



# **Proyecto**

## **PSP-PDM 2**

## **TRIMESTRE**

**José Manuel Piñero Ávila  
Luciana Perez Martinez**



# Índice

<b>Introducción</b>	<b>4</b>
Descripción General de QuizzHub	4
Objetivos del Proyecto	4
Funcionalidades Principales	5
<b>Diseño de la aplicación</b>	<b>6</b>
Bocetos y estructura de algunas pantallas	6
Pantalla de Inicio	9
<b>Identificación de funcionalidades</b>	<b>10</b>
Modo multijugador	10
Selección de categorías	11
Temporizador	12
<b>Estructura de base de datos</b>	<b>14</b>
<b>Configuración del entorno de Desarrollo</b>	<b>16</b>
Herramientas utilizadas (Android Studio, Firebase, Github, etc..)	16
Configuración inicial del proyecto	18
Instalación de dependencias y librerías necesarias	19
<b>Desarrollo de funcionalidades básicas</b>	<b>20</b>
Implementación del sistema de preguntas y respuestas	20
Diseño de interfaz gráfica y experiencia de usuario	23



<b>Implementación de funcionalidades avanzadas</b>	<b>30</b>
Desarrollo del modo multijugador con sockets	30
<b>Conclusiones</b>	<b>33</b>
Resultados obtenidos	34
Posibles mejoras y futuras actualizaciones	34
Reflexión sobre el desarrollo del proyecto	35



# Introducción

## Descripción General de Quizzhub

QuizzHub es una aplicación de preguntas y respuestas diseñada para desafiar el conocimiento de los usuarios en diversas categorías. Su principal atractivo es el modo multijugador, donde los jugadores pueden competir en tiempo real respondiendo preguntas dentro de un tiempo límite. La aplicación está diseñada para dispositivos Android y se centra en ofrecer una experiencia de usuario intuitiva y atractiva.

## Objetivos del Proyecto

El objetivo que tenemos con el desarrollo de QuizzHub es proporcionar una plataforma de entretenimiento educativo que permita a los usuarios aprender y competir de manera dinámica. Algunos de los objetivos específicos del proyecto incluyen:

- Desarrollar una aplicación eficiente y bien optimizada para dispositivos Android.
- Implementar un sistema de preguntas y respuestas con selección de categorías y un temporizador.
- Incluir un modo multijugador con conectividad en red para desafiar a otros jugadores.
- Garantizar seguridad en el almacenamiento y transmisión de datos.
- Diseñar una interfaz intuitiva basada en principios de usabilidad.



## Funcionalidades Principales

QuizzHub cuenta con diversas funcionalidades clave que permiten una experiencia de juego fluida y atractiva. Entre ellas se encuentran:

- **Modo individual y multijugador:** Posibilidad de jugar solo o competir con otros jugadores en partidas en tiempo real.
- **Selección de categorías:** Los usuarios pueden elegir entre diferentes temas de preguntas para personalizar su experiencia de juego.
- **Temporizador:** Cada pregunta cuenta con un tiempo límite para responder, aumentando la dificultad y dinamismo.
- **Sistema de puntuaciones:** Los jugadores obtienen puntos en función de la rapidez y precisión de sus respuestas.
- **Almacenamiento de preguntas en Firebase:** Base de datos que permite gestionar preguntas de manera dinámica y escalable.
- **Interfaz atractiva y accesible:** Diseño pensado para mejorar la experiencia del usuario con una navegación sencilla.

Todas esas funcionalidades crean la experiencia de juego fluida que queremos conseguir con QuizzHub desde un principio, ya sea en solitario o junto con tus amigos.



# Diseño de la Aplicación

## Boceto y estructura de Pantallas

Hemos realizado los bocetos de las Activitys usando la herramienta **Figma**.

En ella hemos podido crear y dar forma a los Wireframes de Quizzhub. Junto con el diseño del flujo principal y el diseño de interfaz.

Es aquí donde comienza todo.

Las pantallas siguen un flujo principal que sería el siguiente:

**Login Activity:** El usuario hace login con sus credenciales registradas en la base de datos.

**Register Activity:** El usuario registra sus credenciales si carece de credenciales anteriormente registradas en la base de datos.

**Menú Principal Activity:** El usuario una vez logueado, permanece en el menú principal en el que podrá navegar por distintas funciones o botones que le llevarán a otras Activities.

**Jugar Activity:** El jugador hace clic en el botón jugar y elegirá entre dos modos de juego (Contrarreloj, que es juego en solitario o Sala Online). Este último le llevará a **Sala Online Activity**, en el que podrá generar un número de sala que Firebase generará para él e introducirlo en el campo de texto para unirse a la sala.

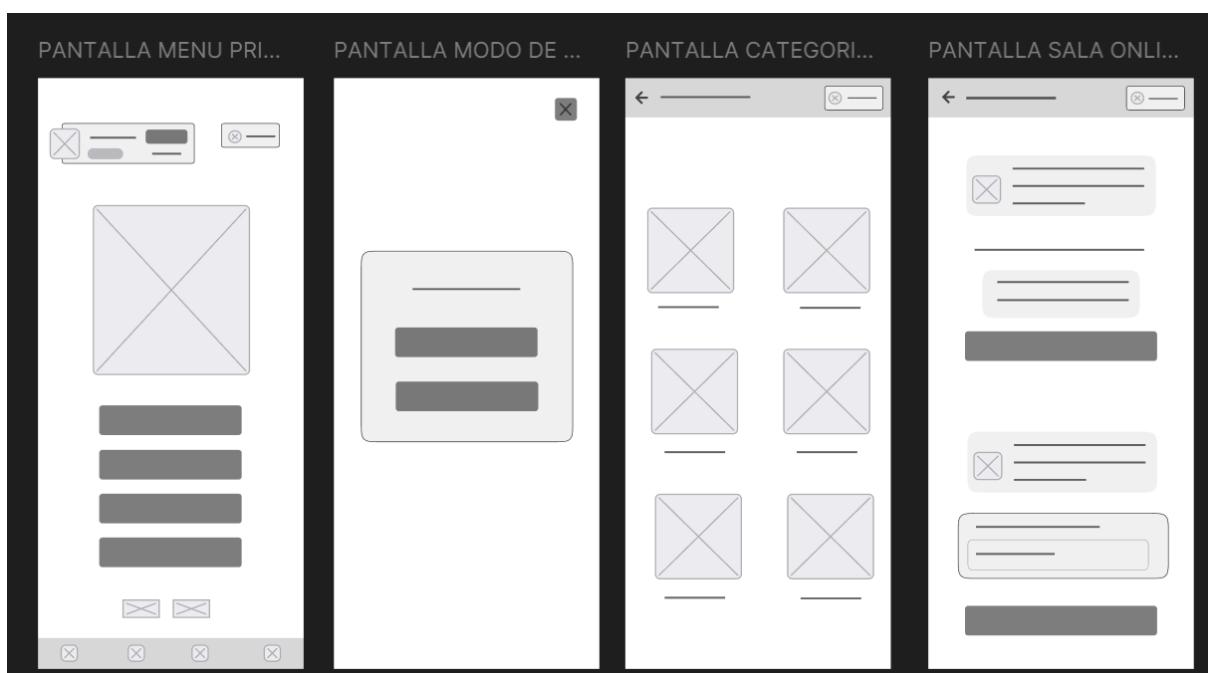


Una vez se unan los jugadores necesarios para iniciar la partida, el flujo seguiría en **Partida Activity**, en los que los jugadores van a jugar una partida en la que tiene que responder preguntas para puntuar .

