IES RIBERA DE CASTILLA

MÓDULO PROYECTO

CFGS Desarrollo de Aplicaciones Multiplataforma

Informática y Comunicaciones

MiPadel

Tutor individual: *Cristina Silvan Pardo*

Tutor colectivo: *Cristina Silvan pardo*

Año: *2025*

Fecha de presentación: *11/12/2025*

Nombre y Apellidos: Saúl Mantecón de Caso

Email: saulmantecon9vdc@gmail.com

Tabla de contenido

1 Identificación proyecto.

MiPadel es una aplicación móvil desarrollada para la plataforma Android cuyo propósito principal es facilitar la organización, gestión y seguimiento de partidos de pádel entre jugadores aficionados. El proyecto consiste en el diseño y la implementación de un sistema que permite crear partidos, gestionar las posiciones disponibles, unirse a encuentros ya existentes y registrar los resultados una vez finalizados. Además, la aplicación ofrece funcionalidades complementarias como la visualización del historial de partidos, la generación automática de estadísticas personales y la posibilidad de editar el perfil.

La aplicación ha sido desarrollada utilizando Kotlin y Jetpack Compose como tecnologías principales en el cliente, junto con los servicios de Firebase para la autenticación, el almacenamiento de datos y la persistencia en la nube. Asimismo, se ha seguido una arquitectura basada en el patrón MVVM con el fin de garantizar una estructura modular, mantenible y escalable. En conjunto, MiPadel constituye un proyecto completo que integra diseño de interfaz, lógica de negocio, sincronización de datos en tiempo real y una experiencia de usuario orientada tanto a la funcionalidad como a la simplicidad.

2 Organización de la memoria

La memoria del proyecto se estructura en varios apartados que permiten presentar de forma clara todo el proceso seguido hasta la creación de la aplicación:

- **Identificación del proyecto:** Se incluye la información básica del trabajo (título, autor y datos académicos) junto con una breve descripción que sitúa el proyecto.
- **Descripción general del proyecto:** Se explican los objetivos planteados, el enfoque seguido para desarrollarlo y los criterios metodológicos que se han tenido en cuenta. También se detallan las herramientas y tecnologías utilizadas.

- **Descripción general del producto:** Se presenta la aplicación desarrollada, sus funcionalidades principales, el tipo de usuario al que va dirigida y la forma en que se relaciona con otros elementos del sistema. Además, se resumen las técnicas o arquitecturas aplicadas y el proceso de despliegue o instalación.
- **Planificación y presupuesto:** Se describe la organización temporal del trabajo, las fases de desarrollo y una estimación de los recursos necesarios para llevar a cabo el proyecto.
- **Documentación técnica:** Incluye los requisitos del sistema, el análisis previo, el diseño de la aplicación (base de datos, interfaz y estructura interna), la implementación y las pruebas realizadas.
- **Manuales de usuario e instalación:** Se detallan los pasos para usar la aplicación y cómo instalarla en el entorno correspondiente.
- **Conclusiones y posibles ampliaciones:** Se presenta una valoración final del proyecto, las dificultades encontradas y las mejoras que podrían incorporarse en el futuro.
- **Bibliografía:** Recoge las fuentes de información consultadas durante el desarrollo.
- **Anexos:** Incluye material complementario como capturas, fragmentos de código, documentación adicional o elementos de diseño.

3 Descripción general del proyecto.

El proyecto MiPadel nace de una necesidad real detectada en mi entorno cercano. Desde que comencé el segundo curso del ciclo DAM, los fines de semana intentaba jugar al pádel en la pista municipal de mi pueblo. Sin embargo, el proceso para organizar un partido resultaba poco práctico: para reservar era necesario acudir físicamente al bar del pueblo, sin saber de antemano si la pista estaba libre, y muchas veces era complicado reunir a cuatro personas disponibles para jugar.

A partir de esta problemática surgió la idea de desarrollar una aplicación que facilitara tanto la consulta de reservas como la organización de partidos entre usuarios del mismo entorno. Con el avance del curso y a medida que aprendía nuevas tecnologías, muchos ejercicios y pequeños proyectos fueron orientándose hacia esta idea general, hasta que finalmente decidí convertirla en mi Proyecto Final.

Mi intención ha sido trasladar a una aplicación real todo lo aprendido durante el ciclo y, al mismo tiempo, crear una herramienta útil que en un futuro pueda utilizarse en mi propio pueblo si se desarrolla y amplía adecuadamente.

3.1 Objetivos

El proyecto tiene como finalidad desarrollar una aplicación móvil que permita gestionar de forma ágil partidos de pádel. Los objetivos principales son:

- Permitir consultar la disponibilidad de pistas sin tener que desplazarse físicamente.
- Facilitar la creación de partidos indicando ubicación, fecha y número de jugadores necesarios.
- Ofrecer un sistema mediante el cual los usuarios puedan unirse a partidos ya creados si quedan plazas libres.
- Diseñar una interfaz clara e intuitiva que se adapte a las necesidades reales de los usuarios.
- Integrar en un único proyecto los conocimientos adquiridos a lo largo del ciclo formativo, especialmente en programación con Kotlin y desarrollo en Android Studio.

Aunque en el ciclo se trabajó con Kotlin y las bases del desarrollo en Android, para sacar adelante este proyecto he tenido que aprender por mi cuenta varias tecnologías que no se ven en clase. Herramientas como Jetpack Compose, Firestore o algunos patrones de diseño más que son relativamente nuevos y no forman parte del temario principal, así que tuve que investigar, ver ejemplos y apoyarme en documentación oficial para poder utilizarlas correctamente.

Este aprendizaje extra no sustituye lo que he visto durante el curso, sino que me ha servido para completar y ampliar esos conocimientos y poder desarrollar una aplicación más realista y funcional.

3.2 Cuestiones metodológicas

El desarrollo se ha planteado siguiendo una metodología iterativa. En lugar de intentar construir la aplicación completa desde el comienzo, se ha dividido en pequeñas fases funcionales: autenticación, estructura básica del modelo de datos, creación de partidos, unión de jugadores, estados del partido, vistas finales y estadísticas. Cada una de estas fases se ha probado de manera independiente antes de avanzar a la siguiente.

A lo largo del desarrollo se han aplicado principios como:

- Arquitectura MVVM para separar correctamente la lógica, los datos y la interfaz.
- Uso de corrutinas y StateFlow para la gestión reactiva del estado.
- Validación continua sobre dispositivo físico para asegurar un comportamiento realista.
- Documentación incremental de funciones, decisiones de diseño y cambios relevantes.

Este enfoque ha permitido ajustar la aplicación a medida que evolucionaba el aprendizaje técnico y aparecían nuevas necesidades.

3.3 Entorno de trabajo (tecnologías de desarrollo y herramientas)

El entorno principal de desarrollo ha sido Android Studio, programando en Kotlin y utilizando Jetpack Compose para la creación de la interfaz gráfica. La gestión del proyecto se ha realizado con Gradle, incorporando librerías modernas del ecosistema Android como Navigation Compose, Material 3, Coil para la carga de imágenes y componentes de Jetpack para la gestión del estado y el ciclo de vida.

