorios utilizados durante el desarrollo del presente proyecto
  - ***Services*** en esta carpeta se encuentran los servicios de dominio que representan aquellas operaciones que pertenecen a nuestro dominio (y que por tanto,

sería recomendable que se realizasen dentro de nuestras entidades) pero que por razones técnicas o de otro calado han de extraerse fuera de ellas [8].

- ***Generated*** es una carpeta generada automáticamente tras la implementación del *plugin intl e intl-utils*.
- ***l10n*** es la carpeta en la que se encuentra un archivo llamado *intles.arb* en el cual se encuentran todos los textos en idioma español. De forma que cuando se quiera añadir otro idioma, se podrá añadir otro archivo para ese idioma y cambiado el valor de cada variable. De esta forma funcionarán de forma eficiente los servicios de traducción.

5.3 Funciones básicas

En este apartado se van a mostrar algunas de las pantallas principales de la aplicación para comprender mejor su funcionamiento. Además se va a explicar el propóstio de estas y cómo conseguir que funcionen correctamente.

Al iniciar la aplicación se puede apreciar un *Splash* (El logotipo de la aplicación en pantalla durante unos segundos de forma estática) que se mantiene en la pantalla durante tres segundos.

Posteriormente se abre la pantalla de Login. Actualmente esta sólo permite el método de autenticación mediante *Google*. Esto se debe a que originalmente se implementarían también el método de inicio de sesión mediante Facebook y Apple pero se desestimó esta propuesta por cuestiones de tiempo. Aun así es una de las funcionalidades que se pretende implementar para versiones futuras de la aplicación. Así pues, sólo tienes que elegir con qué cuenta de *Google* vas a entrar y tu usuario se registrará en la BBDD de *Firebase*. Además, tu nombre de usuario de *Google* y tu foto de perfil de la cuenta se usarán como foto de perfil y nombre de la cuenta de FunWorld.

Una vez la sesión se ha iniciado, el usuario se encuentra en la ventana principal llamada *Home*. En esta pantalla se mostrarán una serie de planes destacados y recomendaciones para los usuarios. Además se podrá ver el tipo de planes destacados dependiendo de una temática concreta.

La página siguiente que se muestra en las imágenes es la página de búsqueda. Es muy sencilla puesto que tan solo depende de un *widget* de barra buscadora que nos permite encontrar un plan por su nombre y de un *GridViewBuilder* que nos muestra todos los planes que se han creado en la app en esa ubicación. Así pues, es muy sencillo encontrar tu plan favorito simplemente por su nombre.

A continuación se adjuntan las imágenes de estas tres pantallas mencionadas:

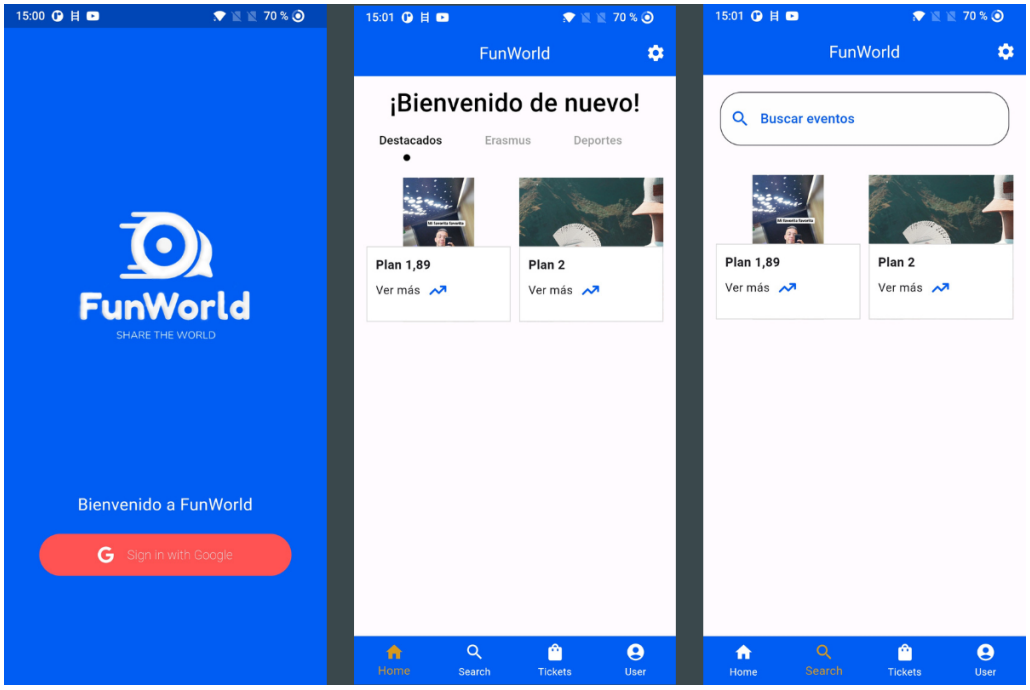


Figura 5.5: Visión de las primeras pantallas de la app

Respecto a las dos siguientes ventanas principales, cumplen dos funciones clave: el almacenamiento de los tickets y los ajustes de usuario.

En cuanto a la pestaña de almacenamiento de los tickets, tiene una función realmente sencilla. Cada vez que un usuario compre un ticket, este se almacena en la ventana de tickets correspondiente al usuario que lo ha adquirido. Esto justifica la creación de una variable userId que permite añadir al ticket el identificador del usuario que lo ha comprado. Si no se ha comprado ningún ticket todavía se puede ver esta ventana donde se muestra un botón que te redirige a la página concreta de búsqueda de planes.

Es necesario destacar la ventana de *User*. Esta contiene todos los ajustes del usuario y el resto de acciones que este puede ejecutar dentro de la aplicación. Se puede acceder a esta pestaña mediante el botón inferior derecho o el botón superior derecho (rueda dentada). Una vez dentro de la pantalla de *User*, se puede visualizar el nombre de usuario y la foto de perfil. Ambos se pueden editar, además de otros ajustes, *clickando* encima de la imagen de perfil. Además se pueden apreciar diversas acciones como: establecer una ubicación concreta, compartir la aplicación para conseguir descuentos, introducir cupones, añadir un código de administrador si se desea crear planes... y mucho más.

Así pues se muestran estas dos pantallas:

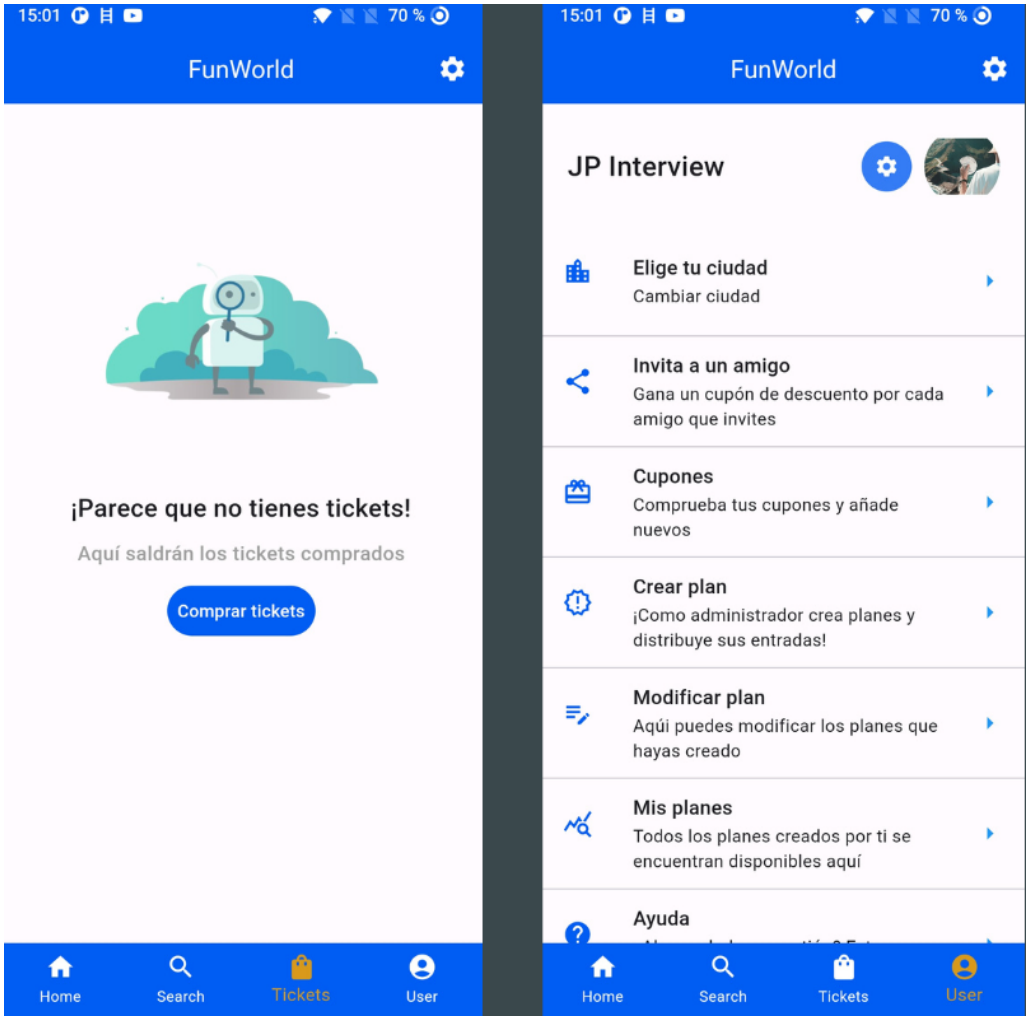


Figura 5.6: Visión de las primeras pantallas de la app (segunda parte)

5.4 Plugins destacados

En este apartado se procede a destacar los *plugins* más importantes utilizados en esta aplicación.

Si bien todos los *plugins* que se han añadido al proyecto son interesantes. Se considera que hay varios necesarios de mención, ya sea por su complejidad o por el contrario, por la simplicidad que ha aportado al código su implementación. Con esto es necesario argumentar que siempre se ha tratado de realizar buenas prácticas y optimizar el proyecto de la mejor forma posible.

Mencionar que no he añadido a esta lista los plugins necesarios para instalar *Firebase* ni ninguna de sus funcionalidades. Pues se entiende que el hecho de utilizar estas funciones en el proyecto obliga a instalar sus correspondientes añadidos.

Así pues, estos son algunos de los *plugins* más destacados del proyecto:

- ***Freezed y Freezed Anotation*** : estos ya han sido comentados por formar parte de uno de los patrones utilizados en el desempeño de este proyecto. Sin embargo , y por la gran labor de resolución que han dado era necesario mencionarlos en esta lista.

- ***Provider***: este también ha sido mencionado anteriormente pero es muy importante para el desarrollo del proyecto. Este *widget* permite manejar los estados de la aplicación de forma que podamos editar a la vez que la app se ejecuta y nos va a ayuda a compartir el estado de nuestra app a través de nuestras diferentes pantallas, de manera muy sencilla. Agregando un par de líneas de código se obtienen los resultados pertinentes.

