

PRACTICA IMDBAPP V2.0

**PROGRAMACIÓN
MULTIMEDIA**

Alejandro Sánchez Gil
2ºDAM CURSO 2024/2025



Castilla-La Mancha

ÍNDICE

1. Enlace Github.....	1
2. Introducción.....	1
2.1 Proposito.....	1
3. Alcance.....	1
3.1 Objetivos.....	1
4. Requisitos del sistema.....	1
4.1 Requisitos funcionales.....	1
4.2 Requisitos no funcionales.....	2
5. Diseño de la arquitectura.....	2
5.1 Arquitectura de la aplicación.....	2
5.2 Componentes principales.....	2
6. Diseño de la interfaz de usuario.....	3
6.1 LoginActivity:.....	3
6.2 EditUserActivity:.....	4
6.3 EditAddressActivity:.....	5
6.4 MainActivity:.....	6
7. Esquema de base de datos.....	7
8. Diagramas.....	8
8.1 Diagrama UML.....	8
8.2 Diagrama de casos.....	9
8.3 Diagrama E-R (Tablas).....	9
9. Lógica del software.....	10
9.1 Descripción del flujo de la aplicación.....	10
9.2 Código de lógica principal.....	10
10. Pruebas.....	26
10.1 Estrategia de pruebas.....	26
11. Implementación y despliegue.....	26
10.1 Herramientas de desarrollo.....	26
10.2 Proceso de despligue.....	26
11. Mantenimiento y actualizaciones.....	26
Mejoras futuras.....	26
12. Bilbiografía.....	27

1. Enlace Github

https://github.com/AlexSG002/SanchezGil_AlejandroIMDbAPPv2.0

2. Introducción

• 2.1 Proposito

El objetivo de este documento es facilitar manuales de usuario, información y guía sobre el funcionamiento y las distintas características de la aplicación “IMDbApp V2.0” solicitada en la Unidad 5, este documento contará con diversos apartados para facilitar la comprensión completa del programa al usuario

3. Alcance

La aplicación esta desarrollada para dispositivos con sistema operativo Android con versión mínima 7.0 (Nougat) y superior, cuenta con una interfaz de usuario clara y sencilla para facilitar la interacción del usuario eliminando elementos complejos y enfocándonos en utilidades simples.

• 3.1 Objetivos

- Permitir a los usuarios iniciar sesión con Facebook y a través de email y clave.
- Facilitar al usuario las opciones de edición de perfil (Nombre, dirección, teléfono, foto).
- Guardar tanto en local como en la nube los datos del usuario y de las películas favoritas de cada usuario.
- Obtención de la dirección a través de la herramienta places con autocompleción de la dirección indicada y señalización de la misma a través de un mapa de Google Maps.
- Guardar tanto en la base de datos local como en la nube los logs de inicio de sesión y cierre de sesión, manejados a través del ciclo de vida de la app, y mantener un historial en la nube.
- Sincronización de la base de datos en tiempo real al modificar algún dato del usuario o al añadir o eliminar una película de favoritos.

4. Requisitos del sistema

• 4.1 Requisitos funcionales

- La aplicación debe de ejecutar correctamente el inicio de sesión por los nuevos métodos Facebook y email.
- La aplicación debe mostrar correctamente la información del usuario en las pestañas pertinentes
- La aplicación debe modificar correctamente la información del usuario.

- La aplicación debe mostrar correctamente el mapa con la dirección indicada por el usuario.

- **4.2 Requisitos no funcionales**

- La aplicación debe solicitar todos los permisos necesarios para funcionar correctamente
- La aplicación debe de informar al usuario de las actualizaciones que ocurran en su entorno.

5. Diseño de la arquitectura

- **5.1 Arquitectura de la aplicación**

Principalmente tendríamos una arquitectura MVVM ya que contamos con Actividades que manejan la lógica y modelos con información, también tenemos vistas donde representamos elementos gráficos que interactúan con el usuario y ViewModel aunque no visible el usuario puede modificar su información y verla representada en una de las vistas.

- **5.2 Componentes principales**

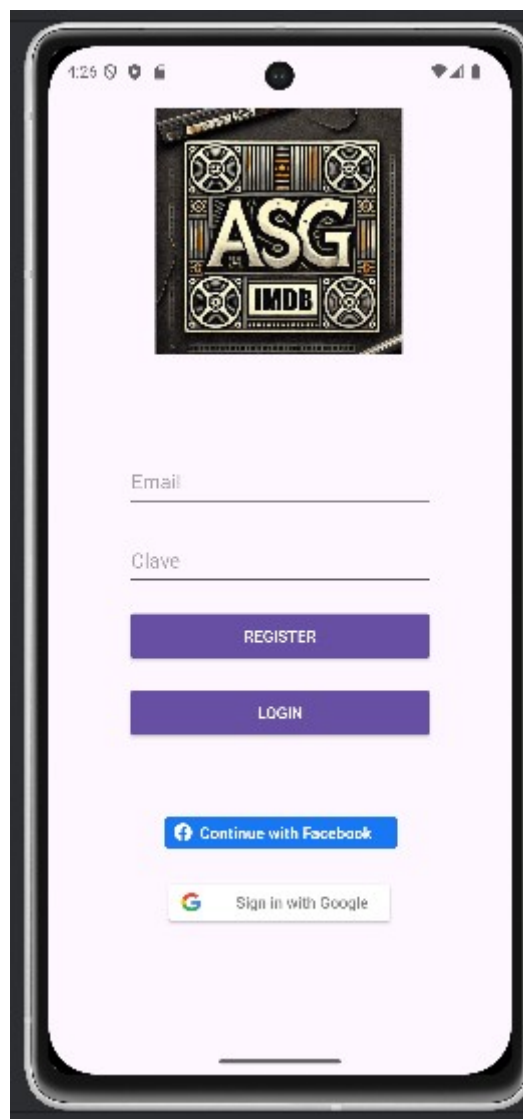
- LoginActivity: Incluye las nuevas funciones del inicio de sesión a través de Facebook y el registro e inicio de sesión con email y clave además de la obtención de los datos de la nube a través de Firestore, también una instancia de la clase Lifecycle para manejar la entrada de logs.[3][4][5]
- MainActivity: Declara instancias de las clases de sincronización para mantener sincronizadas la base de datos local y en la nube, manejando la generación de logs utilizando los métodos en la clase Lifecycle para actualizar en tiempo real al actualizar la base de datos, también encontramos el método para descryptar los datos que recibimos de la base de datos, ya que desde MainActivity los pasamos a EditUser donde podremos ver los datos descryptados.[2]
- EditUserActivity: Es la actividad en la que podremos ver y editar todos los detalles de usuario, encontramos campos para modificar el nombre, el teléfono pudiendo seleccionar el prefijo de nuestro país, la dirección a través de otra Actividad (EditAddressActivity) y la imagen que la podremos cargar a través de la cámara, la galería o una URL, también podremos visualizar nuestro correo electrónico pero no se puede modificar.
- EditAddressActivity: En esta actividad seleccionaremos nuestra dirección a través de la herramienta de Google Places, que nos mostrará un mapa con la dirección que introduzcamos, la cual aparecerá en una lista que se irá autocompletando a medida que vayamos escribiendo una dirección.[6]
- AppLifecycleManager: Esta clase es la que maneja la lógica del registro del historial de inicio y cierre de sesión, nos guarda en formato texto la fecha y hora de ambas acciones, que se manejan en métodos de tipo Activity, como onResume y onPause por ejemplo.[1]

- **RapidApiKeyManager:** Esta clase tiene como función renovarnos automáticamente la key de RapidApi cuando se nos agotan las llamadas de una key, simplemente pasamos a la siguiente Key.
- **HomeFragment:** Implementada la función de cambio de Keys.
- **FavoritesSync:** Es la clase que se encarga de la lógica de la actualización de la base de datos en la nube de la tabla favoritos, obteniendo los datos de la base de datos local, sus métodos se ejecutan en otras clases como MainActivity.
- **UsersSync:** Igual que en FavoritesSync, se encarga de la lógica de la actualización de la base de datos en la nube, en este caso de la tabla usuarios.
- **IMDbDatabaseHelper:** Modificada la clase de FavoritesDatabaseHelper de la versión anterior para incluir la tabla de usuarios, al igual que los métodos de inserción de usuario y de actualización de login y logout local.
- **MovieAdapter:** Simplemente implementa una instancia de FavoritesSync para que al añadir o eliminar una película de la base de datos local se actualice también en la nube en tiempo real.

