UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Aplicación de Resultados de Fútbol programada utilizando Flutter

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Llácer Ibáñez, José Luis

Tutor/a: Martínez Zaldívar, Francisco José

CURSO ACADÉMICO: 2023/2024

Resumen ejecutivo

La memoria del TFG del nombre Aplicación de Resultados de Fútbol programada utilizando Flutter debe desarrollar en el texto los siguientes conceptos, debidamente justificados y disctuidos, centrados en el ámbito de la ingeniería de Telecomunicaciones.

CONCEPT (ABET)	**CONCEPTO (traducción)**	**Cumple (S/N)**	**Página**
1.IDENTIFY	1. IDENTIFICAR:		
1.1 Problem, statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	4-6
1.2 Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	12-17
1.3 Setting goals	1.3. Establecimiento de objetivos	S	6
2.. FORMULATE	2. FORMULAR:		
2.1 Creative solution generation (analyis)	2.1. Generación de soluciones creativas (análisis)	S	24-37
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	8-9
3. SOLVE	3. RESOLVER		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	51

3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	46-49

Índice

Capítulo 1: Introducción

Este trabajo se centra en las tecnologías que se han empleado para programar una aplicación que crea un juego en el que los usuarios tienen que intentar predecir resultados de fútbol de partidos reales. Utilizando el *framework* Flutter, un lenguaje de programación multiplataforma se ha creado esta aplicación en la que amigos y compañeros pueden elegir distintos resultados y retarse entre ellos a ver quién de todos consigue acertar el resultado. Algunos resultados serán elegidos por los usuarios y otros se asignarán por sorteo. El objetivo es que todos los resultados estén contemplados y que hagan que el partido esté vivo para los jugadores de la aplicación hasta el último momento.

En este trabajo se combinaron esas dos pasiones, la programación y el fútbol para crear este método de juego. Se ha decidido utilizar este lenguaje ya que ayuda a poder indagar en el campo de la programación multiplataforma y combinar distintos conocimientos, habilidades y herramientas para poder crear una aplicación moderna e interesante en el contexto de ocio.

A lo largo del proyecto se mostrarán las distintas partes de la aplicación, así como sus componentes y con qué herramientas y de qué manera se han desarrollado. Con la ayuda de las figuras se ofrece una visión más gráfica para que el lector pueda observar los distintos datos y el aspecto y composición de las distintas partes de la aplicación.

Capítulo 2: Objetivos

El objetivo principal de este trabajo de fin de grado es el de aprender a crear una aplicación moderna combinando distintas tecnologías aprendiendo sobre su funcionamiento y aplicación.

En primer lugar, se tuvo el dilema de decidir qué tecnología se utilizaba para desarrollar dicho proyecto. Contemplando distintas opciones se llegó a la conclusión de utilizar Flutter ya que de este modo la aplicación podía estar disponible tanto como para Andorid, para iOS y para plataforma web programando un solo código. La programación del front-end de este código es muy intuitiva y ayuda a poder plasmar el diseño con mayor facilidad que en lenguajes más comunes como React, a través de los elementos llamados *widgets*.

El segundo objetivo, se basaba principalmente en decidir, en qué iba a consistir esta aplicación. ¿Qué problema podía resolver? ¿Puede tener utilidad más allá de para ser presentado como trabajo de final de grado? Se me ocurrió poder combinar el proyecto con una de mis pasiones que es el fútbol. Se creó un método de juego simulando una porra de fútbol en el que los usuarios predicen los distintos resultados pero con la excepción de que los resultados no elegidos se sortean entre los jugadores haciendo que siempre haya un ganador entre ellos.

Con la combinación de esta idea y esta tecnología se pretendía conseguir una aplicación completa que abarcara varios aspectos fundamentales, como una interfaz asequible para cualquier usuario, gestión de datos únicos de usuarios y

de información en este caso de fútbol y la buena interacción de todos elementos en un entorno multiplataforma.

Capítulo 3: Metodología de Trabajo

En este apartado se detallarán los distintos procesos que se han empleado para la realización del trabajo. En primer lugar se presentarán las distintas tareas realizadas para el logro de los objetivos planteados, el flujograma con el orden y los tiempos destinados a cada tarea y por último una introducción a las tecnologías empleadas y metodologías realizadas.

3.1 Presentación de Tareas

A continuación se detallarán las distintas tareas que se plantearon para poder conseguir el objetivo marcado con éxito.

- Investigación de las distintas tecnologías para el desarrollo de una aplicación multiplataforma y elección del más apropiado.
- Aprendizaje del lenguaje y *framework* elegido, en este caso Dart mediante el uso de cursos de Udemy, videos de Youtube y documentación.
- Elección de una idea a desarrollar viable con la tecnología apropiada.
- Diseño de las distintas pantallas y fases de la aplicación previas a su desarrollo de programación.
- Programación del front-end con los distintos componentes necesarios.
- Implementación de la tecnología Firebase para el almacenamiento y uso de los datos del backend.
- Pruebas y testeo de la aplicación en distintos dispositivos comprobando su funcionamiento y su responsividad en los distintos dispositivos.
- Elaboración de la memoria del proyecto desarrollado.

3.2 Diagrama Temporal

En la siguiente figura se muestra el diagrama temporal, tanto de las fases del proyecto como de las fases de aprendizaje. Se enseña la primera toma de contacto durante el Erasmus en Milán y el parón que hubo sin progresar con esta herramienta debido a otro proyecto personal que no tenía nada que ver con la tecnología. Una vez finalizado se retomó el contacto con Flutter y se consiguió el conocimiento necesario para llegar a desarrollar dicho proyecto.

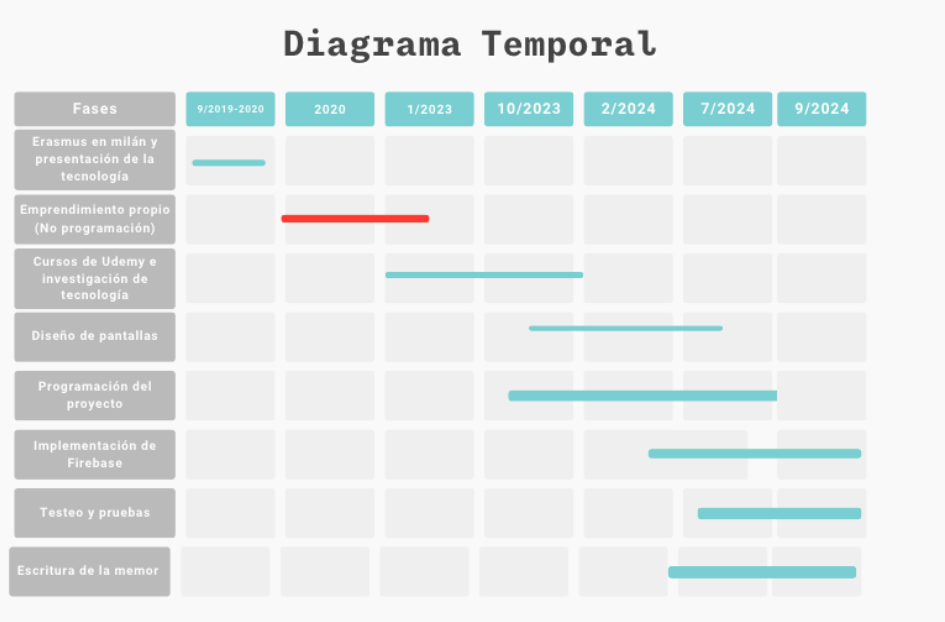

Figura 1: Diagrama temporal del desarrollo del proyecto

3.3 Desarrollo del trabajo y Metodologías empleadas

La programación de las aplicaciones es algo que cada vez es más utilizado hoy en día y que otorga la posibilidad de poder desarrollar una idea o ideas propias. Aunque pueda parecer extraño, en primer lugar se empezó por la investigación de las distintas tecnologías y concretamente la empleada para primero saber que es posible hacer y que no. Antes de desarrollar una idea es fundamental saber si es posible poder llevarla a cabo. Es de esperar que se van a encontrar

obstáculos en el camino y situaciones que no se han podido preveer pero es fundamental intuir si se va a poder llevar a cabo, por lo menos un prototipo del cual establecer una base. Además conforme se consigan los objetivos empleados, debido a la pasión y a la ambición, es posible querer introducir nuevas tecnologías y características que hagan a la aplicación más funcional e interactiva.

Este *framework* lo descubrí durante mis estudios realizados durante mi erasmus en Milán, donde me matriculé en una asignatura cuyo objetivo final era el desarrollo de una aplicación de móvil multiplataforma. Aunque allí obtuve bastantes conocimientos opté por un desarrollo más personal e individual a través de un curso en Udemy donde profundicé mis conocimientos.

El fútbol siempre ha sido algo que me ha apasionado y poder combinarlo con la programación y una idea innovadora me sirvió de motivación para poder llevar a cabo este proyecto una vez vi lo que era capaz de hacer, una vez finalizado la asignatura y los cursos propuestos. Como he dicho antes, sabía que me iba a exponer a problemas y a obstáculos que no podía contemplar pero la experiencia en la carrera de telecomunicaciones, daba la capacidad para poder comprender e intentar resolver cualquier problema que se ponga por delante.

