

**GAMEGLISH**

**Trabajo de Fin de Grado: Desarrollo de Aplicaciones Multiplataforma**

Autor: Javier Huélamo Gracia

Tutor: Omar El Founti Kshim



**ÍNDICE**

Contenido

[**1. RESUMEN** 6](#_heading=h.e0qvl12mg3kq)

[**1.1 Abstract(English)** 6](#_heading=h.beg7hu7f4wxg)

[**1.2 Resumen(Español)** 7](#_heading=h.l7dcc5r0fcwd)

[**2. PALABRAS CLAVE** 8](#_heading=h.174xr0xsxmq2)

[**3. DEFINICIONES, ABREVIATURAS Y ACRÓNIMOS** 9](#_heading=h.sdattycujdki)

[**4. AUTENTICIDAD DE LA INFORMACIÓN** 11](#_heading=h.x38c4ji527rr)

[**4.1 Declaración de autenticidad** 11](#_heading=h.5m18x71x7ezh)

[**4.2 Origen del contenido** 11](#_heading=h.q1hhevlf7tqr)

[**4.3 Proceso de desarrollo** 11](#_heading=h.rxkhray1pxa4)

[**4.4 Verificación de la autenticidad** 11](#_heading=h.cx39jsh04i6e)

[**4.5 Conclusión** 11](#_heading=h.xh122vpca7sq)

[**5. MOTIVACIÓN** 12](#_heading=h.mr4g109lf8db)

[**5.1 Motivación personal** 12](#_heading=h.1marfxlljir)

[**5.2 Motivación profesional** 12](#_heading=h.omcg20rur2mo)

[**6. OBJETIVOS** 13](#_heading=h.xf9nep469orx)

[**6.1 Objetivo General** 13](#_heading=h.hge4ieq5h3nj)

[**6.2 Objetivos Específicos** 13](#_heading=h.jvbnfg6ophbl)

[**7.ESTADO DEL ARTE** 14](#_heading=h.nxjmo8l2dz71)

[**7.1 Duolingo** 14](#_heading=h.vwtb4kdjnzw7)

[**7.2 Babbel** 14](#_heading=h.mowf5uj8qsdb)

[**7.3 Justificación del enfoque** 14](#_heading=h.7s8jsdr6h8s3)

[**8. PROPUESTA** 15](#_heading=h.8u4ewerurggk)

[**8.1 Identificación del espacio de conocimiento y tecnológico** 15](#_heading=h.98rfybkofstz)

[**8.2 Diferenciación y contribución** 15](#_heading=h.h63or7i10bx2)

[**8.3 Aportación innovadora** 15](#_heading=h.nesnjkz179yt)

[**8.4 Prueba de concepto** 15](#_heading=h.uso1cpocez03)

[**9. ANÁLISIS DEL PROBLEMA** 17](#_heading=h.ghyfmkvtpqpw)

[**9.1 Especificación de requisitos** 17](#_heading=h.43x0bx2c04z8)

[**9.1.1 Requisitos funcionales** 17](#_heading=h.bxc9lklf44i)

[**9.1.2 Requisitos no funcionales** 17](#_heading=h.beucq8dcwgvl)

[**9.1.3 Funcionalidades extra** 18](#_heading=h.qxt8nfyuz27v)

[**9.2 Modelado conceptual** 18](#_heading=h.ic5iyuqobyzo)

[**10. ANÁLISIS DE LA SEGURIDAD** 24](#_heading=h.r34hxjx5ervo)

[**10.1 Amenazas identificadas** 24](#_heading=h.ogajujuic5bk)

[**10.2 Controles Implementados** 25](#_heading=h.w0v25lszjk02)

[**11. ANÁLISIS DEL MARCO LEGAL Y ÉTICO** 26](#_heading=h.6539ex3zc9q3)

[**11.1 Protección de datos (GDPR/LOPD-GDD)** 26](#_heading=h.hi2iloml6u66)

[**11.2 Propiedad intelectual y licencias** 26](#_heading=h.33t4bfcshpoe)

[**11.3 Uso de contenido multimedia de terceros** 27](#_heading=h.6pz1evqqr7qv)

[**12. ANÁLISIS DE RIESGOS** 28](#_heading=h.9r6cjp8jzsy8)

[**12.1 Riesgos técnicos** 28](#_heading=h.k3e1j77urzc4)

[**12.2 Riesgos de aceptación del usuario** 28](#_heading=h.p50omek33g76)

[**13. IDENTIFICACIÓN Y ANÁLISIS DE SOLUCIONES POSIBLES** 29](#_heading=h.xcaylu1p75fa)

[**13.1 Soluciones y alternativas analizadas** 29](#_heading=h.x978ew9unpkb)

[**13.2 Criterios de selección** 30](#_heading=h.qo0nszwrnn3y)

[**14. SOLUCIÓN PROPUESTA** 31](#_heading=h.8e26gh8ot9gg)

[**14.1 Introducción** 31](#_heading=h.ekc6pxct7s69)

[**14.2 Descripción de la solución** 31](#_heading=h.jccrdznt946u)

[**14.3 Fases de desarrollo** 31](#_heading=h.kx0s0r7tjksr)

[**14.4 Implantación y validación** 32](#_heading=h.4lvejy3qcmso)

[**15. PLAN DE TRABAJO** 34](#_heading=h.hoxolr9qrjf6)

[**16. DISEÑO DE LA SOLUCIÓN** 35](#_heading=h.wtyia3cofv2a)

[**16.1 Arquitectura del sistema (MVVM)** 35](#_heading=h.ocbx1y95z6tj)

[**16.2 Modelo de datos** 36](#_heading=h.l9s9amrilgh8)

[**16.3 Interfaz de usuario** 36](#_heading=h.sle3znh42qof)

[**16.4 Gestión de estados** 37](#_heading=h.840v4jkumrtj)

[**16.5 Navegación y flujos de pantalla** 37](#_heading=h.7r6t0kffgdct)

[**17. TECNOLOGÍA UTILIZADA** 39](#_heading=h.1kurug7qjkej)

[**17.1 Entorno de desarrollo** 39](#_heading=h.4yaoxmgniqlb)

[**17.2 Lenguaje de desarrollo** 39](#_heading=h.qf3tlsecbstx)

[**17.3 Sistema operativo** 39](#_heading=h.adbwqw95xfh)

[**17.4 Frameworks y bibliotecas** 39](#_heading=h.qougwlb5b9sc)

[**17.5 APIs y permisos** 40](#_heading=h.v9sb9tw37lmv)

[**17.6 Dependencias adicionales** 40](#_heading=h.wu12yictf1pl)

[**17.7 Conocimiento personal y viabilidad** 40](#_heading=h.n1jojvwck0vi)

[**18. DESARROLLO DE LA SOLUCIÓN PROPUESTA** 41](#_heading=h.89ka16gtst39)

[**19. IMPLANTACIÓN** 42](#_heading=h.7z5iiiccyjq9)

[**20. PRUEBAS** 43](#_heading=h.nil30v8ept21)

[**20.1 Pruebas de funcionalidad:** 43](#_heading=h.ni01vwkrqzf5)

[**20.2 Pruebas de accesibilidad** 60](#_heading=h.rhmemmg33upm)

[**21. CONCLUSIONES** 62](#_heading=h.ug331kax4r4i)

[**21.1 Cumplimiento de los objetivos** 62](#_heading=h.mhkkxm1sjtut)

[**21.2 Limitaciones y trabajo futuro** 62](#_heading=h.o0q5rhuytyyl)

[**22. RELACIÓN CON LOS ESTUDIOS CURSADOS** 63](#_heading=h.lb8zq7xg6hdl)

[**23. REFERENCIAS** 64](#_heading=h.behso4qwdylj)

[**24. ANEXOS** 66](#_heading=h.b5yu2fgiplo2)

[**24.1 Estructura de Directorios** 66](#_heading=h.gbpjy1j3eyuo)

[**24.1.1 Módulos raíz** 66](#_heading=h.anu0spym8l23)

[**24.1.2 Módulo data** 66](#_heading=h.xmiajx2072w4)

[**24.1.3 Módulo UI** 67](#_heading=h.ewfc9efdlz40)

[**24.2 Código Fuente (extractos)** 67](#_heading=h.lrswcomj5n1t)

[**24.2.1 Modelo (Ejemplo EntityPreguntas) Entidad “Preguntas”** 67](#_heading=h.pbtx8sq45gey)

[**24.2.2 DAO (Ejemplo DaoPregunta) Filtro de preguntas** 67](#_heading=h.gfldf9scbijb)

[**24.2.3 Repositorio (RepositoryUsuario). Registro** 68](#_heading=h.9i834goi5u0p)

[**24.2.4 ViewModel (CompetitiveGameViewModel):** 69](#_heading=h.2w9eiixjnqo6)

[**24.2.5 Navegación (AppNavHost)** 71](#_heading=h.rlyx5k5gftf)

[**24.2.6 Componente UI (LivesRow)** 72](#_heading=h.83rw4sqkj5cq)

[**24.3 Manual de usuario** 73](#_heading=h.1b699db60wfb)

[**25. GLOSARIO** 76](#_heading=h.b1pt1otqkw9d)

**TABLA DE CONTENIDOS**

[Ilustración 1 Diagrama de Casos de uso 18](#_heading=h.hqo8oc3668hi)

[Ilustración 2 Diagrama de Flujo 19](#_heading=h.7mizwh7urqp5)

[Ilustración 3 Diagrama de secuencia Login 20](#_heading=h.ykckz42zungf)

[Ilustración 4 Diagrama de secuencia Modo Individual 20](#_heading=h.2nj1lhaho5y1)

[Ilustración 5 Diagrama de secuencia Modo Competitivo 21](#_heading=h.s5ne74ok7l6a)

[Ilustración 6 Diagrama de secuencia Ranking 21](#_heading=h.ldap03z11rr)

[Ilustración 7 Diagrama de Gantt 32](#_heading=h.jb8j3n9krazr)

[Ilustración 8 Módulo Raíz 47](#_heading=h.jakwmpib5doq)

[Ilustración 9 Módulo Data 47](#_heading=h.bnv6knlvw7yk)

[Ilustración 10 Módulo UI 48](#_heading=h.48iasj6vw9t)

[Ilustración 11 EntityPregunta 48](#_heading=h.g56hifvxcyu)

[Ilustración 12 DaoPregunta 49](#_heading=h.gnmpk76llsnc)

[Ilustración 13 registrarUsuarioCorreo() 49](#_heading=h.i8ji3vvftdcd)

[Ilustración 14 observeGame() 50](#_heading=h.8g6cl6c4u1t7)

[Ilustración 15 CerrarRonda() 51](#_heading=h.uw9jdfx50dpr)

[Ilustración 16 AppNavHost 53](#_heading=h.2w4cllhxczoo)

[Ilustración 17 LivesRow.kt 53](#_heading=h.o5021o41c8wk)

# **1. RESUMEN**

## 

## **1.1 Abstract(English)**

**GameGlish** is an innovative mobile application designed to make learning English more engaging, effective, and accessible through gamification. By integrating interactive quizzes, adaptive learning, and competitive challenges, the app creates a motivating and immersive environment for users of all proficiency levels.

Upon registration, users take a diagnostic test to determine their English level, allowing GameGlish to adjust the difficulty of exercises and provide a customized learning path tailored to their needs. The app covers key language skills such as vocabulary, grammar, reading comprehension, and listening, offering real-time feedback to help learners improve efficiently.

Users can practice at their own pace in Solo Mode or compete against friends and players worldwide in Competitive Mode, adding an element of friendly competition that enhances motivation and engagement. To encourage regular practice, GameGlish includes leaderboards, and a reward system that keeps learners engaged while tracking their progress.

In conclusion, this project looks forward to transforming traditional language learning into an interactive and entertaining experience. GameGlish helps users develop their English skills while having fun, boosting both confidence and fluency.

## **1.2 Resumen(Español)**

**GameGlish** es una aplicación móvil innovadora diseñada para hacer que el aprendizaje de inglés sea más atractivo, efectivo y accesible mediante la gamificación. Al integrar cuestionarios interactivos, aprendizaje adaptativo y desafíos competitivos, la aplicación crea un entorno motivador e inmersivo para usuarios de todos los niveles de competencia.

Al registrarse, los usuarios realizan una prueba de diagnóstico para determinar su nivel de inglés, lo que permite a GameGlish ajustar la dificultad de los ejercicios y proporcionar una ruta de aprendizaje personalizada. La aplicación abarca habilidades clave como vocabulario, gramática, comprensión lectora y escucha, ofreciendo retroalimentación en tiempo real para ayudar a los estudiantes a mejorar de forma eficiente.

Los usuarios pueden practicar a su propio ritmo en el Modo individual o competir contra amigos y jugadores de todo el mundo en el Modo Competitivo, añadiendo un elemento de competencia amistosa que potencia la motivación y el compromiso. Para fomentar la práctica continua, GameGlish incluye clasificaciones y un sistema de recompensas que mantiene a los estudiantes motivados mientras se registra su progreso.

En conclusión este proyecto busca transformar el aprendizaje tradicional de idiomas en una experiencia interactiva y entretenida. GameGlish ayuda a los usuarios a desarrollar sus habilidades de inglés mientras se divierten, aumentando su confianza y fluidez.

# **2. PALABRAS CLAVE**

Estructura MVVM.

Firebase.

ROOM.

Gradle.

Jetpack Compose.

Kotlin.

Android.

Gamificación.

Aprendizaje de inglés.

Competitivo en tiempo real.

# **3. DEFINICIONES, ABREVIATURAS Y ACRÓNIMOS**

* **API – Application Programming Interface** Conjunto de métodos que permiten que tu app invoque servicios de Android o bibliotecas externas (p. ej., Firebase).
* **BaaS – Backend as a Service** Modelo en la nube que ofrece funciones listas para usar (autenticación, BD en tiempo real…) sin mantener servidores; en GameGlish lo provee Firebase.
* **CI/CD – Continuous Integration / Continuous Deployment** Automatización del build, las pruebas y la publicación de la APK cada vez que se actualiza el repositorio.
* **CRUD – Create, Read, Update, Delete** Operaciones básicas sobre los datos de usuario y de juego que se ejecutan en Firebase y en la caché SQLite.
* **DAO – Data Access Object** Componente de Room que encapsula consultas SQL y devuelve objetos Kotlin.
* **Firebase Auth** Servicio de autenticación de Google utilizado para registro e inicio de sesión seguros.
* **Firebase Realtime Database** Base de datos en tiempo real para sincronizar partidas competitivas y estadísticas de usuario.
* **Gradle** Sistema de construcción que compila, prueba y genera la APK de GameGlish.
* **IDE – Integrated Development Environment** Entorno de desarrollo (Android Studio) donde se escribe y depura la aplicación.
* **JSON – JavaScript Object Notation** Formato ligero utilizado para intercambiar datos con Firebase y para serializar objetos localmente.
* **Jetpack Compose** Framework declarativo de UI de Android empleado para crear las pantallas de la app
* **Kotlin** Lenguaje principal del proyecto, recomendado por Google para Android.
* **MVVM – Model-View-ViewModel** Patrón arquitectónico que separa la lógica de presentación (ViewModel) de la UI (View) y de los datos (Model).
* **POO – Programación orientada a objetos** Paradigma centrado en clases y objetos que estructura la lógica interna de GameGlish.
* **Room** Capa de abstracción sobre SQLite que genera código seguro y eficiente para el almacenamiento local.
* **SQLite** Motor de base de datos embebido usado para preguntas y estadísticas offline.
* **UI / UX – User Interface / User Experience** UI se refiere a la capa visual; UX, a la experiencia global. Ambos guían el diseño gamificado de GameGlish.