- ***GetX*** : es una herramienta muy liviana y poderosa. Combina características de alto rendimiento de gestión de estado, inyección de dependencias inteligente y gestión de rutas ágil.

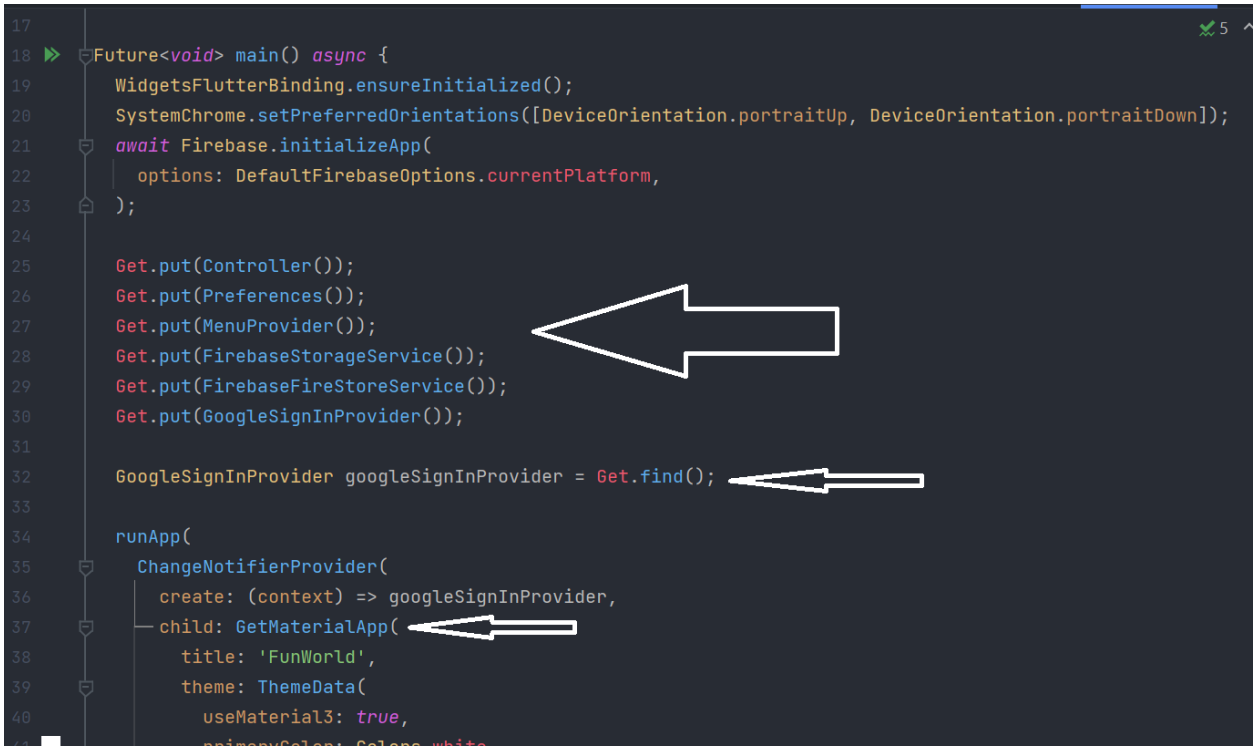
  Esta sigue tres principios fundamentales, que son prioridades para todos los aspectos de la biblioteca de un programador.

  - **1. Rendimiento**: *GetX* se centra en ofrecer el mejor rendimiento posible utilizando la menor cantidad de recursos posible
  - **2. Productividad**: *GetX* utiliza una sintaxis concisa y fácil de usar que acelera el proceso de desarrollo y aumenta la productividad del programador.
  - **3. Organización**: *GetX* permite el desacoplamiento completo entre las vistas y la lógica empresarial, lo que permite una mejor organización del código.

  Mediante el uso de GetX, se logra una mayor versatilidad en la implementación y gestión de componentes en Flutter. Además, facilita el manejo de la lógica de negocio de una manera más sintáctica y cómoda. También ofrece métodos para el manejo de objetos y propiedades observables que se actualizan en tiempo real, brindando una experiencia más dinámica y fluida.

  En el caso personal del proyecto, se decidió implementar como una forma de gestionar mejor los controladores, los provider e incluso algunos servicios usando los claúsulas *Get.put* y *Get.find*.

  A continuación se muestra un ejemplo de uso en el código. En la primera imagen del *main.dart* se puede ver como se inicia el *widget* principal de *GetX* y como se cargan las dependencias con *Get.put*. Estas son extraidas posteriormente por las clases que las necesiten mediante el uso de *Get.find*, como podemos ver en *splashpage.dart*.



```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp, DeviceOrientation.portraitDown]);
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

  Get.put(Controller());
  Get.put(Preferences());
  Get.put(MenuProvider());
  Get.put(FirebaseStorageService());
  Get.put(FirebaseFireStoreService());
  Get.put(GoogleSignInProvider());

  GoogleSignInProvider googleSignInProvider = Get.find();

  runApp(
    ChangeNotifierProvider(
      create: (context) => googleSignInProvider,
      child: GetMaterialApp(
        title: 'FunWorld',
        theme: ThemeData(
          useMaterial3: true,
```

Figura 5.7: Uso de *GetX* en Main.dart

```dart
class _SplashPageState extends State<SplashPage> {
  final Controller _controller = Get.find();
  final Preferences _preferences = Get.find();
  final FirebaseFireStoreService _firebaseService = Get.find();
  final FirebaseStorageService _storageService = Get.find();
  late NavigatorState _navigatorState;

  @override
  void initState() {
    super.initState();
    initializeApp();
  }
```

Figura 5.8: Uso de *GetX* en la pantalla de Splash.

- ***Share Plus*** : es el *widget* que permite compartir enlaces de todo tipo de varias formas diferentes desde cualquier tipo de dispositivo. Es muy sencillo de utilizar y ha sido añadido a esta lista por la gran cantidad de facilidades que ha supuesto para el desarrollo del código. Un ejemplo de su uso se puede ver en la siguiente imagen de la página que permite compartir tu enlace de referid@:

```
InkWell(
  onTap: () async {
    final dynamicLink = await FirebaseDynamicLinks.instance.buildLink(dynamicLinkParams);
    setState(() {
      dynamicLinkUri = dynamicLink;
    });
    await Share.share('¡Estoy utilizando FunWorld! Únete para acceder a los mejores planes y eventos \n\n$dy
  },
  borderRadius: BorderRadius.circular(30.0),
  child: Ink(
    height: 55.0,
```

Figura 5.9: Uso de Share en la pantalla de Invitación.

- ***Geolocator*** : este *widget* tiene el cometido principal de mostrar la ubicación exacta en la que se encuentra el usuario. Por tanto resulta fundamental para el desarrollo de este proyecto.

  Añadir que, en futuras versiones, se considera implementar el *widget de Google Maps* con tal de seleccionar una ubicación concreta. Sin embargo se ha tenido que dejar en el *Backlog* debido a la falta de tiempo para realizar una implementación así, pues era muy tediosa. Se puede ver el uso de este *widget* en las siguientes líneas de código para la ventana usada para obtener la ubicación actual del usuario:

```
Future<void> _determinePosition() async {
  bool serviceEnabled;
  LocationPermission permission;

  serviceEnabled = await Geolocator.isLocationServiceEnabled();
  if (!serviceEnabled) {
    Fluttertoast.showToast(msg: S.current.changeCityLocationOn);
  }
  permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
      Fluttertoast.showToast(msg: S.current.changeCityLocationOff);
    }
  }
  if (permission == LocationPermission.deniedForever) {
    Fluttertoast.showToast(msg: S.current.changeCityLocationOffForever);
  }
  Position position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
  try {
    List<Placemark> placemarks = await placemarkFromCoordinates(position.latitude, position.longitude);
    Placemark place = placemarks[0];
    setState(() {
      currentPosition = position;
      currentAddress = "${place.locality} ,${place.country}";
      pais = "${place.country}";
      ciudad = "${place.locality}";
      user = user.copyWith(city:ciudad);
      user = user.copyWith(country:pais);
    });
    user = await _firebaseService.putUser(user);
  } catch (e) {
    debugPrint(e.toString());
  }
}
```

Figura 5.10: Uso del método determine-position usando Geolocator en la pantalla de Cambio de ubicación.

- ***Intl*** : este es un *plugin* muy importante para cambiar el lenguaje de la aplicación. Como ya se ha explicado anteriormente, este nos permite crear variables con nuestros textos en diferentes idiomas para que ese adapten dependiendo de la región en la que se encuentra el usuario. Para invocar a estas variables es necesario usar la cláusula S.current.NombreDeTuVariable y estas se encuentran de la siguiente manera escritas:

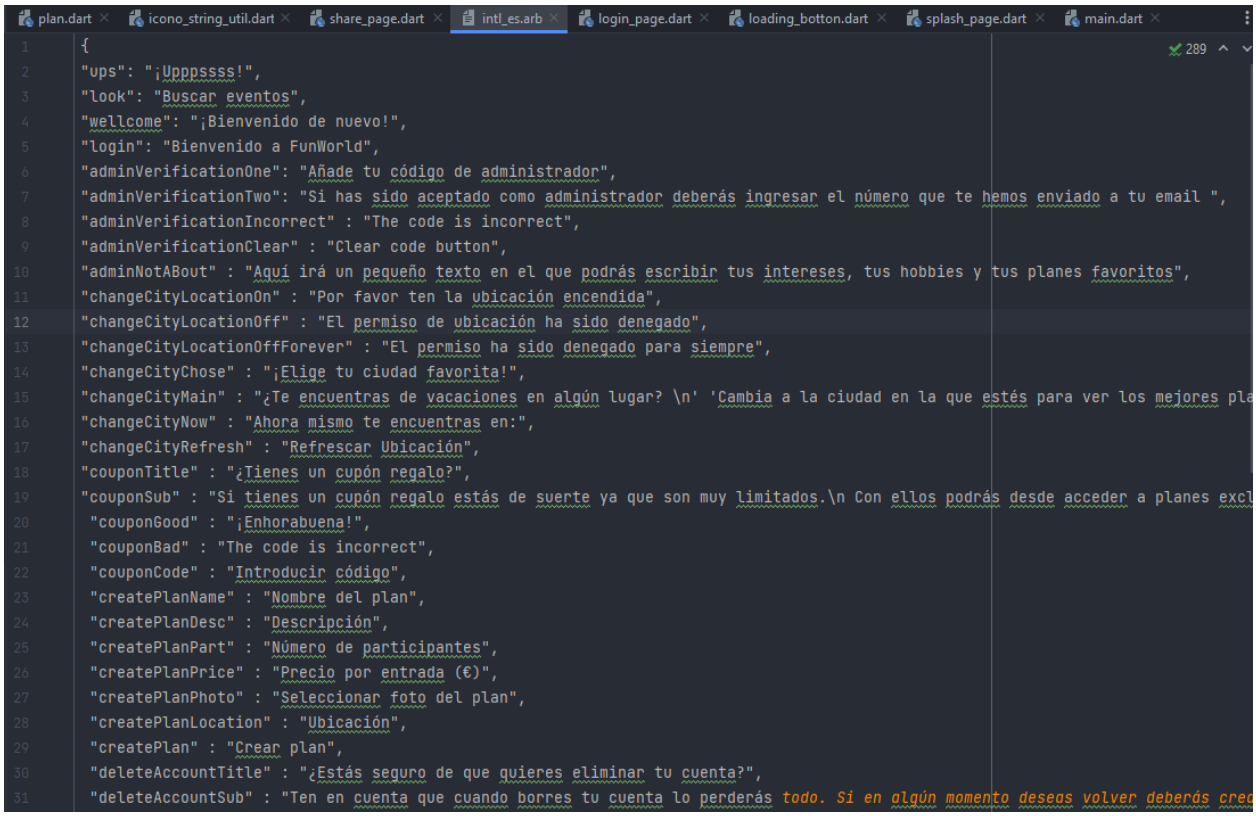


Figura 5.11: variables de texto en Intl.arb.

