UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Gestor de películas

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Bespal, Yevgeniy

Tutor: Escobar Román, Santiago

2018-2019

Resumen

En un mundo donde todo está al alcance de las manos, o mejor dicho de los dedos de los usuarios, es necesario aprovechar esta ventaja con aplicaciones que sean realmente útiles en el día a día de las personas. Los dispositivos telefónicos y Tablet muchas veces disponen de complementos y aplicaciones que no hacen otra cosa que entorpecer al individuo, lo cual se traduce en una pérdida de tiempo efectiva. Por esta razón, en este trabajo de fin de grado se desarrollará una aplicación de información sobre películas para dispositivos móviles y *tablets*, con el fin de llegar a un mayor número de usuarios y tener la valoración positiva.

Con esta aplicación se busca agilidad, funcionalidad y facilidad de manejo con el fin de poder consultar todo lo relacionado con el mundo cinematográfico. Una aplicación sencilla, pero a la vez útil basada en la capacidad de consultar en todo momento las películas mejor valoradas, recientemente estrenadas, las más populares, etc. Con tan solo desbloquear el dispositivo y en unos cuantos segundos se puede obtener la información sin acudir a otros dispositivos como el ordenador.

Para su desarrollo se van a utilizar las tecnologías actuales y más importantes en este sector laboral como: el Entorno de Desarrollo Integrado (IDE) Android Studio, el lenguaje de programación Kotlin, las librerías Retrofit, RxJava, OkHttp, Picasso, etc. Aparte para el diseño se utilizará Material Design. Toda la información se obtendrá a través de la API que nos proporciona The Movie Database.

Palabras clave: android, aplicación móvil, gestor de películas, the movie database api.

Abstract

In the world where everything is within of a hand's reach, or rather within users' fingertips reach, it is necessary to take advantage of the applications that are really useful in people's daily lives. Mobile devices and Tablet often have accessories and applications that do nothing but hinder the individual, which passes into an effective loss of time. For this reason, in this final thesis an application of information about movies will be developed for mobile devices and tablets, in order to reach a greater number of users and have a positive assessment.

With this application we look for flexibility, functionality and ease of use in order to be able to research everything related to the cinematographic world. A simple application but at the same time useful, based on the possibility to check at all times the best rated movies, recently released ones, the most popular ones, etc. You just have to unlock the device and in a few seconds you can get the information with no need to use other devices such the computer.

For its development, the current and most important technologies in this labor sector will be used, such as: the Integrated Development Environment (IDE) Android Studio, the Kotlin programming language, the Retrofit libraries, RxJava, OkHttp, Picasso, etc. Besides that, for the design Material Design will be used. All information will be obtained through the API provided by The Movie Database.

Keywords: android, mobile application, movie manager, the movie database api

Resum

En un món on tot està a l'abast de les mans, o millor dit dels dits dels usuaris, és necessari aprofitar aquest avantatge amb aplicacions que siguen realment útils en el dia a dia de les persones. Els dispositius telefònics I Tablet moltes vegades disposen de complements i aplicacions que no fan una altra cosa que entorpir a l'individu, la qual cosa es tradueix en una pèrdua de temps efectiva. Per aquesta raó, en aquest treball de fi de grau es desenvoluparà una aplicació informativa per a dispositius mòbils i tablets, amb la finalitat d'arribar a un major nombre d'usuaris i tindre la valoració positiva.

Amb aquesta aplicació es busca agilitat, funcionalitat i facilitat de maneig amb la finalitat de poder consultar tot el relacionat amb el món cinematogràfic. Una aplicació senzilla, però alhora útil basada en la capacitat de consultar en tot moment les pel·lícules millor valorades, recentment estrenades, les més populars, etc. Amb tan sols desbloquejar el dispositiu i en uns quants segons es pot obtindre la informació sense acudir a altres dispositius com l'ordinador.

Per al seu desenvolupament s'utilitzaran les tecnologies actuals i més importants en aquest sector laboral com: l'Entorn de Desenvolupament Integrat (IDE) Android Studio, llenguatge de programació Kotlin, les llibreries Retrofit, RxJava, OkHttp, Picasso, etc. A part per al disseny s'utilitzarà Material Design. Tota la informació s'obtindrà a través de la API que ens proporciona The Movie Database.

Paraules clau: android, aplicació mòbil, gestor de pel·lícules, the movie database api.

Tabla de contenido

Índice de tablas

Tabla de ilustraciones

1. Introducción

En este apartado se proporciona una breve explicación sobre el presente trabajo, describiendo su motivación, contexto, objetivos y para terminar, la estructura de este documento.

1.1 Motivación

Hoy en día muchas personas son apasionadas del séptimo arte. Cada mes salen a producción miles de películas y nuevos episodios y temporadas de las series que nos mantienen pegados a la pantalla. A muchos de nosotros nos pasa que al querer visionar alguna película o serie de la que ya hemos disfrutado, después de algún tiempo no recordamos el nombre de ésta. Por ello surge la idea de tener una lista ya sea de nuestras películas favoritas o que aún tenemos pendientes de ver.

Tampoco siempre estamos al tanto de las novedades y a veces olvidamos las fechas de lanzamiento de las películas o simplemente queremos ser notificados cuando salga la película a la cartelera del cine.

En otras ocasiones simplemente queremos buscar una película buena y con un determinado género para verla, sea solos o en compañía. En este caso también nos podrá ayudar la aplicación.

1.2 Objetivos

El principal objetivo de esta aplicación es proporcionar una ayuda a las personas que usuarios asiduos a este mundo. Es decir, facilitarles una herramienta para la gestión de películas, en la que poder añadir las películas favoritas, gestionar las visiones futuras, mejor valoradas etc. La aplicación proporcionará un listado de las películas con el *rating* por los críticos y el género deseado para que podamos elegir la película que queramos ver.

Aparte, otro de los objetivos clave es la notificación previa al estreno de alguna película preseleccionada por nosotros. Podremos especificar el tiempo de antelación del aviso en la parte de opciones.

Respecto al objetivo del autor, sería profundizar en el desarrollo de aplicaciones móviles bajo el sistema operativo Android y aprender buenas técnicas de desarrollo, patrones de diseño, arquitecturas y finalmente el lenguaje de programación que se utiliza mucho hoy en día, Kotlin.

1.3 Estructura de la memoria

El documento está estructurado en los siguientes nueve apartados:

1. **Introducción**: se describe la motivación y los objetivos del proyecto.
2. **Estudio estratégico**: en este apartado se documentan otras aplicaciones similares a este proyecto que hay en el mercado.
3. **Especificación**: en este apartado se describe la arquitectura que se utiliza en el proyecto, así como la metodología del desarrollo, los requisitos no funcionales, casos de uso y diseño de la aplicación.
4. **Tecnologías y herramientas**: se describen las tecnologías y herramientas utilizadas a lo largo del proyecto.
5. **Implementación**: se describe el proceso de implementación incluyendo codificación, librerías, algoritmos destacados, etc.
6. **Pruebas**: se presentan los aspectos relevantes al proceso de las pruebas de la aplicación.
7. **Resultado final**: se expone el ejemplo mediante las capturas de una aplicación totalmente funcional.
8. **Conclusiones y futuros trabajos**: se describen las conclusiones y las mejoras que se podrían desarrollarse en futuro.
9. **Referencias**: se incluyen las referencias de las fuentes utilizadas a lo largo del proyecto.

2. Estudio estratégico

En esta parte se estudian y se analizan varias aplicaciones que tienen una funcionalidad muy parecida a la de la aplicación del proyecto. Después se hace una breve conclusión.

Entre las aplicaciones disponibles en Google Play se van a analizar varias como: Movie Manager (Liam O.Gorman) y Movie Manager (Anónimo).

2.1 Aplicación (Movie Manager)

2.1.1 Descripción

La aplicación Movie Manager está disponible en la tienda de aplicaciones Google en el momento de redactar este documento [5]. El autor de esta obra es Liam O Gorman. La aplicación es compatible con Android 4.1 y versiones posteriores. Y es la primera versión lanzada del autor.

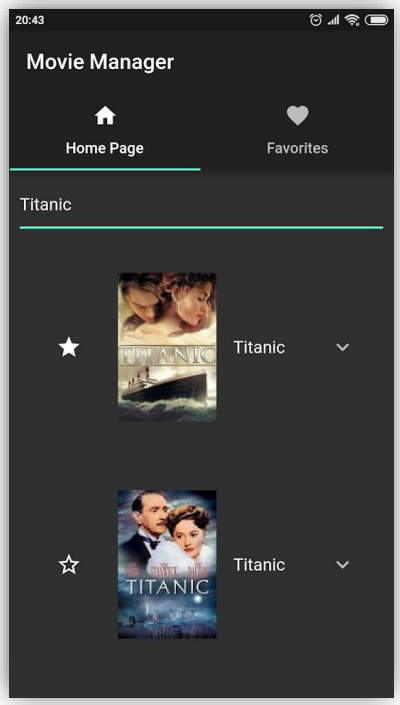

Ilustración 1: App Movie Manager

Es una aplicación que permite la búsqueda de películas en el servidor y posteriormente la posibilidad de guardar estas en la lista de películas favoritas en la base de datos.

2.1.2 Principales funcionalidades

Las principales funcionalidades de esta aplicación son:

- Crear una lista de todas las películas que se han visto.
- Buscar cualquier película usando la base de datos de películas (TMDb).
- Guardar cualquier película usando el icono de estrella.
- Eliminar las películas con el ícono de la basura.

2.2 Aplicación (Movie Manager)

2.2.1 Descripción

La aplicación Movie Manager está disponible en la tienda de aplicaciones Google en el momento de redactar este documento [15]. El autor de esta obra es un centro de aprendizaje y desarrollo de sistemas de información de gestión MIS. La aplicación es compatible con Android 4.0.3 y versiones posteriores. Y es la primera versión lanzada del autor.

Es una aplicación que permite mostrar las últimas películas en el cine con descripción, votos y calificaciones.

Cabe destacar que la actual aplicación no está del todo desarrollada, ya que faltan por implementar las configuraciones, la posibilidad de iniciar la cuenta del usuario en la aplicación, cerrarla y posiblemente la opción de búsqueda. Pero a pesar de todo ello se puede tener esta aplicación como ejemplo para una comparación/análisis.

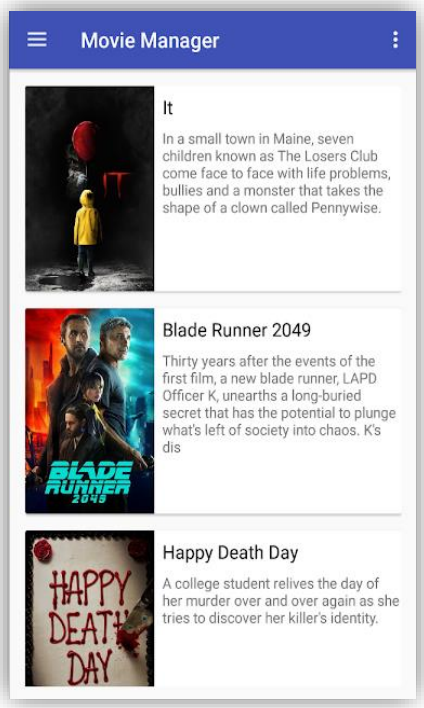

Ilustración 2: App Movie Manager

2.2.2 Principales funcionalidades

Las principales funcionalidades que nos trae esta aplicación son:

- Mostrar la lista de las últimas películas en el cine.
- Mostrar la lista de las próximas películas que van a aparecer en la cartelera de los cines.
- Mostrar la descripción de cada una de las películas en la lista.
- Mostrar la calificación de cada una de las películas.
- Mostrar los votos de los usuarios pertenecientes a cada película.

2.3 Análisis

La primera aplicación propone una funcionalidad bastante interesante de búsqueda de películas, añadirlas a la lista de las películas favoritas y borrarlas. No obstante, la aplicación tiene una interfaz muy pobre y le falta mucha funcionalidad como por ejemplo aplicar filtros a la búsqueda y ordenar las películas según la clasificación o popularidad. También le falta mostrar los detalles de las películas como la calificación, el año de lanzamiento, el autor, etc. Otro de los detalles que se puede destacar es que a priori el usuario debe saber el nombre de la película para poder encontrarla, ya que al principio no aparece una lista con la cual al hacer *scroll* para poder encontrar la adecuada para nuestro propósito.

La segunda aplicación es un poco lo contrario a la primera, tiene una interfaz amigable, pero respecto a la funcionalidad, a pesar de que la aplicación aún no está acabada, le falta una búsqueda para poder encontrar las películas, también los filtros, los ajustes, la ordenación, la actualización para descargar los datos actuales del servidor, etc.

2.4 Conclusión

Tanto la primera como la segunda aplicación analizadas presentan unas funcionalidades muy interesantes. Con el análisis realizado podemos priorizar los requisitos mediante una tabla MoSCow.

Debe tener *(Must have)*	• Una lista de películas favoritas. • Una búsqueda. • Calificación de películas. • Detalle de película. • Actualización de listas. • Una lista de películas populares. • Una lista de películas mejor valoradas. • Una lista de películas disponibles en los cines.
Debería tener (*Should have)*	• Una lista de próximos estrenos. • Un filtro. • Una lista con películas para ver. • Ordenar por. • Ajustes. • Una interfaz intuitiva.
Podría tener (*Could have)*	• Reproducir *trailers*. • Ayuda. • Cuenta de usuario. • Comentarios de las personas. • Avisos de las películas que se estrenan próximamente.
No tendrá *(Won't have)*	• Videos con películas enteras. • Información de los autores. • Información de los actores.

Tabla 1: MoSCow

Como se puede observar la tabla tiene prioridades de mayor a menor.

Lo que se pretende con esta aplicación es buscar las películas permitiendo así ordenarlas por calificación o popularidad y filtrar las películas. También es posible mostrar la lista de las películas que se encuentran en cartelera en la actualidad o las que se estrenarán próximamente, así como reproducir los "*trailers*" de los *films*.

Otra de las funcionalidades claves de la aplicación es añadirlas en la lista de las películas favoritas, así como también a la lista de películas para ver.

La aplicación no tiene videos con las películas completas, información sobre el autor o el reparto de la película.

Como futuras mejoras se pueden proponer apartados de la tabla anterior no incluidos en la aplicación.

3. Especificación

En esta sección se describe la metodología de desarrollo que ha elegido el autor, así como la arquitectura global que se utiliza en el proyecto y las capas de las que se compone la arquitectura. También se describen varios aspectos que se han tenido en cuenta a la hora de desarrollar la aplicación.

Por otra parte, se describen los requisitos funcionales y no funcionales de la aplicación. Y se representan los bocetos de la interfaz de las distintas pantallas utilizadas por la aplicación.

Para empezar, se describe la arquitectura Model-View-ViewModel en general y cada una de las capas.