Para la parte de backend y persistencia de datos se han utilizado los servicios de Firebase, concretamente:

- **Firebase Authentication**, utilizado para el registro, inicio de sesión y mantenimiento de la sesión del usuario.

- **Firestore Database**, empleado como base de datos NoSQL para almacenar usuarios, partidos, posiciones, estadísticas y el resto de la información necesaria para el funcionamiento de la aplicación.

Como herramientas complementarias también se han utilizado:

- **GitHub**, para el control de versiones y seguimiento del desarrollo.

- **Figma**, para crear prototipos y definir el diseño visual de la aplicación antes de implementarlo.

- Documentación oficial de **Android**, **Jetpack Compose** y **Firebase**, empleada como referencia técnica continua durante el desarrollo.

La gestión de imágenes de perfil se ha resuelto mediante ImgBB, un servicio externo donde se alojan las fotografías subidas por los usuarios.

4 Descripción general del producto

4.1 Visión general del sistema: límites del sistema, funcionalidades básicas, usuarios y/o otros sistemas con los que pueda interactuar.

4.1.1 Visión general del sistema.

La aplicación desarrollada es un sistema destinado a la gestión y organización de partidos de pádel, permitiendo a cualquier usuario consultar partidos activos, crear nuevas reservas, unirse como jugador y relacionarse con otros usuarios mediante un sistema de solicitudes de amistad.

El objetivo principal es facilitar la coordinación de partidos dentro de un entorno local como un pueblo o club deportivo, ofreciendo un medio rápido para saber si hay reservas disponibles y encontrar jugadores interesados en unirse.

La aplicación está pensada para usuarios habituales de pistas de pádel que necesitan una herramienta sencilla para gestionar partidos y para conectarse con otros jugadores sin necesidad de utilizar grupos de mensajería o depender del establecimiento que gestiona la pista.

4.1.2 Límites del sistema.

La aplicación es autónoma, aunque depende de servicios externos para su funcionamiento:

- **Firebase Authentication** para el inicio de sesión y el registro de usuarios.
- **Firebase Firestore** para almacenar partidos, posiciones, solicitudes, estadísticas y perfiles.
- **ImgBB** como almacenamiento externo para las imágenes de perfil.

La aplicación no forma parte de un sistema mayor ni interactúa con servicios adicionales. Todo el funcionamiento se concentra en el dispositivo Android y en los servicios en la nube mencionados.

4.1.3 Funcionalidades Básicas.

- Gestión de usuarios:
  - Registro mediante email, nombre y nombre de usuario.
  - Inicio de sesión con Firebase Authentication.
  - Opción de mantener la sesión iniciada.
  - Edición del perfil.

- Gestión de partidos:
  - Creación de partidos indicando ubicación, fecha y hora.
  - Posibilidad de unirse a un partido seleccionando una posición vacía.
  - Visualización del estado de cada partido: pendiente, listo, jugando o finalizado.
  - Eliminación del partido por parte del creador.
  - Introducción del resultado final mediante sets.

- Participación en partidos:
  - Unirse o abandonar un partido (siempre que no esté en juego).
  - Visualización de jugadores, imágenes y posiciones.

- Comunidad y relaciones entre usuarios:
  - Búsqueda de usuarios por nombre.
  - Envío, recepción, aceptación y rechazo de solicitudes de amistad.
  - Lista de amigos integrada en la pantalla de Comunidad.

- Estadísticas:
  - Historial de partidos finalizados.
  - Cálculo de partidos jugados, victorias, derrotas y porcentaje de victorias.
  - Visualización organizada del rendimiento del usuario.
- Personalización y preferencias:
  - Selección de tema: claro, oscuro o siguiendo el sistema.
  - Autologin mediante DataStore.

Usuarios y autenticación.

Existe un único tipo de usuario: usuario registrado.

Para utilizar la aplicación es obligatorio crear una cuenta válida mediante correo electrónico.

El proceso de autenticación se realiza íntegramente con Firebase Authentication y permite mantener la sesión iniciada mediante DataStore si el usuario lo solicita.

No existen roles diferentes ni niveles de permisos especiales: todos los usuarios comparten el mismo conjunto de funcionalidades.

4.1.4 Compatibilidad con sistemas.

La aplicación está diseñada específicamente para **Android**, desarrollada con Jetpack Compose y Kotlin.

Requiere un dispositivo con Android 8.0 (API 26) o superior y conexión a internet para sincronizar datos con Firebase e ImgBB.

4.2 Descripción breve de métodos, técnicas o arquitecturas(m/t/a) utilizadas.

El proyecto emplea una arquitectura basada en MVVM (Model–View–ViewModel), ampliamente utilizada en el desarrollo moderno para Android. Este patrón facilita:

- Separación clara entre interfaz, lógica y datos.
- Mejor organización del código y facilidad de mantenimiento.
- Un flujo de datos estable gracias a StateFlow y LiveData.

Además, se han utilizado las siguientes técnicas y herramientas:

- Jetpack Compose para la construcción declarativa de interfaces.
- Navigation Compose para la gestión estructurada de pantallas y navegación.
- DataStore Preferences para almacenar ajustes como el modo de tema y el autologin.
- Coil para la carga eficiente de imágenes desde URLs externas.
- Repositorios para encapsular el acceso a Firebase y mantener el ViewModel libre de código dependiente de la red.
- Programación asíncrona con corrutinas, lo que mejora el rendimiento y la fluidez de la aplicación.
- Control de versiones con GitHub, donde se ha gestionado el desarrollo del proyecto.

Estas decisiones técnicas han permitido construir una aplicación modular, escalable y fácil de extender en futuras ampliaciones.

4.3 Despliegue de la aplicación indicando plataforma tecnológica, instalación de la aplicación y puesta en marcha.

4.3.1 Plataforma tecnológica.

La aplicación ha sido desarrollada y probada en:

- **Sistema operativo del desarrollador:** Windows 11.

- **Entorno de desarrollo:** Android Studio.
- **Lenguaje de programación:** Kotlin.
- **Framework de UI:** Jetpack Compose.
- **Servicios externos:** Firebase Auth, Firestore, ImgBB.

4.3.2 Instalación de la aplicación.

La aplicación puede instalarse de dos formas:

Instalación desde Android Studio (modo desarrollador):

1. Clonar el repositorio del proyecto.
2. Abrirlo en Android Studio.
3. Conectar un dispositivo Android o iniciar un emulador.
4. Ejecutar la aplicación con el botón *Run*.

5 Planificación y presupuesto.

5.1 Planificación del proyecto.

La planificación del proyecto abarca desde el 15 de septiembre hasta el 11 de diciembre, coincidiendo con el periodo oficial de prácticas.

Durante este tiempo, la dedicación no fue uniforme. Las primeras semanas estuvieron muy centradas en formación autónoma, ya que tecnologías clave como Jetpack Compose, Firebase Firestore, Firebase Authentication o el manejo de estados en Compose no tenía casi conocimiento, por lo que fue necesario dedicar bastante tiempo a aprender conceptos, revisar documentación oficial, videotutoriales y entender patrones modernos como MVVM aplicados a Android.

Fases de trabajo.

La evolución del proyecto siguió una secuencia lógica, aunque priorizando las pantallas y conceptos más simples que me permitían ir teniendo una curva de aprendizaje progresiva, sin empezar con lo más difícil.

