

**PROYECTO**  
**BUNNY KINGDOM DELUXE**  
**DELUXE APPS**

**CICLO FORMATIVO DE GRADO SUPERIOR:**  
**Desarrollo de Aplicaciones Multiplataforma**

**AUTORES:**  
**Jorge María Huguet**  
**Miguel Ángel Segovia Freeman**

### **Licencia**

**Esta obra está bajo una licencia Reconocimiento-Compartir bajo la misma licencia 3.0 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.**

## **DEDICATORIAS/AGRADECIMIENTOS**

Gracias a los testers Gonzalo Rozada, Andrés Calamardo, Carlos Teso, Beatriz Aguayo, José Aniceto por las pruebas de usuario realizadas para mejorar la usabilidad de la app.

Gracias a Luis Jaén por la revisión de la traducción.

Gracias a las diseñadoras gráficas Patricia Logghe y Leticia López por los diseños, correcciones de color y sugerencias de estilos.

## RESUMEN

Deluxe Apps es una empresa compuesta por dos socios: Jorge María Huguet y Miguel Ángel Segovia Freeman.

La empresa nace debido al aumento de la popularidad de los juegos de mesa y su “deluxificación”, término popular para indicar adoptado por la comunidad de los juegos de mesa que indica que se ha mejorado el juego de mesa, ya sea pintando las miniaturas que lo acompañan, imprimiendo componentes en 3D, crear insertos para que los componentes no se muevan, o como en el caso de esta empresa desarrollando aplicaciones para ser usadas junto al juego.

Deluxe Apps se dedica al desarrollo de aplicaciones para juegos de mesa buscando dinamizar las partidas y aprovechar los puntos fuertes de cada juego.

El objetivo de la empresa es mejorar la experiencia de los jugadores desarrollando aplicaciones que fomenten la competitividad sana, cooperación, dinamizar las partidas...

Los posibles clientes son aquellas empresas desarrolladoras o distribuidoras de juegos de mesa que quieran “deluxificar” su juego.

Para el proyecto Deluxe Apps ha desarrollado la aplicación Bunny Kingdom Deluxe, donde han sido contratados para dinamizar el juego y fomentar la competitividad, para ello se ha la aplicación tiene 3 actividades principales, Partidas, Perfiles y Ranking.

En la actividad partidas los usuarios podrán crear nuevas partidas que ayudará a contar los puntos fácilmente y se guardaran los datos en la base de datos para poder ser consultados en la sección “ver partidas” donde se podrá consultar las partidas jugadas anteriormente y ver el detalle de estas.

En la actividad perfiles, el usuario creará los perfiles de los jugadores que serán los que jueguen las partidas, la imagen del perfil puede ser modificada por una de la galería o realizar una foto en el momento o si no fuera el caso la aplicación selecciona una al azar de las que dispone al app, también se podrá consultar en el detalle del perfil la máxima puntuación alcanzada por el jugador, en número de victorias o el número de partidas jugadas, entre otros.

En la actividad ranking, el usuario puede consultar el ranking de los perfiles creados y puede consultar y ordenar por máxima puntuación, partidas ganadas o porcentaje de victorias.

## ABSTRACT

Deluxe Apps is a company comprised of two partners: Jorge María Huguet and Miguel Ángel Segovia Freeman. The company was born out of the increasing popularity of board games and their "deluxification," a term adopted by the board game community to indicate an improvement made to a board game, such as painting accompanying miniatures, 3D printing components, creating inserts to keep components in place, or, in the case of this company, developing applications to be used alongside the game.

Deluxe Apps is dedicated to developing applications for board games with the aim of enhancing gameplay and capitalizing on the strengths of each game. The company's objective is to improve the players' experience by developing applications that promote healthy competition, cooperation, and dynamic gameplay.

The potential clients for Deluxe Apps are game developers or distributors who want to "deluxify" their games. For the Deluxe Apps project, they have developed the Bunny Kingdom Deluxe application, where they have been hired to enhance the game and foster competitiveness. The application features three main activities: Games, Profiles, and Rankings.

In the Games activity, users can create new games that help keep track of points easily, and the data is stored in the database for later reference in the "view games" section, where users can review previously played matches and see their details.

In the Profiles activity, users create player profiles that will participate in the matches. The profile picture can be chosen from the gallery, taken in the moment, or, if not available, the application selects one randomly from the available options. Users can also view additional details in the profile, such as the player's highest score, number of victories, or total number of matches played, among others.

In the Rankings activity, users can check the ranking of the created profiles and sort them by highest score, number of wins, or win percentage.

# Índice de contenido

1.	INTRODUCCIÓN.....	8
2	CONTEXTO FUNCIONAL Y TECNOLÓGICO .....	9
2.1	CONTEXTO FUNCIONAL .....	9
2.2	CONTEXTO TECNOLÓGICO .....	9
3	NECESIDADES DEL SECTOR PRODUCTIVO .....	10
3.1	ANÁLISIS DE LA SITUACIÓN ACTUAL.....	10
3.2	NECESIDADES DEL CLIENTE Y OPORTUNIDAD DE NEGOCIO .....	10
3.3	EL NUEVO PROYECTO: BUNNY KINGDOM DELUXE .....	11
3.3.1	TIPO DE PROYECTO .....	11
3.3.2	CARACTERÍSTICAS REQUERIDAS AL PROYECTO.....	11
3.3.3	OBLIGACIONES FISCALES, LABORALES Y DE PREVENCIÓN DE RIESGO.....	12
3.3.4	AYUDAS / SUBVENCIONES .....	13
4	DISEÑO DEL PROYECTO.....	14
4.1	FASES DEL PROYECTO.....	14
4.1.1	ANÁLISIS.....	15
4.1.2	DISEÑO.....	16
4.1.3	IMPLEMENTACIÓN .....	27
4.1.4	PRUEBAS .....	27
4.2	OBJETIVOS A CONSEGUIR .....	29
4.3	PREVISIÓN DE LOS RECURSOS MATERIALES Y HUMANOS NECESARIOS.....	30
4.4	PRESUPUESTO ECONÓMICO. ....	30
5	PLANIFICACIÓN DE LA EJECUCIÓN DEL PROYECTO.....	31
5.1	FASE DE ANÁLISIS.....	31
5.2	FASE DE DISEÑO.....	38
5.3	FASE DE IMPLEMENTACIÓN .....	44
5.4	FASE DE PRUEBAS.....	61
5.4.1	CASO DE PRUEBA 1: COMPROBACIÓN REGISTRO DE USUARIO.....	61
5.4.2	CASO DE PRUEBA 2: COMPROBACIÓN DE NAVEGACIÓN ENTRE ACTIVIDADES. ....	61
5.4.3	CASO DE PRUEBA 3: CREAR NUEVO PERFIL Y MODIFICAR DATOS.....	62
5.4.4	CASO DE PRUEBA 4: COLORES DE LOS JUGADORES.....	62
5.4.5	CASO DE PRUEBA 5: COMPROBAR RECYCLERVIEW RESUMEN TURNO.....	63
5.4.6	CASO DE PRUEBA 6: GUARDAR LA PARTIDA EN BBDD.....	63
5.4.7	CASO DE PRUEBA 7: RANKINGS ORDENADOS.....	64
6	DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN.....	64
7	FUENTES.....	65
8	ANEXOS .....	65

## Índice de figuras

Ilustración 1 Estandartes.....	17
Ilustración 2 AlertDialog Confirmación.....	17
Ilustración 3 AlertDialog Informativo .....	17
Ilustración 4 Switch música.....	17
Ilustración 5 Splash Screen.....	18
Ilustración 6 Actividad Login.....	19
Ilustración 7 Actividad Registrarse.....	19
Ilustración 8 Actividad Partidas .....	20
Ilustración 9 Actividad SeleccionJugador .....	20
Ilustración 10 Actividad SeleccionJugador con marcos seleccionados.....	21
Ilustración 11 Actividad ResumenTurno .....	21
Ilustración 12 Actividad ResumenTurno con feudos cargados .....	22
Ilustración 13 Actividad NuevoFeudo .....	22
Ilustración 14 Actividad VerPartidas.....	23
Ilustración 15 Actividad DetallePartidas.....	23
Ilustración 16 Actividad Perfiles .....	24
Ilustración 17 Actividad DetallePerfil (Nuevo Perfil).....	24
Ilustración 18 Actividad DetallePerfil .....	25
Ilustración 19 Actividad Ranking .....	25
Ilustración 20 Estructura Json usada en la aplicación.....	26
Ilustración 21 Boceto Actividad Login.....	38
Ilustración 22 Boceto Actividad Registrarse .....	39
Ilustración 23 Boceto Actividad Partidas .....	39
Ilustración 24 Boceto Actividad VerPartidas.....	40
Ilustración 25 Boceto Actividad Perfiles.....	40
Ilustración 26 Boceto Actividad Ranking .....	41
Ilustración 27 Boceto Actividad NuevoFeudo .....	41
Ilustración 28 Estructura Json .....	42
Ilustración 29 gradle .....	44
Ilustración 30 Firebase.....	45
Ilustración 31 Arquitectura Android Studio.....	46
Ilustración 32 Estructura ViewModel .....	48
Ilustración 33 Estructura repositorio(1).....	50
Ilustración 34 Estructura repositorio(2).....	51
Ilustración 35 Código Actividad Partida(1).....	53
Ilustración 36 Código Actividad Partida(2).....	54
Ilustración 37 Código Adapter Perfiles (1).....	56
Ilustración 38 Código Adapter Perfiles (2).....	57
Ilustración 39 POJO Perfil (1) .....	58
Ilustración 40 POJO Perfil (2) .....	59

## 1. INTRODUCCIÓN

Este documento responde a la realización del **módulo de Proyecto** del CFGS en **Desarrollo de aplicaciones multiplataforma**. El módulo de Proyecto complementa la formación establecida para el resto de los módulos profesionales que integran el título en las funciones de análisis del contexto, diseño del proyecto y organización de la ejecución.

Propósito del proyecto: El propósito de este proyecto es desarrollar una aplicación móvil que complemente o mejore la experiencia del jugador sobre el juego de mesa Bunny Kingdom de Richard Garfield. La aplicación deberá proporcionar funciones adicionales, como gestión de puntuación, contador de turnos, para enriquecer y mejorar la experiencia general del juego.

Una de las características destacadas de nuestra aplicación es la posibilidad de crear perfiles de jugador. Esto significa que podrás guardar los datos de cada jugador, como sus nombres, avatares y estadísticas de juego. Esto hace que sea aún más fácil y personalizado comenzar una partida. Además, podrás actualizar y editar los perfiles en cualquier momento, manteniendo todo organizado y accesible.

La base de datos de la aplicación será gestionada por Firebase, guardará los datos con un email asociado para que el usuario al acceder a su aplicación, solo vea sus propias partidas y perfiles.



## 2 CONTEXTO FUNCIONAL Y TECNOLÓGICO

A continuación, se describen los contextos funcionales y tecnológicos en los que se enmarca el proyecto.

### 2.1 CONTEXTO FUNCIONAL

El proyecto se enmarca en el desarrollo de una aplicación móvil que complementa el juego de mesa Bunny Kingdom, brindando funcionalidades adicionales para mejorar la experiencia de juego. A continuación, se detallan las principales características funcionales de la aplicación:

1. **Registro de usuarios:** Los nuevos usuarios podrán crear una cuenta en la aplicación proporcionando información como nombre de usuario, dirección de correo electrónico y contraseña.
2. **Inicio de sesión:** Los usuarios registrados podrán acceder a sus perfiles y a las funcionalidades de la aplicación a través de un proceso de inicio de sesión. Esto requerirá que ingrese su dirección de correo electrónico y su contraseña.
3. **Recuperación de contraseña:** Se proporcionará una opción para restablecer la contraseña en caso de que los usuarios la olviden. Esto implica el envío de un enlace de restablecimiento de contraseña al correo electrónico asociado con la cuenta del usuario.
4. **Creación y visualización de partidas:** La aplicación brindará al usuario la capacidad de iniciar una partida en el juego de mesa. Se elegirán los jugadores que jugarán además de sus colores correspondientes. Una vez listo la partida comenzará y se procederá a la gestión de turnos. Una vez finalizada la partida se podrá visualizar las partidas jugadas con información de los jugadores y la puntuación obtenida por estos, además de la opción de sacar una foto al tablero de juego como prueba para la posteridad.
5. **Creación de perfiles:** La aplicación permitirá a los jugadores crear perfiles personalizados, donde podrán ingresar su nombre y foto de perfil. Estos perfiles almacenarán el progreso del jugador, como puntuaciones, partidas jugadas y partidas ganadas entre otros.
6. **Gestión de puntos:** La aplicación permitirá a los jugadores registrar y gestionar los puntos obtenidos durante el juego. Se podrán introducir los puntos obtenidos por cada jugador en cada ronda o turno, y la aplicación calculará y mantendrá actualizada la puntuación total de cada jugador.
7. **Gestión de turnos:** La aplicación facilitará la gestión de turnos durante el juego de mesa. Se podrá llevar un seguimiento de qué jugador le toca jugar en cada turno.
8. **Rankings:** La aplicación generará rankings o clasificaciones basadas en las puntuaciones de los jugadores. Se podrán visualizar los rankings globales o por categorías específicas, lo que permitirá a los jugadores comparar su desempeño con el de otros jugadores y establecer competencias amistosas. Además, se podrán establecer filtros y criterios de clasificación.

### 2.2 Contexto Tecnológico

El proyecto se ha desarrollado utilizando las siguientes tecnologías:

1. **Android Studio:** Se ha utilizado Android Studio como el entorno de desarrollo integrado (IDE) principal para la creación de la aplicación móvil. Android Studio proporciona herramientas específicas para el desarrollo de aplicaciones Android y es compatible con el lenguaje de programación Java.
2. **Java:** El lenguaje de programación Java ha sido utilizado para el desarrollo de la lógica de la aplicación en Android Studio. Java es un lenguaje ampliamente utilizado en el desarrollo de aplicaciones Android y ofrece una amplia gama de bibliotecas y API que facilitan la creación de funcionalidades.

3. Gradle: Gradle ha sido utilizado como sistema de construcción de la aplicación Android. Gradle simplifica la gestión de dependencias y la compilación del código fuente, lo que facilita el proceso de desarrollo y permite integrar bibliotecas externas necesarias para la aplicación.
4. Firebase Authentication: Firebase Authentication proporciona el sistema de autenticación de usuarios para el registro y el inicio de sesión en la aplicación. Con Firebase Authentication, los usuarios podrán crear cuentas y acceder a la aplicación de manera segura mediante credenciales de correo electrónico y contraseña, o mediante proveedores de inicio de sesión social como Google o Facebook.
5. Firebase Firestore: Firebase Firestore ha sido utilizado como la base de datos en la nube para almacenar y recuperar datos en tiempo real. Firestore es una base de datos NoSQL flexible y escalable que permite almacenar y sincronizar datos estructurados, incluidas las imágenes. Se utilizará para almacenar información de perfiles de usuario, puntuaciones, configuraciones de partidas, entre otros datos relevantes.
6. Almacenamiento de Firebase: El almacenamiento de Firebase se utilizará para almacenar las imágenes de perfil de los usuarios y las fotos de las partidas. Proporciona un sistema de almacenamiento en la nube seguro y escalable para cargar y descargar archivos desde la aplicación.

Estas tecnologías seleccionadas proporcionan un conjunto sólido de herramientas para el desarrollo de aplicaciones Android, permitiendo la implementación de funciones de autenticación segura de usuarios y una base de datos en la nube eficiente para almacenar información y archivos multimedia como las fotos de la partida, o las imágenes de los perfiles.

### 3 NECESIDADES DEL SECTOR PRODUCTIVO

A continuación, se identifican las necesidades detectadas en el sector productivo que originan la oportunidad de negocio que se detalla en los siguientes puntos.

#### 3.1 *Análisis de la situación actual*

Cada año salen al mercado multitud de juegos de mesa, algunas empresas adaptan juegos de mesa a aplicaciones móviles, pocos juegos tienen aplicaciones complementarias o profesionales, las existentes son desarrolladas por personas independientes.

#### 3.2 *Necesidades del cliente y oportunidad de negocio*

La necesidad de los posibles clientes son ampliar sus ventas y promocionarse también a través de app store, para ello nuestra empresa desarrolla aplicaciones para mejorar la experiencia del usuario, ser visibles en la app store, y aumentar el alcance de las ventas.

La empresa no necesita una oficina, al ser una startup cada uno desarrolla desde su casa con su propio pc.

El ámbito de actuación es a nivel mundial al ser publicadas las aplicaciones en la app store, La aplicación será desarrollada en castellano y Miguel Ángel al ser bilingüe traduce los textos al inglés. Hay alrededor de 1.5-1.8 mil millones de personas que hablan inglés en el mundo y 548 millones de personas castellanoparlantes.

### 3.3 El nuevo proyecto: *Bunny Kingdom Deluxe*

#### 3.3.1 Tipo de proyecto

El proyecto se enmarca en la creación de una empresa que dará respuesta a las necesidades detectadas en el mercado de aplicaciones móviles. La empresa ya existe y busca abrir una nueva línea de negocio/producto en el ámbito del desarrollo de software. Esta expansión permitirá diversificar y ampliar el alcance de la empresa, brindando soluciones innovadoras a través de aplicaciones móviles.