Una vez elegida la idea, se decidió a diseñar las diferentes pantallas y prestaciones que debía tener la aplicación, ya que como se insiste a lo largo de todo el trabajo cuanta más antelación y mejor se planifique más fácil será el desarrollo de la programación mediante código. A diferencia de mediante aplicaciones de diseño, cambiar algo mediante código es mucho más costoso en tiempo. Que la aplicación fuera intuitiva y atractiva era vital y por ello me apoyé en el consejo de una diseñadora gráfica que supervisara los diseños de la aplicación.

Una vez finalizado todo el código y sus testeos en distintos dispositivos, se empezó a escribir esta memoria y a pensar en futuras prestaciones que hagan a la aplicación más intuitiva y dinámica en el uso de sus usuarios.

Capítulo 4: Elección de tecnologías y herramientas

Como se ha mencionado anteriormente, antes de realizar el proceso de programar se tenía que tomar la decisión de qué lenguaje y qué tecnología se iba a implementar y el porqué de esta elección. Hoy en día existe el problema de que existen multitud de lenguajes y de *framework*s y elegir el idóneo puede ser una tarea complicada. Hay que tener claro los objetivos para poder usar ese lenguaje para programar de manera eficiente y robusta la aplicación a desarrollar. Se debe tener en consideración, además del lenguaje el editor de código y otras herramientas que van a ser útiles dentro de nuestro proyecto.

4.1 Elección del lenguaje de programación

Actualmente, y cada vez más, se dispone de una gran oferta de lenguajes de programación a elegir. Entre los requisitos, se destaca, que el lenguaje este enfocado a la programación para teléfonos móviles ya que queremos que nuestra aplicación sea de uso predominante en este dispositivo o tablet en su defecto. También se debe de tener en cuenta, como se ha mencionado antes, que el lenguaje sea conocido, para poder acceder a documentación completa, ya que ante un nivel de principiante, es necesario apoyarse en ella y en tutoriales para lograr el objetivo. Otro criterio que debería de cumplir es el de desarrollo multiplataforma. Para el éxito de la aplicación es necesario que esté disponible en Android y iOS y no crear una barrera a los potenciales clientes.

Por lo tanto, destacan entre la gran variedad de alternativas dos por encima del resto que son React Native, que utiliza JavaScript, y Flutter, que está basado en

el lenguaje de programación Dart. Para ello se van a definir los dos, con sus ventajas e inconvenientes y porque Flutter fue la decisión final tomada.

Flutter es un *framework* de código abierto que fue desarrollado por Google y se presentó en 2017. Al tener esta característica hace que muchos usuarios puedan contribuir a él, haciéndolo más completo y mejorando sus prestaciones. Google actualiza con frecuencia este lenguaje para ofrecer mejor rendimiento y cada vez más prestaciones.

Ventajas:

- Tiene una interfaz de gran rendimiento y adaptada a la experiencia de los usuarios.
- Existe una amplia documentación y comunidad de desarrolladores especializados en Flutter.
- Al ser multiplataforma, la interfaz es única para los distintos tipos de dispositivos.
- En mi propia experiencia personal, el desarrollo del front-end en Flutter es mucho más intuitivo y fácil de programar que en Javacript.
- Según ABAMobile Flutter (1) está integrado con funciones de prueba para poder testear fácilmente la aplicación además de que sus actualizaciones son más sencillas que en Javascript.

Inconvenientes:

- Sólo permite crear una aplicación multiplataforma, no es apto para el desarrollo de aplicaciones nativas. Esto hace que pueda tener algunas limitaciones en el desarrollo de la aplicación en algunas de las plataformas.
- El lenguaje en el que está basado Flutter, Dart, no es tan conocido como Javascript.

React Native, al igual que Flutter, también es un *framework* de código abierto, pero en este caso está basado en Javascript y fue desarrollado por Facebook en 2015. Javascript, a diferencia de Dart, es uno de los lenguajes más conocidos del mundo y permite que, al adquirir el conocimiento de ese lenguaje de programación, se puedan emplear estos conocimientos para desarrollar software o páginas web y tener un perfil más amplio para la búsqueda de trabajo en el futuro.

Ventajas:

- Es un *framework* muy conocido, lo que hace que haya una gran variedad de desarrollo de terceros y documentación para aprender.
- Al estar basado en un lenguaje de programación tan famoso como Javascript, hace que nuestro perfil sea más completo y no tan específico como podría ser con Flutter.

Inconvenientes:

- Al igual que con Flutter, no permite el desarrollo de aplicaciones nativas aunque por otro lado, ese no es el objetivo de nuestro trabajo, pero sí que influye en la personalización que le podemos otorgar a la aplicación.
- Según ABAMobile ofrece una interfaz con un peor rendimiento si la comparamos con Flutter.

Después de analizar todas las opciones, es difícil destacar un claro favorito. Las dos alternativas ofrecen características interesantes que hacen difícil decantarse por una o por otra.

La decisión final fue Flutter, principalmente por su facilidad e intuitividad para desarrollar el front-end. Esta característica parece fundamental a la hora de generar la aplicación, ya que en dispositivos móviles, en mi opinión, la facilidad de interacción entre la pantalla y el usuario es mucho más importante que en una aplicación de escritorio. Aunque Javascript pueda dar un perfil más completo para nuestro futuro, Flutter, por otro lado proporciona un perfil más

específico y diferenciador. Por último también fue determinante a la hora de tomar la decisión el hecho de que la interfaz es de mejor rendimiento y rápida.

Por todo lo anterior se decantó que la mejor decisión es Flutter aunque hay que tener en cuenta que cualquiera de las dos habría sido igualmente válida, ya que las dos tienen documentación y comunidad de sobra para poder aprender y adquirir conocimientos para poder completar el proyecto con éxito.

4.2 Elección de herramienta de editor código

A la hora de elegir el editor de código aunque hay una gran cantidad de alternativas, la herramienta por excelencia es Visual Studio Code. Ningún otro editor de código puede competir por fama y rendimiento con éste. VSCode es un editor de código fuente desarrollado por Microsoft en 2015. Es un software libre y multiplataforma que está disponible para multitud de sistemas operativos como Windows, GNU/Linux y macOs. Según un artículo escrito por Webinars, en una cuesta realizada por Stack Overflow (2) a más de 80000 programadores en mayo de 2021, Visual Studio Code es el entorno de desarrollo más usado con una amplia diferencia sobre su perseguidor. Frente a la pregunta "¿Qué entorno de desarrollo usaste regularmente en el pasado año y con cual quieres trabajar el siguiente?, el 71,86% de los encuestados eligió Visual Studio Code. Se ha consultado la encuesta más actual, realizada en 2023 y el porcentaje ha aumentado al 73,71% de los votos.

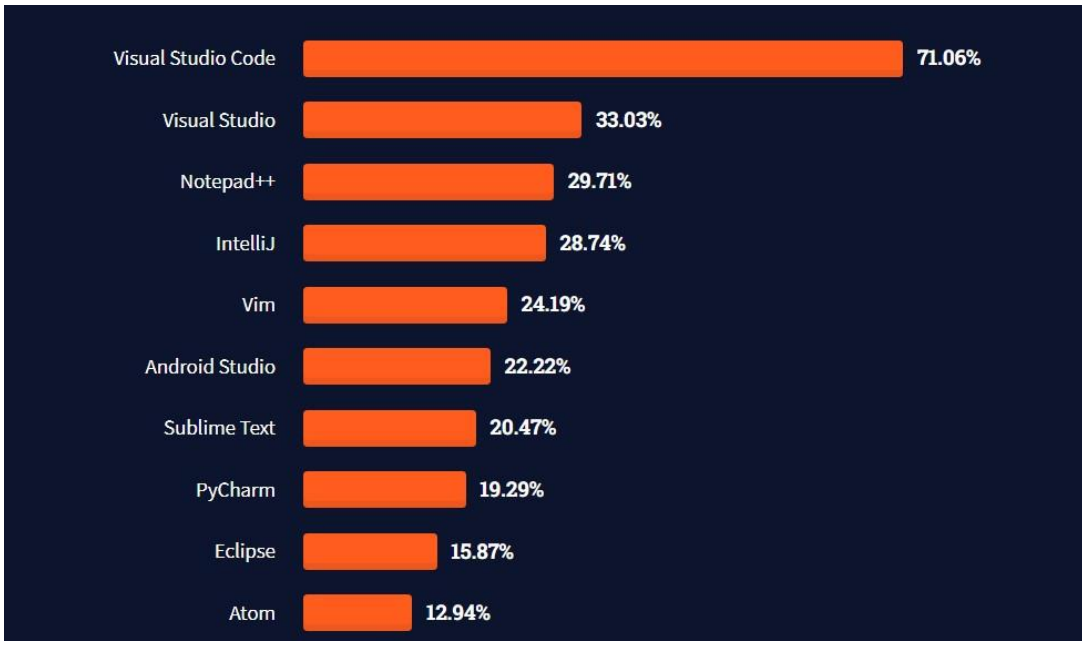

Figura 2: Resultado de la encuesta realizada por StackOverflow en 2023 a la pregunta: Qué entorno de desarrollo usaste regularmente en el pasado año y con cual quieres trabajar el siguiente?, con opción múltiple (3) (4)

Visual Studio Code tiene multitud de características que dan razón de ser a esa gran predominancia en el mercado. Tiene IntelliSense que es una característica relacionada con la edición de código, facilitando la edición con autocompletado y resaltado de sintaxis. También tiene función de depuración que facilita la detección de errores.

