MEMORIA DEL PROYECTO

2ºDesarrollo de Aplicaciones Multiplataforma (DAM)

Marcos González, Cristian Tenorio y Diego Rodríguez

CONTENIDO

1. Introducción....2
2. Tecnologías utilizadas....2
3. Pasos de desarrollo....3
4. Mockups y diseño de pantallas....4
5. Explicación de las actividades de la APP....5
SplashActivity....5
LoginActivity:....6
RegisterActivity:....10
ReviewDao.kt....15
AppDatabase....17
ReviewAdapter....20
MainActivity....23
AndroidManifest.xml....29
6. Layouts (diseño)....30
7. Problemas encontrados y sus Soluciones....30
8. Diagrama de clases y arquitectura....31

1. INTRODUCCIÓN

El proyecto consiste en una aplicación móvil Android llamada **MyList**, para gestionar reseñas de películas y series. Permite al usuario registrarse, iniciar sesión y añadir, editar o borrar reseñas. Cada reseña incluye título, género y comentario, y se puede filtrar por género.

2. TECNOLOGÍAS UTILIZADAS

- Kotlin: lenguaje principal para Android.

- Android Studio: IDE utilizado.

- Room (SQLite): para almacenamiento local de reseñas.

```
implementation("androidx.room:room-runtime:2.8.1")
kapt("androidx.room:room-compiler:2.8.1")
implementation("androidx.room:room-ktx:2.8.1")
```

- Firebase Authentication: para gestión segura de usuarios.

```
implementation(platform( notation = "com.google.firebase:firebase-bom:34.3.0"))
implementation("com.google.firebase:firebase-storage")
implementation("com.google.firebase:firebase-auth")
implementation("com.google.firebase:firebase-firestore")
```

- Material Design 3: interfaz moderna y estilizada.

- RecyclerView: para mostrar la lista de reseñas dinámicamente.

```
implementation("androidx.appcompat:appcompat:1.7.0")
implementation("androidx.constraintlayout:constraintlayout:2.2.0")
implementation("androidx.recyclerview:recyclerview:1.3.1")
implementation("com.google.android.material:material:1.9.0")
```

3. PASOS DE DESARROLLO

3.1 Configuración del proyecto

- Se creó un proyecto Android nuevo con Kotlin y tema Material3 DayNight NoActionBar en themes.xml.

- Configuramos Firebase:

- Descargamos el archivo google-services.json.
- Colocamos las dependencias en build.gradle (firebase-auth-ktx, firebase-firestore-ktx) y plugin com.google.gms.google-services.
- Esto permitió inicializar Firebase

3.2 Desarrollo de la autenticación

- SplashActivity: detecta si el usuario tiene la sesión iniciada o no, es la actividad principal.
- LoginActivity: inicio de sesión del usuario.
- RegisterActivity: registro de usuario con validación básica.
- Se utilizó FirebaseAuth para iniciar sesión o crear usuarios.

3.3 Implementación de la base de datos local

En ReviewDao se encuentra la creación de la tabla y los métodos:

- Entidad Reviews: data class UserReview con id, titulo, genero y comentario.
- DAO (UserReviewDao): funciones insert(), update(), delete(), getAll() y getByGenero().

Y luego la base de datos en:

- AppDatabase: extiende RoomDatabase y conecta al DAO.

3.4 Desarrollo de la pantalla principal

La interfaz principal es MainActivity que permite visualizar, añadir, editar, eliminar y filtrar reseñas por género.

En esta actividad implementamos Room para realizar las operaciones CRUD a través de UserReviewDao, luego utilizamos un RecyclerView junto con el ReviewAdapter para mostrar la lista de reseñas con un layout personalizado item_resena.xml.

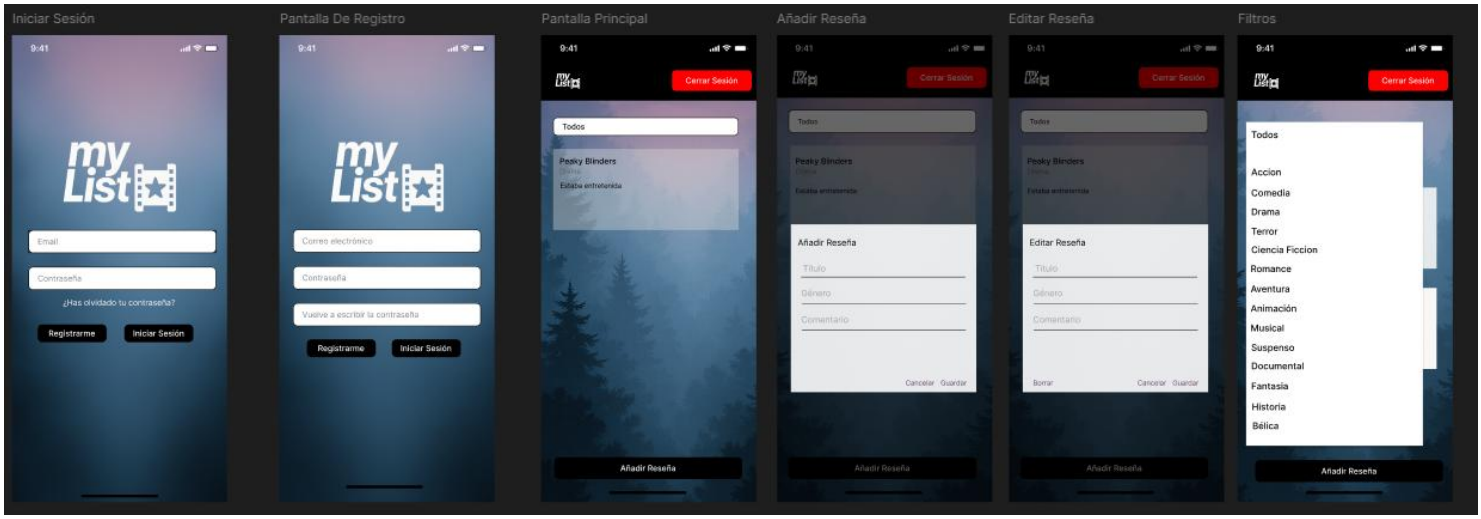
Para el filtrado decidimos usar un Spinner adaptado a un array que contiene los géneros.

Incluimos un botón para añadir una nueva reseña, el cual abre un diálogo para crearla.

Al hacer clic sobre una reseña, se muestra otro diálogo que permite editarla o eliminarla.

También hay un botón para cerrar sesión y una pequeña validación que, en caso de que el usuario no esté logueado, lo redirige a LoginActivity.

4. MOCKUPS Y DISEÑO DE PANTALLAS.



5. EXPLICACIÓN DE LAS ACTIVIDADES DE LA APP

SplashActivity

Qué hace: Es la pantalla inicial que decide a dónde enviar al usuario según si está logueado o no, no es visible.

Librerias:

- **import android.content.Intent:** Sirve para abrir otra pantalla o realizar una acción en el sistema, como enviar datos entre actividades.
- **import android.os.Bundle:** Permite guardar y recuperar información cuando una actividad se crea o se reinicia.
- **import androidx.appcompat.app.AppCompatActivity:** Es la clase base de las actividades que ofrece compatibilidad con versiones antiguas de Android.
- **import com.google.firebase.auth.FirebaseAuth:** Se usa para manejar el registro, inicio y cierre de sesión de usuarios en Firebase.

