Filmosis

Proyecto Intermodular DAM, Maria Ana Sanz

Curso 2023-2024

Integrantes

Jiménez Aldasoro, Kaiet
Arrondo Villaplana, Aritz
García Galera, Adrián

Índice

1 Introducción

1.1 Breve descripción

Filmosis es una aplicación dedicada a los amantes del cine que buscan una experiencia completa para descubrir, explorar y valorar películas. Esta plataforma combina la riqueza de información de IMDb con una función de valoración integrada, permitiendo a los usuarios sumergirse en el vasto mundo del cine de una manera interactiva y social.

1.2 Objetivos

- Exploración de Películas: Filmosis ofrece una amplia base de datos de películas donde los usuarios pueden explorar información detallada sobre películas, desde el elenco hasta las críticas.
- Listas Personalizadas: Los usuarios pueden crear listas personalizadas de películas, como "Mis Favoritas", "Por Ver", o cualquier categoría que deseen. Esta función ayuda a organizar y compartir preferencias cinematográficas.
- Noticias y Actualizaciones: Filmosis mantiene a los usuarios actualizados sobre las últimas noticias y eventos en la industria cinematográfica, asegurándose de que estén informados sobre estrenos, premios y más.
- Perfil de Usuario: Cada usuario tiene su propio perfil personalizado, donde pueden ver sus valoraciones, listas y contribuciones. También pueden conectarse con amigos y descubrir lo que están viendo y valorando.

Filmosis se esfuerza por crear una comunidad vibrante de cinéfilos, proporcionando una plataforma interactiva y social para explorar y disfrutar del mundo del cine.

2 Tecnologías empleadas

2.1 Kotlin

- Retrofit: Retrofit es una biblioteca popular para Android que simplifica el consumo de servicios web RESTful. Ofrece una forma declarativa y eficiente de definir y realizar solicitudes HTTP, así como de procesar las respuestas.
- Gson: Biblioteca de Java / Kotlin desarrollada por Google que facilita el análisis y la serialización de objetos Java en formato JSON y viceversa. Es ampliamente utilizada en aplicaciones Android para el intercambio de datos entre el cliente y el servidor, así como para el almacenamiento de datos en archivos JSON.
- CarouselRecyclerView: Libearía de Android que permite crear y mostrar carruseles de elementos dentro de un RecyclerView. Esta biblioteca simplifica la implementación de carousels horizontales o verticales en aplicaciones Android, lo que proporciona una experiencia de usuario más atractiva y dinámica.
- CircleImageView: CircleImageView es una biblioteca de Android que proporciona una vista personalizada para mostrar imágenes redondeadas en aplicaciones Android. Esta biblioteca simplifica la implementación de imágenes circulares en las interfaces de usuario de las aplicaciones, lo que mejora la estética y la coherencia visual.
- Dokka: Herramienta de generación de documentación para proyectos Kotlin y Java. Esta herramienta, desarrollada por JetBrains, permite generar documentación de manera automática a partir del código fuente, facilitando la creación y mantenimiento de documentación técnica para proyectos de software.
- Glide: Glide es una biblioteca de gestión de imágenes para Android que facilita la carga, la visualización y el almacenamiento en caché de imágenes de forma eficiente en aplicaciones Android. Desarrollado por Bumptech, Glide se ha convertido en una de las bibliotecas más populares para el manejo de imágenes en el ecosistema Android.

2.2 Bibliotecas adicionales

- Firebase: Firebase es una plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google. Proporciona una variedad de servicios en la nube que ayudan a los desarrolladores a crear, mejorar y hacer crecer sus aplicaciones de manera rápida y efectiva. Estos servicios incluyen desde herramientas para el desarrollo de aplicaciones hasta análisis de usuarios y monetización. Las herramientas de Firebase que hemos utilizado son las siguientes:

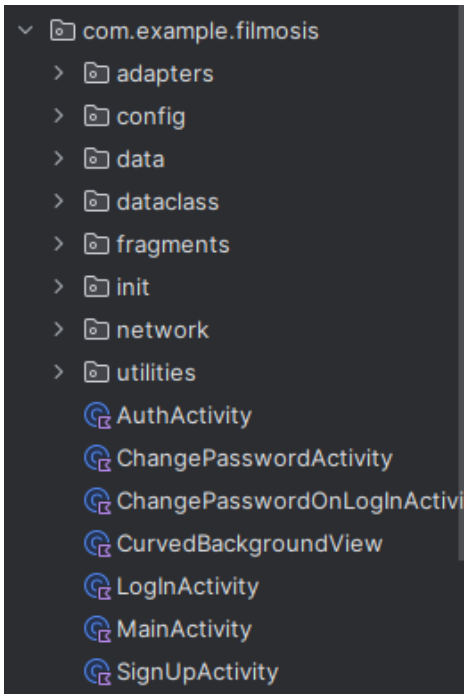
  - Firebase Analytics: Servicio de análisis de usuarios que permite a los desarrolladores comprender el comportamiento de los usuarios en sus aplicaciones, lo que les ayuda a tomar decisiones informadas para mejorar la experiencia del usuario y el rendimiento de la aplicación.
  - Firebase Authentication: Servicio de autenticación de usuarios que permite a los desarrolladores autenticar usuarios de forma segura en sus aplicaciones mediante diversos métodos de autenticación, lo que garantiza la seguridad y la privacidad de los datos del usuario.
  - Firebase Firestore: Firestore es un servicio de base de datos en la nube de documentos en tiempo real y escalable que permite a los desarrolladores almacenar, sincronizar y consultar datos de manera flexible y eficiente.
  - Firebase Storage: Servicio de almacenamiento en la nube capaz de almacenar y servir archivos de usuario, como imágenes, videos, archivos de audio y otros archivos multimedia, de forma segura y eficiente.