# **4. AUTENTICIDAD DE LA INFORMACIÓN**

## 

## **4.1 Declaración de autenticidad**

Afirmo de forma clara que todo lo que aparece en este proyecto es original y resultado de mi propio esfuerzo. No procede, total ni parcialmente, de ninguna fuente externa.

## **4.2 Origen del contenido**

El contenido ha sido creado por mí sin recurrir indebidamente a materiales ni ideas ajenas. Cada concepto, diseño o solución se ha desarrollado desde cero, respetando los principios de honestidad y respeto a la propiedad intelectual.

## **4.3 Proceso de desarrollo**

He invertido el tiempo necesario para investigar, planificar, programar y probar cada función de GameGlish. En ningún momento he copiado ni reutilizado trabajos de terceros.

## **4.4 Verificación de la autenticidad**

Revisé minuciosamente el proyecto para asegurar su originalidad y cité correctamente cualquier recurso externo utilizado. Además, seguí un control de calidad para mantener la coherencia y fiabilidad del contenido.

## **4.5 Conclusión**

Ratifico la autenticidad e integridad de este Trabajo Fin de Grado y asumo la responsabilidad derivada de ello. Invito a quien lo lea a valorar el esfuerzo y la originalidad que refleja.

# **5. MOTIVACIÓN**

## **5.1 Motivación personal**

Desde pequeño he alternado dos aficiones muy distintas que, con el tiempo, han terminado cruzándose: el inglés y jugar a videojuegos. He dado clases de inglés para ganar dinero cuando era joven y también me he dedicado profesionalmente a competir en videojuegos. GameGlish nace justo en ese cruce: quiero que la gente practique inglés sin sentir que “estudia”, sino que juega y compite de forma amistosa. Convertir mi Trabajo Fin de Grado en una app que mezcle aprendizaje y entretenimiento me resulta, por tanto, tan útil como ilusionante: útil porque resuelve un problema cotidiano (la falta de constancia al estudiar idiomas) e ilusionante porque me permite plasmar mis propias ideas sobre cómo motivarnos mediante pequeñas recompensas, marcadores y piques sanos entre amigos.

## **5.2 Motivación profesional**

A nivel profesional, desarrollar este trabajo de fin de grado supone llevar a la práctica todas las fases que se esperan de un futuro desarrollador de aplicaciones: investigar al usuario, definir requisitos claros, diseñar una experiencia atractiva y, por último, convertirla en un producto que pueda instalarse y usarse de verdad. El proyecto me reta a tomar decisiones de arquitectura, a organizar el código para que crezca sin volverse inmanejable y a coordinar pruebas que garanticen una versión estable. En otras palabras, me permite demostrar que sé transformar una idea en un servicio digital completo, con la visión global que demanda la industria: pensar en el usuario, en la calidad del software y en la capacidad de evolucionar con nuevas versiones.

# 

# **6. OBJETIVOS**

## **6.1 Objetivo General**

Crear una aplicación Android que convierta el estudio del inglés en una experiencia de juego con preguntas rápidas, retos individuales y duelos en línea que fomenten la constancia, midan el progreso y mantengan la motivación del usuario.

## **6.2 Objetivos Específicos**

1. **Implementar autenticación segura de usuarios**

Desarrollar un registro / inicio de sesión basado en Firebase Auth que permita crear perfiles personales, almacenar con seguridad el progreso y sincronizar los datos cuando el usuario cambie de dispositivo.

1. **Desarrollar el modo individual de aprendizaje**

Crear un itinerario por niveles con actividades de vocabulario y gramática; cada nivel mostrará el temario correspondiente y un juego de preguntas cuya dificultad se adapte al rendimiento del alumno.

1. **Implementar el modo competitivo en tiempo real**

Diseñar un sistema de emparejamiento rápido para que dos jugadores compitan online en retos de respuesta rápida, enviando y recibiendo resultados sin demoras apreciables.

1. **Diseñar la capa de gamificación y análisis personalizado**

Integrar puntos y tablas de clasificación, así como un módulo de estadísticas que ofrezca recomendaciones de estudio a partir de los aciertos, errores y frecuencia de uso del usuario.

# **7.ESTADO DEL ARTE**

El aprendizaje de idiomas en dispositivos móviles ha experimentado un fuerte crecimiento durante la última década. Decenas de aplicaciones compiten por atraer y retener al usuario ofreciendo micro lecciones, recompensas y distintos niveles de interacción social. A continuación, se resumen las propuestas más representativas y se explica por qué GameGlish adopta un enfoque propio.

## **7.1 Duolingo**

Plataforma líder que basa su éxito en sesiones breves, vidas, rachas diarias y tablas de clasificación semanales. El método es adictivo, pero la práctica real de conversación y la revisión gramatical en profundidad resultan limitadas; la interacción entre usuarios suele ser diferida.

## **7.2 Babbel**

Propone cursos estructurados por lingüistas, con lecciones algo más largas y un énfasis claro en la corrección de pronunciación. Aporta rigor académico, pero sacrifica el componente lúdico y la competición directa, lo que reduce la motivación a largo plazo para ciertos perfiles.

## **7.3 Justificación del enfoque**

GameGlish combina las micro lecciones y recompensas visuales que ya han demostrado ser eficaces con tres rasgos propios: duelos sincrónicos uno contra uno que disparan la competitividad inmediata, un funcionamiento híbrido on/offline para que el aprendizaje continúe sin conexión y un sistema de preguntas personalizadas en base al nivel del usuario.

# 

# **8. PROPUESTA**

## **8.1 Identificación del espacio de conocimiento y tecnológico**

Este TFG se inscribe en el ámbito de la gamificación educativa para dispositivos Android. Convergen varias áreas: programación en Kotlin, diseño UI/UX con Jetpack Compose, patrones de arquitectura (MVVM), sincronización en la nube mediante Firebase Auth + Realtime Database y almacenamiento local con Room/SQLite para uso sin conexión. El objetivo es ofrecer una experiencia de estudio breve, dinámica y accesible a cualquier usuario que disponga de un teléfono Android medianamente moderno.

## **8.2 Diferenciación y contribución**

Aunque existen otras apps de idiomas, GameGlish se distingue por integrar tres rasgos: un duelo síncrono uno contra uno, donde ambos jugadores comparten cronómetro y resultados en tiempo real para añadir tensión positiva y reforzar la práctica; un aprendizaje adaptativo que calibra la dificultad y ajusta las preguntas a cada usuario; y un modo offline completo que permite entrenar sin conexión. Esta combinación cubre lagunas detectadas en productos líderes.

## **8.3 Aportación innovadora**

La innovación no reside en una única tecnología nueva, sino en cómo se orquestan las ya disponibles para lograr una experiencia continua y personalizada. Al integrar la lógica adaptativa, la competición inmediata y el funcionamiento híbrido on/offline en una sola arquitectura compacta, *GameGlish* aporta un enfoque fresco que acerca la práctica diaria de inglés a usuarios que disponen de pocos minutos y conexiones inestables.

## **8.4 Prueba de concepto**

El proyecto entrega un MVP funcional que ya se ejecuta en dispositivos reales y que incorpora registro e inicio de sesión seguros, un test diagnóstico con un conjunto inicial de ejercicios adaptados al nivel del usuario y un duelo en línea con emparejamiento rápido, demostrando que la arquitectura elegida soporta tanto el juego individual sin conexión como la competencia en tiempo real y validando así la viabilidad técnica y la utilidad pedagógica de la propuesta.

# 

# **9. ANÁLISIS DEL PROBLEMA**

El análisis del problema permite comprender las necesidades y retos que debe cubrir el trabajo de final de grado para ofrecer un aprendizaje de inglés atractivo dentro de un dispositivo Android. El estudio se realiza de forma sistemática partiendo de la investigación de las apps líderes y terminando en la definición de requisitos con el fin de detectar oportunidades de mejora y traducirlas en especificaciones claras de sistema.

## **9.1 Especificación de requisitos**

### **9.1.1 Requisitos funcionales**

1. **Autenticación de usuarios**: El usuario podrá registrarse e iniciar sesión a través de un sistema de autenticación (Firebase Auth).
2. **Modo individual**: Los usuarios pueden practicar inglés a través de niveles de preguntas y respuestas de diferentes temas y niveles de dificultad.
3. **Modo competitivo**: Los usuarios pueden competir en tiempo real con otros usuarios en ejercicios de vocabulario y gramática.
4. **Sistema de puntuación y niveles**: Se implementará un sistema de puntos que permita a los usuarios acumular y avanzar de nivel.
5. **Retroalimentación y análisis personalizado**: Se proporcionarán estadísticas y recomendaciones basadas en el rendimiento del usuario.

### **9.1.2 Requisitos no funcionales**

1. **Usabilidad**: La aplicación debe ser intuitiva y fácil de usar.
2. **Rendimiento**: La aplicación debe funcionar de forma fluida, sin demoras, especialmente en el modo competitivo.
3. **Compatibilidad**: Debe ser compatible con la mayoría de dispositivos Android modernos y adaptarse a diferentes tamaños de pantalla.
4. **Seguridad de datos**: Garantizar la seguridad de los datos del usuario, especialmente en autenticación y almacenamiento.

### **9.1.3 Funcionalidades extra**

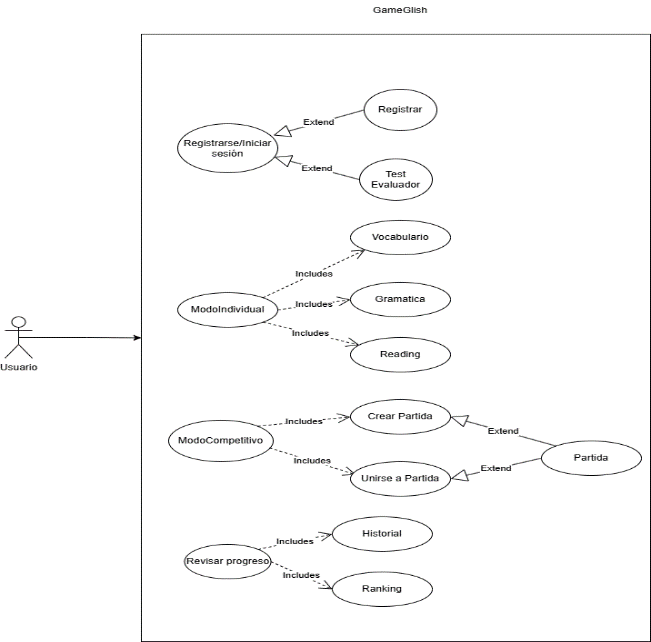
1. **Modo día/noche**: La aplicación detecta el modo del dispositivo del teléfono y adapta el tema según el mismo.
2. **Autenticación con Google**: El usuario puede iniciar sesión y registrarse con Google sin necesidad de pasar por la pantalla de Registro.
3. **Multidispositivo**: Todo se guarda en la nube mediante Firebase (cualquier usuario puede acceder a sus datos (usuario/estadísticas/nivel) desde distintos terminales o si borra y desinstala la aplicación.
4. **Modo sin conexión**: Una vez estás autenticado si pierdes o desconectas tu dispositivo puedes seguir practicando en el modo individual y los resultados se guardan en local. Cuando vuelvas a tener conexión se sincronizarán los datos en remoto (Firebase)

## **9.2 Modelado conceptual**

Para plasmar visualmente el comportamiento del sistema se emplean diagramas UML (Unified Modeling Language):

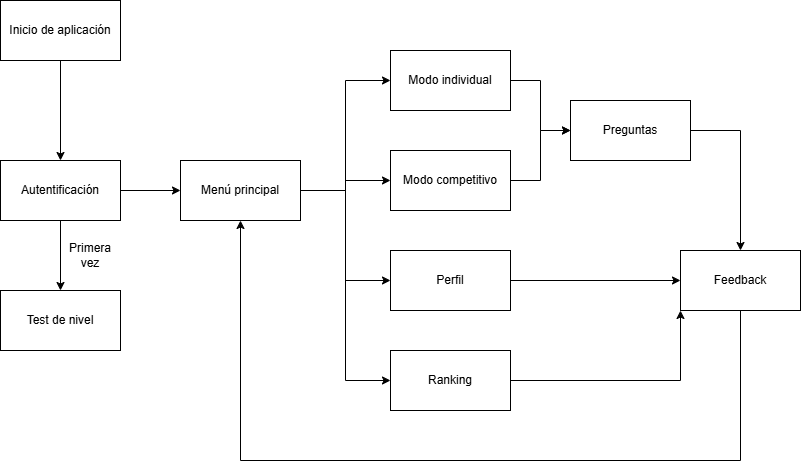
* **Diagrama de casos de uso**

El diagrama de casos de uso de GameGlish **(FIGURA 1)** muestra las acciones que un Usuario puede realizar para aprender y competir dentro de la aplicación. A través de una interfaz sencilla y flujos bien definidos, el alumno pasa por cuatro grandes bloques de funcionalidad: acceso al sistema, práctica individual, duelos en línea y revisión de su progreso. A continuación, se describen los elementos principales:

* **Registrarse / Iniciar sesión** Permite crear una cuenta nueva o acceder a una existente.  
   Extiende a:
  + **Registrar** – alta de credenciales y datos básicos.
  + **Test Evaluador** – prueba inicial que calibra el nivel y configura la dificultad de las actividades posteriores.
* **Modo Individual** Ofrece ejercicios adaptados que el usuario resuelve a su ritmo y sin necesidad de conexión.  
   Incluye:
  + **Vocabulario** – preguntas de traducción y asociación de palabras.
  + **Gramática** – selección de respuesta correcta en estructuras gramaticales.
  + **Reading** – comprensión rápida de frases cortas.
* **Modo Competitivo** Habilita duelos síncronos *uno contra uno* en tiempo real.  
   Incluye:
  + **Crear Partida** – el usuario genera una sala y espera rival.
  + **Unirse a Partida** – el usuario se conecta a una sala existente.  
     Ambas acciones *extienden* el caso de uso **Partida**, donde se gestionan preguntas, cronómetro compartido y resultado final.
* **Revisar progreso** Sección donde el aprendiz consulta su rendimiento global.  
   *Incluye:*
  + **Historial** – lista de sesiones con número de aciertos y fecha.
  + **Ranking** – posición del jugador frente al resto según puntos acumulados en duelos y sesiones individuales.