Flujo:

Cómo funciona (onCreate()):

- Revisa si **FirebaseAuth.currentUser** es null. (si el usuario no está logueado)

- Si no hay usuario: abre **LoginActivity**.

- Si ya hay usuario: abre **MainActivity**.

- Llama a **finish()** para cerrar el **SplashActivity**.

```
class SplashActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val auth = FirebaseAuth.getInstance()
        val currentUser = auth.currentUser

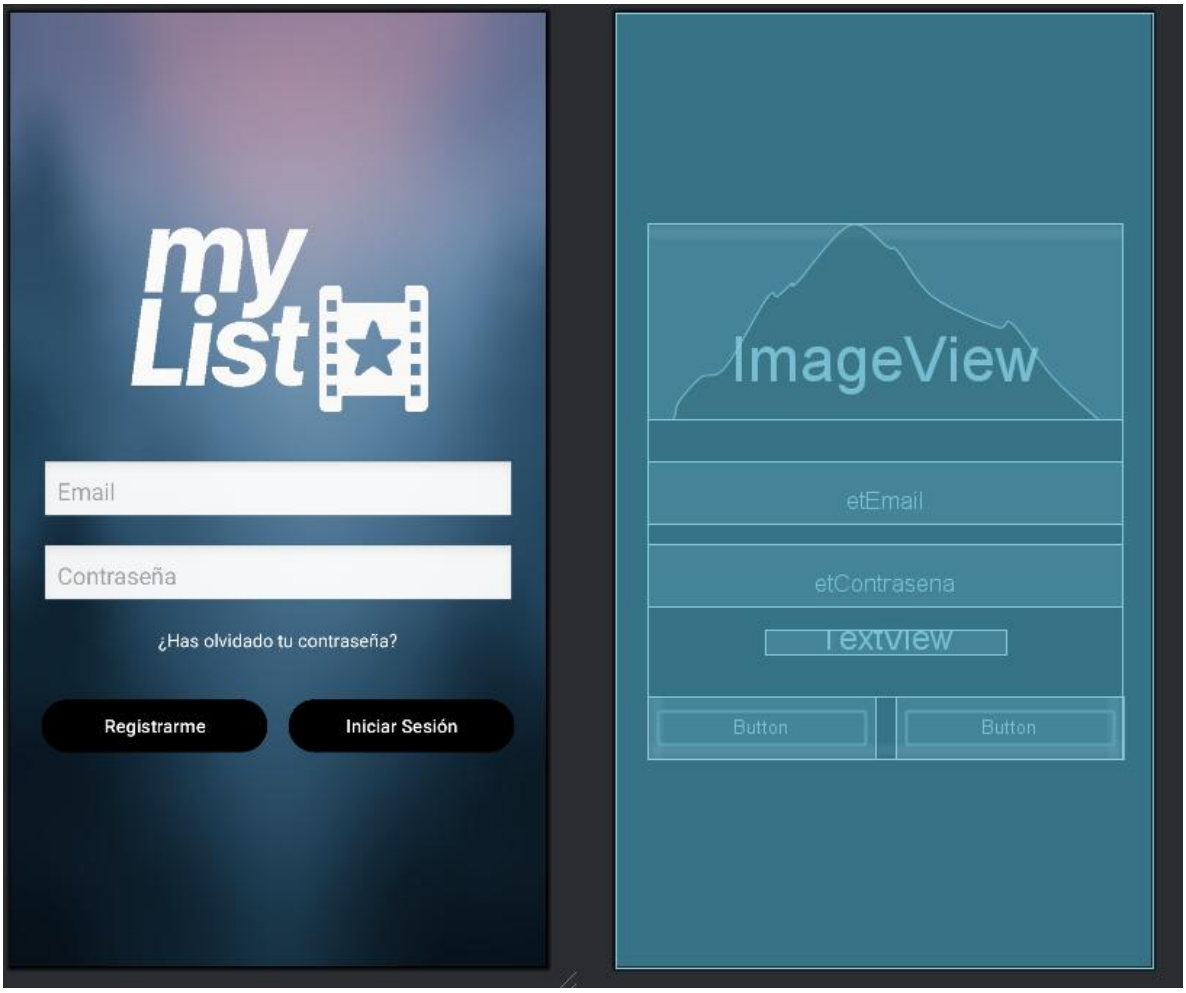
        if (currentUser == null) {
            // No está logueado
            startActivity(Intent( packageContext = this,  cls = LoginActivity::class.java))
        } else {
            // Ya logueado
            startActivity(Intent( packageContext = this,  cls = MainActivity::class.java))
        }
        finish()
    }
}
```

LoginActivity:

Qué hace: Permite a los usuarios iniciar sesión o acceder al apartado de registrarse.

Layout asociado (diseño): activity_login.xml

```
setContentView(R.layout.activity_login)
```



Librerias:

- **import android.content.Intent:** Sirve para abrir otra pantalla o realizar una acción en el sistema, como enviar datos entre actividades.
- **import android.os.Bundle:** Permite guardar y recuperar información cuando una actividad se crea o se reinicia.
- **import android.widget.*:** Contiene los componentes visuales básicos de Android, como Button, TextView, EditText, etc.
- **import androidx.appcompat.app.AppCompatActivity:** Es la clase base de las actividades que ofrece compatibilidad con versiones antiguas de Android.
- **import com.google.firebase.FirebaseApp:** Inicializa Firebase en la aplicación para poder usar sus servicios.
- **import com.google.firebase.auth.FirebaseAuth:** Se usa para manejar el registro, inicio y cierre de sesión de usuarios en Firebase.

Componentes:

- **EditText etEmail** y **etContrasena**: ingresar email y contraseña.

```
2 usos
private lateinit var etEmail: EditText
2 usos
private lateinit var etContrasena: EditText
```

- **Button btnLogin**: iniciar sesión.

```
private lateinit var btnLogin: Button
```

- **Button btnRegistrar**: abrir **RegisterActivity**.

```
private lateinit var btnRegistrar: Button
```

```
// Botón Registrar -> abrir la pantalla de registro
btnRegistrar.setOnClickListener {
    startActivity(Intent( packageContext = this, cls = RegisterActivity::class.java))
}
```

- **TextView tvOlvidar**: recuperar contraseña (De momento no hemos implementado el método, solo una pequeña “alerta”).

```
private lateinit var tvOlvidar: TextView
```

```
        // Olvidaste contraseña
        tvOlvidar.setOnClickListener {
            Toast.makeText(
                context = this,
                text = "Función temporalmente no disponible",
                duration = Toast.LENGTH_LONG
            ).show()
        }
    }
```

- **FirebaseAuth auth**: manejar la autenticación.

```
private lateinit var auth: FirebaseAuth
```

```
// Inicializar Firebase
FirebaseApp.initializeApp( context = this)
auth = FirebaseAuth.getInstance()
```

Para conectar las variables con los botones del layout:

Flujo:

- Botón Login: llama a **loginUsuario()**.

```
// Botón Login
btnLogin.setOnClickListener { loginUsuario() }
```

- El método **loginUsuario()** revisa que no haya campos vacíos

- Inicia sesión con Firebase (**auth.signWithEmailAndPassword()**).

- Éxito: muestra mensaje y abre **MainActivity** (limpia las actividades para que no puedan darle al boton de volver, y aparecer en LoginActivity de nuevo).