CAPÍTULO 6
Implantación

En este capítulo se procede a mostrar varias de las pantallas de la aplicación en un flujo orgánico. Con ello se pretende mostrar lo que podría ver cualquier usuario que accediera a la misma.

6.1 Primeros pasos en la aplicación

Para comenzar se procede a abrir la aplicación ya instalada en el dispositivo correspondiente. En este momento se muestra un *Splash* o animación estática con el logotipo de *FunWorld*. Después aparece un *Loading button* para iniciar sesión en la aplicación mediante cuenta de *Google*.

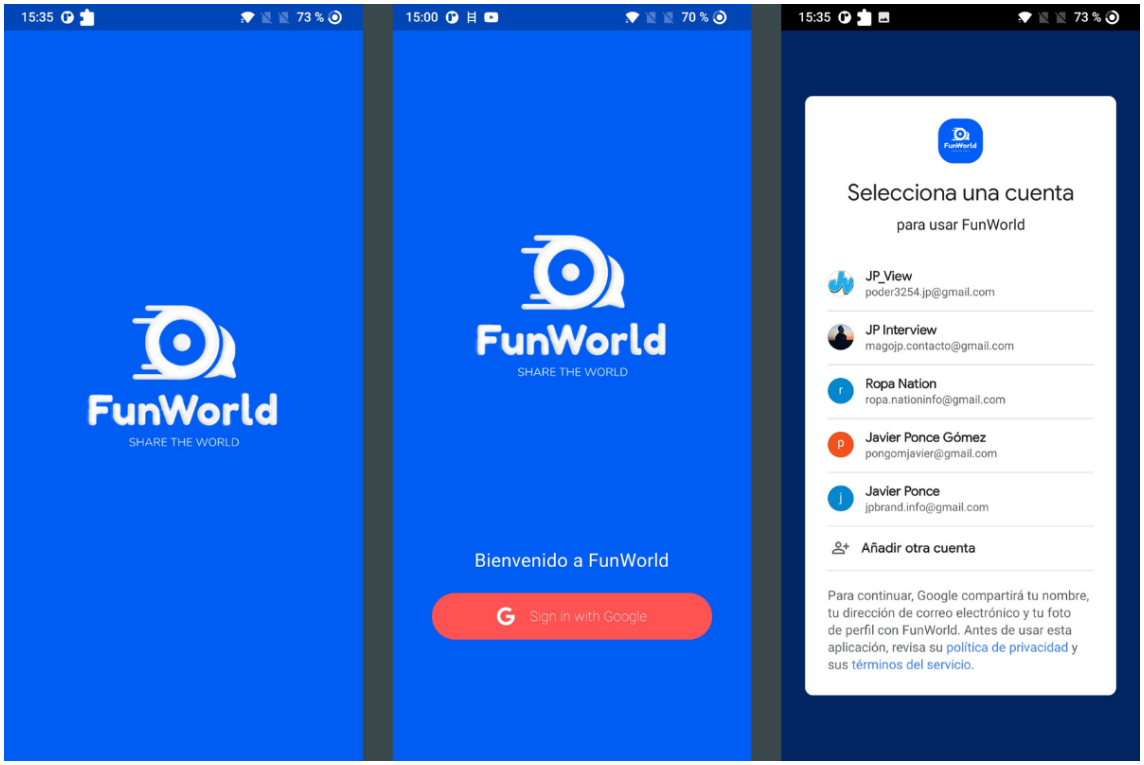


Figura 6.1: Tres primeras imágenes de la aplicación en su Inicio de sesión

Para desarrollar este código se ha implementado el *Provider* que contiene los métodos de *GoogleSignIn* como *googleLogin* y *googleLogout*. En este caso no se realizará una explicación exhaustiva del código. Pues se basa en una modificación de métodos creados por Firebase y que se han debido de adaptar al código a crear

También se ha creado la página del *Splash*. En esta página es se piden los permisos y se inicializa el *Controller* mediante el *plugin* de *Getx* (que ya se ha mencionado) y la página de *UserData* para gestionar el inicio de sesión. También se muestra la página de inicio de sesión que, a nivel de interfaz es muy sencilla mientras que la parte de lógica es más compleja y permite visualizar las diversas salidas del código así como su integración con el servicio de autenticación de *Firebase*.

6.2 Al iniciar sesión

Cuando el usuario ha iniciado sesión o se ha registrado con su cuenta de *Google* se puede ver una pantalla principal de *Home* con varias opciones en la parte inferior para cambiar de pantalla.

Si nos dirigimos a la siguiente pantalla, a la derecha, podemos visualizar una página de búsquedas donde se realizará un filtro de los planes según su nombre. Si no hay ningún plan para buscar, aparecerá una imagen estática indicando que no hay planes creados disponibles.

Posteriormente, en la siguiente página, se encontrarán todos los tickets adquiridos por los usuarios. En este caso particular, nadie ha comprado entradas. Por ello, se muestra una imagen estática y un botón para adquirir entradas.

A continuación se muestran las pantallas mencionadas.

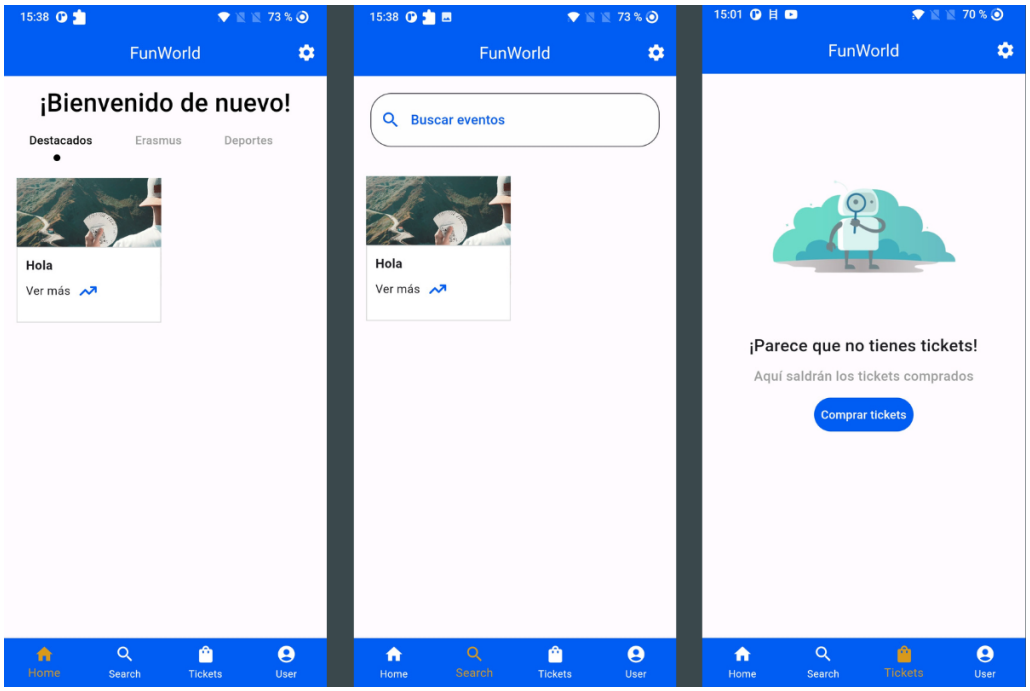


Figura 6.2: Capturas de pantalla del *Home*, *Search* y *Ticket Page*

6.3 Ajustes del usuario

Se muestra la última opción de la barra inferior. Esta opción está dedicada íntegramente a los Ajustes. En esta sección el foco se va a centrar en la visualización de la pestaña y las opciones del usuario.

En primer lugar se aprecia un nombre del usuario y una foto de perfil. Por lo general esta imagen y este nombre serán rescatados de la cuenta de *Google* del usuario con tal de facilitar más la creación del perfil y el entendimiento por parte de los propios usuarios. Además se encuentran varias opciones a realizar que se encuentran integradas mediante un *Provider* creado a propósito dependiendo de si el usuario es un creador o un usuario normal.

Si se quisiera editar el perfil, se puede ver un icono de rueda dentada al lado de la imagen de perfil del usuario. Al pulsar en este icono se muestra una pantalla con los datos pertenecientes al usuario junto con un botón de *Compartir perfil*.

Si se da un toque sobre la imagen del usuario, se podrán editar todos los datos del mismo. Hasta la propia foto de perfil se puede elegir desde la galería del usuario.