La planificación real del proyecto puede dividirse en las siguientes fases:

1. Investigación y aprendizaje inicial.

Durante las primeras semanas, el tiempo se dedicó principalmente a comprender las tecnologías que iban a utilizarse. Además de repasar bases de Kotlin y Android, fue necesario aprender desde cero conceptos como el desarrollo con Jetpack Compose, la integración con Firebase o las nuevas formas de gestionar estados y navegación en Android moderno. Esta fase fue clave para plantear la estructura de la aplicación.

2. Implementación de pantallas base (inicio de sesión y registro).

Una vez asentada la parte teórica, el proyecto comenzó por lo más esencial: el acceso a la aplicación. Se desarrollaron las pantallas de inicio de sesión y registro de usuarios, junto con la conexión al sistema de autenticación. Esta fase permitió validar que la aplicación funcionaba correctamente con un usuario real.

3. Estructura principal de la aplicación.

Posteriormente se definió la navegación interna y la estructura de la aplicación mediante un componente principal que actúa como contenedor de todas las pantallas. También se incorporó el menú lateral (drawer) y la barra superior, elementos necesarios para ofrecer una experiencia de usuario coherente.

4. Desarrollo de la pantalla de perfil.

La siguiente fase consistió en la creación de la pantalla de Perfil, permitiendo al usuario modificar sus datos o actualizar su foto. Aquí se integró el sistema externo de almacenamiento de imágenes y la gestión de la información del usuario dentro de la aplicación.

5. Desarrollo de la pantalla de Comunidad.

Una de las funcionalidades clave del proyecto fue permitir a los usuarios buscar a otras personas y relacionarse con ellas. Se implementó la búsqueda, las solicitudes de amistad y la gestión de amigos.

6. Primera versión de estadísticas

Antes de completar el sistema de partidos, se preparó una pantalla de estadísticas, aunque inicialmente sin datos reales, a la espera del módulo que completaría la parte central de la aplicación.

7. Desarrollo de la pantalla Home y sistema de partidos

La fase más extensa del desarrollo fue la relacionada con los **partidos**: creación, ocupación de plazas, estados del partido y registro de resultados. Esta parte se dejó para el final porque dependía de que el resto de la aplicación ya estuviera construida y estable.

8. Pruebas finales, corrección de errores y revisión del código

Durante todo el proyecto se fueron corrigiendo errores conforme aparecían, pero en las últimas semanas se realizó una revisión completa: ajustes visuales, mejoras en la claridad del código, corrección de estados inconsistentes y comprobación de que cada pantalla ofrecía la experiencia esperada.

Aunque el proyecto está oficialmente planteado para unas 30 horas de trabajo, en la práctica la dedicación real ha sido mucho mayor. Entre el proceso de aprendizaje de nuevas tecnologías, la planificación del sistema, el desarrollo de todas las pantallas y funcionalidades, las pruebas continuas y la elaboración de la memoria, estimo que el esfuerzo total invertido ronda las **300 horas** aproximadamente.

5.2 Presupuesto del proyecto.

El desarrollo de la aplicación MiPadel supone una inversión combinada de tiempo, conocimientos y recursos técnicos. Aunque se trata de un proyecto académico, es posible estimar su coste económico aproximado considerando los principales elementos necesarios para su creación.

El coste más significativo corresponde al tiempo dedicado al diseño, implementación, pruebas y documentación del proyecto. Tomando como referencia un coste estándar de un desarrollador junior 12 €/hora el coste asociado al desarrollo sería:

300 horas × 12 €/hora = 3.600 €

Todas las herramientas empleadas son gratuitas o cuentan con planes gratuitos suficientes para el alcance del proyecto, por lo que el coste es 0 €.

Herramientas utilizadas:

- Android Studio (gratuito).
- Kotlin y Jetpack Compose (tecnologías libres).
- Firebase (plan gratuito).
- ImgBB para alojamiento de imágenes (plan gratuito).

- GitHub para control de versiones (gratuito).
- Figma para diseño de prototipos (plan gratuito).

El desarrollo se ha realizado con un equipo personal ya disponible, por lo que no supone un coste asociado al proyecto. Se considera un coste 0 €.

El hosting de datos se realiza a través de Firebase en su plan gratuito, suficiente para el volumen del proyecto y sin costes adicionales.

ImgBB también se utiliza dentro del límite del plan gratuito.

Coste total de hosting: 0 €.

6 Documentación Técnica: análisis, diseño, implementación y pruebas.

6.1 Especificación de requisitos.

En este apartado se detallan los requisitos que debe cumplir la aplicación *MiPadel*, es decir, las funciones que ofrece y las condiciones necesarias para utilizarlas. A diferencia de la visión general anterior, aquí se describen de forma concreta las acciones que puede realizar el usuario y las validaciones básicas que aplica el sistema. Para organizarlo mejor, los requisitos se dividen en funcionales y no funcionales.

6.1.1 Requisitos funcionales.

1. **Registro y autenticación.**
    - La aplicación permite registrar nuevos usuarios introduciendo nombre, nombre de usuario, correo y contraseña.
    - El inicio de sesión se realiza mediante correo y contraseña a través de Firebase Authentication.
    - Se valida que todos los campos sean obligatorios y que el correo tenga un formato válido.

- El usuario puede marcar la opción de “mantener sesión iniciada”, almacenada con DataStore.

2. **Gestión del perfil.**
   - El usuario puede consultar y editar sus datos básicos: nombre, nombre de usuario, nivel y foto de perfil.
   - La foto de perfil puede cambiarse seleccionando una imagen del dispositivo, que se sube automáticamente a ImgBB.
   - La URL de la imagen se guarda en Firestore.
   - Se impiden valores vacíos en campos obligatorios.

3. **Gestión de la comunidad.**
   - El usuario puede buscar a otros usuarios por nombre de usuario.
   - Se pueden enviar, aceptar o rechazar solicitudes de amistad.
   - Se muestra la lista de amigos actuales.
   - El usuario puede eliminar amigos, actualizando la relación en Firestore.
   - Se evita enviar solicitudes duplicadas o a uno mismo.

4. **Organización de partidos.**
   - Los usuarios pueden crear partidos indicando ubicación, fecha, hora.
   - Un partido contiene cuatro posiciones (dos equipos de dos jugadores) que los usuarios pueden ocupar.
   - El creador del partido puede modificarlo o borrarlo mientras no esté finalizado.

- Cualquier usuario puede unirse a un partido ocupando una posición disponible.

- Un usuario puede abandonar un partido si este todavía no está en juego.

5. **Sistema de juego y resultados.**
   - Cuando un partido comienza, el creador puede introducir los resultados de hasta tres sets.

   - El sistema valida que los juegos introducidos sean números válidos.

   Al finalizar un partido:

   - Se determina automáticamente el equipo ganador.

   - Se actualizan estadísticas personales de los usuarios: partidos jugados, ganados y perdidos.

   - El partido pasa a la colección de partidos finalizados.

6. **Historial y estadísticas.**
   - El usuario puede consultar sus estadísticas globales.

   - Se muestra un historial de todos los partidos finalizados en los que ha participado.