- Error: muestra mensaje de error.

```
    private fun loginUsuario() {
        val email = etEmail.text.toString().trim()
        val password = etContrasena.text.toString().trim()

        // Validación básica
        if (email.isEmpty() || password.isEmpty()) {
            Toast.makeText( context = this,  text = "Por favor ingresa email y contraseña",  duration = Toast.LENGTH_SHORT).show()
            return
        }

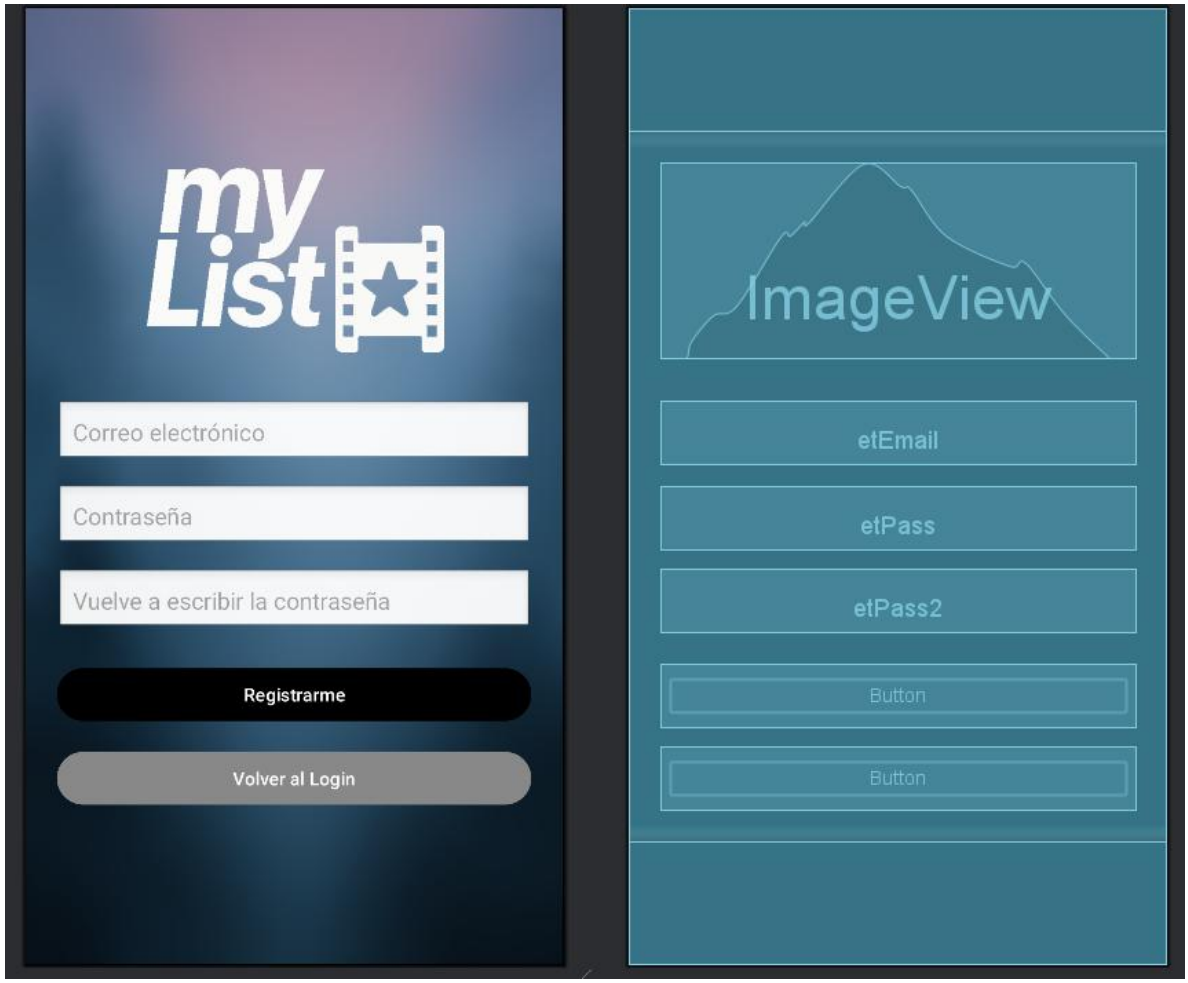
        // Iniciar sesión con Firebase
        auth.signInWithEmailAndPassword( p0 = email,  p1 = password)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    Toast.makeText( context = this,  text = "Login exitoso",  duration = Toast.LENGTH_SHORT).show()
                    // Navegar a MainActivity si el inicio de sesión es exitoso  Tabulador to complete
                    val intent = Intent( packageContext = this,  cls = MainActivity::class.java)
                    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                    startActivity(intent)
                    finish()
                } else {
                    // Mostrar error detallado
                    Toast.makeText(
                        context = this,
                        text = "Error: ${task.exception?.localizedMessage}",
                        duration = Toast.LENGTH_LONG
                    ).show()
                }
            }
    }
}
```

RegisterActivity:

Qué hace: Permite crear cuentas nuevas y guardar la información básica en Firestore o volver al Login.

Layout asociado (diseño): activity_register.xml

```
setContentView(R.layout.activity_register)
```



Librerias:

- **import android.content.Intent**: Sirve para abrir otra pantalla o realizar una acción en el sistema, como enviar datos entre actividades.
- **import android.os.Bundle:** Permite guardar y recuperar información cuando una actividad se crea o se reinicia.
- **import android.widget.Button:** Representa un botón que el usuario puede presionar para ejecutar una acción.
- **import android.widget.EditText:** Campo de texto que permite al usuario escribir información, como un nombre o una contraseña.
- **import android.widget.Toast:** Muestra mensajes cortos en pantalla para notificar al usuario (por ejemplo, "Inicio de sesión exitoso").
- **import androidx.appcompat.app.AppCompatActivity:** Es la clase base de las actividades que ofrece compatibilidad con versiones antiguas de Android.
- **import com.google.firebase.auth.FirebaseAuth:** Se usa para manejar el registro, inicio y cierre de sesión de usuarios en Firebase.
- **import com.google.firebase.firestore.FirebaseFirestore:** Permite conectarse y realizar operaciones (leer, escribir, actualizar, eliminar) en la base de datos Firestore de Firebase.

Componentes:

- **EditText etEmail, etPass, etPass2**: email y contraseñas.

```
private lateinit var etEmail: EditText
2 usos
private lateinit var etPass: EditText
2 usos
private lateinit var etPass2: EditText
```

- **Button btnRegistrar**: registrar usuario.

```
private lateinit var btnRegistrar: Button
```

- **Button btnVolverLogin**: volver a **LoginActivity**.

```
private lateinit var btnVolverLogin: Button
```

```
// Botón para volver al Login
btnVolverLogin.setOnClickListener {
    startActivity(Intent( packageContext = this, cls = LoginActivity::class.java))
    finish()
}
```

- **FirebaseAuth auth** : crear cuentas.

```
private lateinit var auth: FirebaseAuth
```

```
auth = FirebaseAuth.getInstance()
```

- **FirebaseFirestore db** : guardar datos del usuario.

```
private val db: FirebaseFirestore by lazy { FirebaseFirestore.getInstance() }
```

Para conectar las variables con los botones del layout:

```
etEmail = findViewById( id = R.id.etEmail)
etPass = findViewById( id = R.id.etPass)
etPass2 = findViewById( id = R.id.etPass2)
btnRegistrar = findViewById( id = R.id.btnRegistrar)
btnVolverLogin = findViewById( id = R.id.btnVolverLogin)
```

Flujo:

- Botón Volver Login: abre **LoginActivity**.

```
// Botón para volver al Login
btnVolverLogin.setOnClickListener {
    startActivity(Intent( packageContext = this,  cls = LoginActivity::class.java))
    finish()
}
```

- Botón Registrar: llama a **registrarUsuario()**.

```
    // Botón para registrar usuario
    btnRegistrar.setOnClickListener { registrarUsuario() }
}
```

- El método **registrarUsuario()** valida: campos vacíos, contraseñas iguales, longitud mínima.