Existe un **temporizador de 30 segundos por pregunta** y un marcador individual con la puntuación de cada uno.

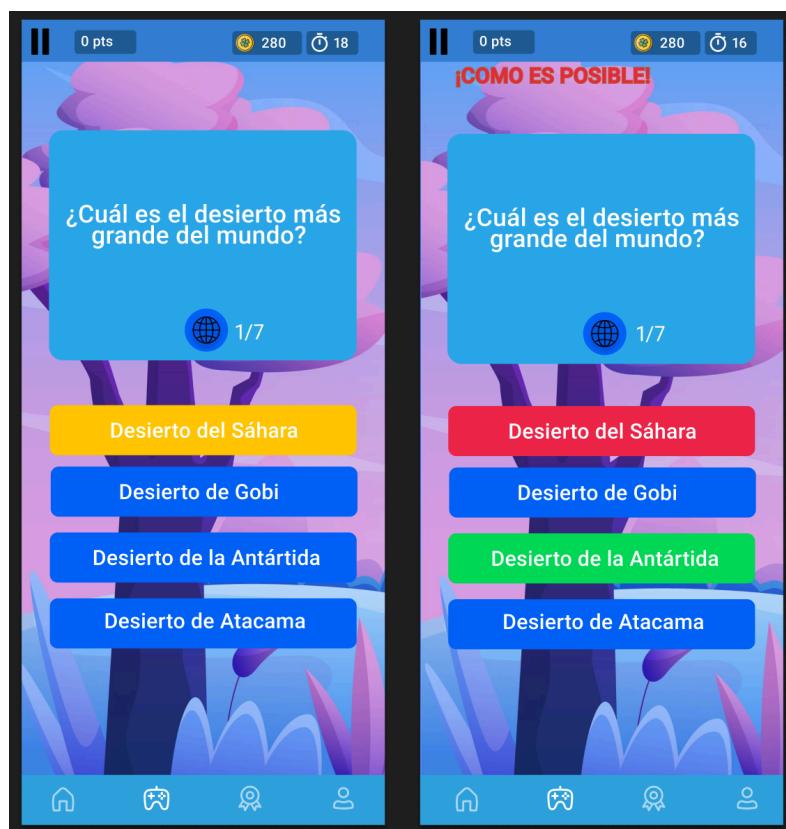
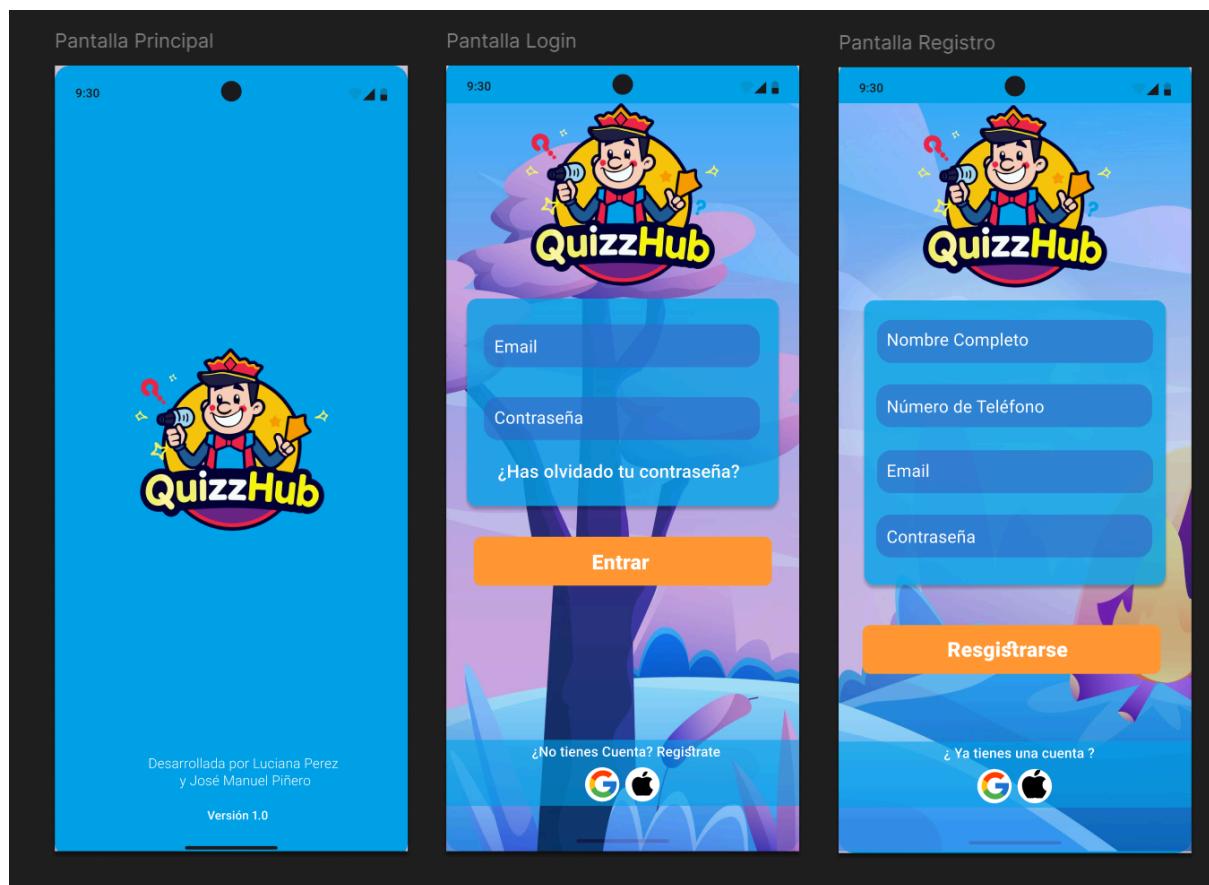
Al terminar la partida, saldría el ranking de la partida ordenado por puntuación. Es aquí donde terminaría el **flujo principal** de QuizzHub.

A continuación algunos Wireframes de QuizzHub a modo de vistazo general, junto con alguna vista de Activity.



Vamos a ver ahora algunas Activities, las de Login, Registro, Menú Principal, Partida....

En el mismo orden que la acabamos de mencionar.





## Pantalla de Inicio

Cuando iniciamos la Aplicación se muestra Un Splash Screen , en el que se muestra el logo de la Aplicación junto con información adicional de tipo de versión y desarrollada por...

Imagen de Splash Screen





# Identificación de funcionalidades

## Modo multijugador

El modo multijugador de QuizzHub permite a los jugadores competir en tiempo real en una sala de juego. Un usuario actúa como **host** y crea la sala, mientras que otros jugadores pueden unirse mediante un código de acceso o una búsqueda de salas disponibles.

### Características del modo multijugador:

- **Conexión en tiempo real:** Se utilizarán **sockets** para la comunicación entre jugadores.
- **Sincronización de preguntas:** Todos los jugadores recibirán la misma pregunta al mismo tiempo. En este caso está configurado de forma aleatoria.
- **Turnos y tiempo límite:** Cada jugador tiene un tiempo determinado para responder, y una vez finalizado el turno, se muestra el ranking de la partida con las puntuaciones de cada jugador.
- **Sistema de puntuación:** Se otorgan puntos en función de la rapidez y exactitud de las respuestas.
- **Cierre de la partida:** Al finalizar todas las preguntas, se muestra la clasificación final con las puntuaciones de cada jugador.