Otra característica que, lo diferencia de sus competidores, es el gran uso de extensiones que permiten personalizar más si cabe la edición del código. Con las extensiones, se permite al usuario agregar funcionalidad adicional adecuada al lenguaje y la conexión con otros servicios. Estas extensiones no afectan al rendimiento del editor debido a que se ejecutan en procesos independientes.
Visual Studio Code nos permite probar la aplicación desarrollada en multitud de simuladores de dispositivos tanto en el sistema Android como iOS. Esto permite poder probar si la aplicación es responsiva y por lo tanto, el buen funcionamiento de ésta. Para los dispositivos iOS se debe usar la aplicación

Xcode habilitada para Mac que se detallará más adelante en el apartado de testeo y pruebas.

Sumado a lo ya dicho, se debe mencionar la buena integración con Git que facilita la colaboración de programadores a un mismo proyecto a través de Github extendiendo así más aún su popularidad.

4.3 Elección de la herramienta para guardar la base de datos

Una aplicación completa y escalable tiene que disponer de la capacidad de poder almacenar datos de sus clientes. La elección de esta herramienta es de vital importancia tanto a nivel de programación como en la estructura de la aplicación.
Para almacenar los datos se ha elegido la herramienta Firebase (4). En esta sección se van a detallar sus pretensiones y su uso general, más adelante se concretará como se ha usado específicamente en este proyecto.

Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Fue creada en 2011 pero es propiedad de Google desde 2014. Dispone de una gran variedad de herramientas simplificando la gestión de las distintas necesidades de la aplicación en una mima plataforma. Entre sus funcionalidades se destaca las utilizadas para el proyecto actual, como son la autentificación de los usuarios y el almacenamiento de sus datos y el manejo de otros datos útiles para la aplicación.

Concretamente, para el manejo de datos de la aplicación y de los usuarios se ha utilizado la herramienta llamada Cloud Firestore. Cloud Firestore es una base de datos NoSQL flexible y escalable que mantiene los datos sincronizados de la aplicación en la nube. Su flexibilidad admite estructuras de datos jerárquicas organizando sus datos en documentos, organizados en colecciones. Además, utiliza la sincronización de datos para actualizar los datos de cualquier

dispositivo conectado. Estos datos son almacenados en caché permitiendo a la aplicación leer, escuchar y consultar datos incluso con el dispositivo fuera de línea.

Todas estas herramientas la hacen idóneo para el desarrollo de una aplicación de móvil y el hecho de ser soportado por Google que es la misma entidad que soporta el *framework* empleado, Flutter la hacen valedora de ser utilizada en este proyecto, ya que su conectividad con este *framework* es la recomendada.

Capítulo 5: Descripción de la aplicación

La idea de la aplicación parte de la base de hacer un juego para usuarios que vean y disfruten del fútbol. En España es bastante común o por lo menos antes lo era, jugar a intentar acertar el resultado de un partido en concreto. Para ello, cada uno apuntaba su nombre y el resultado que elegía añadiendo una cantidad de dinero simbólica para participar. Quien acertase el resultado, obtendría el bote acumulado. La aplicación se basa en facilitar el juego a los usuarios pero ofreciendo una serie de alternativas.

5.1 Explicación de la idea a desarrollar

El funcionamiento de la porra, como todos los juegos tienen una serie de limitaciones. Lógicamente se necesita un gran número de participantes para hacer el juego atractivo ya que si sólo juegan cinco personas, hay muchos resultados que se van a quedar sin asignar y por ello, nadie obtendrá el bote, haciendo normalmente que se acumule para el siguiente partido a pronosticar. Aunque el número de jugadores, por el contrario, sea elevado, puede aun así ocurrir que ninguno acierte el resultado ya que hay muchas combinaciones de goles. Además tenemos el factor emocional que genera este deporte. Si se realiza una porra entre seguidores de un equipo rara vez los jugadores pondrán un resultado donde su equipo salga perjudicado.

Por ello nace esta aplicación llamada i1porra. La plataforma permite, además de información sobre los partidos en tiempo real, la posibilidad de que cada jugador elija una serie de resultados, sorteando el resto. De esta manera tenemos dos alicientes diferenciadores, la seguridad de que habrá algún ganador al final del partido y la emoción constante durante el juego ya que en todo momento

durante el encuentro habrá un ganador moméntaneo y en el mejor de los casos dos jugadores a un gol de ganar, manteniendo de esta manera la expectación durante todo el encuentro e involucrando al grupo entero. Este método también pone solución a la limitación de la porra del número de jugadores haciendo el juego atractivo incluso con sólo dos jugadores.

El método de elección de resultados se ha intentado realizar de la manera más justa posible basándose en el juego Catán. Pongamos el caso, por simplicidad de que existen tres jugadores, el jugador 1 elegirá un resultado, luego elegirá otro el resultado 2 y después el jugador 3. Ahora el jugador 3 volverá a elegir otro resultado y el orden se invertirá haciendo que ahora el jugador 2 elija en segundo lugar y el jugador 1 en último lugar. Este método de elección trata de compensar la ventaja del jugador 1 de elegir en primer lugar, compensado por el hecho de que el jugador 3, en este caso el último elija dos resultados de manera consecutiva.

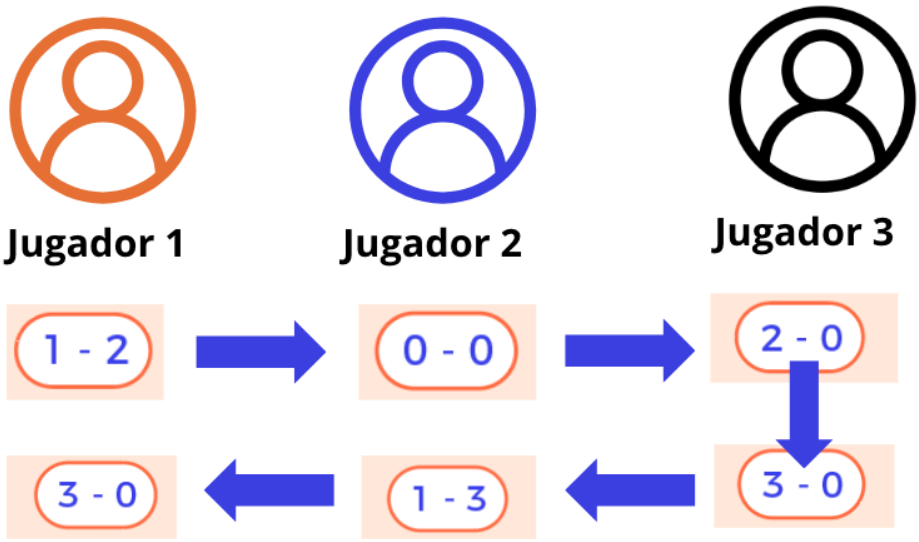

Figura 3. Explicación gráfica del método de elección de la aplicación.

5.2 Elección del nombre y los colores de la aplicación

Debido a este juego y utilizando un juego de palabras se ha llegado al nombre de i1porra, juntando letras y números para hacer más atractivo el logo. Con la i latina minúscula y el 1 simulamos una portería que será la imagen que aparecerá dentro del icono de la aplicación.

Figura 4: Icono de la aplicación.

Los colores elegidos han sido azul y naranja ya que son dos colores no complementarios que dan un gran contraste haciendo que la aplicación sea más visual y por ende más intuitiva.

5.3. API de datos

Como es evidente, es estrictamente necesario el manejo de datos en tiempo real de los partidos de fútbol que se muestran en la aplicación. Por lo tanto, es necesario poder disponer de una API de datos que facilite estos datos a tiempo real.

Una API(*Application Programming Interface*) son un conjunto de definiciones y procedimientos que tienen el propósito de facilitar la comunicación entre aplicaciones de software según una serie de reglas. En este caso, se utiliza una API web que proporciona datos para que los dispositivos vinculados con la aplicación y sus usuarios puedan leer, transferir e interactuar con estos datos.

Se puede encontrar bastante oferta en este mercado. Esta gran variedad de alternativas conlleva un análisis minucioso, la API tiene que poder proveer todos los datos necesarios para el buen funcionamiento de la aplicación. Cuanto más completa sea la alternativa elegida, más fácil será de añadir futuras actualizaciones en el futuro y más información se le mostrará al usuario, mejorando su experiencia.

Hay que tener en cuenta que estos datos suelen conllevar un coste. Normalmente una versión gratuita muy limitada y una versión de pago mejorada que puede dar las necesidades necesarias para su desarrollo. El precio suele ser proporcional a la cantidad de llamadas que se hagan a la API.