3.1 Metodología de desarrollo

El autor ha preferido elegir una metodología ágil eXtreme Programming (Programación Extrema) ya que lo que busca es mejorar la calidad de desarrollo. Y por lo tanto ha seguido las siguientes fases en el proceso:

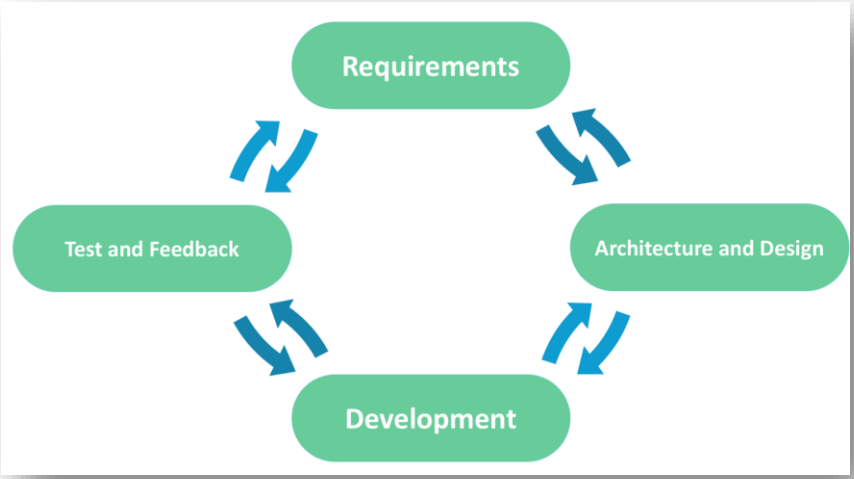

Ilustración 3: Programación extrema

- Planificación del proyecto: en esta fase del proyecto se definen las historias de usuario y los requisitos de la aplicación.
- Diseño: en esta fase se establece la arquitectura de la aplicación, así como su interfaz. Se traducen los requisitos en una representación software.
- Codificación: en esta fase se implementa la funcionalidad de la aplicación.
- Pruebas: en esta fase se evalúa el comportamiento de la aplicación realizando unas pruebas para detectar posibles errores o inconsistencias.

3.2 Arquitectura MVVM

Se ha decidido por este patrón arquitectónico ya que se acopla perfectamente para la realización de las aplicaciones que tienen la estructura de cliente-servidor. Aparte los desarrolladores de Google propusieron un conjunto de librerías para facilitar el desarrollo de las aplicaciones con la dicha arquitectura.

Como podemos ver en la Ilustración 4: Arquitectura cliente-servidor tenemos una arquitectura cliente-servidor de una aplicación Android. Esta arquitectura se compone de tres capas [1]:

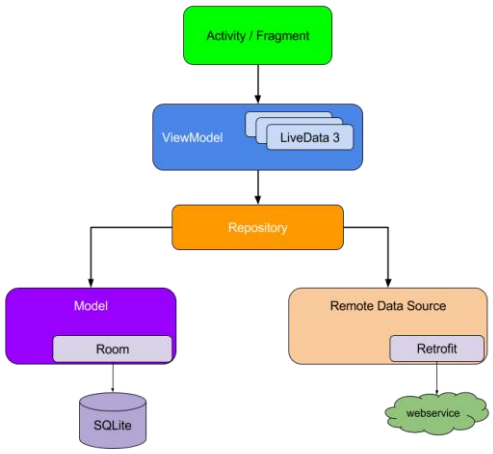

Ilustración 4: Arquitectura cliente-servidor

- View (Capa de interfaz de usuario): en la que hay clases que determinan la interfaz.
- ViewModel: aquí están las clases de tipo ViewModel que se explican más tarde en este apartado tres.
- Model (Capa de modelo): aquí se encuentran las clases que trabajan con las bases de datos y la API que proporciona el servidor. Aparte tenemos las clases entidades que van a servir tanto para las peticiones como para las respuestas del servidor. Finalmente tenemos la clase repositorio que decide que fuente de los datos de las propuestas utilizar.

Tal y como se muestra en la Ilustración 4: Arquitectura cliente-servidor la dependencia de las capas se sigue con las flechas. Es muy importante seguir esta regla ya que a la hora de hacer las pruebas sobre la aplicación no tendremos que resolver las dependencias.

3.2.1 Capa de interfaz de usuario (View)

En esta capa como se ha descrito antes están las clases pertenecientes a la interfaz del usuario, que bien podrían ser los fragmentos o las actividades. Cada una de estas clases maneja los componentes de vista y su comportamiento. Esta capa tiene la referencia hacía la capa de ViewModel de donde se obtienen los datos con ayuda del patrón de diseño Observer (Observador) que se describirá detalladamente en el apartado de la implementación.

3.2.2 Capa modelo-vista (ViewModel)

En esta capa se encuentran las clases que contienen los objetos LiveData. LiveData es un contenedor que almacena los datos y que funciona bajo los principios del patrón de diseño Observer (Observador). LiveData puede hacer principalmente varias cosas [9]:

- En él se puede almacenar cualquier objeto.
- Se puede suscribir a dicho objeto e ir obteniendo los objetos que se almacenan allí.

La principal funcionalidad de la clase ViewModel es que permite dejar a los objetos en estado vivo al girar la pantalla ya que al girarla no se reinicia la clase como bien se puede observar en la Ilustración 5: ViewModel. Esta capa tiene la referencia hacía la capa de Modelo.

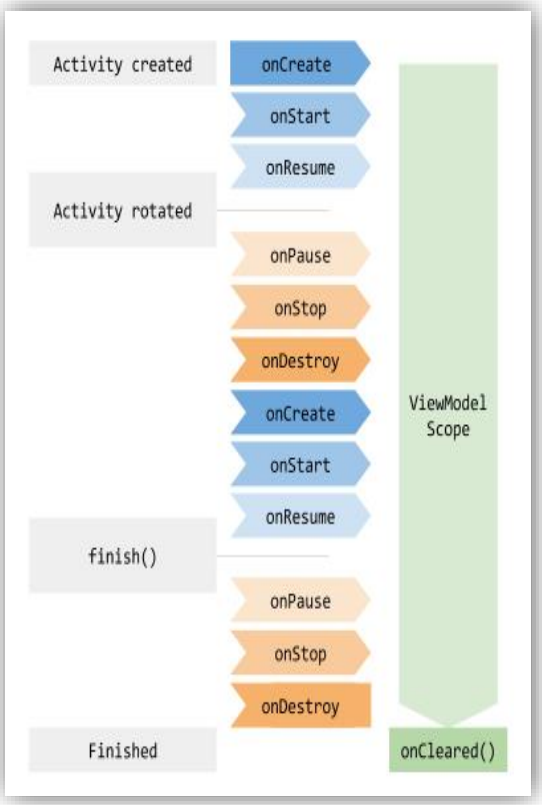

Ilustración 5: ViewModel

3.2.3 Capa Modelo (Model)

En esta capa se encuentran como se ha descrito anteriormente las clases de las entidades, las clases que sirven para trabajar con la API y las Bases de Datos es decir, con las fuentes de datos. Y también las clases de las librerías Paging, Retrofit, Room y OkHttp Persistence que nos proporciona Google y por lo tanto que se explican detalladamente en el apartado de implementación librerías. Por otra parte, está la clase Repositorio que sigue el patrón de diseño Repository Pattern (Repositorio) y que nos permite desacoplar el acceso a la base de datos y la API del servidor. Así mismo se podrían añadir en futuro más fuentes de acceso a datos. Esta estructura nos facilita la construcción de las pruebas unitarias sin necesidad de tener acceso a datos.

3.3 Aplicación modo offline

Muchas de las aplicaciones que trabajan con los servidores, es decir tienen la arquitectura cliente-servidor, hoy en día tienen la funcionalidad de poder trabajar localmente sin el acceso a internet. Esta aplicación también soporta el modo offline, es decir se trabaja con los datos que se obtienen al hacer las peticiones al servidor cuando hay acceso a internet. Por ello se utiliza un conjunto de componentes que proporciona Google que se explicarán detalladamente en su apartado correspondiente.

3.4 Problema que resuelve la arquitectura MVVM

Al desarrollar una aplicación para el dispositivo Android, la cual nos permite la orientación de pantalla, nos enfrentamos a un problema. Cuando se gira el dispositivo la actividad o el fragmento se rediseña y por lo tanto los componentes del diseño pierden sus referencias lo que puede producir un memory leak (fuga de memoria) y la aplicación producirá un error y se cerrará.

En la clase de la capa de vista hay una referencia hacía el componente del diseño que muestra la lista de los elementos obtenidos mediante la petición. Cuando realizamos la petición al servidor y al instante giramos la pantalla, se pierde la referencia hacía esta vista de la interfaz y si no lo tratamos provocaría una excepción. El Google nos propone una solución elegante, el componente ViewModel. Este componente se encuentra activo hasta que se destruye finalmente la actividad o el fragmento. Los pasos para conseguir la resolución del problema son:

- Se registra este componente en la clase de la capa de vista.
- Se subscribe a los cambios del objeto observable que se encuentra en ViewModel [13].

Al girar el dispositivo la clase es capaz de cancelar la suscripción y suscribirse de nuevo a los observables, lo que resuelve este problema es siempre tratar con los datos actualizados y no tener que volver a enviar peticiones cada vez que se gira la pantalla, sobre todo si la petición dura un tiempo prolongado lo que podría reflejarse notablemente en el rendimiento de la aplicación.

3.5 Diagramas de clases

A continuación, se representan algunos de los diagramas de clases que componen la aplicación. Cabe destacar que algunas de las clases se han obviado debido a que tienen la misma funcionalidad y estructura a los mostrados a continuación.

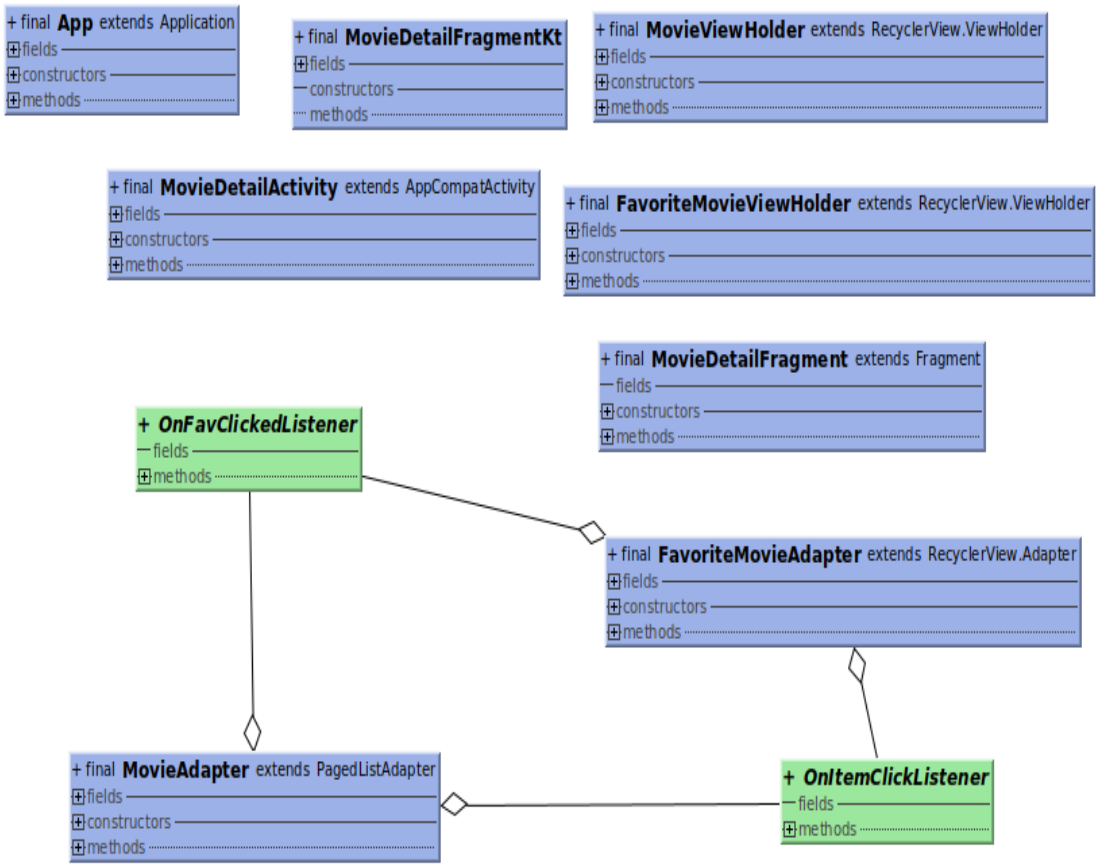

Ilustración 7: Diagrama de clases

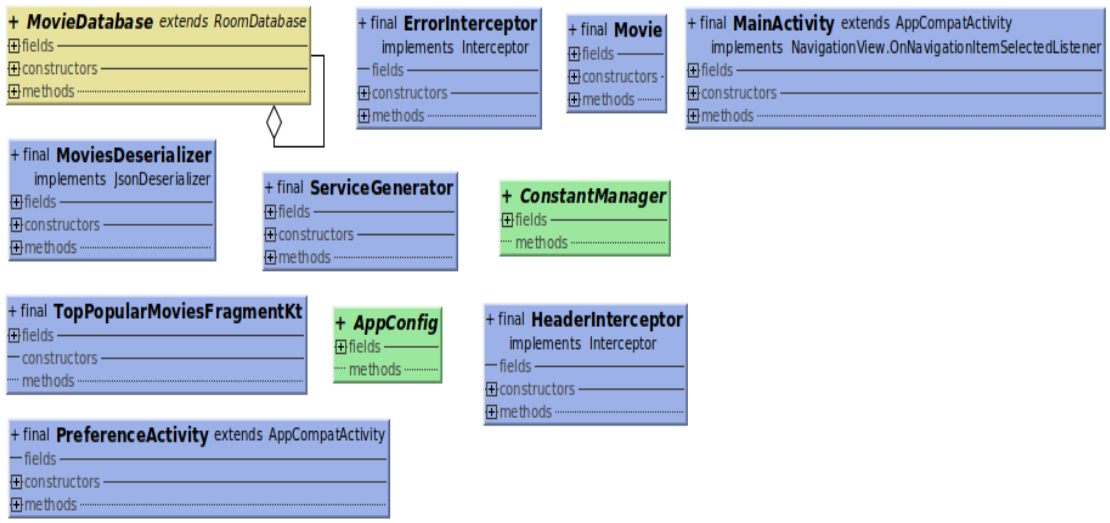

Ilustración 6: Diagrama de clases

Ilustración 8: Diagrama de clases

3.6 Especificación de requisitos

3.6.1 Requisitos no funcionales

Con la aplicación se pretende conseguir los siguientes requisitos no funcionales:

Eficiencia	• Tiempo de respuesta del servidor más baja posible. • Utilización de recursos en Android eficientemente. • La aplicación no debe ocupar mucho espacio de almacenamiento.
Usabilidad	• El usuario debe fácilmente aprender a utilizar la aplicación. • La aplicación debe permitir la protección a errores de usuario. • La interfaz debe ser lo más eficiente posible. • La aplicación debe permitir la operatividad.
Confiabilidad	• La aplicación debe estar disponible en todo el momento. • La aplicación debe recuperarse de los fallos.
Seguridad	• La aplicación no utilizará ninguna información del usuario o del dispositivo para ser enviada al internet. • En el caso de que exista la cuenta del usuario, la contraseña será encriptada y los datos serán visibles exclusivamente al usuario que posee los mismos.
Mantenimiento	• La aplicación será testeada y sujeta a los cambios.
Compatibilidad	• La aplicación será compatible a partir de la versión de API 19.