Se procede, a continuación, a mostrar las pantallas comentadas:

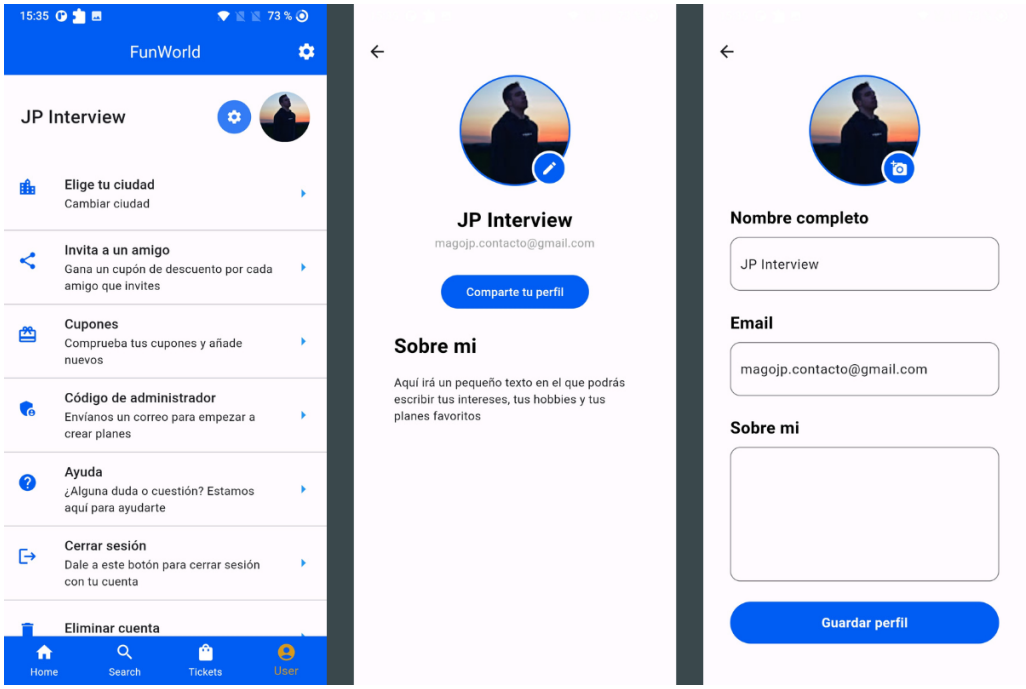


Figura 6.3: Capturas de pantalla del *Seetings*, *Profile* y *Edit Profile*

6.4 Varias opciones del usuario

En la pantalla de Ajustes se pueden vislumbrar varias opciones que un usuario normal pueda ejecutar. En este caso se mostrarán las primeras dos opciones que este puede ejecutar desde la ventana presente.

En primer lugar, el usuario añade la ubicación en la que se encuentra para crear los planes. Si bien se pretende que, en un futuro muy muy cercano, el usuario pueda elegir donde se encuentra... Actualmente lo interesante es que el usuario pueda crear solo planes para la ubicación exacta en la que se encuentra.

La siguiente pestaña mostrada consiste en un panel de invitación donde se genera un código Qr dinámico. Este permite al usuario invitar a personas que, al registrarse en la aplicación a través de ese enlace, puedan tener varios beneficios.

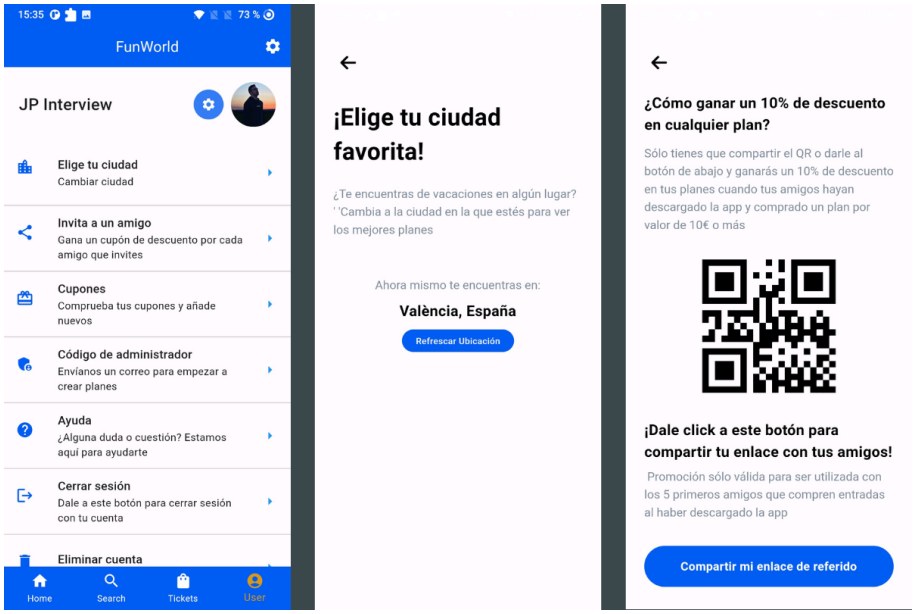


Figura 6.4: Capturas de pantalla del *Seetings*, *Change city* y *Share*

6.5 Ajustes de creador

En esta sección se pretende mostrar dos de las siguientes opciones que todo usuario posee dentro de la aplicación.

Como tercera opción dentro de esta pantalla de Ajustes, se puede visualizar una opción de Cupones. Esta opción sirve para introducir códigos exclusivos creados por la aplicación o a modo de colaboraciones con otras empresas. También sirve como un incentivo para el usuario para seguir a las redes sociales de ***FunWorld*** con el propósito de estar al tanto de posibles promociones y lanzamientos.

Posteriormente se encuentra la opción que permite a todo usuario autorizado por los desarrolladores a convertirse en un creador de planes. Para ello, un usuario cualquiera puede enviar un correo al equipo de desarrollo solicitando una clave de acceso que permitirá al usuario comenzar a crear planes. Decir que, el equipo de desarrollo le otorgará esta clave solo si lo considera necesario. Una vez se ha introducido este código se desbloquearán nuevas opciones dentro de la ventana de Ajutes. Así el creador podrá ver las opciones de *Crear plan* , *Modificar plan* y *Mis planes*.

Así pues, se muestra a continuación una imagen de las pantallas mencionadas.

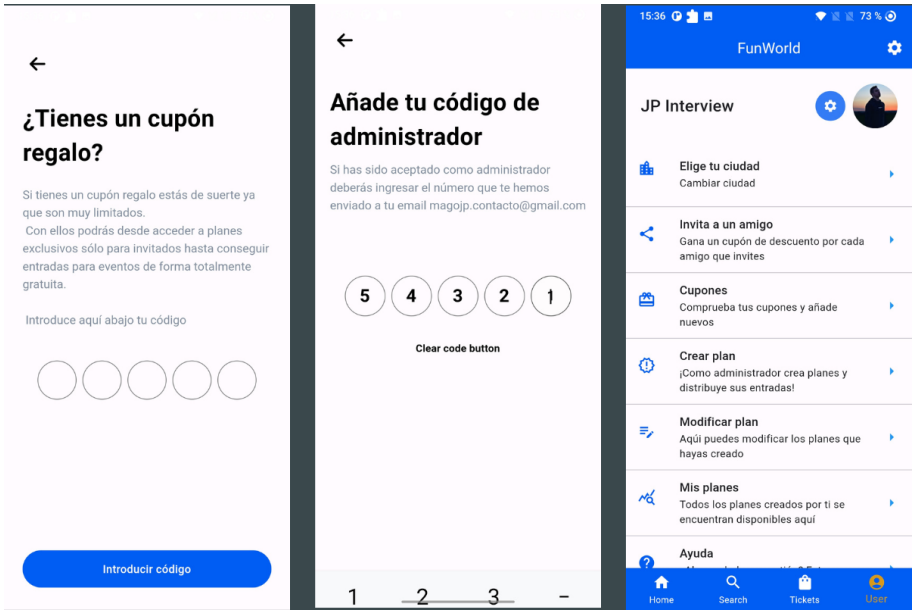


Figura 6.5: Capturas de pantalla del *Seetings*, *Change city* y *Share*

6.6 Modificar plan y Ayuda

Se procede a mostar varias pestañas antes de crear un plan. En este momento, si el usuario tratara de modificar algun plan se encontraría con que no es posible. Esto se debe a que no se puede modificar ningún plan si no se ha creado ninguno todavía. Por ello, se puede ver una imagen que muestra cómo no se ha creado ningún evento aún y permite darle a un botón para comenzar con la creación de un plan.

También se muestra la ventana de ayuda. En ella se puede enviar un correo electrónico al equipo de desarrollo para aclarar las dudas pertinentes. Simplemente con pulsar el botón que se encuentra en la parte inferior de la pantalla se puede hacer.

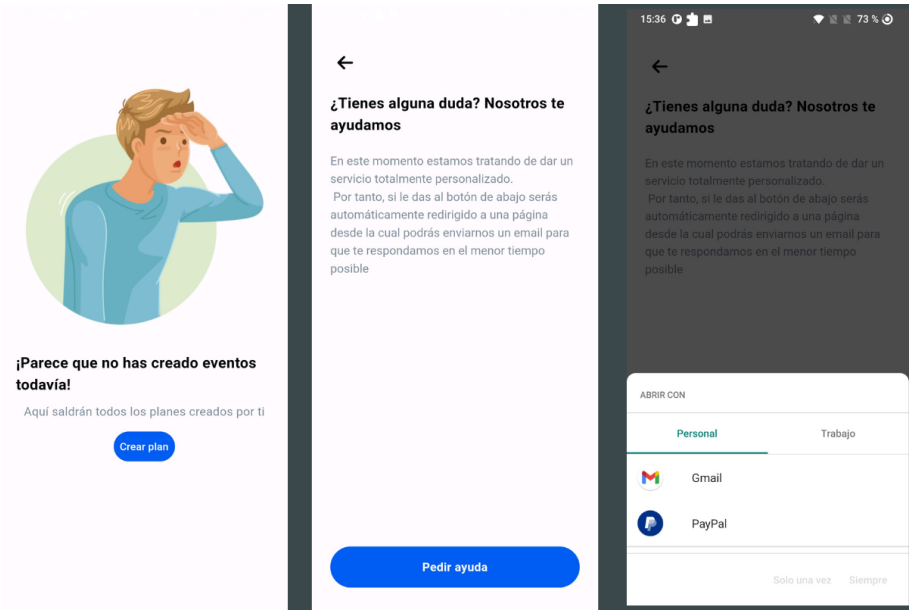


Figura 6.6: Capturas de pantalla del *Modify plan* y *Help*

6.7 Cerrar sesión, eliminar cuenta y crear plan

Para terminar con algunas de las opciones que todo usuario puede realizar, se muestra cómo eliminar una cuenta de forma sencilla mediante un botón. Una vez este es pulsado, saltará una ventana de diálogo que preguntará si esa persona está segura de su deseo de borrar la cuenta antes de realizar el procedimiento.

Finalmente se muestra la opcion de cerrar sesión. Esta tiene una interfaz muy similar a la anteriormente mostrada y permite, como bien indica el nombre, cerrar la sesión actual y redirigir al usuario de nuevo a la ventana de *Login* donde el usuario debería introducir de nuevo sus credenciales de *Google* en caso de querer entrar con su anterior cuenta, o cambiar de cuenta si desea entrar con otro usario.

Una vez mostradas todas las ventanas a las que cualquier usuario tiene acceso, se procede a crear un plan mediante la opción de *Crear plan* que ha sido desbloqueada tras convertirse el usuario en creador. En esta ventana, el usuario puede establecer el nombre del plan, su descripción, el número de asistentes, el precio de las entradas, la imagen del mismo, la hora de inicio y de fin, y la localización del plan.

A continuación se muestran las pantallas.