   - Los partidos se ordenan desde el más reciente al más antiguo.

   - Para cada partido se muestran jugadores, resultado y fecha.

7. **Navegación e interfaz.**
   - La aplicación utiliza navegación mediante Jetpack Compose Navigation.
   - Se gestionan pantallas principales: Home, Perfil, Comunidad, Estadísticas, Inicio de sesión y Registro.

- Cada pantalla muestra información actualizada gracias al uso de StateFlow y ViewModels.

6.1.2 Requisitos no funcionales.

1. **Usabilidad.**
   - La interfaz debe ser clara, sencilla y coherente con Material Design 3.
   - El tamaño del texto y la distribución se adaptan a diferentes tamaños de pantalla Android.
2. **Rendimiento.**
   - Las operaciones más frecuentes (carga de partidos, usuarios, solicitudes…) deben ejecutarse sin bloqueos, utilizando corrutinas y flujos de datos.
   - Se evita cargar repetidamente los mismos usuarios mediante cachés locales en memoria.
3. **Seguridad**
   - Las credenciales de acceso se gestionan exclusivamente mediante Firebase Authentication.
   - Los datos de usuario se almacenan en Firestore con permisos regulados por reglas de seguridad.
   - La aplicación nunca guarda contraseñas ni información sensible en local.
4. **Persistencia de datos.**
   - Firestore almacena usuarios, amistades, partidos y estadísticas.
   - DataStore se usa para preferencias locales (sesión iniciada y tema).
   - Las imágenes de perfil se alojan en ImgBB y solo se guarda la URL.
5. **Compatibilidad.**

- La aplicación está diseñada para funcionar en dispositivos Android modernos utilizando Android Studio y Kotlin.

- No es compatible con iOS debido al tipo de tecnología utilizada.

6. **Mantenibilidad.**
   - Se sigue una estructura clara basada en MVVM.

   - El código está comentado y separado en capas: vistas, viewmodels y repositorios.

   - Se han evitado dependencias innecesarias para facilitar ampliaciones futuras.

6.2 Análisis del sistema.

El análisis del sistema explica cómo está organizada internamente la aplicación MiPadel y cómo se relacionan sus diferentes partes para que funcione correctamente. Antes de entrar en el diseño y la implementación, este apartado sirve para entender la estructura general del proyecto.

La aplicación sigue un funcionamiento típico de cliente–servidor: el móvil ejecuta la lógica principal y la interfaz, mientras que Firebase se encarga de tareas externas como la autenticación de usuarios y el almacenamiento de datos en la nube.

6.2.1 Componentes principales del sistema.

- **Interfaz de usuario:** desarrollada con Jetpack Compose, es la capa visible con la que interactúa el usuario.
  Cada pantalla (login, registro, home, perfil, comunidad, estadísticas…) está diseñada como un composable independiente, y todas ellas se conectan mediante el sistema de navegación de Android.
  La UI no accede directamente a la base de datos; en su lugar consulta o modifica datos a través de la capa de ViewModels.

- **Capa lógica de presentación (Viewmodels):** cada pantalla tiene asociado un viewmodel que gestiona el estado de la interfaz utilizando StateFlow, ejecuta operaciones en segundo plano mediante corrutinas, solicita

información a los repositorios y la transforma para que sea fácil de mostrar.

- **Capa de acceso a datos (Repositorios):** contienen las funciones que interactúan con Firebase y ImgBB, son los responsables de registrar y autenticar usuarios, leer y actualizar datos en Firestore, subir imágenes a ImgBB y obtener la url, resolver estados de amistad y filtrar búsquedas y controlar la finalización de partidos y actualización de estadísticas.
  Estos devuelven resultados encapsulados con Result<T>, permitiendo gestionar errores de forma clara.

- **Servicios externos:** el sistema depende de Firebase e ImgBB.

6.2.2 Flujos principales del sistema.

- Inicio de sesión y carga del usuario:
  1. El usuario introduce sus credenciales.
  2. Firebase Authentication valida los datos.
  3. Si son correctos, la aplicación descarga su documento de Firestore.
  4. La información del usuario se almacena durante la sesión en CurrentUserManager.

- Gestión de partidos:
  1. Un usuario crea un partido indicando ubicación, fecha y nivel.
  2. El documento se guarda en Firestore con cuatro posiciones vacías.
  3. Otros usuarios pueden ocupar una posición.
  4. Cuando el partido se inicia, el creador introduce resultados.
  5. Al finalizar se calcula al ganador, el partido pasa a la colección partidos_finalizados y se actualizan las estadísticas de los usuarios que participaron en él.

- Sistema de amistad:
  1. Un usuario busca otros usuarios por nombre de usuario.
  2. Puede enviar solicitudes de amistad (evitando duplicados).
  3. El receptor puede aceptarla o rechazarla.

4. Si se acepta, ambos añaden su UID a la lista de amigos.

- Estadísticas:
    1. Se consultan los partidos finalizados del usuario ordenados por fecha.
    2. Se calcula el número de victorias, derrotas y porcentaje de victorias.
    3. Se muestra un historial visual con los jugadores, fecha y resultado de cada partido.

6.2.3 Relación entre los componentes.

- **La UI** muestra estados y envía acciones.
- **Los ViewModels** actúan como intermediarios y controlan la lógica.
- **Los Repositorios** hacen las operaciones reales contra Firebase/ImgBB.
- **Firestore** es la fuente de verdad de los datos persistentes.

Esta separación asegura que la aplicación sea **escalable, mantenible y fácil de extender** en futuras versiones.

6.3 Diseño del sistema:

6.3.1 Diseño de la Base de Datos

La aplicación utiliza Firebase Firestore como base de datos NoSQL. Esto significa que no existe una estructura fija de tablas como en un sistema relacional, sino que la información se organiza en colecciones formadas por documentos que guardan datos en formato clave–valor.

En MiPadel se han definido 4 colecciones principales:

Usuarios.

Cada documento representa un usuario registrado en la aplicación. Esta colección permite mostrar perfiles, cargar amigos, estadísticas y fotos de usuario.

uid: identificador único del usuario (Firebase Auth).

nombre: nombre y apellidos.

email.

username: nombre visible dentro de la aplicación.

nivel: nivel aproximado del jugador (asignado inicialmente).

amigos: lista de UID de usuarios aceptados como amigos.

fotoPerfilUrl: enlace a ImgBB si el usuario ha subido una imagen.

partidosJugados, partidosGanados, partidosPerdidos

fechaRegistro: fecha de alta.

Amistades.

Gestiona el sistema de solicitudes de amistad entre usuarios. El diseño permite evitar duplicados, reenviar solicitudes y gestionar correctamente todas las transiciones de estado.

docId: ID del documento.

user1 / user2: usuarios implicados.

estado: pendiente, aceptado, rechazado o eliminado.

enviadoPor: UID del usuario que envió la solicitud.

Partidos.

Almacena exclusivamente los partidos activos, es decir, los que tienen estado pendiente, listo o en juego.

id: identificador del partido.

creadorId.

ubicacion.

nivel (nivel aproximado del partido).

fecha.

posiciones: lista de 4 strings, cada posición contiene un UID o una cadena vacía.

maxJugadores.

estado: pendiente / listo / jugando.

sets: solo se rellena cuando está finalizado.