Tipo de proyecto a desarrollar en la empresa: Desarrollo de software.

El enfoque principal de la empresa será el desarrollo de aplicaciones móviles para diversas plataformas, como Android y iOS. Se emplearán tecnologías y lenguajes de programación adecuados, como Java, Kotlin o Swift, para crear aplicaciones de alta calidad y funcionalidades personalizadas que satisfagan las necesidades de los clientes.

El proyecto incluirá todas las fases del desarrollo de software, desde el análisis de requisitos y diseño hasta la implementación, prueba y entrega final de la aplicación. Se pondrá énfasis en la usabilidad, la experiencia del usuario y la optimización del rendimiento para asegurar que las aplicaciones desarrolladas cumplan con los estándares de calidad y sean altamente competitivas en el mercado de app store.

La empresa aprovechará la naturaleza flexible de una startup, donde cada miembro del equipo puede trabajar desde su hogar, sin necesidad de una oficina física. Esto permite reducir los costos de operación y promover un ambiente de trabajo ágil y eficiente.

Con este nuevo proyecto, la empresa buscará posicionarse como un referente en el desarrollo de aplicaciones móviles, ofreciendo soluciones innovadoras y de calidad que satisfagan las necesidades de sus clientes y contribuyan a su crecimiento y éxito en el mercado.

#### 3.3.2 Características requeridas al proyecto

Elementos diferenciadores:

- **Interfaz intuitiva y atractiva:** La aplicación contará con un diseño atractivo relacionado al juego de mesa y una interfaz de usuario intuitiva que mejore la experiencia del usuario y facilite la navegación por las diferentes funcionalidades.
- **Integración con Firebase:** El uso de Firebase proporcionará una experiencia de registro y autenticación de usuarios segura, así como un almacenamiento eficiente de imágenes y una base de datos en tiempo real para la gestión de puntuaciones, rankings y partidas.
- **Funcionalidades completas:** La aplicación ofrecerá todas las características necesarias para complementar y mejorar el juego de mesa, proporcionando una experiencia de juego enriquecida y competitiva.

**Entorno específico:** El proyecto se desarrollará utilizando Android Studio como el entorno de desarrollo integrado (IDE) principal. Se utilizará el lenguaje de programación Java para el desarrollo de la lógica de la aplicación y se aprovechará la plataforma Firebase para la implementación de las funcionalidades de registro, autenticación y base de datos en tiempo real.

Justificación de las herramientas a utilizar:

- **Android Studio:** Es el IDE estándar para el desarrollo de aplicaciones Android, proporcionando herramientas y recursos específicos para la plataforma.
- **Java:** Es uno de los lenguajes de programación más utilizados en el desarrollo de aplicaciones Android, con una amplia comunidad de desarrolladores y bibliotecas disponibles.
- **Firebase:** Proporciona una solución integral y fácil de usar para el registro de usuarios, autenticación segura, almacenamiento de imágenes y base de datos en tiempo real, lo cual cumple con los requisitos funcionales del proyecto.
- **Photoshop:** Es el software más popular de edición y creación de imágenes, los usuarios pueden realizar una amplia variedad de tareas relacionadas con la edición de imágenes. Estas incluyen ajustar el brillo, contraste y tonalidades de una imagen, recortar y redimensionar, eliminar imperfecciones y objetos no deseados, y aplicar efectos especiales y filtros.

Primera aproximación a los recursos tanto humanos como materiales necesarios: Recursos humanos:

- **Equipo de desarrollo de aplicaciones móviles:** Incluye desarrolladores con experiencia en Android, Java y Firebase, así como diseñadores de interfaces de usuario y expertos en experiencia de usuario.
- **Equipo de ilustradores:** Una diseñadora gráfica que realice los diseños necesarios para implementarlos en la interfaz.
- **Equipo de traducción:** Un traductor bilingüe encargado de traducir los textos de la aplicación al inglés.

Recursos materiales:

- Ordenadores o portátiles para el desarrollo de la aplicación.
- Dispositivos móviles para realizar pruebas de usabilidad.
- Acceso a Internet para la descarga de herramientas, librerías y recursos necesarios durante el desarrollo.
- El juego de mesa "Bunny Kingdom".

### 3.3.3 Obligaciones fiscales, laborales y de prevención de riesgo

Principales obligaciones fiscales:

- **Alta en Hacienda:** La empresa deberá darse de alta en el Registro de Empresas y obtener un número de identificación fiscal (NIF) para cumplir con sus obligaciones tributarias.
- **Declaración de impuestos:** La empresa deberá presentar las declaraciones de impuestos correspondientes, como el Impuesto sobre Sociedades (IS) y el Impuesto sobre el Valor Añadido (IVA), de acuerdo con la normativa fiscal vigente en el país.

**Principales obligaciones laborales:**

- **Contratación de personal:** Si la empresa requiere contratar empleados, deberá cumplir con la normativa laboral y realizar los contratos adecuados, asegurándose de respetar los derechos laborales y las condiciones de trabajo.
- **Seguridad Social:** La empresa deberá afiliar a sus empleados al régimen de la Seguridad Social y realizar los correspondientes pagos de cotizaciones sociales.
- **Cumplimiento de horarios y descansos:** La empresa deberá respetar las regulaciones laborales en cuanto a horarios de trabajo, descansos, vacaciones y días festivos.

**Principales obligaciones de prevención de riesgos:**

- **Evaluación de riesgos:** La empresa deberá realizar una evaluación de los posibles riesgos laborales que puedan surgir en el desarrollo de su actividad y tomar las medidas necesarias para prevenirlos o minimizarlos.
- **Formación en prevención de riesgos laborales:** La empresa deberá proporcionar a sus empleados la formación necesaria en materia de prevención de riesgos laborales, asegurando que estén informados y capacitados para trabajar de manera segura.
- **Mantenimiento de condiciones de seguridad:** La empresa deberá mantener las instalaciones y equipos en condiciones seguras, realizar inspecciones periódicas y llevar a cabo las acciones correctivas necesarias para garantizar la seguridad y salud en el trabajo.

### 3.3.4 Ayudas / subvenciones

El Impuesto sobre Sociedades se trata de un tributo que grava los beneficios obtenidos por las sociedades mercantiles durante el desarrollo de su actividad. En Sociedades, tendremos que determinar qué ingresos y qué gastos son fiscalmente imputables y deducibles, respectivamente, siguiendo la normativa vigente en cada momento, y atendiendo a los criterios establecidos por la dirección general de tributos y otros organismos fiscales de esta índole. El tipo general es del 25% sobre los beneficios anuales obtenidos por la empresa. Para las sociedades recién constituidas y que supongan el inicio de una actividad económica, se aplicará el 15% de tipo impositivo reducido durante los dos primeros años (como novedad en los Presupuestos Generales del Estado de 2022 junto a la subida de las cuotas de Autónomo se ha aprobado la rebaja del 15 al 10% del tipo mínimo de este tributo para empresas de nueva creación).

El Impuesto sobre Actividades Económicas es un tipo de gravamen a las actividades económicas realizadas tanto por personas físicas como jurídicas. Es decir, se aplica tanto a autónomos como a Pymes y su cuantía varía en función de la actividad económica. Si se factura menos de 1 millón de euros o la empresa fue constituida hace menos de dos años, no hay deber de pagarlo.

En cualquier caso, aunque la empresa esté exenta de su pago, sí que debe darse de alta en el IAE a través del modelo 036 o 037 de la Agencia Tributaria. Se pueden revisar estos pagos a través de deducciones, como contratar a personas menores de 30 años, así como las deducciones por gasto en innovación tecnológica, investigación y desarrollo. Aunque esto suelen ser opciones más al alcance de grandes empresas que de las Pymes.

También existe la posibilidad de pagar menos impuestos con la condición de pagarlos más adelante a través por ejemplo de los créditos fiscales. En estos casos se alegan motivos o circuns-

tancias especiales como compensar pérdidas de años anteriores, hacer reestructuraciones o saneamiento de cuentas. Esto permite no pagar el impuesto de sociedades hasta que se tengan beneficios suficientes que compensen las pérdidas.

Uno de los principales gastos que se puede deducir son los seguros de Responsabilidad Civil y Comercio Multirriesgo.

## 4 DISEÑO DEL PROYECTO

Dando por hecho la viabilidad del proyecto, en este apartado se concretarán las fases necesarias para llevarlo a cabo, y cumplir con los objetivos que se establezcan, teniendo en cuenta los recursos necesarios.

### 4.1 Fases del proyecto

#### 1. Investigación y análisis:

1. Investigación exhaustiva sobre el juego y sus mecánicas.
2. Análisis de las necesidades y expectativas de los jugadores en términos de complementos digitales para el juego.
3. Identificación de las funcionalidades clave que se pueden implementar en la aplicación para mejorar la experiencia de juego.

#### 2. Diseño de la interfaz de usuario:

1. Creación de un diseño de interfaz de usuario de la aplicación, teniendo en cuenta la usabilidad y la estética.
2. Definición de pantallas y los flujos de navegación de la aplicación.
3. Integración de elementos visuales relacionados con el juego de mesa para mantener la coherencia y la familiaridad.

#### 3. Desarrollo de la aplicación:

1. Implementación de la estructura de la aplicación utilizando un lenguaje de programación Java usando Android Studio.
2. Creación de funcionalidades principales de la aplicación, como el login, la gestión de turnos, contoneo de puntos o la creación de perfiles con imágenes.
3. Conexión con el backend de Firebase el cual se usará para la gestión de inicio de sesión, base de datos y almacenamiento de imágenes.

#### 4. Pruebas y depuración:

1. Realización de pruebas exhaustivas de la aplicación para garantizar su funcionalidad, rendimiento y compatibilidad.
2. Identificación y solución de los posibles errores o problemas de la aplicación.
3. Realización de pruebas de usabilidad con usuarios reales para obtener retroalimentación sobre la experiencia de juego y realizar mejoras.

#### 5. Lanzamiento y mantenimiento:

1. Preparación de la aplicación para su lanzamiento en las tiendas de aplicaciones correspondientes (App Store, Google Play, etc.).
2. Realización de una estrategia de marketing para promocionar la aplicación entre los jugadores y entusiastas del juego de mesa.
3. Recopilación de comentarios de los usuarios y realiza actualizaciones periódicas para mejorar la aplicación y añadir nuevas características

### 4.1.1 Análisis

Durante el análisis del proyecto se creó un documento docs compartido por Google Drive para repartir las tareas e ir añadiendo nuevas tareas según fueran necesarias, este documento está dividido por actividades y cada actividad tiene sus tareas asignadas.

#### General

- Botón Mute en todas las actividades.
- Música continua poner en activity LifeCycle.
- Cambiar el texto del tema nocturno a negro.
- Bloquear el giro de pantalla.
- Traducciones.
- Comprobar Layouts.
- Botón atrás, que no se cierre la app.
- Estandartes → decir donde te encuentras.
- Estandartes → no navegar al mismo sitio en el que estás.
- Cambiar el icono de la app.
- Cambiar la velocidad de la animación de los estandartes.

#### Login

- Validación Usuario/Contraseña (No campos vacíos, notificación error login)
- Has olvidado la contraseña
- Validación de usuario logueado.

#### Registro usuario

- Validación no campos vacíos.
- Creación de usuarios.

#### Partidas

- Botón “Cerrar Sesión” y cerrar sesión de usuario.

#### Nueva Partida

- Crear actividad para seleccionar perfiles que van a jugar.
- Al seleccionar el perfil cambiar el color del marco.
- Añadir TextView de puntuación de ronda.
- Bucle 4 rondas.
- Llamada a la bbdd (POST).
- Tras acabar el turno 4 hacer una ventana para incluir puntuación de pergaminos.
- Al finalizar la partida hacer foto.
- Guardar el dato de puntuación de cada jugador para ser usado en cada ronda.
- ClickListener para poder editar feudo en caso de error.
- Cuando acaba la partida dirigir a DetallePartida con el resultado final.
- Confirmación AlertDialog terminar turno.
- Cargar perfiles en selección de jugadores.
- Validación para salir de la partida al pulsar en estandartes.
- AlertDialog información puntos al terminar turno.
- AlertDialog informar nueva ronda.
- Redirigir desde la Firebase a DetallePartida cuando se acaba la partida.

#### Añadir Feudo

- Validación torres > 0.
- Lógica para que al marcar un recurso aparezca un borde “seleccionado”.
- Calcular puntos.

**Ver partidas**

- Llamada a la bbdd.
- ClickListener vista detalle.
- Hacer layout responsive.

**Detalle partidas**

- Llamada a la bbdd.
- Eliminar de la bbdd.
- Confirmación eliminación.
- Quitar botón Eliminar → por requisitos de cliente se ha decidido que no es necesario.
- El texto de los colores está en castellano, porque así se guarda en la bbdd, hay que buscar alternativa para que sea independiente del idioma.

**Perfiles**

- Botón “Nuevo Perfil”.
- Llamada a la bbdd.
- Resolución de fotos.
- Resolución de marcos.

**Detalle Perfil**

- Modificar foto perfil.
- EditText nombre.
- Boolean “Modificar/Añadir”.
- Llamada a la bbdd.
- Añadir botón eliminar perfil.
- Actualizar partidas ganadas.
- Actualizar la máxima puntuación.
- Actualizar partidas jugadas.

**Ranking**

- Llamada bbdd.
- Añadir botón ordenar puntuación.
- Añadir botón ordenar partidas ganadas.
- Ordenar datos por puntuación.
- Ordenar datos por victorias.
- Añadir botón ordenar por % victorias.
- Ordenar datos por % victorias.

**4.1.2 Diseño**

La fuente de los textos es Enchanted Land se escogió esta tipografía porque es similar a la caligrafía usada en los textos medievales.

La aplicación tiene de fondo imágenes que relacionan la actividad en la que se encuentra con lo que se quiere representar. En las actividades Login y Registro la imagen de fondo es una puerta de madera cerrada ya que aún no ha accedido a la aplicación, al pulsar el botón “Entrar” si los datos de login son correctos suena una puerta de madera abriéndose. En la actividad Partidas se muestra un camino con un poste con varias señales indicando diferentes direcciones, ya que hay varias decisiones que tomar y cada una lleva a una actividad diferente. Al pulsar en “Nueva Partida” o “Ver Partidas” la imagen de fondo es una biblioteca ya que es ahí donde se consultan los registros y se guarda la información. La actividad “Perfiles” y “DetallePerfil” tienen de imagen de fondo un muro de piedra donde se cuelgan los cuadros que son los diferentes perfiles creados por el usuario. Por último, la actividad Ranking tiene de fondo el salón del trono.



La aplicación tiene cuatro actividades principales: Login, Partidas, Perfiles, Ranking. Para navegar entre las diferentes actividades en la parte superior tiene 3 estandartes que funcionaran como pestañas en un explorador de internet, para identificar donde se encuentra el usuario el estandarte es de color naranja, los otros 2 permanecen rojos.



*Ilustración 1 Estandartes*

Los AlertDialog tienen de fondo un pergamino para mostrar el mensaje o solicitar la confirmación y los botones son sellos de cera, cuando el mensaje es por un error al introducir datos o porque está realizando una acción que no se puede además de informar con el mensaje, el teléfono móvil también vibrará.



*Ilustración 2 AlertDialog Confirmación*



*Ilustración 3 AlertDialog Informativo*

En la parte inferior izquierda hay un botón switch que al seleccionarlo para o reanuda la música de la aplicación, este botón está incluido en todas las actividades en la misma posición.



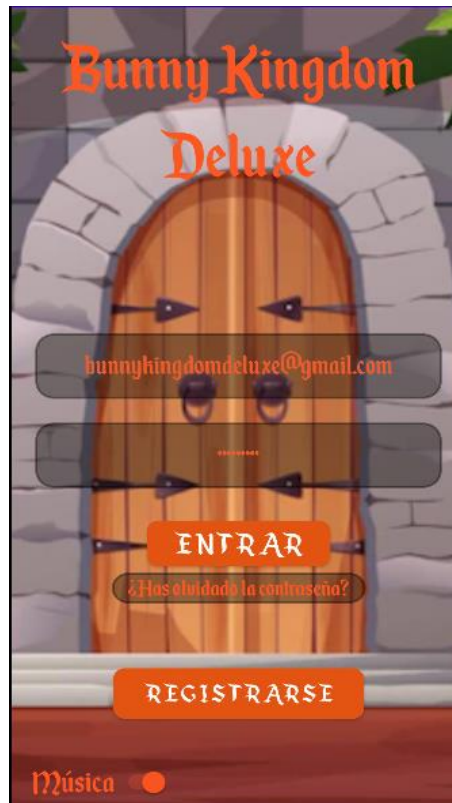
*Ilustración 4 Switch música*

Al abrir la aplicación aparece una Splash Screen como pantalla de carga para indicar que se está abriendo la aplicación, el color escogido de fondo es el naranja (#FF5722), que hemos escogido como uno de los colores temáticos de la aplicación al ser el color de las zanahorias.