Al final entre todas las ofertas se eligió la API llamada “API-football” por su gran disponibilidad de datos en tiempo real y por su precio competitivo rozando los 9 € al mes por 76000 llamadas al día a la aplicación. En estos momentos es mucho más que suficiente para el desarrollo y las primeras pruebas. No obstante, es una gran limitación que se tendrá que abordar más adelante ya que el número de llamadas será sobrepasado con facilidad en cuanto se ponga a disposición de múltiples usuarios. Para obtener datos en tiempo real la aplicación debe estar constantemente llamando a la API para que se actualice e informe a los usuarios de los últimos eventos de cada partido. Gracias a esta tecnología es posible proporcionar datos de los partidos y los equipos a los usuarios como los goles, el tiempo de juego y estadísticas previas.

5.4 Esquema General de la Aplicación

En la siguiente figura se muestra el esquema general de la aplicación, con todas las rutas que el usuario puede recorrer. Es un esquema general de lo que, de aquí para delante se detallará en profundidad.

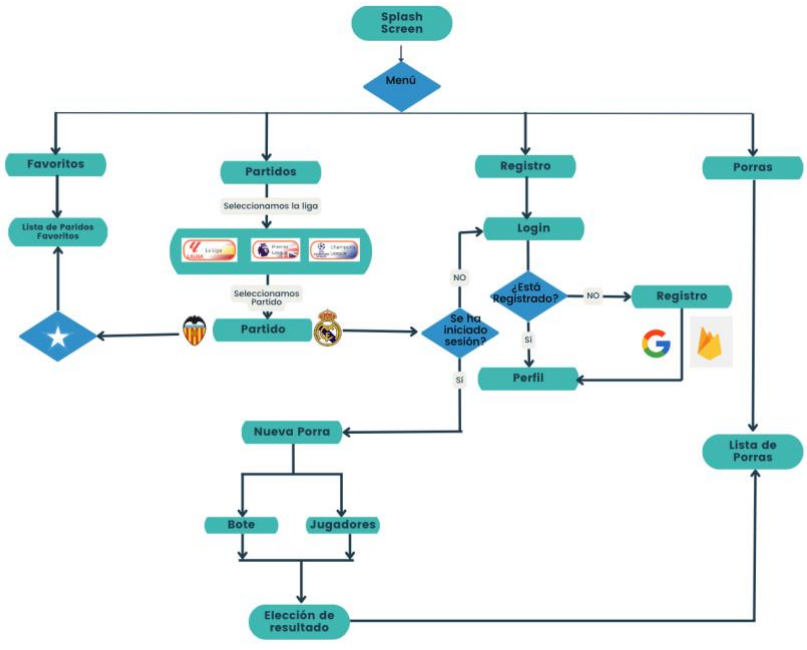

Figura 5: Diagrama de flujo general de la aplicación

Capítulo 6: Desarrollo de la aplicación

Una vez expuestas las herramientas a plantear y maquetada la aplicación se dispone a programar y detallar más todos los elementos que componen el proyecto. En este apartado se complementarán a un nivel de programación la estructura general de la aplicación, los *widgets* más importantes utilizados y la estructura del back-end.

6.1 Estructura general de Flutter y elementos básicos

6.1.1 Estructura general de una aplicación en Flutter

Al comenzar un proyecto de Flutter, se crean una serie de carpetas y archivos que manejan distintas características básicas de la aplicación.

Carpeta android y iOS

Se destacan los archivos que hacen posible la característica multiplataforma que tiene este lenguaje. Con una carpeta llamada Android manejando las características de Android y la carpeta iOS donde se manejan las características necesarias para correr la aplicación en dispositivos con el sistema operativo iOS.

Carpeta build

En la carpeta build se generan los archivos compilados para que la aplicación se ejecute correctamente en cualquier dispositivo. Como ejemplo aquí se encuentra el .apk de Android (lo que podríamos entender como el ejecutable de Android).

Carpeta lib

La carpeta lib contiene todo el código fuente del proyecto. Esta carpeta se irá llenando de archivos y carpetas que irán ampliando las prestaciones de la aplicación mostrando su estructura.

Archivo main.dart

El archivo main.dart es el archivo desde donde se ejecuta la aplicación y se manejan las características más fundamentales como por ejemplo las rutas. Desde aquí también se pueden definir características del front-end y del back-end.

Archivo pubsepc.yaml

Se destaca también el archivo "pubsepc.yaml" donde se documentan los elementos externos que introducimos en la aplicación. Estos elementos pueden ser entre otros imágenes, fuentes y más importante los distintos paquetes de la biblioteca de Flutter que facilitan o habilitan características para programar la aplicación. Por ejemplo para usar el paquete provider que se detalla más adelante se debe especificar en este archivo para poder acceder a la librería de archivos que lo componen. Además de incluir modificaciones estructurales mediante estos paquetes es posible integrar *widgets* creados por otros desarrolladores con sus especificaciones. De esta manera se aumenta la comunidad y se facilita la programación sobre todo para nuevos desarrolladores.

6.1.2 Widgets

La interfaz de una aplicación de Flutter la forman los *widgets*. Un *widget* es cada elemento que compone la estructura de la aplicación sobre todo del front-end. Incluso los textos que se observan también son *widgets*, en este caso un *widget* llamado "Text". Cada widget tiene distintas propiedades prestablecidas para ser modificadas al gusto del programador. Algunas vienen con un valor por defecto. Aquí, continuando con el ejemplo del *widget* "Text", se puede modificar características como por ejemplo el color, el tamaño o el tipo de fuente.

Los *widgets* relacionados entre sí forman un árbol de *widgets*. Cada widget tendrá sus hijos, que serán los widgets que estén dentro de él siendo éste a su vez el padre. Los widgets son subclases de *StatelessWidgets* o *StatefulWidget* dependiendo de si se quiere guardar el estado o no de la aplicación. (5) Estos conceptos se detallarán en el siguiente punto donde se explica el manejo de estados.

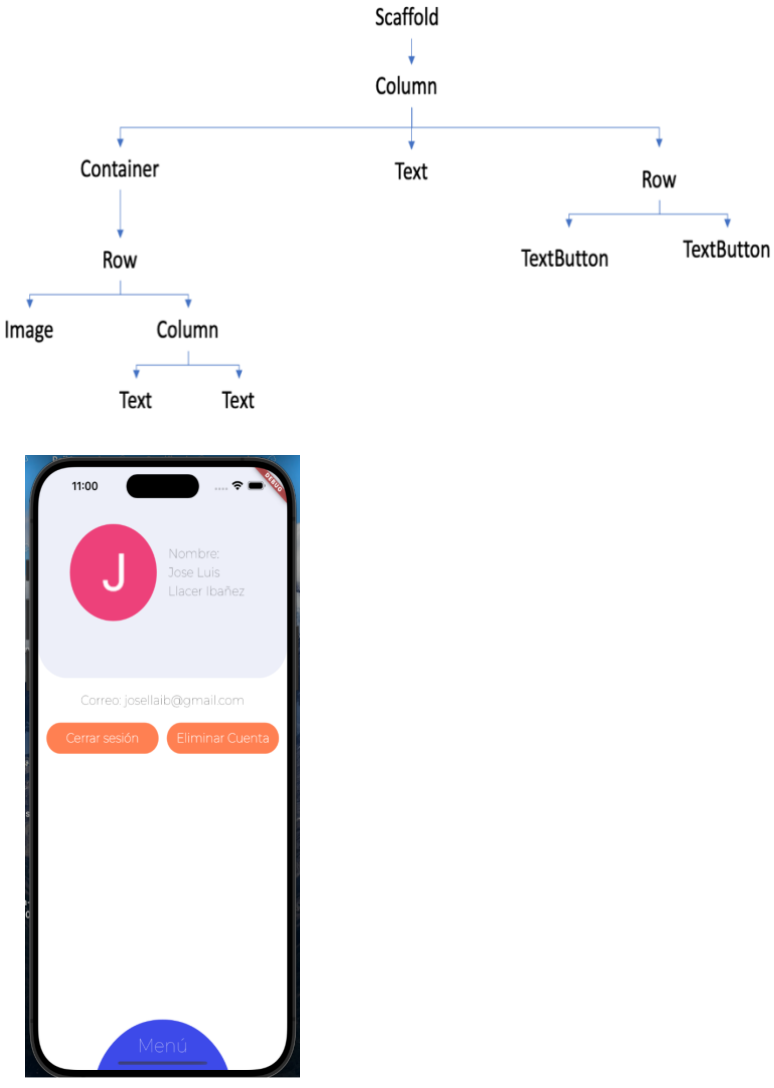

Figura 6: A la izquierda la estructura de widgets de la página de perfil de la aplicación

Como se puede observar en la figura, se puede ver como un *widget* es hijo de otro al contenerlo. En este caso se visualiza la página de perfil de usuario en la aplicación y con el esquema se aprecia como están organizados los componentes en la aplicación. Para organizar los componentes se usan filas con el *widget* "Row" y columnas con el *widget* "Column". El *widget* "Scaffold" es el que se usa para estructurar la página donde se puede configurar el fondo y otras especificaciones.