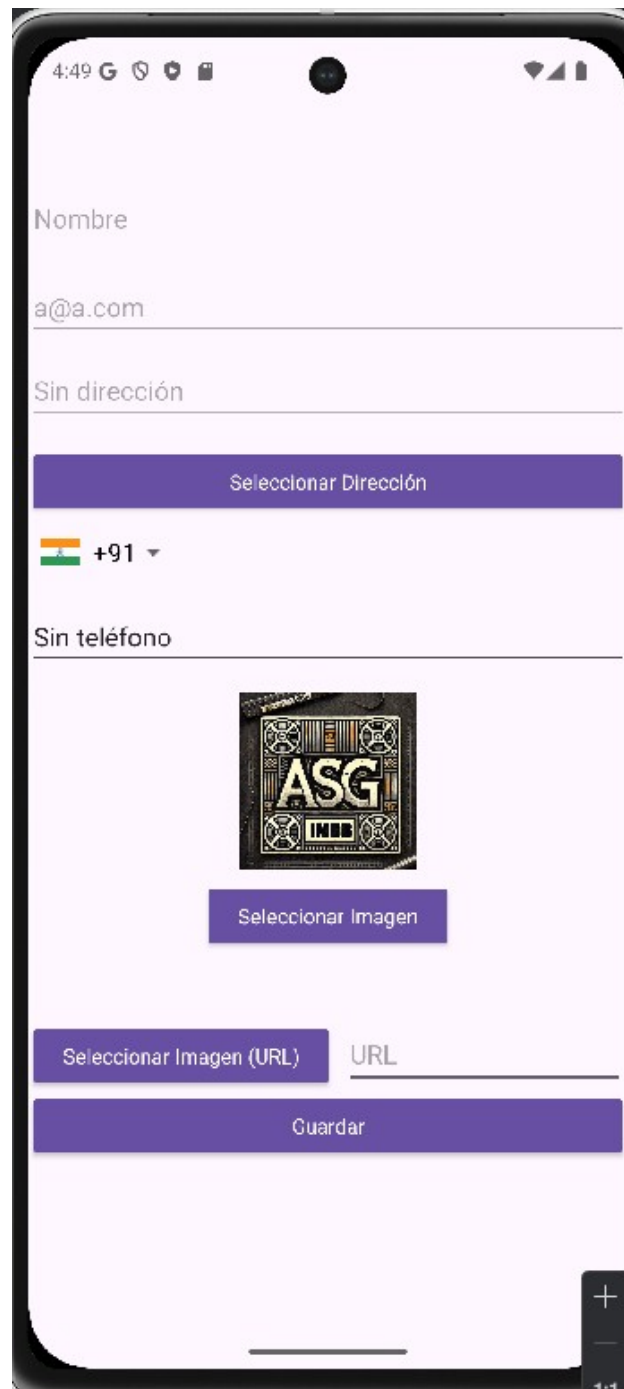
6. Diseño de la interfaz de usuario

• 6.1 LoginActivity:

Podemos ver introducidos los botones y los campos de texto para las nuevas funciones de inicio de sesión:



- **6.2 EditUserActivity:**



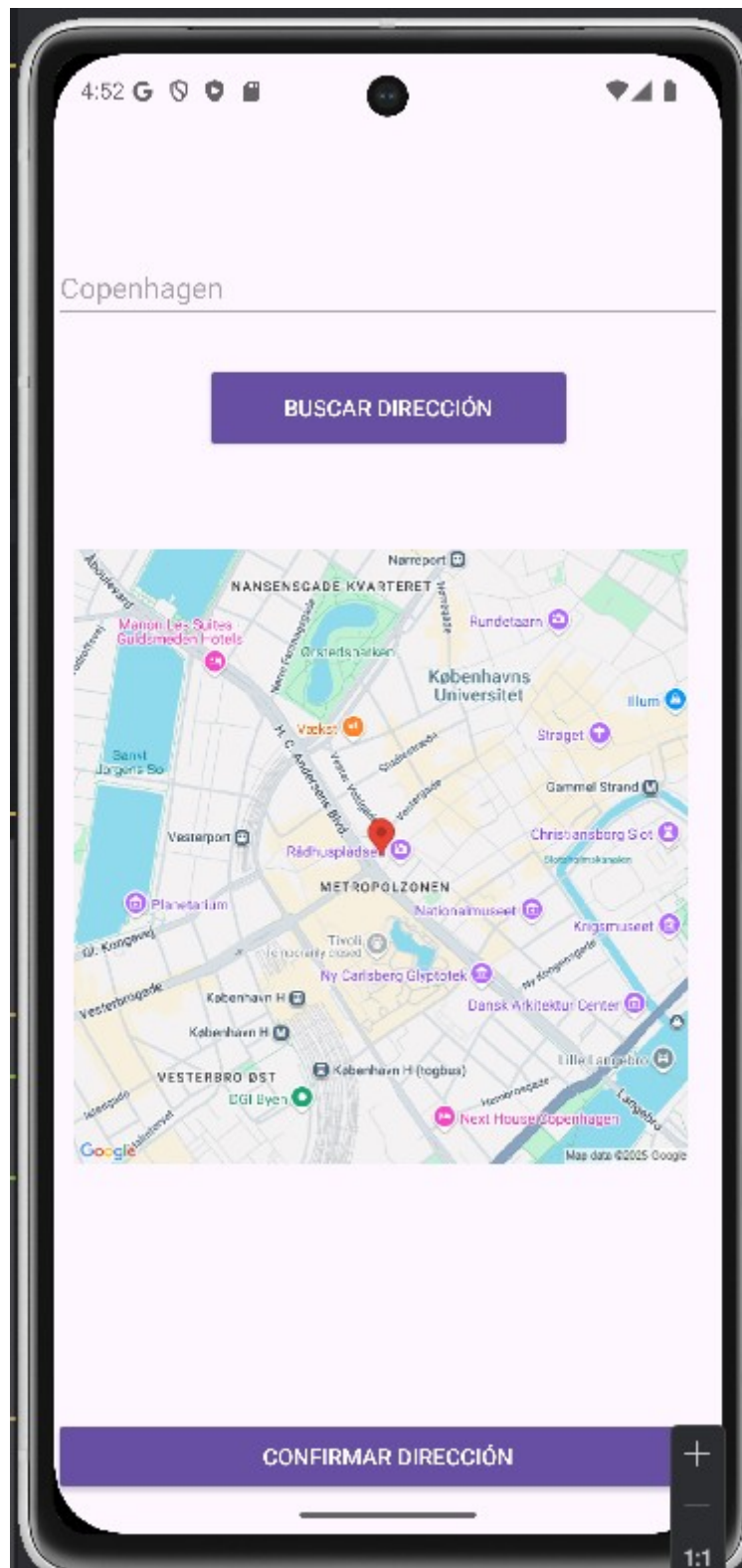
The screenshot shows a mobile application interface for editing user information. The form includes the following fields and controls:

- Nombre:** A text input field.
- a@a.com:** An email address input field.
- Sin dirección:** A text input field for the address.
- Seleccionar Dirección:** A purple button to select an address.
- +91:** A dropdown menu for the country code, currently showing the Indian flag and '+91'.
- Sin teléfono:** A text input field for the phone number.
- Image Selection:** A section containing a placeholder image (a square with 'ASG' and 'IMB' text) and a purple button labeled 'Seleccionar Imagen'.
- Image Selection (URL):** A purple button to select an image from a URL.
- URL:** A text input field for the image URL.
- Guardar:** A large purple button at the bottom to save the changes.

The interface is clean with a light purple background and dark text. The status bar at the top shows the time as 4:49 and various system icons. A small '+' icon is visible in the bottom right corner of the app area.

Aquí podremos insertar los datos del usuario, tras hacer clic en guardar se introducirán todos los datos de campos rellenados en la base de datos local y en la nube.

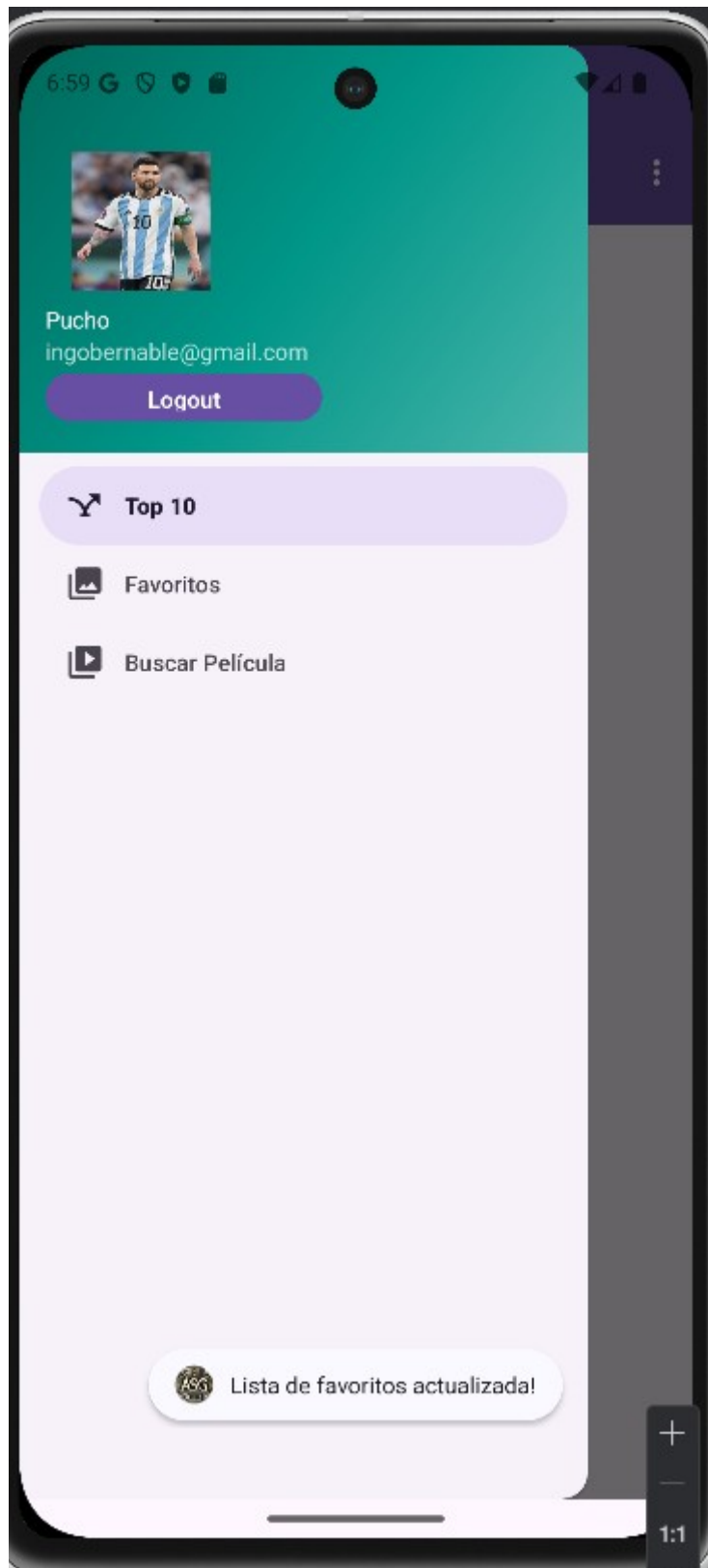
- **6.3 EditAddressActivity:**



Aquí es donde podremos introducir nuestra dirección y nos aparecerá un mapa de Google Maps con la ubicación que hayamos seleccionado.