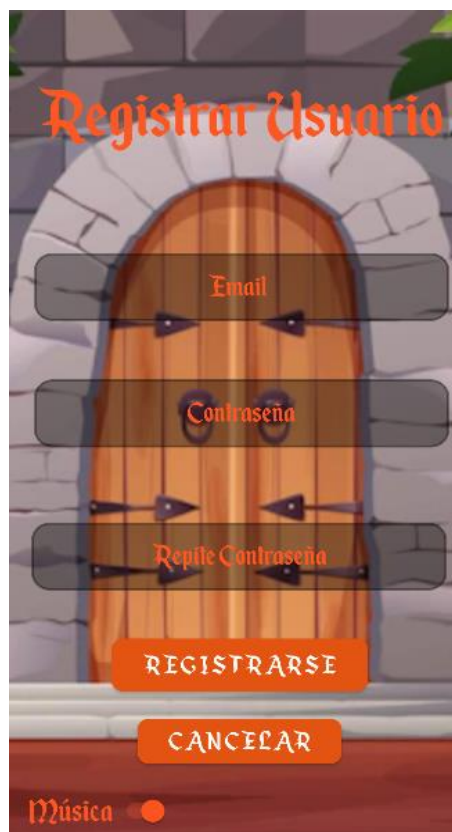


*Ilustración 5 Splash Screen*

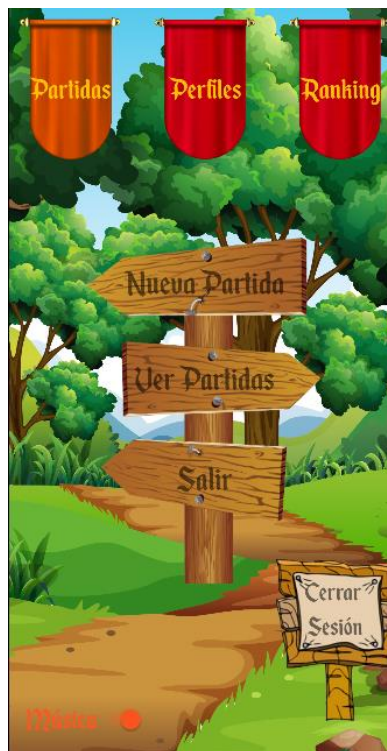
La actividad Login tiene 2 campos, uno para introducir el email y el otro para la contraseña, un botón para acceder, bajo el botón tiene el texto por si el usuario ha olvidado la contraseña y necesita restablecerla, también incluye un botón para en caso de no estar registrado ir al formulario de registro de la actividad “Registrarse”.

*Ilustración 6 Actividad Login*

La actividad registrarse contiene los campos de texto para poder hacer el registro en la base de datos y acceder a la aplicación.

*Ilustración 7 Actividad Registrarse*

La actividad Partidas incluye diferentes botones para navegar a diferentes actividades como “Nueva Partida”, “Ver Partidas”, “Salir”, “Cerrar Sesión”. Si pulsa en “Salir” la app se cierra, y si pulsa en “Cerrar Sesión” el usuario es redirigido a la actividad Login y se cierra la sesión en la app.



*Ilustración 8 Actividad Partidas*

Al pulsar en “Nueva Partida” el usuario navegará a “Selección Jugadores” donde al entrar aparecerá un AlertDialog para dar instrucciones de cómo usar esa activador, si pulsa sobre los diferentes jugadores el marco que rodea la foto de perfil cambia de color para escoger el color que va a usar cada jugador en el tablero.



*Ilustración 9 Actividad SeleccionJugador*



*Ilustración 10 Actividad SeleccionJugador con marcos seleccionados*

Tras elegir a los jugadores y pulsar “Empezar”, el usuario verá un resumen del turno que se está jugando donde se muestra el número de ronda, el nombre del jugador activo y los puntos que ha conseguido, también verá los feudos que ha introducido que ha conseguido, para ello tiene que pulsar en el botón “Añadir Feudo” y se enviará al formulario de “NuevoFeudo” tras seleccionar los recursos y torres necesarios al pulsar en “Añadir” el feudo se cargará en resumen turno, si ha terminado el turno al pulsar el botón “Finalizar turno”, el usuario verá un AlertDialog pidiendo confirmación. Cuando la partida acabe se mostrará la actividad “Detalle Partida” con la información de la partida que se acaba de jugar y dará la opción de hacer una foto para que se guarde en la bbdd.



*Ilustración 11 Actividad ResumenTurno*





Ilustración 12 Actividad ResumenTurno con feudos cargados



Ilustración 13 Actividad NuevoFeudo

Al pulsar en “Ver Partidas” desde la actividad “Partidas”, el usuario verá un registro de todas las partidas que ha jugado y cada partida tendrá la información de la fecha, nombre de los jugadores y puntos conseguidos, al pulsar sobre cualquiera de ellas navegará a la actividad “DetallePartida” para ver más información sobre la partida seleccionada, como el color de los jugadores, la posición en el ranking de puntos de esa partida y la foto si se hizo alguna al terminar la partida.

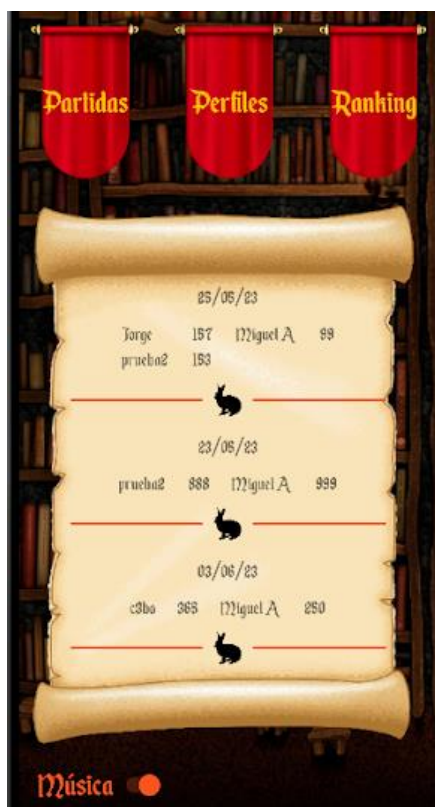
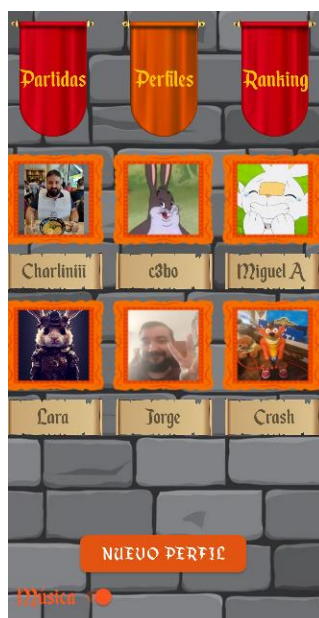


Ilustración 14 Actividad VerPartidas



Ilustración 15 Actividad DetallePartidas

En la actividad “Perfiles” se mostrará el nombre y la imagen de perfil de los perfiles ya creados y un botón “Nuevo Perfil” para poder crear uno nuevo, si se pulsa el botón se mostrará la actividad “DetallePerfil” con los valores de los datos por defecto y si no se añade una foto al perfil la aplicación escoge una aleatoriamente. Si se pulsa sobre un perfil ya creado navegará a la actividad “DetallePerfil” donde se puede modificar el nombre y la imagen de perfil por una de la galería o una hecha en el momento desde la cámara, y se mostrará los datos de partidas ganadas, partidas jugadas y máxima puntuación.



*Ilustración 16 Actividad Perfiles*

Si se pulsa el botón “Nuevo Perfil” se verá la actividad “Detalle Perfil” con el perfil con valores por defecto y si no se añade una foto al perfil la aplicación escoge una aleatoriamente, si se pulsa sobre un perfil redireccionará a la actividad “Detalle Perfil” donde se puede modificar la imagen de perfil por una de la galería o una hecha en el momento desde la cámara, también se puede modificar el nombre, y se mostrará los datos de partidas ganadas, partidas jugadas y máxima puntuación.



*Ilustración 17 Actividad DetallePerfil (Nuevo Perfil)*





Ilustración 18 Actividad DetallePerfil

Para terminar la pantalla ranking muestra un título del ranking que se está viendo, por defecto están ordenados por máxima puntuación y también tiene 3 botones para que sea ordenado de las 3 formas disponibles máxima puntuación, victorias, porcentaje de victorias.

Los tres primeros jugadores en vez de mostrar un número se sustituyen por una corona de oro, plata o cobre sucesivamente.

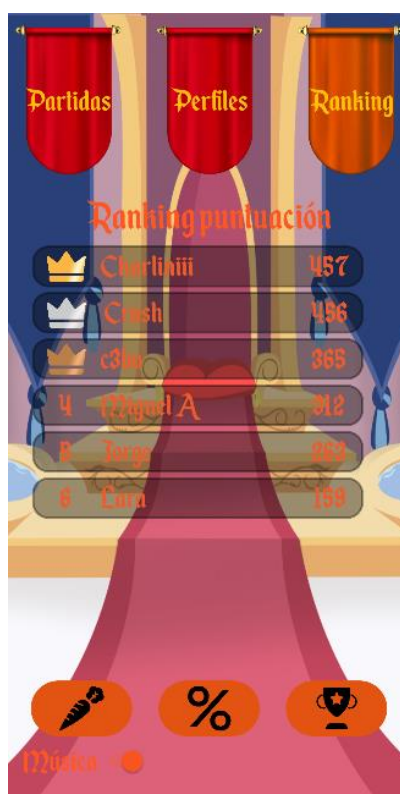
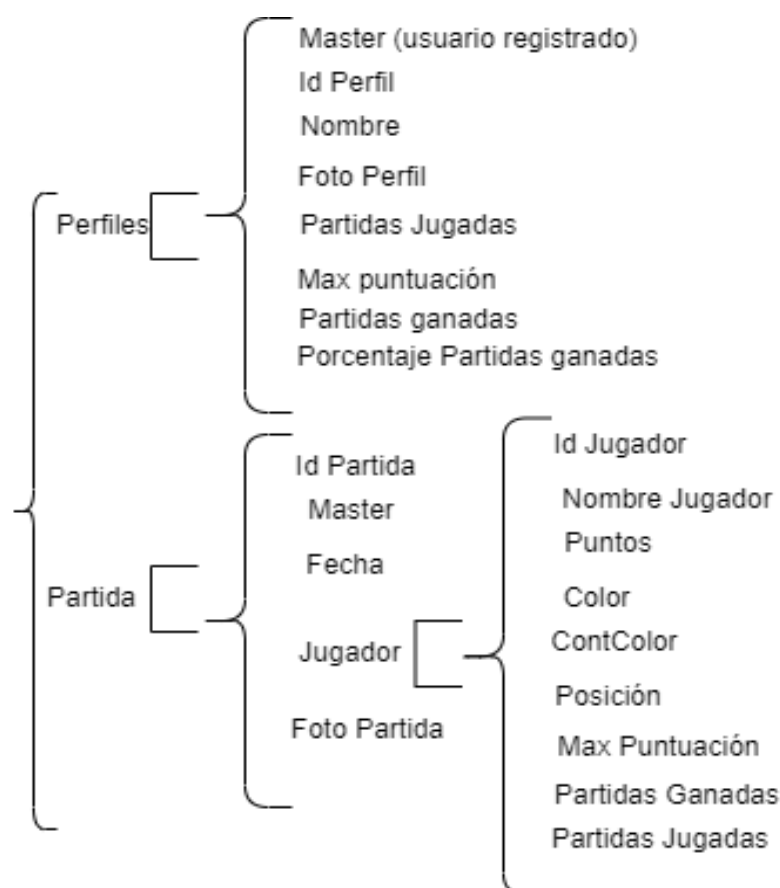


Ilustración 19 Actividad Ranking

Firebase usa archivos Json para almacenar los datos, la estructura usada para este proyecto es la que se muestra en la imagen inferior.

Perfiles es un array que contiene Master, que es el email del usuario registrado y que está usando la aplicación, Id Perfil generado automáticamente al crear el perfil, Nombre introducido por el usuario, Foto Perfil es la imagen que se carga desde la librería, se toma con la cámara de fotos o la aplicación elige una al azar si no has decidido anteriormente, Partidas Jugadas, Max puntuación, Partidas ganadas y Porcentaje Partidas ganadas son datos que se modifican en función de las partidas jugadas.

Partidas es un array que contiene los atributos Id Partida generado automáticamente al crear la partida, Master, que es el email del usuario registrado y que está usando la aplicación, fecha en la que se ha jugado la partida, Foto Partida es la imagen se toma con la cámara de fotos al finalizar la partida, y Jugador es un array que contiene Id Jugador que es el Id Perfil del jugador seleccionado, Nombre Jugador es el nombre del perfil, Puntos son puntos conseguidos en la partida, Color es el color escogido, ContColor es un número usado para calcular el color escogido, Posición es el número del puesto en el que ha quedado al finalizar la partida, Max Puntuación es el número usado para comparar si ha superado su récord y modificarlo en Perfil, Partidas Ganadas recoge el dato de Perfiles y lo actualiza si la Posición es 1, y Partidas Jugadas es un contador de partidas que se actualiza al finalizar una partida.



*Ilustración 20 Estructura Json usada en la aplicación*

### 4.1.3 Implementación

Para el desarrollo del proyecto hemos utilizado Android Studio que es el entorno de desarrollo utilizado para crear la aplicación móvil. Se basa en el lenguaje de programación Java, que se utiliza para desarrollar la lógica de la aplicación. Gradle se utiliza como sistema de construcción para administrar dependencias y compilar el código fuente.

Firebase Authentication se utilizará como sistema de autenticación de usuarios, permitiendo a los usuarios registrarse e iniciar sesión de forma segura mediante credenciales de correo electrónico y contraseña.

Firebase Firestore es una base de datos en la nube utilizada para almacenar y recuperar datos en tiempo real. Proporciona una estructura flexible y escalable para almacenar información como perfiles de usuario, puntuaciones y configuraciones de partidas. También se utiliza el almacenamiento de Firebase para almacenar imágenes de perfil de usuario y fotos de partidas. Proporciona un sistema seguro y escalable para cargar y descargar archivos desde la aplicación.

La diseñadora gráfica usará Photoshop para editar las imágenes y eliminar el fondo de las imágenes e iconos que necesiten de una capa Alpha para que se añada transparencia y la imagen no tape lo que hay detrás.

### 4.1.4 Pruebas

Las pruebas que hemos planteado realizar para comprobar el correcto funcionamiento de la aplicación son las siguientes:

#### 4.1.4.1 Caso de prueba 1: Comprobación registro de usuario.

- Descripción:  
Comprobar que tras rellenar el formulario de registro se ha guardado en la bbdd y puede hacer login el usuario.
- Condiciones de ejecución:  
Crear un usuario desde cero.
- Entrada:  
No se necesita ningún dato previo para realizar la prueba.
- Resultado esperado:  
BBDD actualizada con el usuario nuevo y puede hacer login en la aplicación.

#### 4.1.4.2 Caso de prueba 2: Comprobación de navegación entre actividades.

- Descripción  
Desplazarse entre actividades comprobando que la opción seleccionada te envía a la actividad deseada.
- Condiciones de ejecución  
No es necesario
- Entrada  
No se necesita ningún dato previo para realizar la prueba.

- Resultado esperado

El usuario puede desplazarse a cualquier actividad sin que la aplicación provoque algún error.

#### **4.1.4.3 Caso de prueba 3: Crear nuevo perfil y modificar datos.**

- Descripción:

Crear un perfil nuevo introduciendo el nombre y las diferentes opciones de cambiar la imagen de perfil, comprobar que se almacena en la bbdd y se muestra en el recycler-view de la aplicación.

- Condiciones de ejecución

Estar ya logueado en la aplicación.

- Entrada

Tener un perfil ya creado para hacer login.

- Resultado esperado

Se puede crear un nuevo perfil escogiendo la foto desde la galería o haciéndola desde la cámara de fotos del móvil o si no se escoge ninguna opción la aplicación elige una foto aleatoria de las que tiene almacenadas.

#### **4.1.4.4 Caso de prueba 4: Colores de los jugadores**

- Descripción

Comprobar que al pulsar sobre un jugador el marco cambia de color y al comenzar la partida comprobar que la aplicación guarda el color correcto de cada jugador en la bbdd.

- Condiciones de ejecución

Estar ya logueado en la aplicación e iniciar partida.

- Entrada

Tener varios perfiles ya creados para poder seleccionar jugadores, tener los marcos de diferentes colores ya guardados en la app.

- Resultado esperado

Se puede crear un nuevo perfil escogiendo la foto desde la galería o haciéndola desde la cámara de fotos del móvil o si no se escoge ninguna opción la aplicación elige una foto aleatoria de las que tiene almacenadas.

#### **4.1.4.5 Caso de prueba 5: Comprobar recyclerView ResumenTurno.**

- Descripción

Comprobar que al seleccionar las torres y recursos de NuevoFeudo se cargan los datos correctamente en el recyclerView de ResumenTurno y se puede hacer clic sobre el feudo y modificarlo

- Condiciones de ejecución

Jugar un turno de la partida.