Tabla 2: Requisitos no funcionales

3.6.2 Requisitos funcionales

A continuación, se presenta el diagrama de casos de uso y se detallan algunos de los requisitos funcionales que el actor considera:

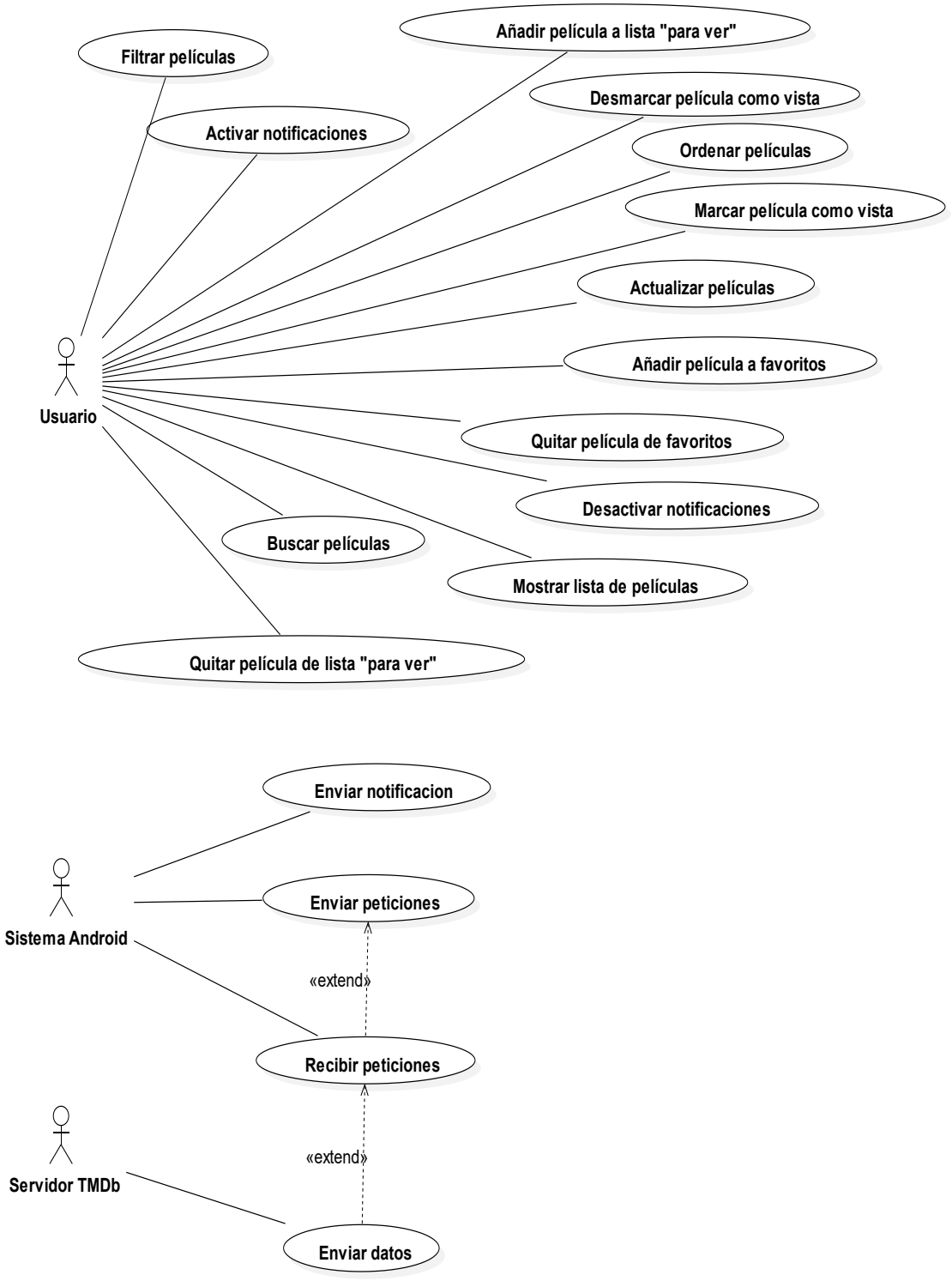

Ilustración 9: Diagrama de casos de uso

CU-01	Buscar películas.
Objetivos	Buscar películas por el título.
Descripción	El usuario podrá buscar películas en las listas de estas.
Precondición	Debe estar en la ventana de una de las listas y presionar sobre el icono de la lupa. Si se inicia la aplicación por primera vez, para poder buscar, el usuario debe estar conectado a internet. Si la aplicación ya ha sido iniciada anteriormente y se hayan obtenido las películas del servidor, usuario podrá buscar en la base de datos local incluso sin tener el acceso a la conexión de internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3) Presionar sobre el icono de lupa. 4) Introducir el texto.
Postcondición	Las películas han sido encontradas. No hay películas con el mismo título.
Excepciones	Paso 4: Si el usuario no está conectado a internet se le avisa de ello.
Comentarios	Ninguno

Tabla 3: CU-01 (Buscar películas)

CU-02	Filtrar películas.
Objetivos	Filtrar la lista de las películas por género, año o ambas.
Descripción	El usuario podrá filtrar las películas en las listas de estas.
Precondición	Debe estar en la ventana de una de las listas y presionar sobre el icono del filtro. Si se inicia la aplicación por primera vez, para poder filtrar el usuario debe estar conectado a internet. Si la aplicación ya ha sido iniciada anteriormente y se hayan obtenido las películas del servidor, el usuario podrá filtrar en la base de datos local incluso sin tener el acceso a la conexión de internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3) Presionar sobre el icono de filtro. 4) Seleccionar los filtros adecuados. 5) Aplicar filtros.
Postcondición	Las películas han sido filtradas.
Excepciones	Paso 5: Si el usuario no está conectado a internet se le avisa de ello.
Comentarios	Ninguno

Tabla 4: CU-02 (Filtrar películas)

CU-03	Añadir película a favoritos.
Objetivos	Añadir la película a la lista de favoritos.
Descripción	El usuario podrá añadir la película a la lista de favoritos.
Precondición	La película no debe haber sido añadida a la lista anteriormente. Debe estar en la ventana de una de las listas, o en los detalles de la película y presionar sobre el elemento en forma de estrella Si se inicia la aplicación por primera vez, para poder añadir alguna película el usuario debe estar conectado a internet. Si la aplicación ha sido iniciada anteriormente y se han obtenido las películas del servidor, el usuario podrá añadir la película favorita a la base de datos local incluso sin tener el acceso a la conexión de internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3.1) Presionar sobre el elemento estrella que no haya sido marcado anteriormente. 3.2) Presionar sobre la película. 4) Presionar sobre el elemento estrella que no haya sido marcado anteriormente.
Postcondición	La película ha sido añadida a los favoritos. Aviso de que la película haya sido añadida.
Excepciones	Ninguno
Comentarios	Ninguno

Tabla 5: CU-03 (Añadir película a favoritos)

<table>
<tr><td>CU-04</td><td>Quitar película de favoritos.</td></tr>
<tr><td>Objetivos</td><td>Quitar la película de la lista de favoritos.</td></tr>
<tr><td>Descripción</td><td>El usuario podrá quitar la película de la lista de favoritos.</td></tr>
<tr><td>Precondición</td><td>Para poder quitar alguna película el usuario debe tener películas añadidas en la lista de los favoritos.

Debe estar en la ventana de una de las listas o en los detalles de la película y presionar sobre el elemento en forma de estrella.</td></tr>
<tr><td>Secuencia de pasos</td><td>1) Iniciar aplicación.

2) Posicionarse sobre la ventana en la que hay una lista de películas.

<table><tr><td>3.1) Presionar sobre el elemento estrella que haya sido marcado anteriormente.</td><td>3.2) Presionar sobre la película

4) Presionar sobre el elemento estrella que haya sido marcado anteriormente.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>La película ha sido quitada de la lista de las películas favoritas.

Aviso de que la película ha sido quitada.

Deshacer la acción en tiempo de cinco segundos.</td></tr>
<tr><td>Excepciones</td><td>Ninguno</td></tr>
<tr><td>Comentarios</td><td>Ninguno</td></tr>
</table>

Tabla 6: CU-04 (Quitar película de favoritos)

CU-05	Activar notificaciones.
Objetivos	Activar las notificaciones de usuario.
Descripción	El usuario podrá activar las notificaciones de las películas nuevas y de las que quiere ver.
Precondición	La opción debe de estar desactivada. Debe posicionarse sobre la ventana de opciones y marcar la opción de activar notificaciones.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana de las opciones. 3) Activar la opción de notificaciones.
Postcondición	Aviso de que la opción ha sido activada.
Excepciones	Ninguno
Comentarios	Ninguno

Tabla 7: CU-05 (Activar notificaciones)

CU-06	Añadir película a lista “para ver”.
Objetivos	Añadir la película a la lista “para ver”.
Descripción	El usuario podrá añadir la película a la lista “para ver”.
Precondición	La película no debe de haber sido añadida con anterioridad a la lista. Debe estar en la ventana de una de las listas o en los detalles de la película y presionar sobre el elemento desactivado en forma del reloj. Si se inicia la aplicación por primera vez, para poder añadir alguna película el usuario debe estar conectado a internet. Cuando la aplicación ya haya sido iniciada anteriormente y se hayan obtenido las películas del servidor, el usuario podrá añadir la película a la base de datos local incluso sin tener el acceso a la conexión de internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3.1) Presionar sobre el elemento en forma del reloj que no haya sido marcado anteriormente. / 3.2) Presionar sobre la película 4) Presionar sobre el elemento en forma del reloj que no haya sido marcado anteriormente.
Postcondición	Aviso de que la película ha sido añadida.
Excepciones	Ninguno
Comentarios	Ninguno

Tabla 8: CU-06 (Añadir película a lista “para ver”)

<table>
<tr><td>CU-07</td><td>Quitar película de lista “para ver”.</td></tr>
<tr><td>Objetivos</td><td>Quitar la película de la lista de “para ver”.</td></tr>
<tr><td>Descripción</td><td>El usuario podrá quitar la película de la lista “para ver”.</td></tr>
<tr><td>Precondición</td><td>La película debe de haber sido añadida con anterioridad a la lista.

Debe estar en la ventana de una de las listas o de los detalles de la película y presionar sobre el elemento activado en forma de reloj.</td></tr>
<tr><td>Secuencia de pasos</td><td>1) Iniciar aplicación.

2) Posicionarse sobre la ventana en la que hay una lista de películas.

<table><tr><td>3.1) Presionar sobre el elemento en forma del reloj que haya sido marcado anteriormente.</td><td>3.2) Presionar sobre la película

4) Presionar sobre el elemento en forma del reloj que haya sido marcado anteriormente.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>La película ha sido quitada de la lista.

Aviso de que la película ha sido quitada.

Deshacer la acción en tiempo de cinco segundos.</td></tr>
<tr><td>Excepciones</td><td>Ninguno</td></tr>
<tr><td>Comentarios</td><td>Ninguno</td></tr>
</table>

Tabla 9: CU-07 (Quitar película de lista “para ver”)

CU-08	Ordenar películas.
Objetivos	Ordenar las películas.
Descripción	El usuario podrá ordenar las películas según la clasificación, popularidad, fecha de lanzamiento o el título.
Precondición	Debe estar en la ventana de una de las listas y presionar sobre el icono de ordenar. Si se inicia la aplicación por primera vez, para poder ordenar el usuario debe estar conectado a internet. Si la aplicación ya ha sido iniciada anteriormente y se han obtenido las películas del servidor, el usuario podrá ordenar en la base de datos local incluso sin tener el acceso a la conexión de internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3) Presionar sobre el icono de ordenar. 4) Seleccionar la opción deseada.
Postcondición	Las películas han sido ordenadas.
Excepciones	Paso 5: Si el usuario no está conectado a internet se le avisa de ello.
Comentarios	Ninguno

Tabla 10: CU-08 (Ordenar películas)

<table>
<tr><td>CU-09</td><td>Marcar película como vista.</td></tr>
<tr><td>Objetivos</td><td>Marcar película como vista.</td></tr>
<tr><td>Descripción</td><td>El usuario podrá marcar la película como vista.</td></tr>
<tr><td>Precondición</td><td>Debe estar en la ventana de una de las listas o en los detalles de la película y presionar sobre el elemento desactivado en forma de ojo.

Si se inicia la aplicación por primera vez, para poder marcar alguna película como vista el usuario debe estar conectado a internet.

Si la aplicación ya ha sido iniciada anteriormente y se han obtenido las películas del servidor, usuario podrá marcar la película como vista en la base de datos local incluso sin tener el acceso a la conexión de internet.</td></tr>
<tr><td>Secuencia de pasos</td><td>1) Iniciar aplicación.

2) Posicionarse sobre la ventana en la que hay una lista de películas.

<table><tr><td>3.1) Presionar sobre el elemento en forma del ojo que no haya sido marcado anteriormente.</td><td>3.2) Presionar sobre la película

4) Presionar sobre el elemento en forma del ojo que no haya sido marcado anteriormente.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>La película ha sido marcada como vista.

Aviso de que la película ha sido marcada como vista.</td></tr>
<tr><td>Excepciones</td><td>Ninguno</td></tr>
<tr><td>Comentarios</td><td>Ninguno</td></tr>
</table>

Tabla 11: CU-09 (Marcar película como vista)

CU-10	Desactivar notificaciones.
Objetivos	Desactivar notificaciones de usuario.
Descripción	El usuario podrá desactivar las notificaciones de las películas nuevas y de las que quiere ver.
Precondición	La opción debe de estar activada. Debe posicionarse sobre la ventana de opciones y desmarcar la opción de activar notificaciones.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana de las opciones. 3) Desactivar la opción de notificaciones.
Postcondición	Aviso de que la opción ha sido desactivada.
Excepciones	Ninguno
Comentarios	Ninguno

Tabla 12: CU-10 (Desactivar notificaciones)

CU-11	Mostrar lista de películas.
Objetivos	Mostrar la lista de las películas.
Descripción	El usuario podrá mostrar la lista de las películas.
Precondición	Debe estar en la ventana de una de las listas. Si se inicia la aplicación por primera vez, para poder ver las películas en la lista el usuario debe estar conectado a internet.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas.
Postcondición	Las películas se muestran en la lista. Las películas no se muestran en la lista.
Excepciones	Paso 2: Si el usuario no está conectado a internet se le avisa de ello.
Comentarios	Ninguno

Tabla 13: CU-11 (Mostrar lista de películas)

CU-12	Desmarcar película como “vista”.
Objetivos	Desmarcar película como “vista”.
Descripción	El usuario podrá desmarcar la película como “vista”.
Precondición	La película debe de haber sido marcada anteriormente como vista. Debe estar en la ventana de una de las listas o en los detalles de la película y presionar sobre el elemento activado en forma de ojo.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3.1) Presionar sobre el elemento en forma del ojo que haya sido marcado anteriormente. 3.2) Presionar sobre la película 4) Presionar sobre el elemento en forma del ojo que haya sido marcado anteriormente.
Postcondición	La película ha sido desmarcada como “vista”. Aviso de que la película ha sido desmarcada como “vista”.
Excepciones	Ninguno
Comentarios	Ninguno

Tabla 14: CU-12 (Desmarcar película como vista)

CU-13	Actualizar películas.
Objetivos	Actualizar las películas de una lista.
Descripción	El usuario podrá actualizar las películas de una lista.
Precondición	Debe estar conectado a internet. Debe estar en la ventana de una de las listas, al principio de la lista y hacer el *swipe* con el dedo hacia abajo.
Secuencia de pasos	1) Iniciar aplicación. 2) Posicionarse sobre la ventana en la que hay una lista de películas. 3) Posicionarse al principio de la lista. 4) Hacer *swipe* con el dedo hacía abajo.
Postcondición	La lista ha sido actualizada. La lista no ha sido actualizada.
Excepciones	Paso 4: Si el usuario no está conectado a internet se le avisa de ello.
Comentarios	Ninguno

Tabla 15: CU-13 (Actualizar películas)

3.7 Diseño de la aplicación

A continuación, se representan los bocetos de la aplicación que se relacionan con los requisitos funcionales descritos en el apartado anterior. Así como se describen detalladamente cada una de las vistas de la aplicación en la que se utilizarán los distintos componentes del *material design*.

Cabe destacar que la actual versión implementa la funcionalidad del giro de la pantalla y por lo tanto el usuario podrá interactuar con las vistas en modo apaisado.

Todos los bocetos son creados con la ayuda de la herramienta online Moqups [10].