- **6.4 MainActivity:**

Aquí podremos ver reflejados los cambios de la imagen y el nombre que hemos hecho en EditUserActivity, en la barra lateral:



7. Esquema de base de datos

Ampliación de la base de datos local con la nueva tabla de usuarios:

idUsuario	nombre	email	loginRe...	logout...	direccion	telefono
1UarxKL...	Nombre ...	xxthedar...	2025-02...	2025-02...	Sin dirección	Sin teléfono
3Qe5Sw...	Alex Lan...	lanncerr...	2025-02...		Sin dirección	Sin teléfono
ek73pdkl...		a@a.com	2025-02...	2025-02...	Sin dirección	Sin teléfono
tLVPCLe...	Nombre ...	copete@...	2025-02...	2025-02...	Sin dirección	Sin teléfono
mzkIYPe...	Nombre ...	aa@aa.c...	2025-02...	2025-02...	Sin dirección	Sin teléfono
sCB5Y7...		anelka@...	2025-02...	2025-02...	Sin dirección	Sin teléfono
lLgzV4ff...		noesta@...	2025-02...	2025-02...	Sin dirección	Sin teléfono
gk3OwC...	Nombre ...	b@b.com	2025-02...	2025-02...	Sin dirección	Sin teléfono
nfFhutA...	Pruebas ...	pruebas...	2025-02...	2025-02...	Sin dirección	Sin teléfono
jt06xUe...	Pucho	ingobern...	2025-02...	2025-02...	BUPumenGrr8817LtMNxbCA==	+8RHPG7efUjzrvz8AnU5yw==

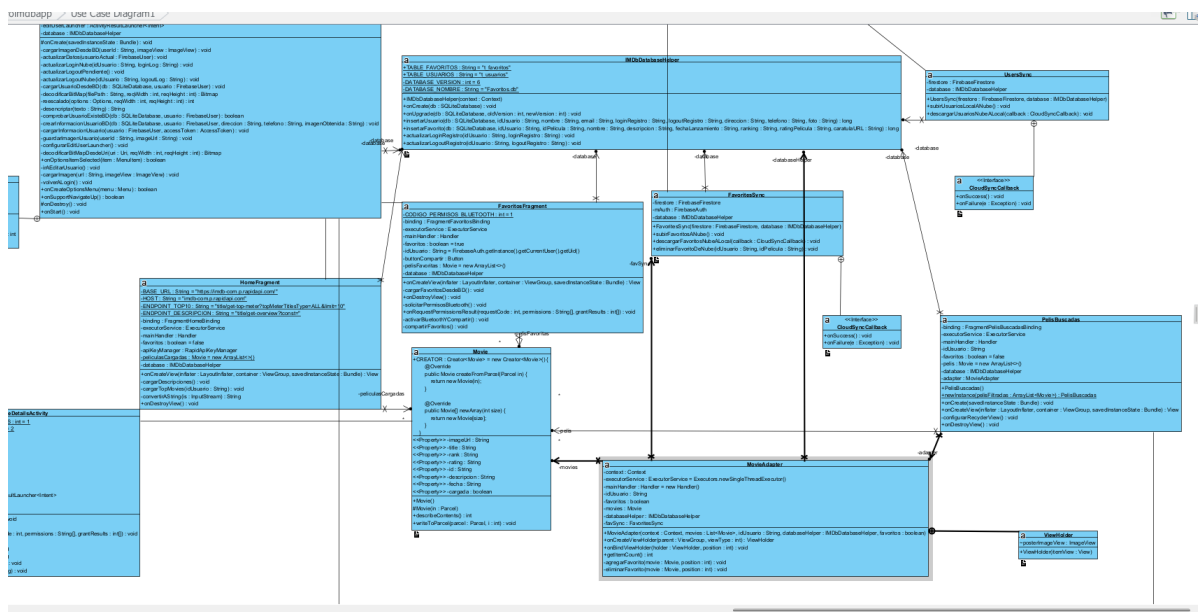
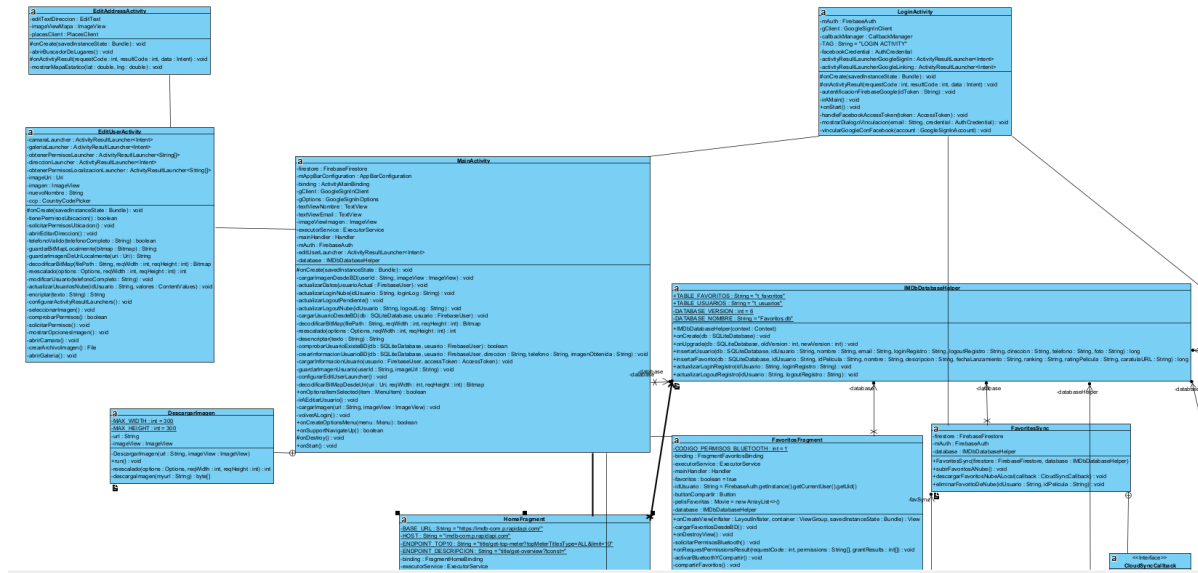
También hemos introducido la base de datos en la nube a través de Firestore:

Collection	Document ID	Field	Value
usuarios	jt06xUeovegugpdA3RKuofH5wXy1	direccion	"BUPumenGrr8817LtMNxbCA=="
		email	"ingoernable@gmail.com"
		foto	"/data/user/0/com.pmdm.snchezgil_alejandroimdbapp/files/user_image_1"
		idUsuario	"jt06xUeovegugpdA3RKuofH5wXy1"
		loginRegistro	[{"0": "2025-02-04 18:57:00", "1": "2025-02-04 18:58:24", "2": "2025-02-04 18:59:27"}]
		logoutRegistro	[{"0": "2025-02-04 18:56:44", "1": "2025-02-04 18:58:14", "2": "2025-02-04 18:58:10", "3": "2025-02-04 18:58:31", "4": "2025-02-04 18:58:24"}]
		nombre	"Pucho"

Que se actualizan en tiempo real para mantenerse sincronizadas.

8. Diagramas

- **8.1 Diagrama UML**



9. Lógica del software

• 9.1 Descripción del flujo de la aplicación

1. Inicio: El usuario debe poder iniciar sesión con cualquiera de los 3 métodos ofrecidos.
2. Una vez iniciada la sesión se abrirá la pestaña principal donde será recibido por el top 10
3. Al hacer clic en los 3 puntos del menú podrá hacer clic en EditUser lo que le redirigirá a otra Actividad donde podrá editar sus datos
4. En la actividad EditUser podemos seleccionar establecer una dirección al hacer clic en el botón nos llevará a otra actividad donde podremos introducir nuestra ubicación y la podremos ver representada en un mapa de Google Maps (EditAddressActivity).
5. El usuario puede guardar su dirección y volverá a EditUser donde puede terminar de introducir sus datos y guardar los datos que se verán reflejados en la barra lateral de detalles de usuario.