- Entrada  
Tener jugadores ya cargados.
- Resultado esperado  
El feudo se carga correctamente, se añade la puntuación, si se modifica el feudo la puntuación se modifica al volver a cargar con los datos correctos.

#### **4.1.4.6 Caso de prueba 6: Guardar la partida en bbdd.**

- Descripción  
Comprobar que al finalizar la partida se guarda con los datos correctos en la bbdd y se muestra en el recyclerView de VerPartidas.
- Condiciones de ejecución  
Jugar una partida.
- Entrada  
Tener jugadores ya cargados.
- Resultado esperado  
La bbdd se actualiza con la partida cargada correctamente y se muestra en el recyclerView VerPartidas.

#### **4.1.4.7 Caso de prueba 7: Rankings ordenados.**

- Descripción  
Comprobar que los ranking cargan la información correcta al pulsar el botón correspondiente y son ordenados por puntuación, victorias o porcentaje de victorias
- Condiciones de ejecución  
Ninguna
- Entrada  
Tener varios perfiles con datos diferentes.
- Resultado esperado  
Los rankings se ordenan correctamente al pulsar sobre el botón correspondiente.

## **4.2 Objetivos a conseguir**

Objetivos del proyecto:

- Cumplir con las fechas establecidas para las diferentes tareas.
- Realizar todas las tareas propuestas.
- Buena comunicación.
- Saber trabajar en equipo.

Objetivos de la aplicación:

- Lograr descargas 5000 descargas internacionales.
- Desarrollar 8 aplicaciones en 1 año.

Objetivo de la empresa:

I.E.S. Juan de la Cierva	Curso: 2022 / 2023	Página 29 de 65
--------------------------	--------------------	-----------------

- Establecerse en el mercado.
- Ser una empresa de referencia en el desarrollo de aplicaciones para juegos de mesa.

### **4.3 Previsión de los recursos materiales y humanos necesarios**

Para la elaboración del proyecto sería necesario 2 desarrolladores para realizar el código de la aplicación, crear la bbdd, funcionalidades y métodos de la aplicación.

Los diseños de las imágenes de fondo de la aplicación y algunos recursos necesarios para la interfaz serán creados por una diseñadora gráfica.

También es necesario disponer del juego de mesa para entender su funcionamiento y sobre ello establecer los requisitos que necesita cumplir la aplicación.

Se necesitará un pc por persona para poder trabajar en simultáneo.

El cálculo de horas estimado es de 150 horas para los desarrolladores y 50 horas para la diseñadora gráfica.

La base de datos se almacena en Firebase Firestore.

### **4.4 Presupuesto económico.**

El sueldo medio de un programador junior en España es de 15€/hora con las 150 horas necesarias para el desarrollo del proyecto sería un total de 2250€.

El salario medio de un diseñador gráfico en España es de 12€/hora siendo necesarias 50 horas para el diseño de las imágenes necesarias el precio sería de 60€.

Cada desarrollador y la diseñadora gráfica usará su pc personal para el desarrollo del proyecto.

Android Studio es un IDE de software libre.

El juego de mesa será cedido por la distribuidora no es necesario gasto.

En la medida de lo posible se usarán iconos libres de derechos de librerías gratuitas o con licencia Creative Commons.

También se utilizará música con licencia Creative Commons.

Firebase Firestore es gratuito siempre que no se necesite más recursos de los que ofrece el plan gratuito que es 50.000 usuarios activos al mes, 600.000 operaciones de escritura, 1.500.000 operaciones de lectura, 600.000 operaciones de borrado, 1000 llamada a la AP al mes, 5GB de fotos.

## 5 PLANIFICACIÓN DE LA EJECUCIÓN DEL PROYECTO

A continuación, se detallan las actividades/tareas/procedimientos por cada una de las fases del proyecto previamente establecidas.

### 5.1 Fase de Análisis

#### 1. Estudio de las necesidades a cubrir

Durante el proceso de análisis, se identificaron diversas necesidades que la aplicación debe cubrir. Estas necesidades se centran en mejorar la experiencia de juego de los usuarios y brindarles funcionalidades adicionales para enriquecer su participación. A continuación, se detallan algunas de las principales necesidades identificadas:

- **Gestión de puntuación:** Los jugadores requieren una forma fácil y conveniente de registrar y mantener un seguimiento de sus puntuaciones durante las partidas. La aplicación móvil debe permitirles ingresar los puntos obtenidos en cada ronda o turno, realizar cálculos automáticos y mantener actualizadas las puntuaciones individuales y totales de cada jugador.
- **Control de turnos:** Es esencial tener un sistema claro y preciso para determinar el orden de los turnos de los jugadores. La aplicación móvil debe facilitar la gestión de turnos, mostrar qué jugador le toca jugar en cada turno y proporcionar una forma intuitiva de avanzar al siguiente turno.
- **Registro de partidas:** Los jugadores desean tener un registro histórico de las partidas jugadas, incluyendo información como los jugadores involucrados, las puntuaciones obtenidas y cualquier otra estadística relevante. La aplicación móvil debe permitirles crear y visualizar partidas anteriores, lo que les permitirá revivir los momentos destacados y comparar su rendimiento en diferentes partidas.
- **Interfaz intuitiva y atractiva:** Los usuarios buscan una interfaz de usuario amigable y visualmente atractiva que les permita navegar por la aplicación sin dificultades. La aplicación móvil debe ser intuitiva y fácil de usar, con controles claros y bien organizados, para brindar una experiencia de usuario fluida y agradable.

#### 2. Estudio de la situación actual

A continuación, se presentan algunos puntos relevantes identificados durante este análisis:

- **Registro y seguimiento manual de puntuaciones:** En la situación actual, los jugadores deben llevar un registro manual de las puntuaciones obtenidas durante el juego de mesa Bunny Kingdom. Esto puede ser propenso a errores y requerir tiempo adicional para calcular y mantener actualizadas las puntuaciones de cada jugador.
- **Gestión de turnos manual:** La determinación y seguimiento de los turnos de los jugadores se realiza de manera manual, lo que puede llevar a confusiones y retrasos durante el juego. Es importante contar con un sistema más eficiente y automatizado que simplifique esta tarea y evite posibles errores.
- **Falta de registro histórico:** Actualmente, no existe una forma sistemática de registrar y visualizar partidas anteriores. Los jugadores pueden perder la oportunidad de revivir momentos destacados, analizar su progreso a lo largo del tiempo y comparar su rendimiento con el de otros jugadores.
- **Limitaciones de la comunicación entre jugadores:** Durante el juego de mesa, la comunicación entre los jugadores se realiza principalmente de forma verbal y depende de su habilidad para recordar y transmitir información. Esto puede llevar a malentendidos o confusiones, especialmente en situaciones complejas del juego.

- Falta de herramientas adicionales: Los jugadores pueden desear contar con herramientas adicionales, como asistentes de puntuación, recordatorios de turnos, opciones de configuración personalizada, entre otros, para mejorar su experiencia y disfrute del juego.

### 3. Establecimiento de los requisitos del proyecto

#### Requisitos Funcionales:

##### Registro de usuarios:

- Los nuevos usuarios podrán crear una cuenta proporcionando un nombre de usuario, dirección de correo electrónico y contraseña.
- Se verificará la validez de la información ingresada durante el registro.

##### Inicio de sesión:

- Los usuarios registrados podrán acceder a sus perfiles y a las funcionalidades de la aplicación mediante un proceso de inicio de sesión.
- Se solicitará la dirección de correo electrónico y contraseña para autenticar al usuario.

##### Recuperación de contraseña:

- Se proporcionará una opción para restablecer la contraseña en caso de que los usuarios la olviden.
- Se enviará un enlace de restablecimiento de contraseña al correo electrónico asociado con la cuenta del usuario.

##### Creación y visualización de partidas:

- Los usuarios podrán iniciar una partida en el juego, seleccionando los jugadores y sus respectivos colores.
- Se almacenará información de las partidas jugadas, incluyendo los jugadores, puntuaciones y una foto del tablero de juego.
- Los usuarios podrán visualizar las partidas anteriores, revisar información de los jugadores y las puntuaciones obtenidas.

##### Creación de perfiles:

- Los jugadores podrán crear perfiles personalizados ingresando su nombre y foto de perfil.
- Los perfiles almacenarán el progreso del jugador, incluyendo puntuaciones, partidas jugadas y partidas ganadas.

##### Gestión de puntos:

- Los jugadores podrán registrar y gestionar los puntos obtenidos durante el juego.
- Se permitirá introducir los puntos obtenidos por cada jugador en cada ronda o turno.
- La aplicación calculará y mantendrá actualizada la puntuación total de cada jugador.
-



**Gestión de turnos:**

- La aplicación facilitará la gestión de turnos durante el juego de mesa.
- Se mostrará claramente qué jugador le toca jugar en cada turno.

**Rankings:**

- La aplicación generará rankings o clasificaciones basadas en las puntuaciones de los jugadores.
- Los jugadores podrán visualizar rankings globales o por categorías específicas.
- Se permitirá establecer filtros y criterios de clasificación para personalizar la visualización de los rankings.

**Requisitos No Funcionales:**

- Interfaz intuitiva y atractiva.
- La aplicación contará con una interfaz de usuario amigable y visualmente atractiva.
- Los controles y elementos de la interfaz serán claros y bien organizados para una fácil navegación.

**Compatibilidad multiplataforma:**

- La aplicación será compatible con dispositivos móviles iOS y Android.
- Se asegurará de que la aplicación funcione correctamente en diferentes tamaños de pantalla y resoluciones.

**Rendimiento eficiente:**

- La aplicación responderá de manera rápida y eficiente a las interacciones del usuario.
- Se minimizarán los tiempos de carga y se optimizará el rendimiento en general.

**Seguridad de los datos:**

- Se implementarán medidas de seguridad para proteger la información de los usuarios, como encriptación de contraseñas y transmisiones seguras.

**Disponibilidad y escalabilidad.**

- La aplicación estará disponible para su acceso en todo momento, evitando caídas del servidor o interrupciones prolongadas del servicio.
- Se diseñará la aplicación de manera escalable, de modo que pueda manejar un crecimiento en el número de usuarios y partidas sin comprometer su rendimiento.

**Eficiencia en el uso de recursos:**

- La aplicación utilizará de manera eficiente los recursos del dispositivo móvil, como memoria, capacidad de almacenamiento y consumo de energía.

**Internacionalización y localización:**

- Se contemplará la posibilidad de internacionalización para adaptar la aplicación a diferentes idiomas y regiones.
- Se permitirá a los usuarios seleccionar su idioma preferido y ajustar configuraciones regionales, si es aplicable.

**Requisitos Técnicos:****Plataforma móvil:**

- La aplicación se desarrollará para dispositivos móviles utilizando tecnologías multiplataforma como React Native o Flutter, garantizando su compatibilidad con iOS y Android.

**Almacenamiento de datos:**

- Se utilizará una base de datos adecuada para almacenar la información de los usuarios, partidas y puntuaciones.
- Se evaluarán opciones como Firebase, MongoDB o MySQL para determinar la solución más adecuada.

**Integración con servicios externos:**

- La aplicación se integrará con servicios externos, como servicios de autenticación (por ejemplo, Firebase Auth) y servicios de almacenamiento en la nube (por ejemplo, Firebase Storage), según sea necesario.

**Pruebas y depuración:**

- Se implementarán pruebas exhaustivas de la aplicación para garantizar su correcto funcionamiento y detectar posibles errores y fallos.
- Se utilizarán herramientas de depuración adecuadas para identificar y corregir problemas durante el desarrollo y el despliegue.

**4. Valoración comparativa de las posibles soluciones****Evaluación de frameworks y tecnologías:**

- Investigar y evaluar diferentes frameworks y tecnologías de desarrollo móvil multiplataforma, como React Native, Flutter, Xamarin, etc.
- Analizar las ventajas y desventajas de cada opción en términos de rendimiento, compatibilidad, facilidad de desarrollo, comunidad de soporte, etc.
- Comparar las características y capacidades de cada framework en relación con los requisitos del proyecto, como la integración con servicios externos y el rendimiento en diferentes plataformas móviles.

**Análisis de opciones de almacenamiento de datos:**

- Investigar y evaluar diferentes opciones de almacenamiento de datos, como bases de datos SQL, NoSQL, servicios en la nube, etc.
- Comparar aspectos como la escalabilidad, la seguridad, el costo y la facilidad de integración con la tecnología seleccionada.

- Considerar la capacidad de almacenamiento, el rendimiento de consulta y la sincronización de datos en tiempo real, según los requisitos de la aplicación.

Evaluación de servicios de autenticación y almacenamiento en la nube:

- Investigar y comparar servicios de autenticación y almacenamiento en la nube, como Firebase Authentication, AWS Cognito y Firebase Storage.
- Analizar la facilidad de integración, la seguridad, la escalabilidad y los costos asociados con cada servicio.
- Evaluar la compatibilidad con la tecnología seleccionada y la capacidad de manejar los requisitos de autenticación y almacenamiento de la aplicación.

Evaluación de opciones de implementación y despliegue:

- Investigar y comparar diferentes opciones de implementación y despliegue de la aplicación móvil, como tiendas de aplicaciones, servicios de distribución beta, etc.
- Analizar los requisitos y restricciones de cada plataforma de distribución y seleccionar la opción más adecuada para llegar al público objetivo de la aplicación.

Análisis de costos:

- Evaluar los costos asociados con las diferentes soluciones consideradas, incluyendo el desarrollo, la infraestructura, los servicios externos y el mantenimiento a largo plazo.
- Comparar los costos en función de los requisitos y la viabilidad económica del proyecto.

#### 5. Identificación de las necesidades que implica el nuevo proyecto en la empresa.

La identificación de las necesidades que implica el nuevo proyecto en la empresa es un paso crucial para comprender el impacto y los beneficios que la aplicación móvil complementaria para el juego de mesa Bunny Kingdom brindará a la organización. A continuación, se presenta un ejemplo de cómo puedes identificar estas necesidades:

Mejorar la experiencia del cliente:

- La empresa busca ofrecer una experiencia de juego más completa y enriquecedora a los usuarios del juego de mesa Bunny Kingdom.
- La aplicación móvil complementaria proporcionará funcionalidades adicionales que permitirán a los jugadores gestionar sus partidas, puntuaciones y perfiles de manera más fácil y conveniente.
- La mejora en la experiencia del cliente puede resultar en una mayor satisfacción, fidelización y recomendación de la marca.

Competir en el mercado:

- La empresa desea mantenerse competitiva en el mercado de juegos de mesa, donde la digitalización y las aplicaciones móviles están ganando popularidad.

- La aplicación móvil complementaria ayudará a la empresa a destacarse al proporcionar un valor adicional a los jugadores y brindar una experiencia moderna e interactiva.
- Esto puede ayudar a la empresa a atraer nuevos clientes y retener a los existentes frente a la competencia.

#### Gestión eficiente de las partidas y puntuaciones:

- La empresa busca mejorar la gestión de las partidas y las puntuaciones de los jugadores del juego de mesa Bunny Kingdom.
- La aplicación permitirá a los jugadores registrar y gestionar sus puntuaciones de manera más automatizada y precisa.
- Esto simplificará el proceso de seguimiento de resultados y estadísticas, evitando la necesidad de llevar registros manuales y facilitando la organización de torneos o competiciones.

#### Recopilación de datos y análisis:

- La empresa busca recopilar datos relevantes sobre el rendimiento de los jugadores y las tendencias de juego en el juego de mesa Bunny Kingdom.
- La aplicación móvil complementaria permitirá recopilar información sobre las partidas jugadas, puntuaciones obtenidas y preferencias de juego.
- Estos datos podrán ser utilizados para realizar análisis y tomar decisiones estratégicas en función del comportamiento y las preferencias de los jugadores.

#### Fomentar la interacción y la comunidad de jugadores:

- La empresa busca fomentar la interacción y la comunidad entre los jugadores de Bunny Kingdom.
- La aplicación proporcionará funcionalidades como rankings, perfiles personalizados y la posibilidad de compartir fotos del tablero de juego.
- Estas características fomentarán la competencia amistosa, la comunicación entre los jugadores y la creación de una comunidad activa en torno al juego.

Para solventar los problemas que plantea el proyecto puede ser necesario contratar personal, formarlo en determinadas metodologías/herramientas, comprar equipos...

#### 6. Estudio de viabilidad de la solución elegida teniendo en cuenta no solo los beneficios económicos.

A continuación, se presentan algunos factores que pueden influir en la viabilidad de la solución:

##### Viabilidad técnica:

- Evaluar la capacidad técnica de implementar la solución elegida, teniendo en cuenta los recursos disponibles, las habilidades del equipo de desarrollo y la compatibilidad con las plataformas móviles objetivo (iOS, Android).

- Analizar la complejidad técnica de las funcionalidades requeridas y la disponibilidad de tecnologías y herramientas adecuadas para su implementación.

#### Viabilidad operativa:

- Evaluar si la solución se integra de manera efectiva con los procesos operativos existentes en la empresa.
- Determinar si la solución es escalable y puede manejar el crecimiento del número de usuarios, partidas y datos sin comprometer su rendimiento y funcionalidad.