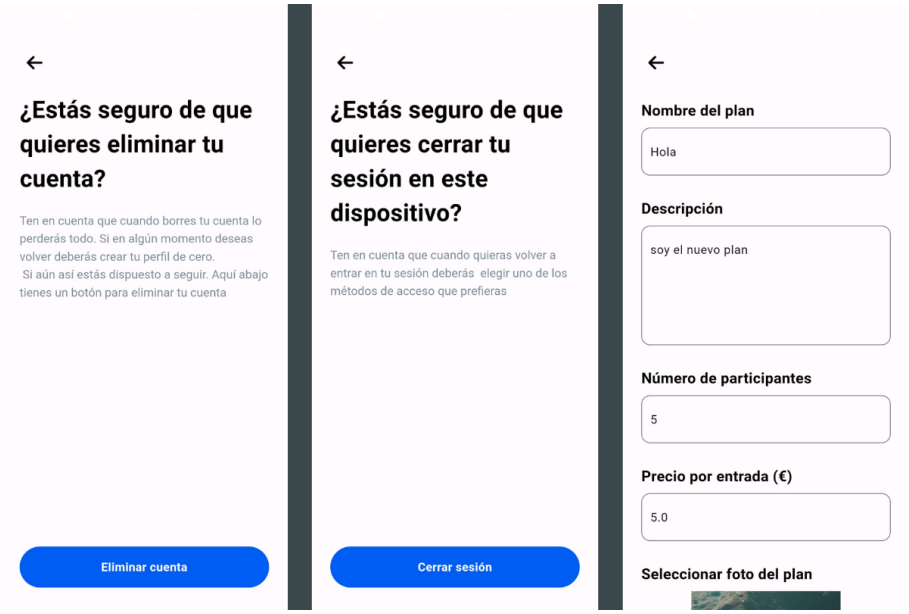


Figura 6.7: Capturas de pantalla del *Seetings*, *Change city* y *Share*

6.8 Visualización del plan creado

Una vez se han terminado de crear el plan con una determinada configuración, se procede a visualizarlo en las pantallas de la propia aplicación. Si el usuario se desplaza a la pantalla principal podrá ver un marco donde se encuentra la imagen del plan recién creado junto con su nombre.

Así mismo, si se pulsa en el marco, se podrá visualizar una ventana que tiene una vista en detalle del plan creado donde se puede encontrar toda la información del plan junto con las opciones de retroceso, de compartir el enlace del plan o incluso, si el usuario lo desea, de conseguir una entrada para el plan en cuestión.

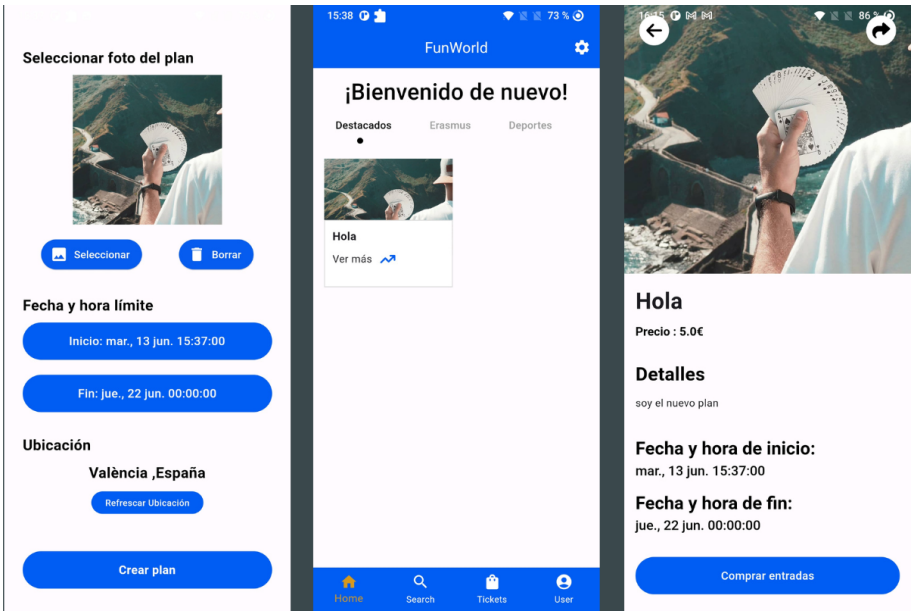


Figura 6.8: Capturas de pantalla del *Seetings*, *Change city* y *Share*

6.9 Compartir plan y comprar tickets

En las siguientes imágenes se muestra como el usuario al darle *click* al icono superior derecho puede elegir mediante qué medios desea compartir el plan.

Finalmente, se muestra cómo el usuario decide comprar un ticket para el plan. Luego se dirige a la pestaña inferior de Tickets y es ahí donde va a poder encontrar el ticket que acaba de adquirir.

Para poder enseñar este ticket, simplemente es necesario pulsar en el ticket que se quiera mostrar. Así aparecerá en pantalla un código Qr que se genera automáticamente. Este se prevee que tenga una id específica del plan en cuestión que se pueda detectar.

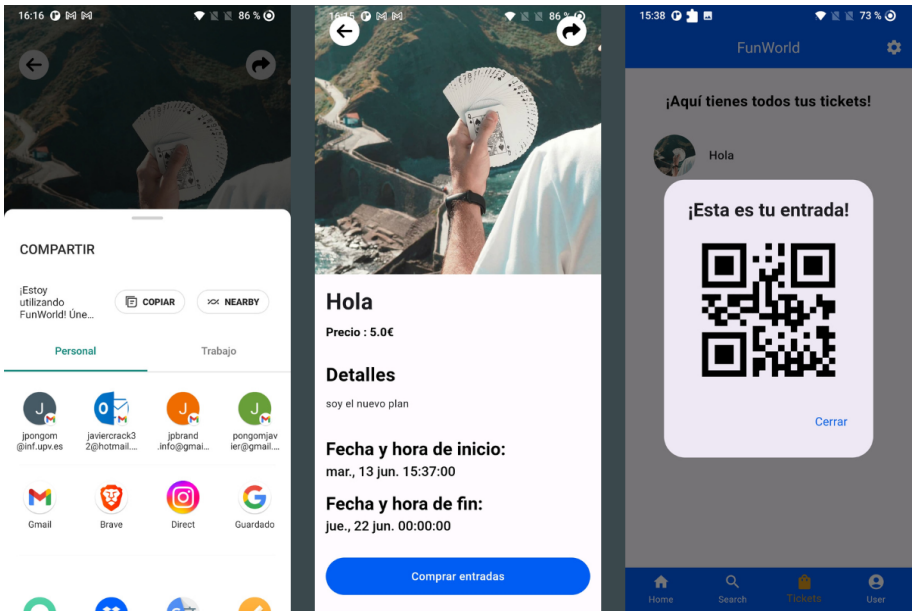


Figura 6.9: Capturas de pantalla del *Seetings*, *Change city* y *Share*

CAPÍTULO 7
Pruebas

En este capítulo se establece un plan de pruebas para nuestra aplicación. Posteriormente se analizarán sus resultados, y se llevarán a cabo las modificaciones que se consideren necesarias con tal de solucionar los posibles problemas que puedan surgir en dichas pruebas.

7.1 Desarrollo guiado por pruebas (TDD)

Para la realización de las pruebas de este proyecto se ha seguido la metodología TDD, también conocida como *Test Driven Development*. Esta es una metodología de programación que sigue un enfoque distinto. En lugar de comenzar directamente a escribir el código, se empieza por crear las pruebas correspondientes. Luego se desarrolla el código para que las pruebas pasen de manera exitosa y finalmente se lleva a cabo la refactorización del código corregido.

El propósito principal de esta técnica es obtener un código más sólido y un proceso de desarrollo más ágil. Son tres los aspectos fundamentales que garantizan que las pruebas se mantengan fáciles de entender y realizar:

- **1. Se genera documentación consistente**: al escribir pruebas antes de escribir código, se crea una forma de documentación actualizada continuamente para guiar el proceso de desarrollo.

- **2. Se realiza un rediseño regular del código**: el código se modifica y mejora periódicamente para mantenerlo en un estado que sea fácilmente verificable y comprensible.

- **3. Se establece una red de seguridad para los cambios**: así se establece un entorno confiable que permite realizar cambios de manera segura sin temor a introducir errores. Esto se debe a que las pruebas actúan como una red de seguridad.

En resumen, el TDD es una metodología que fomenta la escritura de pruebas antes del código, promoviendo la creación de una documentación continua, la realización de rediseños periódicos y la confianza en la implementación de cambios. Esto se traduce en un código más robusto y un proceso de desarrollo más eficiente.

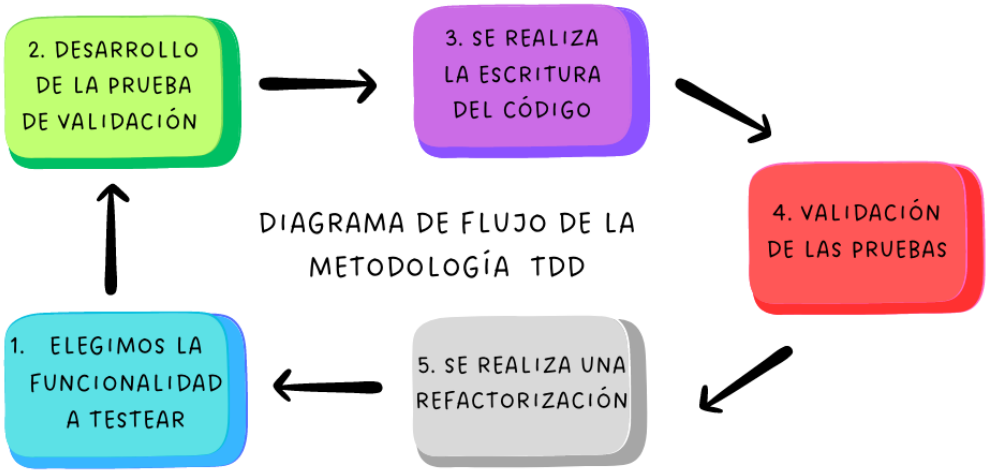


Figura 7.1: Diagrama de flujo TDD.

A continuación, se procede a explicar cada uno de los procesos a desarrollar en el uso de esta metodología:

- 1. Se comienza con plantear una funcionalidad de la cual se requiera hacer un *testing*.
- 2. Con tal de integrar esta funcionalidad en el proyecto, se desarrolla lo que se conoce como una prueba de validación. En esta parte del proceso, el desarrollador tiene que conocer, de forma clara, los requerimientos, condicionales y casos bordes.
- 3. El siguiente paso a realizar es llevar a cabo la escritura del código. Gracias a este se irán validando las diferentes pruebas.
- 4. Se procede a realizar una validación de las pruebas. Si en la fase anterior hemos conseguido pasar todas las pruebas establecidas, en este paso ya podemos decir que el software es válido y pasa los requerimientos establecidos. Si, por el contrario, se obtiene una serie de fallos, significa que la funcionalidad no se ha desarrollado correctamente y que, por tanto, se debe repetir el proceso. Una vez realizados los cambios pertinenetes, la ejecución de los test va otorgando resultados positivos.
- 5. Una vez se ha verificado la funcionalidad sin errores, se inicia el proceso de reestructuración o refactorización del código (que implica modificar desde nombres de las variables hasta elementos innecesarios). Estas sucesivas reestructuraciones contribuyen a mantener el producto de manera continua en el tiempo.