• 9.2 Código de lógica principal

- Nuevos métodos de inicio de sesión en LoginActivity:

```
Button buttonRegistroEmail = findViewById(R.id.buttonRegistro);
Button buttonInicioEmail = findViewById(R.id.buttonLogin);
//Botón para registrar una cuenta por mail.
buttonRegistroEmail.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String mail = "";
        String password = "";
        if (editTextEmail.getText().toString().matches(Regex: "^[\\w-\\.]+@([\\w-]+\\.){1,4}[\\w-]{2,4}$")) {
            if (!String.valueOf(editTextPassword.getText()).isEmpty()) {
                mail = String.valueOf(editTextEmail.getText());
                password = String.valueOf(editTextPassword.getText());
            } else {
                Toast.makeText(context: LoginActivity.this, text: "La clave no puede estar vacia", Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(context: LoginActivity.this, text: "El email introducido no es correcto", Toast.LENGTH_SHORT).show();
            return;
        }
        mAuth.createUserWithEmailAndPassword(mail, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    Toast.makeText(context: LoginActivity.this, text: "Registro exitoso", Toast.LENGTH_SHORT).show();
                    irAMain();
                } else if (task.getException() instanceof FirebaseAuthUserCollisionException) {
                    FirebaseAuthUserCollisionException exception = (FirebaseAuthUserCollisionException) task.getException();
                    String email = exception.getEmail();
                    Toast.makeText(context: LoginActivity.this, text: "Cuenta ya en uso, porfavor inicia sesión con el método seleccionado.", Toast.LENGTH_SHORT).s
                } else {
                    Toast.makeText(context: LoginActivity.this, text: "Registro fallido, revisa si la contraseña que has introducido tiene más de 6 caracteres.", T
                }
            }
        });
    }
});
```

Aquí comprobamos que el mail sea correcto antes de intentar crear la cuenta con el método `mAuth.createUser` también comprobamos si la cuenta ya está en uso para que no se pueda registrar en caso de tener otro tipo de inicio de sesión activado en la cuenta.

El método de inicio de sesión es igual pero utilizamos `mAuth.signInWithEmailAndPass`.

Aquí tenemos el nuevo método de inicio de sesión a través de facebook, utilizando el AccessToken para autenticar el usuario:

```
}
//Método que maneja el inicio de sesión con Facebook.
private void handleFacebookAccessToken(AccessToken token) { 1 usage
    Log.d(TAG, "handleFacebookAccessToken:" + token);

    AuthCredential credential = FacebookAuthProvider.getCredential(token.getToken());
    FirebaseAuth user = mAuth.getCurrentUser();
    // Usuario no autenticado, intenta iniciar sesión con Facebook
    mAuth.signInWithCredential(credential)
        .addOnCompleteListener( activity: this, task -> {
            if (task.isSuccessful()) {
                // Inicio de sesión exitoso
                Log.d(TAG, "signInWithCredential:success");
                FirebaseAuth newUser = mAuth.getCurrentUser();
                Toast.makeText( context: LoginActivity.this, text: "Autenticación con Facebook exitosa.", Toast.LENGTH_SHORT).show();
                irAMain();
            } else {
                if (task.getException() instanceof FirebaseAuthUserCollisionException) {
                    // Manejar la colisión de cuentas
                    FirebaseAuthUserCollisionException exception = (FirebaseAuthUserCollisionException) task.getException();
                    String email = exception.getEmail();
                    Log.w(TAG, "signInWithCredential:failure", task.getException());

                    // Luego le muestra un diálogo al usuario para iniciar sesión con Google y luego vincular.
                    mostrarDialogoVinculacion(email, credential);
                } else {
                    // Otro tipo de error
                    Log.w(TAG, "signInWithCredential:failure", task.getException());
                    Toast.makeText( context: LoginActivity.this, text: "Autenticación con Facebook fallida.", Toast.LENGTH_SHORT).show();
                }
            }
        });
}
```

También comprueba que si hay una cuenta de Google iniciada nos mostrará un dialogo para vincular las cuentas que ejecutará un nuevo método de vinculación “linkWithCredential”:

```
}
//Método para vincular Google con Facebook.
private void vincularGoogleConFacebook(GoogleSignInAccount account) { 1 usage
    AuthCredential googleCredential = GoogleAuthProvider.getCredential(account.getIdToken(), si: null);

    // Reautenticar con Google
    mAuth.signInWithCredential(googleCredential)
        .addOnCompleteListener( activity: this, task -> {
            if (task.isSuccessful()) {
                // Vincular la credencial de Facebook con la cuenta de Google
                mAuth.getCurrentUser().linkWithCredential(facebookCredential)
                    .addOnCompleteListener( activity: this, linkTask -> {
                        if (linkTask.isSuccessful()) {
                            Log.d(TAG, "linkWithCredential:success");
                            FirebaseAuth linkedUser = linkTask.getResult().getUser();
                            Toast.makeText( context: LoginActivity.this, text: "Cuentas de Google y Facebook vinculadas exitosamente.", Toast.LENGTH_SHORT).show();
                            irAMain();
                        } else {
                            Log.w(TAG, "linkWithCredential:failure", linkTask.getException());
                            Toast.makeText( context: LoginActivity.this, text: "Vinculación de cuentas fallida.", Toast.LENGTH_SHORT).show();
                        }
                    });
            } else {
                Log.w(TAG, "signInWithCredential:failure during linking", task.getException());
                Toast.makeText( context: LoginActivity.this, text: "Autenticación con Google fallida para vincular.", Toast.LENGTH_SHORT).show();
            }
        });
}
```

- MainActivity:

Aquí en MainActivity hemos introducido múltiples métodos con diferentes funciones, empezamos obteniendo los usuarios de la nube para descargarlos a local, esto lo hacemos con un método que está manejado dentro de UsersSync, a la vez también descargamos la tabla de favoritos de la base de datos de la nube, en esencia vinculando las dos bases de datos en tiempo real:

```
protected void onCreate(Bundle savedInstanceState) {
    //Configuramos el EditUserLauncher donde pasamos los datos.
    configurarEditUserLauncher();
    //Declaramos e inicializamos una nueva instancia de UsersSync para vincular los usuarios de las bases de datos.
    UsersSync usersSync = new UsersSync(firestore, database);
    //Descargamos los usuarios existentes en la nube al almacenamiento local.
    usersSync.descargarUsuariosNubeALocal(new UsersSync.CloudSyncCallback() {
        @Override 1 usage
        public void onSuccess() {
            SQLiteDatabase nuevoDb = database.getReadableDatabase();
            // Una vez sincronizados, comprobamos si el usuario existe en la base local.
            if (comprobarUsuarioExisteBD(nuevoDb, usuario)) {
                cargarUsuarioDesdeBD(nuevoDb, usuario);
            } else {
                // Si no existe, se crea la información local
                crearInformacionUsuarioBD(nuevoDb, usuario, direccion: null, telefono: null, imagenObtenida: null);
                cargarInformacionUsuario(usuario, accessToken);
            }
            //Actualizamos los datos de usuario para actualizar los logins y subimos la información a la nube.
            actualizarDatos(usuario);
            usersSync.subirUsuariosLocalANube();
        }
        //En caso de error no hace nada, log para verificar estado de la base de datos.
        @Override 1 usage
        public void onFailure(Exception e) {
            Log.e("tag: MainActivity", msg: "Error sincronizando datos desde la nube: " + e.getMessage());
        }
    });
    //Declaramos e inicializamos una nueva instancia de la base de datos de las películas favoritas para sincronizar las bases de datos.
    FavoritesSync favoritesSync = new FavoritesSync(firestore, database);
    //Nos descargamos la base de datos de la Nube y la actualizamos a la local.
    favoritesSync.descargarFavoritosNubeALocal(new FavoritesSync.CloudSyncCallback() {
        @Override 1 usage
        public void onSuccess() {
            Toast.makeText(context: MainActivity.this, text: "Lista de favoritos actualizada!", Toast.LENGTH_SHORT).show();
        }
        @Override 1 usage
        public void onFailure(Exception e) {
            Log.e("tag: MainActivity", msg: "Error sincronizando datos desde la nube: " + e.getMessage());
        }
    });
}
```

Tras esto tenemos la actualización de datos de log tanto en la nube como de manera local, ya que al iniciar la aplicación se deben vincular los logs:


```

221 }
222
223 //Método para actualizar los datos de logs.
224 @ private void actualizarDatos(FirebaseUser usuarioActual) { 1 usage
225     SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");
226     String formattedDate = sdf.format(System.currentTimeMillis());
227     String loginLog = formattedDate;
228     IMDBDatabaseHelper dbHelper = new IMDBDatabaseHelper(getApplicationContext());
229     dbHelper.actualizarLoginRegistro(usuarioActual.getId(), loginLog);
230     actualizarLoginNube(usuarioActual.getId(), loginLog);
231     actualizarLogoutPendiente();
232 }
233 //Método para actualizar el login de la nube.
234 private void actualizarLoginNube(String idUsuario, String loginLog) { 1 usage
235     FirebaseFirestore firestore = FirebaseFirestore.getInstance();
236     Map<String, Object> data = new HashMap<>();
237     data.put( key: "loginRegistro", FieldValue.arrayUnion(loginLog));
238     firestore.collection( collectionPath: "usuarios").document(idUsuario) DocumentReference
239         .set(data, SetOptions.merge()) Task<Void>
240         .addOnSuccessListener(aVoid ->
241             Log.d( tag: "Firestore", msg: "Login de usuario " + idUsuario + " actualizado en la nube"))
242         .addOnFailureListener(e ->
243             Log.e( tag: "Firestore", msg: "Error actualizando login de usuario " + idUsuario, e));
244 }
245 //Método que guarda el último logout en SharedPreferences en caso de que el usuario destruya la actividad para que al hacer login se vuelva a guardar.
246 private void actualizarLogoutPendiente() { 1 usage
247     SharedPreferences preferences = getSharedPreferences( name: "user_prefs", Context.MODE_PRIVATE);
248     String lastLogout = preferences.getString( key: "last_logout", defaultValue: null);
249     if (lastLogout != null) {
250         IMDBDatabaseHelper dbHelper = new IMDBDatabaseHelper(getApplicationContext());
251         FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
252         if (user != null) {
253             dbHelper.actualizarLogoutRegistro(user.getId(), lastLogout);
254             Log.d( tag: "LoginActivity", msg: "Logout pendiente (" + lastLogout + ") guardado en la BD local para: " + user.getEmail());
255         }
256     }
257     SharedPreferences.Editor editor = preferences.edit();
258     editor.remove( key: "last_logout");
259     editor.apply();
260 }