#### Viabilidad legal y de seguridad:

- Asegurarse de que la solución cumpla con los requisitos legales y normativas vigentes en cuanto a protección de datos, privacidad y seguridad de la información.
- Evaluar los posibles riesgos de seguridad y establecer medidas adecuadas para mitigarlos, como el cifrado de datos y la autenticación segura.

#### Viabilidad de integración:

- Analizar la capacidad de la solución para integrarse con otros sistemas o servicios externos, como servicios de autenticación, almacenamiento en la nube u otras API relevantes.
- Evaluar la compatibilidad y la facilidad de integración con los sistemas existentes en la empresa, como sistemas de gestión interna o bases de datos.

#### Viabilidad de mantenimiento y soporte:

- Evaluar la capacidad de mantener y actualizar la aplicación móvil en el futuro, considerando aspectos como la disponibilidad de recursos, la evolución de las plataformas móviles y las posibles mejoras y actualizaciones requeridas.
- Evaluar la disponibilidad de soporte técnico y la capacidad de resolver problemas y brindar actualizaciones en caso de ser necesario.

## 7. Corrección de posibles errores

La corrección de posibles errores es un aspecto importante dentro del desarrollo de un proyecto, ya que busca identificar y solucionar cualquier problema o falla que pueda surgir durante el proceso. A continuación, se presentan algunos puntos clave relacionados con la corrección de errores:

1. **Identificación temprana de errores:** Es fundamental realizar pruebas exhaustivas durante todas las etapas del desarrollo para identificar errores lo antes posible. Esto puede incluir pruebas unitarias, pruebas de integración y pruebas de aceptación, entre otras.
2. **Registro y documentación de errores:** Es importante mantener un registro detallado de todos los errores encontrados, incluyendo su descripción, gravedad, pasos para reproducirlos y cualquier información adicional relevante. Esto facilitará la posterior corrección y seguimiento de los problemas.
3. **Priorización de errores:** Una vez identificados, los errores deben ser priorizados según su gravedad e impacto en la funcionalidad y experiencia del usuario. Esto permitirá abordar primero los errores críticos que afecten de manera significativa el correcto funcionamiento de la aplicación.
4. **Análisis de causas raíz:** Es importante investigar y comprender las causas subyacentes de los errores para poder corregirlos de manera efectiva. Esto implica identificar si los errores son producto de problemas de código, configuración, diseño o cualquier otro factor, y tomar las acciones necesarias para abordar esas causas.

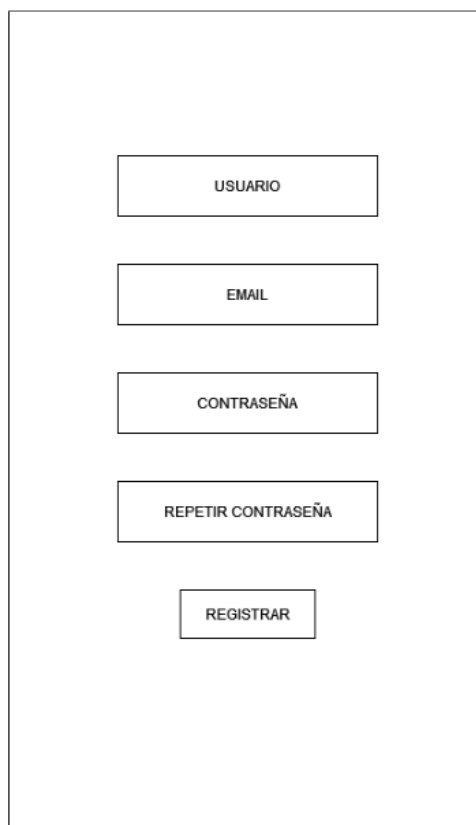
5. Desarrollo de soluciones: Una vez identificados los errores y sus causas, se deben desarrollar soluciones apropiadas. Esto puede implicar modificar el código, ajustar la configuración, rediseñar funcionalidades o realizar cualquier otra acción necesaria para solucionar los problemas encontrados.
6. Pruebas de validación: Después de realizar las correcciones, es importante volver a probar la aplicación para asegurarse de que los errores hayan sido solucionados y no hayan surgido nuevas complicaciones como resultado de las correcciones realizadas.
7. Mantenimiento continuo: A lo largo del proyecto y después de su finalización, es fundamental mantener un enfoque constante en la corrección de errores. Esto implica monitorear y abordar cualquier problema que surja durante el uso de la aplicación, así como implementar actualizaciones y mejoras periódicas para mantenerla en un estado óptimo.

## 5.2 Fase de diseño

1. Diseño de la arquitectura: Hemos decidido para nuestro proyecto utilizar el patrón de model-view-viewmodel que separa la lógica de negocio de la interfaz del usuario, de forma que podremos obtener los datos de la firestore de una forma eficiente.
2. Diseño de los interfaces: Utilizando draw.io hemos establecido los siguientes esquemas de las interfaces:

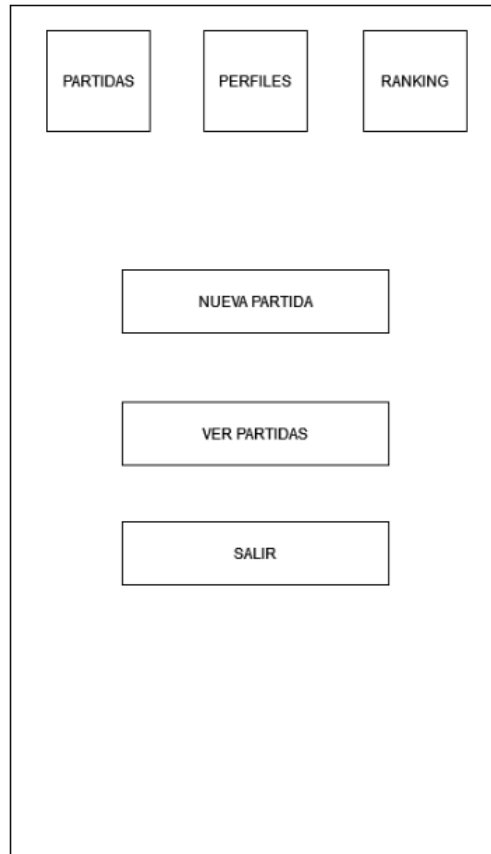
Diagrama de boceto de la actividad Login. El formulario está contenido dentro de un rectángulo vertical. En el interior, los elementos están distribuidos verticalmente: un campo de texto etiquetado 'TITULO', un campo de texto etiquetado 'USUARIO', un campo de texto etiquetado 'CONTRASEÑA', un botón etiquetado 'LOGIN', y un botón etiquetado 'REGISTRARSE'.

*Ilustración 21 Boceto Actividad Login*



A vertical registration form layout. It consists of five rectangular input fields stacked vertically, each containing a label: 'USUARIO', 'EMAIL', 'CONTRASEÑA', 'REPETIR CONTRASEÑA', and 'REGISTRAR'.

*Ilustración 22 Boceto Actividad Registrarse*



A menu layout for 'Partidas'. At the top, there are three square buttons labeled 'PARTIDAS', 'PERFILES', and 'RANKING'. Below them, there are three rectangular buttons stacked vertically, labeled 'NUEVA PARTIDA', 'VER PARTIDAS', and 'SALIR'.

*Ilustración 23 Boceto Actividad Partidas*



Ilustración 24 Boceto Actividad VerPartidas

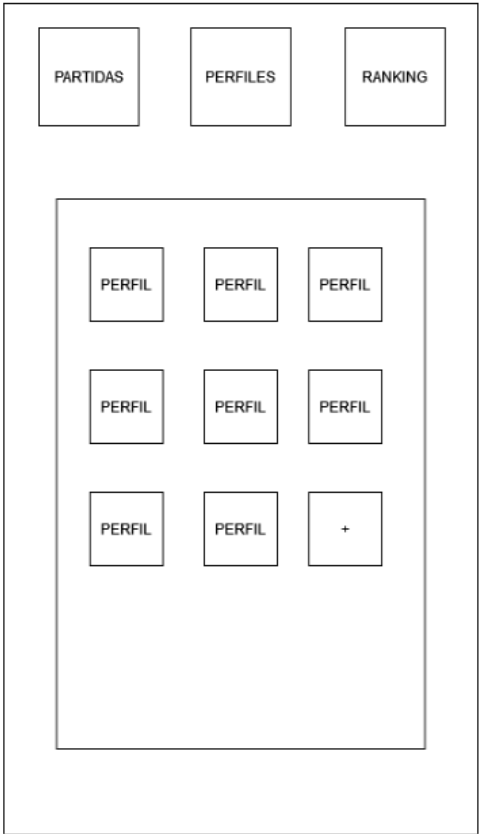


Ilustración 25 Boceto Actividad Perfiles



PARTIDAS

PERFILES

RANKING

1	JUGADOR	10000
2	JUGADOR	50000
3	JUGADOR	100
4	JUGADOR	0
5	JUGADOR	0
6	JUGADOR	0

Ilustración 26 Boceto Actividad Ranking

PARTIDAS

PERFILES

RANKING

-

0

+

RECURSOS

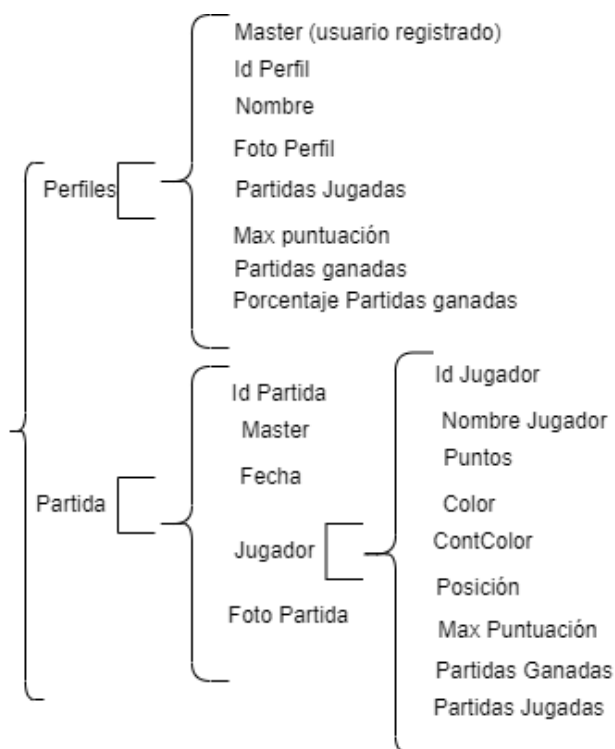
CANCELAR

AÑADIR

Ilustración 27 Boceto Actividad NuevoFeudo

## 3. Diseño de los datos:

Los datos de la aplicación se almacenarán de la siguiente manera:



*Ilustración 28 Estructura Json*

## 4. Diseño de los procedimientos de los procedimientos:

Los nombres de los métodos y sus argumentos pueden variar en función de las necesidades detectadas durante el desarrollo de la aplicación.

1. Iniciar sesión: Procedimiento: `loginUser(email: string, password: string)` Descripción: Este procedimiento permite a los usuarios registrados iniciar sesión en la aplicación móvil. Pasos:
  - Verificar si el correo electrónico y la contraseña proporcionados son válidos.
  - Autenticar al usuario y permitirle acceder a las funcionalidades de la aplicación.
  - Redirigir al usuario a la pantalla principal de la aplicación.
2. Crear partida: Procedimiento: `createGame(players: Player[], colors: Color[])` Descripción: Este procedimiento permite a los usuarios crear una nueva partida del juego Bunny Kingdom. Pasos:
  - Verificar si todos los jugadores seleccionados cumplen los requisitos necesarios para participar en una partida.
  - Asignar colores a cada jugador.
  - Iniciar la partida y establecer el orden de los turnos.
3. Registrar puntuación: Procedimiento: `recordScore(player: Player, round: number, score: number)` Descripción: Este procedimiento permite a los jugadores registrar su puntuación obtenida en cada ronda del juego. Pasos:

- Verificar si el jugador y la ronda proporcionados son válidos.
  - Registrar la puntuación obtenida por el jugador en la ronda especificada.
  - Actualizar la puntuación total del jugador.
4. Visualizar partidas jugadas: Procedimiento: `getPlayedGames(user: User)` Descripción: Este procedimiento permite a los usuarios visualizar las partidas jugadas anteriormente con información de los jugadores y la puntuación obtenida. Pasos:
- Obtener las partidas jugadas por el usuario especificado.
  - Recopilar la información de los jugadores y la puntuación obtenida en cada partida.
  - Mostrar la información al usuario en una lista o tabla.
5. Recuperar contraseña: Procedimiento: `resetPassword(email: string)` Descripción: Este procedimiento permite a los usuarios solicitar el restablecimiento de su contraseña en caso de que la hayan olvidado. Pasos:
- Verificar si el correo electrónico proporcionado está asociado a una cuenta registrada.
  - Generar un token de restablecimiento de contraseña y enviarlo al correo electrónico del usuario.
  - Permitir al usuario ingresar una nueva contraseña utilizando el token de restablecimiento.
  - Actualizar la contraseña del usuario en la base de datos.
6. Ver rankings globales: Procedimiento: `getGlobalRankings()` Descripción: Este procedimiento permite a los usuarios visualizar los rankings globales de puntuaciones en el juego Bunny Kingdom. Pasos:
- Obtener las puntuaciones de todos los jugadores registrados en la base de datos.
  - Ordenar las puntuaciones de mayor a menor.
  - Mostrar los rankings globales al usuario, con la posición de cada jugador y su puntuación correspondiente.
7. Filtrar rankings por categoría: Procedimiento: `filterRankingsByCategory(category: string)` Descripción: Este procedimiento permite a los usuarios filtrar los rankings de puntuaciones por una categoría específica, como "Jugador del mes" o "Mayor puntaje en una partida". Pasos:
- Obtener las puntuaciones de todos los jugadores registrados en la base de datos.
  - Filtrar las puntuaciones según la categoría especificada.
  - Ordenar las puntuaciones filtradas de mayor a menor.
  - Mostrar los rankings filtrados al usuario, con la posición de cada jugador y su puntuación correspondiente.

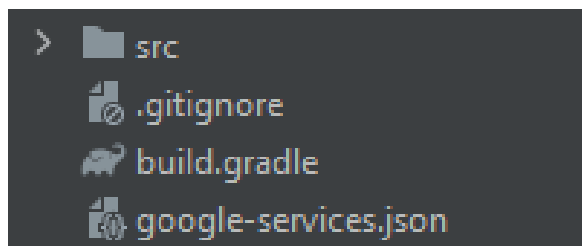
8. Realizar controles de turno: Procedimiento: performTurnControl(game: Game, currentPlayer: Player) Descripción: Este procedimiento realiza los controles necesarios para determinar si es el turno válido de un jugador y si puede realizar acciones dentro de su turno. Pasos:

- Verificar si el jugador actual es el próximo en el orden de turnos establecido en la partida.
- Comprobar si el jugador tiene suficientes recursos o requisitos para realizar las acciones permitidas en su turno.
- Actualizar el estado de la partida y las acciones disponibles para el jugador.

### 5.3 Fase de Implementación

#### 1. Preparación del entorno de implementación

- Se ha instalado Android Studio, junto con el emulador de Pixel 5, para el desarrollo de la aplicación móvil.
- Se ha asegurado de tener la configuración más reciente del IDE para aprovechar las últimas funcionalidades y mejoras.
- Se han agregado las siguientes dependencias al archivo Gradle del proyecto:

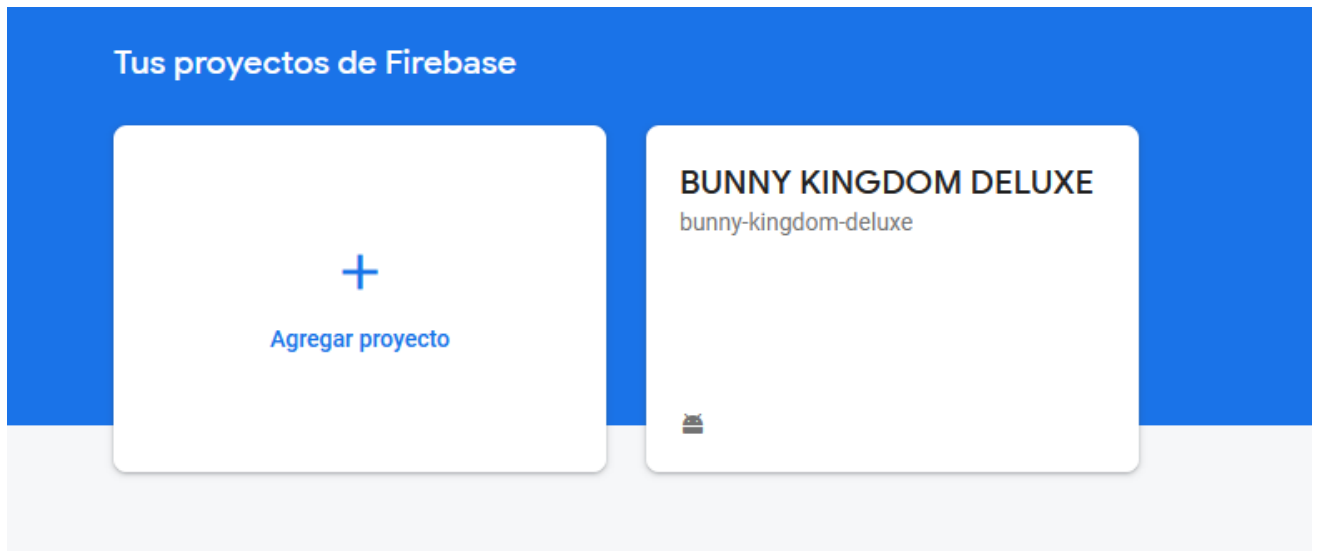


*Ilustración 29 gradle*

```
implementation platform('com.google.firebase:firebase-bom:31.5.0')
implementation 'com.google.firebase:firebase-firestore-ktx'
implementation 'com.google.firebase:firebase-analytics-ktx'
implementation 'androidx.appcompat:appcompat:1.5.1'
implementation 'com.google.android.material:material:1.7.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
implementation 'com.google.firebase:firebase-auth-ktx'
implementation 'com.google.firebase:firebase-storage-ktx'
implementation 'com.github.bumptech.glide:glide:4.15.1'
implementation 'com.facebook.shimmer:shimmer:0.5.0'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.4'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.0'
```

Estas dependencias incluyen componentes y bibliotecas necesarias para el proyecto, como Firebase Firestore para el almacenamiento de datos, Firebase Analytics para el seguimiento y análisis, Glide para la carga de imágenes, entre otras.