- TMDB (The Movie Database): Plataforma en línea que proporciona una amplia base de datos de información relacionada con películas, programas de televisión y personas asociadas con la industria del entretenimiento. TMDb ofrece una variedad de datos útiles para cinéfilos, desarrolladores de aplicaciones y profesionales de la industria del entretenimiento.

3 Estructura del proyecto

3.1 Organización de directorios y archivos

La estructura de los diferentes paquetes en la aplicación es la siguiente:



- **Adapters:** En este paquete se encuentran principalmente los adaptadores para todos los RecyclerView, necesarios para mostrar la información y enlazar los datos en sus respectivos elementos.
- **Config**: Aquí se encuentra solo una clase, *DatosConexion.kt*, con campos estáticos que guardan constantes como las URLs base de la API de TMDB, para poder acceder desde fuera, o la misma API key, para poder usarla en todas las consultas de manera consistente.
- **Data:** En data se almacenan otros dos paquetes, *access* y *model*. El primero hace de repositorio de consultas para interactuar con una base de datos o con APIs, en nuestro caso, existen dos clases, *MoviesAccess.kt* y *PersonsAccess.kt*, donde se guardan las distintas funciones para las peticiones a TMDB. En cambio, en *model* se encuentran las clases para mapear las respuestas de las consultas a objetos / estructuras de Kotlin, lo que viene siendo un *Dataclass*. La idea entre los dos paquetes es similar a la arquitectura de DAO (*Data Access Object*) y DTO (*Data Transfer Object*).
- **Dataclass**: En este paquete, se almacenan dataclasses similares a las que hay en data.model, pero para uso interno de la aplicación y no para mapear directamente las respuestas de TMDB.
- **Fragments**: Aquí simplemente se guardan todos los fragmentos de la aplicación.

- **Init**: En init se encuentran clases para inicializar instancias de distintas fuentes. En este caso solo está *FirebaseInitializer.kt*, la cual inicializa todas las intancias necesarias de cada librería de Firebase, como la propia base de datos *Firestore* o el servicio de autenticación de usuarios *FirebaseAuth*. De esta manera, se puede acceder a éstas instancias desde cualquier punto del proyecto sin tener que crear una cada vez que se quiera acceder.

- **Network:** En este paquete se almacenan archivos relacionados con la conexión o internet. En el caso de Filmosis, se encuentra el servicio de Retrofit, que permite realizar peticiones HTTP, en nuestro caso a TMDB. También dentro existe el paquete de *interfaces*, que contiene interfaces para declarar los endpoints de las APIs.

- **Utilities:** En utilites se guardan distintos archivos, mas generales, de ayuda o simplemente utilidades para la aplicación. Por ejemplo, clases con métodos auxiliares para Firebase, o Mappings de recursos como imágenes con sus correspondientes IDs.

Por último, en la raíz del paquete principal, se encuentran las Activities del proyecto, para poder visualizarlas de mejor manera y acceder de manera rápida porque son los componentes mas importantes de la aplicación, junto con los fragmentos.

4 Configuración del entorno de desarrollo

4.1 Herramientas necesarias

Lo primero obviamente es instalar el IDE de Android Studio desde su página oficial. Este proceso se cubre de manera más extensa en el siguiente apartado *Instrucciones de Instalación*.

No hace falta ningun tipo de herramienta / configuración compleja para ejecutar el proyecto, basta con cumplir los siguientes requerimientos:

- SDK de Android: El Android Software Development Kit (SDK) contiene las bibliotecas y herramientas necesarias para desarrollar aplicaciones Android. La versión necesaria para ejecutar el proyecto es la 34, por lo que se recomienda descargarla y configurarla en las preferencias.

- Instalación de Emulador: Con la instalación de Android Studio, viene por defecto un dispositivo instalado, pero si se quiere personalizar o instalar un dispositivo mas potente, debe ser con la versión 9.0 de Android como mínimo. También es posible conectar un dispositivo real via USB o Wi-Fi.

4.2 Instrucciones de instalación

- **Instalación de Android Studio**

1. Descarga el instalador de Android Studio desde el sitio web oficial: `https://developer.android.com/studio`.
2. Ejecuta el archivo de instalación descargado.
3. Sigue las instrucciones del instalador para completar la instalación. Asegúrate de seleccionar las opciones de configuración que desees durante el proceso de instalación.
4. Una vez finalizada la instalación, abre Android Studio en tu computadora.
5. Durante la primera ejecución, Android Studio te guiará a través del proceso de configuración inicial, que incluye la instalación de componentes adicionales como el SDK de Android, el emulador y herramientas de desarrollo.
6. Sigue las instrucciones en pantalla para completar la configuración inicial. Puedes elegir las preferencias de usuario y las ubicaciones de instalación según tus necesidades.
7. Una vez finalizada la configuración inicial, Android Studio estará listo para ser utilizado para desarrollar aplicaciones Android.