Para organizar la aplicación los *widgets* se suelen programar en archivos independientes. Así, es más fácil ubicarlos y poder reutilzarlos y poder utilizarlos en otras partes de la aplicación con las características que el programador defina. De esta manera se simplifica el código, haciendo que sea más fácil de editar el front-end, trasladando los cambios a toda la aplicación automáticamente. En la siguiente figura se observa cómo se ha reutilizado un *widget*. En la pantalla de la izquierda es un botón para seleccionar la liga y en la de la derecha se usa como título de esa liga.

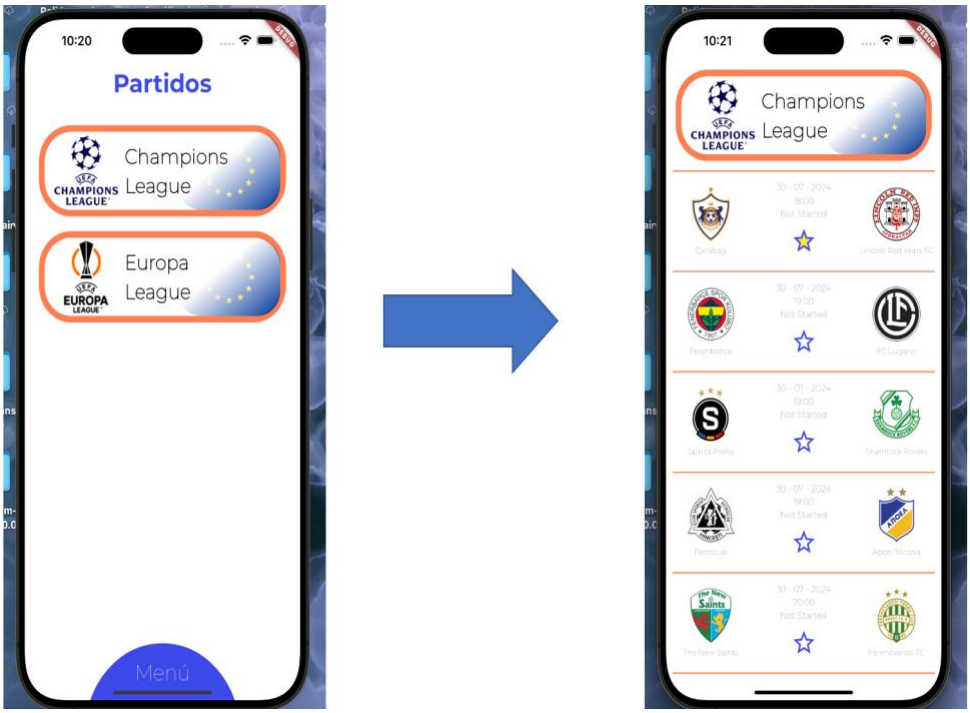

Figura 7: Ejemplo de reutilización del widget como botón en la izquierda y como título en la de la derecha

6.2 Manejo de estados de la aplicación

6.2.1 Introducción del concepto de estado

Una aplicación se compone de estados. Un estado podemos definirlo como la situación o el aspecto que tiene una pantalla de la aplicación en un momento concreto. Una pantalla puede tener un estado o múltiples, dependiendo de la interfaz que se quiera otorgar. Por poner un ejemplo citado anteriormente, el menú escondido sería un estado y el menú desplegado claramente otro.

Se pueden clasificar los estados principalmente en dos grandes grupos: los estados efímeros y los estados de aplicación. El primero es el estado que puede contener un solo *widget*, es un estado como bien explica su nombre, que tiene un inicio y un final y que tras la realización de la acción se acaba. Por ejemplo,

destacar un *widget*, seleccionar el "*Burger Menu*" desplegando sus opciones o el ejemplo de nuestro menú desplegable mencionado en el anterior párrafo. Una vez seleccionada la opción el estado desplegado se acaba y se pasa a un estado escondido, de ahí su característica efímera.

Por otro lado, los estados de aplicación son necesarios compartirlos a lo largo de la aplicación, incluso mantenerlo entre sesiones de usuario si es necesario. En este sentido podemos poner el ejemplo de las preferencias del usuario, un modo oscuro, el estado de autentificación o por ejemplo más concretamente en nuestro ejemplo el número de porras realizadas o los marcadores elegidos por los usuarios durante la partida.

Hay varias maneras de controlar los estados en Flutter. En Flutter, podemos clasificar los elementos llamados *Widgets* en "*Stateless Widgets*", *widgets* con un estado, es decir que no cambiaran nunca y "*Stateful Widgets*" que son *widgets*
que cambiarán o que pueden cambiar de estado. En esta aplicación se ha usado para manejar los estados una herramienta llamada Provider que se detallará en el siguiente punto.

6.2.2 Manejo de estados con Provider

El *provider* (6) es una herramienta de Flutter para manejar los estados de nuestro proyecto. Sobre todo tiene como intención ser usada en estados de aplicación que se mantienen durante la aplicación.

Se puede definir la estructura de los *widgets* como un árbol parecido a un árbol genealógico en el que hay un *widget* que almacena otros *widgets* dentro de él al que podemos definir como sus hijos. El *provider* ayuda a manejar variables o información entre *widgets* con distinto padre o incluso abuelo, facilitando la

codificación en este caso. Como se observa en la figura 7 el provider se sitúa por encima de un *widget* que los dos *widgets* que van a usar la variable comparten. De esta manera podemos informar a los dos *widgets* de la variable y que actúen en consecuencia manejando de esta manera su estado utilizando sólo *Stateless Widgets*.

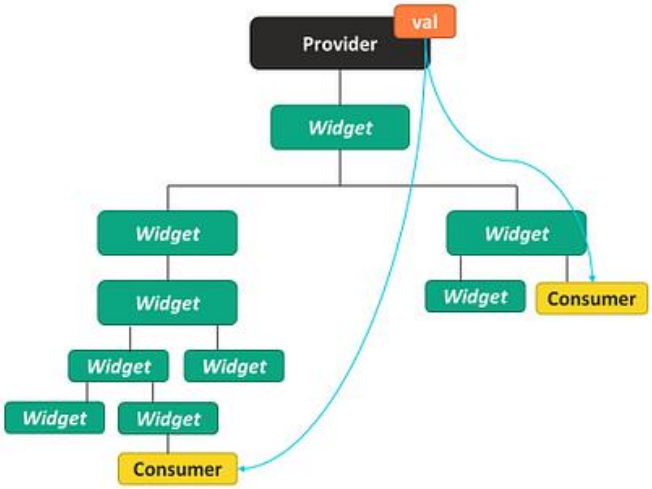

Figura 8: Esquema de árbol de widgets que pretende explicar como funciona el provider en Flutter

Esta característica facilita mucho la programación ya que podemos tener todas las variables de una parte de la aplicación juntas en una parte del código pudiendo manejar las funciones y los estados desde esa ubicación ayudando a tener una estructura mucha más clara, y por lo tanto mucho más flexible a la hora de hacer cambios en la aplicación.

En este proyecto el concepto de *provider* se ha utilizado prácticamente en su totalidad de las pantallas. De este modo, se ha unificado la selección de colores y de tipografía pudiendo usarlos esos colores y características específicas de este proyecto en cualquier parte de éste. También se puede poner el ejemplo en el que se ha usado para hacer las peticiones a la API en la que se recibe la información de los partidos. Tanto el *provider* correspondiente a los colores y el *provider* correspondiente de los partidos, como los otros *providers* se han puesto

en lo más alto del árbol para que su información pueda llegar a cualquier *widget* de la aplicación.

6.3 Autenticación del Usuario

Como se ha explicado antes, es vital para que el proyecto tenga éxito que los usuarios puedan tener cada uno sus propios datos y sus preferencias. Para ello hay que posibilitar que el usuario se autentique, y así, se registre en la aplicación. En la autenticación del usuario es imprescindible tener en cuenta factores como la seguridad ya que una brecha en este ámbito puede perjudicar al usuario y a la plataforma.

6.3.1 Firebase Auth

Como se ha introducido antes, Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil creada por Google. El sistema de autenticación de Firebase se llama Firebase Auth (7) y es un sistema basado en token.