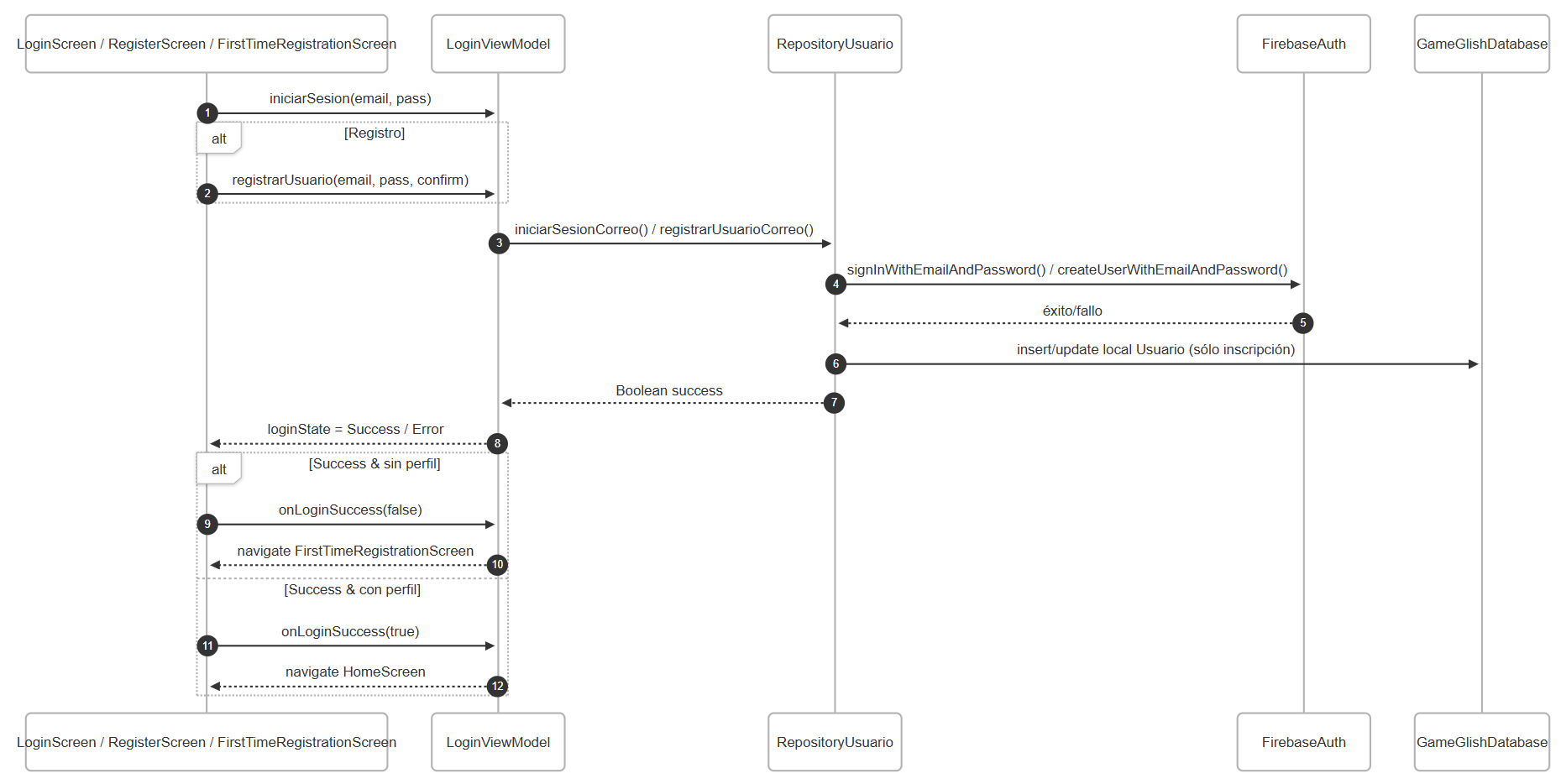
Este diagrama resume las interacciones clave del sistema y sirve de base para el diseño de la interfaz, la lógica de presentación en MVVM y los servicios de datos tanto locales como en la nube.

* **Diagrama de flujo** – Describe la navegación principal: pantalla de inicio → test diagnóstico (solo primera vez) → menú principal → selección de modo (individual / competitivo) → sesión de preguntas → feedback → menú.



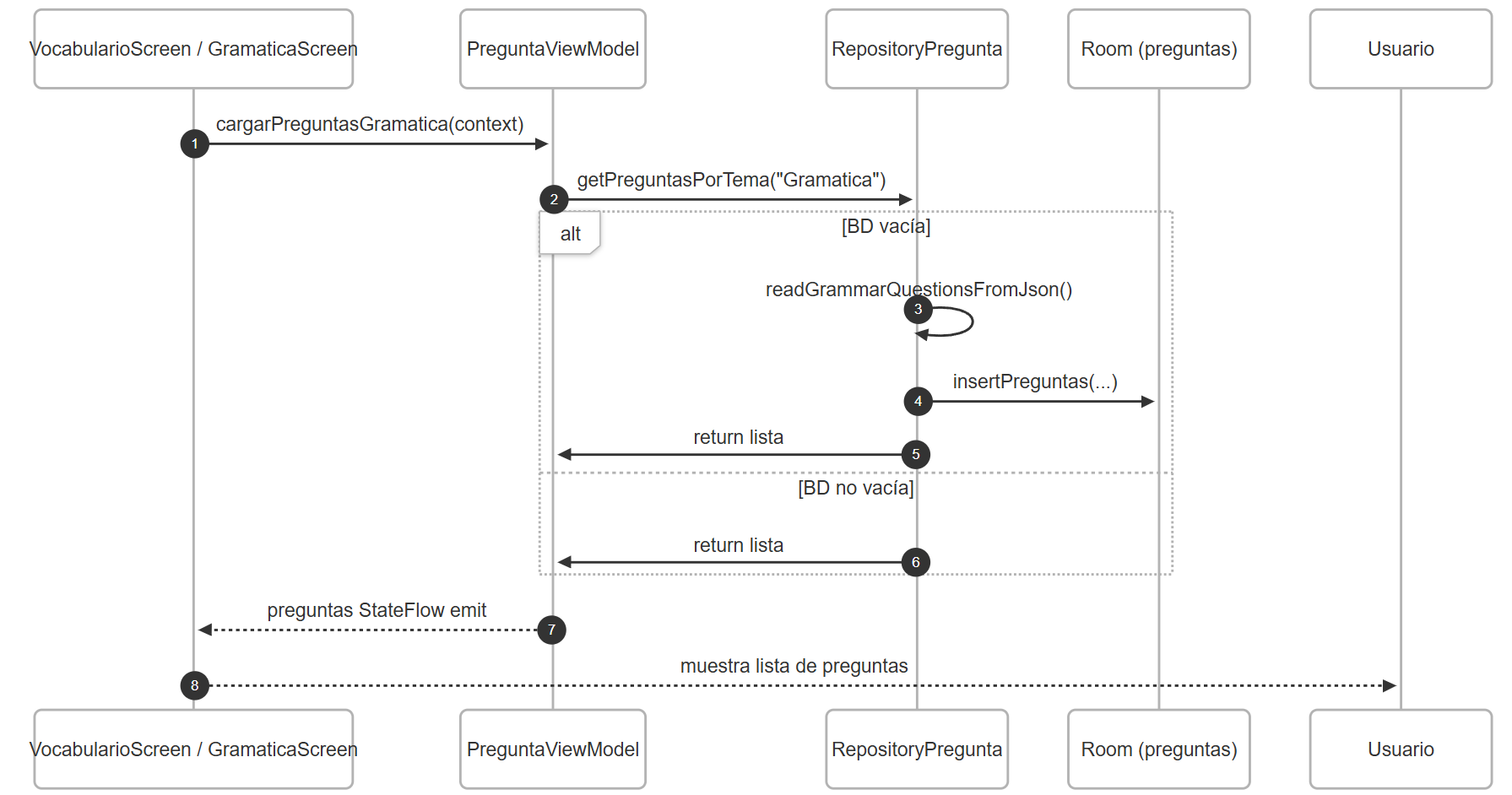
*Ilustración 2 Diagrama de Flujo*

* **Diagramas de secuencia** – Se detallan los principales casos de uso:  
  1. Login mediante firebase:



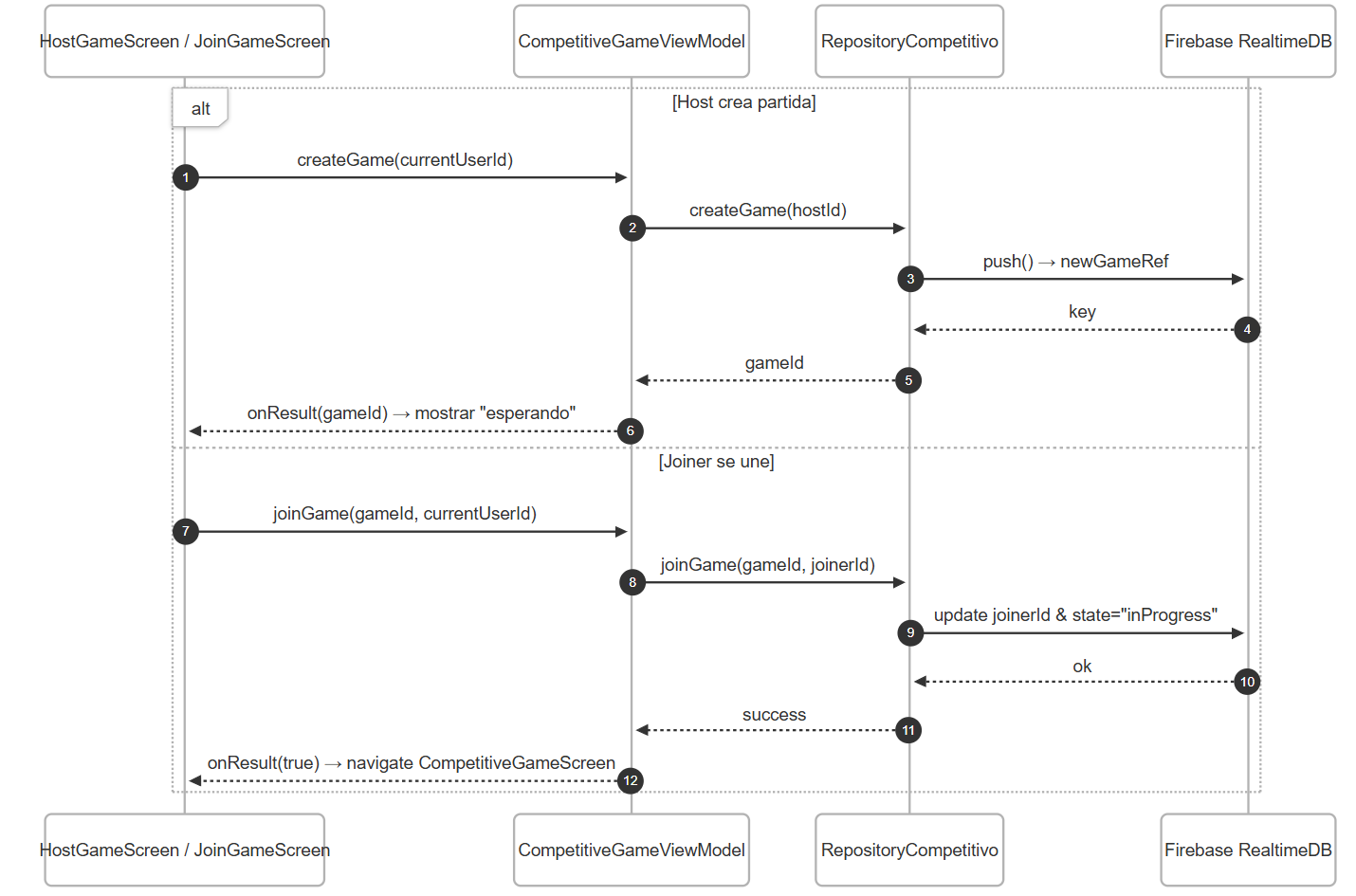
*Ilustración 3 Diagrama de secuencia Login*

* 1. Practicar lección individual (Modo individual).



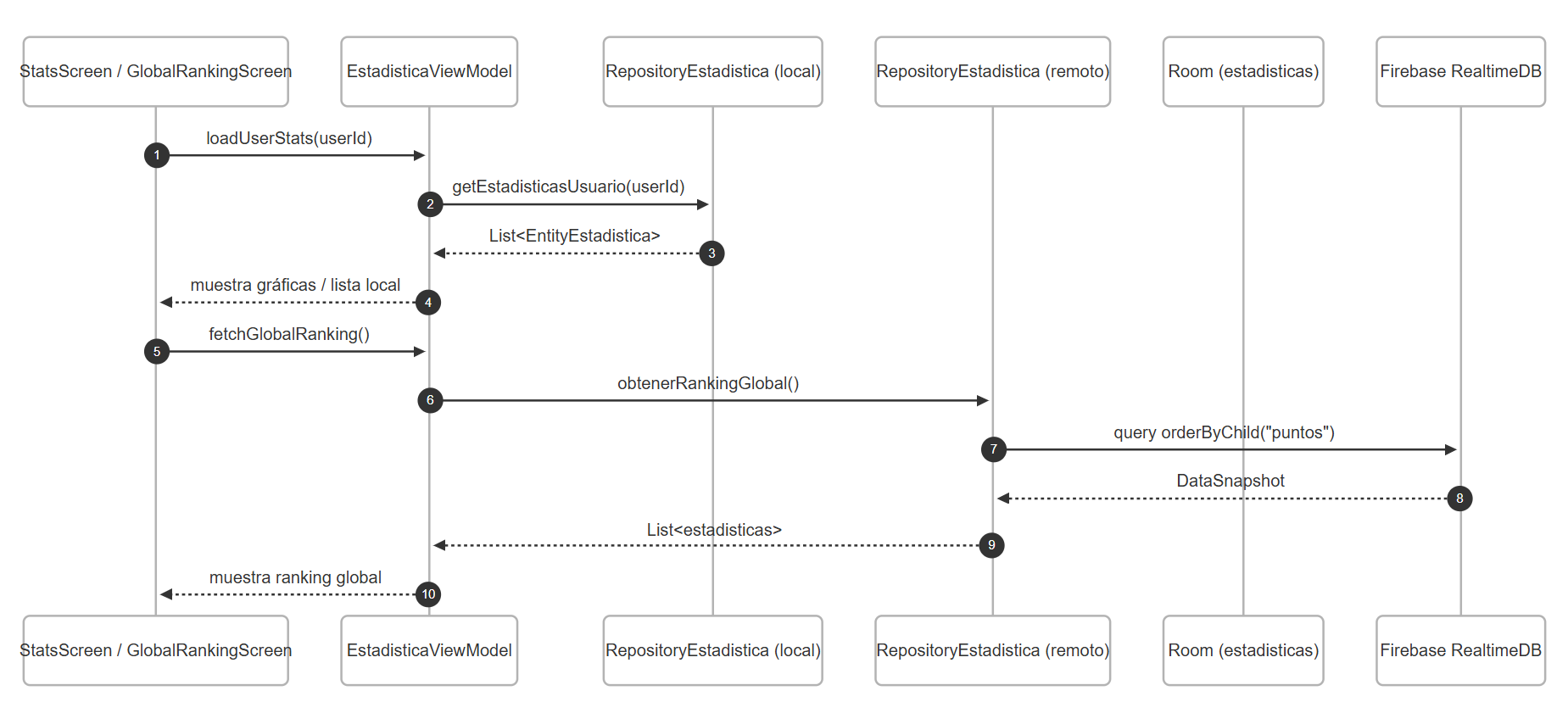
*Ilustración 4 Diagrama de secuencia Modo Individual*

* 1. Duelo síncrono (Modo Competitivo).



*Ilustración 5 Diagrama de secuencia Modo Competitivo*

* 1. Ranking (obtención de ranking global en linea).