```

```

261     editor.apply();
262 }
263 //Método para actualizar el logout de la nube.
264 private void actualizarLogoutNube(String idUsuario, String logoutLog) { 1 usage
265     FirebaseFirestore firestore = FirebaseFirestore.getInstance();
266     Map<String, Object> data = new HashMap<>();
267     data.put( key: "logoutRegistro", FieldValue.arrayUnion(logoutLog));
268     firestore.collection( collectionPath: "usuarios").document(idUsuario) DocumentReference
269         .set(data, SetOptions.merge()) Task<Void>
270         .addOnSuccessListener(aVoid ->
271             Log.d( tag: "Firestore", msg: "Logout de usuario " + idUsuario + " actualizado en la nube"))
272         .addOnFailureListener(e ->
273             Log.e( tag: "Firestore", msg: "Error actualizando logout de usuario " + idUsuario, e));
274 }

```

También obtenemos los datos del usuario desde la base de datos para establecer sus datos en la barra lateral en caso de que se hayan actualizado:

```

281 //Método para cargar el usuario desde la base de datos local.
282 @ private void cargarUsuarioDesdeBD(SQLiteDatabase db, FirebaseUser usuario) { 1 usage
283     String sql = "SELECT * FROM t_usuarios WHERE idUsuario = ?";
284     Cursor cursor = db.rawQuery(sql, new String[]{usuario.getId()});
285
286     if (cursor.moveToFirst()) {
287         int colNombre = cursor.getColumnIndex( key: "nombre");
288         int colEmail = cursor.getColumnIndex( key: "email");
289         int colFoto = cursor.getColumnIndex( key: "foto");
290
291         String nombre = "";
292         String email = "";
293         String foto = "";
294
295         if (colNombre != -1) {
296             nombre = cursor.getString(colNombre);
297         }
298         if (colEmail != -1) {
299             email = cursor.getString(colEmail);
300         }
301         if (colFoto != -1) {
302             foto = cursor.getString(colFoto);
303         }
304
305         //Cargamos los datos en la barra lateral de información de usuario.
306         textViewNombre.setText(nombre);
307         textViewEmail.setText(email);
308
309         if (foto != null && !foto.isEmpty()) {
310             File imageFile = new File(foto);
311             if (imageFile.exists()) {
312                 Bitmap scaledBitmap = BitmapFactory.decodeFile(imageFile.getAbsolutePath(), reqWidth: 300, reqHeight: 300);
313                 imageViewImagen.setImageBitmap(scaledBitmap);
314             } else {
315                 Log.e( tag: "MainActivity", msg: "Archivo de imagen no encontrado: " + foto);
316             }
317         } else {
318             //Cargamos una imagen por defecto si no se encuentra la foto.
319             imageViewImagen.setImageResource(R.drawable.default_image);
320         }
321     }
322 }

```

También tenemos los nuevos métodos para decodificar la imagen que obtengamos para evitar errores de imágenes demasiado grandes y el método para descryptar los datos obtenidos de la base de datos:

```

}
//Método para decodificar el BitMap de la imagen de perfil.
private Bitmap decodificarBitMap(String filePath, int reqWidth, int reqHeight) { 1 usage
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(filePath, options);

    options.inSampleSize = reescalado(options, reqWidth, reqHeight);

    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeFile(filePath, options);
}

//Método para reescalar la imagen y evitar cuelgues.
@ private int reescalado(BitmapFactory.Options options, int reqWidth, int reqHeight) { 1 usage
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {
        final int halfHeight = height / 2;
        final int halfWidth = width / 2;
        while ((halfHeight / inSampleSize) >= reqHeight && (halfWidth / inSampleSize) >= reqWidth) {
            inSampleSize *= 2;
        }
    }
    return inSampleSize;
}

//Método para descryptar los datos obtenidos de la bd.
@ private String descryptar(String texto) throws Exception { 2 usages
    String secret = "MyDifficultPassw";
    SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(StandardCharsets.UTF_8), algorithm: "AES");
    Cipher cipher = Cipher.getInstance(transformation: "AES/ECB/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decrypted = cipher.doFinal(Base64.decode(texto, Base64.DEFAULT));
    return new String(decrypted, StandardCharsets.UTF_8);
}

```

Aquí tenemos los métodos para comprobar si el usuario existe en la base de datos local y para insertar usuarios nuevos:

```

}
//Método para comprobar que existe el usuario en la BD local.
private boolean comprobarUsuarioExisteBD(SQLiteDatabase db, FirebaseUser usuario) { 1 usage
    if (usuario != null) {
        String sql = "SELECT idUsuario FROM t_usuarios WHERE idUsuario = ?";
        Cursor cursor = db.rawQuery(sql, new String[]{usuario.getId()});

        boolean existe = cursor.moveToFirst();

        cursor.close();

        return existe;
    }
    return false;
}

//Método para crear un usuario en la BD local.
@ private void crearInformacionUsuarioBD(SQLiteDatabase db, FirebaseUser usuario, String direccion, String telefono, String imagenObtenida) { 1 usage
    database.insertarUsuario(db, usuario.getId(), usuario.getDisplayName(), usuario.getEmail(), loginRegistro: null, logoutRegistro: null, direccion, telefono, imagenObt
}

```


También encontramos el método para cargar la información inicial del usuario al crear su cuenta ya sea con email, Google o Facebook:

```
75 }
76 //Método para cargar la información del usuario en la barra lateral al iniciar sesión.
77 private void cargarInformacionUsuario(FirebaseUser usuario, AccessToken accessToken) { 1 usage
78 //Si el usuario no es nulo obtendrá los datos del usuario de Firebase.
79 if (usuario != null) {
80     usuario.reload().addOnCompleteListener(task -> {
81         String nombreCuenta = usuario.getDisplayName();
82         String emailCuenta = usuario.getEmail();
83         Uri imagenCuenta = usuario.getPhotoUrl();
84         if (accessToken != null) {
85             textViewNombre.setText(nombreCuenta);
86             textViewEmail.setText("Conectado con Facebook");
87             GraphRequest request = GraphRequest.newGraphPathRequest(accessToken,
88                 graphPath: "/me/picture",
89                 new GraphRequest.Callback() {
90                     @Override
91                     public void onCompleted(@NonNull GraphResponse graphResponse) {
92                         try {
93                             JSONObject data = graphResponse.getJSONObject();
94                             if (data != null) {
95                                 JSONObject pictureData = data.getJSONObject( name: "data");
96                                 String imageUrl = pictureData.getString( name: "url");
97                                 guardarImagenUsuario(usuario.getId(), imageUrl);
98                                 cargarImagen(imageUrl, imageViewImagen);
99                             }
100                         } catch (JSONException e) {
101                             e.printStackTrace();
102                         }
103                     }
104                 });
105             Bundle parameters = new Bundle();
106             parameters.putBoolean("redirect", false);
107             parameters.putInt("height", 300);
108             parameters.putInt("width", 300);
109             request.setParameters(parameters);
110             request.executeAsync();
111         } else {
112             //Establecemos los datos en los textView.
113             textViewNombre.setText(nombreCuenta);
114         }
115     });
116 }
```

También tenemos el método para guardar la imagen de perfil de Google para guardarla en la base de datos local que luego al sincronizarse se subirá a la base de datos en la nube:

```
private void guardarImagenUsuario(String userId, String imageUrl) { 2 usages
// Guardar en SQLite
IMDbDatabaseHelper dbHelper = new IMDbDatabaseHelper( context: this);
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("foto", imageUrl);
db.update( table: "t_usuarios", values, whereClause: "idUsuario = ?", new String[]{userId});
db.close();

// Guardar en Firestore
FirebaseFirestore firestore = FirebaseFirestore.getInstance();
Map<String, Object> data = new HashMap<>();
data.put( k: "foto", imageUrl);
firestore.collection( collectionPath: "usuarios").document(userId) DocumentReference
    .set(data, SetOptions.merge()) Task<Void>
    .addOnSuccessListener(aVoid -> Log.d( tag: "Firestore", msg: "Foto de perfil actualizada en la nube"))
    .addOnFailureListener(e -> Log.e( tag: "Firestore", msg: "Error actualizando foto de perfil", e));
}
```