Partidos_Finalizados.

Cuando un partido termina, se elimina de partidos y se mueve a partidos_finalizados.
Esto se hace por evitar que la colección partidos crezca indefinidamente, ralentizando consultas y por tener un historial separado, más fácil de consultar desde la pantalla de estadísticas.

Esta colección se consulta en la pantalla de Estadísticas.

id

creadorId

ubicacion

fecha

posiciones

sets: lista de resultados de sets.

ganador: 1 o 2

Aunque Firestore es NOSQL, se puede describir las relaciones:

- Usuario crea Partido.
- Usuario – Usuario, relación gestionada mediante Amistad.
- Partido – Usuario, mediante UID.
- PartidoFinalizado, conserva información de Partido.

6.3.2 Diseño de la Interfaz de usuario.

La interfaz de MiPadel ha sido diseñada siguiendo los principios de Material Design 3, adaptados a Jetpack Compose. El enfoque principal ha sido conseguir una experiencia visual limpia, moderna y fácil de usar, especialmente porque la aplicación está orientada a usuarios casuales que buscan rapidez y simplicidad al organizar partidos.

Para ello, antes de implementar las pantallas se creó un pequeño prototipo visual en Figma, que sirvió como guía para mantener coherencia en colores, iconografía, márgenes y jerarquía visual.

La navegación de la aplicación se estructura mediante:

- NavHost + Navigation Compose.

- MainScaffold, componente base que integra Appbar dinámico, Drawer lateral, FAB button y bottomsheet para crear partidos, snackbar global.

6.3.2.1 Pantallas de la aplicación.

1. Inicio de sesión.

Permite acceder con correo y contraseña.

Incluye checkbox “mantener sesión iniciada”.

Informa de errores mediante snackbar.

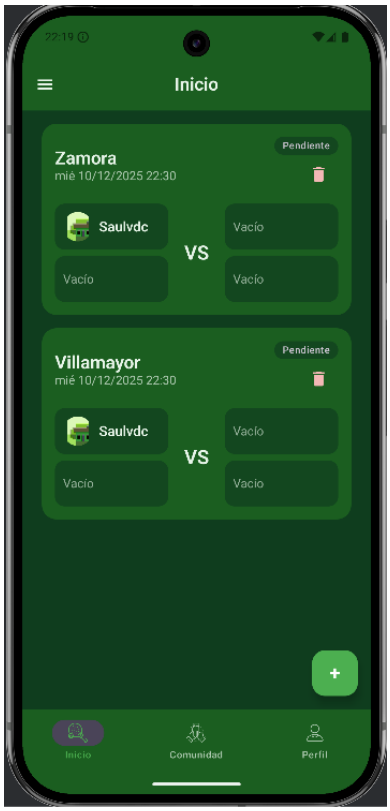
Muestra overlay de carga mientras se valida al usuario.

2. Registro.

Permite crear una cuenta con nombre de usuario, nombre completo, correo electrónico y contraseña.

Muestra información ante errores o registro correcto.

3. Home.



Pantalla principal.

Muestra todos los partidos activos ordenados por fecha. Puedes unirte haciendo clic en los espacios vacíos y salirte del partido volviendo a hacer clic a tu posición.

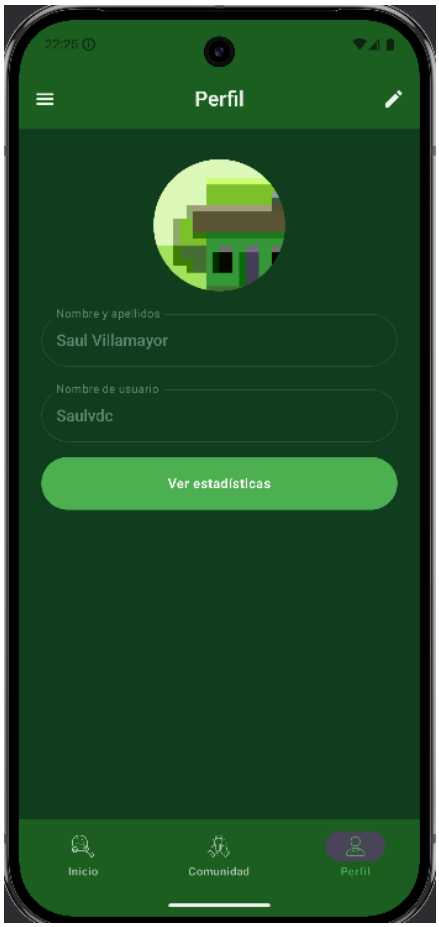
El botón fab abre el bottomsheet para crear el partido.

4. Perfil.

Muestra nombre y apellidos, nombre de usuario y foto de perfil, pudiendo editar los campos mencionados.

Botón Ver estadísticas que te lleva a pantalla estadísticas.

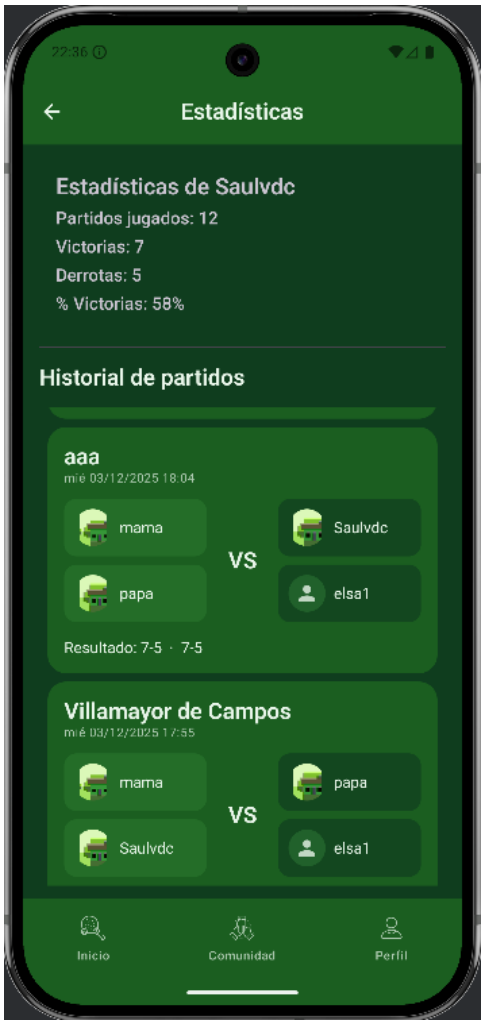
5. Comunidad.



Permite buscar usuarios por nombre de usuario.

Gestiona solicitudes de amistad: enviar, aceptar, rechazar y eliminar.

Muestra lista de amigos.



Muestra estadísticas personales.

Incluye historial de partidos finalizados, mostrando el resultado y en verde al equipo ganador.

6.3.3 Diseño de la Aplicación.

La aplicación se ha desarrollado siguiendo un diseño modular, orientado a la claridad y la mantenibilidad. Aunque no se utiliza un patrón MVVM rígido al 100%, se aplica una estructura muy similar:

- **ViewModel**: Lógica y estado.
- **Repository**: Acceso a Firebase e ImgBB.
- **UI Compose**: Interfaz declarativa conectada con estados StateFlow.