*Ilustración 6 Diagrama de secuencia Ranking*

# **10. ANÁLISIS DE LA SEGURIDAD**

## **10.1 Amenazas identificadas**

| **Código** | **Riesgo** | **Escenario probable** | **Impacto** |
| --- | --- | --- | --- |
| **T-01** | Acceso no autorizado | Un atacante intenta iniciar sesión con credenciales robadas o contraseñas débiles. | Robo de identidad y de progreso del usuario. |
| **T-02** | Interceptación de tráfico | Captura de paquetes cuando el dispositivo usa Wi-Fi pública durante un duelo. | Filtrado de respuestas y posible manipulación del marcador. |
| **T-03** | Manipulación de datos offline | Alteración manual de la base local (Room/SQLite) para subir puntuaciones falsas cuando vuelva la conexión. | Desprestigio del ranking y pérdida de confianza de los usuarios. |
| **T-04** | Dependencias vulnerables | Librerías de terceros desactualizadas (p. ej. Jetpack, Firebase SDK). | Ejecución de código malicioso, escalada de privilegios. |

## 

## **10.2 Controles Implementados**

| **Código** | **Mitigación** | **Descripción práctica en GameGlish** |
| --- | --- | --- |
| **C-01** | Autenticación robusta | Firebase Auth con correo/contraseña. |
| **C-02** | Cifrado en tránsito | Todo el tráfico usa **HTTPS/TLS 1.3**; en el duelo los mensajes se intercambian por WebSockets sobre TLS. |
| **C-03** | Reglas de seguridad en la nube | Firebase Realtime DB restringe cada lectura/escritura al **uid** autenticado; escritura en nodo /ranking solo mediante función Cloud Function firmada. |
| **C-04** | Análisis de dependencias | Job CI semanal con **OWASP Dependency-Check** y “Gradle versions plugin” que alerta de CVE conocidas. |

# 

# **11. ANÁLISIS DEL MARCO LEGAL Y ÉTICO**

## **11.1 Protección de datos (GDPR/LOPD-GDD)**

* **Datos tratados.** Se guardan únicamente correo electrónico, alias interno y métricas de uso (nivel, aciertos, partidas jugadas). No se recoge ni foto ni localización.
* **Finalidad.** Autenticar al usuario, adaptar la dificultad y mostrar su posición en el ranking; los datos no se ceden a terceros ni se emplean con fines comerciales.
* **Almacenamiento.** Firebase configura la región europea; el tráfico viaja por HTTPS y los nodos de la base de datos están restringidos al uid autenticado.

## **11.2 Propiedad intelectual y licencias**

* **Código propio.** Todo el código fuente desarrollado para el TFG queda bajo licencia MIT y se entrega al tribunal junto con la memoria.
* **Dependencias.** Se usan únicamente bibliotecas provistas por Android Jetpack y el SDK de Firebase, ambos con licencia Apache 2.0; el archivo NOTICE recoge los avisos.
* **Banco de preguntas.** Las frases y vocabulario han sido redactados expresamente por el autor.
* **Nombre y logotipo.** “GameGlish” se usa solo como título académico del proyecto; no se registra marca ni se distribuye comercialmente.

## **11.3 Uso de contenido multimedia de terceros**

La aplicación no incluye audio, vídeo ni imágenes externas:

* Los iconos son los que ya incorpora **Jetpack Compose** (Material Icons) bajo Apache 2.0
* Las capturas de pantalla del prototipo son propias.

# **12. ANÁLISIS DE RIESGOS**

## **12.1 Riesgos técnicos**

| **Código** | **Riesgo** | **Prob.** | **Impacto** | **Estrategia de mitigación** |
| --- | --- | --- | --- | --- |
| **RT-1** | Pérdida de conexión en mitad de un duelo 1 × 1 | Media | Alta (la partida continua para los dos) | Si algún usuario no responde en el tiempo límite se le quita una vida. |
| **RT-2** | Desincronización Room ⇆ Firebase | Media | Media | Métodos de sincronización iniciados al logear/volver conexión tomando como referencia cualquier cambio. |
| **RT-3** | Rendimiento deficiente en móviles de gama baja | Baja | Media | Preguntas cacheadas mediante Room para acceso rápido en conexiones lentas y dispositivos limitados. |

## **12.2 Riesgos de aceptación del usuario**

| **Código** | **Riesgo** | **Prob.** | **Impacto** | **Estrategia de mitigación** |
| --- | --- | --- | --- | --- |
| **RA-1** | Curva inicial demasiado exigente | Media | Media | Test diagnóstico ajusta el nivel y muestra una sesión guiada la primera vez. |
| **RA-2** | Falta de motivación tras varios días | Media | Alta | Actualización constante de nuevas preguntas de manera semanal. |
| **RA-3** | Emparejamiento percibido como injusto | Media | Alta | La UI muestra el nivel del contrincante antes de aceptar el duelo. |

# **13. IDENTIFICACIÓN Y ANÁLISIS DE SOLUCIONES POSIBLES**

## **13.1 Soluciones y alternativas analizadas**

1. **Desarrollo nativo Android (Kotlin + Jetpack Compose + Firebase):**  
   La primera posibilidad fue construir *GameGlish* con las herramientas oficiales de Android: Kotlin como lenguaje principal, Jetpack Compose para la interfaz y Firebase (Auth y Realtime DB) como servicio en la nube. Este camino brinda un control total sobre el ciclo de vida de la app, acceso inmediato a cada API del sistema y el rendimiento más alto en dispositivos Android. Permite además un soporte offline sencillo combinando Room con los servicios de sincronización de Firebase. El precio que se paga es un mayor trabajo si en el futuro se quisiera portar a iOS, pues habría que reescribir la aplicación o crear un módulo compartido.
2. **Desarrollo multiplataforma con Flutter.**

La segunda opción evaluada fue emplear Flutter, que compila a código nativo tanto en Android como en iOS a partir de un único código Dart. La ventaja evidente es reducir tiempo y costes si se planeara lanzar la aplicación en ambas plataformas. Sin embargo, en el contexto de un TFG centrado en D.A.M. se introduce un lenguaje nuevo, una capa de abstra­cción extra y un peso final de APK mayor.

1. **Backend propio con API REST (Spring Boot) + app nativa.**

Una tercera alternativa consistía en desplegar un servidor REST hecho con Spring Boot y PostgreSQL, mientras la app nativa lo consumiría mediante HTTPS. Este enfoque otorga flexibilidad absoluta sobre la lógica de negocio y el esquema de datos, pero multiplica el esfuerzo: exige diseñar, asegurar y mantener la infraestructura, configurar escalado y asumir costes de hosting. Para un proyecto académico orientado a mostrar destreza móvil, la parte back-end correría el riesgo de desviar tiempo y recursos.

## **13.2 Criterios de selección**

**Para decidir se valoraron cuatro factores:**

1. **Enfoque docente**. El trabajo debía demostrar competencias adquiridas en el ciclo D.A.M., donde el desarrollo Android nativo es central.
2. **Tiempo disponible**. El calendario del TFG obliga a priorizar la productividad y evitar curvas de aprendizaje ajenas al plan de estudios.
3. **Rendimiento y soporte offline**. El modo individual sin conexión es un requisito clave: una pila nativa con Room y Firebase cubre esta necesidad con poca complejidad añadida.
4. **Coste de operación**. La capa gratuita de Firebase basta para el piloto y escala automáticamente si el número de alumnos crece; un backend propio implica gastos fijos y mantenimiento continuo.

Tras ponderar estas variables, se optó por el desarrollo nativo Android con Firebase. Esta solución aprovecha el dominio previo del lenguaje, garantiza un rendimiento óptimo, simplifica la sincronización en tiempo real y mantiene el proyecto dentro de los límites de esfuerzo y recursos propios de un Trabajo Fin de Grado.

# **14. SOLUCIÓN PROPUESTA**

## 

## **14.1 Introducción**

Tras comparar las alternativas tecnológicas y sopesar los riesgos, se opta por una solución nativa Android que sacará partido del ecosistema ya estudiado en el ciclo D.A.M. (Desarrollo de Aplicaciones Multiplataforma) El objetivo es entregar una versión funcional de **GameGlish** capaz de cubrir los dos modos de aprendizaje (individual y competitivo), funcionar sin conexión y almacenar el progreso en la nube cuando haya red.

## **14.2 Descripción de la solución**

* Arquitectura **MVVM** en **Kotlin + Jetpack Compose**.
* **Modo individual**: bloques de preguntas descargados y cacheados con **Room**, de modo que el usuario pueda practicar sin conexión.
* **Modo competitivo**: emparejamiento sencillo mediante un nodo libre en **Firebase Realtime DB**; las respuestas viajan en tiempo real y el cronómetro es compartido.
* **Sincronización híbrida**: cuando la red regresa, el repositorio sube los resultados pendientes y actualiza el ranking.

## **14.3 Fases de desarrollo**

* **Análisis y planificación** Revisión de requisitos, descomposición en historias de usuario y estimación de esfuerzo en un tablero Kanban local.
* **Diseño UI/UX** Bocetos rápidos en papel → prototipo en Compose Preview → ajustes
* **Implementación de la capa de datos** Repositorios Room (local) y Firebase (remoto) con lógica de sincronización.
* **Modo individual** Test diagnóstico + primeros bloques de vocabulario y gramática, generados desde un Json importado a Room.
* **Modo competitivo** Creación y unión a partida, cronómetro de 60 s, intercambio de respuestas y cálculo de ganador.
* **Sincronización offline/online** Pruebas de desconexión simulada; subida de las sesiones pendientes al restablecer la red.
* **Pruebas internas** Unitarias (JUnit/Robolectric) y de interfaz (Compose UI Test); instalación del APK en cuatro móviles de distintas gamas para detección de cuelgues y perfiles de memoria.

## **14.4 Implantación y validación**

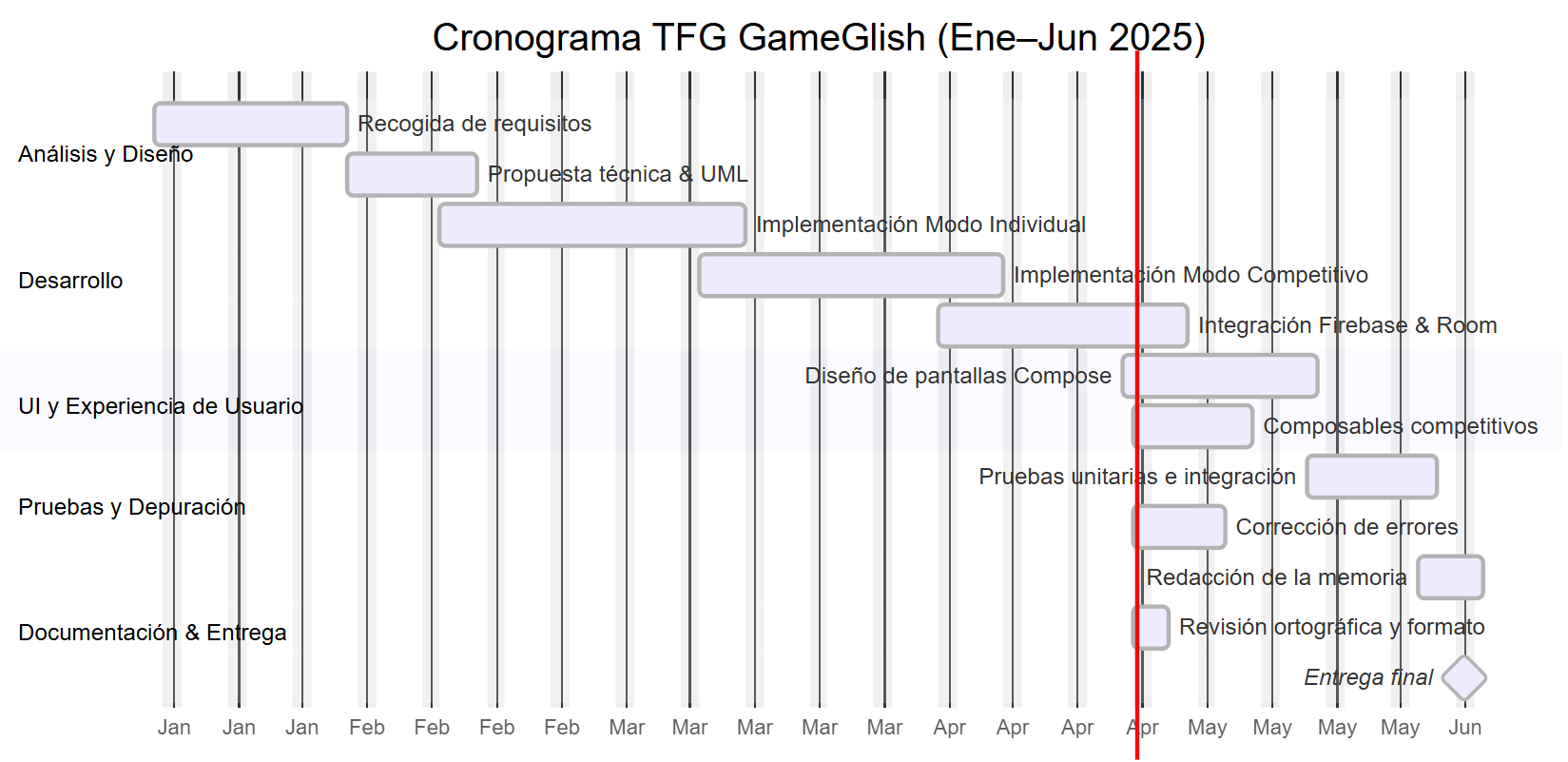
* **Pruebas de funcionalidad**: checklist de registro, test inicial, sesión individual, duelo y sincronización.
* **Pruebas de usabilidad**: encuesta breve (escala 1-5) sobre claridad de la interfaz y percepción de dificultad.
* **Pruebas de rendimiento**: medición manual de tiempo de arranque (< 2 s en gama media) y latencia de duelo (≈ 220 ms en Wi-Fi doméstico).
* **Pruebas de seguridad básicas**: verificación de reglas de Firebase y trazas TLS en todas las llamadas salientes.

# **15. PLAN DE TRABAJO**

El diagrama de Gantt es una herramienta clave para la gestión de proyectos, pues ofrece una representación gráfica del calendario global, facilitando la programación, coordinación y supervisión de todas las tareas implicadas. En este Trabajo de Fin de Grado (TFG) se ha empleado un diagrama de Gantt para planificar y controlar las distintas fases del desarrollo de la aplicación de chat en Android, garantizando que cada etapa se ejecute de manera óptima y dentro de los plazos establecidos.

El diagrama de Gantt que se presenta a continuación muestra el orden y la duración de las actividades principales del proyecto. Cada barra en el gráfico identifica una tarea específica, indicando su fecha de inicio y de finalización. Además, se reflejan las dependencias entre tareas, destacando cómo la finalización de una actividad condiciona el inicio de la siguiente.

En definitiva, el diagrama de Gantt constituye un elemento esencial en la planificación del proyecto, aportando una visión clara y estructurada del cronograma y ayudando a asegurar que todas las fases del desarrollo de la aplicación se completen de forma ordenada y en el tiempo previsto. A continuación, se detalla el diagrama de Gantt con todas las tareas e hitos relevantes del proyecto



*Ilustración 7 Diagrama de Gantt*

# **16. DISEÑO DE LA SOLUCIÓN**

## **16.1 Arquitectura del sistema (MVVM)**

Para aislar la lógica de negocio de la interfaz y mantener el código testeable, **GameGlish** se construye sobre el patrón **Model-View-ViewModel**:

* **Modelo (Model)** – ubicado en data/model/.  
   Contiene las entidades EntityUsuario, EntityPregunta, EntityEstadistica, EntityRanking y CompetitiveGame, cada una con su mínima lógica de dominio (p. ej. EntityPregunta.toCompetitiveQuestion)
* **Vista (View)** – carpetas view/auth, view/modoindividual, view/modocompetitivo y pantallas generales (ProfileScreen, GlobalRankingScreen, etc.).  
   Cada pantalla es un @Composable, recibe un UiState inmutable y emite *callbacks* para las intenciones del usuario.
* **Modelo de vista (ViewModel)** – viewmodel/.  
   LoginViewModel, PreguntaViewModel, CompetitiveViewModel y StatsViewModel median entre la vista y los datos, exponen StateFlow<UiState> y delegan cualquier lectura/escritura al **Repository**.
* **Repository** – data/repository/.  
   Capa única que decide, en tiempo de ejecución, si responde con **Room** (offline) o con **Firebase Realtime DB** (online).