- **Cambiar la versión del SDK a la API 34**

1. En la barra de menú, selecciona `Tools` (Herramientas) y luego `SDK Manager` (Administrador de SDK).
2. En la ventana del Administrador de SDK, selecciona la pestaña `SDK Platforms` (Plataformas SDK).
3. En la lista de versiones de Android, busca y marca la casilla correspondiente a la API 34 (Android 12). Puedes desmarcar otras versiones si solo deseas trabajar con la versión 34.
4. Haz clic en `Apply` (Aplicar) para confirmar los cambios y comenzar la descarga e instalación de la API 34.
5. Una vez completada la instalación, cierra la ventana del Administrador de SDK.
6. Asegúrate de que tu proyecto esté configurado para utilizar la API 34. Puedes hacerlo abriendo el archivo `build.gradle` en la carpeta `app` de tu proyecto y configurando `compileSdkVersion` y `targetSdkVersion` para que coincidan con la API 34.
7. Vuelve a sincronizar tu proyecto con los archivos de compilación haciendo clic en `Sync Now` (Sincronizar Ahora) si es necesario.

- **Instalación de emulador en Android Studio**

1. En la barra de menú, selecciona `Tools` (Herramientas) y luego `AVD Manager` (Administrador de AVD).

2. En la ventana del Administrador de AVD, haz clic en el botón `Create Virtual Device` (Crear Dispositivo Virtual).
3. Selecciona un dispositivo de la lista (por ejemplo, Pixel 4) y haz clic en `Next` (Siguiente).
4. Selecciona una imagen del sistema para el dispositivo virtual. Puedes descargar diferentes versiones de Android haciendo clic en `Download` (Descargar) junto a la versión que desees.
5. Una vez descargada la imagen del sistema, selecciónala y haz clic en `Next`.
6. Configura las opciones de hardware según tus preferencias y haz clic en `Finish` (Finalizar) para crear el dispositivo virtual.
7. El dispositivo virtual se agregará a la lista en el Administrador de AVD. Para iniciarlo, haz clic en el botón de reproducción al lado del nombre del dispositivo.
8. Espera a que el emulador se inicie y se cargue completamente. Una vez listo, podrás usarlo para probar y depurar tus aplicaciones.

5 Guía de Uso

5.1 Instrucciones para Configurar y Ejecutar el Proyecto

5.1.1 Mediante el Código Fuente (src)

1. **Configuración inicial:**
   - Abre Android Studio.
   - Selecciona "Abrir un proyecto existente" y navega hasta el directorio donde se encuentra el código fuente del proyecto.
   - Selecciona el archivo `build.gradle` en el directorio raíz del proyecto y haz clic en "Abrir".

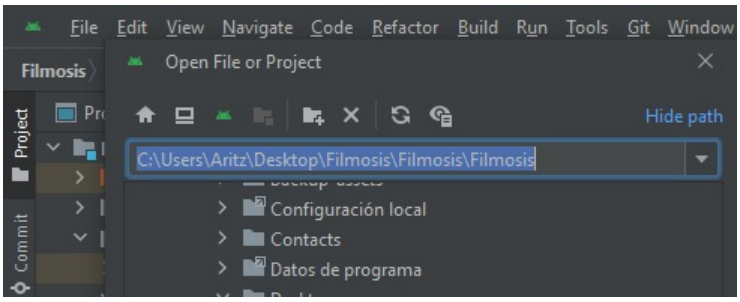


Figure 1: Interfaz de Android Studio mostrando el proceso de apertura del proyecto existente.

2. **Espera a que Android Studio sincronice el proyecto:**
   - Android Studio puede necesitar tiempo para descargar e instalar las dependencias del proyecto, así que espera a que la sincronización se complete.
3. **Configuración del dispositivo virtual o dispositivo físico:**
   - Puedes ejecutar tu aplicación en un dispositivo virtual o en un dispositivo físico conectado.
   - Para crear un dispositivo virtual, ve a `Tools > AVD Manager`, y crea un nuevo dispositivo virtual según tus preferencias.
   - Para ejecutar en un dispositivo físico, asegúrate de tener habilitado el modo de desarrollador y la depuración USB en el dispositivo, y conéctalo a tu computadora.
4. **Ejecución del proyecto:**
   - Selecciona el dispositivo en el que deseas ejecutar la aplicación (dispositivo virtual o físico) en la barra de herramientas.
   - Haz clic en el botón "Run" (el ícono de reproducción) o presiona `Shift + F10`.
   - Espera a que Android Studio compile el proyecto y lo ejecute en el dispositivo seleccionado.

5.1.2 Mediante la APK Instalándola

1. **Generación de la APK:**
   - En Android Studio, ve a `Build > Build Bundle(s) / APK(s) > Build APK(s)`.
   - Espera a que Android Studio termine de compilar el proyecto. La APK generada estará disponible en el directorio `app/build/outputs/apk/debug/`.
2. **Transferencia de la APK a un Dispositivo:**
   - Conecta tu dispositivo Android a tu computadora mediante un cable USB.
   - Transfiere la APK generada al almacenamiento del dispositivo.
3. **Instalación de la APK:**
   - En tu dispositivo Android, abre un explorador de archivos.
   - Navega hasta donde hayas transferido la APK.
   - Toca la APK para comenzar el proceso de instalación.
   - Sigue las instrucciones en pantalla para completar la instalación.