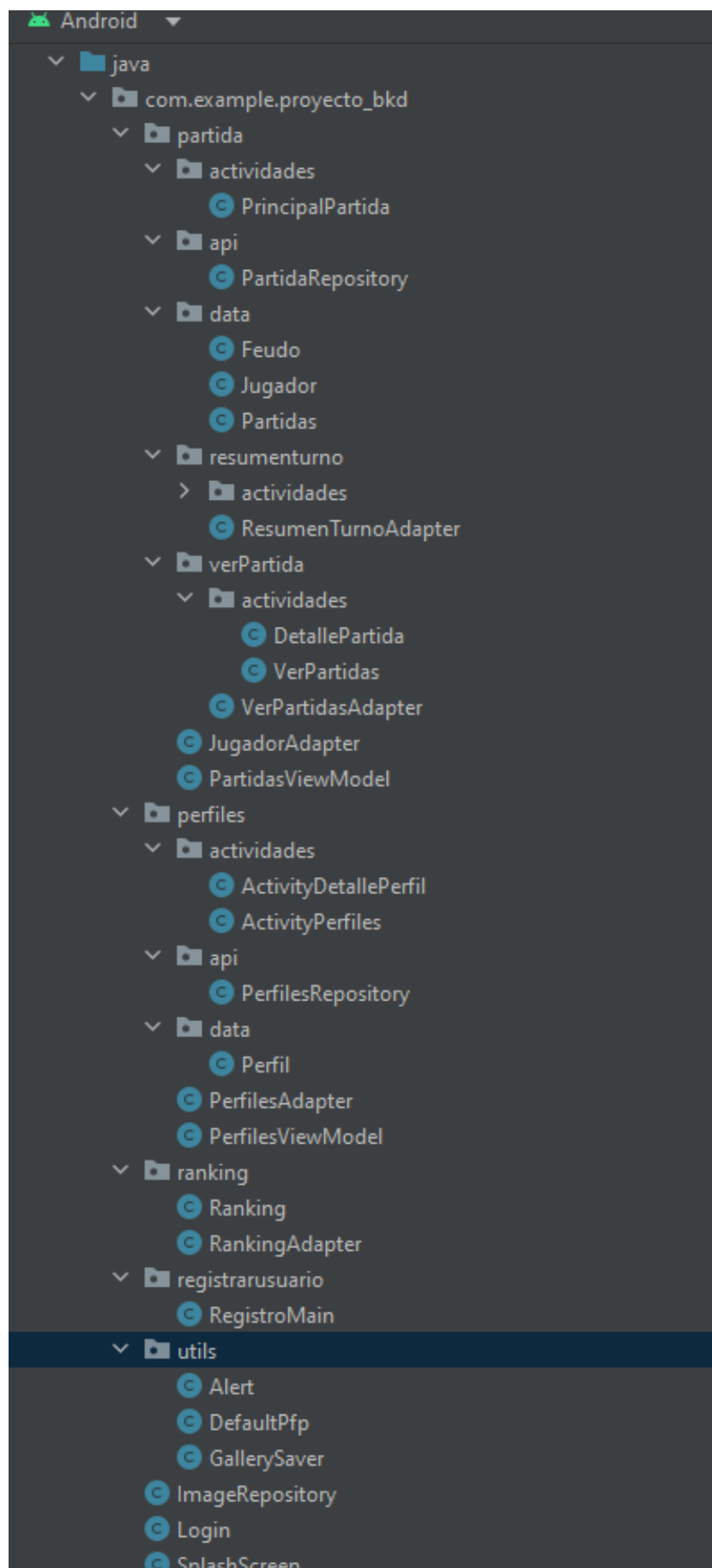
- Se ha creado un proyecto nuevo en Firebase para habilitar las funcionalidades requeridas, como Firestore y el almacenamiento.
- Se han realizado las configuraciones necesarias en Firebase para vincular la aplicación con los servicios proporcionados por Firebase.



*Ilustración 30 Firebase*

## 2. Desarrollo de la arquitectura

Como hemos comentado antes la arquitectura elegida es la de MVVM por lo tanto la estructura será la siguiente:

*Ilustración 31 Arquitectura Android Studio*

Durante la implementación del proyecto, se ha seguido la práctica de asignar a cada actividad de la aplicación su propio controlador, lo cual ayuda a mantener una estructura organizada y modular. Además, para aquellas actividades que requieren acceder a los datos de Firebase, se ha creado una clase denominada Repository para manejar dicha interacción. A continuación, se detalla cómo se ha implementado esta estructura en el proyecto:

1. Controladores de actividad:

- Para cada actividad en la aplicación, se ha creado un controlador correspondiente. Estos controladores se encargan de manejar la lógica específica de cada actividad, como el manejo de eventos, la interacción con los componentes de la interfaz de usuario y la navegación entre actividades.
- Los controladores de actividad ayudan a mantener una separación clara de responsabilidades y permiten una fácil mantenibilidad y escalabilidad del código.

2. Clase Repository:

- Se ha creado una clase denominada Repository para manejar la comunicación con Firebase y acceder a los datos necesarios.
- La clase Repository encapsula la lógica de acceso a Firebase, como leer y escribir datos en Firestore, autenticar usuarios, realizar operaciones de almacenamiento, entre otras.
- Al tener una clase Repository dedicada, se promueve el principio de responsabilidad única y se simplifica la gestión de datos de Firebase en la aplicación.

Con esta estructura, cada actividad tiene su propio controlador para manejar su lógica específica, mientras que la clase Repository se encarga de gestionar las interacciones con Firebase. Esto permite una organización clara y modular del código, facilitando su mantenimiento y facilitando el desarrollo de nuevas funcionalidades.

3. Desarrollo de los interfaces, los datos y los procedimientos

ViewModels

Tendrán esta estructura:

```

1 package com.example.proyecto_bkd.partida;
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15
16 public class PartidasViewModel extends ViewModel {
17     2 usages
18     private MutableLiveData<List<Partidas>> partidasData;
19     2 usages
20     private MutableLiveData<Partidas> partidasLiveData;
21     9 usages
22     private PartidaRepository partidaRepository;
23     2 usages
24     private ImageRepository imageRepository;
25     1 usage Miguel A. Freeman
26     public interface InsercionCallback{
27         1 usage Miguel A. Freeman
28         void callback();
29     }
30
31     JohnHawkinsGit + 1
32     public void init() {
33         partidaRepository = PartidaRepository.getInstance();
34         partidasData = partidaRepository.getListaPartidasLivedata();
35         partidasLiveData = partidaRepository.getPartidasLivedata();
36         imageRepository = ImageRepository.getInstance();
37     }
38
39     1 usage JohnHawkinsGit
40     public void getPartidas(String email) {partidaRepository.getPartidasPorDueño(email);}
41
42     1 usage Miguel A. Freeman
43     public void insertarPartida(Partidas partidas, InsercionCallback insercionCallback){...}
44
45     2 usages Miguel A. Freeman
46     public void SubirFotoPartida(Partidas partida, Uri uri){...}
47
48     2 usages JohnHawkinsGit
49     public void getPartida(String idPartida) { partidaRepository.getPartida(idPartida); }
50
51     public void getPartidas(String email) {partidaRepository.getPartidasPorDueño(email);}
52
53     1 usage Miguel A. Freeman
54     public void insertarPartida(Partidas partidas, InsercionCallback insercionCallback){...}
55
56     2 usages Miguel A. Freeman
57     public void SubirFotoPartida(Partidas partida, Uri uri){...}
58
59     2 usages JohnHawkinsGit
60     public void getPartida(String idPartida) { partidaRepository.getPartida(idPartida); }
61
62     JohnHawkinsGit
63     public void eliminarPartida(String idPartida) { partidaRepository.eliminarPartida(idPartida); }
64
65     1 usage JohnHawkinsGit
66     public LiveData<List<Partidas>> getPartidasLivedata() { return partidasData; }
67
68     2 usages JohnHawkinsGit
69     public LiveData<Partidas> getPartidaLivedata() { return partidasLiveData; }
70 }

```

Ilustración 32 Estructura ViewModel



Usaremos para este ejemplo `PartidasViewmodel`, que tendrá la estructura que hemos usado para las clases de este tipo.

La función principal de `PartidasViewModel` es proporcionar y gestionar los datos necesarios para la vista de la actividad relacionada con las partidas del proyecto.

- `partidasData`: Es un objeto de tipo `MutableLiveData` que almacena una lista de objetos `Partidas`. Representa los datos de las partidas disponibles y se actualiza en tiempo real.
- `partidasLiveData`: Es un objeto de tipo `MutableLiveData` que almacena un objeto `Partidas`. Representa los datos de una partida específica y también se actualiza en tiempo real.
- `partidaRepository`: Es una instancia de `PartidaRepository`, que es una clase responsable de acceder y gestionar los datos relacionados con las partidas, como obtener una lista de partidas, insertar una nueva partida, modificar una partida existente, etc.
- `imageRepository`: Es una instancia de `ImageRepository`, que es una clase responsable de gestionar las imágenes asociadas a las partidas, como subir una imagen al servidor y obtener la URL de descarga.
- `init()`: Es un método que se llama para inicializar el `PartidasViewModel` y configurar las instancias de los repositorios y los objetos `LiveData` correspondientes.
- `getPartidas()`: Es un método que permite obtener la lista de partidas asociadas a un correo electrónico específico. Este método utiliza el `partidaRepository` para obtener los datos de las partidas.
- `insertarPartida()`: Es un método para insertar una nueva partida en la base de datos. Toma como parámetros un objeto `Partidas` y un objeto de tipo `InsercionCallback`. Este método utiliza el `partidaRepository` para insertar la partida y ejecuta el callback proporcionado una vez que se ha completado la operación.
- `SubirFotoPartida()`: Es un método para subir una foto asociada a una partida. Toma como parámetros un objeto `Partidas` y una `Uri` que representa la ubicación de la imagen en el dispositivo. Este método utiliza el `imageRepository` para subir la imagen al servidor y actualiza la partida con la URL de descarga correspondiente.
- Otros métodos como `getPartida()`, `eliminarPartida()`, `getPartidasLivedata()` y `getPartidaLivedata()` proporcionan funcionalidades adicionales para obtener información de partidas específicas, eliminar partidas y obtener los objetos `LiveData` correspondientes para observar cambios en los datos.

```

1  package com.example.proyecto_bkd.perfiles.api;
2
3  import ...
12
9 usages  Miguel A. Freeman
13  public class PerfilesRepository {
14      3 usages
15      private static PerfilesRepository instance;
16      1 usage
17      private static final String NOMBRE_COLECCION = "perfiles";
18      6 usages
19      private final CollectionReference coleccion;
20      4 usages
21      private MutableLiveData<List<Perfil>> listaPerfilesLivedata;
22      8 usages
23      private MutableLiveData<Perfil> perfilLiveData;
24
25      4 usages 3 implementations  Miguel A. Freeman
26      public interface UploadPerfilesCallback{
27          1 usage 3 implementations  Miguel A. Freeman
28          void onSuccess(String id);
29          1 usage 3 implementations  Miguel A. Freeman
30          void onFailure(Exception exception);
31      }
32
33      1 usage  Miguel A. Freeman
34      public PerfilesRepository() {
35          listaPerfilesLivedata = new MutableLiveData<>();
36          perfilLiveData = new MutableLiveData<>();
37          coleccion = FirebaseFirestore.getInstance().collection(NOMBRE_COLECCION);
38      }
39
40      //singleton
41      Miguel A. Freeman
42      public static PerfilesRepository getInstance() {
43          if (instance == null) {
44              instance = new PerfilesRepository();
45          }
46          return instance;
47      }
48

```

Ilustración 33 Estructura repositorio(1)

```

39 public void getPerfilesPorDueño(String email) {
40     coleccion.whereEqualTo("email", email).get().addOnSuccessListener(queryDocumentSnapshots -> {
41         List<Perfil> listaPerfiles = new ArrayList<>();
42         for (QueryDocumentSnapshot document : queryDocumentSnapshots) {
43             Perfil perfil = document.toObject(Perfil.class);
44             perfil.setId(document.getId());
45             listaPerfiles.add(perfil);
46         }
47         listaPerfilesLiveData.postValue(listaPerfiles);
48     }).addOnFailureListener(e -> {
49         listaPerfilesLiveData.postValue(null);
50     });
51 }
52
53 //recupera solo un perfil con el id
54 1 usage Miguel A. Freeman
55 public void getPerfil(String idPerfil) {
56     coleccion.document(idPerfil).get().addOnSuccessListener(documentSnapshot -> {
57         Perfil perfil = documentSnapshot.toObject(Perfil.class);
58         perfil.setId(documentSnapshot.getId());
59         perfilLiveData.postValue(perfil);
60     }).addOnFailureListener(e -> {
61         perfilLiveData.postValue(null);
62     });
63 }
64 3 usages Miguel A. Freeman
65 public void insertarPerfil(Perfil perfil, UploadPerfilesCallback listener) {
66     coleccion.add(perfil).addOnSuccessListener(documentReference -> {
67         perfilLiveData.postValue(perfil);
68         listener.onSuccess(documentReference.getId());
69     }).addOnFailureListener(e -> {
70         perfilLiveData.postValue(null);
71         listener.onFailure(e);
72     });
73 }
74 public void modificarPerfil(String idPerfil, Perfil nuevoPerfil) {
75     coleccion.document(idPerfil).set(nuevoPerfil).addOnSuccessListener(aVoid -> {
76         nuevoPerfil.setId(idPerfil);
77         perfilLiveData.postValue(nuevoPerfil);
78     }).addOnFailureListener(e -> {
79         perfilLiveData.postValue(null);
80     });
81 }
82 1 usage Miguel A. Freeman
83 public void eliminarPerfil(String idPerfil){
84     coleccion.document(idPerfil).delete().addOnSuccessListener(aVoid ->{
85     });
86 }
87 1 usage Miguel A. Freeman
88 public MutableLiveData<List<Perfil>> getListaPerfilesLiveData() {
89     return listaPerfilesLiveData;
90 }
91 1 usage Miguel A. Freeman
92 public MutableLiveData<Perfil> getPerfilLiveData() { return perfilLiveData; }
93 }
94 }
95 }

```

Ilustración 34 Estructura repositorio(2)