Aquí tenemos la configuración del launcher donde configuramos los datos que queremos obtener del intent.

```
//Método que configura el editUserLauncher para establecer los datos que debemos pasar y obtener del intent.
private void configurarEditUserLauncher() { 1 usage
    editUserLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if (result.getResultCode() == RESULT_OK && result.getData() != null) {
                    Intent data = result.getData();
                    String nombreActualizado = data.getStringExtra( name: "nombreActualizado");
                    String fotoActualizada = data.getStringExtra( name: "imagenActualizada");
                    String direccionActualizada = data.getStringExtra( name: "direccionActualizada");
                    String telefonoActualizado = data.getStringExtra( name: "telefonoActualizado");
                    if (nombreActualizado != null) {
                        textViewNombre.setText(nombreActualizado);
                    }

                    if (fotoActualizada != null) {
                        Uri uri = Uri.parse(fotoActualizada);
                        imageViewImagen.setImageURI(uri);
                    }
                }
            }
        }
    );
}
```

Aquí tenemos el método que se ejecuta para las acciones de las distintas opciones del menú:

```
//Método del menú.
@Override 6 usages
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_editUser) {
        irAEditarUsuario();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Aquí tenemos el método para ir a EditUserActivity donde establecemos los parámetros y le pasamos la información que queremos ver allí desde un intent:

```
//Método para ir a editar usuario desde el menú.
private void irAEditarUsuario() { 1 usage
    Intent i = new Intent( packageContext: MainActivity.this, EditUserActivity.class);
    String nombreCuenta = "";
    String idUsuario = mAuth.getCurrentUser().getUid();
    if (textViewNombre != null) {
        nombreCuenta = String.valueOf(textViewNombre.getText());
    }
    String emailCuenta = String.valueOf(textViewEmail.getText());
    i.putExtra( name: "idUsuario", idUsuario);
    i.putExtra( name: "nombre", nombreCuenta);
    i.putExtra( name: "email", emailCuenta);

    SQLiteDatabase db = database.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "SELECT telefono, direccion, foto FROM t_usuarios WHERE idUsuario = ?", new String[]{idUsuario});
    if (cursor.moveToFirst()) {
        String telefonoEncriptado = "";
        int colTelefono = cursor.getColumnIndex( s: "telefono");
        if (colTelefono != -1) {
            telefonoEncriptado = cursor.getString(colTelefono);
        }
        String direccionEncriptada = "";
        int colDireccion = cursor.getColumnIndex( s: "direccion");
        if (colDireccion != -1) {
            direccionEncriptada = cursor.getString(colDireccion);
        }
        String foto = "";
        int colFoto = cursor.getColumnIndex( s: "foto");
        if (colFoto != -1) {
            foto = cursor.getString(colFoto);
        }

        String telefonoDesencriptado = telefonoEncriptado;
        String direccionDesencriptada = direccionEncriptada;
        try {
            telefonoDesencriptado = desencriptar(telefonoEncriptado);
        } catch (Exception e) {
```

- Métodos en EditUserActivity:

Aquí obtenemos todos los datos desde el intent para establecerlos en los campos de texto o en el imageView:

```

EditText editTextDireccion = findViewById(R.id.editTextDir);
EditText editTextURL = findViewById(R.id.editTextURL);
//Inicializamos la variable de CountryCodePicker también al elemento gráfico del layout.
ccp = findViewById(R.id.countryCodePicker);

imagen = findViewById(R.id.imageView3);
//Configuramos los ActivityResultLaunchers que se encargan de obtener los permisos.
configurarActivityResultLaunchers();

//Obtenemos los datos del intent.
if (getIntent().hasExtra( name: "nombre")) {
    editTextNombre.setText(getIntent().getStringExtra( name: "nombre"));
}
if (getIntent().hasExtra( name: "email")) {
    editTextEmail.setText(getIntent().getStringExtra( name: "email"));
}
if (getIntent().hasExtra( name: "telefono")) {
    editTextTelefono.setText(getIntent().getStringExtra( name: "telefono"));
}
if (getIntent().hasExtra( name: "direccion")) {
    editTextDireccion.setText(getIntent().getStringExtra( name: "direccion"));
}

if (getIntent().hasExtra( name: "imagenUri")) {
    String fotoPath = getIntent().getStringExtra( name: "imagenUri");
    if (fotoPath != null && !fotoPath.isEmpty()) {
        Uri uri = Uri.fromFile(new File(fotoPath));
        Bitmap bitmap = decodificarBitMap(fotoPath, reqWidth: 300, reqHeight: 300);
        imagen.setImageBitmap(bitmap);
    }
}
}

```

Launchers para obtener los permisos de localización y establecer la localización desde el intent de EditAddressActivity:

```

//Launcher para obtener los permisos de localización.
obtenerPermisosLocalizacionLauncher = registerForActivityResult(
    new ActivityResultContracts.RequestMultiplePermissions(),
    result -> {
        boolean permisosConcedidos = result.getOrDefault(Manifest.permission.ACCESS_FINE_LOCATION, defaultValue: false) ||
            result.getOrDefault(Manifest.permission.ACCESS_COARSE_LOCATION, defaultValue: false);

        if (permisosConcedidos) {
            abrirEditarDireccion();
        } else {
            Toast.makeText( context: this, text: "Permiso de ubicación denegado", Toast.LENGTH_SHORT).show();
        }
    });

//Launcher para obtener a direccion de la actividad EditAddress.
direccionLauncher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    result -> {
        if (result.getResultCode() == RESULT_OK && result.getData() != null) {
            String direccion = result.getData().getStringExtra( name: "direccionSeleccionada");
            editTextDireccion.setText(direccion);
        }
    });
//Declaramos los botones

```

Declaración de botones y métodos para obtener las imágenes establecidos desde URL:

```
//Declaramos los botones.
Button buttonImagen = findViewById(R.id.buttonImagen);
Button buttonGuardar = findViewById(R.id.buttonGuardar);
Button buttonDir = findViewById(R.id.buttonDireccion);
Button buttonImagenURL = findViewById(R.id.buttonImagen2);
//Al pulsar el botón de imagen URL obtiene la imagen de la url introducida en el campo de texto.
buttonImagenURL.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String urlString = editTextURL.getText().toString().trim();
        if (!urlString.toLowerCase().endsWith(".jpg") &&
            !urlString.toLowerCase().endsWith(".jpeg") &&
            !urlString.toLowerCase().endsWith(".png")) {
            Toast.makeText(context, EditUserActivity.this,
                text: "La URL debe terminar en .jpg, .jpeg o .png",
                Toast.LENGTH_SHORT).show();
            return;
        }
        //Descargamos la imagen con un hilo ya que es una tarea Asincrona.
        new Thread() -> {
            try {
                URL url = new URL(urlString);
                HttpURLConnection connection = (HttpURLConnection) url.openConnection();
                connection.setDoInput(true);
                connection.connect();
                InputStream input = connection.getInputStream();
                Bitmap bitmap = BitmapFactory.decodeStream(input);
                //Escala la imagen.
                Bitmap scaledBitmap = Bitmap.createScaledBitmap(bitmap, dstWidth: 300, dstHeight: 300, filter: true);
                String localPath = guardarBitMapLocalmente(scaledBitmap);
                imageUri = Uri.fromFile(new File(localPath));
                runOnUiThread() -> {
                    imagen.setImageBitmap(scaledBitmap);
                    Toast.makeText(context, EditUserActivity.this,
                        text: "Imagen cargada correctamente",
                        Toast.LENGTH_SHORT).show();
                };
            } catch (Exception e) {
```

Botones para poner ir a EditAddressActivity, para establecer la imagen desde la galería o cámara y para guardar los datos y pasarlos como intent para cambiarlos dinámicamente en la barra lateral de usuario:


```

buttonDir.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (tienePermisosUbicacion()) {
            abrirEditarDireccion();
        } else {
            solicitarPermisosUbicacion();
        }
    }
});

//Botón que abre el método para seleccionar imagen.
buttonImagen.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { seleccionarImagen(); }
});

//Botón para guardar los datos.
buttonGuardar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Obtenemos los datos de los campos y revisamos diferentes excepciones.
        String telefono = String.valueOf(editTextTelefono.getText());
        String prefijo = ccp.getSelectedCountryCodeWithPlus();
        String telefonoCompleto = prefijo + telefono;
        nuevoNombre = editTextNombre.getText().toString().trim();
        if (nuevoNombre.isEmpty()) {
            Toast.makeText(context, EditUserActivity.this, text: "El nombre no puede estar vacío.", Toast.LENGTH_SHORT
            return;
        }

        if (!telefonoValido(telefonoCompleto)) {
            Toast.makeText(context, EditUserActivity.this, text: "Número de teléfono inválido.", Toast.LENGTH_SHORT).s
            return;
        }
        //Establecemos los datos en el intent.
        Intent resultIntent = new Intent();
        resultIntent.putExtra(name: "nombreActualizado", nuevoNombre);
        if (imageUri != null) {
            resultIntent.putExtra(name: "imagenActualizada", imageUri.toString());
        }
        setResult(RESULT_OK, resultIntent);
    }
});

```

Métodos para comprobar los permisos de ubicación y manejar el flujo de ejecución correctamente:

```

222     });
223
224     }
225     //Método que comprueba si tenemos los permisos de ubicación concedidos.
226     private boolean tienePermisosUbicacion() { 1 usage
227         return ContextCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED ||
228             ContextCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_COARSE_LOCATION) == PackageManager.PERMISSION_GRANTED;
229     }
230     //Método que solicita los permisos de ubicación.
231     private void solicitarPermisosUbicacion() { 1 usage
232         //Dependiendo de la versión solicitamos unos permisos u otros.
233         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
234             obtenerPermisosLocalizacionLauncher.launch(new String[]{
235                 Manifest.permission.ACCESS_FINE_LOCATION,
236                 Manifest.permission.ACCESS_COARSE_LOCATION
237             });
238         } else {
239             obtenerPermisosLocalizacionLauncher.launch(new String[]{Manifest.permission.ACCESS_FINE_LOCATION});
240         }
241     }

```

Métodos para abrir la ActivityEditarDirección, comprobar el teléfono válido y los métodos para guardar en directorio local los archivos de imagen tanto de la cámara como los de la galería, esto lo hacemos configurando un files_path también en AndroidManifest:

```

21     }
22     //Método que nos envía a editar dirección con el launcher para pasar los datos a través del intent.
23     private void abrirEditarDireccion() { 2 usages
24         Intent intent = new Intent( packageContext, EditUserActivity.class);
25         direccionLauncher.launch(intent);
26     }
27     //Método que comprueba que el teléfono introducido es válido.
28     private boolean telefonoValido(String telefonoCompleto) { 1 usage
29         if (telefonoCompleto == null || telefonoCompleto.trim().isEmpty()) {
30             Log.d( tag: "EditUserActivity", msg: "Teléfono vacío o nulo.");
31             return false;
32         }
33         //Con una expresión regular.
34         boolean valido = telefonoCompleto.matches( regex: "\\+[1-9]\\d{1,14}$" ) && telefonoCompleto.length() > ccp.getSelectedCountryCodeWithPlus().length();
35         Log.d( tag: "EditUserActivity", msg: "Teléfono: " + telefonoCompleto + " - Válido: " + valido);
36         return valido;
37     }
38     //Método para guardar como Bitmap de manera local la imagen escogida por el usuario para establecer como foto de perfil.
39     @ private String guardarBitmapLocalmente(Bitmap bitmap) throws IOException { 1 usage
40         String fileName = "user_image_" + System.currentTimeMillis() + ".jpg";
41         File localFile = new File(getFilesDir(), fileName);
42         FileOutputStream out = new FileOutputStream(localFile);
43         bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 90, out);
44         out.flush();
45         out.close();
46         Log.d( tag: "EditUserActivity", msg: "Imagen guardada localmente en: " + localFile.getAbsolutePath());
47         return localFile.getAbsolutePath();
48     }
49     //Si se trata de una uri también guardamos la imagen en el almacenamiento local.
50     @ private String guardarImagenDeUriLocalmente(Uri uri) throws IOException { 1 usage
51         InputStream inputStream = getContentResolver().openInputStream(uri);
52         if (inputStream == null) {
53             Log.e( tag: "EditUserActivity", msg: "No se pudo abrir el InputStream de la URI: " + uri);
54             return null;
55         }
56
57         String fileName = "user_image_" + System.currentTimeMillis() + ".jpg";
58         File localFile = new File(getFilesDir(), fileName);
59     }

```

Métodos para decodificar las imágenes y reescalarlas:

```

291     }
292
293     //Método para decodificar el Bitmap.
294     private Bitmap decodificarBitmap(String filePath, int reqWidth, int reqHeight) { 1 usage
295         // Primero, decodifica solo las dimensiones
296         BitmapFactory.Options options = new BitmapFactory.Options();
297         options.inJustDecodeBounds = true;
298         BitmapFactory.decodeFile(filePath, options);
299
300         // Calcula el factor de escalado
301         options.inSampleSize = reescalado(options, reqWidth, reqHeight);
302
303         // Decodifica el bitmap ya escalado
304         options.inJustDecodeBounds = false;
305         return BitmapFactory.decodeFile(filePath, options);
306     }
307     //Método para reescalar las imágenes.
308     @ private int reescalado(BitmapFactory.Options options, int reqWidth, int reqHeight) { 1 usage
309         final int height = options.outHeight;
310         final int width = options.outWidth;
311         int inSampleSize = 1;
312
313         if (height > reqHeight || width > reqWidth) {
314             final int halfHeight = height / 2;
315             final int halfWidth = width / 2;
316             while ((halfHeight / inSampleSize) >= reqHeight && (halfWidth / inSampleSize) >= reqWidth) {
317                 inSampleSize *= 2;
318             }
319         }
320         return inSampleSize;
321     }

```

Métodos modificar el usuario en la bd:

```
//Método que modifica al usuario en la base de datos local.
private void modificarUsuario(String telefonoCompleto) { 1 usage
    IMDbDatabaseHelper database = new IMDbDatabaseHelper( context: this);
    SQLiteDatabase db = database.getWritableDatabase();
    String idUsuario = getIntent().getStringExtra( name: "idUsuario");
    EditText editTextDireccion = findViewById(R.id.editTextDir);
    String direccion = String.valueOf(editTextDireccion.getText());
    ContentValues valores = new ContentValues();
    valores.put("nombre", nuevoNombre);
    try {
        //Ciframos teléfono y dirección.
        String telefonoEncriptado = encriptar(telefonoCompleto);
        String direccionEncriptada = encriptar(direccion);
        valores.put("telefono", telefonoEncriptado);
        valores.put("direccion", direccionEncriptada);
    } catch (Exception e) {
        Log.e( tag: "EditUserActivity", msg: "Error cifrando datos: " + e.getMessage());
        // Si ocurre un error, se pueden almacenar sin cifrar o abortar la operación
        valores.put("telefono", telefonoCompleto);
        valores.put("direccion", direccion);
    }

    if (imageUri != null) {
        try {
            String localPath = guardarImagenDeUriLocalmente(imageUri);
            if (localPath != null) {
                valores.put("foto", localPath);
            } else {
                Log.e( tag: "EditUserActivity", msg: "Error al guardar la imagen localmente.");
            }
        } catch (IOException e) {
            Log.e( tag: "EditUserActivity", msg: "Error al guardar la imagen: " + e.getMessage(),
        }
    }
}
```

Método para guardar el usuario en la nube actualizado y encriptado:


```

}
//Método para actualizar los usuarios en la nube.
@ private void actualizarUsuariosNube(String idUsuario, ContentValues valores) { 1 usage
    //Creamos un nuevo objeto HashMap y establecemos los valores que queremos.
    Map<String, Object> data = new HashMap<>();
    if (valores.containsKey("nombre")) {
        data.put("nombre", valores.getAsString( key: "nombre"));
    }
    if (valores.containsKey("telefono")) {
        data.put("telefono", valores.getAsString( key: "telefono"));
    }
    if (valores.containsKey("direccion")) {
        data.put("direccion", valores.getAsString( key: "direccion"));
    }
    if (valores.containsKey("foto")) {
        data.put("foto", valores.getAsString( key: "foto"));
    }
    //Guardamos la colección en Firestore.
    FirebaseFirestore firestore = FirebaseFirestore.getInstance();
    firestore.collection( collectionPath: "usuarios").document(idUsuario) DocumentReference
        .update(data) Task<Void>
        .addOnSuccessListener(aVoid ->
            Log.d( tag: "Firestore", msg: "Usuario " + idUsuario + " actualizado en la nube"));
        .addOnFailureListener(e ->
            Log.e( tag: "Firestore", msg: "Error actualizando usuario " + idUsuario, e));
    }
//Método para encriptar en base 64.
@ private String encriptar(String texto) throws Exception { 2 usages
    String secret = "MyDifficultPassw";
    SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(StandardCharsets.UTF_8), algorithm: "AES");
    Cipher cipher = Cipher.getInstance( transformation: "AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encrypted = cipher.doFinal(texto.getBytes(StandardCharsets.UTF_8));
    return Base64.encodeToString(encrypted, Base64.DEFAULT);
}
//Método para configurar los launchers de galería y cámara, que manejan la obtención de permisos y el

```

Métodos para la obtención de los permisos de imagen y el flujo de selección de imagen:

```

});

//Launcher de galería que obtiene la foto del resultado de la actividad.
galeriaLauncher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            if (result.getResultCode() == RESULT_OK && result.getData() != null) {
                imageUri = result.getData().getData();
                imagen.setImageURI(imageUri);
            }
        }
    });

//Launcher para obtener los permisos que comprueba la versión para solicitar unos u otros.
obtenerPermisosLauncher = registerForActivityResult(
    new ActivityResultContracts.RequestMultiplePermissions(),
    result -> {
        Boolean permisosCamara = result.getOrDefault(Manifest.permission.CAMERA, defaultValue: false);
        Boolean permisosAlmacenamiento;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            permisosAlmacenamiento = result.getOrDefault(Manifest.permission.READ_MEDIA_IMAGES, false);
        } else {
            permisosAlmacenamiento = result.getOrDefault(Manifest.permission.READ_EXTERNAL_STORAGE, false);
        }
        if (permisosCamara && permisosAlmacenamiento) {
            mostrarOpcionesImagen();
        } else {
            Toast.makeText( context: this, text: "Permisos denegados. No se puede seleccionar la imagen",
                Toast.LENGTH_LONG);
        }
    });

//Método que maneja el flujo de selección de imagen o solicitar permisos.
private void seleccionarImagen() { 1 usage
    if (comprobarPermisos()) {
        mostrarOpcionesImagen();
    } else {
        solicitarPermisos();
    }
}

```

- Métodos en EditAddress que utilizando las herramientas Places y Autocomplete podemos establecer la ubicación y un mapa de la misma:

```

24 public class EditAddressActivity extends AppCompatActivity {
27     private ImageView imageViewMapa; 2 usages
28     private PlacesClient placesClient; 1 usage
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_edit_address);
34         //Iniciamos variables de elementos gráficos y botones.
35         editTextDireccion = findViewById(R.id.editTextSeleccionarDir);
36         imageViewMapa = findViewById(R.id.imageViewMapa);
37         Button buttonBuscar = findViewById(R.id.button2);
38         Button buttonConfirmar = findViewById(R.id.button3);
39         //Comprobamos si la herramienta Places está inicializada, si no la inicializamos con la Key
40         //que nos han facilitado.
41         if (!Places.isInitialized()) {
42             Places.initialize(getApplicationContext(), apiKey, "AIzaSyAER7D-uvYpB063wZjz9z3Ae6uLgAci-0U");
43         }
44         //Creamos un nuevo cliente de la herramienta Places.
45         placesClient = Places.createClient(context, this);
46         //Al hacer clic en el botón de buscar abrimos el Buscador de Lugares que es la barra de buscar localizaciones.
47         buttonBuscar.setOnClickListener(v -> abrirBuscadorDeLugares());
48         //Al confirmar obtenemos la dirección del editText y la devolvemos a EditUser a través del intent.
49         buttonConfirmar.setOnClickListener(v -> {
50             String direccionSeleccionada = editTextDireccion.getText().toString();
51             Intent resultIntent = new Intent();
52             resultIntent.putExtra(name, "direccionSeleccionada", direccionSeleccionada);
53             setResult(RESULT_OK, resultIntent);
54             finish();
55         });
56     }
57     //Abrimos el buscador de lugares con una lista de campos de la herramienta Places que nos obtiene los campos de las localizaciones, es decir su id, su nombre etc.
58     //Y utilizando la herramienta de la librería Autocomplete obtenemos las localizaciones exactas.
59     private void abrirBuscadorDeLugares() { 1 usage
60         List<Place.Field> fields = Arrays.asList(Place.Field.ID, Place.Field.NAME, Place.Field.LAT_LNG);
61         Intent intent = new Autocomplete.IntentBuilder(AutocompleteActivityMode.OVERLAY, fields)
62             .build(context, this);
63         //Se lo pasamos como un intent para establecerlo en el campo de dirección.
64         startActivityForResult(intent, requestCode, 1);
65     }

```

```

        startActivityForResult(intent, requestCode, 1);
    }
    //Obtenemos el dato del intent y con la latitud y longitud obtenidas mostramos el mapa estático de la dirección seleccionada.
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == 1 && resultCode == RESULT_OK) {
            Place place = Autocomplete.getPlaceFromIntent(data);
            editTextDireccion.setText(place.getName());
            mostrarMapaEstatico(place.getLatLng().latitude, place.getLatLng().longitude);
        }
    }
    //Método para mostrar el mapa estático que simplemente obtiene del enlace la imagen según las coordenadas.
    private void mostrarMapaEstatico(double lat, double lng) { 1 usage
        String url = "https://maps.googleapis.com/maps/api/staticmap?center=" + lat + "," + lng + "&zoom=15&size=600x600&markers=color:red%7C" + lat + "," + lng + "&key=AIzaSyAER7D-uvYpB063wZjz9z3Ae6uLgAci-0U";
        //Método para descargar la imagen en un hilo ya que es una tarea asincrónica y establecemos un hilo.
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    InputStream in = new URL(url).openStream();
                    final Bitmap bitmap = BitmapFactory.decodeStream(in);
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() { imageViewMapa.setImageBitmap(bitmap); }
                    });
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
    }

```

- Clases FavoritesSync y UsersSync

Cuentan principalmente con los métodos de sincronización de la bd local con la nube y viceversa:

```

25         this.firestore = firestore;
26         this.database = database;
27     }
28     //Método para subir favoritos a la nube, obtenemos los datos haciendo una consulta
29     //favoritos de la base de datos local donde sacamos los datos de las películas fav
30     public void subirFavoritosANube() { 1 usage
31         mAuth = FirebaseAuth.getInstance();
32         FirebaseUser currentUser = mAuth.getCurrentUser();
33         if (currentUser == null) {
34             Log.e(tag: "FavoritesSync", msg: "El usuario no está autenticado.");
35             return;
36         }
37
38         String idUsuario = currentUser.getId();
39         SQLiteDatabase db = database.getReadableDatabase();
40
41         Cursor cursor = db.rawQuery(
42             sql: "SELECT * FROM " + IMDBDatabaseHelper.TABLE_FAVORITOS + " WHERE id
43             new String[]{idUsuario}
44         );
45
46         if (cursor != null && cursor.moveToFirst()) {
47             Log.d(tag: "FavoritesSync", msg: "Favoritos encontrados para el usuario: "
48
49             do {
50                 // Rescatar índices de columnas
51                 int colIdPelicula = cursor.getColumnIndex( columnName: "idPelicula");
52                 int colTitulo = cursor.getColumnIndex( columnName: "nombrePelicula");
53                 int colDescripcion = cursor.getColumnIndex( columnName: "descripcionPel
54                 int colFecha = cursor.getColumnIndex( columnName: "fechaLanzamiento");
55                 int colRanking = cursor.getColumnIndex( columnName: "rankingPelicula");

```

```

// Creamos el objeto HashMap que subiremos a Firestore con los datos recogidos.
Map<String, Object> usuario = new HashMap<>();
usuario.put("idUsuario", idUsuario);
usuario.put("nombre", nombre);
usuario.put("email", email);
//Guardamos los campos de logs como campo de union array para unir los campos anteri
usuario.put("loginRegistro", FieldValue.arrayUnion(LoginRegistro));
usuario.put("logoutRegistro", FieldValue.arrayUnion(LogoutRegistro));
usuario.put("direccion", direccion);
usuario.put("telefono", telefono);
usuario.put("foto", foto);
//Variable para el siguiente con logs.
final String finalIdUsuario = idUsuario;
//Creamos la colección.
firestore.collection( (collectionPath: "usuarios") CollectionReference
    .document(idUsuario) DocumentReference
    //Lo guardamos como merge para que los datos de los logs se mantengan y simp
    .set(usuario, SetOptions.merge()) Task<Void>
    .addOnSuccessListener(aVoid ->
        Log.d(tag: "Firestore", msg: "Usuario " + finalIdUsuario + " subido a
    .addOnFailureListener(e ->
        Log.e(tag: "Firestore", msg: "Error al subir el usuario " + finalIdUs
    )
}
cursor.close();
}
//Método para descargar los usuarios de la nube al local.
public void descargarUsuariosNubeALocal(final CloudSyncCallback callback) { 2 usages
    //Obtenemos la colección por su nombre.
    firestore.collection( (collectionPath: "usuarios") CollectionReference
        .get() Task<QuerySnapshot>
        .addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
            @Override
            public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                //Abrimos la base de datos.
                SQLiteDatabase db = database.getWritableDatabase();
                //Obtenemos los datos de una snapshot del documento de donde vamos a ir
                List<DocumentSnapshot> documentos = queryDocumentSnapshots.getDocuments()
                for (DocumentSnapshot document : documentos) {
                    String idUsuario = document.getId();

```

10. Pruebas

- **10.1 Estrategia de pruebas**

Las principales pruebas que he considerado son las siguientes:

- Comprobar el flujo de cambio de datos de principio a fin.
- Pruebas de rendimiento, comprobamos que el programa es eficiente.
- Pruebas de estrés pulsamos varias veces sobre el botón dirección por ejemplo.
- Comprobar los campos de texto con entradas no válidas.

11. Implementación y despliegue

- **10.1 Herramientas de desarrollo**

- IDE: Android Studio.
- Lenguaje de programación: Java.
- SDK de Android: API nivel 34 (Android 14/Upside Down Cake).

- **10.2 Proceso de despliegue**

Lo ideal sería lanzar la aplicación en Google Play Store, para ello deberíamos de configurar el archivo build.gradle para firmar la aplicación, exportar el apk para instalarlo en dispositivos android y publicar la aplicación en google play.

11. Mantenimiento y actualizaciones

- **Mejoras futuras**

Yo principalmente incluiría mejoras sobretodo visuales o auditivas como animaciones o detalles de sonido o elementos para valorar películas o incluso la posibilidad de subir reseñas como tal, también miraría la función de añadir amigos que puedan compartir y ver sus listas de favoritos desde la misma app.

12. Bibliografía

- [1] *Ciclo de vida de la actividad.* (s. f.). Android Developers.
<https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es-419>
- [2] *Criptografía.* (s. f.). Android Developers.
<https://developer.android.com/privacy-and-security/cryptography?hl=es-419>
- [3] *Documentación de Firestore | Google Cloud.* (s. f.). Google Cloud.
<https://cloud.google.com/firestore/docs?hl=es-419>
- [4] Facebook. (s. f.). *Facebook Login.* Facebook Developers.
<https://developers.facebook.com/docs/facebook-login/android/>
- [5] *Firestore | Firebase.* (s. f.). Firebase. <https://firebase.google.com/docs/firestore?hl=es-419>
- [6] *Place Autocomplete.* (s. f.). Google For Developers.
<https://developers.google.com/maps/documentation/places/web-service/autocomplete?hl=es-419>