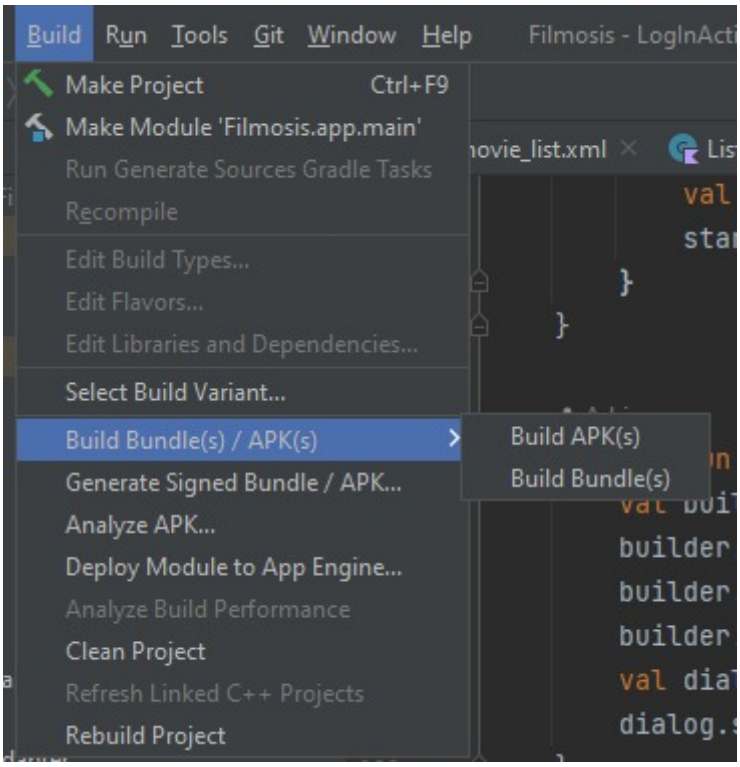


Figure 2: Captura de pantalla de Android Studio durante el proceso de generación de la APK.

4. **Ejecución de la Aplicación:**
    - Una vez instalada, puedes encontrar y ejecutar la aplicación como cualquier otra aplicación en tu dispositivo Android.

5.2 Funcionalidades principales y cómo utilizarlas

5.2.1 Página de Inicio

La página de inicio cuenta con tres botones principales:

- **Iniciar sesión con Google:** Permite a los usuarios iniciar sesión utilizando sus cuentas de Google.
- **Crear una cuenta:** Permite a los nuevos usuarios crear una cuenta en la aplicación.
- **Iniciar sesión con usuario y contraseña:** Permite a los usuarios iniciar sesión utilizando sus credenciales de usuario.

Figure 3: Captura de pantalla del proceso de instalación de la APK en un dispositivo Android.

Además, en la página de inicio se muestran recomendaciones de películas y películas en tendencia, junto con un buscador para encontrar películas, actores y directores.

5.2.2 Página de Explorar

En la página de explorar, los usuarios pueden:

- **Filtrar por géneros:** Se puede filtrar la lista de películas por género.
- **Ver películas recientes, populares, mejor valoradas y próximos estrenos:** Se muestran listas de películas organizadas por diferentes criterios.

5.2.3 Perfil de Usuario

En el perfil de usuario, los usuarios pueden gestionar su cuenta, cambiar su foto de perfil y contraseña.

5.2.4 Mis Listas

En la sección de "Mis Listas", los usuarios pueden ver y gestionar las listas de películas que han creado.

6 Arquitectura

El proyecto de Android Studio sigue una arquitectura basada en el patrón Modelo-Vista-ViewModel (MVVM) junto con elementos de la arquitectura limpia (Clean Architecture).

6.1 Componentes de la Arquitectura

- **Modelo:** Las *data classes* representan el Modelo en la arquitectura MVVM. Estas clases contienen los datos utilizados en la aplicación, así como la lógica relacionada con los datos.
- **Vista:** Los layouts XML definidos en el proyecto representan la Vista en la arquitectura MVVM. Estos layouts definen la estructura visual de la interfaz de usuario de la aplicación.
- **ViewModel:** Los conectores mencionados probablemente actúan como el ViewModel en la arquitectura MVVM. Estos conectores actúan como intermediarios entre la Vista y el Modelo, proporcionando los datos necesarios para la interfaz de usuario.
- **Firebase:** Se utiliza Firebase para la conexión externa, lo que implica que el proyecto también integra una capa de datos remotos. Firebase puede encajar en la arquitectura limpia como un Adaptador de Interfaces o una capa de Infraestructura, proporcionando acceso a los datos externos.

7 Componentes del Proyecto

7.1 Layouts

Los layouts definidos en el proyecto definen la estructura visual de la interfaz de usuario de la aplicación. Están escritos en XML y pueden incluirse tanto en actividades como en fragmentos.

7.2 Activities y Fragments

Las actividades y fragmentos representan las diferentes pantallas y componentes de la interfaz de usuario de la aplicación.

7.3 Adapters

Los adapters se utilizan para vincular conjuntos de datos a los componentes de la interfaz de usuario, como los RecyclerViews. Estos adapters pueden proporcionar los datos necesarios desde las data classes.