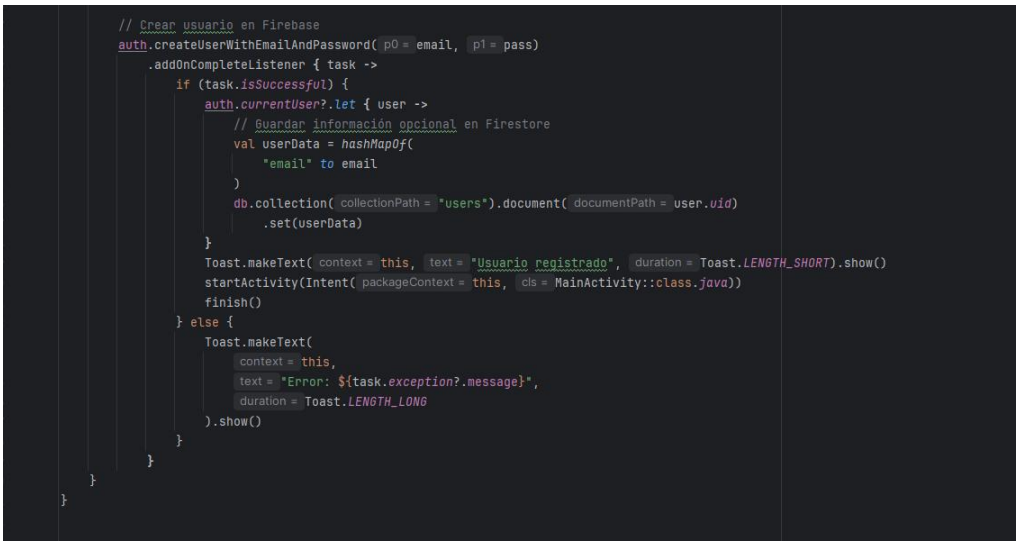
- Crea usuario con **auth.createUserWithEmailAndPassword()**.

- Éxito: guarda email en Firestore y abre **MainActivity** (limpia las actividades para que no puedan darle al boton de volver, y aparecer en RegisterActivity de nuevo).

- Error: muestra mensaje de error.

```
1 uso
private fun registrarUsuario() {
    val email = etEmail.text.toString().trim()
    val pass = etPass.text.toString().trim()
    val pass2 = etPass2.text.toString().trim()

    // Validaciones
    when {
        email.isEmpty() || pass.isEmpty() || pass2.isEmpty() -> {
            Toast.makeText( context = this,  text = "Completa todos los campos",  duration = Toast.LENGTH_SHORT).show()
            return
        }
        pass != pass2 -> {
            Toast.makeText( context = this,  text = "Las contraseñas no coinciden",  duration = Toast.LENGTH_SHORT).show()
            return
        }
        pass.length < 6 -> {
            Toast.makeText( context = this,  text = "La contraseña debe tener al menos 6 caracteres",  duration = Toast.LENGTH_SHORT).show()
            return
        }
    }
```



```
        // Crear usuario en Firebase
        auth.createUserWithEmailAndPassword( p0 = email,  p1 = pass)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    auth.currentUser?.let { user ->
                        // Guardar información opcional en Firestore
                        val userData = hashMapOf(
                            "email" to email
                        )
                        db.collection( collectionPath = "users").document( documentPath = user.uid)
                            .set(userData)
                    }
                    Toast.makeText( context = this,  text = "Usuario registrado",  duration = Toast.LENGTH_SHORT).show()
                    startActivity(Intent( packageContext = this,  cls = MainActivity::class.java))
                    finish()
                } else {
                    Toast.makeText(
                        context = this,
                        text = "Error: ${task.exception?.message}",
                        duration = Toast.LENGTH_LONG
                    ).show()
                }
            }
    }
}
```

ReviewDao.kt

Qué hace:

Define las operaciones principales de acceso a la base de datos local

mediante **Room**.

Permite consultar, insertar, actualizar y eliminar reseñas de usuarios almacenadas en la tabla reviews.

Se comunica con la clase AppDatabase y es utilizada desde MainActivity para gestionar la lista de reseñas mostrada al usuario.

Librerias:

- **import androidx.room.Dao:** Indica que la interfaz está marcada como Data Access Object (DAO). Permite a Room generar automáticamente el código necesario para acceder a la base de datos y realizar las operaciones CRUD (crear, leer, actualizar y borrar).
- **import androidx.room.Delete:** Se utiliza para marcar un método dentro del DAO que eliminará un registro de la base de datos. Room genera automáticamente la instrucción SQL DELETE correspondiente.
- **import androidx.room.Entity:** Convierte una clase en una tabla de base de datos. Cada propiedad de la clase se transforma en una columna dentro de esa tabla. En este caso, la clase UserReview es la entidad que representa la tabla reviews.
- **import androidx.room.Insert:** Marca un método del DAO para insertar datos en la tabla. Room genera el código SQL INSERT INTO de forma automática al compilar.
- **import androidx.room.PrimaryKey:** Define cuál campo de la entidad será la clave primaria de la tabla. Normalmente se usa con autoGenerate = true para que Room asigne automáticamente un valor único a cada fila.
- **import androidx.room.Query:** Permite escribir consultas SQL personalizadas dentro del DAO. Por ejemplo, @Query("SELECT * FROM reviews") devuelve todas las reseñas guardadas.
- **import androidx.room.Update:** Marca un método que actualizará registros existentes en la tabla según su clave primaria. Room genera el SQL UPDATE necesario de forma automática.

Componentes:

USERREVIEW.KT

- @Entity → convierte la clase en una tabla SQL llamada reviews, con los campos id, titulo, género y comentarios, los cuales se vuelven columnas
- @PrimaryKey (autoGenerate = true): id → es la clave primaria y Room la autogenera con el valor por defecto 0L es típico para indicar que aún no tiene id antes de insertar

```
@Entity(tableName = "reviews") // Tabla

data class UserReview(

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

    val titulo: String,

    val genero: String,

    val comentario: String
)
```

USERREVIEWDAO.KT

- @Dao → Indica que la interfaz forma parte del sistema de acceso a datos de Room.

```
@Dao
interface UserReviewDao {
```

- @Query("SELECT * FROM reviews") → Devuelve todas las reseñas guardadas.

```
@Query( value = "SELECT * FROM reviews")
```

- @Query("SELECT * FROM reviews WHERE genero = :genero") → Devuelve reseñas filtradas por género.

```
@Query( value = "SELECT * FROM reviews WHERE genero = :genero")
```

- @Insert → Anotación del método de insert.

```
@Insert
```

- @Update → Anotación del método de update.

```
@Update
```

- @Delete → Anotación del método de delete.

Funciones suspend:

Todas las funciones están marcadas como suspend para ejecutarse dentro de **corrutinas** y no bloquear el hilo principal.