## Flujo del modo multijugador:

1. El **host** crea la sala y espera a que los demás jugadores se unan.
2. Cuando se completa el número mínimo de jugadores, el host inicia la partida.
3. Todos los jugadores responden a la serie de preguntas dentro del tiempo límite.
4. Se actualiza la puntuación después de cada pregunta.
5. Al finalizar la partida, se muestra el ranking con los resultados.

## Selección de categorías (No Implementado, siguiente versión)

Antes de comenzar una partida, los jugadores van a poder seleccionar la categoría de preguntas que desean responder. Esto permite personalizar la experiencia de juego según sus preferencias o conocimientos.

Esta función está implementada solo de forma visual , en la pantalla categorias pero no ha sido implementada aún ya que no lo requería. Será implementada en siguientes versiones , junto con mostrar de qué categoría pertenece cada pregunta que nos aparece en la interfaz de partida.

## Características de la selección de categorías:

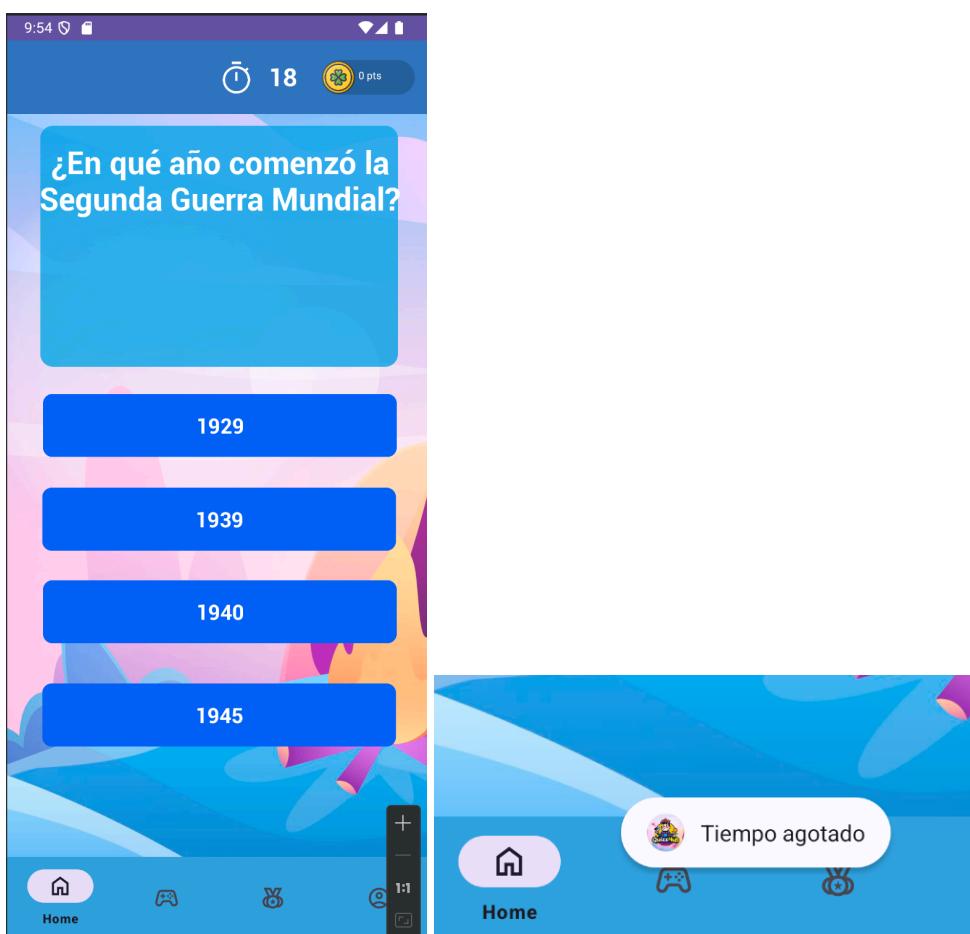
- **Diferentes temas de preguntas:** Historia, Ciencia, Deportes, Entretenimiento, Tecnología, entre otros.
- **Filtrado de preguntas:** La base de datos mostrará solo las preguntas de la categoría elegida.



- **Modo aleatorio:** Si el jugador lo prefiere, puede seleccionar un modo aleatorio que combine preguntas de todas las categorías.
- **Disponibilidad en ambos modos:** Funcionalidad disponible en modo individual y multijugador.

## Temporizador

El temporizador es una funcionalidad esencial en **QuizzHub** que añade un elemento de presión y estrategia al juego. Cada pregunta tiene un tiempo límite dentro del cual los jugadores deben responder. Hemos configurado , 30 segundos desde que se muestra la pregunta.





## Características del temporizador:

- **Indicador visual:** Un contador numérico muestra el tiempo restante.
- **Respuesta automática:** Si un jugador no responde antes de que termine el tiempo, se registra como incorrecto y pasa a la siguiente pregunta.
- **Penalización por tardanza:** Se pueden otorgar más puntos a quienes respondan más rápido para incentivar la agilidad.

## Flujo del temporizador:

1. Se muestra la pregunta junto con el temporizador en pantalla.
2. El contador de tiempo comienza a disminuir hasta llegar a cero.
3. Si el jugador responde antes de que el tiempo se agote, se valida la respuesta y se asignan puntos.
4. Si el tiempo se acaba sin respuesta, se marca como incorrecto y se pasa a la siguiente pregunta.

El temporizador no solo hace el juego más dinámico, sino que también aumenta la competencia en el modo multijugador al premiar la rapidez.