Esta plataforma ayuda a programar la autentificación de manera clara y sencilla. Utilizando el paquete de Firebase y de Firebase Auth de pub.dev y mediante una instalación en la aplicación a través de la consola de Firebase es posible habilitar la identificación de los usuarios en la aplicación sin un complejo código. Es importante resaltar que además de habilitar la autentificación del usuario con las credenciales típicas como el usuario y la contraseña es posible hacerlo mediante el uso de otras plataformas como Google, Facebook, teléfono…

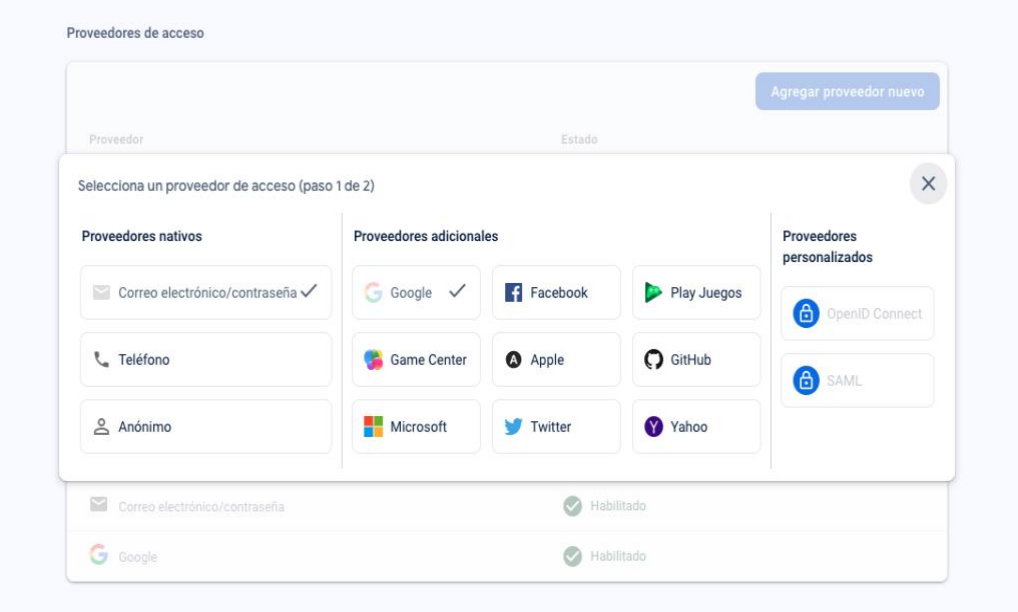

Figura 9: Captura de pantalla de la plataforma Firebase mostrando la variedad de opciones de autenticación disponibles.

Firebase Auth facilita un análisis acerca de la autenticación de los usuarios (7). Se puede observar en el perfil personal del desarrollador, estadísticas demográficas al igual que los distintos métodos usuarios para identificarse por parte de los usuarios, pudiendo de esta manera, mejorar y priorizar alguno de ellos.

Es completamente insuperable el precio de este servicio, ya que salvo la autenticación por SMS, todos los demás servicios tienen un coste nulo. Aun así el coste por SMS es bastante económico ya que solo sería necesario pagar a partir de 10000 usuarios por mes.

	Plan Spark (gratuito)	Plan Blaze (según el consumo)
Autenticación por SMS	10,000 por mes para EE. UU., India y Canadá 10,000 por mes para todos los demás países	$0.01 $0.06
Autenticación (todos los demás servicios)	Gratis	Gratis

Figura 10: Tabla mostrando los planes de pago de Firebase Auth a través del teléfono móvil.

6.3.2 Autenticación en i1porra

Concretamente en el proyecto se ha comenzado por introducir dos métodos de autenticación. En primer lugar se habilitó el registro de usuarios mediante su correo y su contraseña. A pesar de que esta solución es universal, parecía necesario introducir como mínimo otro método para facilitar el registro y para conseguir más información del usuario.

Por este motivo, se decidió habilitar el registro de usuario por Gmail. Se eligió esta plataforma debido a su popularidad e información compartida. Con el registro de un nuevo usuario la aplicación puede acceder a su nombre y a su imagen mostrando la página de perfil más personal y poder personalizar más la aplicación al usuario. Se le permite al usuario acceder fácilmente sin tener que confirmar el correo electrónico para verificar su identidad y se aumenta la probabilidad de que el correo introducido sea el habitual del usuario. Consiguiendo esta comunicación directa con el usuario, se le puede informar acerca de promociones, actualizaciones y notificación que se crea que le pueda interesar.

A nivel técnico se destaca la facilidad de programación que se ha comentado anteriormente.

6.4 Base de datos

Una vez se han obtenido los datos del usuario es necesario obtener los datos que posibilitan en funcionamiento del principal objetivo del proyecto. Con la intención de unificar todo en una misma plataforma y facilitando así la programación y la adaptación se ha decidido utilizar la plataforma de Firebase perteneciente a Google para almacenar dichos datos.

6.4.1 Firestore Database

La base de datos de Firebase es una base de datos escalable NoSQL en la nube. Está diseñada para proporcionar a los desarrolladores una forma de sincronizar, recuperar y almacenar datos en tiempo real. Entre los beneficios que se mostrarán a continuación se destaca el hecho de que al ser de Google, comparte empresa con la autenticación descrita en el anterior punto y con el lenguaje de programación empleado para el desarrollo de la aplicación.

Esta base de datos permite la sincronización en tiempo real que es fundamental para el objetivo de la aplicación ya que se tiene que trabajar con los datos en tiempo real pertenecientes a los distintos encuentros de fútbol ofrecidos. Aunque sea necesario disponer de conexión para obtener estos datos, Firebase permite soporte sin conexión almacenando datos localmente en los dispositivos permitiendo ofrecer otras prestaciones a los usuarios sin conexión si fuera necesario.

Firestore ofrece una fuerte escalabilidad permitiendo manejar millones de conexiones simultáneas y grandes conjuntos de datos sin comprometer el rendimiento. Esta característica permite poder crecer y promocionar la aplicación sin miedo a una saturación, por lo menos en este ámbito.

6.4.2 Estructura de la base de datos

La estructura de la base de datos que ofrece esta plataforma tiene un estilo jerárquico basado en el concepto de colecciones y documentos.

Las colecciones son contenedores que contienen documentos. Estas colecciones permiten organizar los datos de una manera que facilita la consulta y la administración. Estas colecciones, a su vez pueden contener otras colecciones que posibilitan una mayor división de agrupaciones lógicas de datos relacionados, haciendo fácil poder acceder a datos concretos.

Los documentos son los registros individuales dentro de Firestore que contienen los valores. A estos documentos se les identifica con un ID y admiten varios tipos de datos como cadenas, números, booleanos, matrices, mapas… facilitando la programación y usando esos mismos tipos de datos para poder mostrarlos al usuario en la interfaz. A su vez esos documentos pueden tener una sub-colección dentro de éstos.

Para el proyecto de i1porra en concreto se ha establecido la siguiente estructura de datos que se detalla en la siguiente figura. La colección raíz es la colección llamada userData donde se almacenan los datos de los usuarios registrados en la aplicación. Cada documento dentro de userData tiene como nombre el id único de cada usuario y tienen dentro dos colecciones llamadas data y porras. La primera lógicamente contiene los datos del usuario y la segunda contiene las distintas porras que ha creado ese usuario. Cada porra tiene su id único y una colección de campos con los detalles de la porra. Entre estos campos se destaca un array de objetos que especifican el nombre y los resultados de cada jugador, los datos de los equipos que disputan el partido y la fecha del partido. Todos estos datos son utilizados a lo largo de la *app* para comunicárselos al usuario en el momento concreto.

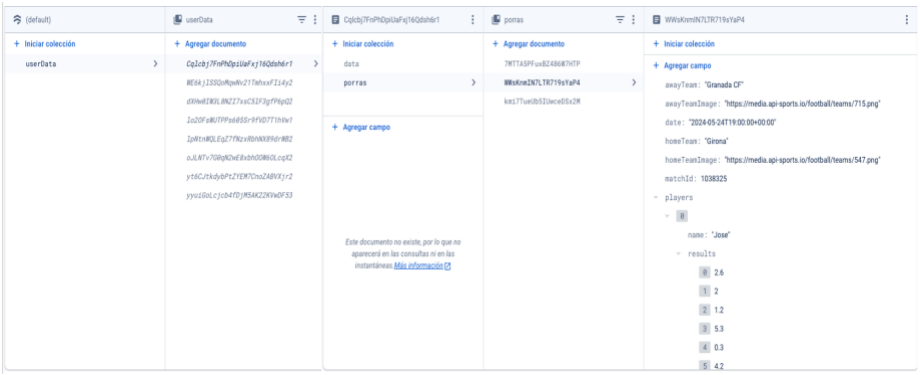

Figura 11: Firestore Database de la aplicación i1porra

Capítulo 7: Front-end de la aplicación

Una vez se han decidido las diferentes tecnologías y recursos a emplear para el desarrollo del proyecto, se tiene que programar la interfaz del usuario. Esta parte es de extrema relevancia ya que es la comunicación del usuario con la aplicación. La interfaz debe de ser atractiva y principalmente intuitiva. El usuario debe de poder seguir con facilidad el proceso de la aplicación y tiene que parecerle obvio que tiene que pulsar y que tiene que rellenar al usar la aplicación. Si la interfaz no es clara, puede perder interés y tener una mala opinión del producto. Es decir que el producto debe tener buena usabilidad y experiencia de usuario.

En este capítulo se mostrará las diferentes pantallas para poder visualizar la interfaz del usuario. En cada pantalla se explicará su naturaleza y objetivo detallando técnicamente los aspectos más relevantes como los widgets más relevantes que se han utilizado.

7.1 SplashScreen

La *splash screen* es la página que aparece justo antes de iniciar la aplicación. Tiene una doble función, por un lado, mostrar al usuario que la aplicación se está inicializando y por otro dejar tiempo para que los recursos se carguen de manera que la página de inicio esté lista.