7.4 Data Classes

Las data classes representan los modelos de datos utilizados en la aplicación. Contienen la información necesaria para representar los objetos en la interfaz de usuario.

7.5 Conectores

Los conectores se utilizan para interactuar con fuentes de datos externas, como Firebase. Actúan como intermediarios entre los datos externos y la lógica de la aplicación.

8 Base de datos

Hemos empleado como base de datos Firestore Database, una de las opciones que nos ofrece Firebase. Firestore Database es una base de datos NoSQL, lo que significa que permite consultar datos fuera de las estructuras tradicionales que se encuentran en las bases de datos relacionales. Para almacenar los datos, hemos creado tres colecciones: Favorites, Lists y Users.

8.1 Users

En esta colección guardaremos los datos del inicio de sesión. Es decir, guardaremos el correo electrónico, nombre de usuario, fecha de nacimiento y, por último, nombre y apellidos.

8.2 Lists

En esta colección guardaremos la información de la lista. Cuando el usuario cree una lista en la base de datos, se guardará la fecha, el ID, nombre de la lista, la descripción y una lista de películas vacía. Cuando el usuario comience a añadir películas, se guardará la información de la película en la lista de películas.

9 Documentación de código

Este proyecto ha sido documentado utilizando Dokka, una herramienta de generación de documentación para proyectos Kotlin. Dokka analiza el código fuente y genera documentación en formato HTML, Markdown o Javadoc a partir de los comentarios del código y las anotaciones especiales.

9.1 Convenciones de nomenclatura:

Las convenciones de nomenclatura son reglas y estándares utilizados para nombrar variables, funciones, clases y otros elementos dentro del código. Estas convenciones son importantes para mantener la consistencia y la legibilidad en el código base.

En este proyecto, se siguen las siguientes convenciones de nomenclatura:

- CamelCase para variables y funciones.
- PascalCase para clases y tipos.
- snake_case para constantes y variables.
- UPPERCASE para constantes inmutables.

Se recomienda seguir estas convenciones de nomenclatura en todo el código para mantener la coherencia y facilitar la comprensión del mismo.

9.2 Comentarios de código

- Para documentar clases, funciones y propiedades, se utilizan comentarios de bloque en formato KDoc, comenzando con /** y terminando con */.
- Los comentarios KDoc pueden incluir etiquetas especiales como @param para describir los parámetros de una función, @return para describir el valor de retorno, y otras etiquetas como @throws, @see, entre otras.

```kotlin
/**
 * Resuelve el color del tema.
 * @param context El contexto de la aplicación.
 * @param attr El atributo de color a resolver.
 * @return El color resuelto.
 */
Adrian
private fun resolveThemeColor(context: Context, attr: Int): Int {
    val typedValue = TypedValue()
    val theme = context.theme
    theme.resolveAttribute(attr, typedValue, resolveRefs: true)
    return typedValue.data
}
```

Figure 4: Ejemplo de función documentada con dokka

- Además de los comentarios KDoc, Dokka también admite anotaciones especiales dentro del código fuente, como @Deprecated para marcar elementos obsoletos y proporcionar una razón para su deprecación.

10 Pruebas

Durante el desarrollo de Filmosis, se llevaron a cabo diversas pruebas para garantizar la calidad y el funcionamiento adecuado de la aplicación. A continuación, se describen las principales pruebas realizadas:

10.1 Pruebas Unitarias

Las pruebas unitarias se centran en probar unidades individuales de código, como métodos o funciones, de manera aislada del resto del sistema. Estas pruebas se utilizan para verificar que cada unidad de código funcione correctamente según lo esperado. En Filmosis, se escribieron pruebas unitarias para verificar el comportamiento de funciones específicas dentro de los componentes de la aplicación, como la lógica de negocios, el manejo de datos y las interacciones con las API externas.

10.2 Pruebas de Integración

Las pruebas de integración evalúan la interacción entre diferentes componentes o módulos de la aplicación para asegurar que funcionen correctamente juntos. Estas pruebas son útiles para identificar problemas de comunicación o interoperabilidad entre los distintos elementos de la aplicación. En el caso de Filmosis, se realizaron pruebas de integración para verificar la comunicación adecuada entre las diferentes capas de la aplicación, como la interfaz de usuario, la lógica de negocios y el acceso a datos.

10.3 Pruebas de Aceptación del Usuario (UAT)

Las pruebas de aceptación del usuario implican validar la funcionalidad de la aplicación desde la perspectiva del usuario final. Estas pruebas se centran en asegurar que la aplicación cumpla con los requisitos y expectativas del usuario, así como en identificar posibles áreas de mejora desde una perspectiva de experiencia de usuario. En el caso de Filmosis, se realizaron pruebas de UAT para evaluar la usabilidad, la accesibilidad y la satisfacción general del usuario con la aplicación.

10.4 Pruebas Unitarias en Fragmentos

Se llevaron a cabo pruebas unitarias en los fragmentos de la aplicación para garantizar su correcto funcionamiento y comportamiento esperado. A continuación, se describen las pruebas realizadas en los fragmentos ‘PeliculaSeleccionadaFragment‘ y ‘ExploreFragment‘:

10.4.1 PeliculaSeleccionadaFragment

En el fragmento ‘PeliculaSeleccionadaFragment‘, se realizaron varias pruebas para verificar el correcto funcionamiento del cálculo del rating de la película.