3.7.1 Navegación principal

En esta maqueta se representa una navegación lateral desplegable con el título de la aplicación, el logotipo y los menús correspondientes a la dicha navegación.

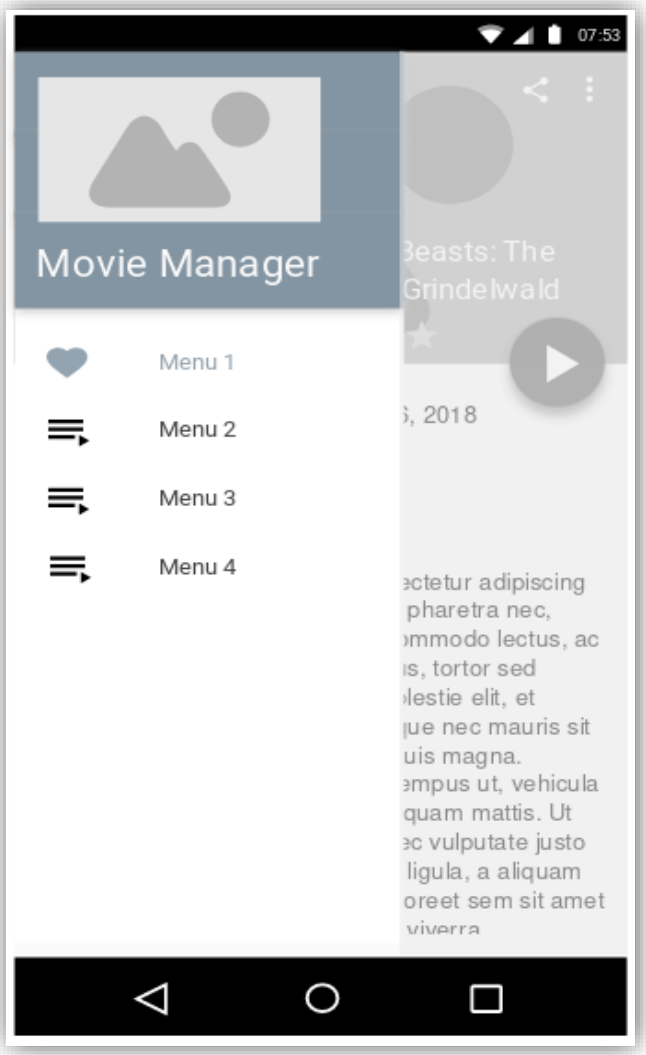

Ilustración 10: Navegación principal

3.7.2 Lista de los elementos

En esta maqueta se representan distintos componentes como la barra de herramientas, las opciones que se encuentran en la barra de herramientas y la lista de los elementos desplazables verticalmente. Las opciones son: desplegar la navegación lateral, buscar los elementos, mostrar el submenú con las opciones de filtrar y ordenar los elementos.

Cada uno de los elementos tiene una imagen de portada, el título, la fecha, la descripción, la calificación en forma de estrellas y las opciones de guardar / eliminar el elemento de la lista de favoritos, guardar / eliminar el elemento de la lista de "para ver" y por último la opción de marcar / desmarcar la película como vista.

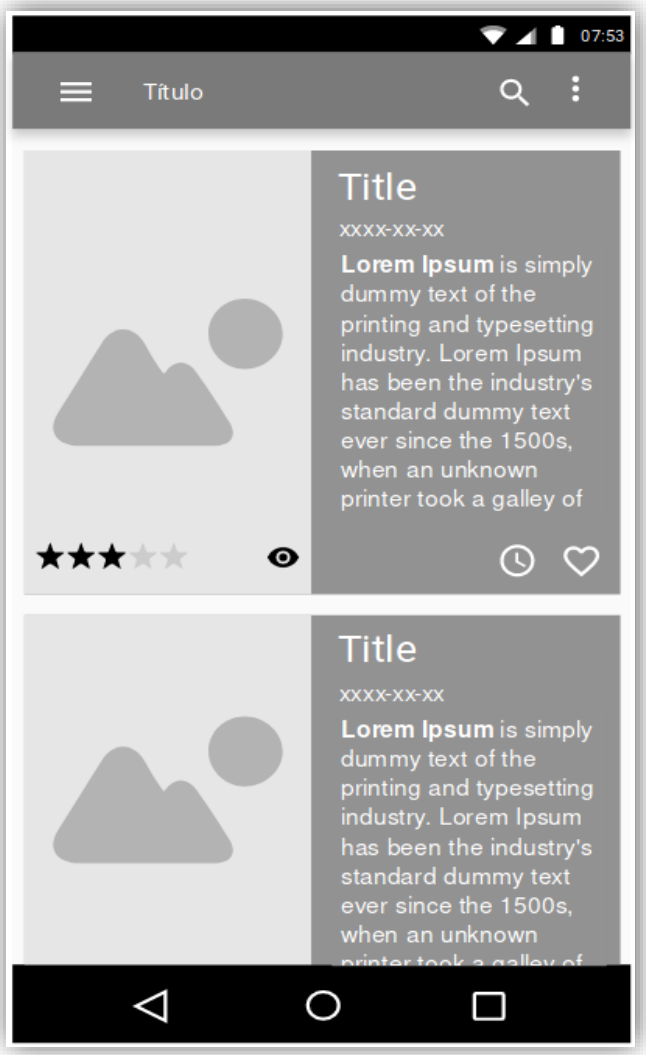

Ilustración 11: Lista de los elementos

3.7.3 Detalles del elemento

En esta maqueta se representan los detalles de un elemento de la lista de elementos. Se muestra la barra de herramientas desplegada con una imagen en el fondo y las opciones de volver a la pantalla anterior, compartir el elemento y mostrar el submenú. También se muestra la imagen de la portada, el título, la fecha la descripción y las opciones de reproducir el *trailer*, guardar / eliminar el elemento de la lista de favoritos, guardar / eliminar el elemento de la lista de "para ver" y por último la opción de marcar / desmarcar la película como vista.

Esta vista es desplazable verticalmente debido a que puede haber elementos con descripciones extensas y que dependiendo de las resoluciones de los dispositivos pueden sobrepasar el área de la pantalla y por lo tanto no ser visibles. Cabe destacar también que la barra de herramientas al ser desplazada se contrae.

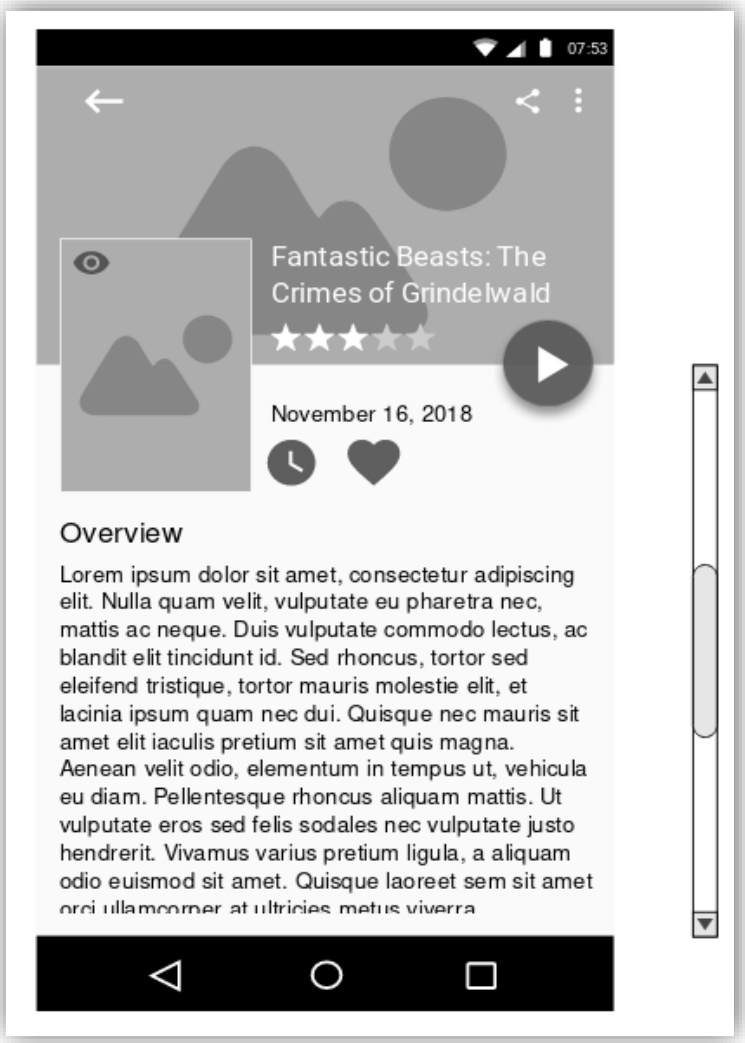

Ilustración 12: Detalles del elemento

3.7.4 Opciones

En esta maqueta se muestran las configuraciones con las que puede interactuar el usuario. Las configuraciones pueden ser de varios tipos:

- De marcar / desmarcar.
- Seleccionables.

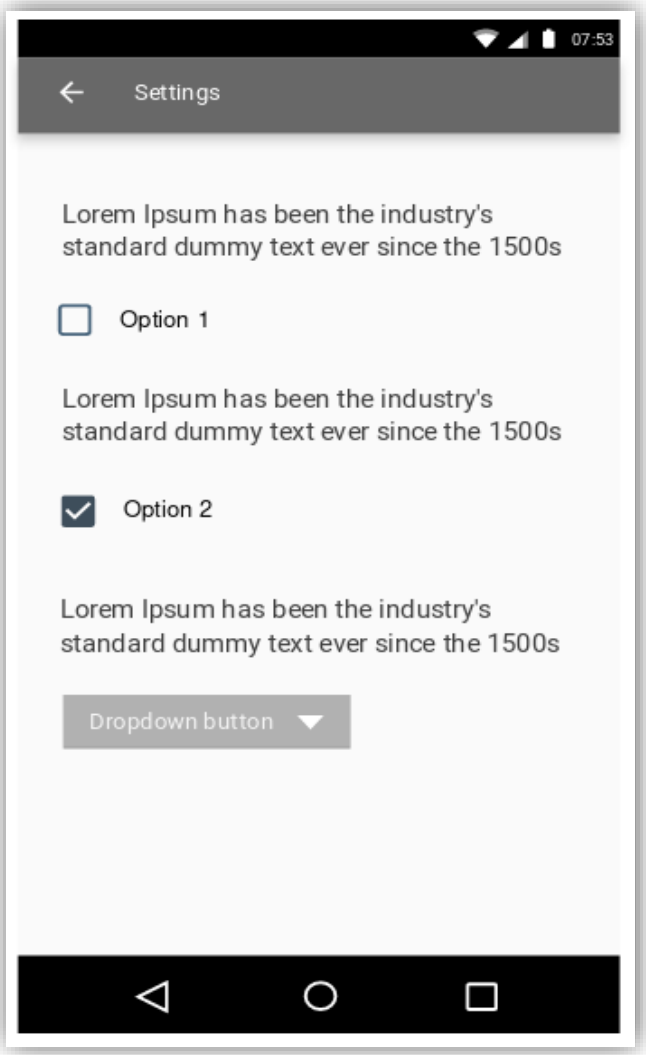

Ilustración 13: Opciones

3.7.5 Modo apaisado

En la maqueta actual se representa la vista cuando la aplicación se encuentra en modo apaisado, es decir cuando se gira el dispositivo. Se representa en una misma pantalla por un lado la lista de los diferentes elementos y por otro el detalle de cada elemento seleccionado.

Cabe destacar que la primera vez que se acceda a una de las listas siempre aparece el primer elemento seleccionado.

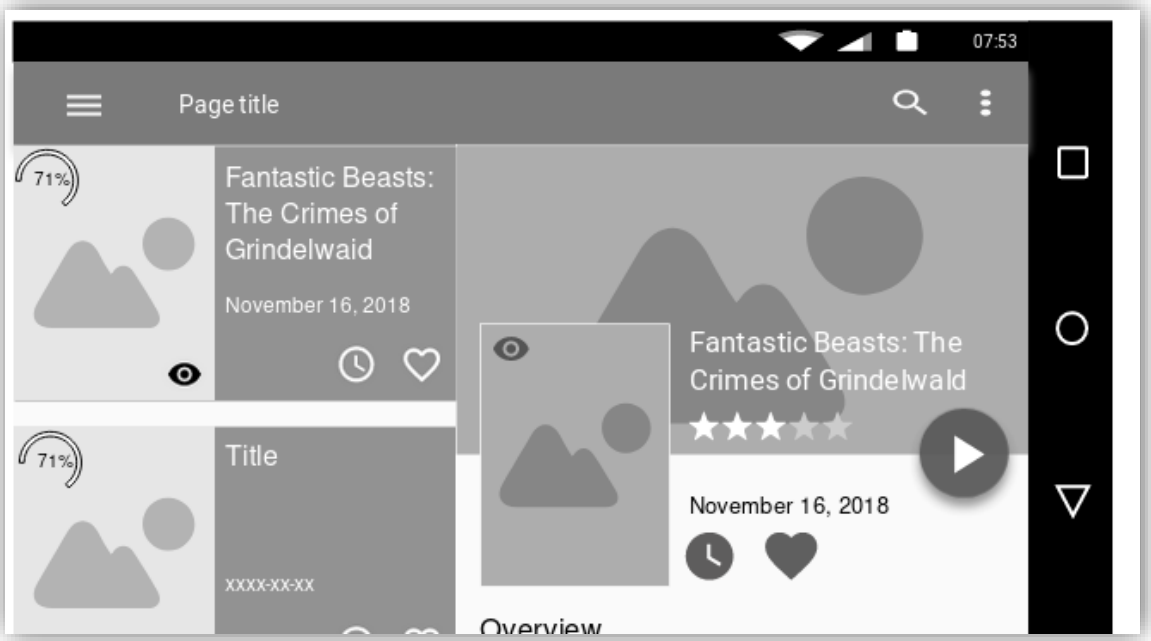

Ilustración 14: Modo apaisado

4. Tecnologías y herramientas

En el actual apartado se describen las tecnologías utilizadas a lo largo del proyecto. Dichas tecnologías se descomponen en lenguaje de programación, herramientas y entornos de desarrollo.

4.1 Lenguaje de programación

Ilustración 15: Kotlin

A lo largo del proyecto se utiliza el lenguaje de programación Kotlin [8]. Kotlin es un lenguaje de programación desarrollado por la compañía JetBrains cuyo lenguaje se puede utilizar para crear aplicaciones móviles, programas de escritorio o web.

Kotlin se ejecuta sobre la máquina virtual de java (JVM) y al compilarse, se compila en el código de bytes. Dicho esto, se puede decir que como en el caso de Java, podemos ejecutar la aplicación programada en Kotlin donde está instalada la máquina virtual de java. Por lo tanto, la gama de plataformas para las que se puede crear aplicaciones en Kotlin es extremadamente amplia: Windows, Linux, Mac OS, iOS, Android.

Kotlin ha experimentado la influencia de muchos lenguajes: Java, Scala, Groovy, C#, JavaScript, Swift y permite escribir programas tanto en estilo orientado a objetos como funcional. Tiene una sintaxis clara y comprensible y es bastante fácil de aprender.

4.2 Herramientas

Ilustración 17: StarUML

StarUML es una herramienta UML que pertenece a la compañía MKLab. La herramienta es utilizada en el proyecto para representar el diagrama de los distintos casos de uso utilizados en la aplicación, como bien se puede ver en la ilustración 9 generada por dicha herramienta.

Ilustración 18: Postman

Postman es una herramienta de cliente de API que ayuda a probar los diferentes API's. Nos permite probar la misma solicitud en diferentes entornos con variables específicas del entorno. En el proyecto actual es utilizado para ver las respuestas en el formato JSON enviadas por el servidor.