Así pues, existen varios tipos de pruebas que se pueden utilizar para evaluar soluciones de software. Al igual que otros marcos de desarrollo, *Flutter* proporciona una serie de herramientas para que esta tarea sea rápida y sencilla. Los tipos de pruebas más comunes en este entorno son: *widget*, *golden*, pruebas unitarias y de integración .

Para lograr la mayor cobertura de código posible, se decidió el uso los dos últimos tipos de pruebas porque, además, son totalmente complementarios. En las siguientes secciones se presentan y se explica su uso en el sistema.

En términos de entorno y gestión de herramientas de prueba, se ha usado, de nuevo, el entorno de desarrollo de *Android Studio* junto con todas sus oportunidades para la creación de pruebas (P.e: Ejecutor de pruebas de Gradle).

En este entorno, todos los conjuntos de pruebas se han ejecutado de forma individual o conjunta.

7.1.1. Pruebas unitarias

Las pruebas unitarias son aquellas que consisten en aislar una parte del código y comprobar que esta funciona a la perfección. Son pequeños tests que validan el comportamiento de un objeto y la lógica del mismo. Se consideran, además, una parte imprescindible en el desarrollo de cualquier tipo de *software*.

Para este proyecto se han llevado a cabo diversos tests. En su mayoría se han realizado para la carpeta *entities*, la cual, como ya se ha mencionado, contiene los modelos de datos de la app. Las tres clases comprendidas en su interior son importantes. Sin embargo, se han priorizado los tests en el modelo de **planes** y de **user** pues son los que se encuentran presentes en la mayoría de casos de uso.

Tras realizar el paso a paso de la creación de pruebas según los desarrolladores de *Android Studio*[9], se ha podido comprobar el desarrollo correcto de varias funcionalidades.

Algunas de las pruebas que se han implementado son:

- Los datos del plan estén completos (Se comprueba que no estén vacios para poder crear un plan)
- cuando un plan se cree, la hora de inicio no pueda ser posterior a la de fin.
- El usuario debe tener todos los datos (Se comprueba que se rellenen todos los campos)
- Los identificadores para cada usuario y plan son únicos con respecto al resto.
- Cada correo electrónico pertenece a un único usuario (Para la clase User)

7.1.2. Pruebas de integración

Las pruebas de integración, también conocidas como pruebas integradas, se definen como un método de prueba de software que involucra el análisis de procesos relacionados con la combinación o ensamblaje de componentes y su interacción con varias partes del sistema. Esto se puede dar a nivel operativo, de archivos, de hardware, etc.

Esencialmente, las pruebas de integración se encargan de comprobar las interfaces entre subsistemas o grupos de componentes del programa o aplicación analizados. Asi se puede asegurar su correcto funcionamiento. Estos tipos de test cubren las porciones de código relacionadas con los controladores y lógicas de negocio.

Si se pone el foco en la aplicación desarrollada, las pruebas de integración se encargarán de abarcar y testear todo el abanico de posibles comportamientos de los manejadores de estados. Sucede pues, que estos interactúan con gran cantidad de servicios externos. Al principio, es necesario que interactúen también con los *mocks*.

Estos *mocks* o maquetas son objetos de prueba que sirven para emular o imitar el comportamiento de objetos reales sin su implementación todavía realizada. Un ejemplo de ello fue crear *mocks* de objetos de las entidades utilizadas cuando la base de datos no se había implementado todavía.

Un beneficio de estos tipos de objetos es que se pueden simular todos las casuísticas de un determinado caso de uso, desde excepciones y fallos hasta situaciones ideales.

Por tanto, algunas de las pruebas de integración que se han realizado en este proyecto son:

- Recibir un plan sin ningún tipo de error
- Comprobar que cuando un plan se cree contenga todos los datos necesarios.
- Al seleccionar la ubicación que se muestre en tiempo real.

CAPÍTULO 8
Conclusiones

Tras la realización del presente proyecto y su respectiva memoria, no se termina la labor. Se plantea en este punto, tanto la relación del proyecto con los estudios cursados como las conclusiones obtenidas una vez terminada esta primera versión de la aplicación. Además, se propone en el último apartado una serie de hechos a considerar en el futuro y en los que se debe trabajar para seguir mejorando la aplicación.

8.1 Relación del trabajo desarrollado con los estudios cursados

Es necesario comentar que, sin los conocimientos del Grado en Ingeniería Informática y en especial, de la rama de *Ingeniería de Software*, no habría sido posible el desarrollo de este proyecto. A continuación, se procede a realizar un repaso de la influencia que las diversas asignaturas han tenido en el desarrollo del presente trabajo.

Cabe decir que, realmente no se notó un progreso en el aprendizaje hasta el tercer curso del grado, cuando fue necesario elegir la rama a cursar. En ese momento, empezó a crecer el conocimiento inconsciente de todo lo visto anteriormente. Con ello, la curva de aprendizaje se tornó mucho más sencilla.

Durante el primer periodo de tercero se cursó la asignatura de *Ingeniería del Software*, la cual ha permitido la comprensión y desarrollo de los modelos de dominio, diagramas uml y de casos de uso creados para el presente proyecto.

Así pues, ya en el segundo periodo, asignaturas complementarias tales como *Proceso de software* y *Diseño de software* han resultado también esenciales para este desarrollo. Por un lado, *Proceso de software* prepraraba al alumno para el 'mundo real' y le permitía construir un sistema de trabajo en base a unas metodologías ágiles que permitían, no sólo gestionar su tiempo de mejor manera, sino también ser más profesional (en mi caso usando la metodología SCRUM en mi proyecto y tomándola como guía). Por el otro lado, *Diseño de software* permitía usar y comprender los patrones de diseño y con ello asimilar principios de código limpios. Esto ha causado que en el proyecto se hayan utilizado patrones tales como el *Observer* o *Singleton* y que se haya tratado de realizar un código más legible y modular, donde cada funcionalidad está representada donde debe.

Destacar también la asignatura de *Calidad de Software*, la cual ha constituido una parte fundamental en lo que a análisis respecta. Gracias a esta se ha comprendido mejor la inspección de los tipos de requisitos a cumplir durante el desarrollo del proyecto. También ha permitido construir una aplicación de calidad siguiendo con los estándares pertinentes.

Añadir que el periodo de prácticas de empresa ha permitido la mejor comprensión del mundo profesional y de los valores del trabajo en equipo. Mucho de lo aprendido esos meses ha sido complementario a los principios estudiados en el grado.

Finalmente, se han requerido varias competencias transversales desarrolladas durante todo el grado. Las más destacables han sido:

- **Análisis y resolución de problemas:** Esta ha sido una de las habilidades más recurrentes y relevantes durante la realización de este trabajo. Gracias a ella, se ha analizado la problemática a resolver, se ha establecido y seguido un plan de acción, y posteriormente se ha verificado su correcto funcionamiento.

- **Diseño y planificación:** Dado que se trata de un proyecto individual, ha sido crucial contar con esta habilidad, considerando y anticipando posibles limitaciones de recursos y otros aspectos relevantes. Como resultado de este proceso, se ha creado un producto personalizado y adaptado a las necesidades específicas en un periodo de tiempo establecido.

8.2 Conclusiones

Para abordar las conclusiones del proyecto, es necesario afirmar que se ha cumplido con el objetivo principal de desarrollar una aplicación móvil para la gestión de actividades socioculturales en un área geográfica.

Para ello, se ha decidido crear una aplicación multiplataforma para *Android* y *iOS* utilizando *Flutter* como marco de desarrollo. Es necesario destacar que no se tenía prácticamente ningún conocimiento previo al respecto y que, por ello, se podría considerar que el inicio del desarrollo fue , aún si cabe, más tedioso de lo esperado. Con ello, cuando se comenzó a desarrollar la capa de presentación o interfaz gráfica de usuario, muchas veces la desesperación aunaba y se llegó a plantear un cambio de las tecnologías a utilizar.

También se ha estudiado en profundidad el campo de servicios en la nube y el desarrollo *backend*. Tras estudiar diferentes entornos como pueden ser *Azure* o *AWS* se decidió emplear finalmente *Firebase* junto a sus funcionalidades de *Auth*,*Firestore*, *Storage* y *Dynamic Link*. Para el uso de *Firebase* también fue necesario un estudio previo, pues no se conocía absolutamente nada y en el periodo de prácticas de empresa se había trabajado prácticamente como un desarrollador de *frontend*. A pesar del esfuerzo, se puede decir que, sin ningún ápice de duda, la decisión fue acertada .

Se han aplicado los principios y las bases marcadas por la arquitectura limpia o *Clean Architecture*. Además se ha complementado junto con la metodología de tests automatizados TDD, que resulta fundamental en la tarea de validar y validar un sistema de principio a fin.

En el plano personal, ha sido un trabajo que ha puesto a prueba en varias ocasiones los límites de mis conocimientos. Si bien es cierto que ha sido un proceso interesante y muy gratificante, sería conveniente crear un equipo de trabajo para desarrollarlo mejor. Esto se debe pues a que, una persona sin prácticamente conocimiento previo de las tecnologías y con una fecha límite de entrega, no puede alcanzar algunos objetivos que un proyecto así requiere.

Si bien es cierto que, a pesar de que el tiempo era escaso y no estaba aprovechado de forma eficiente debido a la falta de experiencia, se puede considerar que ha merecido

totalmente la pena llevarlo a cabo. El realizar un trabajo de *Full-Stack developer* no es nada sencillo, y menos sin experiencia previa. Sin embargo, se espera haber estado a la altura.

8.3 Trabajo futuro

En un comienzo se planteaba crear una aplicación más vigorosa. Sin embargo, factores como la inexperiencia, la ineficiencia, el tiempo y el coste temporal/económico de algunas acciones, ha conducido a que ciertas funcionalidades hayan quedado en el *Backlogg* para ser añadidas en un futuro.

En una futurible versión de la aplicación se plantea la introducción de un plan de traducción para que esta pueda ser realmente *global*. Pues ahora mismo, la app se encuentra únicamente en castellano y ya se está trabajando en el desarrollo de la versión en inglés para poder ser lanzada al mercado europeo cuanto antes.

Actualmente, se está trabajando ya en una pasarela de pago para los planes y un mapa para elegir las ubicaciones donde se quieren realizar los planes. Actualmente, sólo se puede crear un plan en la zona en la que se encuentra el creador. En cuanto a la pasarela de pago, todavía no se ha implementado debido a que primaban los planes gratuitos. Así, se quiere llevar a cabo este proceso cuanto antes pues es el responsable de la fuente de monetización que se plantea para esta aplicación a futuro.