- **testCalculoEstrellas_devuelve_la_calificación_esperada**: Esta prueba se encarga de verificar si el cálculo del rating se realiza correctamente y devuelve la calificación esperada. Para esto, se simulan diferentes escenarios, incluyendo una calificación de cero y la calificación máxima, y se verifica si el rating calculado coincide con el rating esperado.

- **testCalculoEstrellas_puntaje_cero**: En esta prueba se verifica el comportamiento del cálculo del rating cuando la calificación de la película es cero. Se asegura de que el rating calculado sea igual a cero, como se espera.
- **testCalculoEstrellas_puntaje_maximo**: Esta prueba evalúa el cálculo del rating cuando la calificación de la película alcanza el valor máximo posible. Se verifica si el rating calculado coincide con el número máximo de estrellas, asegurando que el rating se muestre correctamente en la interfaz de usuario.

Estas pruebas aseguran que el cálculo del rating en el fragmento 'PeliculaSeleccionadaFragment' funcione correctamente en una variedad de situaciones, garantizando una experiencia consistente para el usuario.

10.4.2 ExploreFragment

En el fragmento 'ExploreFragment', se llevaron a cabo pruebas para validar el funcionamiento de la vista de búsqueda y los botones de filtro. En la prueba 'testSearchView', se simula el ingreso de una consulta de búsqueda y se verifica que el resultado de la búsqueda sea visible en el RecyclerView correspondiente. En la prueba 'testFilterButtons', se realiza una acción de clic en un botón de filtro y se asegura que el RecyclerView de películas filtradas se actualice correctamente.

Estas pruebas unitarias en los fragmentos son fundamentales para garantizar un comportamiento consistente y sin errores en la aplicación Filmosis.

10.4.3 AuthActivityUnitTest

La clase `AuthActivityUnitTest` contiene pruebas unitarias para verificar el comportamiento de la sesión de usuario al iniciar la aplicación. A continuación, se describe la prueba realizada en esta clase:

- **test session with existing session**: Esta prueba verifica el comportamiento de la sesión cuando ya existe una sesión iniciada al iniciar la aplicación. Se simula una sesión previamente iniciada almacenando un correo electrónico y un proveedor en las preferencias compartidas. Luego, se llama al método `session()` de la actividad de autenticación para verificar su comportamiento. Finalmente, se verifica que, como resultado de tener una sesión iniciada, la visibilidad del diseño principal (`mainLayout`) de la actividad sea invisible, indicando que la actividad debería redirigir al usuario automáticamente a la pantalla principal de la aplicación, omitiendo la pantalla de inicio de sesión.

Esta prueba unitaria asegura que el comportamiento de la sesión de usuario en la aplicación sea coherente y que se maneje correctamente la existencia de una sesión previamente iniciada al iniciar la aplicación.

10.4.4 MoviesAccessTest

La clase `MoviesAccessTest` contiene pruebas unitarias para verificar el funcionamiento del acceso a películas en la aplicación Filmosis. A continuación, se describen las pruebas realizadas en esta clase:

- **listPopularMovies_success**: Esta prueba verifica el comportamiento de la función `listPopularMovies()` cuando la solicitud de películas populares tiene éxito. Se simula la respuesta exitosa del servicio de API de TMDb, lo que debería proporcionar una lista de películas. Se espera que la lista de películas devuelta no esté vacía, lo que indica que se recuperaron películas populares correctamente.
- **listPopularMovies_failure**: Esta prueba verifica el comportamiento de la función `listPopularMovies()` cuando la solicitud de películas populares falla. Se simula una respuesta de error del servicio de API de TMDb con un código de estado 404. Se espera que la lista de películas devuelta sea nula, indicando que no se pudieron recuperar películas populares debido a un error en la solicitud.

Estas pruebas unitarias garantizan que el acceso a películas en la aplicación Filmosis se maneje correctamente tanto en casos de éxito como de fallo, proporcionando una experiencia de usuario consistente y confiable.

10.5 Pruebas de Interfaz de Usuario (UI)

Para asegurar el correcto funcionamiento de la interfaz de usuario de la aplicación, se realizaron pruebas de UI utilizando Espresso, un framework de pruebas de interfaz de usuario para aplicaciones Android. A continuación, se detallan las pruebas realizadas en la clase `CrearCuentaGUI`, que se encargan de verificar el proceso de creación de cuenta en la aplicación:

10.5.1 CrearCuentaGUI

La clase `CrearCuentaGUI` contiene pruebas de interfaz de usuario para el proceso de creación de cuenta en la aplicación. Estas pruebas cubren diversos aspectos de la interfaz de usuario, incluyendo la interacción con diferentes elementos de la pantalla y la validación de los datos ingresados. A continuación, se describen las acciones realizadas y las verificaciones realizadas en estas pruebas:

- **Crear cuenta con datos válidos:** Se simula el proceso de creación de cuenta ingresando datos válidos en los campos requeridos, como nombre de usuario, correo electrónico y contraseña. Se verifica que al hacer clic en el botón "CREAR CUENTA", se complete el proceso de registro correctamente sin errores.
- **Intento de crear cuenta con una cuenta existente:** Se simula un escenario en el que se intenta crear una cuenta con datos que ya están

asociados a una cuenta existente en la base de datos. Se espera que al hacer clic en el botón "CREAR CUENTA", se muestre un mensaje de error indicando que la cuenta ya existe y que el proceso de registro no se complete.