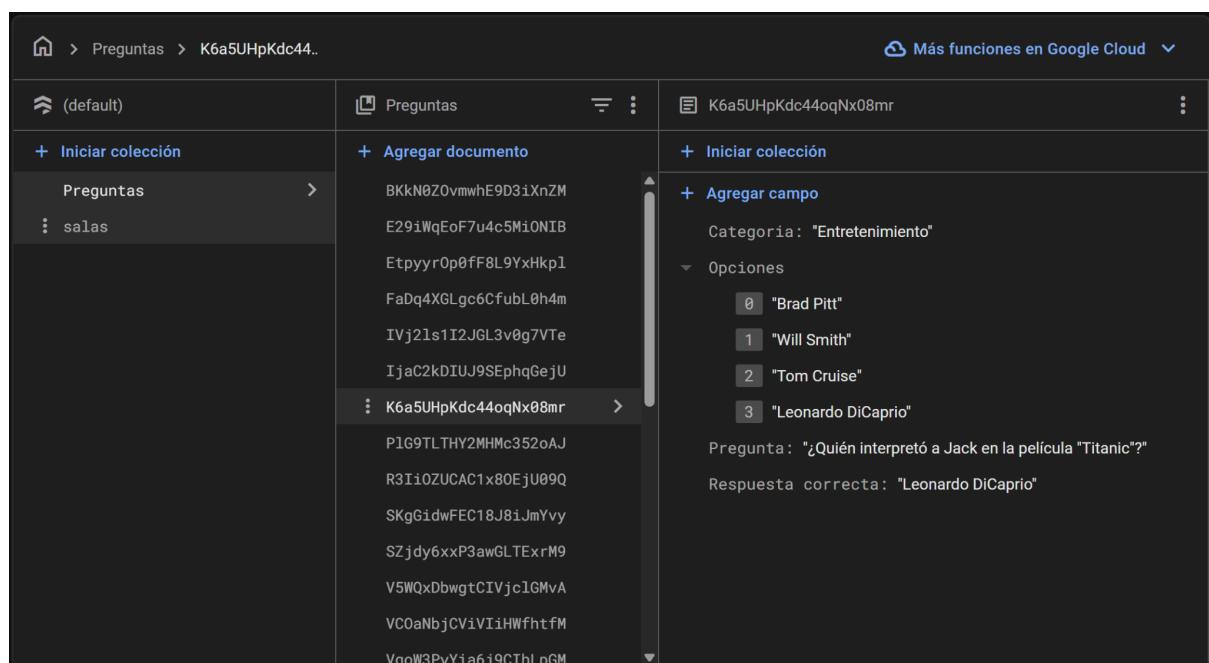
# Estructura de base de datos

Para la estructura de base de datos hemos utilizado Firebase. Una vez creado nuestro proyecto en Firebase se configura correctamente el Authenticator y la base de datos para las preguntas y crear la sala y almacenar a los jugadores. Antes de todo esto es necesario configurar correctamente Firebase con nuestro proyecto en Android Studio para que este se conecte de forma correcta en la red con nuestro proyecto Firebase.

Así podremos ver los cambios o las analíticas en tiempo real en la consola de Firebase que nos muestra la interfaz en Google.



A continuación mostramos cómo es la estructura de la base de datos que contiene las preguntas en QuizzHub , dentro de nuestro proyecto Firebase.



The screenshot displays the Firebase Realtime Database interface. On the left, the database structure is shown with a tree view:

- (default)
- + Iniciar colección
- Preguntas >
- : salas

In the center, under the "Preguntas" collection, there is a list of document IDs:

- BKkN0Z0vmwhE9D3iXnZM
- E29iWqEoF7u4c5MiONIB
- EtpyyrOp0fF8L9YxHkp1
- FaDq4XGLgc6CfubL0h4m
- IVj2ls1I2JL3v0g7VTe
- IjaC2kDIUJ9SEphqGejU
- K6a5UHpKdc44oqNx08mr >
- P1G9TLTHY2MHMc352oAJ
- R3IiOZUCAC1x80EjU09Q
- SKgGidwFEC18J8iJmYvy
- SZjdy6xxP3awGLTExrM9
- V5WQxDbwgtCIVjc1GMvA
- VCOaNbjCViVIiHWfhtfM
- VqoW3PyYia6j9CIhLpGM

On the right, the details for the document "K6a5UHpKdc44oqNx08mr" are expanded:

- + Iniciar colección
- + Agregar campo
- Categoría: "Entretenimiento"
- Opciones
  - 0 "Brad Pitt"
  - 1 "Will Smith"
  - 2 "Tom Cruise"
  - 3 "Leonardo DiCaprio"
- Pregunta: "¿Quién interpretó a Jack en la película "Titanic"?"
- Respuesta correcta: "Leonardo DiCaprio"



En **Firestore**, hemos utilizado una colección principal llamada "**Preguntas**", donde cada documento representa una pregunta con los siguientes campos:

## Colección: Preguntas

Cada documento tendrá un ID generado automáticamente y contendrá la siguiente información:

- **Pregunta (String)**: Texto de la pregunta.
- **Opciones (Array de Strings)**: Cuatro opciones de respuesta.
- **Respuesta Correcta (String)**: La opción correcta de la pregunta
- **Categoría (String)**: Tema al que pertenece la pregunta.

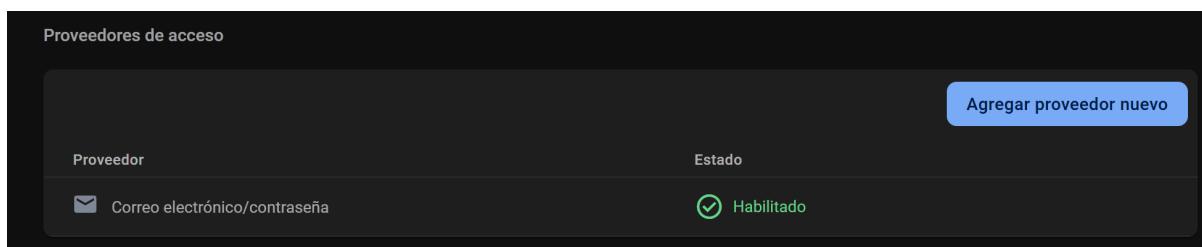
## Base de datos para Usuario

Para los usuarios que se registren desde la interfaz de la App, vamos a almacenarlos en FireBase Authenticator, en la base de datos que nos ofrece. Antes de nada hemos debido de configurar correctamente esta herramienta, activando que la forma de loguearse sea a través de correo electrónico.

Aunque se pueden muchas más:

- Gmail
- Icloud
- FaceBook
- etc..

Aquí una visión general de la base de datos que contiene los usuarios registrados hasta ahora, junto con el método de acceso configurado para la app (Correo electrónico)



The screenshot shows a table titled "Proveedores de acceso" (Access Providers) with two columns: "Proveedor" (Provider) and "Estado" (Status). The provider listed is "Correo electrónico/contraseña" (Email/Password), which is marked as "Habilitado" (Enabled). A blue button at the top right says "Agregar proveedor nuevo" (Add new provider).

Proveedor	Estado
Correo electrónico/contraseña	Habilitado



Buscar por dirección de correo electrónico, número de teléfono o UID de usuario					Agregar usuario	G	⋮
Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario			
lucisimulacion@gmail.c...	✉	6 mar 2025	6 mar 2025	2Im5fAVk1xNEbTZi0Os3UNLx...			
josesimulacion@gmail....	✉	6 mar 2025	6 mar 2025	1NuvcZfo31UN1JpNfiQDxpad...			
pepe@medac.es	✉	13 feb 2025	13 feb 2025	Hmrs4aKc1NTJjnCGurHLOcc...			
lucimedac@gmail.com	✉	13 feb 2025	13 feb 2025	5yXVz7tSYVddxmGblgmWD7x...	⠇	⋮	
josemedac@gmail.com	✉	13 feb 2025	13 feb 2025	VGWbuw3T8rglzmek1vnL0UN...			

# Configuración del entorno de Desarrollo

Para el desarrollo de QuizHub, lo hemos configurado un entorno adecuado para garantizar un flujo de trabajo eficiente y estructurado. A continuación, detallamos las herramientas utilizadas, la configuración inicial del proyecto y la instalación de las dependencias necesarias.

## Herramientas Utilizadas

Para el desarrollo de QuizHub, se han empleado diversas herramientas que permiten la gestión del código, la base de datos y la conectividad en red.

### Android Studio

*Función: Entorno de desarrollo (IDE) para la creación de aplicaciones Android.*

Características clave: Emulador integrado para pruebas. Compatibilidad con Kotlin y Java. Integración con Firebase. Gestión de dependencias con Gradle.



## Firebase Firestore

*Función: Base de datos en la nube para almacenar preguntas y puntuaciones.*

Características clave: Almacenamiento en tiempo real y escalable. Soporte para consultas indexadas. Sincronización en tiempo real con listeners.

## Firebase Authentication

*Función: Sistema de autenticación para gestionar el inicio de sesión de los usuarios.*

Métodos utilizados: Autenticación con correo y contraseña. Inicio de sesión con Google.