- **instance:** Es una variable estática que almacena una única instancia de `PerfilesRepository` utilizando el patrón Singleton. Esto garantiza que solo haya una instancia de la clase en toda la aplicación.
- **NOMBRE\_COLECCION:** Es una constante que representa el nombre de la colección en Firebase Firestore donde se almacenan los perfiles.
- **coleccion:** Es una referencia a la colección de perfiles en Firebase Firestore. Se inicializa en el constructor de la clase utilizando `FirebaseFirestore.getInstance().collection(NOMBRE_COLECCION)`.
- **listaPerfilesLivedata:** Es un objeto de tipo `MutableLiveData` que almacena una lista de objetos `Perfil`. Representa los perfiles de usuario y se actualiza en tiempo real.
- **perfilLiveData:** Es un objeto de tipo `MutableLiveData` que almacena un objeto `Perfil`. Representa un perfil de usuario específico y también se actualiza en tiempo real.
- **UploadPerfilesCallback:** Es una interfaz que define dos métodos `onSuccess()` y `onFailure()` para manejar los resultados de las operaciones de carga de perfiles.
- **getInstance():** Es un método estático que devuelve la instancia única de `PerfilesRepository` utilizando el patrón Singleton. Si no existe una instancia, se crea una nueva.
- **getPerfilesPorDueño():** Es un método que recupera los perfiles de usuario asociados a un correo electrónico específico. Utiliza la referencia a la colección de perfiles y realiza una consulta en Firestore para obtener los perfiles que coinciden con el correo electrónico. Luego, convierte los documentos de la consulta en objetos `Perfil` y actualiza el objeto `listaPerfilesLivedata`.
- **getPerfil():** Es un método que recupera un perfil de usuario específico utilizando su ID. Utiliza la referencia a la colección y realiza una consulta en Firestore para obtener el perfil correspondiente al ID. Luego, convierte el documento en un objeto `Perfil` y actualiza el objeto `perfilLiveData`.
- **insertarPerfil():** Es un método para insertar un nuevo perfil de usuario en Firebase Firestore. Toma como parámetros un objeto `Perfil` y un objeto de tipo `UploadPerfilesCallback`. Utiliza la referencia a la colección y agrega el perfil como un nuevo documento. Luego, actualiza el objeto `perfilLiveData` con el perfil insertado y llama al método `onSuccess()` del `UploadPerfilesCallback` para indicar que la operación se realizó correctamente.
- **modificarPerfil():** Es un método para modificar un perfil de usuario existente en Firebase Firestore. Toma como parámetros el ID del perfil y un objeto `Perfil` actualizado. Utiliza la referencia a la colección y actualiza el documento correspondiente al ID con los datos del nuevo perfil. Luego, actualiza el objeto `perfilLiveData` con el perfil modificado.
- **eliminarPerfil():** Es un método para eliminar un perfil de usuario en Firebase Firestore. Toma como parámetro el ID del perfil y utiliza la referencia a la colección para eliminar el documento correspondiente al ID.
- Otros métodos como `getListaPerfilesLivedata()` y `getPerfilLiveData`.

```

55
56 FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
57 if (currentUser != null) {
58     String email = currentUser.getEmail();
59     String uid = currentUser.getUid();
60     // aquí podemos acceder a la info del usuario, podemos usar su mail para relacionarlo a los perfiles y las partidas
61 } else {
62     // El usuario no está autenticado, debes enviarlo a la actividad de inicio de sesión
63 }
64
65 Log.d( tag: "SONANDO PARTIDA", msg: SplashScreen.mp.isPlaying() + "");
66
67 JohnHawkinsGit
68 sMPartida.setOnClickListener(new View.OnClickListener() {
69     JohnHawkinsGit
70     @Override
71     public void onClick(View view) {
72         if (sMPartida.isChecked()) {
73             SplashScreen.mp.start();
74             SplashScreen.music = true;
75         } else {
76             SplashScreen.mp.pause();
77             SplashScreen.music = false;
78         }
79     }
80 });
81
82 JohnHawkinsGit
83 bImgRanking.setOnClickListener(new View.OnClickListener() {
84     JohnHawkinsGit
85     @Override
86     public void onClick(View view) {
87         Intent intent = new Intent( packageContext: PrincipalPartida.this, Ranking.class);
88         startActivity(intent);
89         finish();
90     }
91 });
92
93 bImgPerfiles.setOnClickListener(new View.OnClickListener() {
94     JohnHawkinsGit
95     @Override
96     public void onClick(View view) {
97         Intent intent = new Intent( packageContext: PrincipalPartida.this, ActivityPerfiles.class);
98         startActivity(intent);
99         finish();
100     }
101 });
102
103 JohnHawkinsGit
104 verPartida.setOnClickListener(new View.OnClickListener() {
105     JohnHawkinsGit
106     @Override
107     public void onClick(View view) {
108         Intent intent = new Intent( packageContext: PrincipalPartida.this, VerPartidas.class);
109         startActivity(intent);
110         finish();
111     }
112 });
113
114 JohnHawkinsGit
115 newGame.setOnClickListener(new View.OnClickListener() {
116     JohnHawkinsGit
117     @Override
118     public void onClick(View view) {
119         Intent intent = new Intent( packageContext: PrincipalPartida.this, SeleccionPerfiles.class);
120         startActivity(intent);
121         finish();
122     }
123 });
124
125 JohnHawkinsGit
126 salir.setOnClickListener(new View.OnClickListener() {
127     JohnHawkinsGit
128     @Override
129     public void onClick(View view) { finish(); }
130 });
131
132 JohnHawkinsGit

```

Ilustración 35 Código Actividad Partida(1)

```

120      cerrarSesion.setOnClickListener(new View.OnClickListener() {
121          @Override
122          public void onClick(View view) {
123              FirebaseAuth.getInstance().signOut();
124              Intent intent = new Intent( packageContext PrincipalPartida.this, Login.class);
125              startActivity(intent);
126              finish();
127          }
128      });
129  }
130
131  @Override
132  protected void onPause() {
133      super.onPause();
134      SplashScreen.mp.pause();
135  }
136
137  @Override
138  protected void onResume() {
139      super.onResume();
140      if (SplashScreen.music) {
141          SplashScreen.mp.start();
142      } else {
143          SplashScreen.mp.pause();
144          sMPartida.setChecked(false);
145      }
146  }
147
148  @Override
149  public void onBackPressed() {
150      AlertDialog.Builder builder = new AlertDialog.Builder( context: PrincipalPartida.this);
151
152      View confirmDialogView = getLayoutInflater().inflate(R.layout.alert_salir, root: null);
153      builder.setView(confirmDialogView);
154
155      TextView btnConfirmar = confirmDialogView.findViewById(R.id.tSalirDialog);
156      TextView btnCancelar = confirmDialogView.findViewById(R.id.tCancelarDialog2);
157      AlertDialog alertDialog = builder.create();
158      alertDialog.getWindow().setBackgroundDrawableResource(android.R.color.transparent);
159
160      btnConfirmar.setOnClickListener(v -> {
161          finish();
162      });
163
164      btnCancelar.setOnClickListener(v -> {
165          // cancelar
166          alertDialog.dismiss();
167      });
168      alertDialog.show();
169  }
170  }

```

Ilustración 36 Código Actividad Partida(2)

- **onCreate():** Este método se ejecuta cuando se crea la actividad. En este método se realiza la configuración inicial de la actividad. A continuación, se detalla lo que hace este método:
  1. Se establece el diseño de la actividad mediante `setContentView(R.layout.activity_partida)`.
  2. Se inicializan y se obtienen referencias a los elementos de la interfaz de usuario (botones, `textViews`, `switch`) mediante `findViewById()`.
  3. Se carga una animación desde un recurso de animación utilizando `AnimationUtils.loadAnimation()` y se aplica a algunos elementos de la interfaz de usuario mediante el método `setAnimation()`.
  4. Se obtiene la instancia actual del usuario autenticado utilizando `FirebaseAuth.getInstance().getCurrentUser()`. Si el usuario no está autenticado, se debe redirigir a la actividad de inicio de sesión.
  5. Se configura un listener para el interruptor `sMPartida` que controla la reproducción o pausa de un archivo de audio utilizando la clase `SplashScreen`. Si el interruptor está activado, se reproduce el archivo de audio y se establece la variable estática `music` en `true`; de lo contrario, se pausa el archivo de audio y se establece `music` en `false`.
  6. Se configuran listeners para los botones `blmgRanking`, `blmgPerfiles`, `verPartida`, `newGame`, `salir` y `cerrarSesion`. Cada uno de estos botones inicia una actividad específica mediante la creación de un `intent` y la llamada a `startActivity()`. Además, se llama al método `finish()` para finalizar la actividad actual en algunos casos.
- **onPause() y onResume():** Estos métodos se ejecutan cuando la actividad entra en segundo plano o vuelve al primer plano, respectivamente. En el método `onPause()`, se pausa la reproducción del archivo de audio utilizando `SplashScreen.mp.pause()`. En `onResume()`, se verifica el estado de reproducción de audio (`SplashScreen.music`) y se reanuda la reproducción si es necesario.
- **onBackPressed():** Este método se llama cuando se presiona el botón de retroceso del dispositivo. Muestra un diálogo de confirmación utilizando un `AlertDialog` personalizado. Si se confirma la salida, se llama al método `finish()` para finalizar la actividad.

```

20 public class PerfilesAdapter extends RecyclerView.Adapter<PerfilesAdapter.ViewHolder> {
21     4 usages
22     private List<Perfil> datos = new ArrayList<>();
23
24     2 usages Miguel A. Freeman
25     public interface ItemClickListener {
26         Miguel A. Freeman
27         void onClick(View view, Perfil perfil);
28     }
29
30     3 usages
31     private ItemClickListener clickListener;
32
33     1 usage Miguel A. Freeman
34     public void setClickListener(ItemClickListener itemClickListener) {
35         this.clickListener = itemClickListener;
36     }
37
38     Miguel A. Freeman +1
39     public void setResults(List<Perfil> datos){
40         this.datos = datos;
41         notifyDataSetChanged();
42     }
43
44     4 usages Miguel Ángel desde clase +2
45     public class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
46         1 usage
47         private ImageView imagen_marco, imagen_perfil;
48         2 usages
49         private TextView nombre;
50         1 usage
51         private ImageButtons imagen_madera;
52         public ViewHolder(@NonNull View itemView) {
53             super(itemView);
54             imagen_marco = itemView.findViewById(R.id.id_marco);
55             nombre = itemView.findViewById(R.id.id_perfiles_nombre);
56             imagen_madera = itemView.findViewById(R.id.id_imagen_madera);
57             imagen_perfil = itemView.findViewById(R.id.id_imgProfile);
58             itemView.setOnClickListener(this);
59         }
60
61         Miguel Ángel desde clase
62         public TextView getNombre() { return nombre; }
63
64         Miguel A. Freeman +1
65         @Override
66         public void onClick(View view) {
67             if (clickListener != null) {
68                 clickListener.onClick(view, datos.get(getAdapterPosition()));
69             }
70         }
71     }
72
73     Miguel Ángel desde clase +1
74     @NonNull
75     @Override
76     public PerfilesAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
77         View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.perfiles_view, parent, attachToRoot: false);
78         return new ViewHolder(v);
79     }
80
81     JohnHawkinsGit +2
82     @Override
83     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
84         Perfil p = datos.get(position);
85
86         if (p.getPfpImg() != null) {
87             String imgUrl = p.getPfpImg();
88             Glide.with(holder.itemView).requestManager
89                 .load(imgUrl).requestBuilder<Drawable>
90                 .into(holder.imagen_perfil);
91         }
92     }
93 }

```

Ilustración 37 Código Adapter Perfiles (1)



```

82     } else {
83         if (position == 0) {
84             holder.imagen_perfil.setImageResource(R.drawable.conejo11);
85         }
86         if (position == 1) {
87             holder.imagen_perfil.setImageResource(R.drawable.conejo2);
88         }
89         if (position == 2) {
90             holder.imagen_perfil.setImageResource(R.drawable.conejo3);
91         }
92         if (position == 3) {
93             holder.imagen_perfil.setImageResource(R.drawable.conejo4);
94         }
95         if (position == 4) {
96             holder.imagen_perfil.setImageResource(R.drawable.conejo5);
97         }
98         if (position == 5) {
99             holder.imagen_perfil.setImageResource(R.drawable.conejo6);
100        }
101        if (position == 6) {
102            holder.imagen_perfil.setImageResource(R.drawable.conejo7);
103        }
104        if (position == 7) {
105            holder.imagen_perfil.setImageResource(R.drawable.conejo8);
106        }
107        if (position == 8) {
108            holder.imagen_perfil.setImageResource(R.drawable.conejo10);
109        }
110        if (position == 9) {
111            holder.imagen_perfil.setImageResource(R.drawable.conejo1);
112        }
113    }
114    holder.getNombre().setText(p.getNombre());
115 }
116
117 Miguel Ángel desde clase +1
118 @Override
119 public int getItemCount() { return datos.size(); }
120
121
122 }

```

*Ilustración 38 Código Adapter Perfiles (2)*

- PerfilesAdapter es un adaptador de RecyclerView que se utiliza para mostrar una lista de perfiles en una interfaz de usuario. El adaptador se encarga de inflar la vista de cada elemento de la lista y vincular los datos correspondientes a cada vista.
- La clase PerfilesAdapter define una interfaz ItemClickListener que se utiliza para manejar los clics en los elementos de la lista de perfiles.
- La clase ViewHolder se utiliza para mantener las referencias a los elementos de la vista de cada elemento de la lista. También implementa la interfaz View.OnClickListener para manejar los clics en los elementos de la lista.
- El método onCreateViewHolder() se llama cuando se necesita crear una nueva instancia de ViewHolder. En este método, se infla la vista del elemento de la lista a partir de un archivo de diseño (R.layout.perfiles\_view) y se crea una nueva instancia de ViewHolder con esa vista.
- El método onBindViewHolder() se llama cuando se necesita enlazar los datos de un perfil específico con la vista correspondiente en el ViewHolder. Aquí se obtiene el perfil de la lista de perfiles en la posición especificada y se configuran los datos en la vista. Si el perfil tiene una URL de imagen, se carga utilizando la biblioteca Glide y se muestra en el ImageView correspondiente. Si el perfil no tiene una URL de imagen, se muestra una imagen predeterminada basada en la posición en la lista.
- El método getItemCount() devuelve el número total de perfiles en la lista.

- El método `setResults()` se utiliza para actualizar los datos del adaptador con una nueva lista de perfiles. Se llama a `notifyDataSetChanged()` para notificar al adaptador que los datos han cambiado y que debe actualizar la interfaz de usuario.

```

5 public class Perfil implements Serializable {
6
7     2 usages
8     private String id;
9
10    3 usages
11    private String email;
12
13    4 usages
14    private String nombre;
15
16    2 usages
17    private String pfpImg;
18
19    4 usages
20    private String partidasJugadas;
21
22    4 usages
23    private String maxPuntuacion;
24
25    4 usages
26    private String partidasGanadas;
27
28    4 usages
29    private String porcentajeGanadas;
30
31    Miguel A. Freeman
32    public Perfil() {
33    }
34
35    public Perfil(String email, String nombre, String partidasJugadas, String maxPuntuacion, String partidasGanadas, String porcentajeGanadas) {
36        this.email = email;
37        this.nombre = nombre;
38        this.partidasJugadas = partidasJugadas;
39        this.maxPuntuacion = maxPuntuacion;
40        this.partidasGanadas = partidasGanadas;
41        this.porcentajeGanadas = porcentajeGanadas;
42    }
43
44    Miguel A. Freeman +1
45    public Perfil(String nombre, String partidasJugadas, String maxPuntuacion, String partidasGanadas, String porcentajeGanadas) {
46        this.nombre = nombre;
47        this.partidasJugadas = partidasJugadas;
48        this.maxPuntuacion = maxPuntuacion;
49        this.partidasGanadas = partidasGanadas;
50        this.porcentajeGanadas = porcentajeGanadas;
51    }
52
53    Miguel A. Freeman
54    public String getId() { return id; }
55
56    Miguel A. Freeman
57    public String getEmail() { return email; }
58
59    Miguel A. Freeman
60    public String getNombre() { return nombre; }
61
62    2 usages Miguel A. Freeman
63    public String getPartidasJugadas() { return partidasJugadas; }
64
65    5 usages Miguel A. Freeman
66    public String getMaxPuntuacion() { return maxPuntuacion; }
67
68    5 usages Miguel A. Freeman
69    public String getPartidasGanadas() { return partidasGanadas; }
70
71    5 usages Miguel A. Freeman
72    public String getPorcentajeGanadas() { return porcentajeGanadas; }
73
74    Miguel A. Freeman
75    public void setId(String id) { this.id = id; }
76
77    Miguel A. Freeman
78    public void setEmail(String email) { this.email = email; }
79
80    Miguel A. Freeman
81    public void setNombre(String nombre) { this.nombre = nombre; }
82
83    Miguel A. Freeman
84    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
85
86    Miguel A. Freeman
87    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
88
89    Miguel A. Freeman
90    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
91
92    Miguel A. Freeman
93    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
94
95    Miguel A. Freeman
96    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
97
98    Miguel A. Freeman
99    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
100
101    Miguel A. Freeman
102    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
103
104    Miguel A. Freeman
105    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
106
107    Miguel A. Freeman
108    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
109
110    Miguel A. Freeman
111    public void setEmail(String email) { this.email = email; }
112
113    Miguel A. Freeman
114    public void setId(String id) { this.id = id; }
115
116    Miguel A. Freeman
117    public void setNombre(String nombre) { this.nombre = nombre; }
118
119    Miguel A. Freeman
120    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
121
122    Miguel A. Freeman
123    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
124
125    Miguel A. Freeman
126    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
127
128    Miguel A. Freeman
129    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
130
131    Miguel A. Freeman
132    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
133
134    Miguel A. Freeman
135    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
136
137    Miguel A. Freeman
138    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
139
140    Miguel A. Freeman
141    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
142
143    Miguel A. Freeman
144    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
145
146    Miguel A. Freeman
147    public void setEmail(String email) { this.email = email; }
148
149    Miguel A. Freeman
150    public void setId(String id) { this.id = id; }
151
152    Miguel A. Freeman
153    public void setNombre(String nombre) { this.nombre = nombre; }
154
155    Miguel A. Freeman
156    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
157
158    Miguel A. Freeman
159    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
160
161    Miguel A. Freeman
162    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
163
164    Miguel A. Freeman
165    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
166
167    Miguel A. Freeman
168    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
169
170    Miguel A. Freeman
171    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
172
173    Miguel A. Freeman
174    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
175
176    Miguel A. Freeman
177    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
178
179    Miguel A. Freeman
180    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
181
182    Miguel A. Freeman
183    public void setEmail(String email) { this.email = email; }
184
185    Miguel A. Freeman
186    public void setId(String id) { this.id = id; }
187
188    Miguel A. Freeman
189    public void setNombre(String nombre) { this.nombre = nombre; }
190
191    Miguel A. Freeman
192    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
193
194    Miguel A. Freeman
195    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
196
197    Miguel A. Freeman
198    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
199
200    Miguel A. Freeman
201    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
202
203    Miguel A. Freeman
204    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
205
206    Miguel A. Freeman
207    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
208
209    Miguel A. Freeman
210    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
211
212    Miguel A. Freeman
213    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
214
215    Miguel A. Freeman
216    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
217
218    Miguel A. Freeman
219    public void setEmail(String email) { this.email = email; }
220
221    Miguel A. Freeman
222    public void setId(String id) { this.id = id; }
223
224    Miguel A. Freeman
225    public void setNombre(String nombre) { this.nombre = nombre; }
226
227    Miguel A. Freeman
228    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
229
230    Miguel A. Freeman
231    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
232
233    Miguel A. Freeman
234    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
235
236    Miguel A. Freeman
237    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
238
239    Miguel A. Freeman
240    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
241
242    Miguel A. Freeman
243    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
244
245    Miguel A. Freeman
246    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
247
248    Miguel A. Freeman
249    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
250
251    Miguel A. Freeman
252    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
253
254    Miguel A. Freeman
255    public void setEmail(String email) { this.email = email; }
256
257    Miguel A. Freeman
258    public void setId(String id) { this.id = id; }
259
260    Miguel A. Freeman
261    public void setNombre(String nombre) { this.nombre = nombre; }
262
263    Miguel A. Freeman
264    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
265
266    Miguel A. Freeman
267    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
268
269    Miguel A. Freeman
270    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
271
272    Miguel A. Freeman
273    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
274
275    Miguel A. Freeman
276    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
277
278    Miguel A. Freeman
279    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
280
281    Miguel A. Freeman
282    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
283
284    Miguel A. Freeman
285    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
286
287    Miguel A. Freeman
288    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
289
290    Miguel A. Freeman
291    public void setEmail(String email) { this.email = email; }
292
293    Miguel A. Freeman
294    public void setId(String id) { this.id = id; }
295
296    Miguel A. Freeman
297    public void setNombre(String nombre) { this.nombre = nombre; }
298
299    Miguel A. Freeman
300    public void setPartidasJugadas(String partidasJugadas) { this.partidasJugadas = partidasJugadas; }
301
302    Miguel A. Freeman
303    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
304
305    Miguel A. Freeman
306    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
307
308    Miguel A. Freeman
309    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
310
311    Miguel A. Freeman
312    public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
313
314    Miguel A. Freeman
315    public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
316
317    Miguel A. Freeman
318    public void setPartidasGanadas(String partidasGanadas) { this.partidasGanadas = partidasGanadas; }
319
320    Miguel A. Freeman
321    public void setPorcentajeGanadas(String porcentajeGanadas) { this.porcentajeGanadas = porcentajeGanadas; }
322
3
```