Para conectar con otras partes del proyecto:

- La interfaz UserReviewDao se instancia automáticamente a través de la clase AppDatabase.
- Desde MainActivity, se obtiene el DAO mediante db.userReviewDao() para ejecutar operaciones CRUD sobre las reseñas.
- Los métodos del DAO son llamados desde ViewModel o directamente en corutinas dentro de la actividad principal.

Flujo:

- getAll() → Obtiene y muestra todas las reseñas guardadas en el RecyclerView.

```
suspend fun getAll(): List<UserReview>
```

- getByGenero(genero) → Filtra la lista según el género seleccionado en el Spinner.

```
suspend fun getByGenero(genero: String): List<UserReview>
```

- insert(review) → Se ejecuta al añadir una nueva reseña desde el diálogo “Añadir”.

```
suspend fun insert(review: UserReview)
```

- update(review) → Se ejecuta cuando el usuario edita una reseña existente.

```
suspend fun update(review: UserReview)
```

- delete(review) → Se llama al eliminar una reseña mediante el botón correspondiente.

```
suspend fun delete(review: UserReview)
```

AppDatabase

Qué hace:

La clase AppDataBase implementa de forma eficiente una base de datos local persistente usando ROOM. Su diseño garantiza seguridad, eficiencia y escablidad, permitiendo que la aplicación gestione datos estructurados de manera moderna y fiable dentro delo entorno.

```
@Database(entities = [UserReview::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage
    abstract fun userReviewDao(): UserReviewDao

    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null

        1 Usage
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    klass = AppDatabase::class.java,
                    name = "app_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Librerias:

- **android.content.Context:** proporciona acceso al entorno global de la aplicación, permitiendo obtener recursos, acceder a bases de datos o iniciar componentes del sistema.
- **androidx.room.Database:** define una clase como base de datos Room, indicando las entidades y la versión del esquema.
- **androidx.room.Room:** proporciona métodos para crear o acceder a la base de datos de manera sencilla y segura, gestionando automáticamente las conexiones.
- **androidx.room.RoomDatabase:** clase abstracta que actúa como punto principal de acceso a los datos.
  Define la estructura de la base de datos y los métodos para obtener los *DAO* (Data Access Objects).

Componentes:

La clase se declara con @**Database**, que define las siguientes características:

- **entities =[UserReview:class]:** la base de datos tiene una tabla asociada a la entiedad UserReview.
- **version = 1:** versión actual del esquema de base de datos.
- **exportSchema=false:** evita la exportación del esquema a un archivo externo.

Flujo:

- **abstract fun userReviewDao():** UserReviewDao -> este método proporciona acceso al objeto UserReviewDao, que contiene las operaciones CRUD (Create, Read, Update, Delete) sobre la entidad UserReview.
- **@Volatile** asegura que los cambios en la variable INSTANCE sean visibles a todos los hilos.
- El bloque **synchcronized** garantiza la creación segura de la base de datos en entornos multihilos.
- **Room.databaseBuilder(…)** crea o recupera la base de datos denominada “app_database”
- Se utiliza **context.applicationContext** para evitar fugas de memoria.
  Si la instancia ya existe, se reutiliza y e ncaso contrario, se crea una nueva.

Ventajas del uso de Room:

- Valida las consultas SQL en tiempo de compilación.
- Reduce el riesgo de errores y fugas de memoria.

ReviewAdapter

Qué hace:

El archivo ReviewAdapter define el adaptador encargado de mostrar la lista

de reseñas guardadas en la base de datos dentro de un **RecyclerView**. Su función principal es vincular los datos del modelo UserReview con el diseño visual de cada ítem (item_resena.xml), permitiendo además manejar acciones cuando el usuario selecciona una reseña.

Librerias:

- **Import UserReview:** Importa la clase UserReview, que representa la entidad o modelo de datos donde se guarda la información de cada reseña (título, género y comentario). Permite usarla dentro del adaptador o donde se necesite mostrar los datos.
- **import android.view.LayoutInflater:** Sirve para convertir (inflar) un archivo XML de diseño en un objeto View que se puede mostrar en pantalla. Es muy usado en RecyclerView para crear cada elemento de la lista.
- **import android.view.View:** Es la clase base de todos los componentes visuales de Android (botones, textos, etc.). Permite manejar acciones como clics, visibilidad o estilos de cualquier elemento gráfico.
- **Import android.view.ViewGroup:** Representa un contenedor que puede tener dentro otras vistas (View). En el caso del RecyclerView, se usa para colocar correctamente cada elemento dentro de la lista.
- **import android.widget.TextView:** Permite mostrar texto en la interfaz de usuario. En un RecyclerView, se usa normalmente para enseñar el título, género o comentario de una reseña.
- **import androidx.recyclerview.widget.RecyclerView:** Es el componente de Android que muestra listas o colecciones de elementos de forma eficiente. Se usa junto con un adapter y un ViewHolder para mostrar los datos de UserReview en forma de lista o tarjetas.

Componentes:

-
```
) : RecyclerView.Adapter<ReviewAdapter.ReviewViewHolder>() {
```
RecyclerView.Adapter<ReviewAdapter.ReviewViewHolder> → Clase base que conecta la lista de reseñas con la interfaz.
- ReviewViewHolder → Clase interna que representa cada ítem de la lista (vista individual).
```
class ReviewViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
```
-
```
    val titulo: TextView = itemView.findViewById( id = R.id.textTitulo)
```
TextView titulo → Muestra el título de la reseña.
- TextView genero → Muestra el género asociado a la reseña.
```
val genero: TextView = itemView.findViewById( id = R.id.textGenero)
```
- TextView comentario → Muestra el comentario del usuario.

```
val comentario: TextView = itemView.findViewById( id = R.id.textComentario)
```

- ```
  private var reviews: List<UserReview>,
  ```
  List<UserReview> reviews → Lista de reseñas cargadas desde la base de datos.
- onClick: (UserReview) -> Unit → Función lambda que define qué ocurre al pulsar sobre una reseña.

```
private val onClick: (UserReview) -> Unit
```

Para conectar las variables con los elementos del layout:

En el método onCreateViewHolder, se infla el layout item_resena.xml con LayoutInflater y se asocian los TextView correspondientes a cada campo (textTitulo, textGenero, textComentario).

Cada instancia de ReviewViewHolder contiene las referencias a estos elementos para poder asignarles los valores del modelo UserReview.

Flujo:

- **onCreateViewHolder()**
  - Se ejecuta cuando el RecyclerView necesita crear una nueva vista.
  - Infla el diseño item_resena.xml y devuelve un ReviewViewHolder asociado.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ReviewViewHolder {
    val view = LayoutInflater.from( context = parent.context)
        .inflate( resource = R.layout.item_resena, root = parent, attachToRoot = false)
    return ReviewViewHolder( itemView = view)
}
```

Layout del diseño de la reseña (item_resena.xml)

- **onBindViewHolder()**
  - Asigna los valores del objeto UserReview a los TextView del ítem correspondiente.
  - Configura el evento setOnClickListener para detectar clics sobre cada reseña y ejecutar la acción pasada como parámetro.
```

```
override fun onBindViewHolder(holder: ReviewViewHolder, position: Int) {
    val review = reviews[position]
    holder.titulo.text = review.titulo
    holder.genero.text = review.genero
    holder.comentario.text = review.comentario

    // Maneja el clic en un ítem y ejecuta la acción pasada como parámetro
    holder.itemView.setOnClickListener {
        onClick(review)
    }
}
```

- **getItemCount()**
  - Devuelve el número total de reseñas disponibles en la lista.