Ilustración 19: Moqups

Moqups es una herramienta online que permite crear maquetas. En este proyecto se ha utilizado para crear las maquetas de aplicación para el dispositivo Android.

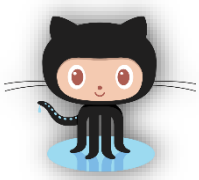

Ilustración 20: Github

Github es una plataforma de trabajo colaborativo y control de versiones basado en web para los desarrolladores de software. En este proyecto se ha utilizado para conseguir el desarrollo eficiente y poder, en cualquier momento, volver al estado determinado del desarrollo.

Ilustración 22: DB browser for SQLite

DB browser for SQLite como bien su título lo indica es una herramienta para la gestión de la base de datos SQLite. Por lo tanto, como bien se puede deducir se ha utilizado en este proyecto para ver el comportamiento correcto a la hora de crear las tablas y guardar los registros en la base de datos.

4.3 Entorno de desarrollo

Ilustración 23: Android Studio

Android Studio es utilizado en este proyecto como el entorno de desarrollo para desarrollar la aplicación. Generalmente se utiliza para el desarrollo de aplicaciones para los distintos dispositivos que llevan el sistema operativo Android. Debido a su sencillez y variedad de herramientas que proporciona permite un desarrollo cómodo y bastante rápido. Entre muchas herramientas que proporciona el Android Studio se pueden destacar los siguientes:

- **AVD Manager:** permite crear un dispositivo virtual para poder testear la aplicación sobre la que estamos trabajando.

- **Perfiladores en tiempo real:** las herramientas de perfilado integradas proporcionan estadísticas en tiempo real para la CPU, la memoria y la actividad de red de la aplicación.

- **Versión de control:** esta herramienta permite interactuar fácilmente con el sistema de gestión de versiones. En este caso se ha utilizado el Git.

- **Logcat:** es una herramienta de línea de comandos que representa los mensajes de registro enviados utilizando varios métodos. También permite mostrar los mensajes especificados por el desarrollador con la ayuda de la clase Log.

- **SDK Manager:** en general, se trata de un conjunto de diversas herramientas de desarrollo que permiten a los programadores de software crear aplicaciones.

5. Implementación

En este apartado se describen los puntos importantes de la implementación de la aplicación. Por una parte, se describe la preparación del entorno y librerías utilizadas para el desarrollo. Por otra parte, se explica la implementación por pasos de algunas partes claves del desarrollo, los puntos de atasco y las soluciones de estos, así como también la generación de la aplicación.

5.1 Preparación del entorno de desarrollo

5.1.1 Instalación de JDK

Para instalar el entorno de desarrollo Android Studio fue necesario instalar antes la última versión de Java Development Kit. Para ello se ha accedido a la página oficial de Oracle y se ha descargado la última versión disponible en el momento de redactar el documento.

5.1.2 Instalación de Android Studio

Una vez se ha instalado el JDK se ha procedido a la instalación de la última versión disponible en el momento de redactar el documento de Android Studio. Se han configurado algunos aspectos de la interfaz para proporcionar la comodidad a la hora del desarrollo y finalmente se ha creado un proyecto nuevo y se ha escogido la versión mínima con la API 19.

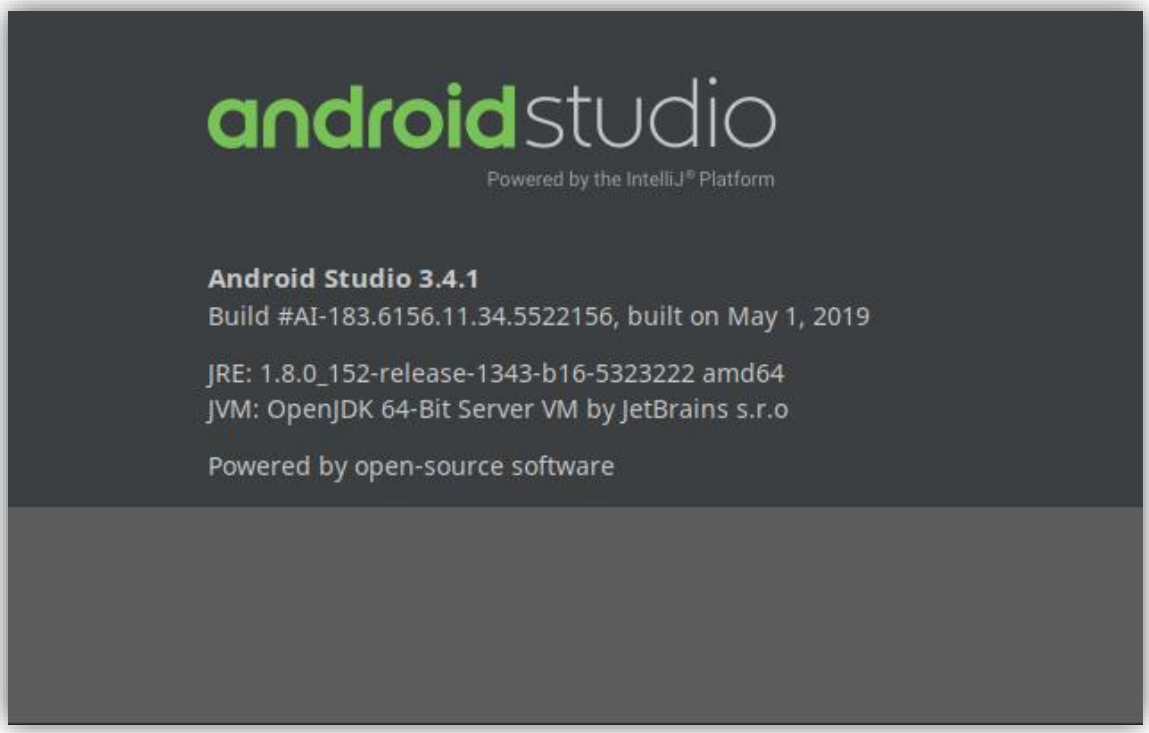

Ilustración 24: Android Studio versión

5.1.3 Configuración de AVD

Después del todo el procedimiento de la instalación, la configuración del entorno y la creación del proyecto se ha procedido a escoger un dispositivo virtual de Android para poder hacer las pruebas, depurar la aplicación y ver los resultados en general.

Ilustración 25: AVD

5.2 Librerías utilizadas

- **OkHttp:** es una librería que permite enviar y recibir solicitudes de red basados en el protocolo HTTP. También se puede destacar que es una librería subyacente para la librería Retrofit que proporciona seguridad de tipos para consumir API's basadas en REST y es la que se explica a continuación.

- **Retrofit:** es una librería que permite crear un cliente REST completo que puede realizar las operaciones de varios tipos como:
  - POST
  - GET
  - PUT
  - DELETE

  Las anotaciones son utilizadas para indicar los tipos y otros aspectos de la solicitud como por ejemplo podrían ser las cabeceras.

- **RxJava:** es una librería para la programación reactiva, en otras palabras, esta librería permite manejar eventos asíncronos. En el proyecto es utilizado para este mismo propósito.

- **Room:** es una librería que permite trabajar con las bases de datos SQLite. Básicamente la librería se divide en tres componentes:
  - Database: contiene el soporte de la base de datos y sirve como el punto de acceso principal para las conexiones a los datos relacionales persistentes de la aplicación.
  - Entity: las entidades representan las tablas dentro de la base de datos.
  - DAO: contiene métodos utilizados para acceder a la base de datos.

- **Paging:** es una librería que permite trabajar con las listas de forma eficiente. Es utilizada sobre todo con las aplicaciones que utilizan la arquitectura cliente-servidor y en la parte del servidor contienen las listas con elementos de gran cantidad que se necesitan traer y representar en una o varias de las listas locales de la aplicación. En este proyecto esta librería es utilizada para representar la lista de las películas obtenidas del servidor [11].

- **Glide:** es una librería utilizada para trabajar con las imágenes. El principal motivo de utilización de esta librería es; teniendo la *url* de la imagen que se encuentra en el servidor, hacer la petición al servidor y poner en cache la respuesta para que cada vez que accedamos a uno de los elementos de la vista de Android que contenga dicha imagen con la misma *url* no se nos baje cada vez.

- **Timber:** es una librería muy útil y es utilizada en este proyecto para poder trabajar con los registros de la aplicación. Añade un toque de simplicidad a la hora de depurar la aplicación.

- **Lottie:** es una librería que permite trabajar con las animaciones y sus comportamientos.

5.3 Implementación por pasos

Lo primero que se ha hecho es crear una clase App la que extiende de la clase predefinida en Android Application. Esta clase sirve para poder inicializar los componentes y las librerías que van a ser necesarios en todo el momento del desarrollo del proyecto [3]. En este caso se definen las preferencias compartidas, la librería Timber, que sirve para depurar fácilmente la aplicación y el contexto de la aplicación general para poder referirse a él en todo momento.

```
/**
 * Best practice
 */
class App : Application() {

    companion object {
        private lateinit var sApplication: Application
        private lateinit var mSharedPreferences: SharedPreferences

        /**
         * Static method which return Application
         */
        @JvmStatic
        fun getApplication(): Application {
            return sApplication
        }

        @JvmStatic
        fun getSharedPreferences() : SharedPreferences {
            return mSharedPreferences
        }
    }

    override fun onCreate() {
        super.onCreate()
        sApplication = this

        // Timber
        if (BuildConfig.DEBUG) {
            Timber.plant(Timber.DebugTree())
        }

        mSharedPreferences = PreferenceManager.getDefaultSharedPreferences( context: this)
    }
}
```

Ilustración 26: clase App

Lo siguiente que se ha hecho ha sido definir una clase principal MainActivity, la que hereda el comportamiento de una Actividad, ya que se extiende de una clase de tipo Activity [6]. En dicha clase se configura el comportamiento de la navegación lateral desplegable que actúa con los diferentes fragmentos de la aplicación y la barra de herramientas. Por defecto, en nuestro caso, esta clase se lanza al ejecutar una aplicación después de la clase definida anteriormente App ya que en el fichero de manifiesto dicha clase se define como la clase ejecutable. Dicha clase nos lanza por defecto un fragmento de tipo TopRatedMoviesFragment.

TopRatedMoviesFragment implementa el código necesario para configurar el menú que aparece en la barra de herramientas y las acciones resultantes de pulsar en algún elemento del menú. En dicho fragmento también se define una lista para presentar los elementos. Debido a que se ha utilizado el patrón arquitectónico MVVM, el fragmento contiene un observador que detecta cuando llegan nuevos elementos de las fuentes ya sea base de datos local o del servidor e inmediatamente los representa.

```
class TopRatedMoviesFragment : Fragment() {
    private lateinit var topRatedMovieViewModel: TopRatedMovieViewModel
    private lateinit var binding: FragmentTopRatedMoviesBinding

    private val movieAdapter: MovieAdapter by lazy {
        MovieAdapter()
    }

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

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentTopRatedMoviesBinding.inflate(inflater, container,  attachToRoot: false)
        return binding.root
    }

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

        recycler_view.setHasFixedSize(true)

        recycler_view.layoutManager = GridLayoutManager(
            activity, grid_layout: 1
        )

        topRatedMovieViewModel = ViewModelProviders.of( fragment: this).get(TopRatedMovieViewModel::class.java)
        binding.viewModel = topRatedMovieViewModel

        recycler_view.adapter = movieAdapter


        topRatedMovieViewModel.getPagedListLiveData().observe( owner: this,
            Observer<PagedList<Movie>> {  it: PagedList<Movie>?
                movieAdapter.submitList(it)
            })

    }

    override fun onCreateOptionsMenu(menu: Menu?, inflater: MenuInflater?) {
        inflater?.inflate(R.menu.menu_top_rated_movies,menu)
        super.onCreateOptionsMenu(menu, inflater)
    }
}
```

Ilustración 27: TopRatedMoviesFragment

Por otra parte, tenemos la clase ViewModel del fragmento descrito anteriormente, dicha clase contiene el objeto de tipo LiveData que es observado por el fragmento TopRatedMoviesFragment y la clase MovieRepository que va a ser explicada a continuación. Antes de que se destruya el fragmento se llama al método onCleared que a su vez llama el método disposeCompositeDisposable del objeto repository que sirve para cancelar los hilos asíncronos que se encuentran en activo y los que pueden provocar una excepción si no los cancelamos antes de salir del fragmento.

Ilustración 28: LiveData

La clase MovieRepository contiene una instancia de la clase MovieDatabase que sirve para poder llamar a los métodos definidos en el DAO ya sea para hacer las consultas o modificar la base de datos. Por otra parte, se crea también un objeto webservice que sirve para poder enviar los datos hacia el servidor o traerlos mediante los métodos definidos en la clase RestService. En el repositorio también se define una "*thread pool*", a la que cada vez que se hace una llamada, por ejemplo, al servidor mediante un hilo se añade dicho hilo a la piscina. En esta clase también se configura uno de los componentes de la librería Paging [12].

```
object MovieRepository {
    private val database = MovieDatabase.getInstance(App.getApplication())
    private var movieDao = database.movieDao()

    private val webservice = ServiceGenerator.createService(RestService::class.java)
    private val compositeDisposable: CompositeDisposable = CompositeDisposable()
    private val mutableLiveData = MutableLiveData<ArrayList<Movie>>()

    private val pagedListLiveData: LiveData<PagedList<Movie>> by lazy {
        val dataSourceFactory = movieDao.getAllMovies()
        val config = PagedList.Config.Builder()
            .setPageSize(ConstantManager.LOADING_PAGE_SIZE)
            .build()
        LivePagedListBuilder(dataSourceFactory, config)
            .setBoundaryCallback(BoundaryCallback( query: "",webservice, database))
            .build() ^lazy
    }

    fun disposeCompositeDisposable() {
        compositeDisposable.dispose()
    }

    fun loadMovies(): LiveData<ArrayList<Movie>>{
        return mutableLiveData
    }

    fun getMovies(): LiveData<PagedList<Movie>> {
        return pagedListLiveData
    }
```

Ilustración 29: MovieRepository

La clase RestService define los métodos necesarios para hacer llamadas al servidor remoto utilizando las anotaciones de tipo GET. Dichos métodos envuelven la respuesta en el objeto Single que nos proporciona la librería JavaRx [14].

```
interface RestService {

    @GET( value: "configuration")
    fun getConfiguration(
    ): Single<ConfigurationRes>

    @GET( value: "movie/top_rated")
    fun getTopRatedMovies(
        @Query(LANGUAGE_REQUEST_PARAM) language: String,
        @Query(PAGE_REQUEST_PARAM) page: Int
    ): Single<ArrayList<Movie>>


    @GET( value: "discover/movie")
    fun discoverMovie(
    ): Single<ArrayList<Movie>>
}
```

Ilustración 30: RestService

Para utilizar la base de datos se han definido los tres elementos necesarios: entidad, DAO y la clase MovieDatabase que contiene un método para instanciar el objeto.