*Ilustración 39 POJO Perfil (1)*

```

66     public String getPfpImg() {return pfpImg;}
67
68     Miguel A. Freeman
69     public void setId(String id) { this.id = id; }
70
71     Miguel A. Freeman
72     public void setEmail(String email) { this.email = email; }
73
74     Miguel A. Freeman
75     public void setNombre(String nombre) { this.nombre = nombre; }
76
77     4 usages Miguel A. Freeman
78     public void setPorcentajeGanadas(String porcentajeGanadas) { return porcentajeGanadas; }
79
80     Miguel A. Freeman
81     public void setPorcentajeGanadas(String porcentajeGanadas) {
82         this.porcentajeGanadas = porcentajeGanadas;
83     }
84
85     Miguel A. Freeman
86     public void setPartidasJugadas(String partidasJugadas) {
87         this.partidasJugadas = partidasJugadas;
88     }
89
90     Miguel A. Freeman
91     public void setMaxPuntuacion(String maxPuntuacion) { this.maxPuntuacion = maxPuntuacion; }
92
93     Miguel A. Freeman
94     public void setPartidasGanadas(String partidasGanadas) {
95         this.partidasGanadas = partidasGanadas;
96     }
97
98     4 usages Miguel A. Freeman
99     public void setPfpImg(String pfpImg) { this.pfpImg = pfpImg; }
100
101 }
102
103
104

```

*Ilustración 40 POJO Perfil (2)*

La clase Perfil implementa la interfaz Serializable, lo que significa que los objetos de esta clase pueden convertirse en una secuencia de bytes para ser transmitidos o almacenados.

Atributos de la clase Perfil:

- id: un identificador único para el perfil.
- email: el correo electrónico del usuario.
- nombre: el nombre del usuario.
- pfpImg: la imagen de perfil del usuario.
- partidasJugadas: el número de partidas jugadas por el usuario.
- maxPuntuacion: la puntuación máxima obtenida por el usuario.
- partidasGanadas: el número de partidas ganadas por el usuario.
- porcentajeGanadas: el porcentaje de partidas ganadas por el usuario.

**Constructores:**

- Perfil(): constructor sin argumentos.
- Perfil(email, nombre, partidasJugadas, maxPuntuacion, partidasGanadas, porcentajeGanadas): constructor que acepta el correo electrónico, nombre, partidas jugadas, puntuación máxima, partidas ganadas y porcentaje de partidas ganadas del usuario.
- Perfil(nombre, partidasJugadas, maxPuntuacion, partidasGanadas, porcentajeGanadas): constructor que acepta el nombre, partidas jugadas, puntuación máxima, partidas ganadas y porcentaje de partidas ganadas del usuario.

**Métodos de acceso (getters) y modificación (setters):**

- Los métodos getId(), getEmail(), getNombre(), getPartidasJugadas(), getMaxPuntuacion(), getPartidasGanadas(), getPfpImg(), getPorcentajeGanadas() devuelven los valores de los atributos correspondientes.

Los métodos setId(), setEmail(), setNombre(), setPartidasJugadas(), setMaxPuntuacion(), setPartidasGanadas(), setPfpImg(), setPorcentajeGanadas() se utilizan para establecer nuevos valores a los atributos correspondientes.

**4. Corrección de posibles errores**

1. Identificación del error:
  - Durante la fase de implementación, es importante identificar cualquier error o anomalía en el código. Esto puede incluir errores de sintaxis, errores lógicos o problemas de funcionamiento del código.
2. Registro del error:
  - Cada error identificado debe ser registrado de manera detallada en un sistema de seguimiento de incidencias o en un registro específico para el control de errores. El registro debe incluir una descripción clara del error, la ubicación del código afectado, los pasos para reproducir el error y cualquier información relevante para comprender y resolver el problema.
3. Análisis del error:
  - Una vez registrado, se debe realizar un análisis exhaustivo del error para comprender su causa raíz y su impacto en el funcionamiento del sistema. Esto puede implicar revisar el código relacionado, realizar pruebas adicionales y utilizar herramientas de depuración para identificar la fuente del error.
4. Reproducción del error:
  - Es importante poder reproducir el error de manera consistente para poder analizarlo y resolverlo de manera efectiva. Esto implica seguir los pasos registrados para reproducir el error y verificar si se produce de manera coherente.
5. Corrección del error:
  - Una vez que se ha identificado la causa raíz del error, se deben aplicar las correcciones necesarias en el código. Esto puede implicar modificar la lógica del programa, corregir errores de sintaxis, optimizar el rendimiento o aplicar soluciones alternativas según corresponda.
6. Pruebas de verificación:
  - Después de corregir el error, se deben realizar pruebas exhaustivas para verificar que el problema haya sido resuelto correctamente. Esto implica ejecutar pruebas unitarias, pruebas de integración y cualquier otro tipo de pruebas relevantes para asegurar que el código funcione como se espera y que el error haya sido solucionado.
7. Documentación y seguimiento:
  - Es importante documentar el proceso de corrección del error, incluyendo los pasos tomados, las modificaciones realizadas en el código y los resultados de las pruebas.

de verificación. Además, se deben establecer mecanismos de seguimiento para monitorear si el error se reproduce nuevamente y tomar medidas preventivas si es necesario.

## 5.4 Fase de pruebas

### 5.4.1 Caso de prueba 1: Comprobación registro de usuario.

- Descripción:  
Comprobar que tras rellenar el formulario de registro se ha guardado en la bbdd y puede hacer login el usuario.
- Condiciones de ejecución:  
Crear un usuario desde cero.
- Entrada:  
No se necesita ningún dato previo para realizar la prueba.
- Resultado esperado:  
BBDD actualizada con el usuario nuevo y puede hacer login en la aplicación.
- Resultado obtenido  
La bbdd se actualiza correctamente y el usuario puede hacer login.
- Evaluación  
La prueba es satisfactoria y obtenemos el resultado deseado.

### 5.4.2 Caso de prueba 2: Comprobación de navegación entre actividades.

- Descripción  
Desplazarse entre actividades comprobando que la opción seleccionada te envía a la actividad deseada.
- Condiciones de ejecución  
No es necesario
- Entrada  
No se necesita ningún dato previo para realizar la prueba.
- Resultado esperado  
El usuario puede desplazarse a cualquier actividad sin que la aplicación provoque algún error.
- Resultado obtenido  
La navegación es correcta y el usuario se puede desplazar a las actividades que selecciona.
- Evaluación  
El usuario puede navegar por todas las actividades sin error.

### 5.4.3 Caso de prueba 3: Crear nuevo perfil y modificar datos.

- Descripción:  
Crear un perfil nuevo introduciendo el nombre y las diferentes opciones de cambiar la imagen de perfil, comprobar que se almacena en la bbdd y se muestra en el recycler-view de la aplicación.
- Condiciones de ejecución  
Estar ya logueado en la aplicación.
- Entrada  
Tener un perfil ya creado para hacer login.
- Resultado esperado  
Se puede crear un nuevo perfil escogiendo la foto desde la galería o haciéndola desde la cámara de fotos del móvil o si no se escoge ninguna opción la aplicación elige una foto aleatoria de las que tiene almacenadas.
- Resultado obtenido  
Se puede crear un perfil nuevo y se puede modificar el nombre y añadir una imagen desde galería o desde la cámara.
- Evaluación  
La prueba ha obtenido el resultado esperado.

### 5.4.4 Caso de prueba 4: Colores de los jugadores

- Descripción  
Comprobar que al pulsar sobre un jugador el marco cambia de color y al comenzar la partida comprobar que la aplicación guarda el color correcto de cada jugador en la bbdd.
- Condiciones de ejecución  
Estar ya logueado en la aplicación e iniciar partida.
- Entrada  
Tener varios perfiles ya creados para poder seleccionar jugadores, tener los marcos de diferentes colores ya guardados en la app.
- Resultado esperado  
Se puede crear un nuevo perfil escogiendo la foto desde la galería o haciéndola desde la cámara de fotos del móvil o si no se escoge ninguna opción la aplicación elige una foto aleatoria de las que tiene almacenadas.
- Resultado obtenido  
Los marcos cambian de color y funciona la validación para que al seleccionar no haya 2 jugadores con el mismo color y haya entre 2 y 4 jugadores seleccionados
- Evaluación  
La prueba ha sido correcta, el resultado es el esperado.

### 5.4.5 Caso de prueba 5: Comprobar recyclerView ResumenTurno.

- Descripción  
Comprobar que al seleccionar las torres y recursos de NuevoFeudo se cargan los datos correctamente en el recyclerView de ResumenTurno y se puede hacer clic sobre el feudo y modificarlo
- Condiciones de ejecución  
Jugar un turno de la partida.
- Entrada  
Tener jugadores ya cargados.
- Resultado esperado  
El feudo se carga correctamente, se añade la puntuación, si se modifica el feudo la puntuación se modifica al volver a cargar con los datos correctos.
- Resultado obtenido  
Al añadir un segundo feudo y posteriores se modifican todos por el nuevo y se duplican.
- Evaluación  
Solucionado, la variable era static y eso impedía que cada feudo fuera independiente, tras esto la prueba es correcta.

### 5.4.6 Caso de prueba 6: Guardar la partida en bbdd.

- Descripción  
Comprobar que al finalizar la partida se guarda con los datos correctos en la bbdd y se muestra en el recyclerView de VerPartidas.
- Condiciones de ejecución  
Jugar una partida.
- Entrada  
Tener jugadores ya cargados.
- Resultado esperado  
La bbdd se actualiza con la partida cargada correctamente y se muestra en el recyclerView VerPartidas.
- Resultado obtenido  
El resultado es correcto, la bbdd se actualiza y al finalizar la partida se guarda con los datos correctos.  
En la actividad VerPartidas se muestra la partida recién jugada.
- Evaluación  
La prueba ha salido como esperaba, el dato de los colores en DetallePartida está en castellano y si se juega en inglés no se modifica, se sustituye por cuadros de colores para solucionar el problema del idioma.

### 5.4.7 Caso de prueba 7: Rankings ordenados.

- Descripción  
Comprobar que los ranking cargan la información correcta al pulsar el botón correspondiente y son ordenados por puntuación, victorias o porcentaje de victorias
- Condiciones de ejecución  
Ninguna
- Entrada  
Tener varios perfiles con datos diferentes.
- Resultado esperado  
Los rankings se ordenan correctamente al pulsar sobre el botón correspondiente.
- Resultado obtenido  
Los rankings muestran los datos correctos y ordenados.
- Evaluación  
La prueba es correcta, al pulsar cada botón se ordena y muestra los datos correctos.

## 6 DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN

El procedimiento de evaluación de las **incidencias** que puedan presentarse durante la realización de las diferentes actividades es el siguiente:

- Identificación de la incidencia: Cuando se detecta una incidencia durante la realización de una actividad, se debe identificar claramente cuál es el problema o situación anormal que ha surtido.
- Registro de la incidencia: Es importante llevar un registro detallado de la incidencia, incluyendo la descripción del problema, la fecha y hora de detección, y cualquier otra información relevante. Esto facilitará su posterior análisis y seguimiento.
- Análisis de la incidencia: Se debe realizar un análisis exhaustivo de la incidencia para comprender su causa raíz y evaluar su impacto en el proyecto. Esto puede implicar revisar los registros, consultar con los miembros del equipo involucrados y realizar pruebas adicionales si es necesario.
- Evaluación del impacto: Se evalúa el impacto de la incidencia en términos de cronograma, presupuesto, recursos y calidad del proyecto. Esto ayuda a determinar si se requieren acciones correctivas o preventivas y cuáles serían las mejores opciones a tomar.
- Acciones correctivas: Una vez evaluado el impacto, se deben definir y aplicar las acciones correctivas necesarias para resolver la incidencia. Esto puede incluir la modificación del plan de actividades, asignación de recursos adicionales, ajuste de plazos, entre otros.
- Seguimiento y control: Se realiza un seguimiento continuo para asegurarse de que las acciones correctivas estén siendo implementadas de manera efectiva y que la incidencia esté siendo resuelta. Además, se establecen mecanismos de control para prevenir la recurrencia de la incidencia en el futuro.

Procedimiento de gestión de cambios en recursos y actividades:

- Identificación del cambio: Cuando surge la necesidad de realizar un cambio en los recursos o actividades del proyecto, se debe identificar claramente cuál es el cambio requerido y cuál es su objetivo.
- Evaluación del cambio: Se evalúa el impacto del cambio en términos de cronograma, presupuesto, recursos y calidad del proyecto. Esto implica analizar cómo afectaría el cambio a las



actividades existentes, si requiere la adición o reasignación de recursos, y si implica modificaciones en los entregables o metas del proyecto.

- Análisis de viabilidad: Se analiza la viabilidad del cambio, considerando factores como la disponibilidad de recursos, la capacidad de absorción del cambio por parte del equipo, y si el cambio está alineado con los objetivos y restricciones del proyecto.
- Aprobación del cambio: Una vez evaluada la viabilidad, se solicita la aprobación del cambio a la autoridad competente, que puede ser el responsable del proyecto, el comité de dirección o cualquier instancia definida en la estructura de gobierno del proyecto.
- Implementación del cambio: Una vez aprobado, se procede a implementar el cambio. Esto implica comunicar el cambio a los miembros del equipo involucrados, actualizar los planes y documentos pertinentes, y realizar las modificaciones necesarias en las actividades y recursos afectados.
- Seguimiento y control: Se realiza un seguimiento continuo para asegurarse de que el cambio se implemente correctamente y cumpla con los objetivos establecidos. Además, se establecen mecanismos de control para monitorear los efectos del cambio y realizar ajustes si es necesario.

## 7 FUENTES

<https://stackoverflow.com/>

<https://www.flaticon.es/>

<https://creativecommons.org/>

<https://www.jamendo.com/>

<https://www.youtube.com/@mouredev>

<https://www.youtube.com/@CristianDavidHenao>

<https://developer.android.com/>

<https://www.remove.bg/es>

<https://chat.openai.com/>

<https://www.dafont.com/es/>

<https://www.google.es/imghp?hl=es&tab=ri&ogbl>

<https://facebook.github.io/shimmer-android/>

<https://www.youtube.com/@AristiDevs>

## 8 ANEXOS

No es necesario ningún anexo.