## GitHub

*Función: Plataforma para el control de versiones y colaboración en el desarrollo del proyecto.*

Flujo de trabajo: Repositorio creado en GitHub. Commits frecuentes para asegurar la trazabilidad del desarrollo.

## Socket Programming

*Función: Manejo de la comunicación en red para el modo multijugador en QuizHub.*

Librerías utilizadas: Java Socket y ServerSocket para la conexión entre jugadores.



## Configuración Inicial del Proyecto

Para comenzar con el desarrollo, se creó un nuevo proyecto en Android Studio con las siguientes configuraciones:

### Creación del Proyecto en Android Studio

*Nombre del Proyecto:* QuizHub.

*Lenguaje de Programación:* Java.

*Mínima versión de Android soportada:* API 24 (Android 7.0 Nougat).

Estructura del Proyecto: Activities y Fragments: Se utilizarán Fragments para la navegación entre pantallas.

MVVM (Model-View-ViewModel): Se implementará este patrón de arquitectura para separar la lógica de la interfaz de usuario.

### Integración con Firebase

Se configuró Firebase en el proyecto a través de Firebase Assistant en Android Studio. Se agregó el archivo **google-services.json en la carpeta app/**. Se habilitaron los servicios de Firestore y Authentication en la consola de Firebase.

### Configuración del Repositorio en GitHub

Se creó un repositorio privado en GitHub. Se vinculó el proyecto con el repositorio mediante Git en Android Studio.



## Instalación de Dependencias y Librerías Necesarias

Para garantizar un correcto funcionamiento de QuizHub, se instalaron diversas dependencias en el archivo build.gradle (Module: app).

### Dependencias para Firebase

Se añadieron las siguientes dependencias para la integración con Firebase Firestore y Firebase Authentication:

```
dependencies {  
    implementation(libs.junit.junit)  
    testImplementation ("junit:junit:4.13.2")  
    // Import the BoM for the Firebase platform  
    implementation(platform("com.google.firebaseio:firebase-bom:33.9.0"))  
  
    // Add the dependency for the Firebase Authentication library  
    // When using the BoM, you don't specify versions in Firebase library dependencies  
    implementation("com.google.firebaseio:firebase-auth")  
    implementation ("com.google.firebaseio:firebase-firebase:24.1.1")  
  
    implementation(platform("com.google.firebaseio:firebase-bom:33.9.0"))  
    implementation("com.google.firebaseio:firebase-analytics")  
    implementation(libs.appcompat)  
    implementation(libs.material)  
    implementation(libs.activity)  
    implementation(libs.constraintlayout)  
    implementation(libs.lifecycle.livedata.ktx)  
    implementation(libs.lifecycle.viewmodel.ktx)  
    implementation(libs.navigation.fragment)  
    implementation(libs.navigation.ui)  
    testImplementation(libs.junit)  
    androidTestImplementation(libs.ext.junit)  
    androidTestImplementation(libs.espresso.core)
```



# Desarrollo de funcionalidades básicas

## Implementación del sistema de preguntas y respuestas

Hemos implementado el sistema de preguntas como ya hemos mencionado anteriormente usando la Base de datos de Firebase.

La lógica usada ha sido la siguiente, en la clase Partida:

```
private void cargarPreguntas() { 1 usage
    db.collection(collectionPath: "Preguntas").get().addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            listaPreguntas = task.getResult().getDocuments();
            mostrarNuevaPregunta();
        } else {
            Toast.makeText(context: this, text: "Error cargando preguntas", Toast.LENGTH_SHORT).show();
        }
    });
}

private void mostrarNuevaPregunta() { 3 usages
    if (listaPreguntas == null || listaPreguntas.isEmpty()) {
        Toast.makeText(context: this, text: "No hay preguntas disponibles", Toast.LENGTH_SHORT).show();
        return;
    }

    if (preguntasRespondidas >= MAX_PREGUNTAS) {
        // Terminar el juego si se alcanzan las 12 preguntas
        terminarJuego();
        return;
    }

    // Obtener el color definido en colors.xml
    int colorBoton = getResources().getColor(R.color.boton_geografia);

    // Restablecer colores de los botones
    btnOpcion1.setBackgroundColor(colorBoton);
    btnOpcion2.setBackgroundColor(colorBoton);
    btnOpcion3.setBackgroundColor(colorBoton);
    btnOpcion4.setBackgroundColor(colorBoton);
}
```

```
// Seleccionar una pregunta aleatoria
int index = new Random().nextInt(listaPreguntas.size());
DocumentSnapshot preguntaActual = listaPreguntas.get(index);

textViewPregunta.setText(preguntaActual.getString(field: "Pregunta"));
List<String> opciones = (List<String>) preguntaActual.get("Opciones");
String respuestaCorrecta = preguntaActual.getString(field: "Respuesta correcta");

btnOpcion1.setText(opciones.get(0));
btnOpcion2.setText(opciones.get(1));
btnOpcion3.setText(opciones.get(2));
btnOpcion4.setText(opciones.get(3));

// Asignar eventos de clic para las opciones
View.OnClickListener listener = v -> verificarRespuesta((Button) v), respuestaCorrecta);
btnOpcion1.setOnClickListener(listener);
btnOpcion2.setOnClickListener(listener);
btnOpcion3.setOnClickListener(listener);
btnOpcion4.setOnClickListener(listener);

// Iniciar el temporizador
iniciarTemporizador();
```



## 1. Método cargarPreguntas()

Este método se encarga de cargar las preguntas almacenadas en Firebase Firestore:

- Obtener las preguntas de Firestore: Utiliza db.collection("Preguntas").get() para obtener todos los documentos de la colección "Preguntas".
- Comprobación de conexión exitosa: El resultado de la operación se maneja en un OnCompleteListener. Si la operación es exitosa (task.isSuccessful()), se asignan las preguntas a la variable listaPreguntas y se llama a mostrarNuevaPregunta() para mostrar la siguiente pregunta.
- Manejo de errores: Si ocurre un error en la carga, se muestra un mensaje de error al usuario mediante un Toast que configuramos

## 2. Método mostrarNuevaPregunta()

Este método muestra una nueva pregunta al jugador:

- Comprobación de disponibilidad de preguntas: Si listaPreguntas está vacía, se muestra un mensaje de error indicando que no hay preguntas disponibles.
- Fin del juego: Si ya se han respondido el número máximo de preguntas (MAX\_PREGUNTAS), se llama a terminarJuego(), lo que finaliza el juego.
- Restablecimiento de los botones: Los botones de opciones se restablecen a un color predeterminado (definido en colors.xml).
- Selección de pregunta aleatoria: Se elige una pregunta al azar de listaPreguntas. Se obtiene la pregunta y las opciones de respuesta del documento y se asignan a los botones correspondientes.
- Asignación de listeners a los botones: Se asignan eventos de clic para cada uno de los botones de respuesta, llamando al método verificarRespuesta() para comprobar si la respuesta seleccionada es correcta.



- Inicio del temporizador: Se inicia un temporizador para controlar el tiempo disponible para responder cada pregunta.

### 3. Método iniciarTemporizador()

Este método inicia el temporizador para la pregunta actual:

- Cancelación del temporizador anterior: Si ya existe un temporizador en ejecución, se cancela utilizando temporizador.cancel().
- Creación de un nuevo temporizador: Se crea un nuevo CountDownTimer con un tiempo definido en TIEMPO\_PREGUNTA (presumiblemente en milisegundos). El temporizador se actualiza cada segundo (1000 ms).
- Actualización del TextView con el tiempo restante: Cada segundo, el temporizador actualiza el TextView (textViewTimer) con los segundos restantes.
- Acción al finalizar el temporizador: Cuando el temporizador llega a cero, se muestra un mensaje de "Tiempo agotado" y se llama a mostrarNuevaPregunta() para pasar a la siguiente pregunta.