- **Validación de campos requeridos:** Se realizan pruebas para verificar que los campos obligatorios, como nombre de usuario, correo electrónico y contraseña, deben ser completados antes de poder crear una cuenta. Se verifica que al intentar crear una cuenta con campos vacíos, se muestren mensajes de error indicando que los campos son obligatorios.
- **Selección de fecha de nacimiento:** Se simula la interacción del usuario con el campo de fecha de nacimiento, seleccionando una fecha específica. Se verifica que la fecha seleccionada se muestre correctamente y que el usuario pueda confirmar la selección.

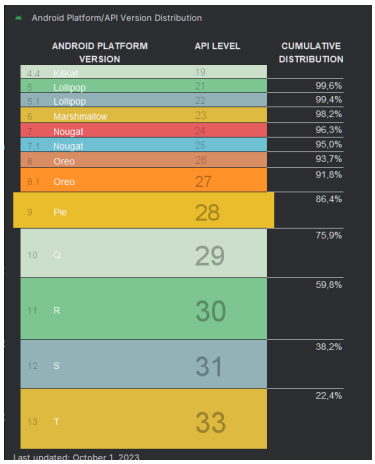
10.5.2 AccederCuentaGUIespresso

La clase `AccederCuentaGUIespresso` contiene pruebas de interfaz de usuario utilizando Espresso para verificar el proceso de inicio de sesión en la aplicación. A continuación, se detallan las acciones realizadas y las verificaciones realizadas en estas pruebas:

- **Acceso a la pantalla de inicio de sesión:** La prueba comienza simulando el clic en el botón "CREAR CUENTA" en la pantalla de inicio. Luego, se hace clic en el botón "VOLVER" en la pantalla de registro para regresar a la pantalla de inicio, y finalmente se hace clic en el botón "ACCEDER" para acceder a la pantalla de inicio de sesión.
- **Intento de inicio de sesión con datos de cuenta incorrectos:** Se simula el ingreso de datos de cuenta incorrectos, como una dirección de correo electrónico incompleta y una contraseña incorrecta. Se espera que al hacer clic en el botón "INICIAR SESIÓN", no se complete el inicio de sesión y se muestren mensajes de error indicando los campos incorrectos.
- **Intento de inicio de sesión con datos de cuenta válidos:** Se ingresan datos de cuenta válidos, incluida una dirección de correo electrónico válida y la contraseña correspondiente. Luego, se hace clic en el botón "INICIAR SESIÓN" y se verifica que el proceso de inicio de sesión se complete correctamente, permitiendo al usuario acceder al contenido protegido de la aplicación.

Estas pruebas de interfaz de usuario garantizan que el proceso de inicio de sesión en la aplicación funcione correctamente y proporcione una experiencia fluida para el usuario.

11 Análisis de requisitos y especificaciones



Esta aplicación solo está operativa para dispositivos móviles que tengan Android como sistema operativo. Para realizar este proyecto, hemos utilizado la versión de distribución API Pie, API LEVEL 28, con una distribución acumulativa del 86.4%. Es decir, esta aplicación solo soporta versiones con Android 9.0 como mínimo.

12 Consideraciones de Seguridad y Privacidad

En nuestro proyecto, nos tomamos muy en serio la seguridad y la privacidad de nuestros usuarios. Implementamos diversas medidas para garantizar la protección de la información confidencial y mantener la integridad de los datos. A continuación, destacamos algunas de estas consideraciones:

- Almacenamiento Seguro: Todos los datos sensibles se almacenan de forma segura utilizando protocolos de encriptación. Esto garantiza que la información confidencial, como contraseñas o datos personales, esté protegida contra accesos no autorizados.
- Política de Contraseñas: Aunque mencionamos que no almacenamos contraseñas, es importante destacar que fomentamos prácticas seguras de contraseñas entre nuestros usuarios. Recomendamos el uso de contraseñas únicas y complejas, así como la habilitación de la autenticación de dos factores cuando sea posible.
- Protección de Datos Personales: Cumplimos con las regulaciones de protección de datos, como el Reglamento General de Protección de Datos (GDPR), para garantizar que los datos personales de nuestros usuarios estén protegidos y se utilicen de manera ética y legal.
- Auditorías de Seguridad: Realizamos auditorías regulares de seguridad para identificar posibles vulnerabilidades en nuestro sistema y tomar medidas correctivas de manera proactiva. Esto nos ayuda a garantizar la integridad y la confidencialidad de los datos de nuestros usuarios.
- Transparencia y Comunicación: Mantenemos una comunicación abierta y transparente con nuestros usuarios sobre las medidas de seguridad y

privacidad implementadas en nuestro proyecto. Esto les permite estar informados y tomar decisiones fundamentadas sobre su participación en nuestra plataforma.

En resumen, nuestra prioridad es garantizar la seguridad y la privacidad de nuestros usuarios en todo momento. Estamos comprometidos a seguir mejorando nuestras prácticas de seguridad y privacidad para mantener la confianza de nuestra comunidad de usuarios.