Hay ciertos añadidos que son interesantes de mencionar y que se consideran para la siguiente versión del proyecto. Por desgracia, no se ha contado con el tiempo suficiente para desarrollarlos. La utilización de más sistemas de autenticación como *Facebook* y *Apple* se quiere implementar de forma inmediata en la próxima versión. También la creación de estadísticas para los planes, de forma que se puedan ver unos ínidices de popularidad así como el número total de entradas vendidas para cada plan. Incluso, se plantea añadir una opción de favoritos para poder guardar planes sin tener la necesidad de adquirir las entradas en el momento.

En un periodo mayor de tiempo, se plantearía que la app tome un caracter de *red social*, de manera que se pudiera *seguir* a los creadores de planes favoritos y que la aplicación avisara cada vez que uno de los creadores favoritos suba un plan. Así se puede ganar, no solo popularidad, sino también confianza entre los usuarios.

CAPÍTULO 9
Anexos

9.1 Anexo: Requisitos funcionales de los actores

En las siguientes páginas se muestran los requisitos funcionales de los diferentes actores del sistema.

Para cada actor se muestran los casos de uso pertinentes a realizar.

Casos de uso (Usuario) :

Caso de uso	Registrar usuario
Descripción	El usuario deberá crear una cuenta usando cualquiera de los servicios disponibles
Precondición	El usuario debe tener cuenta en alguno de los proveedores de servicios disponibles
Secuencia	Pasos: 1 — Acción: Clic a una de las opciones disponibles
Postcondición	El usuario se registrará en el sistema
Excepciones	No se tiene cuenta en ningún proveedor

Caso de uso	Iniciar sesión
Descripción	El usuario deberá iniciar sesión en su cuenta usando un servicio en el que esté registrado
Precondición	El usuario debe tener cuenta registrada en alguno de los proveedores de servicios disponibles
Secuencia	Pasos: 1 — Acción: Clic a una de las opciones disponibles
Postcondición	-
Excepciones	No se tiene cuenta en ningún proveedor

Caso de uso	Buscar planes
Descripción	El usuario podrá buscar planes que estén disponibles para inscripción
Precondición	Debe haber algún plan disponible en la ubicación establecida
Secuencia	Pasos: 1 — Acción: Registro/Inicio de sesión 2 — Dar clic en 'buscar' 3 — Escribir la información a buscar
Postcondición	-
Excepciones	Nombre buscado no existe o es incorrecto

Figura 9.1: Casos de uso y actores.

Caso de uso	Cerrar sesión
Descripción	El usuario podrá salir de su cuenta con tal de entrar en otra o simplemente salir de la app
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos: 1 — Registro/Inicio de sesión; 2 — Ir a ajustes; 3 — Dar clic en 'cerrar sesión'
Postcondición	-
Excepciones	-

Caso de uso	Editar perfil
Descripción	El usuario podrá acceder a su perfil para editar la información de este
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos: 1 — Registro/Inicio de sesión; 2 — Ir a ajustes; 3 — Clic en 'editar perfil'
Postcondición	Se actualizará la información del usuario en el servidor
Excepciones	Información introducida de forma errónea

Caso de uso	Solicitar ayuda
Descripción	El usuario puede pedir ayuda al sistema para resolver sus dudas. Se le pondrá en contacto con un responsable.
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos: 1 — Registro/Inicio de sesión; 2 — Ir a ajustes; 3 — Dar clic en 'ayuda'; 4 — Dar clic en 'solicitar ayuda'
Postcondición	Se enviará un correo electrónico de ayuda
Excepciones	Datos mal introducidos

Figura 9.2: Requisitos funcionales.

<table>
<tr><td>Caso de uso</td><td>Invitar a amigos</td></tr>
<tr><td>Descripción</td><td>El usuario puede invitar a amigos a usar la aplicación mediante un código QR o un enlace</td></tr>
<tr><td>Precondición</td><td>Debe haber iniciado sesión previamente</td></tr>
<tr><td>Secuencia</td><td><table><tr><td>Pasos</td><td>Acción</td></tr><tr><td>1</td><td>Registro/Inicio de sesión</td></tr><tr><td>2</td><td>Ir a ajustes</td></tr><tr><td>3</td><td>Dar clic en ‘invitar amigos’</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Se le envía un enlace invitación a los contactos</td></tr>
<tr><td>Excepciones</td><td>-</td></tr>
</table>

<table>
<tr><td>Caso de uso</td><td>Añadir cupones</td></tr>
<tr><td>Descripción</td><td>El usuario puede usar cupones para obtener ofertas y descuentos en planes</td></tr>
<tr><td>Precondición</td><td>Debe haber iniciado sesión previamente</td></tr>
<tr><td>Secuencia</td><td><table><tr><td>Pasos</td><td>Acción</td></tr><tr><td>1</td><td>Registro/Inicio de sesión</td></tr><tr><td>2</td><td>Ir a ajustes</td></tr><tr><td>3</td><td>Dar clic en ‘añadir cupones’</td></tr><tr><td>4</td><td>Añadir los cupones obtenidos</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Se le añade un cupón de descuento al usuario</td></tr>
<tr><td>Excepciones</td><td>La información introducida es incorrecta o inválida</td></tr>
</table>

Figura 9.3: Requisitos funcionales.

Caso de uso	Guardar tickets
Descripción	El usuario puede acceder en una ventana a los tickets de los planes a los que ha accedido (ya sea por invitación o comprando la entrada)
Precondición	Debe haber iniciado sesión previamente y haber comprado una entrada
Secuencia	Pasos / Acción 1 / Registro/Inicio de sesión 2 / Elegir un plan 3 / Dar clic en 'comprar entradas' 4 / Ir a la sección 'tickets'
Postcondición	Al darle clic se puede ver un QR del ticket
Excepciones	No se tienen tickets

Caso de uso	Hacerse creador
Descripción	El usuario puede convertirse en creador de planes si ha realizado una solicitud previamente al quipo de desarrollo y esta ha sido aceptada
Precondición	Debe haber iniciado sesión previamente y haber sido aprobada la solicitud de creador
Secuencia	Pasos / Acción 1 / Registro/Inicio de sesión 2 / Ir a ajustes 3 / Dar clic en 'Convertirse en creador 4 / Introducir el código de creador proporcionado
Postcondición	El usuario se registrará como un creador a partir de ahora
Excepciones	El código introducido no es válido o el usuario no ha sido aprobado como creador

Figura 9.4: Requisitos funcionales.

Caso de uso	Cambiar datos personales
Descripción	El usuario podrá acceder a su perfil para editar sus datos personales: “nombre”, ”descripción”...
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos: 1 — Registro/Inicio de sesión; 2 — Ir a ajustes; 3 — Clic en ‘editar perfil’; 4 — Editar nombre, descripción...
Postcondición	La información modificada será cambiada en el servidor
Excepciones	Información introducida incorrecta o inválida

Caso de uso	Cambiar foto
Descripción	El usuario podrá acceder a su perfil para editar su foto de perfil
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos: 1 — Registro/Inicio de sesión; 2 — Ir a ajustes; 3 — Clic en ‘editar perfil’; 4 — Cambiar de foto
Postcondición	La información modificada será cambiada en el servidor
Excepciones	-

Figura 9.5: Requisitos funcionales.

Caso de uso	Cambiar ubicación
Descripción	El usuario podrá acceder a su ubicación actual para cambiarla en base a sus preferencias
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos / Acción 1 / Registro/Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'ubicación' 4 / Editar ubicación del usuario
Postcondición	La información modificada será cambiada en el servidor
Excepciones	La ubicación introducida no es válida

Caso de uso	Borrar cuenta
Descripción	El usuario podrá borrar su cuenta
Precondición	Debe haber iniciado sesión previamente
Secuencia	Pasos / Acción 1 / Registro/Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Borrar cuenta'
Postcondición	El usuario borrado se eliminará del servidor
Excepciones	-

Casos de uso (Sistema) :

Caso de uso	Moderar planes
Descripción	El sistema puede valorar si un plan merece estar activo o si de lo contrario no debe realizarse
Precondición	-
Secuencia	Pasos / Acción 1 / Ver planes creados por x usuario 2 / Aceptar o anular plan
Postcondición	El plan se publicará o se eliminará
Excepciones	-

Figura 9.6: Requisitos funcionales.

Caso de uso	Aprobar creadores
Descripción	El sistema le envía al usuario un código de creador o no en base a la valoración obtenida sobre este
Precondición	El usuario debe existir y haber solicitado ser creador
Secuencia	Pasos / Acción 1 / Ver solicitud del usuario 2 / Aceptar o no 3 / Enviar código de creador o desestimación
Postcondición	El usuario puede ser un administrador y una vez ingrese el código se guardará en los servidores
Excepciones	-

Caso de uso	Informar de nuevos planes
Descripción	El sistema mostrará todos los planes que ha aprobado
Precondición	Deben ser planes que el sistema haya aceptado y no anulado
Secuencia	Pasos / Acción 1 / Ver plan creado 2 / Aceptar o no 3 / Mostrar o no en la plataforma
Postcondición	Todos los usuarios podrán ver el plan y se registrará este en el sistema
Excepciones	-

Casos de uso (Creador) :

Caso de uso	Gestionar planes
Descripción	Al creador se le abrirá un abanico de posibilidades por ser creador
Precondición	Debe ser creador para poder realizar la acción
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes
Postcondición	-
Excepciones	-

Figura 9.7: Requisitos funcionales.

Caso de uso	Ver planes creados
Descripción	El sistema mostrará todos los planes creados por el usuario
Precondición	Debe ser creador para poder realizar la acción
Secuencia	Pasos: 1 — Acción: Inicio de sesión Pasos: 2 — Acción: Ir a ajustes Pasos: 3 — Acción: Clic en 'Ver planes'
Postcondición	-
Excepciones	No hay planes creados

Caso de uso	Crear planes
Descripción	El creador podrá lanzar un plan rellenando una serie de datos al respecto
Precondición	Debe ser creador para poder realizar la acción
Secuencia	Pasos: 1 — Acción: Inicio de sesión Pasos: 2 — Acción: Ir a ajustes Pasos: 3 — Acción: Clic en 'Crear plan'
Postcondición	El plan creado será añadido al servidor si el plan es aceptado
Excepciones	La información introducida no es válida o incorrecta

Caso de uso	Modificar planes
Descripción	El usuario podrá modificar cualquier plan que haya creado
Precondición	El usuario debe haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos: 1 — Acción: Inicio de sesión Pasos: 2 — Acción: Ir a ajustes Pasos: 3 — Acción: Clic en 'Modificar plan' Pasos: 4 — Acción: Seleccionar un plan para modificar
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	-

Figura 9.8: Requisitos funcionales.