6.3.3.1 Arquitectura general.

1. Capa de datos:

Contiene todas las funciones que acceden a Firebase, su responsabilidad es leer y escribir datos en Firebase, subir imágenes a ImgBB y mantener un usuario en memoria:

- CommunityRepository.
- HomeRepository.
- PartidoFinalizadoRepository.
- UsuarioRepository.
- CurrentUserManager.
- FirebaseAuthmanager.
- FirebaseFirestoremanager.
- ImageUploader.
- UserPreferencesDataStore.

2. Capa de lógica.

Cada pantalla tiene su viewmodel, estos como ya se mencionó antes exponen estados con StateFlow, reciben eventos desde las pantallas, procesan reglas de negocio (validaciones, cargas de historial, control de amistad, cambio de estados en un partido, etc.) y mantienen la pantalla reactiva:

- CommunityViewmodel.
- CrearPartidoViewModel.
- EstadisticasViewModel.
- HomeViewModel.
- LoginViewModel.
- ProfileViewModel.
- RegisterViewModel.
- SettingsViewModel.

3. Capa de presentación.

Implementada completamente con jetpack compose. No se usan xml, la interfaz es reactiva, cada pantalla es uno o varios composables con estados derivados del viewmodel y el componente estructural que garantiza la coherencia visual es el Scaffold.

6.4 Implementación:

6.4.1 Entorno de desarrollo.

El desarrollo de la aplicación MiPadel se ha realizado sobre un entorno centrado en Android y Kotlin, utilizando herramientas actuales:

En primer lugar, he utilizado Android Studio como entorno de desarrollo principal. Desde aquí se ha gestionado todo el proyecto: creación de módulos, configuración de Gradle, ejecución en emuladores y dispositivos físicos, depuración y generación de builds de la aplicación. Android Studio también ha facilitado el trabajo con vistas en tiempo real de Jetpack Compose y con el sistema de navegación.

El lenguaje de programación utilizado ha sido Kotlin, que es el lenguaje recomendado por Google para el desarrollo de aplicaciones Android. Toda la lógica de la aplicación (viewmodels, repositorios, modelos de datos, etc.) está escrita en Kotlin, aprovechando características como las corrutinas, las funciones de extensión y las data classes.

Para la capa de interfaz gráfica se ha utilizado Jetpack Compose, el toolkit declarativo de Android. Gracias a Compose, las pantallas se han construido como funciones @Composable, usando estados observables (StateFlow, collectAsState) y componentes de Material 3 para mantener un diseño coherente: botones, tarjetas, barras de navegación, diálogos, campos de texto, etc.

La gestión de dependencias y la compilación del proyecto se ha realizado con Gradle, a través de los ficheros habituales build.gradle y el catálogo de versiones (libs.versions.toml). Desde ahí se han añadido las librerías necesarias.

Para el control de versiones se ha utilizado Git, con un repositorio remoto en GitHub, lo que ha permitido mantener un historial de cambios, subir copias de seguridad del proyecto y poder recuperar versiones anteriores en caso de problemas.

Las pruebas se han realizado tanto en emuladores de Android Studio como en un dispositivo físico Android, comprobando el funcionamiento de la aplicación en distintos tamaños de pantalla y en diferentes temas (modo claro y oscuro).

6.4.2 Estructura del código.

El proyecto se organiza en una estructura clara y modular que facilita el mantenimiento, la lectura del código y la ampliación futura de la aplicación.

Librerías utilizadas:

- **Jetpack / Android:**
  - core-ktx: funciones extendidas de Kotlin para Android.
  - lifecycle-runtime-ktx, viewmodel-ktx: corrutinas y ciclo de vida en ViewModels.
  - activity-compose: actividad principal integrada con Compose.
- **Jetpack Compose (UI):**
  - compose-ui, ui-tooling, foundation, runtime.
  - material3 – Componentes visuales modernos.
  - ui-tooling-preview – Para previsualizaciones en Android Studio.

- **Navegación:**
  - navigation-compose – Gestión declarativa de pantallas.
  - navigation-runtime-ktx – Integración con corrutinas y navegación reactiva.
- **Firebase:**
  - firebase-auth – Login, registro y gestión de sesiones.
  - firebase-firestore – Base de datos NoSQL en la nube.
  - firebase-crashlytics – Registro de errores en producción.
- **Persistencia:**
  - DataStore Preferences – Almacena ajustes como "mantener sesión iniciada".
- **Imágenes y networking:**
  - coil-compose – Carga de imágenes desde URL (fotos de perfil de ImgBB).
  - okhttp + logging-interceptor – Llamadas HTTP para subir imágenes a ImgBB.

Cabe destacar que la aplicación combina varios enfoques:

- **Orientación a objetos** en modelos y repositorios.
- **Programación declarativa** para la interfaz (Jetpack Compose).
- **Programación reactiva** mediante StateFlow en ViewModels.
- **Arquitectura modular similar a MVVM**, aunque aplicada con flexibilidad para adaptarse al tipo de proyecto.

6.4.3 Cuestiones de diseño e implementación reseñables.

1. Gestión de imágenes de perfil sin FirebaseStorage.

Uno de los puntos más complejos fue implementar el sistema de subida y visualización de imágenes de perfil sin utilizar Firebase Storage, ya que sus funcionalidades avanzadas requieren el plan de pago. Para resolver esta limitación se diseñó un flujo alternativo apoyado en ImgBB como servicio externo de almacenamiento de imágenes.

Cuando el usuario selecciona o hace una foto de perfil, la aplicación convierte la imagen a un Bitmap, la comprime en formato JPEG y la envía a ImgBB mediante una petición HTTP multipart/form-data utilizando OkHttp. El servidor devuelve una URL pública de la imagen, y esa URL es la que se guarda en la colección de usuarios de Firestore.

De esta forma la aplicación no almacena directamente la imagen, sino solo el enlace devuelto por ImgBB, que luego se utiliza en la interfaz con Coil para mostrar la foto de perfil en cualquier pantalla. Esta solución permite tener imágenes de usuario sin depender de Firebase Storage ni de infraestructura propia.

2. Actualización de estados en pantalla mediante StateFlow.

Toda la aplicación se ha implementado con un enfoque reactivo. Uno de los retos fue conectar repositorios, ViewModels y pantallas de Compose para que:

- Las listas de partidos se actualicen en tiempo real al unirse o salir un jugador.
- El perfil muestre la foto nueva inmediatamente tras subirla.
- El historial de partidos finalizados se actualice automáticamente.

Para ello se emplearon **StateFlow** y **MutableStateFlow**, gestionados desde los ViewModels. Este patrón redujo errores y permitió una UI totalmente reactiva, pero supuso un aprendizaje continuo al no formar parte del temario del ciclo.

3. Control del estado de los partidos sin Cloud Functions.

Firestore ofrece Cloud Functions (pequeñas funciones de backend que se ejecutan en los servidores de Google cuando ocurre un evento, sin que tú tengas que montar ni mantener un servidor propio) para automatizar tareas, pero también forman parte del plan de pago.