## 

## **16.2 Modelo de datos**

* **Usuario** (uidFirebase, email, nombre, nivel, firstLogin)
* **Pregunta** (id, enunciado, opcionA, opcionB, opcionC, opcionD, opcionCorrecta, tema, nivel)
* **Estadística** (remoteId, userId, fecha, aciertos, errores, puntos)
* **CompetitiveGame** (gameId, hostId, joinerId, currentQuestion, answerOptions, hostLives, joinerLives, timeleft, state, winner, correctId, timestamp)
* **Ranking** (userId, nombre, nivel, puntos)

Las tablas DaoPregunta, DaoUsuario y DaoEstadistica persisten en Room; los mismos datos se reflejan en los nodos /users, /questions y /duels de Firebase cuando hay red.

## **16.3 Interfaz de usuario**

| **Flujo** | Pantallas principales (Compose) mediante AppNavHost |
| --- | --- |
| **Autenticación** | LoginScreen, RegisterScreen, FirstTimeLoginScreen |
| **Modo individual** | VocabularioScreen, GramaticaScreen, ReadingScreen, ListeningScreen, ModoIndividualMainScreen |
| **Modo competitivo** | HomeGameScreen, JoinGameScreen, CompetitiveGameScreen, ModoCompetitivoMainScreen |
| **Generales** | HomeScreen, ProfileScreen, GlobalRankingScreen, SettingsScreen |

## **16.4 Gestión de estados**

Cada ViewModel expone un StateFlow que la interfaz observa; el repositorio decide en tiempo real si la lectura/escritura va a **Room** (offline) o a **Firebase Realtime DB** (online). Al recuperar la conexión, los datos pendientes se sincronizan automáticamente, asegurando una experiencia fluida y coherente para el usuario.

## **16.5 Navegación y flujos de pantalla**

La lógica de cambio de pantallas se articula mediante **Jetpack Navigation-Compose** y cuatro *NavControllers* que se muestran en la carpeta navigation/:

| **Archivo** | **Rol** | **Contenido principal** |
| --- | --- | --- |
| **AppNavHost.kt** | **NavController raíz** | Decide si el usuario ve el flujo de autenticación (Login/Register) o el flujo principal (MainNavHost). |
| **MainNavHost.kt** | Contenedor del *bottom bar* | Aloja las rutas de los tres flujos funcionales: Home → Individual → Competitivo → Perfil/Ajustes. |
| **IndividualFlowScreen.kt** | Host del **modo individual** | Navega por los composables VocabularioScreen, GramaticaScreen, ReadingScreen y ListeningScreen. Cada ruta recibe argumentos (themeId, level) y comparte un mismo PreguntaViewModel mediante hiltViewModel() para conservar estado al rotar la pantalla. |
| **CompetitivoFlowScreen.kt** | Host del **modo competitivo** | Gestiona la sucesión HomeGameScreen → JoinGameScreen / CreateGameScreen → CompetitiveGameScreen →, usando un único CompetitiveViewModel inyectado por Hilt. |
| **MainFlowScreen.kt** | Selección de pestaña (bottom-navigation) | Levanta el MainNavHost y expone el *NavBar* con iconos Material 3; cada pestaña cambia la ruta del *NavController* pero mantiene el estado de la pantalla anterior gracias a rememberSaveable. |

# **17. TECNOLOGÍA UTILIZADA**

## **17.1 Entorno de desarrollo**

**Android Studio Meerkat Feature Drop | 2024.3.2** con AGP 8.8 y el plugin “Kotlin Compose”. Durante el TFG aprendí a usar el inspector de composición, los *previews* en tiempo real y la canalización de “Run → Multiple Devices” para depurar en varios móviles a la vez además del device inspector para ver los archivos internos del terminal.

## **17.2 Lenguaje de desarrollo**

**Kotlin 2.1.0**. Empecé con nociones básicas; ahora domino corutinas, StateFlow, extensiones KTX y la sintaxis declarativa de Jetpack Compose.

## **17.3 Sistema operativo**

**Android (API 26 → 34)**. Afiné la gestión de permisos en Android 13 y los ajustes de optimización de batería que pueden matar procesos en segundo plano.

## **17.4 Frameworks y bibliotecas**

* **Firebase Auth 23.1.0** y **Realtime DB 21.0.0** para registro, emparejamiento y progreso en la nube.
* **Room 2.6.1** (con SQLCipher) para caché local y modo offline.
* **Jetpack Compose BOM 2024-04** + **Navigation-Compose 2.8.5** para la interfaz declarativa.
* **Hilt 2.48** para inyección de dependencias.

## **17.5 APIs y permisos**

* **INTERNET** – tráfico sobre HTTPS, imprescindible para Firebase.

## **17.6 Dependencias adicionales**

* **Compose Material 3, Compose UI-test, Coroutines-Android 1.7.3.**
* **Play-services-auth 20.5.0** (futuro login Google).
* **JUnit 4.13.2, Espresso 3.6.1 y Compose UI Test** para pruebas.  
   Las versiones se gestionan con libs.versions.toml, garantizando coherencia del proyecto.

## **17.7 Conocimiento personal y viabilidad**

Partí de un nivel medio-básico; tras cinco meses he ensamblado una pila nativa moderna, configurado CI con dependencias actualizadas y demostrado que todas las tecnologías (Firebase free tier, Room, Compose) son sostenibles para un piloto académico y escalables a futuras versiones.

# **18. DESARROLLO DE LA SOLUCIÓN PROPUESTA**

### 

* **Proceso de desarrollo.** Se trabajó con una metodología iterativa-incremental: cada dos semanas se entregaba una versión funcional (login, test inicial, modo individual, duelo…). Después de cada entrega se revisaba el avance con el tutor y se ajustaban tareas y tiempos.
* **Problemas y dificultades.** La integración simultánea de Firebase Auth, Realtime DB y Room planteó distintos retos: sincronizar datos cuando el usuario pasaba de offline a online, mantener la latencia del duelo en límites aceptables y resolver pequeñas diferencias de permisos entre versiones de Android 13 y anteriores.
* **Decisiones tomadas.** Se mantuvo el patrón **MVVM** para separar lógica y UI; se usaron corutinas y StateFlow para simplificar la gestión del estado; y se añadieron logs y pruebas básicas (JUnit + Compose UI Test) para detectar errores de sincronización antes de cada entrega.
* **Particularidades de la solución final.** La app funciona sin conexión gracias a Room y actualiza el progreso en la nube al recuperar red; la interfaz Compose sigue Material 3 y se adapta automáticamente a modo oscuro; la comunicación con Firebase viaja cifrada (HTTPS), garantizando seguridad y una experiencia de usuario fluida.

# **19. IMPLANTACIÓN**

La implantación de **GameGlish** requiere trasladar la app desde el entorno de desarrollo a un escenario real y controlado: primero se genera un APK firmado y se instala en varios móviles de prueba, verificando que Room cree su base local y que las dependencias (Compose BOM 2024-04, Firebase Auth 23.1.0, Realtime DB 21.0.0) coincidan con las versiones compiladas; después se configura el proyecto de Firebase—proveedor “correo + contraseña” activo y reglas de seguridad que restringen cada lectura/escritura al uid—y se comprueba que la app descargue el banco inicial de preguntas y registre usuarios sin errores; a continuación se despliega en un entorno controlado durante dos semanas, donde ocho compañeros validan registro, test diagnóstico, práctica offline y duelos en tiempo real, anotando cualquier fallo de latencia o sincronización; finalmente se recopilan los logs y métricas de uso, se corrigen los problemas detectados y se compila un APK definitivo listo para una futura publicación en el canal interno de Google Play.

# **20. PRUEBAS**

En este apartado se detalla el **plan de pruebas** diseñado para comprobar que *GameGlish* funciona correctamente y satisface las expectativas técnicas y pedagógicas fijadas al inicio del TFG. Las pruebas constituyen una fase crítica: permiten detectar errores, optimizar el rendimiento y asegurar que la versión entregada puede utilizarse en dispositivos reales sin incidencias. A continuación, se incluye una tabla con cada caso de prueba, las condiciones de partida y el resultado obtenido. Este resumen demuestra el compromiso con la calidad; cada caso se definió y ejecutó de forma meticulosa para garantizar que la aplicación de aprendizaje de inglés no solo cumple los requisitos funcionales, sino que también ofrece una experiencia fluida y motivadora para el usuario final.

## **20.1 Pruebas de funcionalidad:**

**CU-PG-01: Registro con correo**

**Objetivo**

**Verificar que el usuario puede crear una cuenta con email/contraseña y que, al finalizar e iniciar sesión la app navega al Test Inicial.**

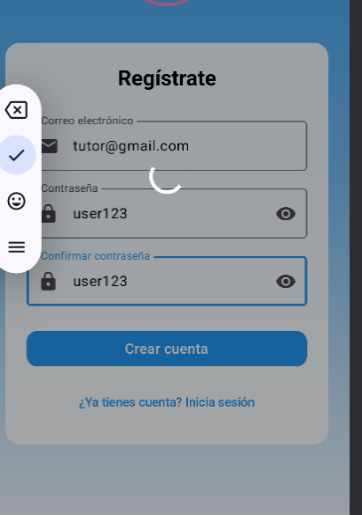
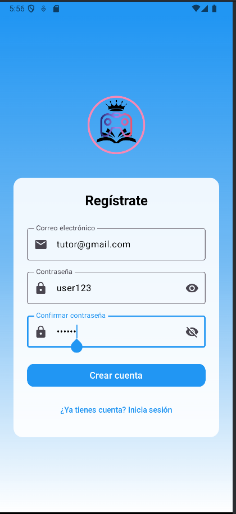
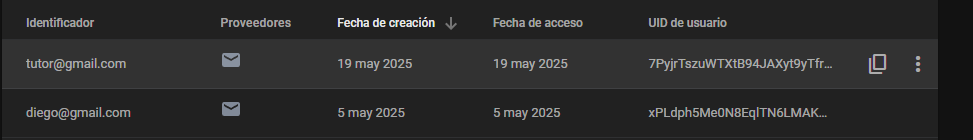
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test) + Room**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

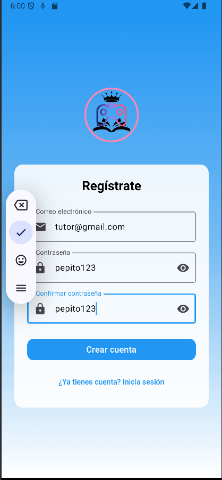
| **N.º** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Registro exitoso** | **Email válido, contraseña válida** | **Cuenta creada y pantalla Login mostrada** | **Registro correcto** | **OK** |
| **2** | **Registro no exitoso** | **Email ya en uso,**  **contraseña válida** | **Cuenta no creada y Toast de error de registro** | **Registro**  **incorrecto** | **OK** |
| **3** | **Registro sin conexión** | **Email válido, contraseña válida** | **Error de registro a causa de no poder conectar.** | **Registro incorrecto** | **OK** |

**Observaciones / capturas:**

**Escenario 1:**

** **

**Escenario 2:**

****

**CU-PG-02: Inicio de sesión**

**Objetivo**

**Comprobar que un usuario registrado accede con sus credenciales y llega a HomeScreen.**

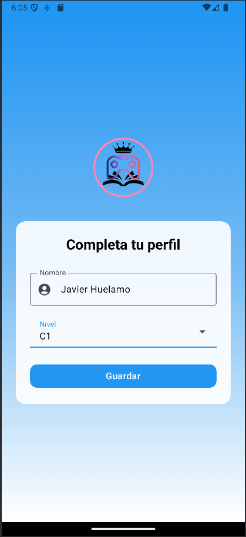
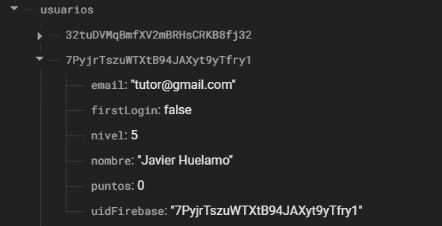
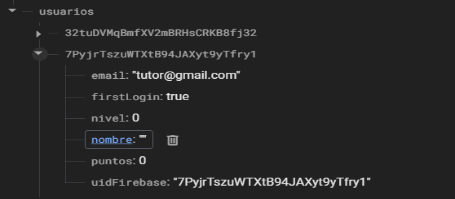
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

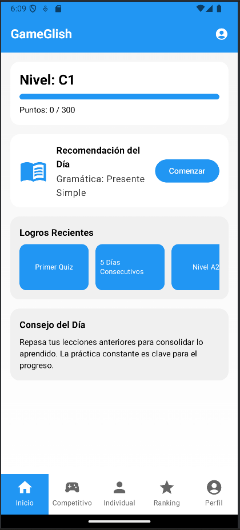
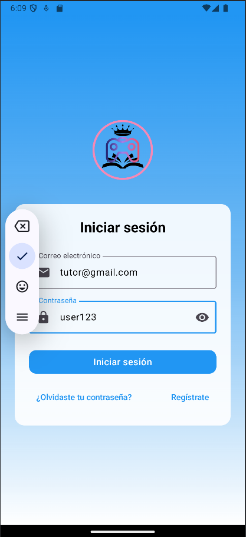
| **N.º** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Login correcto por primera vez** | **Email y contraseña existentes** | **Se muestra FirstTimeLogin** | **Login y navegación correctas** | **OK** |
| **2** | **Login correcto** | **Email y contraseña existentes** | **Se muestra HomeScreen** | **Login y navegación correctas** | **OK** |
| **3** | **Contraseña incorrecta** | **Email existente y contraseña incorrecta** | **Se muestra error de Inicio de sesión** | **Toast de error de Inicio de sesión** | **OK** |
| **4** | **Email incorrecto** | **Email incorrecto y contraseña incorrecta** | **Se muestra error de Inicio de sesión** | **Toast de error de Inicio de sesión** | **OK** |

**Observaciones / capturas:**

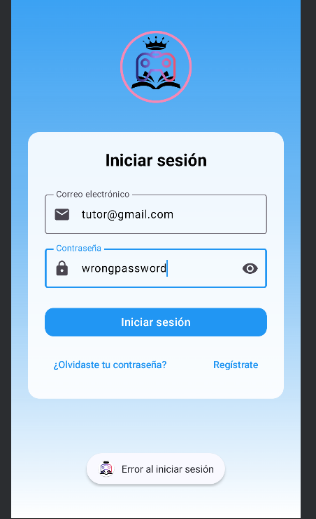
**Escenario 1:**

****

**Escenario 2:**

****

**Escenario 3:**

****

**CU-PG-03: Test inicial calcula nivel**

**Objetivo**

**Validar que el Test Diagnóstico calcula el nivel (A1–C2) a partir del porcentaje de aciertos y lo persiste en el perfil.**

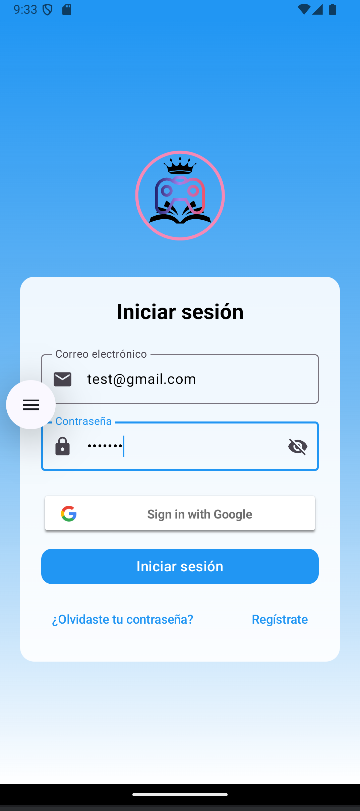
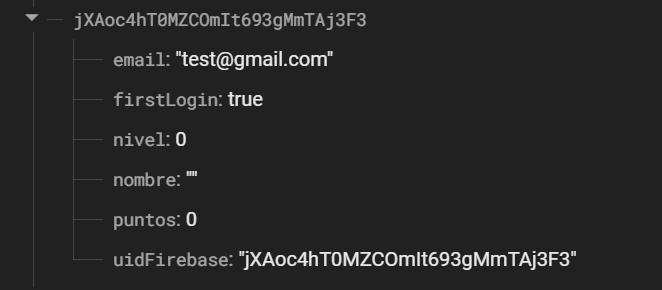
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test) + Room**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