13 Publicación y distribución

13.1 Instrucciones para compilar y generar versiones de la aplicación

Para compilar y generar versiones de la aplicación, sigue estos pasos:

1. Dirígete al repositorio de GitHub de Filmosis: `https://github.com/agarciagale/Filmosis.git`.
2. Haz clic en el botón "Code" y copia el enlace proporcionado.
3. Abre tu terminal y ejecuta el siguiente comando para clonar el repositorio:

```
git clone <enlace_del_repositorio>
```

4. Asegúrate de tener instalado un IDE compatible con Kotlin, como Android Studio.
5. Navega hasta el directorio raíz del proyecto clonado en tu terminal.
6. Ejecuta el siguiente comando para compilar la aplicación:

```
./gradlew build
```

   Esto generará un archivo .jar en el directorio de salida del proyecto.

7. Para generar una versión lista para distribuir, utiliza Gradle con el siguiente comando:

```
./gradlew assembleRelease
```

   Esto generará una versión de la aplicación lista para ser distribuida.

13.2 Proceso de publicación en tiendas de aplicaciones o distribución interna

Para publicar la aplicación en tiendas de aplicaciones como Play Store, sigue estos pasos:

1. Sigue los pasos específicos de cada tienda de aplicaciones. Normalmente, necesitarás crear una cuenta de desarrollador y seguir las pautas de publicación proporcionadas por la tienda.

2. Si deseas distribuir la aplicación internamente, puedes utilizar servicios como Firebase App Distribution o Google Play Console (para distribución cerrada). Configura los ajustes necesarios y sigue las instrucciones proporcionadas por el servicio elegido.

14 Mantenimiento y contribuciones

14.1 Políticas de Ramificación

En nuestro proyecto, hemos adoptado un enfoque de control de versiones utilizando un modelo de ramificación simple. Hemos mantenido dos ramas principales:

- **main**: En esta rama principal, se desarrolla el proyecto de manera activa. Aquí se encuentran las últimas características implementadas y correcciones de errores.

- **Release**: Una vez que el proyecto está listo para ser lanzado como producto, lo subimos a la rama Release. Esta rama contiene versiones estables del software que han pasado por pruebas exhaustivas y están listas para su implementación.

14.2 Proceso de Reporte de Fallos y Gestión de Problemas

Reporte de Fallos:
Si los usuarios encuentran algún fallo o bug, pueden reportarlo de las siguientes maneras:

1. **Portal de Problemas en GitHub**: Los usuarios pueden navegar al apartado de "Issues" en nuestro repositorio de GitHub. Allí, podrán ver los problemas reportados por otros usuarios y contribuir a solucionarlos si lo desean.

2. **Contacto con el Equipo de Desarrollo**: Los usuarios también pueden ponerse en contacto directamente con cualquiera de los desarrolladores para informar sobre un error que hayan encontrado.

3. **Pull Requests**: Si un usuario identifica un error y puede proporcionar una solución, puede enviar un "pull request" con los cambios propuestos. Esto permite una colaboración directa en la resolución de problemas.

Mejoras al Proceso:
Para mejorar nuestro proceso de gestión de problemas y contribuciones, estamos considerando implementar las siguientes mejoras:

- **Portal de Soporte Centralizado**: Estamos trabajando en la creación de un portal de soporte centralizado donde los usuarios puedan registrar sus problemas de manera estructurada, proporcionando información detallada sobre el error encontrado.
- **Formulario de Reporte de Errores**: Estamos desarrollando un formulario de reporte de errores que estandarizará la información recopilada y facilitará el seguimiento y la resolución de problemas.
- **Automatización de Gestión de Problemas**: Estamos explorando opciones para automatizar la gestión de problemas utilizando herramientas de seguimiento de problemas, lo que nos permitirá asignar responsabilidades y realizar un seguimiento más eficiente del progreso de cada problema reportado.

Estas mejoras están destinadas a hacer que nuestro proceso de reporte de fallos y gestión de problemas sea más eficiente y transparente, garantizando una mejor experiencia para nuestros usuarios y una colaboración más efectiva dentro de nuestro equipo de desarrollo.

15 Recursos adicionales

Documentación oficial de Kotlin:
`https://kotlinlang.org/docs/home.html`

Tutoriales en YouTube:
`https://youtu.be/dpURgJ4HkMk?si=k95-dp7pOh9wQ-0w`
`https://www.youtube.com/watch?v=KYPc7CAYJOw`
`https://youtu.be/jjzQvNRVhx8?si=NMKF_urWQw9Lp668`
`https://youtu.be/vJapzH_46a8?si=2y6kzU5f11wO4W0g`
`https://youtube.com/playlist?list=PL_QrnahGtqYAtYIoGodOOkutYTvhq4HMU&si=bJn_PD4a1aPIUULR`

Comunidades y foros:
`https://es.stackoverflow.com/`

Documentación TMDB:
`https://developer.themoviedb.org/reference/intro/getting-started`

ChatGPT:
`https://chat.openai.com/`

16 Contacto

Nos podéis contactar de la siguiente manera:

Adrián García Galera: `agarciagal6@educacion.navarra.es`
Aritz Arrondo Villaplana: `aarrondvil@educacion.navarra.es`
Kaiet Jiménez Aldasoro: `kjimeneald@educacion.navarra.es`