Para suplir su ausencia, se implementó una revisión periódica del estado del partido desde la propia aplicación:

- Cada cierto tiempo se comprueba si el partido ha pasado la hora límite.
- Si es así, su estado cambia de *pendiente* a *jugando* o *finalizado*, según corresponda.

Este mecanismo garantiza un comportamiento coherente incluso sin automatizaciones externas.

6.5 Pruebas.

La validación del funcionamiento de la aplicación se ha realizado mediante pruebas manuales, centradas en comprobar que cada funcionalidad cumplía correctamente los requisitos definidos. Dado el alcance del proyecto opté por un enfoque de pruebas progresivo mientras desarrollaba las distintas pantallas y ViewModel.

- **Pruebas funcionales de la interfaz**: navegación entre pantallas, correcta visualización de la información y actualización reactiva de los datos procedentes de Firestore.
- **Pruebas de autenticación**: creación de usuarios, inicio de sesión, cierre de sesión y persistencia del estado cuando el usuario marca la opción “Mantener sesión iniciada”.

- **Pruebas de gestión de partidos**: creación de partidos, unión y salida de jugadores, cambios de estado (“pendiente”, “listo”, “jugando” y “finalizado”) y validación de la lógica asociada.

- **Pruebas de comunidad**: envío, recepción, aceptación y rechazo de solicitudes de amistad.

- **Pruebas de perfil**: subida de imágenes mediante ImgBB, edición de datos personales y verificación de sincronización con Firestore.

Además, se han corregido múltiples errores detectados durante el desarrollo (comportamientos inesperados, datos no sincronizados, validaciones incorrectas, etc.) y se ha realizado una revisión final completa del proyecto, optimizando lógica, estado, repositorios y pantallas.

7 Manuales de usuario.

Este manual describe el funcionamiento general de la aplicación MiPadel, explicando el uso de cada pantalla, los elementos interactivos y el flujo principal dentro de la aplicación.

7.1 Manual de usuario.

7.1.1 Inicio de sesión.

Es la primera pantalla que ve el usuario, permite acceder a la aplicación introduciendo su correo y la contraseña. Además, pulsando el checkbox de “Mantener sesión iniciada” podrá volver a la aplicación sin tener que meter otra vez sus credenciales.

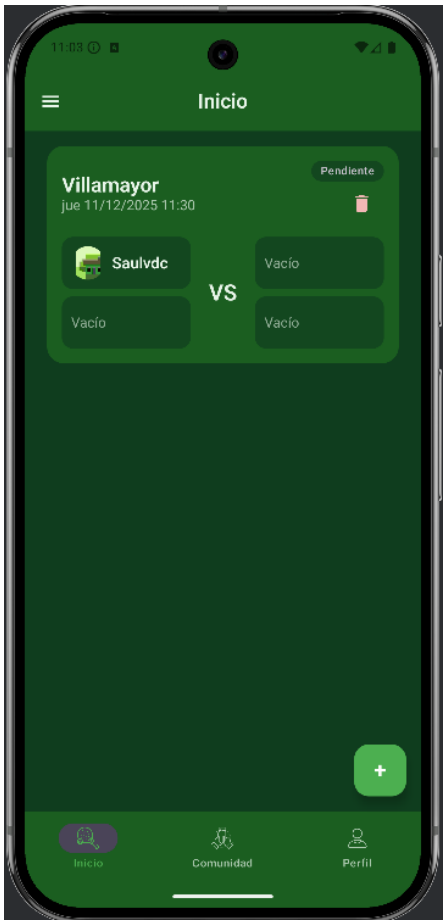
Dependiendo de si el login es correcto o no, saldrán mensajes en el snackbar personalizado.

7.1.2 Registro.

Permite crear una cuenta nueva rellenando los textields que aparecen.

Dependiendo de que los campos estén rellenados y no exista ese nombre de usuario podrás crear la cuenta, además aparecen de éxito o error en el snackbar.

7.1.3 Inicio.



Es la pantalla principal tras iniciar sesión. Muestra todos los partidos creados y activos, los jugadores apuntados en ellos y las acciones disponibles.

Para apuntarse al partido el usuario deberá hacer clic en los espacios vacíos y para desapuntarse tendrá que hacer clic en su posición.

Solo el creador del partido puede eliminarlo haciendo clic al icono de arriba a la derecha del componente.

Para crear un partido el usuario deberá pulsar el Fab button y rellenar en el Bottomsheet la ubicación, fecha y hora. Por último dar al botón crear partido.

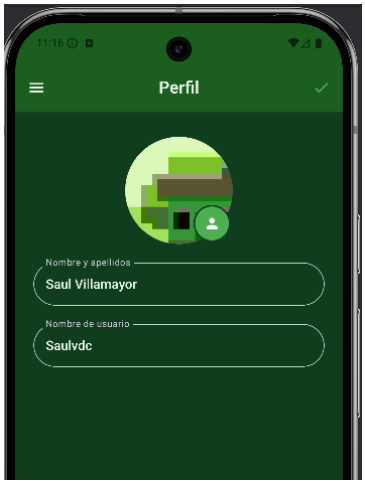
7.1.4 Comunidad.

Gestiona amigos y solicitudes de amistad.

Puedes buscar usuarios por su nombre de usuario, enviar solicitudes de amistad, rechazar o aceptar las que te han enviado y eliminar amigos.

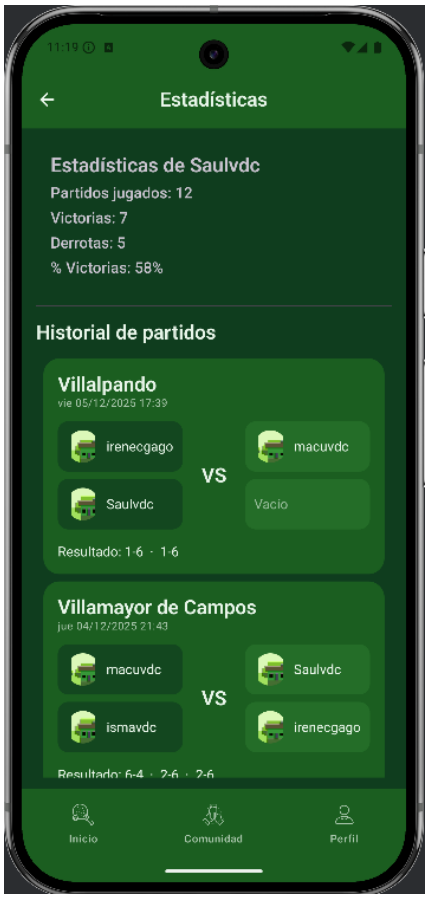
7.1.5 Perfil.

Permite ver y editar los datos del usuario. Para ello se deberá pulsar el lápiz que aparece en el appbar, al pulsar se desbloquearán los textfields para poder editarlos y se podrá cambiar la imagen de perfil.



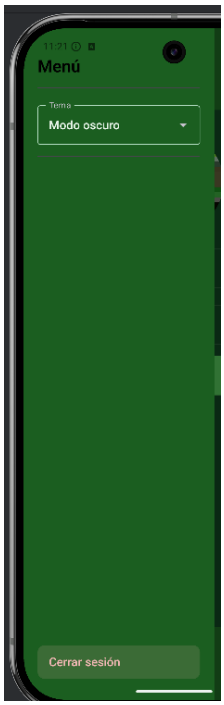
El icono del lápiz desaparece y aparece un tick para que cuando el usuario haya hecho las modificaciones que quiera, pulsar este nuevo icono y se actualizará el usuario.