Caso de uso	Borrar planes
Descripción	El usuario podrá borrar cualquier plan que haya creado
Precondición	El usuario debe haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Modificar plan' 4 / Seleccionar un plan para eliminar
Postcondición	El plan borrado se eliminará del servidor
Excepciones	-

Caso de uso	Ver tickets vendidos
Descripción	El usuario podrá ver todos los tickets vendidos de los correspondientes planes creados
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Ver planes'
Postcondición	-
Excepciones	-

Caso de uso	Ver ingresos acumulados
Descripción	El usuario podrá ver todos los ingresos vendidos de los correspondientes planes creados
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Ver planes' Nota: Los ingresos se realizarán directamente a la cuenta del creador
Postcondición	-
Excepciones	-

Figura 9.9: Requisitos funcionales.

Caso de uso	Cambiar precios
Descripción	El usuario podrá modificar el precio de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos: Acción 1: Inicio de sesión 2: Ir a ajustes 3: Clic en 'Modificar plan' 4: Seleccionar el precio a cambiar
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Caso de uso	Cambiar nombre y descripción del plan
Descripción	El usuario podrá modificar el nombre y descripción de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos: Acción 1: Inicio de sesión 2: Ir a ajustes 3: Clic en 'Modificar plan' 4: Seleccionar nombre y descripción a cambiar
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Figura 9.10: Requisitos funcionales.

Caso de uso	Cambiar número de participantes
Descripción	El usuario podrá modificar el número de participantes de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos: Acción 1: Inicio de sesión 2: Ir a ajustes 3: Clic en 'Modificar plan' 4: Seleccionar el número de participantes
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Caso de uso	Cambiar fecha y hora límite
Descripción	El usuario podrá modificar las fechas y horas de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos: Acción 1: Inicio de sesión 2: Ir a ajustes 3: Clic en 'Modificar plan' 4: Seleccionar las fechas y horas correspondientes
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Figura 9.11: Requisitos funcionales.

Caso de uso	Cambiar foto
Descripción	El usuario podrá modificar la foto de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Modificar plan' 4 / Seleccionar la nueva foto
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Caso de uso	Cambiar ubicación
Descripción	El usuario podrá modificar la ubicación de cualquier plan que haya creado
Precondición	El creador deberá haber creado previamente algún plan que haya sido aceptado
Secuencia	Pasos / Acción 1 / Inicio de sesión 2 / Ir a ajustes 3 / Clic en 'Modificar plan' 4 / Seleccionar la nueva ubicación
Postcondición	El plan modificado quedará guardado en el servidor con la nueva información
Excepciones	La información introducida no es válida o incorrecta

Figura 9.12: Requisitos funcionales.

<table>
<tr><td>Caso de uso</td><td>Cambiar cuenta bancaria</td></tr>
<tr><td>Descripción</td><td>El usuario podrá enviar un correo para cambiar la cuenta bancaria donde se realizarán los correspondientes ingresos de la venta de entradas</td></tr>
<tr><td>Precondición</td><td>Debe ser creador para poder realizar la acción</td></tr>
<tr><td>Secuencia</td><td><table><tr><td>Pasos</td><td>Acción</td></tr><tr><td>1</td><td>Inicio de sesión</td></tr><tr><td>2</td><td>Ir a ajustes</td></tr><tr><td>3</td><td>Clic en ‘Ayuda’</td></tr><tr><td>4</td><td>Enviar un correo solicitando el cambio</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>La información modificada quedará guardada en el servidor con la nueva información</td></tr>
<tr><td>Excepciones</td><td>-</td></tr>
</table>

Figura 9.13: Requisitos funcionales.

9.2 Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**		X		
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**	X			
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**			X	
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este trabajo en relación con los ODS (Objetivos de Desarrollo Sostenible), se puede relacionar de forma plena con los objetivos 3 Salud y bienestar, 4 Educación de calidad,5 Igualdad de género, 8 Trabajo decente y crecimiento económico, 9 Industria, innovación e insfraestructura y 10 Reducción de las desigualdades . En cuanto al resto de objetivos, porque son preguntas de nivel como el hambre, el acceso al agua potable y limpia, la erradicación de la pobreza y la energía asequible y no contaminante, no se puede crear una relación directa.

Si se profundiza más en temas de Salud y bienestar, se considera que se puede abarcar más en lo que respecta al plano de la salud mental. Con ello se considera que tener acceso a cientos de eventos tanto deportivos como educativos en segundos te brinda la posibilidad de mejorar tu bienestar físico y mental creando una atmósfera de aprendizaje cultural que facilite las mejoría física y social. Además permite la relación con personas nuevas y facilita la creación de nuevas relaciones interpersonales ya sea mediante el ejercicio o mediante el acceso a entretenimiento.
Por otro lado, si se abarca el punto 4 de Educación de calidad, considero que se encuentra en la misma atmósfera de lo anteriormente expuesto. El hecho de promulgar el acceso a ciertas actividades permite adquirir nuevos conocimientos tanto sociales como culturales. Esto promueve la mejora de la inteligencia social entre muchas otras.

Si se valora el objetivo 5 Igualdad de género y el 10 Reducción de las desigualdades, es considerable el hecho de que se puedan crear eventos que promulguen esto. Esto implica que este proyecto tiene como fin unir a todas las personas en una comunidad que pueda fomentar, tanto la mejora intrapersonal como interpesonal. Ello sugiere una mayor tranquilidad en cuanto a temas de igualdad, pues dentro de esta comunidad todos somos iguales y no habrá ningún tipo de evento que tenga permitido una exclusión de género o algún tipo de incitación al odio.

Con el apartado 8 de Trabajo decente y crecimiento económico, nos adentramos en algo innegable: esta aplicación favorecerá la economía. Esto se debe a que realmente la aplicación está promoviendo el marketing masivo de todas las actividades que se van a realizar en una determinada área permitiendo, a su vez, la compra de entradas para estas.
Por tanto, al poder venderse entradas de todos los eventos y al poner al día a las personas de estos, se promueve una mayor afluencia de personas en las actividades a realizar. Al mostrar todas las posibilidades a todas las personas y permitirles la compra de entradas, no sólo se favorece a que se realicen más actividades, sino que además se aporta una ayuda al crecimiento económico del sector del entretenimiento.

En lo referente al apartado 9 sobre Industria, innovación e infraestructura, considero que se cumple en un alto grado. Esto se debe a que es una herramienta novedosa que utiliza tecnologías punteras. Además es algo que nunca antes se ha utilizado y que facilita la conexión global de las personas.
Se debe tener en cuenta que estamos hablando de una herramienta con gran facilidad de escalabilidad y que si funciona como se presupone sería tan sencillo como llegar a una ciudad nueva y empezar a disfrutar de todos los planes que esta brinda.

Bibliografía

[1] En: *Top 5 BaaS, You Can Opt for Your Flutter Application* (2021). URL: https://www.rlogical.com/blog/top-5-baas-you-can-opt-for-your-flutter-application.

[2] En: *Kellton Tech Solutions Limited* (2023). URL: https://www.kellton.com/kellton-tech-blog/top-8-mobile-app-development-frameworks-in-2022.

[3] En: *Back4App Blog* (2023). URL: https://blog.back4app.com/baas-providers/.

[4] En: *Application host service BaaS vendors offer automation | top backend provider* (2023). URL: https://www.hitechnectar.com/blogs/top-14-backend-as-a-service-providers/.

[5] En: *Capterra* (2023). URL: https://www.capterra.com/p/160941/Firebase/.

[6] "Arquitectura de software: Qué es y qué tipos existen". En: (2022). URL: https://openwebinars.net/blog/arquitectura-de-software-que-es-y-que-tipos-existen/.

[7] "Best Mobile App Development Frameworks to Use in 2023". En: *Tech Partner for VC-Backed Startups* (2023). URL: https://www.upsilonit.com/blog/top-10-best-mobile-app-development-frameworks.

[8] "Capítulo 4. Lo que aprendí de DDD. Servicios". En: (2021). URL: https://latteandcode.medium.com/cap.

[9] "Cómo crear pruebas nuevas en Android Studio". En: (). URL: https://developer.android.com/studio/test/test-in-android-studio?hl=es-419#create-new-tests.

[10] "Cross-platform mobile frameworks used by global developers 2021". En: *Statista* (2023). URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

[11] This is Finland. "Finlandia: a la cabeza en actividades físicas y deporte". En: (2011). URL: https://finland.fi/es/vida-y-sociedad/finlandia-a-la-cabeza-en-actividades-fisicas-y-deporte/.

[12] "Flutter vs. React Native in 2022". En: *Nomtek* (2023). URL: https://www.nomtek.com/blog/flutter-vs-react-native.

[13] "Framework Flutter como la alternativa móvil a Angular, VueJS o Javascript". En: (2020). URL: https://www.boream.com/insights/framework-flutter-como-la-alternativa-movil-angular-vuejs-o-javascript.

[14] "Freezed". En: (). URL: https://pub.dev/packages/freezed.

[15] Gaptain. "¿Cómo afectan las nuevas tecnologías a la socialización de los menores ?" En: (2019). URL: https://gaptain.com/blog/como-afectan-las-nuevas-tecnologias-a-la-socializacion-de-los-menores/.

[16] "IEEE Recommended Practice for Software Requirements Specifications". En: (1998). URL: https://ieeexplore.ieee.org/document/720574.

[17] "Impacto de la implementación del software de gestión para la fase de análisis de requerimientos funcionales en la Cooperativa Financiera Atuntaqui". En: (2018). URL: http://repositorio.utn.edu.ec/handle/123456789/8223.

[18] Technostacks Infotech. "Top 10 Mobile App Development Frameworks in 2023". En: (2023). URL: https://technostacks.com/blog/mobile-app-development-frameworks/.

[19] "Lo 'freemium' y la publicidad 'in-app', modelos líderes en monetización de aplicaciones". En: *BBVA* (2017). URL: https://www.bbva.com/es/innovacion/freemium-publicidad-in-app-modelos-lideres-monetizacion-aplicaciones/.

[20] "Modelo de negocio freemium: conoce cómo funciona, sus ventajas y 3 ejemplos prácticos". En: *Gabriel Camargo* (2021). URL: https://rockcontent.com/es/blog/freemium/.

[21] "Power Up Your Flutter Development Process by Implementing Clean Architecture and Test-Driven Development". En: (2022). URL: https://betterprogramming.pub/flutter-clean-architecture-test-driven-development-practical-guide-445f388e8604.

[22] "Review On Cross-Platform Mobile Application Development". En: (2022). URL: https://www.ijraset.com/research-paper/cross-platform-mobile-application-development.

[23] "SOLID: los 5 principios que te ayudarán a desarrollar software de calidad". En: (2018). URL: https://profile.es/blog/principios-solid-desarrollo-software-calidad/.

[24] "What's Revolutionary about Flutter". En: (2017). URL: https://medium.com/hackernoon/whats-revolutionary-about-flutter-946915b09514.

[25] "Why Firebase Is A Blessing For Your Mobile App Development?" En: (2022). URL: https://www.techugo.com/blog/firebase-blessing-app-development/..