### 4. Método verificarRespuesta (Button botonSeleccionado, String respuestaCorrecta)

Este método verifica si la respuesta seleccionada por el jugador es correcta:

- Identificación de la respuesta seleccionada: Se obtiene el texto del botón seleccionado (botonSeleccionado.getText().toString()) y se compara con la respuesta correcta.
- Cambio de color de los botones:
  - Si la respuesta es correcta, se cambia el color del botón seleccionado a verde.
  - Si la respuesta es incorrecta, se cambia el color del botón seleccionado a rojo, y luego se muestra la respuesta correcta en verde.



- Cálculo de puntos:
  - Si la respuesta es correcta, se calcula el puntaje basado en el tiempo restante con la fórmula  $100 + (\text{tiempoRestante} / 100)$ . El puntaje no puede ser negativo.
  - El puntaje se suma al total de puntos acumulados (puntaje).
- Actualización del puntaje en Firebase: El puntaje actualizado se guarda en Firebase bajo el documento correspondiente de la sala, asociando el puntaje con el jugador.
- Manejo de retraso visual: Se introduce un retraso de 500 ms para cambiar el color de los botones y mostrar los resultados.
- Incremento del contador de preguntas respondidas: Al final del proceso, se incrementa el contador preguntasRespondidas, que lleva el control del número de preguntas respondidas.

## Diseño de interfaz gráfica y experiencia de usuario

### Activity 1 (Splash Screen)



El usuario ejecuta la aplicación y aparece una Splash Activity previamente configurada a modo de pantalla de carga de los recursos de nuestra aplicación.

Tras esta pantalla la aplicación se abrirá y comenzará su flujo como tal , el usuario podrá hacer Login o registrarse.



## Activity 2 (Login)



Aparece tras la ventana **OnCreate o SplashScreen** nuestra pantalla de **Login** de Aplicación. El flujo será que el usuario ya registrado inicie sesión con sus credenciales para comenzar con la experiencia en QuizzHub. Una vez haga clic en el botón entrar , la aplicación le mostrará la ventana principal del menú principal.

En el caso de que el usuario haya olvidado su contraseña, puede hacer clic en el texto interactivo resaltado de naranja para recuperar su contraseña por correo electrónico.

Si el usuario no tiene cuenta, puede hacer clic en el texto de No tienes cuenta y la app le enviará a la Activity de Registro, que será la pantalla para el registro del Usuario en la aplicación.

## Activity 3 (Registro)



Aparece tras hacer clic en **el texto mencionado anteriormente** dentro de la Activity del Login de Usuario.

Ahora el flujo será llenar la información que se pide al usuario para el registro de su cuenta en la base de datos de nuestra Aplicación. Una vez creada la cuenta y pulsar el registrarte, la App enviará al usuario de nuevo al flujo inicial a la pantalla de login.



## Activity 4 (Recuperacion Contraseña)



Esta Activity es la encargada de gestionar el restablecimiento de tu contraseña en el caso que te olvides de ella .

Comienza con pedirte una verificación de datos como tu Email o tu número de teléfono para confirmar que eres tú y hacer una búsqueda en la base de datos. Una vez se verifique que esa cuenta existe se sigue el flujo hacia la siguiente Activity de este proceso , en este caso se solicita el Código que de restablecimiento que te ha llegado al correo electrónico.

Después de verificar el código , el flujo termina con la creación de una nueva contraseña para tu usuario de Quizzhub y guardarla en la base de datos.



## Activity 5 (Home)



Esta es nuestra Activity de la pantalla **Menú Principal** tras tu Login de usuario, esta sería la pantalla Home, a la que podrás acceder más tarde desde el **NavigationMenu** inferior cuando quieras.

En esta pantalla el usuario podrá navegar entre distintas Activities.

La principal de ellas “**Jugar**” , en la que el flujo de navegación le llevará a elegir el modo de juego y comenzar una partida privada en sala , o contrarreloj en modo solitario.

Seguido a ella “**Categorías**” en las que el usuario podrá elegir una categoría en específico y jugar en solitario a preguntas exclusivamente de esa categoría. Además de no tener tiempo, se puede disfrutar de este modo como un reto personal o por puro aprendizaje, sin presión de tiempo. Esta funcionalidad aún no está implementada, pero es una muy buena idea para implementar en las siguientes versiones de nuestra aplicación.

“**Ajustes**” , en las que el usuario podrá silenciar los efectos de sonido , la música de fondo de la aplicación o cambiar el idioma junto con bloquear la orientación horizontal de la app.

El menú de **navegación inferior** , que le ofrecerá navegar entre distintas Activities, como Modo de juego, Rankings y su perfil personal acompañado de las estadísticas e información.

Se muestra información como las monedas ganadas en el juego, este es un sistema de premio propio. Sistema al que se le sumarán monedas al acabar la partida y haber respondido todas las preguntas correctamente.

Un cambio rápido a los idiomas más usados y para finalizar un botón Salir o logout junto al icono de su perfil, que le realizará el logout de su cuenta llevando de nuevo el flujo de la app a la Activity Login.



## Activity 6 (Categorías)



Si en el flujo de usuario en la Pantalla anterior, hace clic en Categorías, aparecerán las distintas categorías que están presentes en el juego. Como ya hemos mencionado antes esta Activity es un pequeño adelanto visual de cómo será en las siguientes versiones.

## Activity 7 (Ajustes)

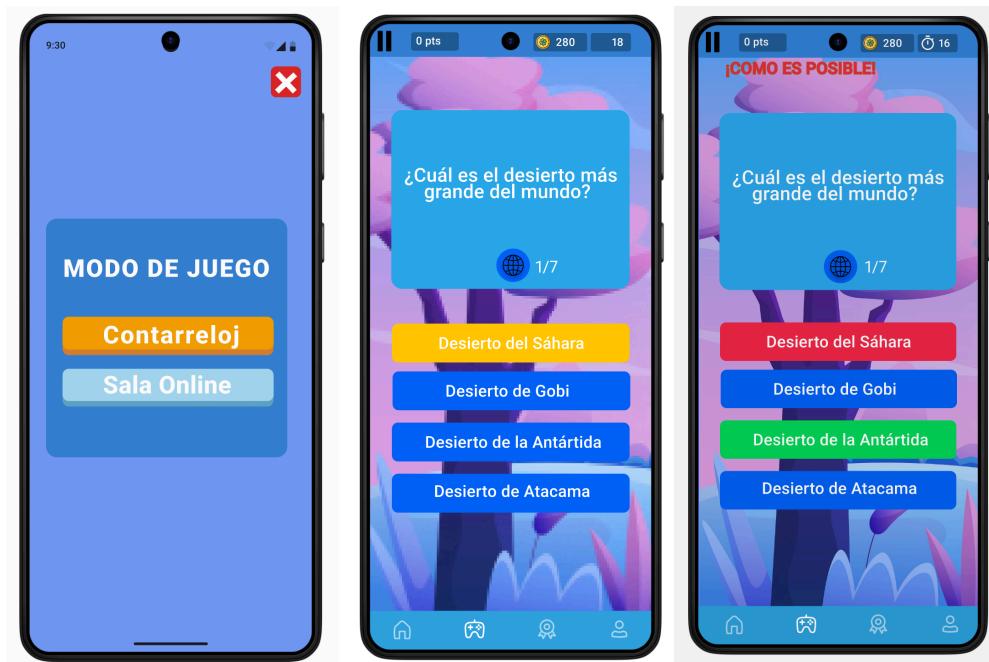


En el caso de que el usuario quisiera navegar a los ajustes , haría clic en el dentro del menú.