```
override fun getItemCount(): Int = reviews.size
```

- **updateData(newList: List<UserReview>)**
  - Permite actualizar la lista de reseñas (reviews) y refrescar el RecyclerView con los nuevos datos mediante notifyDataSetChanged().

```
// Método para actualizar los datos de la lista y refrescar la vista
1 Usage
fun updateData(newList: List<UserReview>) {
    reviews = newList
    notifyDataSetChanged()
}
}
```

MainActivity

Qué hace:

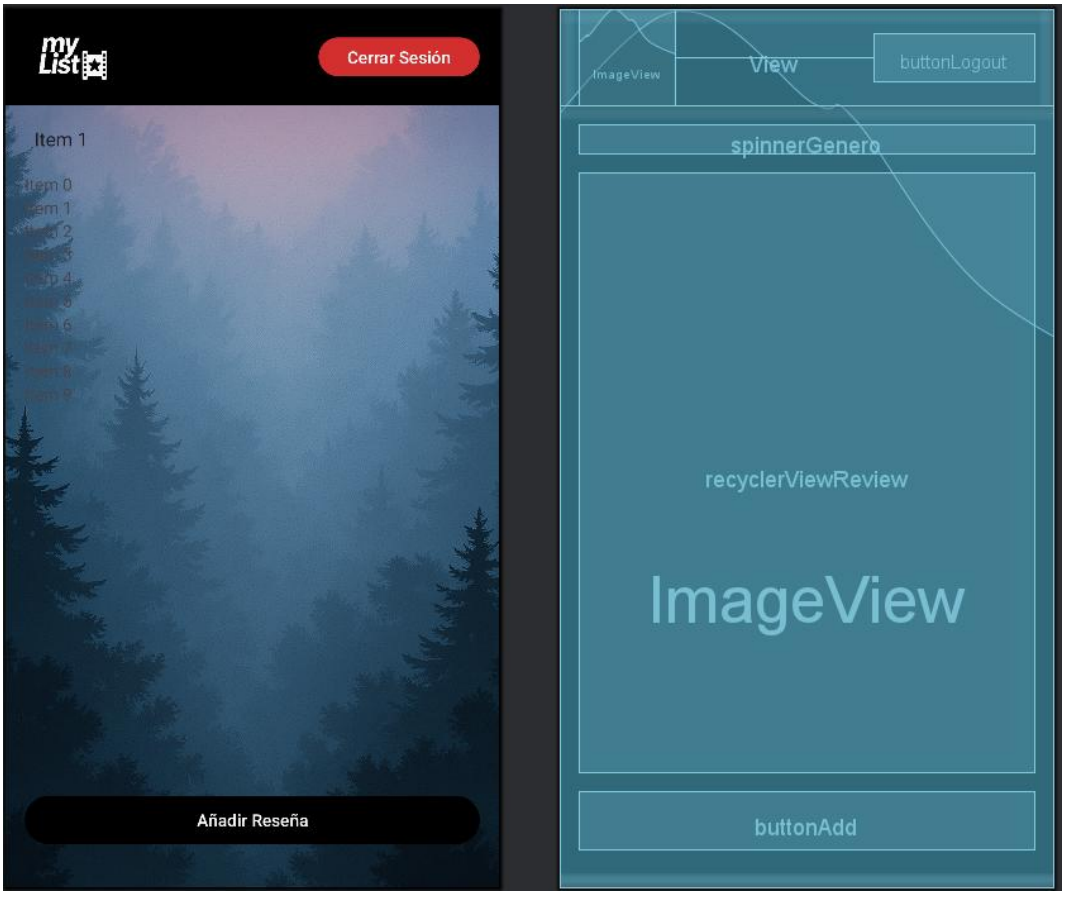
La clase MainActivity consitutye la actividad principal de una aplicación Android desarrollada en Kotlin, cuya función es gestionar reseñas de películas o contenidos audiovisuales.

El código gestiona el ciclo de vida principal de la aplicación y permite al usuario una vez autenticado realizar las siguientes acciones:

- Añadir, editar y eliminar reseñas.
- Filtrar las reseñas por género.
- Cerrar sesión.
- Ver los datos almacenados en una base de datos local persistente.

Layout asociado (diseño): activity_main.xml

```
setContentView(R.layout.activity_main)
```



Librerias:

- **import AppDatabase:** Permite acceder a la base de datos local de la app, generalmente usando Room.
- **import UserReview**: Define el modelo de datos para una reseña de usuario.
- **import UserReviewDao:** Proporciona métodos para acceder y manipular los datos de UserReview en la base de datos.
- **import android.content.Intent:** Permite iniciar nuevas actividades o servicios y pasar datos entre ellas.
- **import android.os.Bundle:** Contiene datos clave-valor que se pasan entre actividades o se usan al crear la actividad.
- **import android.widget.* :** Importa todos los widgets de Android (botones, textos, listas, etc.) para la interfaz de usuario.
- **import androidx.appcompat.app.AlertDialog:** Permite crear y mostrar cuadros de diálogo con opciones personalizadas.
- **import androidx.appcompat.app.AppCompatActivity:** Clase base para actividades que usan las funcionalidades modernas de compatibilidad.
- **import androidx.lifecycle.lifecycleScope:** Permite lanzar corutinas (Una corutina es una función que puede pausar su ejecución y reanudarse más tarde, permitiendo manejar tareas de manera asíncrona o cooperativa.) vinculadas al ciclo de vida de la actividad o fragmento.
- **import androidx.recyclerview.widget.LinearLayoutManager:** Define la disposición lineal (vertical u horizontal) de los elementos en un RecyclerView.
- **import androidx.recyclerview.widget.RecyclerView:** Permite mostrar listas de elementos de manera eficiente y flexible.
- **import com.google.firebase.FirebaseApp:** Inicializa Firebase en la aplicación.
- **import com.google.firebase.auth.FirebaseAuth:** Permite autenticar usuarios con Firebase (correo, Google, etc.).
- **import kotlinx.coroutines.Dispatchers**: Proporciona los hilos en los que se ejecutan corutinas (Main, IO, Default).
- **import kotlinx.coroutines.launch:** Lanza una corutina desde un scope determinado.
- **import kotlinx.coroutines.withContext:** Cambia el contexto de ejecución de una corutina a otro dispatcher.

Componentes:

- **Autenticación Firebase:**
  Se inicializa Firebase y se obtiene una instancia de FireaseAuth.