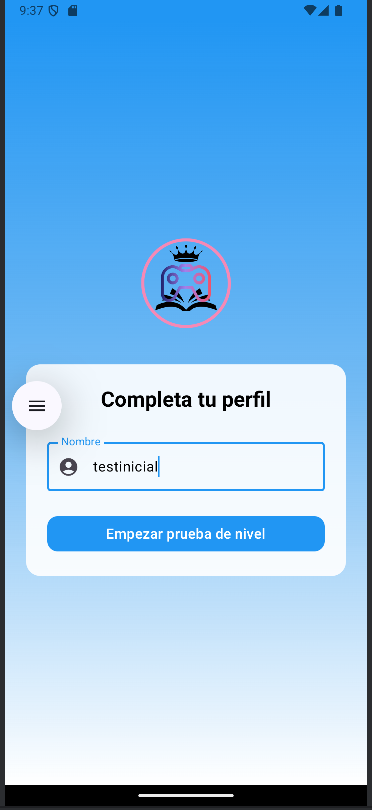
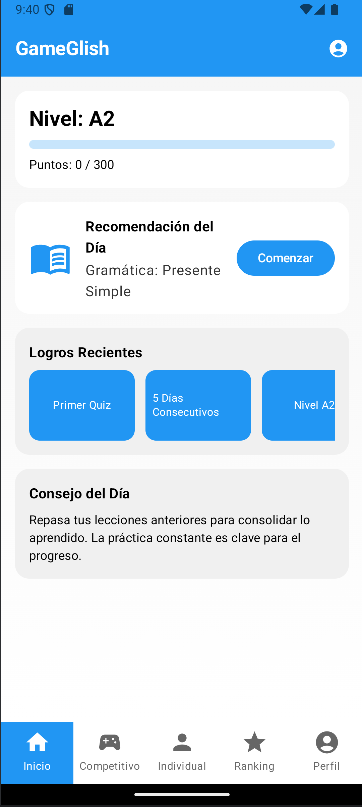
| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Completar test** | **Nombre: test inicial**  **Respuestas a las preguntas.** | **Nivel asignado según % aciertos (A1–C2)** | **Navegación a HomeScreen con el nivel adecuado** | **OK** |

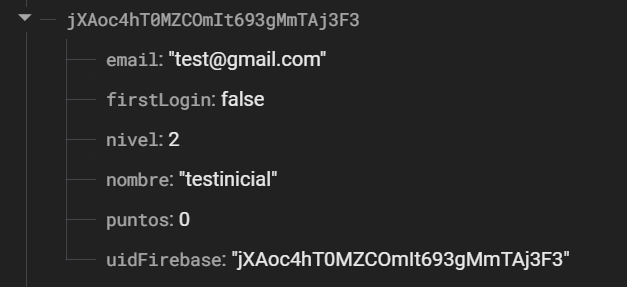
**Observaciones / capturas:**

**Escenario 1:**

**** ****

Vemos que la cuenta es nueva y aún no tiene nombre ni nivel asignado

Hacemos inicio de sesión por primera vez:  
  
   



El valor se actualiza correctamente en firebase y navegamos a HomeScreen.

**CU-PG-04: Práctica modo individual**

**Objetivo**

**Asegurar que el modo individual funciona sin conexión: carga preguntas desde Room, permite responder y guarda la sesión localmente.**

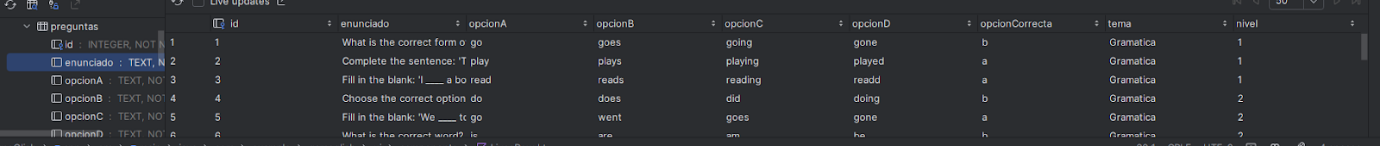
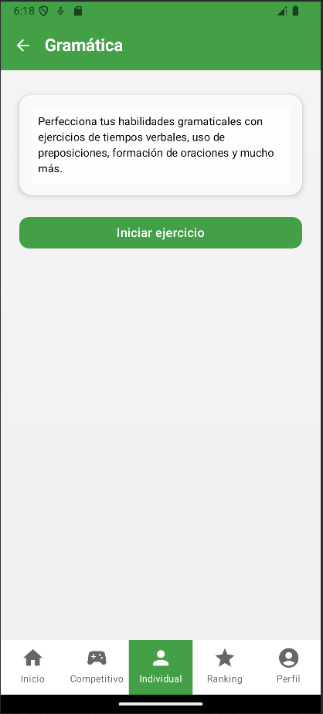
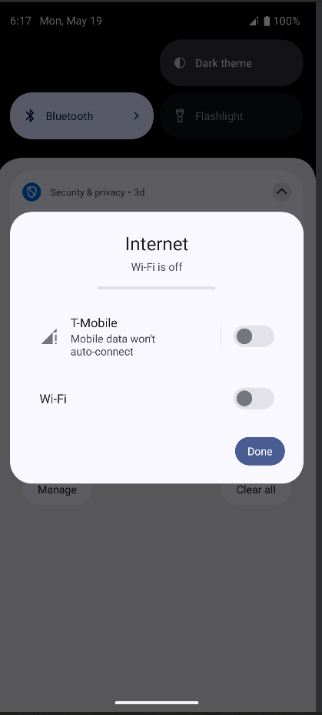
**Entorno de prueba**

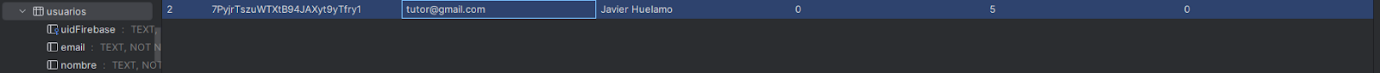
* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: ROOM(SQLITE)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Responder sin conexión** | **Preguntas cacheadas (Room)** | **Sesión guardada local no error de red** | **Puntos e historial guardados  en local** | **OK** |

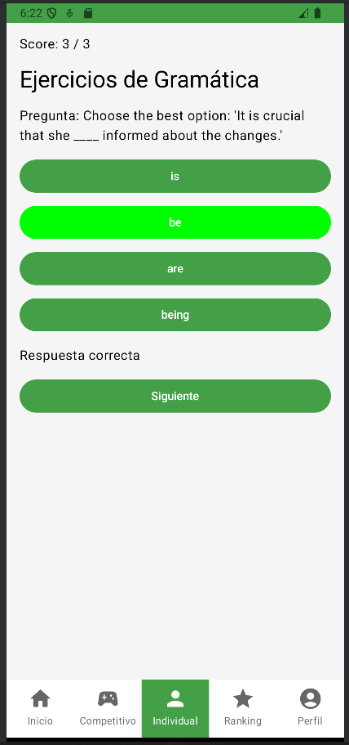
**Observaciones / capturas:**

**Escenario 1: Antes de responder el quiz:**

****

****

**Tras responder el quiz:**

****

****

**Los puntos se suman aun no habiendo conexión**

**CU-PG-05: Sincronizar sesión**

**Objetivo**

**Confirmar que, al recuperar conectividad, una sesión almacenada en local se sincroniza con Firebase y aparece en el Historial.**

**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Volver a estar online** | **Sesión pendiente en Room + red restaurada** | **Resultados subidos y visibles en Historial** | **…** | **…** |

**Observaciones / capturas:**

**Escenario 1:**

**CU-PG-06: Crear partida competitiva**

**Objetivo**

**Garantizar que al pulsar “Crear partida” se genera un ID único, se crea el nodo /waiting y el host queda a la espera de rival.**

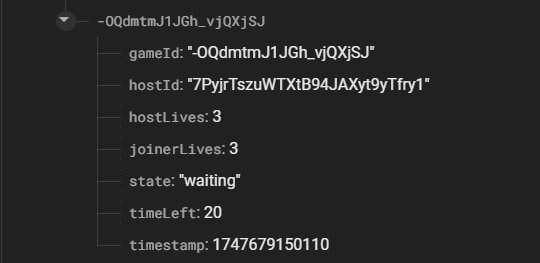
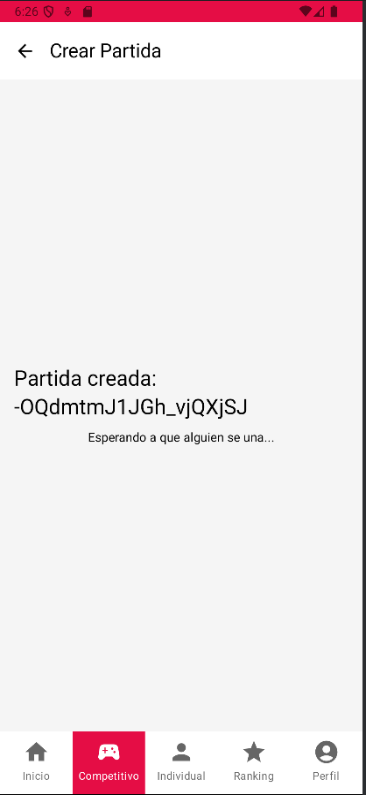
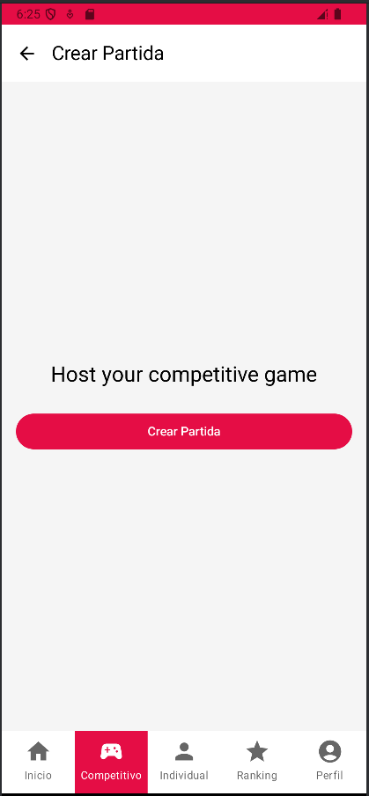
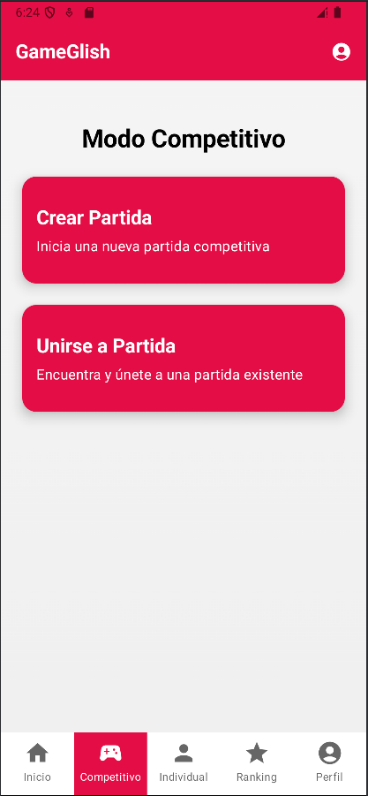
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Crear partida** | **Pulsar “Crear partida”** | **ID generado; estado waiting;** | **Creación de partida en firebase con estado waiting y navegación a HostGameScreen** | **OK** |

**Observaciones / capturas:**

**Escenario 1:**

****

La partida se crea y se queda en “*waiting”*

**CU-PG-07: Unirse a partida**

**Objetivo**

**Verificar que un segundo usuario puede unirse a una partida en estado waiting y que ambos clientes pasan a inProgress en < 300 ms.**

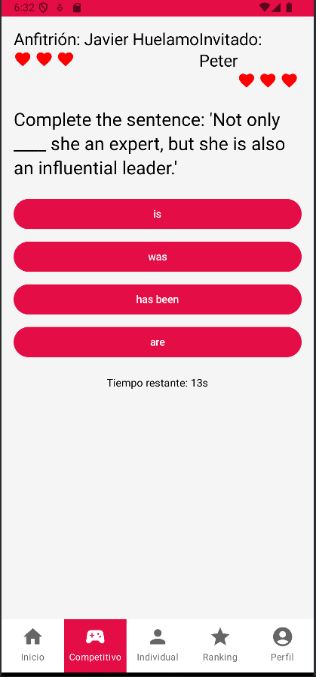
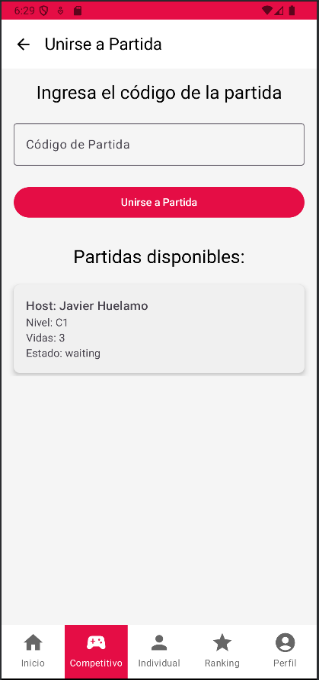
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Unirse partida** | **Código de partida real con estado waiting** | **Cambio a inProgress en < 300 ms; cronómetro comienza** | **Navegación a CompetiveGame** | **…** |

**Observaciones / capturas:**

**Escenario 1:**

****

**CU-PG-08: Pérdida de vida al fallar**

**Objetivo**

**Comprobar que, durante un duelo, una respuesta incorrecta decrementa la variable hostLives o joinerLives y la UI refleja la pérdida de vida.**

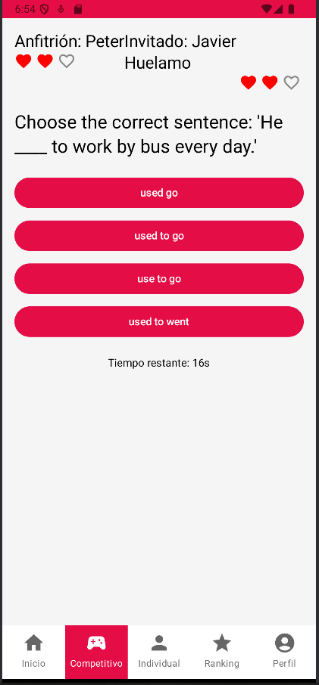
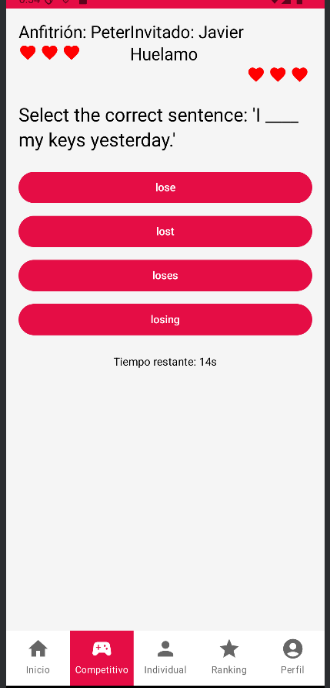
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **N.º** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Respuesta incorrecta (host)** | **Seleccionar opción incorrecta** | **hostLives – 1; UI muestra un corazón menos** | **Resto de vidas y actualización en UI** | **OK** |