Una vez dentro , podrá silenciar opciones de audio como efectos de sonido o la música de la propia app. Y cambiar el idioma (Se reiniciará la app para aplicar los cambios) , gracias a la internalización aplicada en el desarrollo , el usuario podrá ver la app en distintos idiomas.



## Activity 8 (Jugar)



Volviendo al que sería el flujo principal que seguiría el usuario, el que está desarrollado para la funcionalidad principal de nuestra app.

Hacemos clic en el Botón “**Jugar**” dentro del menú principal y el usuario tiene la opción de jugar en solitario en modo Contrarreloj en este caso , o en Sala online con amigos de forma privada en formato Lobby/Sala.

Tras elegir **Contrarreloj** , la partida comenzará con una cuenta regresiva de 20 segundos por preguntas , el usuario deberá de hacer clic en la pregunta que crea correcta (Color Amarillo).

Si este fallase , se marca de color rojo esa misma y de color verde la correcta.

Con una pequeña animación de texto y palabras de atención o decepción.

Si acertase , se marcaría de color verde directamente.

Con una pequeña animación de textos positivos e incremento de puntos de la partida.

Si se elige **Sala Online**, la aplicación llevará a otra Activity que veremos más adelante



# Diseño



The image displays ten screenshots of the QuizzHub mobile application interface, arranged in a grid-like layout. In the center is the QuizzHub logo featuring a cartoon character holding a megaphone and a flag, with the text "QuizzHub" in large, bold letters.

- Top Row:**
  - Left: Login screen with fields for Email and Contraseña, and a forgot password link.
  - Middle: Home screen showing user info (#Miguel\_Martin97, Level 3, 280 points), a 'Jugar' button, and category icons for Geografía, Entretenimiento, Ciencia, Historia, Arte y Literatura, and Deporte.
  - Right: Game mode selection screen with options for Contarreloj (Timer) and Sala Online (Online Room).
- Second Row:**
  - Left: Create Room screen with instructions, a 'Crear Sala' button, and a room creation form.
  - Middle: Room creation confirmation screen with a room code (AC4RT8) and player count (5/5).
  - Right: Blurred screenshot of a game room.
- Third Row:**
  - Left: Quiz screen asking '¿Qué Relación entre Hansel y Gretel, en el Cuento de los hermanos Grimm?'. Options include Padre e hija, Amigos, Hermanos, and Suegro y Nuera.
  - Middle: Ranking screen showing top three players: Jose1977 (1st place), LucianaPM. (2nd place), and Aston19 (3rd place).
  - Right: Settings screen with music, sound, and notifications toggles, and language and country selection.
- Bottom Row:**
  - Left: Profile screen for user #Miguel\_Martin97 (Level 2, 12.569 pts) with a progress bar and statistics section.
  - Middle: Statistics screen detailing performance metrics like Mejor Juego, Mejor Pregunta, Mejor Desafío, and Mejores (Eventos Especiales).
  - Right: Performance screen showing Desempeño (Performance) data.



# Implementación de funcionalidades avanzadas

## Desarrollo del modo multijugador con sockets

El modo multijugador de QuizHub permite a los jugadores conectarse y jugar de forma sincronizada a través de Internet, usando un servidor para gestionar las conexiones entre los jugadores. A continuación, mostramos el desarrollo paso a paso de cómo implementar la funcionalidad avanzada del modo multijugador con sockets para QuizHub en base a lo que hemos hecho.

### 1. Objetivos del Modo Multijugador

El objetivo es permitir que varios jugadores se conecten a una sala de juego y compitan en tiempo real. Cada jugador debe:

- Unirse a una sala con un código único.
- Ver el progreso del juego.
- Sincronizar las respuestas y puntuaciones entre todos los jugadores.
- Ver un ranking al final de la partida.

### 2. Arquitectura de Comunicación

La arquitectura básica de la comunicación en el modo multijugador está basada en sockets, que permite la comunicación en tiempo real entre el servidor y los clientes (jugadores).

- **Servidor (host):** Un jugador actuará como host, que será el encargado de gestionar las conexiones y coordinar el flujo del juego. Este servidor utilizará sockets para enviar y recibir mensajes de los clientes.
- **Clientes:** Los jugadores que se conectan al servidor para unirse a la partida.

### 3. Requisitos Técnicos

Para implementar el modo multijugador con sockets en QuizHub, hemos tomado en cuenta:

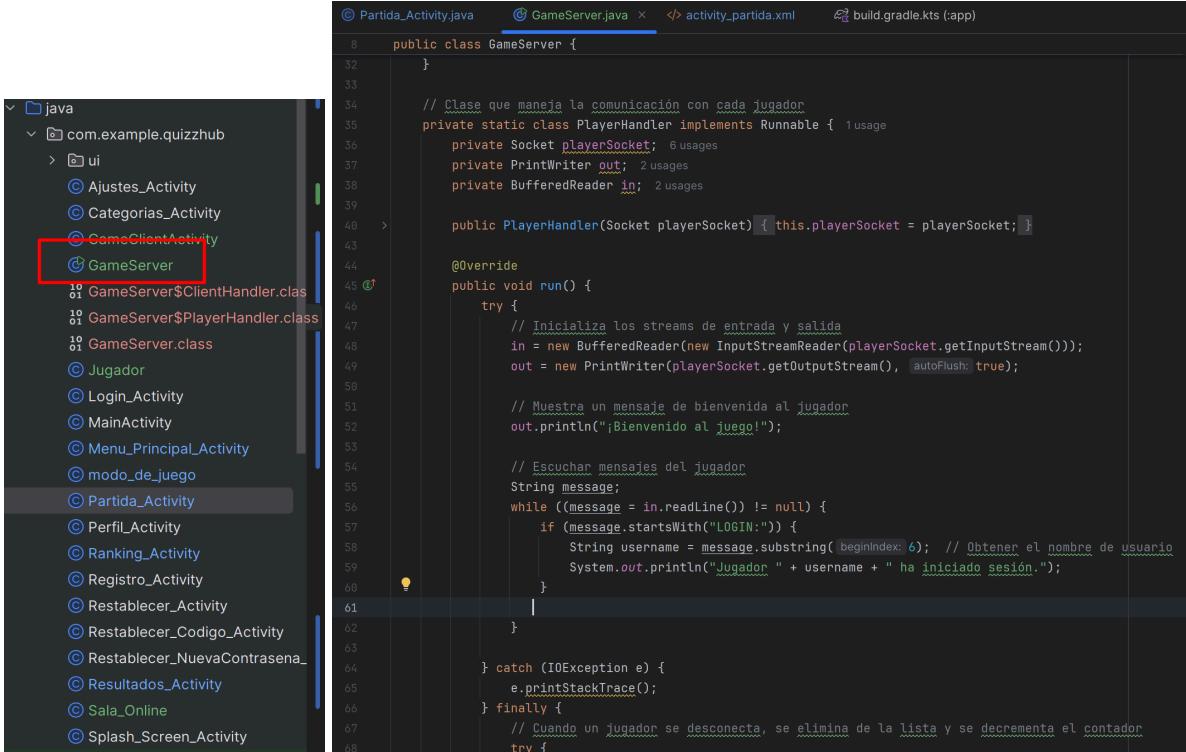
- Uso de Socket Programming en Java.
- Servidor que escucha conexiones y maneja las interacciones de los jugadores.
- Clientes que se conectan al servidor, envían información (respuestas, puntuaciones) y reciben actualizaciones (temporalización, preguntas, resultados).