```
// Inicializar Firebase
FirebaseApp.initializeApp( context = this)
auth = FirebaseAuth.getInstance()
```

Se comprueba si existe un usuario autenticado. En caso contrario, se redirige a la pantalla de inicio de sesión (LoginActivity):

```
// Verificar si hay usuario logueado
val currentUser = auth.currentUser
if (currentUser == null) {
    val intent = Intent( packageContext = this,  cls = LoginActivity::class.java)
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    startActivity(intent)
    finish()
    return
}
```

- **Base de datos local con Room:**

```
// Configurar base de datos
db = AppDatabase.getDatabase( context = this)
reviewDao = db.userReviewDao()
```

Se obtiene una instancia de la base de datos AppDatabase.
Se accede al UserReviewDao, encargado de ejecutar las operaciones CRUD sobre la entidad UserReview.

- **Interfaz de usuario:**

```
// Vincular vistas
logoutButton = findViewById( id = R.id.buttonLogout)
addButton = findViewById( id = R.id.buttonAdd)
generoFiltro = findViewById( id = R.id.spinnerGenero)
recyclerView = findViewById( id = R.id.recyclerViewReview)
```

Los elementos de la interfaz se vinculan con los componentes del diseño actvity_main.xml:

- RecyclerView y Adaptador

```
// Configurar RecyclerView
recyclerView.layoutManager = LinearLayoutManager( context = this)
adapter = ReviewAdapter(reviews) { review -> mostrarDialogoEditar( resena = review) }
recyclerView.adapter = adapter
```

RecyclerView muestra la lista de reseñas almacenadas, usando un adaptador personalizado (ReviewAdapter) que actualiza los datos dinámicamente y permite editar reseñas al seleccionarlas.

- **Spinner (Filtro de género):**

```
generoFiltro.adapter =
    ArrayAdapter( context = this, resource = android.R.layout.simple_spinner_dropdown_item, objects = generos)
```

El Spinner permite filtrar las reseñas por género cinematográfico (acción, comedia, etc)
Al seleccionar un género se ejecuta el método filtrarLista()

Flujo:

- **Filtrado de reseñas:**

```
5 Usages
private fun filtrarLista(genero: String) {
    lifecycleScope.launch( context = Dispatchers.IO) {
        val lista = if (genero == "Todos") reviewDao.getAll() else reviewDao.getByGenero(genero)
        withContext( context = Dispatchers.Main) {
            reviews = lista
            adapter.updateData( newList = reviews)
        }
    }
}
```

Utiliza lifecyclopeScope y Dispatchers.IO para acceder a la base de datos sin bloquear el hilo principal.
Los resultados se muestran en la interfaz a través del adaptador.

- **Añadir una nueva reseña:**

```
private fun mostrarDialogoAñadir() {
    val dialogView = layoutInflater.inflate( resource = R.layout.dialog_resena,  root = null)
    val tituloInput = dialogView.findViewById<EditText>( id = R.id.editTitulo)
    val generoInput = dialogView.findViewById<EditText>( id = R.id.editGenero)
    val comentarioInput = dialogView.findViewById<EditText>( id = R.id.editComentario)
```

Muestra un cuadro de diálogo con campos para título, género y comentario.
Al confirmar, se crea un bojeto UserReview y se inserta en la base de datos mediante reviewDao.insert(resena)

- **Editar o eliminar una reseña existente:**

```
private fun mostrarDialogoEditar(resena: UserReview) {
    val dialogView = layoutInflater.inflate( resource = R.layout.dialog_resena,  root = null)
    val tituloInput = dialogView.findViewById<EditText>( id = R.id.editTitulo)
    val generoInput = dialogView.findViewById<EditText>( id = R.id.editGenero)
    val comentarioInput = dialogView.findViewById<EditText>( id = R.id.editComentario)
```

Carga los datos existentes en diálogo editable.
Permite guardar los cambios, eliminar la reseña o cancelar la operación.
Las modificaciones se actualizan en la base de datos y se refresca el listado.

- **Cierre de sesión:**

```
logoutButton.setOnClickListener {
    auth.signOut()
    val intent = Intent( packageContext = this,  cls = LoginActivity::class.java)
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    startActivity(intent)
    finish()
}
```

El usuario puede cerrar su sesión en Firebase, siendo redirigido al LoginActivity.

AndroidManifest.xml

Qué hace: Declara las actividades y define cuál actividad “clase” es la de inicio.

Actividades:

SplashActivity: Primera pantalla al abrir la app.

LoginActivity: Se abre desde SplashActivity.

RegisterActivity: Solo se abre desde LoginActivity.

MainActivity: Solo se abre desde Login o Register o desde el SplashActivity.

```
<!-- SplashActivity -->
<activity
    android:name=".SplashActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<!-- LoginActivity -->
<activity
    android:name=".LoginActivity"
    android:exported="true" />

<!-- RegisterActivity -->
<activity
    android:name=".RegisterActivity"
    android:exported="false" />

<!-- MainActivity -->
<activity
    android:name=".MainActivity"
    android:exported="false" />
```

6. LAYOUTS (DISEÑO)

- **activity_login.xml** (LoginActivity): Es la interfaz de la actividad de inicio de sesión. Contiene una ImageView (logo), dos EditText (correo electrónico y contraseña), un TextView con el texto "¿Has olvidado tu contraseña?", y los botones Registrar e Iniciar sesión.

- **activity_register.xml** (RegisterActivity): Es la interfaz de la actividad de registro. Incluye una ImageView (logo), tres EditText (correo electrónico, contraseña y repetir contraseña), y los botones Registrar y Volver al login.

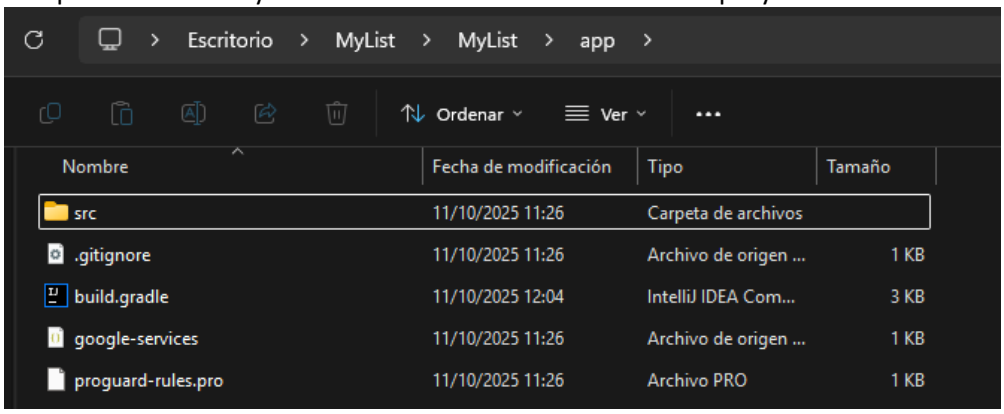
- **activity_main.xml** (MainActivity): Es la interfaz de la actividad principal de la aplicación. Presenta un fondo personalizado, una barra superior (TopBar), una ImageView (logo), el botón de cerrar sesión, un Spinner para filtrar por género, un RecyclerView para mostrar las reseñas, y el botón para agregar una nueva reseña.

- **dialog_resena.xml** (MainActivity): Define un diálogo personalizado para guardar y editar las reseñas, muestra tres TextView: Título, Género y Comentario.

- **item_resena.xml** (ReviewAdapter): Define el diseño de cada reseña individual dentro del RecyclerView. Se muestra en una card con tres TextView: Título, Género y Comentario.

7. PROBLEMAS ENCONTRADOS Y SUS SOLUCIONES

- Nos encontramos con que el archivo google-services.json estaba mal colocado, debía de estar dentro de la carpeta app del proyecto, y al no estar allí, Firebase no se inicializaba y todas las llamadas aFirebaseAuth fallaban, después de esto detectamos dependencias de Firebase incompatibles, algunas versiones de firebase-auth, firebase-core y otras no coincidían, lo que generaba ciertos errores en compilación y en tiempo de ejecución. Lo solucionamos asegurándonos de usar en build.gradle las versiones recomendadas y compatibles entre sí y con la versión de Android de nuestro proyecto.