Figura 12: SplashScreen de la app

7.2 Menú

El menú es un aspecto muy relevante en la programación de la aplicación. Es la herramienta que proporciona el camino rápido por la aplicación, tiene que ser completo y accesible pero al mismo tiempo no ocupar gran parte de la pantalla de forma que moleste al usuario o reste importancias a otros widgets de excesiva relevancia en la aplicación. Existen varias alternativas, de las que podemos destacar la barra permanente de menú llamada "*BottomNavigationBar*" (7),situada en la parte inferior en la pantalla o a veces en la superior y el conocido "*Burger Menu* (8)" o "*Drawer*" que muestra un desplegable de opciones.

Al ser una *app* que simula el juego se ha optado por una "*BottomNavigationBar*" animada. Está situada en la parte inferior, pero al ser pulsada se despliega ocupando la pantalla mostrando claramente las disponibles rutas al usuario. La forma semicircular quiere simular el límite más avanzado del área de un campo de fútbol.

Este menú habilita al usuario a navegar de manera rápida e intuitiva a las diferentes secciones de la aplicación.

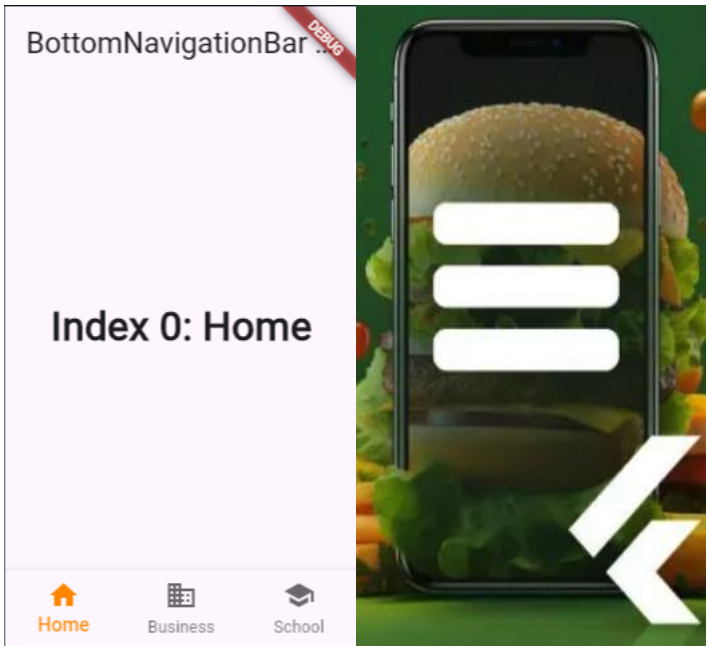

Figura 13: A la izquierda la "BottomNavigationBar" y el ícono por excelencia del "Burger Menu"

Figura 14 diferencia entre el menú recogido y desplegado en la app i1porra

7.3 Pantalla de inicio de sesión

Esta pantalla habilita a los usuarios a registrarse y a poder introducir datos dentro de la aplicación. De esta manera se obtiene información única de cada jugador y un perfil para jugar. Esta comunicación con el usuario se realiza a través de un widget llamado *TextField* que permite al usuario poder escribir en él. Este *widget* permite escribir un texto llamado "*hinText*" que informa al usuario que información tiene que escribir. Este widget se usará en muchas fases del proyecto.

Además, hay que destacar el botón de inicio de sesión de Google facilitando el registro del usuario mediante su Gmail, tal como se ha explicado en el apartado de autenticación.

Figura 15: Pantalla de inicio de sesión

7.4 Lista de Ligas y Partidos

Los partidos de fútbol están clasificados en las diferentes competiciones que se facilitan al usuario. Estas competiciones aparecen sólo si va a ocurrir un partido en los próximos 20 días. En i1porra se asume que su uso será en los días previos al partido, de esta manera no se llena esa pantalla de infinidad de competiciones.

Una vez se pulse en el Widget de una liga, aparecerán los partidos disponibles de esa competición. Aquí el usuario podrá seleccionar el partido en el que esté interesado para realizar la porra.

Las dos listas se han programado mapeando a través de un *ListViewBuilder* que recorre la lista de partidos, en este caso un array de objetos y selecciona la información correspondiente en cada uno de ellos. Se ha creado un método para que los partidos aparezcan en orden cronológico y si el partido se disputa el mismo día el fondo del *widget* cambia de color para hacer que destaque. Se destaca también el icono de la estrella que permite a, usuario guardar sus partidos en la sección de favoritos única para cada usuario mostrada en la imagen de la derecha de la figura 12.

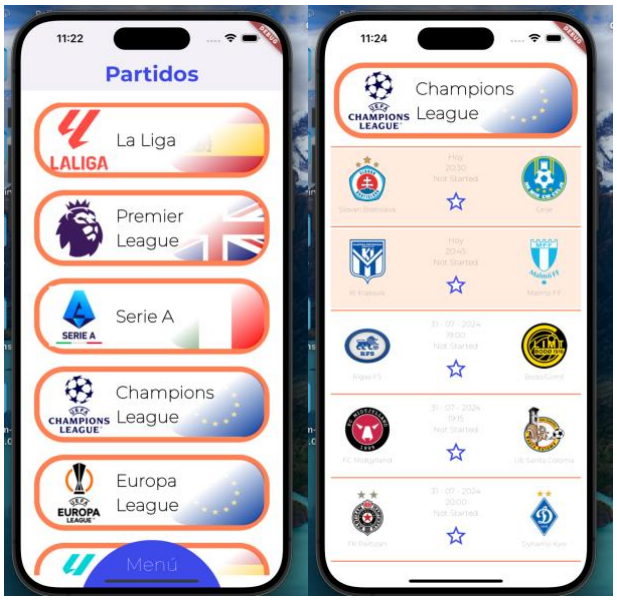

Figura 16: A la izquierda la pantalla con las competiciones y a la derecha la lista de los partidos de esa competición.

7.5 Pantalla de partido

Una vez el usuario selecciona en la pantalla del partidos en el que quiere realizar la porra aparecen las porras anteriores si es que ya ha hecho una para ese partido además del un botón para empezar una nueva. Para evitar que el usuario no realice una porra sin iniciar sesión, en el caso de no haberse iniciado habrá un botón que le llevará directamente a la página de registro. Se incluye un deslizante en el que se diferencian los títulos porra y resumen, para que el usuario pueda alternar entre la pantalla de las porras realizadas y observar datos y estadísticas del partido en cuestión. Por el momento, solo se muestra la clasificación de la liga, pero tal como se comenta en el partido de futuras actualizaciones,

el objetivo es incluir estadísticas diversas para que el jugador tenga más información para poder intentar acertar el resultado.

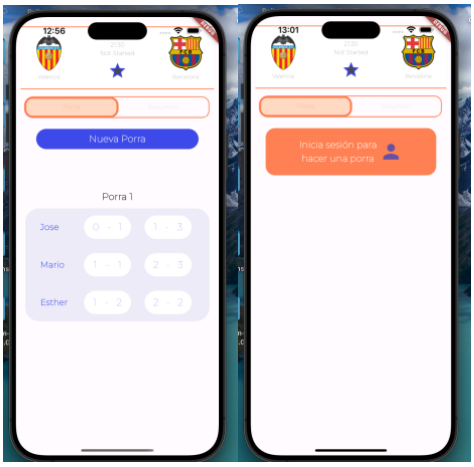

Figura 17: A la izquierda la página del partido con la sesión iniciada, a la derecha sin iniciar sesión.

7.6 Pantallas para crear la porra

Para el proceso de crear la porra se ha decidido crear varias pantallas. Es necesario como, se ha mencionado durante todo el trabajo que la aplicación sea intuitiva y por ese motivo se han usado múltiples pantallas para que el proceso sea más claro y haya más espacio para los botones y los widgets necesarios en el proceso.

7.6.1 Pantalla de Jugadores

El primer paso es introducir a los jugadores que van a realizar la porra. Los jugadores aparecen en el "*Container Widget*" azul a medida que se introducen en el campo de texto (*TextField Widget*). Abajo aparece otro *TextField* para introducir el bote por jugador, y un botón para pasar a la siguiente pantalla donde se eligen los resultados.

En la esquina superior derecha se encuentra un widget que muestra un *Alert Dialog* con las instrucciones del juego. Este *widget* nubla el fondo de la pantalla y muestra las instrucciones para realizar una porra. Estas instrucciones pretenden guiar al usuario insistiendo en el principal objetivo de que aparte de que funcionen todas las prestaciones de la aplicación ,sea intuitiva.

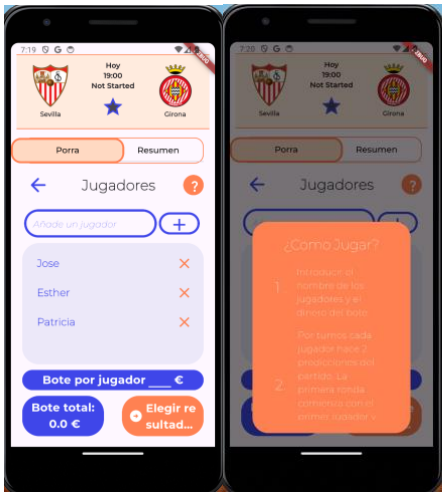

Figura 18: A la izquierda la pantalla para introducir los jugadores y el bote y a la derecha la misma pantalla difundida por el cuadro de diálogo con las instrucciones.