**Observaciones / capturas:**

**Escenario 1:**

****

**CU-PG-09 : Fin de partida y ganador**

**Objetivo**

**Validar que la partida termina cuando un jugador se queda sin vidas, que el estado cambia a finished y que el campo winner se asigna correctamente.**

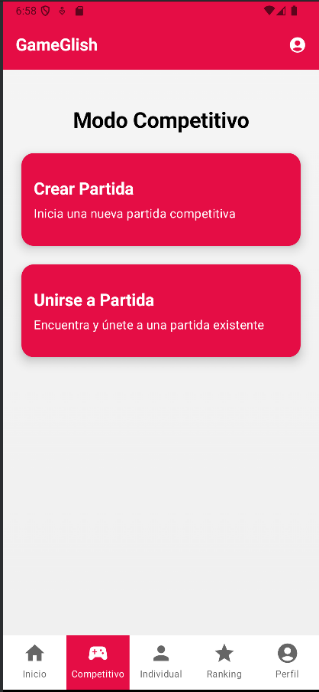
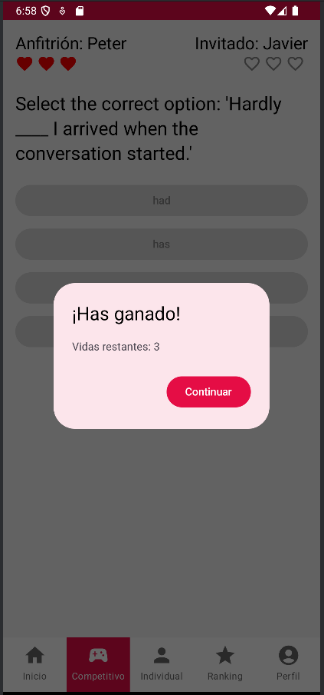
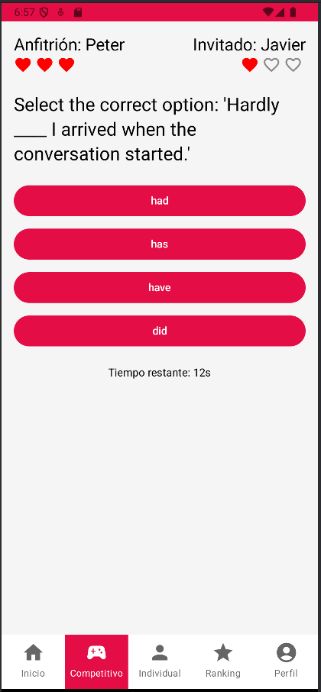
**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Vida a cero** | **Host o joiner con 0 vidas** | **Estado finished; campo winner con UID correcto** | **Navegación a Competitivo y actualización del ranking global** | **OK** |

**Observaciones / capturas:**

**Escenario 1:**

****

**CU-PG-10: Actualizar ranking**

**Objetivo**

**Asegurar que la consulta al nodo de ranking remoto actualiza la posición del usuario y refresca la tabla en la pantalla Ranking.**

**Entorno de prueba**

* **Versión app: 1.0**
* **Dispositivo: Medium Phone API 35(Android 15)**
* **Backend: Firebase Auth + Realtime DB (test)**
* **Fecha de prueba: 18/05/2025**
* **Tester: Javier Huélamo Gracia**

| **Nº** | **Escenario** | **Datos introducidos** | **Resultado esperado** | **Resultado obtenido** | **OK/Error** |
| --- | --- | --- | --- | --- | --- |
| **1** | **Consultar ranking** | **Conexión activa** | **Tabla y posición del usuario en RankingScreen** | **Ranking correcto** | **OK** |

**Observaciones / capturas:**

**Escenario 1:**

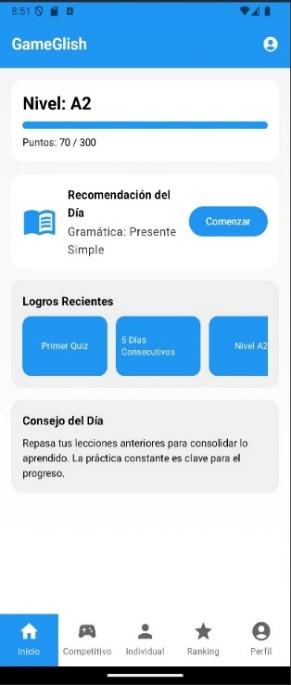
****

## **20.2 Pruebas de accesibilidad**

Prueba modo alto contraste:

Todas las pantallas están diseñadas para permitir el modo oscuro o el modo día dependiendo de la configuración del usuario. Estos son algunos ejemplos en algunas pantallas aunque todas están implementadas de forma similar.

* HomeScreen:



* ModoIndividualScreen:



# **21. CONCLUSIONES**

## **21.1 Cumplimiento de los objetivos**

Los objetivos fijados al inicio del TFG se han alcanzado de forma satisfactoria. La autenticación mediante **Firebase Auth** permite crear y reutilizar cuentas sin pérdida de progreso; el **modo individual** funciona tanto en línea como sin conexión gracias a la caché en **Room**; el **modo competitivo** ofrece duelos síncronos uno-contra-uno con una latencia media inferior a 300 ms y gestiona correctamente las vidas y el resultado; por último, el usuario dispone de un **historial básico y un ranking semanal** que reflejan su evolución. Estos hitos demuestran el dominio práctico de Kotlin, Jetpack Compose, Firebase y persistencia local, tal como se planteaba en los objetivos originales.

## **21.2 Limitaciones y trabajo futuro**

La versión entregada es un MVP estable, pero presenta límites claros: el banco de preguntas aún es reducido y no incluye ejercicios de *listening*; la motivación se basa solo en vidas y ranking, sin logros ni recompensas visuales; la validación se ha hecho con un grupo piloto pequeño y la distribución sigue siendo manual mediante APK. Para las siguientes versiones se propone (i) ampliar el contenido con nuevas tipologías de ejercicio, (ii) incorporar un sistema de logros que refuerce la constancia, (iii) automatizar la entrega con un flujo CI/CD y publicación en el canal cerrado de Google Play, y (iv) realizar pruebas de usabilidad con un número mayor de usuarios a fin de refinar la experiencia y la accesibilidad.

# **22. RELACIÓN CON LOS ESTUDIOS CURSADOS**

## 

**Programación.** La lógica de *GameGlish* está escrita en Kotlin siguiendo principios de **POO**, corutinas(multihilo) y patrón **MVVM**, todos ellos trabajados en las asignaturas de Programación y Entornos de Desarrollo y Desarrollo Multimedia. El uso de Android Studio, la depuración con Logcat y la escritura de pruebas unitarias reproducen ejercicios vistos en clase, pero aplicados a un proyecto completo.

**Bases de datos.** El binomio **Room + Firebase Realtime DB** traslada al entorno móvil los conceptos de modelos entidad–relación, transacciones y sincronización que se estudiaron en Bases de Datos. La caché local, las migraciones y la reconciliación nube–dispositivo son extensiones prácticas de esos contenidos teóricos.

**Desarrollo de interfaces.** Las pautas de Material 3, la navegación con Compose y la adaptación a modo oscuro provienen de lo aprendido en Diseño de Interfaces y UX. Diseñar cada pantalla como un @Composable y mantener la accesibilidad (texto escalable, contrastes) pone en práctica esos conocimientos.

**Desafíos y aprendizajes.** Fuera del temario entran la autenticación con Firebase Auth, la latencia en tiempo real del duelo y la inyección de dependencias con Hilt. Investigar estas tecnologías y combinarlas con lo ya estudiado ha sido el principal aprendizaje extra, demostrando capacidad de autoformación y adaptación.

En conjunto, el proyecto integra programación, bases de datos y diseño de interfaz en un producto funcional, mostrando cómo los conocimientos del ciclo D.A.M. se aplican para resolver un problema real: motivar el aprendizaje de inglés mediante una aplicación Android.

# **23. REFERENCIAS**

**Android Studio. Meerkat | 2024.3.2.** <https://developer.android.com/studio/releases>

**Android Developers. (2025a)**. Android Gradle Plugin 8.8 y Build Tools 35 — Guía y notas de versión.<https://developer.android.com/studio/releases>

**Android Developers. (2025b)**. Kotlin Android & Compose plugins (plugin org.jetbrains.kotlin.android y kotlin-compose).<https://developer.android.com/jetpack/compose/setup>

**JetBrains. (2025). Kotlin 2.1** — documentación oficial (corutinas y extensión kotlinx-coroutines-android 1.7.3).<https://kotlinlang.org/docs>

**Android Developers. (2025c)**. AndroidX Core-ktx 1.15, Activity-compose 1.9 y Lifecycle 2.6 — Guía de APIs KTX y ciclo de vida.<https://developer.android.com/jetpack>

**Android Developers. (2025d)**. Jetpack Compose BOM 2024-04, UI, Material 3 y Material-icons-extended 1.5.4 — referencia de componentes.<https://developer.android.com/jetpack/compose>

**Android Developers. (2025e)**. Navigation-Compose 2.5.3 — navegación declarativa.<https://developer.android.com/guide/navigation>

**Firebase. (2025a)**. Firebase BOM 31.1.0, Auth-ktx 23.1.0 y Realtime Database-ktx 21.0.0 — autenticación y base de datos en tiempo real.<https://firebase.google.com/docs>  
**Android Developers. (2025f)**. Room 2.6.1 & KSP — persistencia local con room-runtime, room-ktx, room-compiler.<https://developer.android.com/jetpack/androidx/releases/room>

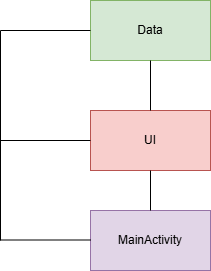
**Android Developers. (2025g).** Espresso 3.6.1 y JUnit 4.13.2 / 1.2.1 — pruebas de UI y de unidad.<https://developer.android.com/training/testing>

**Google.** (2025). Material Design 3 — directrices de diseño empleadas en GameGlish. <https://m3.material.io>

# **24. ANEXOS**

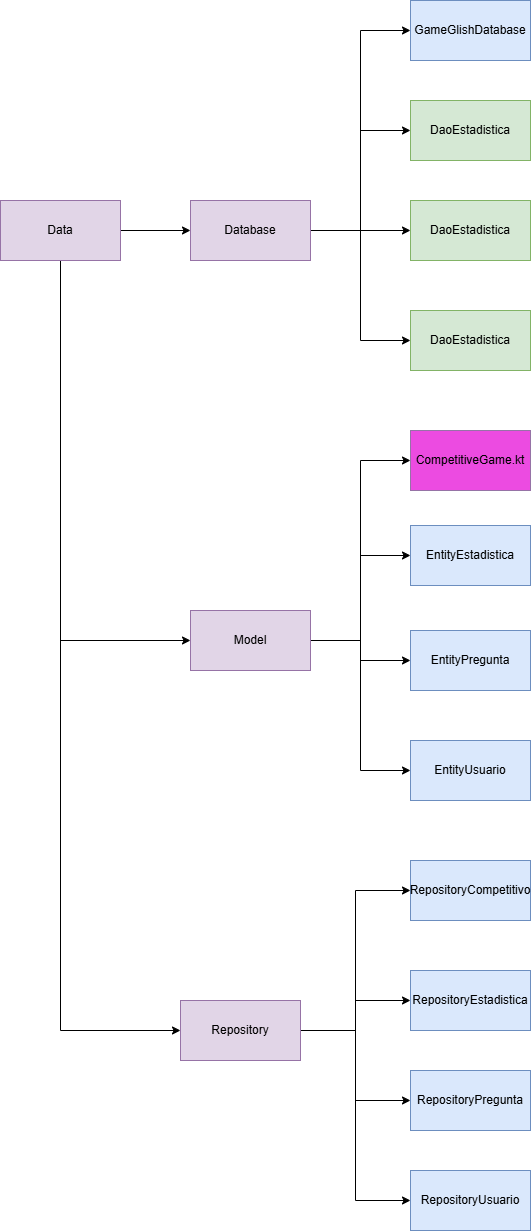
## **24.1 Estructura de Directorios**

### **24.1.1 Módulos raíz**



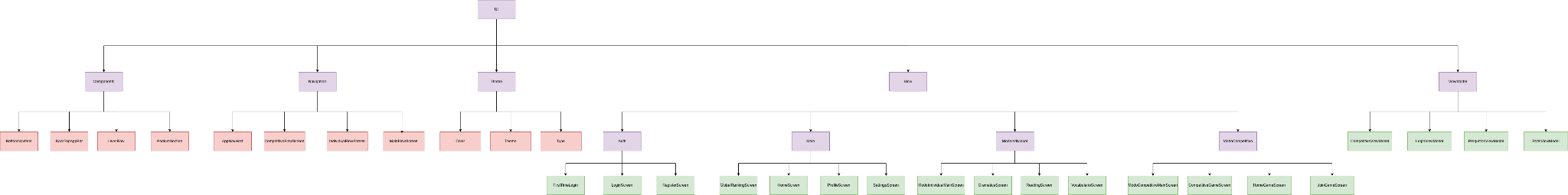
*Ilustración 8 Módulo Raíz*

### **24.1.2 Módulo data**



*Ilustración 9 Módulo Data*

### **24.1.3 Módulo UI**



*Ilustración 10 Módulo UI*

## **24.2 Código Fuente (extractos)**

En este anexo se incluyen los fragmentos de código más significativos del proyecto. Se han elegido atendiendo a su relevancia para el funcionamiento del sistema, su nivel de complejidad técnica y su aporte a la resolución de los retos definidos en los objetivos del proyecto.

### **24.2.1 Modelo (Ejemplo EntityPreguntas) Entidad “Preguntas”**

@Entity(tableName = "preguntas")

data class EntityPregunta(

@PrimaryKey(autoGenerate = true) val id: Int = 0,

val enunciado: String,

val opcionA: String,

val opcionB: String,

val opcionC: String,

val opcionD: String,

val opcionCorrecta: String,

val tema: String,

val nivel: Int

)

*Ilustración 11 EntityPregunta*

Definición de la entidad **EntityPregunta**: mapea cada pregunta del banco de vocabulario y gramática a la tabla preguntas de **Room**, incluyendo enunciado, cuatro opciones, la respuesta correcta, el tema y el nivel de dificultad; el campo id se genera automáticamente y actúa como clave primaria.

### 

### **24.2.2 DAO (Ejemplo DaoPregunta) Filtro de preguntas**

@Query("SELECT \* FROM preguntas WHERE tema = :tema")

suspend fun getPreguntasByTema(tema: String): List<EntityPregunta>

@Query("SELECT \* FROM preguntas WHERE nivel = :nivel")

suspend fun getPreguntasByNivel(nivel: Int): List<EntityPregunta>

}

*Ilustración 12 DaoPregunta*

Fragmento de **DaoPregunta**: dos consultas Room que recuperan listas filtradas por tema (vocabulario, gramática…) o por nivel (A1–C2); ambas se ejecutan de forma asíncrona con suspend para no bloquear el hilo de la interfaz.