```
implementation(platform( notation = "com.google.firebase:firebase-bom:34.3.0"))
implementation("com.google.firebase:firebase-storage")
implementation("com.google.firebase:firebase-auth")
implementation("com.google.firebase:firebase-firestore")
```

- Nos encontramos con un problema en la base de datos, al principio teníamos la versión 1, pero al realizar cambios en la estructura de UserReview (la tabla), la aplicación empezaba a dar errores en la base de datos debido a una incompatibilidad con la tabla existente. Para solucionarlo lo que hicimos fue incrementar la versión en AppDatabase.

```
@Database(entities = [UserReview::class], version = 2, exportSchema = false)
```

8. DIAGRAMA DE CLASES Y ARQUITECTURA.

Diagrama de clases:

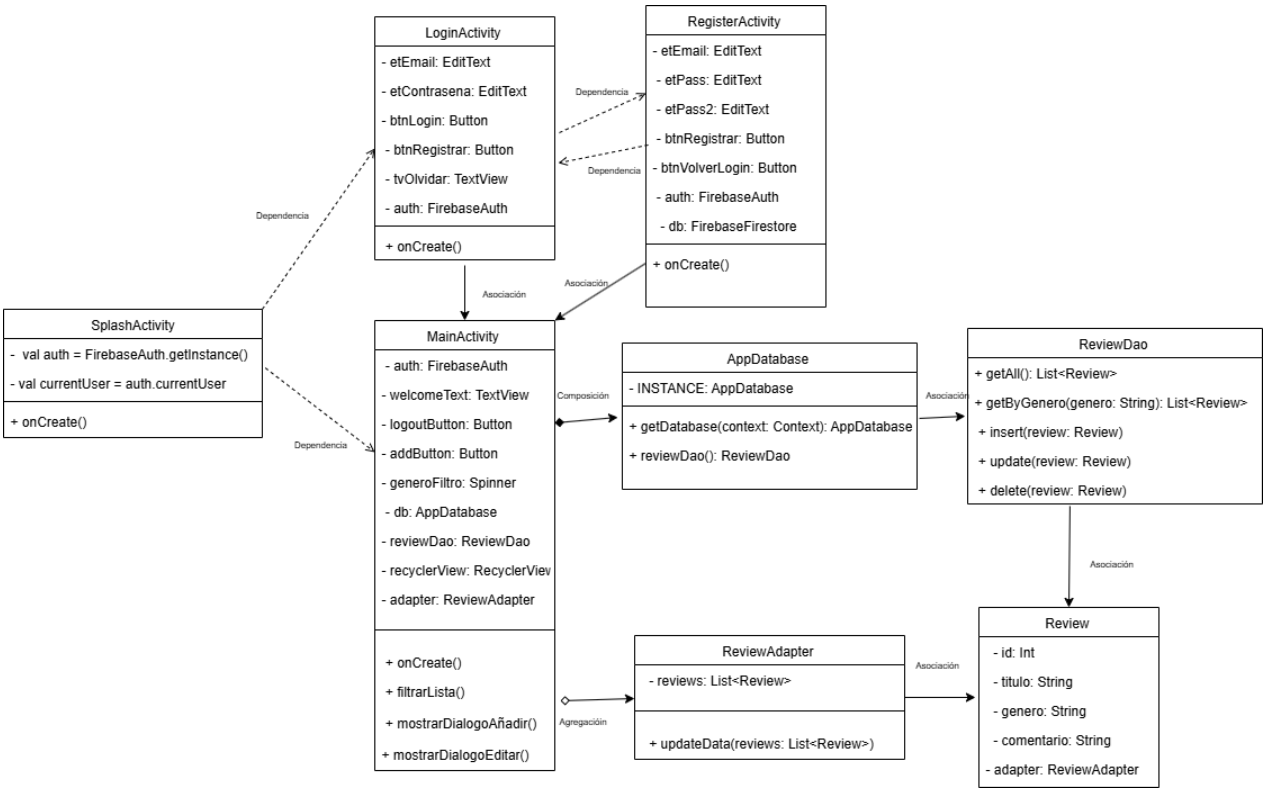
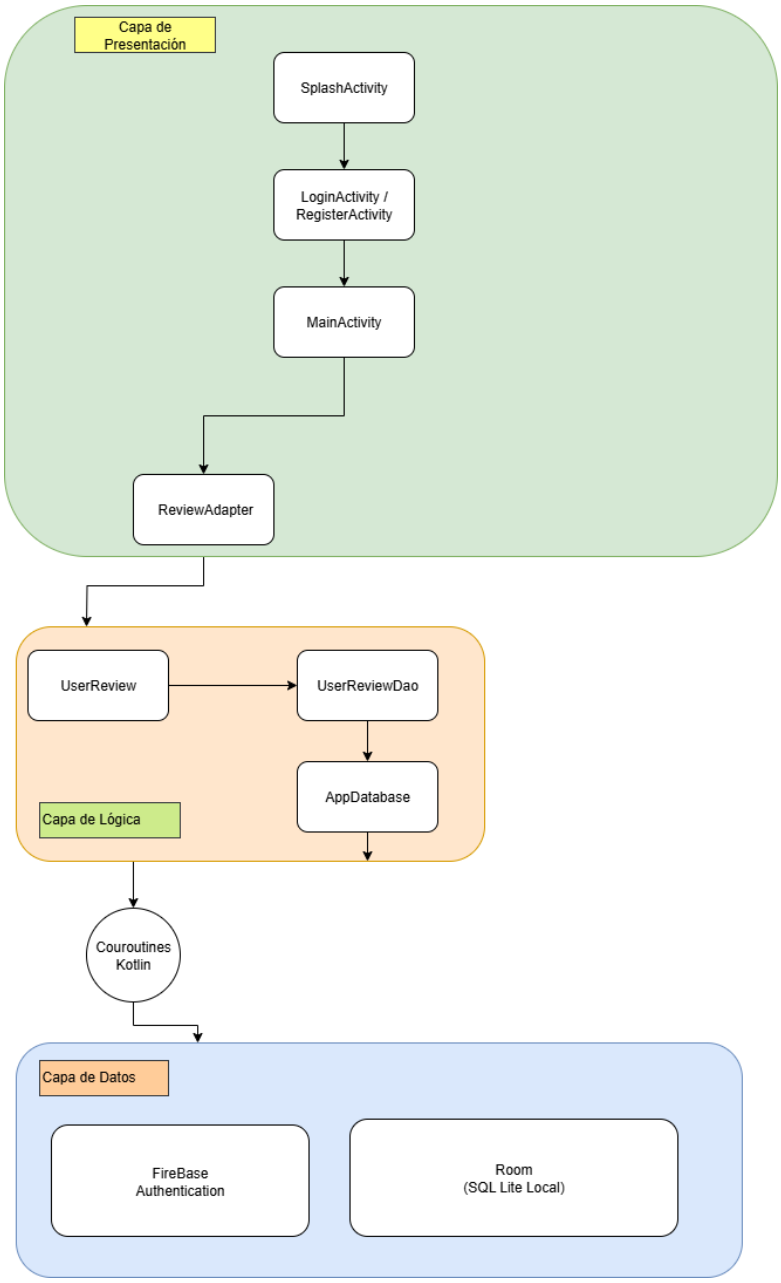


Diagrama de arquitectura:


Capa de Presentación
SplashActivity
LoginActivity / RegisterActivity
MainActivity
ReviewAdapter
UserReview
UserReviewDao
AppDatabase
Capa de Lógica
Couroutines Kotlin
Capa de Datos
FireBase Authentication
Room (SQL Lite Local)