En la Ilustración 31: Entidad se representa una parte de la clase Entidad en la cual se especifican las propiedades que van a ser utilizadas para la base de datos como por ejemplo los nombres de los campos, el nombre de la tabla y la anotación de la clave primaria. Así mismo esta entidad es utilizada también para poder convertir el elemento json recibido por parte del servidor en un POJO.

```
@Entity(tableName = "movie")
data class Movie(

    @PrimaryKey
    @ColumnInfo(name = "id")
    @SerializedName(value = "id")
    val id: Int,

    @ColumnInfo(name = "vote_count")
    @SerializedName(value = "vote_count")
    val voteCount: Int,

    @ColumnInfo(name = "video")
    @SerializedName(value = "video")
    val video: Boolean,

    @ColumnInfo(name = "vote_average")
    @SerializedName(value = "vote_average")
    val voteAverage: Float,

    @ColumnInfo(name = "title")
    @SerializedName(value = "title")
    val title: String,

    @ColumnInfo(name = "popularity")
    @SerializedName(value = "popularity")
    val popularity: Float,

    @ColumnInfo(name = "poster_path")
    @SerializedName(value = "poster_path")
    val posterPath: String,

    @ColumnInfo(name = "original_language")
    @SerializedName(value = "original_language")
    val originalLanguage: String,
```

Ilustración 31: Entidad

En la Ilustración 32: DAO se representan los métodos que interactúan con la base de datos local. Como se puede ver es una simple interfaz que tiene una anotación DAO y los métodos con las consultas. La propiedad onConflict ayuda a resolver los problemas de los objetos idénticos.

```kotlin
@Dao
interface MovieDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertAll(movies: List<Movie>)

    @Update(onConflict = OnConflictStrategy.REPLACE)
    fun udpateMovies(movies: List<Movie>)

    @Query( value: "DELETE FROM movie")
    fun deleteAllMovies()

    @Query( value: "SELECT * FROM movie ORDER BY vote_average DESC")
    fun getAllMovies(): DataSource.Factory<Int, Movie>
}
```

Ilustración 32: DAO

En la siguiente ilustración se representa la especificación de la base de datos local. Se describen las entidades que van a ser utilizadas en dicho modelo, la versión, el nombre y DAO. Cabe destacar que el método getInstance, siguiendo la norma de las buenas prácticas debe ser Synchronized y únicamente debe de haber una sola instancia, es decir, se utiliza el patrón de diseño Singleton [4].

```kotlin
@Database(entities = [Movie::class], version = 1)
abstract class MovieDatabase : RoomDatabase() {
    abstract fun movieDao(): MovieDao

    companion object {
        private lateinit var instance: MovieDatabase
        @Synchronized
        fun getInstance(context: Context): MovieDatabase {
            if (!Companion::instance.isInitialized) {
                instance = Room.databaseBuilder(
                    context.applicationContext,
                    MovieDatabase::class.java, name: "movie_database"
                )
//                      TODO: Remove fallback
                    .fallbackToDestructiveMigration()
                    .addCallback(roomCallback)
                    .build()
            }
            return instance
        }

        private val roomCallback = object : RoomDatabase.Callback() {
            override fun onCreate(db: SupportSQLiteDatabase) {
                super.onCreate(db)
                Timber.tag( tag: "RoomCallback").v( message: "Database is Created")
            }
        }
    }

}
```

Como un ejemplo del algoritmo destacado se puede poner un “deserializador json” el cual simplemente accede al elemento json y solamente coge la parte necesaria del elemento para poder posteriormente convertir el este con facilidad en el POJO. Se puede ver dicho algoritmo en la Ilustración 33: JSON Deserializer [7].

```
class MoviesDeserializer : JsonDeserializer<ArrayList<Movie>> {

    private val TAG = MoviesDeserializer::class.java.simpleName

    override fun deserialize(json: JsonElement?, typeOfT: Type?, context: JsonDeserializationContext?): ArrayList<Movie>? {
        var movies: ArrayList<Movie>? = null
        try {
            val jsonObject = json!!.asJsonObject
            if (PreferencesManager.loadTotalPagesMovies() != 0)
                PreferencesManager.saveTotalPagesMovies(jsonObject.getAsJsonPrimitive(AppConfig.TOTAL_PAGES_FIELD).asInt)
            val moviesJsonArray = jsonObject.getAsJsonArray(AppConfig.MOVIES_ARRAY_DATA)
            movies = ArrayList(moviesJsonArray.size())
            for (i in 0 until moviesJsonArray.size()) {
                val dematerialized = context!!.deserialize<Movie>(moviesJsonArray.get(i), Movie::class.java)
                movies.add(dematerialized)
            }
        } catch (e: JsonParseException) {
            Timber.e(String.format("Could not deserialize Movie element: %s", json.toString()))
        }
        return movies
    }
}
```

Ilustración 33: JSON Deserializer

5.4 Implementación de la arquitectura de tres capas

Para la implementación de tres capas hemos organizado los paquetes de la siguiente forma:

- Capa de vista:
  Dicha capa contiene únicamente las clases que tienen una relación directa con la interfaz de la aplicación. Estas clases contienen comportamientos, eventos y enlaces a datos.

  Como se puede ver en la imagen el paquete es llamado de la misma forma que la capa.

- Capa de modelo de vista:
  El paquete como bien se puede observar es llamado de la misma forma que la capa y contiene las clases ViewModel que se han mencionado en el apartado anterior.

  Cada clase contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre las capas de vista y ViewModel se realiza por medio del patrón de diseño *Observer*.

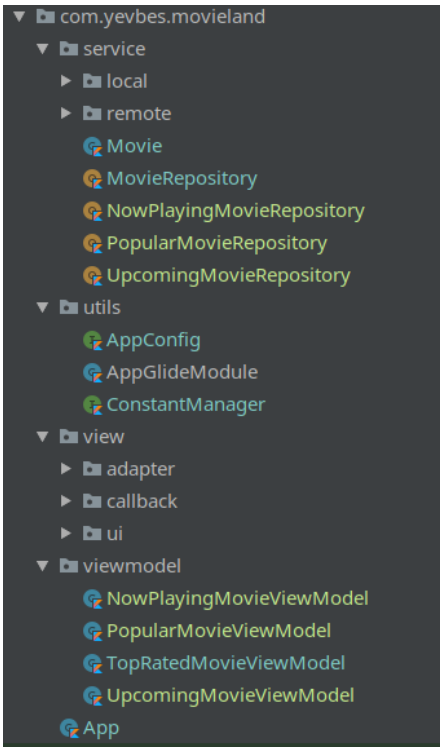

Ilustración 34: Estructura del proyecto

- Capa de Modelo:
  En este caso el paquete se ha llamado *service* y representa las clases de la lógica de negocio. Como se puede observar en la imagen el autor ha decidido dividir dicho paquete en dos sub-paquetes:
  - Local: este sub-paquete contiene las clases que trabajan con las bases de datos cuya implementación se ha mencionado en los apartados anteriores. Es decir, contiene una clase llamada MovieDAO que proporciona la especificación de las consultas que sirven para interactuar con la base de datos local. Por otra parte, contiene también una clase llamada MovieDatabase que es la que se encarga de crear únicamente una sola vez la conexión hacia la base de datos local.

  - Remote: dicho sub-paquete contiene las clases que implementan la funcionalidad necesaria para poder trabajar con el servidor. Como podemos observar una de las clases importantes es ServiceGenerator que genera un servicio REST API para poder trabajar con las llamadas al servidor para recuperar o enviar los datos necesarios. La clase RestService contiene la especificación de las llamadas hacía el servidor.

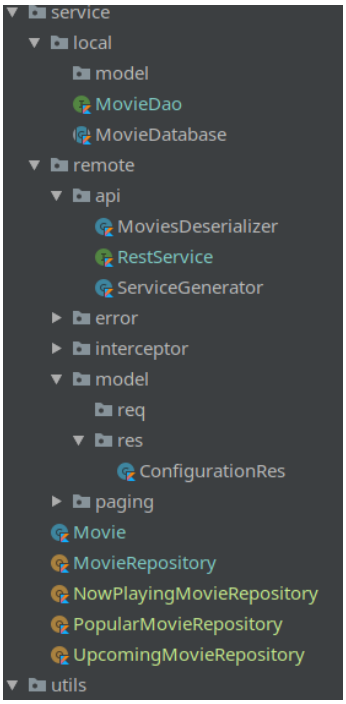

Ilustración 35: Estructura del proyecto

    El paquete error contiene las clases que definen los errores que llegan desde el servidor

    El paquete paging contiene las clases relacionadas con la librería Paging que sirve para poder manejar las listas de películas traídas desde el servidor eficientemente.

    El paquete interceptor contiene las clases que interceptan las operaciones como PUT, GET, DELETE, POST y las modifican. Esto sirve por ejemplo si queremos cada vez que lanzamos una petición al servidor adjuntarle una cabecera con el token que ha generado el servidor para poder autenticarnos.

  Cada de estos sub-paquetes contienen las clases modelo o entidades y también las clases repositorio que sirven para poder decidir sobre que fuente de datos se va a actuar.

Otras clases y sub-paquetes del paquete principal contienen las clases de utilidad para el programador.

5.5 Generación de la aplicación

Para generar un apk se han realizado los siguientes procedimientos:

1. Abrir la ventana de configuración (Run/Debug Configurations).
2. Añadir un elemento de tipo Gradle y darle el nombre.
3. En la pestaña de configuracion (Configuration) elegir el actual proyecto y en la opción “Tasks” escribir assemble, después guardar los ajustes.
4. En la parte desplegable al lado de la opción de ejecutar elegir la configuración que se acaba de crear y darle a la opción de ejecutar.
5. Después de un cierto tiempo se obtiene el apk en el directorio ProjectName\app\build\outputs\apk.

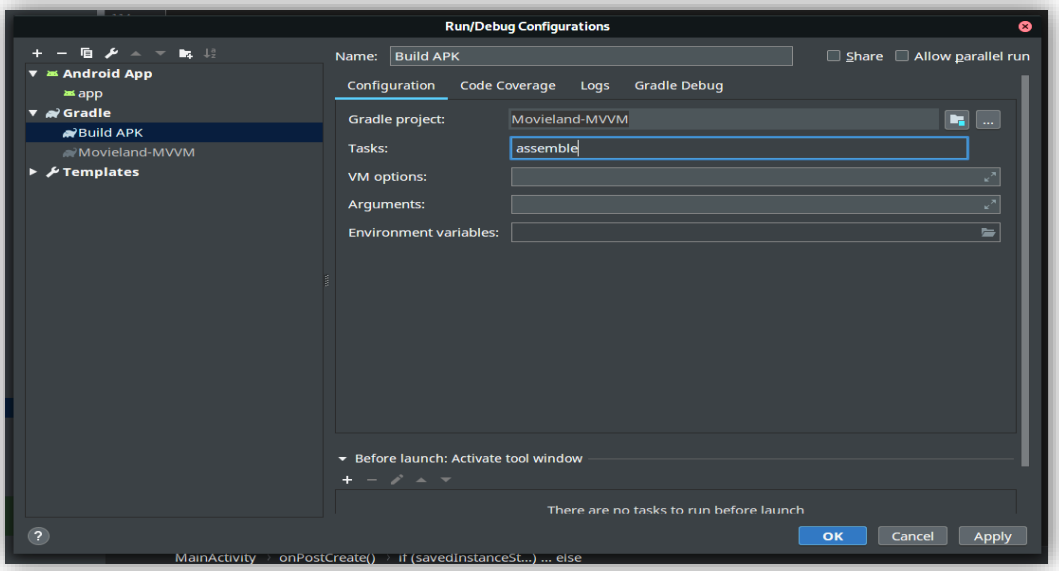

Ilustración 36: APK

6. Pruebas

Este apartado está dedicado exclusivamente para las pruebas de la aplicación. Las pruebas que se realizarán sobre la aplicación serán de funcionalidad, usabilidad y compatibilidad.

6.1 Pruebas de funcionalidad

Hemos realizado tests funcionales con los diferentes tipos de dispositivo incluyendo reales para poder detectar posibles fallos en la aplicación. Los resultados de estos se reflejan en la siguiente tabla. Cabe destacar que los casos de uso se han testeado con diferentes estados de red, es decir con conexión a internet y sin ella.

Casos de uso	Pruebas	Éxito	Comentarios
CU-01	4	Si	La aplicación permite la búsqueda de películas correctamente.
CU-02	0	-	Debido a que la funcionalidad está aún en desarrollo se ha decidido no hacer el test para este caso de uso.
CU-03	4	Si	La aplicación permite añadir la película a la lista de favoritos satisfactoriamente.
CU-04	4	Si	La aplicación permite quitar la película de favoritos correctamente.
CU-05	0	-	Debido a que la funcionalidad está aún en desarrollo por lo que se ha decidido no hacer el test para este caso de uso.
CU-06	4	Si	La aplicación permite añadir la película a la lista “para ver” satisfactoriamente.
CU-07	4	Si	La aplicación permite quitar películas de la lista “para ver” correctamente.
CU-08	0	-	Debido a que la funcionalidad está aún en desarrollo se ha decidido no hacer el test para este caso de uso.
CU-09	4	Si	La aplicación permite marcar películas como vista correctamente.
CU-10	0	-	Debido a que la funcionalidad está aún en desarrollo se ha decidido no hacer el test para este caso de uso.
CU-11	4	Si	La aplicación permite mostrar lista de películas correctamente.
CU-12	4	Si	La aplicación permite desmarcar una película como vista satisfactoriamente.
CU-13	4	Si	La aplicación permite actualizar la lista de las películas correctamente.

Tabla 16: Pruebas de funcionalidad

6.1.1 Conclusiones sobre pruebas de funcionamiento

Como se puede observar, todos los casos de uso se han realizado con éxito y sin fallos. A excepción de los casos de uso que se implementaron parcialmente en la aplicación y el autor ha decidido no incluirlos en las pruebas.

6.2 Pruebas de usabilidad

Se plantean dos pruebas de usabilidad. En la primera prueba se cubren los casos de uso en su totalidad y en la segunda los casos de uso más importantes. Dichas pruebas son realizadas por tres potenciales usuarios reales de entre 18 y 23 años. El autor toma el rol del observador y anota los comentarios para la tabla de resultados.

En las tablas generadas a partir de los comentarios del autor se pueden ver los resultados de las distintas pruebas aplicadas por los tres usuarios mencionados anteriormente.

Los dispositivos utilizados para la prueba son:

- Samsung Galaxy Note 4 Snapdragon – API 25 (dispositivo real)
- Pixel 2 – API 28 (dispositivo virtual)
- Nexus S – API 19 (dispositivo virtual)
- Nexus 4 – API 21 (dispositivo virtual)

6.2.1 Prueba 1

Para la primera prueba el usuario ha seguido los siguientes pasos:

1. Abrir la lista de películas mejor valoradas.
2. Buscar la película “The Imitation Game”.
3. Añadir la película a la lista “para ver”.
4. Abrir la lista “para ver”.
5. Marcar la película como vista.
6. Añadir la película a los favoritos.
7. Quitar la película de la lista y observar que la lista se queda vacía.
8. Posicionarse sobre la lista de películas populares y añadir dos películas cualesquiera a la lista de favoritos.
9. Posicionarse sobre la lista de películas favoritas.
10. Observar que en total hay tres películas y una de ellas es “The Imitation Game” y está marcada como vista.
11. Desmarcar la película “The Imitation Game” como vista.
12. Quitar alguna película de la lista y ver que en total se quedan dos películas.
13. Posicionarse sobre la lista de las películas por estrenar y actualizar la lista.

Usuario 1

Tarea	Caso de uso	Observaciones
1	CU-11	El usuario no tenía que hacer nada ya que al iniciar la aplicación por defecto se abre la lista de películas mejor valoradas.
2	CU-01	Sin incidencias.
3	CU-06	Sin incidencias.
4	CU-11	Sin incidencias.
5	CU-09	El usuario no encontraba la funcionalidad en la aplicación.
6	CU-03	Sin incidencias.
7	CU-07	Sin incidencias.
8	CU-11, CU-03	Sin incidencias.
9	CU-11	Sin incidencias.
10	CU-09	Sin incidencias.
11	CU-12	Sin incidencias.
12	CU-04	Sin incidencias.
13	CU-13	Sin incidencias.