7.6.2 Pantalla de resultados

Una vez introducido el nombre de todos los participantes de la porra llega el momento de elegir los resultados. Los resultados se eligen tal cual se ha explicado en el apartado 5.1 donde se explica la idea a desarrollar, siguiendo el método del famosos juego de mesa catán. El primero elige en primer y en último lugar haciendo que el último de la lista elija dos veces consecutivas un resultado, para hacer, dentro de lo posible, más justo el método de selección.

Debajo del marcador se observan dos botones en los que se muestran el nombre del siguiente y anterior jugador para poder navegar hacia un lado y hacia otro en

la selección. A medida que se eligen los resultados aparecen al lado del jugador correspondiente. Se observa también un botón booleano en el que se permite sortear el orden de elección de los jugadores.

Cuando se acaba la selección se los resultados el botón para pasar al siguiente jugador cambia de color y muestra la palabra empezar, para dar comienzo a la porra y con ello al juego. Este botón lleva al usuario al menú donde se muestran todas las porras realizadas por el jugador.

7.7 Pantalla de porras

A la pantalla de porras que se va a presentar ahora se navega a ella desde el menú. Aquí se observa una lista de las porras anteriores, presentes y futuras realizadas por el jugador. En el *widget* creado se ha intentado compactar toda la información necesaria como los jugadores, el partido y el estado de éste, al igual que un icono para poder eliminarla de la base de datos.

Capítulo 8: Futuras Mejoras y actualizaciones

A pesar de que el proyecto cumple con los requisitos más que necesarios para hacer que la aplicación funcione, siempre hay que pensar en posibles mejoras para poder hacer que el proyecto crezca y se consolide como primera opción para los usuarios, consiguiendo ofrecer, una experiencia más completa al usuario. Estas mejoras traen normalmente consigo el aprendizaje de nuevas tecnologías o herramientas e incluso de nuevos lenguajes de programación. También hay que tener en cuenta que implementar estas tecnologías en la aplicación no es siempre fácil sobre todo por la parte de apple que al ser un sistema operativo más

8.1 Actualización de minuto de resultados

Como cualquier aplicación de deportes que se precie, se necesita que el minuto y el resultado sea en vivo constantemente. Esta característica trae consigo algunos inconvenientes. El más preocupante es la cantidad de llamadas a la API que hay que realizar para poder otorgar esta experiencia al usuario ya que habría que llamar constantemente para poder mantener el resultado del partido acorde con la realidad. Actualmente se tiene una limitación de las llamadas, que a pesar de estar en 76000 por día, no puede suplir la necesidad en cuanto se tenga un número de usuarios un poco elevado.

Para resolver el problema se pretende crear un servidor que centralice estas llamadas y que los usuarios hagan las llamadas a éste. Este servicio se puede obtener a través de Google o con Amazon con un AWS.

8.2 Mayor número de estadísticas

Al igual que los resultados en tiempo real comentados en el apartado anterior también harían la experiencia al usuario más gratificante tener un mayor número de datos para tener más información de cara a elegir el resultado. Se quiere en el apartado de resumen en la pantalla de partidos un mayor número de datos a parte de la clasificación de la liga, como resultados anteriores, o probabilidad de los distintos resultados.

8.3 Notificaciones

Para mantener una comunicación constante entre la aplicación y el usuario es necesario el uso de notificaciones. Existen varios tipos de notificaciones como son las *"push notifications"*, o las notificaciones locales. La diferencia radica en si estas notificaciones son recibidas mientras el usuario está interactuando con ella o no en ese momento. Las *push notifications* llegan incluso cuando la aplicación está bloqueada. Este aspecto está incluido en la gran mayoría de aplicaciones e incita al usuario a utilizar de manera recursiva la aplicación.

La implantación de esta característica es complicada de implementar ya que requiere permisos de los usuarios para poder habilitarlas.

8.4 Presencia en Apple Store y Google Play

Por último pero no por ello menos importante la aplicación tiene que estar disponible en las tiendas de aplicaciones de los dos sistemas operativos mayormente utilizados. La presencia tanto en Apple Store como Google Play requiere un coste asociado. En Apple Store es necesario acreditar una licencia de *Apple developer* que tiene un coste de 99€ anuales en el momento de la escritura del trabajo. En Google Play es un único pago de 20€, pero a diferencia de iOS en esta plataforma consiste de un pago único, no una suscripción anual.

Capítulo 9: Testeo y pruebas

Durante la realización del proyecto ha sido vital el testeo de la aplicación para poder detectar errores y mejoras para actualizar la aplicación. Como se ha mencionado antes, una de las grandes ventajas de Flutter es que es multiplataforma. Gracias a esta característica podemos probar la aplicación en el sistema Android y en iOS.

Con el editor de código Visual Studio Code es posible crear dispositivos virtuales de distintos tamaños y sistemas operativos para probar la aplicación. Al instalar la aplicación en dispositivos de distintos tamaños podemos probar la responsividad de la aplicación y comprobar que se ajusta también a los distintos dispositivos que pueden usar los futuros usuarios.

En el sistema operativo Android, al ser más abierto el proceso de probar la aplicación es sencillo. Tan sólo con la carpeta de Android que Flutter facilita ya se tiene todo lo necesario para probar el funcionamiento de la aplicación en esos dispositivos.

Por otro lado, como ya es conocido, el sistema operativo de iOS es más cerrado. Gracias a programar en un mac se ha podido probar la aplicación en dispositivos virtuales de Apple, en el caso de no haber podido contar con este equipo, habría sido necesario la instalación de una máquina virtual. Este requisito es indispensable, ya que es necesario instalar la aplicación en los dispositivos mediante una aplicación llamada Xcode perteneciente a mac. Esta aplicación es necesaria tanto si se requiere probar en un dispositivo virtual o en un dispositivo físico.

Capítulo 10: Conclusiones

Realizar este proyecto ha sido un proceso complicado y costoso. Por una parte, cabe destacar la inexperiencia, en el momento del inicio tanto en el lenguaje como en la faceta de programador en general. Por otro lado, creo que aprender a realizar este tipo de proyectos abre muchas puertas ya que puedes aplicar lo aprendido para desarrollar otros proyectos incluso con otras tecnologías, además de abrirte muchas puertas en el mercado laboral.

Aparte de la inexperiencia también había que lidiar con el reto de poder diseñar la aplicación, consiguiendo, que además del requisito mínimo de que funcione, fuese intuitiva y agradable para la vista del usuario. Ya que por muy bien que funcione y buen código que tenga, carece de utilidad el proyecto si el usuario no la usa debido a su dificultad. Es importante anticiparse en esta fase ya que como se ha dicho a lo largo del proyecto cambiar cosas con código requiere de mucho más tiempo y mayor complejidad.

Creo que los estudios de telecomunicación han ayudado bastante a la hora de realizar el proyecto ya que durante todos los estudios se potencia mucho la capacidad analítica del estudiante y su capacidad de resolver problemas y la programación consiste en esencialmente eso, saber resolver problemas con las herramientas que se manejan.

Bibliografía

Bibliografía

1. **ABAMobile.** www.abamobile.com. *abamobile*. [En línea] 2022. https://abamobile.com/web/diferencias-react-native-vs-flutter/.
2. **Flores, Frankier.** Open webinars. *Open Webinars*. [En línea] 2022. www.openwebinars.net.
3. **Overflow, Stack.** Stack Overflow. *Stack Overflow*. [En línea] 2023. https://survey.stackoverflow.co/.
4. **Firebase.** Firebase. *Firebase*. [En línea] 2023. firebase.google.com.
5. **Gotoopo, Jean Paul.** Medium. *medium.com*. [En línea] 21 de Enero de 2020. https://jpgotopo.medium.com/cap-6-diferencias-entre-statelesswidget-y-statefulwidget-b5187f864280.
6. **Sánchez, Daniel Herrera.** Medium. *medium*. [En línea] 14 de Marzo de 2022. https://medium.com/bancolombia-tech/flutter-provider-qué-es-para-qué-sirve-y-cómo-utilizarlo-7388d4b206b5.
7. **Flutter.** Flutter. *https://api.flutter.dev*. [En línea] sf. https://api.flutter.dev/flutter/material/BottomNavigationBar-class.html.
8. **Sorathiya, Nidhi.** Dhiwise. *www.dhiwise.com*. [En línea] 20 de Agosto de 2024. https://www.dhiwise.com/post/crafting-the-perfect-flutter-hamburger-menu.
9. **back4app.** back4app. *blog.back4app.com*. [En línea] https://blog.back4app.com/es/que-es-firebase-authentication/.
10. **Firebase.** Firebase. *firebase.google.com*. [En línea] https://firebase.google.com/docs/firestore?hl=es.
11. **Sánchez, Daniel Herrera.** Medium. *medium.com*. [En línea] 25 de Abril de 2022. https://medium.com/bancolombia-tech/gestión-de-estados-en-flutter-qué-es-cómo-puedo-aplicarlo-5f2a7b168c62.