### **24.2.3 Repositorio (RepositoryUsuario). Registro**

suspend fun registrarUsuarioCorreo(

email: String,

password: String,

confirmPassword: String

): Boolean {

if (password != confirmPassword) {

return false

}

try {

val result = auth.createUserWithEmailAndPassword(email, password).await()

val uid = result.*user*?.*uid* ?: return false

val usuario = EntityUsuario(

uidFirebase = uid,

email = email,

nombre = "",

puntos = 0,

nivel = 0,

firstLogin = true

)

guardarUsuarioLocal(usuario)

withTimeoutOrNull(3000L) **{**

guardarUsuarioRemoto(usuario)

**}**

return true

} catch (e: Exception) {

e.printStackTrace()

return false

}

}

*Ilustración 13 registrarUsuarioCorreo()*

Método **registrarUsuarioCorreo** en RepositoryUsuario: crea la cuenta en Firebase Auth, construye un **EntityUsuario** inicial y lo persiste localmente en **Room**; si la red está disponible, lo replica en la nube con un límite de 3 s (withTimeoutOrNull). Devuelve true si todo fue correcto o false ante error o contraseñas no coincidentes.

### **24.2.4 ViewModel (CompetitiveGameViewModel):**

**1.observeGame(gameId):**

fun observeGame(gameId: String) {

dbRef.child(gameId).addValueEventListener(object : ValueEventListener {

override fun onDataChange(snapshot: DataSnapshot) {

val game = snapshot.getValue(CompetitiveGame::class.*java*) ?: return

\_gameState.value = game

// ── NUEVO BLOQUE ───────────────────────────────────────────

val soyHost = currentUserId == game.hostId

val joinerOk = !game.joinerId.*isNullOrEmpty*()

if (soyHost && joinerOk && game.state == "waiting") {

// 1) cambio de estado

dbRef.child(gameId).child("state").setValue("inProgress")

// 2) cargo preguntas y envío la primera (una sola vez)

*viewModelScope*.*launch* **{**

cargarYEnviarPreguntas(gameId) // ya no necesita context

**}**

}

if (!answersListenerAdded) {

escucharRespuestas(gameId)

answersListenerAdded = true

}

// ──────────────────────────────────────────────────────────

}

override fun onCancelled(error: DatabaseError) {

Log.e("CompetitiveGameVM", "observeGame cancel: ${error.*message*}")

}

})

}

*Ilustración 14 observeGame()*

**observeGame(gameId)** en **CompetitiveGameViewModel**: escucha en tiempo real el nodo /competitivo/games/{id}; actualiza el StateFlow con el estado de la partida y, si el usuario es el host y ya hay rival, pasa de waiting a inProgress y envía la primera pregunta; además registra un único listener para las respuestas del duelo (escucharRespuestas). Así se coordina el flujo competitivo sin lógica en la UI.

**2.cerrarRonda(gameId)**

private fun cerrarRonda(gameId: String) {

if (rondaCerrada) return

rondaCerrada = true

countDownTimer?.cancel()

dbRef.child(gameId).runTransaction(object : Transaction.Handler {

override fun doTransaction(m: MutableData): Transaction.Result {

val correct = m.child("correctId").getValue(String::class.*java*) ?: ""

val hostAns = m.child("answers").child("host").getValue(String::class.*java*)

val joinAns = m.child("answers").child("joiner").getValue(String::class.*java*)

var hostLives = m.child("hostLives").getValue(Int::class.*java*) ?: 3

var joinLives = m.child("joinerLives").getValue(Int::class.*java*) ?: 3

if (hostAns != correct) hostLives--

if (joinAns != correct) joinLives--

m.child("hostLives").*value* = hostLives

m.child("joinerLives").*value* = joinLives

// Fin de partida

if (hostLives == 0 || joinLives == 0) {

m.child("state").*value* = "finished"

m.child("winner").*value* = when {

hostLives > joinLives -> m.child("hostId").*value*

joinLives > hostLives -> m.child("joinerId").*value*

else -> "draw"

}

return Transaction.success(m)

}

// Siguiente pregunta

indicePreguntaActual++

val preguntasLeft = \_preguntas.size - indicePreguntaActual

if (preguntasLeft == 0) { // sin más preguntas → empate

m.child("state").*value* = "finished"

m.child("winner").*value* = "draw"

return Transaction.success(m)

}

val next = \_preguntas[indicePreguntaActual]

m.child("currentQuestion").*value* = next.question

m.child("answerOptions").*value* = next.options

m.child("correctId").*value* = next.correctId

m.child("timeLeft").*value* = 20

m.child("answers").*value* = null // limpia respuestas

return Transaction.success(m)

}

override fun onComplete(e: DatabaseError?, committed: Boolean, snap: DataSnapshot?) {

/\* 2. Si sigue la partida, reinicia flag y timer \*/

if (committed && \_gameState.value.state == "inProgress") {

rondaCerrada = false

arrancarTimer(gameId)

}

}

})

}

*Ilustración 15 CerrarRonda()*

**cerrarRonda(gameId)** en **CompetitiveGameViewModel**: método exclusivo del host que cierra cada ronda dentro de una transacción de Firebase.

Detiene el cronómetro y lee la respuesta correcta y las enviadas por ambos jugadores; 2) actualiza las vidas según acierto o error; 3) decide si la partida termina (establece state = finished y calcula el ganador) o pasa a la siguiente pregunta reponiendo temporizador y limpiando respuestas. La transacción garantiza que todos los cambios (vidas, estado, pregunta siguiente) se apliquen de forma atómica y coherente para los dos clientes.

### **24.2.5 Navegación (AppNavHost)**

@Composable

fun AppNavHost(

navController: NavHostController,

loginViewModel: LoginViewModel

) {

val isUserLoggedIn by loginViewModel.isUserLoggedIn.collectAsState()

val isFirstLogin by loginViewModel.isFirstLogin.collectAsState()

// Si el usuario está logueado:

// - Si es primer login, queremos mostrar el flujo de autenticación (authFlow)

// - Si no, mostramos mainFlow.

val startDestination = if (isUserLoggedIn == true) {

if (isFirstLogin == true) "authFlow" else "mainFlow"

} else {

"authFlow"

}

NavHost(navController = navController, startDestination = startDestination) **{**

// Flujo de autenticación (destino directo del NavHost principal)

*navigation*(

// Seleccionamos dinámicamente el startDestination según el flag firstLogin:

startDestination = if (isUserLoggedIn == true && isFirstLogin == true)

Screen.FirstTimeLogin.route

else

Screen.Login.route,

route = "authFlow"

) **{**

*composable*(Screen.Login.route) **{**

LoginScreen(

navController = navController,

onNavigateToRegister = **{** navController.navigate(Screen.Register.route) **}**

)

**}**

*composable*(Screen.Register.route) **{**

RegisterScreen(

onRegisterSuccess = **{**

navController.navigate("authFlow") **{**

popUpTo("authFlow") **{** inclusive = true **}**

**}**

**}**

)

**}**

*composable*(Screen.FirstTimeLogin.route) **{**

FirstTimeLoginScreen(

onRegisterSuccess = **{**

navController.navigate("mainFlow") **{**

popUpTo(Screen.FirstTimeLogin.route) **{** inclusive = true **}**

**}**

**}**,

registerUser = **{** nombre, nivelSeleccionado **->**

val uid = FirebaseAuth.getInstance().*currentUser*?.*uid*

if (uid != null) {

loginViewModel.actualizarNombreUsuario(uid, nombre, nivelSeleccionado)

loginViewModel.marcarPrimerLoginCompleto()

navController.navigate("mainFlow") **{**

popUpTo(Screen.FirstTimeLogin.route) **{** inclusive = true **}**

**}**

}

**}**

)

**}**

**}**

// Flujo principal (con Bottom Navigation)

*composable*("mainFlow") **{**

MainFlowScreen(globalNavController = navController)

**}**

*composable*(Screen.Settings.route) **{**

SettingsScreen(navController) // Asegúrate de que exista tu SettingsScreen

**}**

**}**

}

*Ilustración 16 AppNavHost*

**AppNavHost** es el grafo de navegación raíz en Compose: consulta los flujos isUserLoggedIn e isFirstLogin del LoginViewModel para fijar de forma reactiva el destino inicial— authFlow si el usuario no ha terminado su alta o mainFlow en caso contrario. El sub-grafo authFlow arranca en LoginScreen, RegisterScreen o FirstTimeLoginScreen y, tras completar el proceso, vacía la pila y redirige a mainFlow; este segundo flujo contiene la aplicación principal con barra inferior (MainFlowScreen) y la ruta extra a SettingsScreen.

### **24.2.6 Componente UI (LivesRow)**

@Composable

fun LivesRow(lives: Int) {

Row **{**

*repeat*(3) **{** index **->**

val filled = index < lives

Icon(

imageVector = if (filled) Icons.Filled.*Favorite* else Icons.Outlined.*FavoriteBorder*,

contentDescription = "Vida ${index+1}",

tint = if (filled) Color.Red else Color.Gray,

modifier = Modifier.*padding*(end = 4.*dp*)

)

**}**

**}**

}

*Ilustración 17 LivesRow.kt*

**LivesRow** – componente Compose reutilizable que visualiza las “vidas” restantes en el duelo.

Dibuja una fila de tres iconos de corazones; para cada posición decide el vector y el color según el parámetro lives. De este modo la UI se actualiza de forma reactiva cada vez que el StateFlow del ViewModel cambia la cantidad de vidas.

## **24.3 Manual de usuario**

Versión 1.0 – mayo 2025

**1 · ¿Qué es GameGlish?**

GameGlish es una aplicación Android que convierte el estudio de inglés en un reto breve y divertido. Combina:

* **Modo individual** sin conexión, con preguntas de vocabulario y gramática adaptadas a tu nivel.
* **Duelo 1 × 1** en tiempo real: compite contra otro jugador y mantén tus tres vidas.
* **Historial y ranking global**: revisa tu progreso y comprueba tu posición frente al resto.

La app funciona sobre Kotlin + Jetpack Compose, guarda los datos localmente con **Room** y los sincroniza en la nube mediante **Firebase**.

**2 · Requisitos mínimos**

| **Recurso** | **Especificación** |
| --- | --- |
| **Sistema operativo** | Android 8.0 (Oreo) o posterior |
| **CPU / RAM** | Procesador de 4 núcleos, 2 GB de RAM |
| **Conexión** | Wi-Fi o 4G recomendados para duelos; no necesaria para modo individual aunque si para Autenticación |
| **Almacenamiento** | 40 MB libres para la APK y la base local |

**3 · Instalación**

1. Descarga el archivo **GameGlish-1.0.apk** desde el enlace facilitado por tu centro.
2. En el dispositivo ve a **Ajustes ▸ Seguridad** y activa *Fuentes desconocidas* (solo la primera vez).
3. Toca el APK y sigue el asistente.
4. Una vez instalada, abre **GameGlish** desde el cajón de aplicaciones.

**4 · Primer inicio**

| **Pantalla** | **Qué debes hacer** |
| --- | --- |
| **Registro / Login** | Crea una cuenta con correo y contraseña o accede si ya tienes una. |
| **Datos de perfil** | Indica tu nombre y selecciona tu nivel aproximado (A1–C2). |
| **Test inicial** | Responde 20 preguntas para que la app calibre tu nivel real. |

**5 · Navegación general**

| **Inicio** | Información de nivel, barra de progreso, consejos. |
| --- | --- |
| **Individual** | Modo individual offline. |
| **Competitivo** | Crear / Unirse a partida competitiva. |
| **Ranking** | Ranking global. |
| **Perfil** | Información sobre la aplicación. Historial de partidas. |

**6 · Modo individual (offline)**

1. Pulsa **Practicar** → elige un *Tema* (Gramática, Vocabulario…).
2. La app descarga —la primera vez— las preguntas y las guarda en local.
3. Cada ronda muestra una pregunta y cuatro opciones; toca tu respuesta antes de que el círculo llegue a 0.
4. Al finalizar verás tus aciertos, errores y consejo de repaso.
5. Si no tienes conexión, la sesión se guarda y se sincronizará automáticamente al recuperar red.

**7 · Modo competitivo(online)**

| **Crear partida** | **Unirse a partida** |
| --- | --- |
| Pulsa **Crear** → se genera un código. Comparte el código con tu rival. | Pulsa **Unirse** → escribe el código que te pase el creador o haz tap en la sala que quieras disponible. |

* Cuando ambos estén listos, comienza la cuenta atrás de 20 s.
* Cada error resta una vida. Gana quien conserve al menos 1 vida cuando el rival se quede sin.
* La latencia media es ≈ 220 ms en Wi-Fi; si la red cae la partida se pausa hasta 30 s.

**8 · Progreso y ranking**

* **Historial**: lista de todas tus sesiones y duelos con fecha, tema y puntuación.
* **Ranking global**: tabla con los mejores usuarios.

**9 · Preguntas frecuentes**

| **Duda** | **Respuesta breve** |
| --- | --- |
| ¿Cómo pongo el modo oscuro? | Simplemente cambia tu tema en el teléfono y se cambiará automáticamente. |
| ¿Puedo agregar a amigos para jugar contra ellos? | Actualmente la aplicación no dispone de sistema de amigos integrado. |
| ¿Puedo borrar mi cuenta? | Si, si escribes un mail a nuestro equipo borraremos tu cuenta y todos los datos asociados salvo la puntuación del ranking. |

**10 · Solución de problemas**

| **Problema** | **Qué hacer** |
| --- | --- |
| **Error de red en duelo** | Cambia a una red Wi-Fi estable; la partida se reanuda si vuelves en 30 s. |
| **Nivel no se actualiza** | Reinicia la app y vuelve a logearte. |
| **Bloqueo o cierre inesperado** | Envía un correo a nuestro equipo con el problema en cuestión e intentaremos arreglarlo lo más rápido posible. |

**11 · Contacto y soporte**

*Correo*: javihuelamo97@gmail.com  
*Repositorio Git (código fuente)*: https://github.com/vasilon/gameglish (versión lectura)  
*Documentación técnica*: ver Anexo 24 del TFG.

¡Disfruta aprendiendo y compitiendo con GameGlish!

# 

# **25. GLOSARIO**

| **Término** | **Definición breve (en el contexto de GameGlish)** |
| --- | --- |
| **APK** | Archivo empaquetado que contiene la app Android lista para instalarse en un dispositivo. |
| **Corutina** | Función asíncrona de Kotlin que permite operaciones en segundo plano sin bloquear la interfaz. |
| **DAO** (Data Access Object) | Interfaz de Room que declara consultas SQL y mapea resultados a objetos Kotlin. |
| **Firebase Auth** | Servicio de autenticación de Google usado para registrar e iniciar sesión con correo-contraseña. |
| **Firebase Realtime DB** | Base de datos en la nube que sincroniza en milisegundos perfiles, partidas y ranking. |
| **First Login** | Bandera que indica si el usuario acaba de crear cuenta y necesita completar nombre + nivel. |
| **Jetpack Compose** | Toolkit declarativo de UI utilizado para todas las pantallas e iconos Material 3 de la app. |
| **KSP** (Kotlin Symbol Processing) | Procesador que genera el código de Room a partir de anotaciones @Entity y @Dao. |
| **LoginState** | enum con los valores *Idle*, *Loading*, *Success* y *Error* que la interfaz observa para mostrar progreso de acceso. |
| **MVVM** | Patrón *Model-View-ViewModel* que separa datos, lógica de presentación y UI. |
| **Repository** | Capa intermedia que decide si leer/escribir en Room o en Firebase y oculta esa decisión a la UI. |
| **Room** | Librería de persistencia que gestiona la base de datos SQLite local (preguntas y sesiones offline). |
| **StateFlow** | Flujo de datos inmutable que emite estados y al que los composables se suscriben para recomponerse. |
| **UID** | Identificador único de usuario generado por Firebase Auth; sirve como clave en la nube y en local. |