Tabla 17: Prueba de usabilidad 1 - usuario 1

Usuario 2

Tarea	Caso de uso	Observaciones
1	CU-11	Sin incidencias, en este caso la aplicación estaba posicionada sobre otra lista distinta.
2	CU-01	Sin incidencias.
3	CU-06	El usuario no encontraba la funcionalidad en la aplicación.
4	CU-11	Sin incidencias.
5	CU-09	El usuario no encontraba la funcionalidad en la aplicación.
6	CU-03	Sin incidencias.
7	CU-07	El usuario no sabía cómo realizar la acción.
8	CU-11, CU-03	Sin incidencias.
9	CU-11	Sin incidencias.
10	CU-09	Sin incidencias.
11	CU-12	Sin incidencias.
12	CU-04	Sin incidencias.
13	CU-13	Sin incidencias.

Tabla 18: Prueba de usabilidad 1 - usuario 2

Usuario 3

Tarea	Caso de uso	Observaciones
1	CU-11	El usuario no tenía que hacer nada ya que al iniciar la aplicación por defecto se abre la lista de películas mejor valoradas.
2	CU-01	Sin incidencias.
3	CU-06	Sin incidencias.
4	CU-11	Sin incidencias.
5	CU-09	Sin incidencias.
6	CU-03	Sin incidencias.
7	CU-07	Sin incidencias.
8	CU-11, CU-03	Sin incidencias.
9	CU-11	Sin incidencias.
10	CU-09	Sin incidencias.
11	CU-12	Sin incidencias.
12	CU-04	Sin incidencias.
13	CU-13	Sin incidencias.

Tabla 19: Prueba de usabilidad 1 - usuario 3

6.2.2 Prueba 2

Para la segunda prueba el usuario ha seguido los siguientes pasos:

1. Abrir lista de películas que están por estrenar.
2. Abrir detalle de la película e intentar marcar la película como vista.
3. Añadir la película a la lista de favoritos.
4. Abrir la lista de las películas favoritas.
5. Quitar la película favorita.
6. Buscar la película favorita recién eliminada.

Usuario 1

Tarea	Caso de uso	Observaciones
1	CU-11	Sin incidencias.
2	CU-09	El usuario no ha podido marcar la película ya que esta pertenece a la lista por estrenar y la funcionalidad está deshabilitada.
3	CU-06	Sin incidencias.
4	CU-11	El usuario tuvo que regresar a la ventana anterior para ir a la lista.
5	CU-04	Sin incidencias.
6	CU-01	Debido a que la lista está vacía no se ha podido buscar la película.

Tabla 20: Prueba de usabilidad 2 - usuario 1

Usuario 2

Tarea	Caso de uso	Observaciones
1	CU-11	Sin incidencias.
2	CU-09	El usuario no encontraba la funcionalidad.
3	CU-06	Sin incidencias.
4	CU-11	El usuario tuvo que regresar a la ventana anterior para ir a la lista.
5	CU-04	Sin incidencias.
6	CU-01	Aunque la lista no se encontraba vacía el usuario no pudo encontrar la película anterior.

Tabla 21: Prueba de usabilidad 2 - usuario 2

Usuario 3

Tarea	Caso de uso	Observaciones
1	CU-11	Sin incidencias.
2	CU-09	El usuario no ha podido marcar la película ya que esta pertenece a la lista por estrenar y la funcionalidad está deshabilitada.
3	CU-06	Sin incidencias.
4	CU-11	El usuario tuvo que regresar a la ventana anterior para ir a la lista.
5	CU-04	Sin incidencias.
6	CU-01	Debido a que la lista está vacía no se ha podido buscar la película.

Tabla 22: Prueba de usabilidad 2 - usuario 3

6.2.3 Conclusiones sobre pruebas de usabilidad

Analizando detalladamente el primer test de usabilidad realizado por tres personas distintas podemos llegar a la conclusión de que no todos los usuarios podrían darse cuenta de que existe la funcionalidad de marcar la película como vista ya que algunos de los usuarios sabiendo que tenían que hacer esta tarea no han podido encontrar la opción en la aplicación.

Para el segundo test se han propuesto tareas para comprobar el incumplimiento de esta funcionalidad y así mismo comprobar el estado de la aplicación después de la realización de la misma. En primer lugar se quería marcar una película pendiente de estrenar como una película vista, pero es imposible que una persona pueda verla antes de su estreno. Y por ello la aplicación avisó al usuario de la imposibilidad de la acción. En segundo lugar, se ha propuesto buscar una película en la lista vacía, pero en este caso la funcionalidad de buscar la película está deshabilitada y nuevamente la aplicación le aviso al usuario de ello.

6.3 Pruebas de compatibilidad

Para las pruebas de compatibilidad se ha decidido escoger tres API's cualquieras que entran en el rango [19-28] y otra que no [2]. En el primer caso se ha escogido la API 19 (Android KitKat), API 24 (Android Nougat) y la API 28 (Android Pie), en el segundo caso se ha decidido escoger la API 18 (Jelly Bean). Por último, se hará una prueba en el dispositivo real Samsung Galaxy Note 4 (API 28).

Para aclarar porque se han escogido estas versiones de Android se muestra una captura de pantalla en la que aparece la configuración del archivo Gradle de Android donde se especifica la versión mínima de Android en la que se lanzará la aplicación.

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.yevbes.movieland"
        minSdkVersion 19
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        vectorDrawables.useSupportLibrary = true
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```

Ilustración 37: Gradle

Antes de empezar las pruebas hemos creado cuatro dispositivos virtuales con sus respectivas API's los cuales se pueden ver en la siguiente figura.

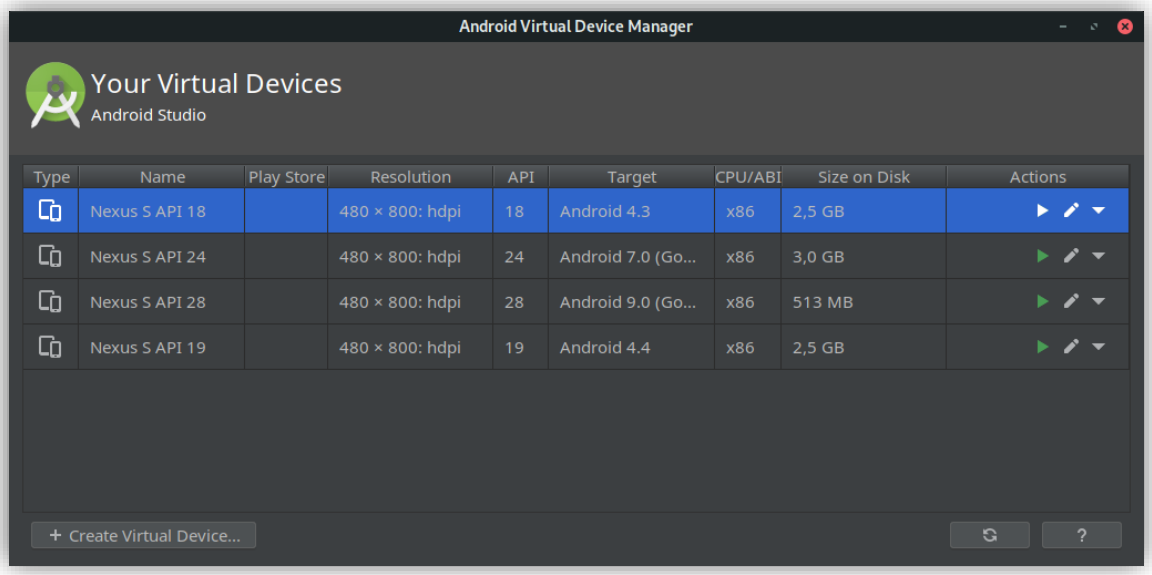

Ilustración 38: AVD

6.3.1 Prueba 1: API 19

Ilustración 39: Prueba de compatibilidad - Prueba 1

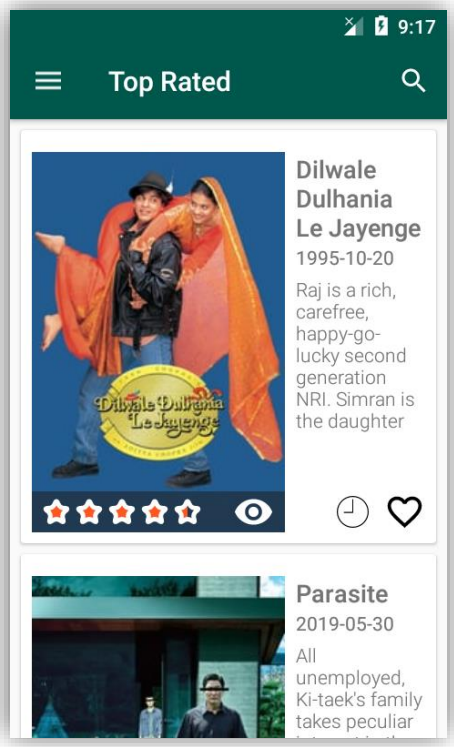

Ilustración 40: Prueba de compatibilidad - Prueba 1

6.3.2 Prueba 2: API 24

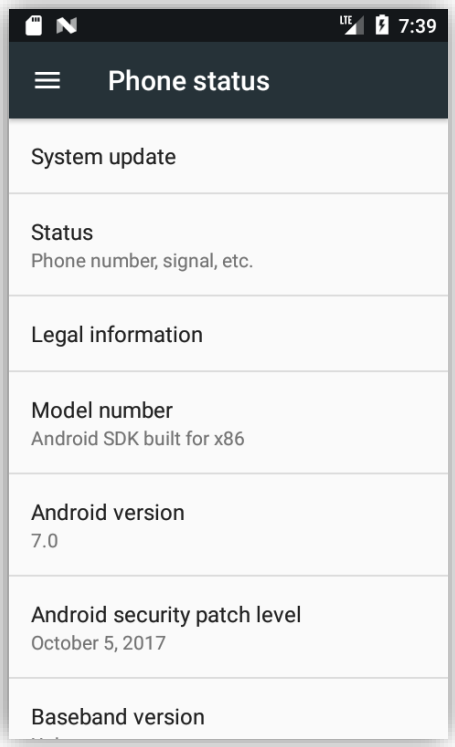

Ilustración 42: Prueba de compatibilidad - Prueba 2

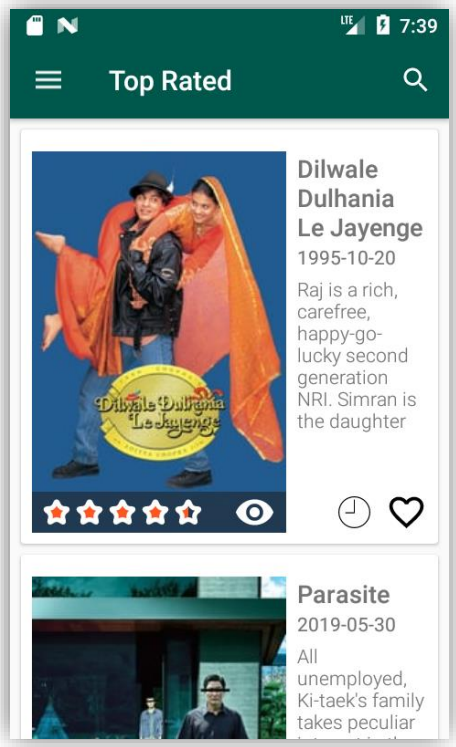

Ilustración 41: Prueba de compatibilidad - Prueba 2

6.3.3 Prueba 3: API 28

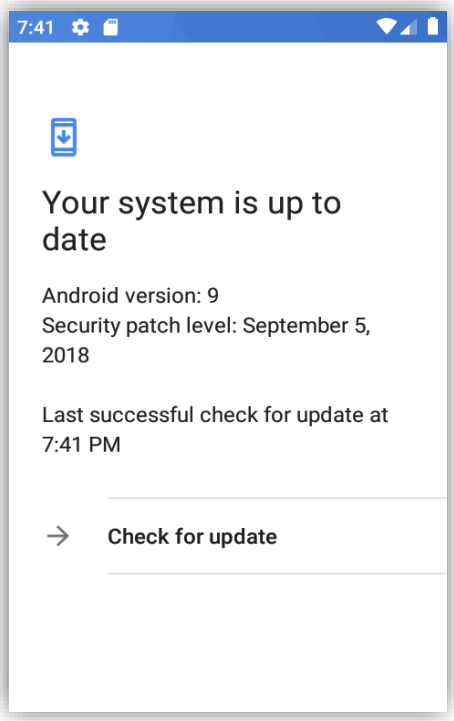

Ilustración 44: Prueba de compatibilidad - Prueba 3

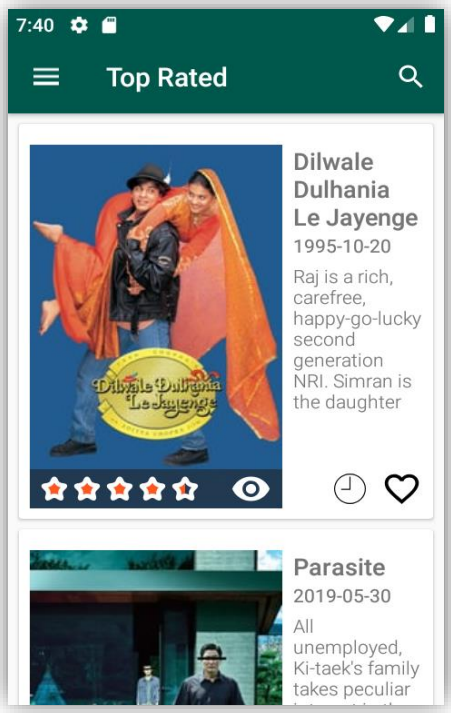

Ilustración 43: Prueba de compatibilidad - Prueba 3

6.3.4 Prueba 4: API 18

Ilustración 46: Prueba de compatibilidad - Prueba 4

Ilustración 45: Prueba de compatibilidad - Prueba 4

6.3.5 Prueba 5: Samsung Note 4 (API 28)

Ilustración 48: Prueba de compatibilidad - Prueba 5

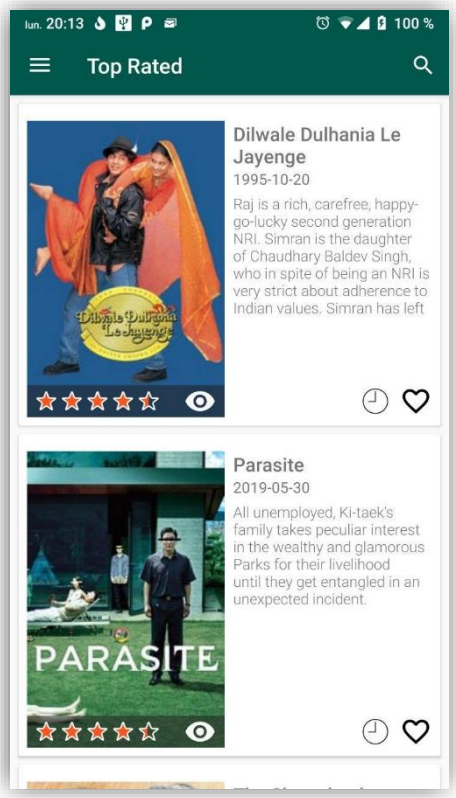

Ilustración 47: Prueba de compatibilidad - Prueba 5

6.3.6 Conclusiones sobre pruebas de compatibilidad

Tal y como se muestra en las capturas la aplicación está instalada y ejecutada en los dispositivos cuyos API's son 19, 24 y 28 así como en el dispositivo real Samsung Galaxy Note 4 que tiene la API 28. En cambio, en el dispositivo cuyo API es 18 no es posible instalar la aplicación, ya que el sistema móvil avisa de que la aplicación necesita una versión posterior de Android para la instalación de la aplicación.