7.1.6 Estadísticas.



Muestra el rendimiento del usuario y su historial de partidos finalizados

7.1.7 Navegación.



Para la navegación general he implementado un BottomNavigationBar para desplazarte por las pantallas principales y un drawer para cambiar el tema de la aplicación y cerrar sesión.

8 Conclusiones y posibles ampliaciones.

El desarrollo de MiPadel ha permitido comprobar que la idea inicial (facilitar la organización de partidos y mejorar la comunicación entre jugadores), puede resolverse de forma práctica mediante una aplicación móvil. A nivel técnico, el proyecto ha servido para consolidar conocimientos del ciclo y para incorporar tecnologías nuevas que no formaban parte del temario, lo que ha supuesto un aprendizaje útil y aplicable a futuros desarrollos.

Aunque la aplicación funciona correctamente, existen varias ampliaciones que podrían implementarse en un futuro para aumentar su utilidad y profesionalizarla:

- Mensajería interna entre jugadores y creación de partidos privados desde la pantalla comunidad.
- Sistema real de reservas de calendario.
- Notifiaciones push.
- Utilizar el nivel del usuario para sugerir partidos adecuados.
- Estadísticas más avanzadas.
- Utilizar las funciones de Firebase Storage y Cloud Functions.
- Hacer una pantalla específica para el usuario, pudiendo cambiar su correo, contraseña…

9 Bibliografía.

https://developer.android.com/?hl=es-419

https://developer.android.com/compose

https://kotlinlang.org/docs/home.html

https://firebase.google.com/docs?hl=es-419

https://imgbb.com/

https://square.github.io/okhttp/

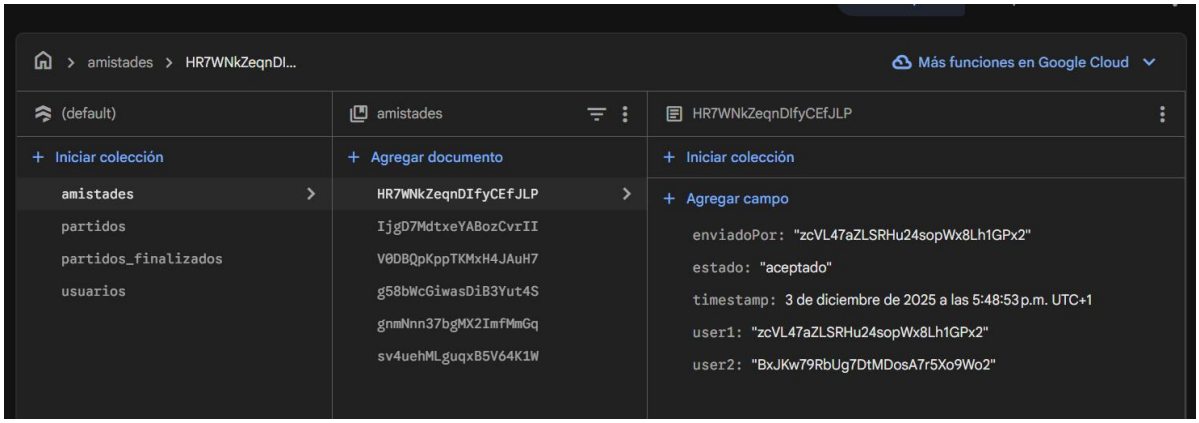
https://coil-kt.github.io/coil/compose/

Herramientas de IA para apoyo en redacción y revisión.

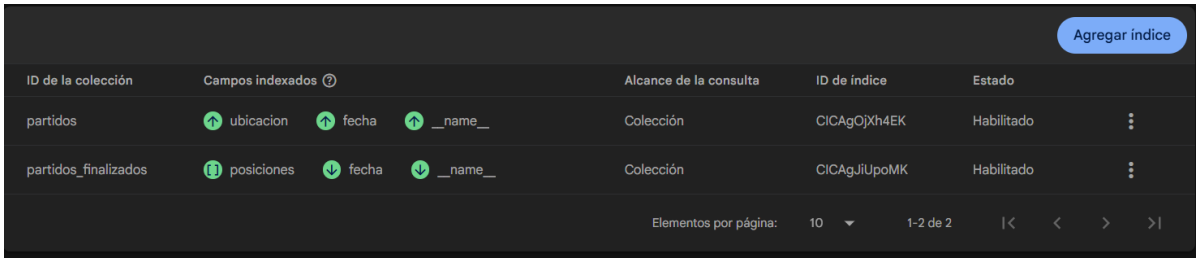
10 Anexos.

10.1 Firestore Database.

Imágenes de como se guardan los datos en Firebase:



Índices de firebase (aceleran las consultas y permiten consultas complejas).



10.2 ImageUploader.

```kotlin
/**
 * Subir imágenes a ImgBB usando una petición HTTP multipart.
 */
2 Usages
object ImageUploader {

    1 Usage
    private const val API_KEY = "c9879b31ac20e1e16727cc68a4c93db3"
    3 Usages
    private const val TAG = "ImageUploader"

    /**
     * Sube una imagen y devuelve la URL pública o null si falla.
     */
    1 Usage
    suspend fun uploadImage(context: Context, imageUri: Uri): String? =
        withContext( context = Dispatchers.IO) {
            try {
                val resolver = context.contentResolver
                val inputStream: InputStream = resolver.openInputStream(imageUri)
                    ?: return@withContext null

                // Convertir a Bitmap para poder comprimirla
                val bitmap = BitmapFactory.decodeStream( is = inputStream)
                inputStream.close()

                val compressedStream = ByteArrayOutputStream()
                bitmap.compress( format = Bitmap.CompressFormat.JPEG, quality = 80, compressedStream)
                val bytes = compressedStream.toByteArray()

                val client = OkHttpClient()
                val requestBody = MultipartBody.Builder()
                    .setType(MultipartBody.FORM)
                    .addFormDataPart( name = "key", value = API_KEY)
                    .addFormDataPart(
                        name = "image",
                        filename = "profile.jpg",
                        body = bytes.toRequestBody( contentType = "image/*".toMediaTypeOrNull())
                    )
                    .build()

                val request = Request.Builder()
                    .url( url = "https://api.imgbb.com/1/upload")
                    .post(requestBody)
                    .build()

                val response = client.newCall(request).execute()
                val bodyString = response.body?.string()

                if (!response.isSuccessful || bodyString == null) {
                    Log.e( tag = TAG, msg = "HTTP error: ${response.code}")
                    return@withContext null
                }

                val json = JSONObject( json = bodyString)

                if (json.optBoolean( name = "success")) {
                    val url = json.getJSONObject( name = "data").getString( name = "url")
                    return@withContext url
                }

            Log.e( tag = TAG, msg = "Error del servidor ImgBB ${json.optString( name = "error")}")
            return@withContext null

        } catch (e: Exception) {
            Log.e( tag = TAG, msg = "Error subiendo imagen: ${e.message}")
            return@withContext null
        }
}
```