## 4. Pasos para Implementar el Modo Multijugador

### 4.1. Crear el Servidor

1. Creación de la clase GameServer (servidor): El servidor se encarga de aceptar conexiones de clientes, gestionarlas y enviar la información relevante durante el juego.



```

public class GameServer {
    ...
    // Clase que maneja la comunicación con cada jugador
    private static class PlayerHandler implements Runnable {
        ...
        public PlayerHandler(Socket playerSocket) { this.playerSocket = playerSocket; }

        @Override
        public void run() {
            try {
                ...
                // Inicializa los streams de entrada y salida
                in = new BufferedReader(new InputStreamReader(playerSocket.getInputStream()));
                out = new PrintWriter(playerSocket.getOutputStream(), true);

                // Muestra un mensaje de bienvenida al jugador
                out.println("¡Bienvenido al juego!");

                // Escuchar mensajes del jugador
                String message;
                while ((message = in.readLine()) != null) {
                    if (message.startsWith("LOGIN:")) {
                        String username = message.substring(6); // Obtener el nombre de usuario
                        System.out.println("Jugador " + username + " ha iniciado sesión.");
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                ...
            }
        }
    }
}

```

### 4.2. Crear el Cliente

El cliente será la parte de la aplicación que se ejecuta en los dispositivos Android de los jugadores. Este cliente se conectará al servidor y podrá enviar y recibir mensajes.

1. **Configuración de la clase GameClient (cliente):** La clase GameClient gestiona la conexión al servidor, el envío de respuestas y la recepción de actualizaciones del servidor.





## 4.3. Gestión de Sala y Jugadores

Para gestionar las conexiones de los jugadores y las salas, usamos Firebase para almacenar información sobre la sala y los jugadores. El servidor puede consultar Firebase para obtener la lista de jugadores y asegurarse de que hay al menos dos jugadores antes de iniciar el juego.

- Cuando un jugador se une a la sala, el servidor recibe la información de Firebase y añade el jugador a la lista.
- Una vez que el número mínimo de jugadores está alcanzado, el servidor puede comenzar el juego, enviando preguntas a todos los jugadores conectados.

## 4.4. Implementación de la Lógica del Juego

- El servidor gestionará el flujo de la partida, enviando preguntas a todos los jugadores.
- Cada jugador enviará sus respuestas al servidor.
- El servidor calculará las puntuaciones y enviará las actualizaciones a todos los jugadores.

## 4.5. Comunicación del Ranking Final

- Una vez que todos los jugadores hayan respondido a todas las preguntas, el servidor calculará el **ranking final** y lo enviará a todos los jugadores.

El ranking puede ser enviado como un mensaje a través del socket, que cada cliente recibirá y mostrará en la interfaz de usuario (UI) de la aplicación.

## 5. Pruebas y Depuración

- **Pruebas locales:** Durante el desarrollo, hemos probado la conexión del servidor y los clientes en la misma máquina (localhost).
- **Pruebas en dispositivos reales:** Probamos las funcionalidades de multijugador en una red local , usando dos máquinas virtuales en android studio.

# Conclusion

## Conclusiones

Desarrollar QuizHub y su modo multijugador ha sido un gran paso hacia una experiencia de juego más interactiva y atractiva. Implementar sockets para la comunicación en tiempo real ha permitido que los jugadores se unan a salas, compitan y vean los resultados al instante, mejorando considerablemente la jugabilidad.



El uso de Firebase ha sido clave para gestionar las salas y la autenticación de los jugadores. Esto ha simplificado la sincronización de datos y la creación de partidas sin necesidad de servidores externos complejos, lo que también facilita el escalado de la aplicación a medida que más jugadores se suman.

## Resultados Obtenidos

Modo multijugador funcional: Ahora los jugadores pueden conectarse, participar en partidas sincronizadas y visualizar las puntuaciones en tiempo real.

Comunicaciones estables: La implementación de sockets ha logrado una comunicación fluida entre el servidor y los clientes, asegurando que las preguntas, respuestas y puntuaciones se actualicen correctamente sin retrasos.

Integración con Firebase: Gracias a Firebase, la gestión de las salas, los jugadores y las puntuaciones se ha optimizado sin necesidad de servidores dedicados adicionales.

Interfaz mejorada: Se ha adaptado la UI para que los jugadores puedan ver el código de la sala, la lista de participantes y el estado del juego de forma clara y sencilla.

Sincronización de resultados: Las respuestas y puntuaciones de cada jugador se actualizan de forma instantánea para todos los participantes, garantizando una experiencia fluida y sin interrupciones.

---

## Posibles Mejoras y Futuras Actualizaciones

Mejor gestión de reconexión: Implementar un sistema para que, si un jugador pierde la conexión, pueda reconnectarse sin afectar la partida en curso.

Optimización de la red: Reducir la latencia y mejorar la eficiencia de las comunicaciones, especialmente para jugadores en diferentes regiones geográficas. En el futuro, podría ser útil utilizar servidores dedicados para mejorar la estabilidad.

Nuevas características en el juego:

- Modos de juego variados: Incorporar nuevas dinámicas, como preguntas por categorías, rondas rápidas o partidas en equipo.
- Interacción social: Añadir un sistema de chat para que los jugadores puedan comunicarse durante la partida.
- Sistema de recompensas: Introducir logros y premios para incentivar la participación y aumentar la competitividad.



Escalabilidad: A medida que crezca el número de jugadores, sería conveniente migrar a una infraestructura en la nube que permita manejar múltiples partidas simultáneamente sin afectar el rendimiento.

Personalización de perfiles: Incluir opciones para que los jugadores personalicen su avatar o perfil, lo que ayudaría a fidelizar usuarios y hacer la experiencia más única.

## Reflexión sobre el Desarrollo del Proyecto

Crear el modo multijugador de QuizHub ha sido un reto técnico apasionante y, al mismo tiempo, un aprendizaje enorme. Durante el desarrollo, he profundizado en la comunicación en tiempo real con sockets, la integración con Firebase y la importancia de diseñar una experiencia de usuario fluida y sin errores.

Uno de los mayores desafíos fue garantizar una comunicación eficiente entre el servidor y los clientes, evitando retrasos o fallos que pudieran afectar la partida. A medida que el proyecto siga creciendo, también aumentarán los retos en la gestión de conexiones y el rendimiento de la red, por lo que será clave seguir optimizando la infraestructura.

A pesar de las dificultades, este proyecto ha sido un gran paso adelante para hacer de QuizHub un juego más dinámico y divertido. Todo lo aprendido en este proceso no solo servirá para seguir mejorando el modo multijugador, sino que también será útil en futuros proyectos que requieran programación en red y gestión de usuarios en tiempo real.

El camino no termina aquí: QuizzHub sigue evolucionando, y hay muchas oportunidades para optimizar y expandir la experiencia de juego en futuras actualizaciones.