7. Resultado final

En el apartado actual se mostrarán las capturas de la aplicación junto con una breve descripción para representar el resultado final del trabajo.

7.1 Menú principal

En la ilustración 49 se muestra el menú de navegación desplegable con el logotipo de la aplicación y el nombre en la cabecera. Debajo de la cabecera se puede ver los menús agrupados por distintas secciones.

Por defecto siempre estará marcado el primer elemento de la navegación que corresponde a “Top Rated” del grupo “Movies”. Al estar marcado el menú siempre se abrirá la lista de las películas mejor valoradas.

Como bien podemos ver en estas capturas hay elementos relacionados con el caso de uso CU-11 Mostrar lista de películas.

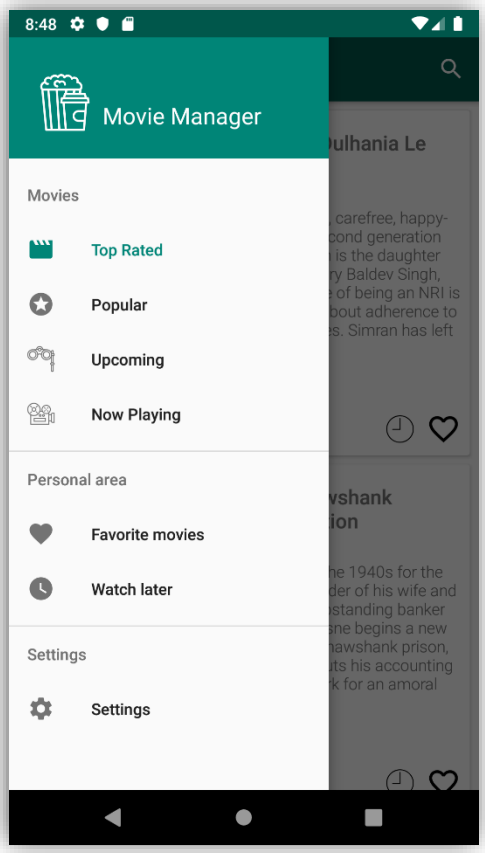

Ilustración 49: Menú principal - lista de películas

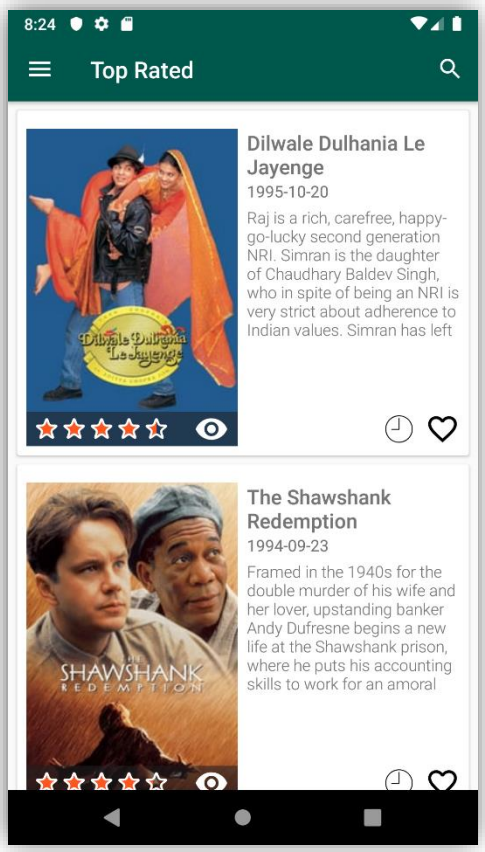

Ilustración 50: Menú principal - navegación

7.2 Listas de películas (CU-11)

La representación con las capturas de pantalla del caso de uso CU-11 (Mostrar lista de películas).

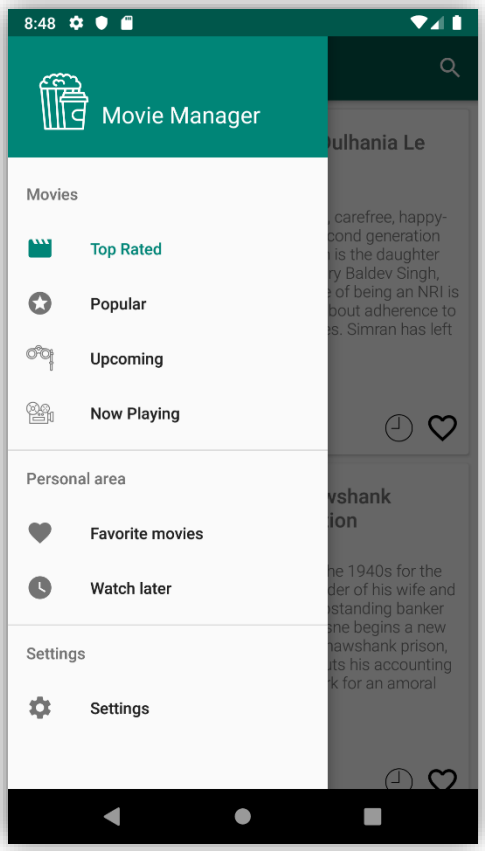

Ilustración 51: Listas de películas - navegación

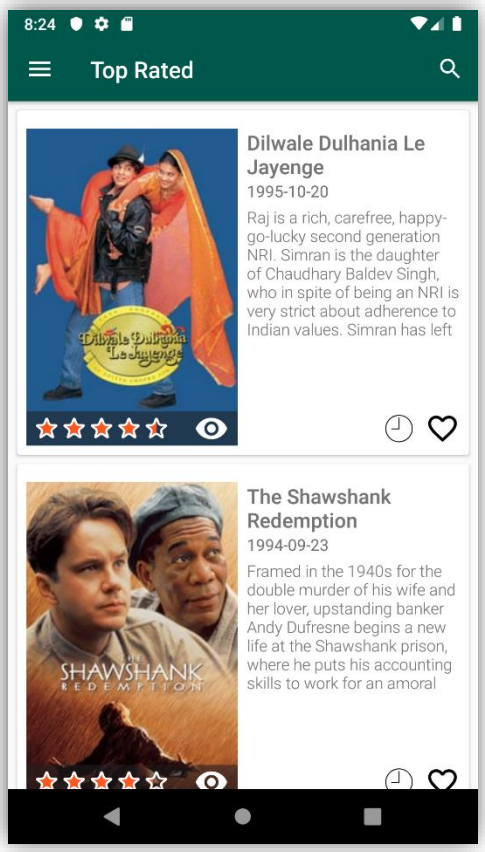

Ilustración 52: Listas de películas - resultado

Como se puede observar en la Ilustración 53: TMDB Server los datos están siempre bien sincronizados.

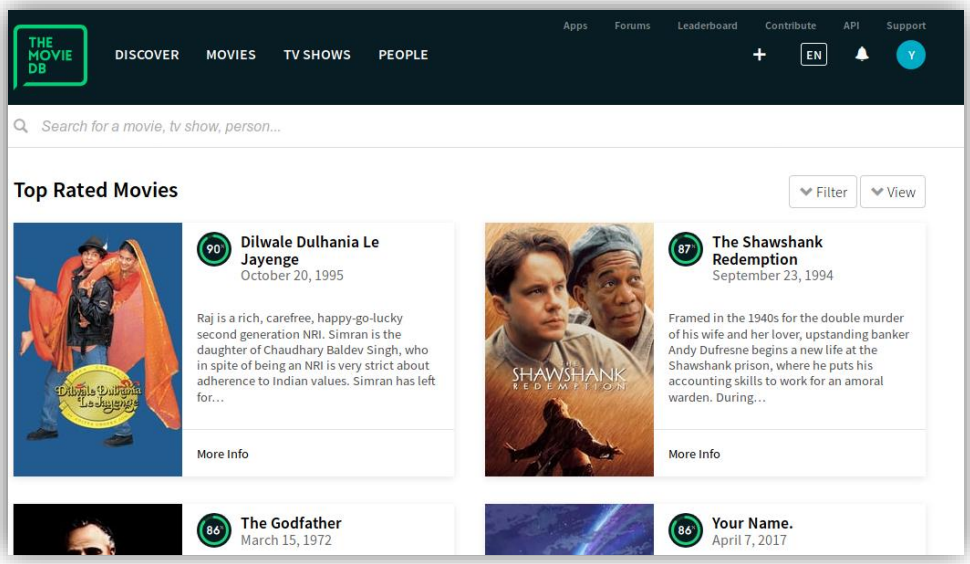

Ilustración 53: TMDB Server

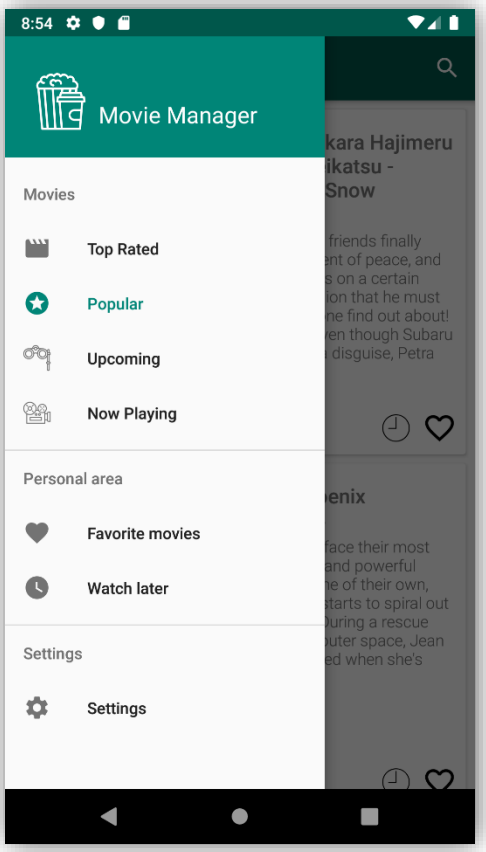

Ilustración 54: Listas de películas - popular opción

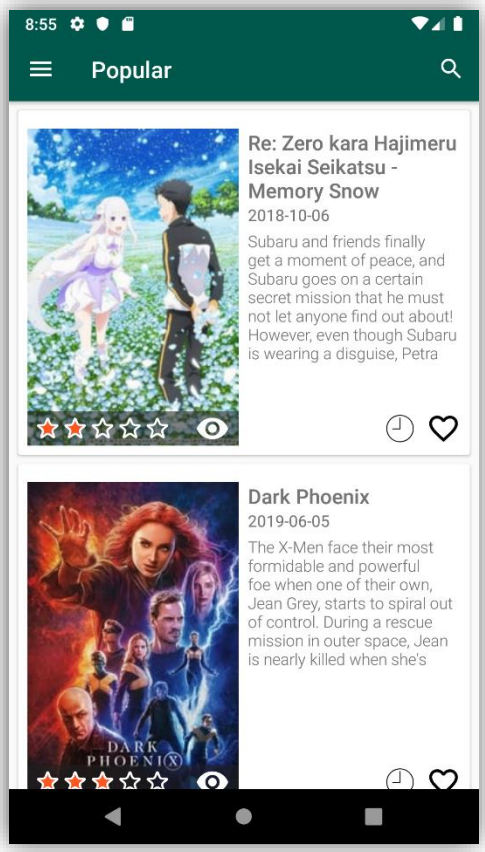

Ilustración 55: Listas de películas - lista popular

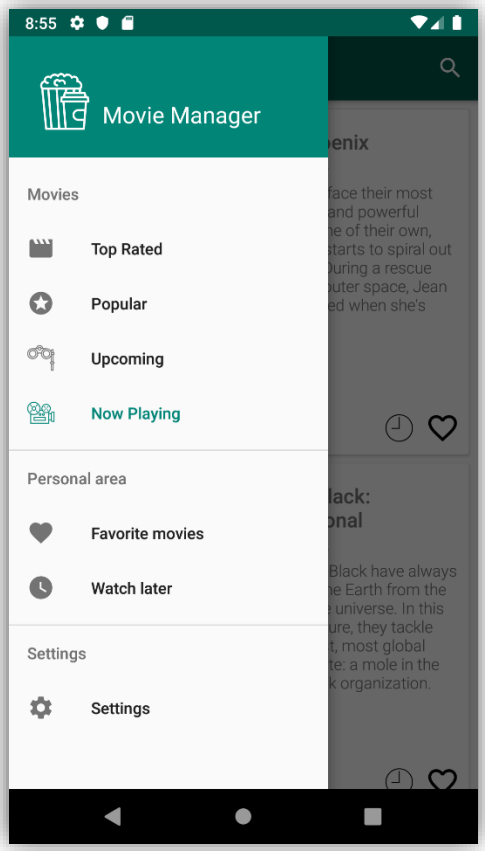

Ilustración 56: Listas de películas - now playing opción

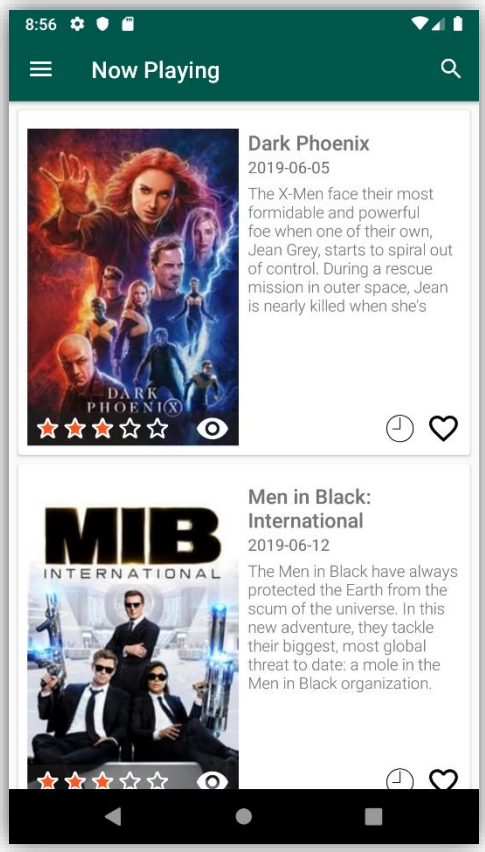

Ilustración 57: Listas de películas - lista now playing

7.3 Buscar películas CU-01

La representación con las capturas de pantalla del caso de uso CU-01 (Buscar película).

En este caso se ha utilizado la lista de películas favoritas para mostrar un ejemplo de la búsqueda.

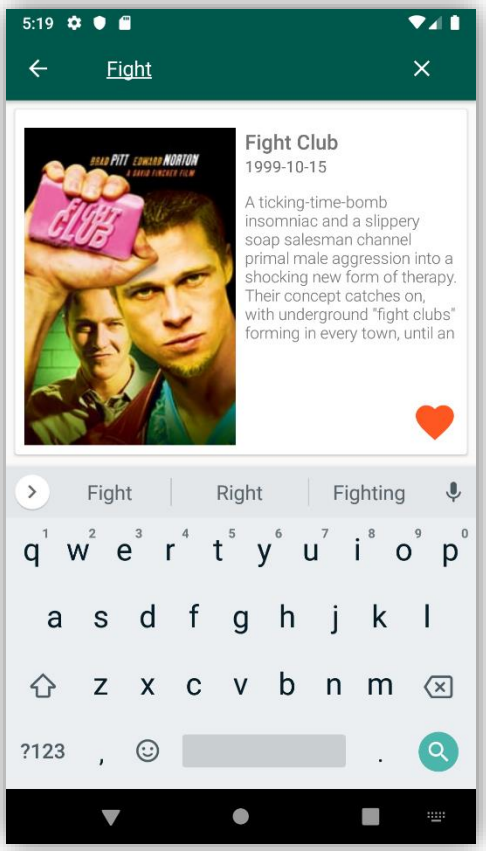

Ilustración 58: Buscar películas

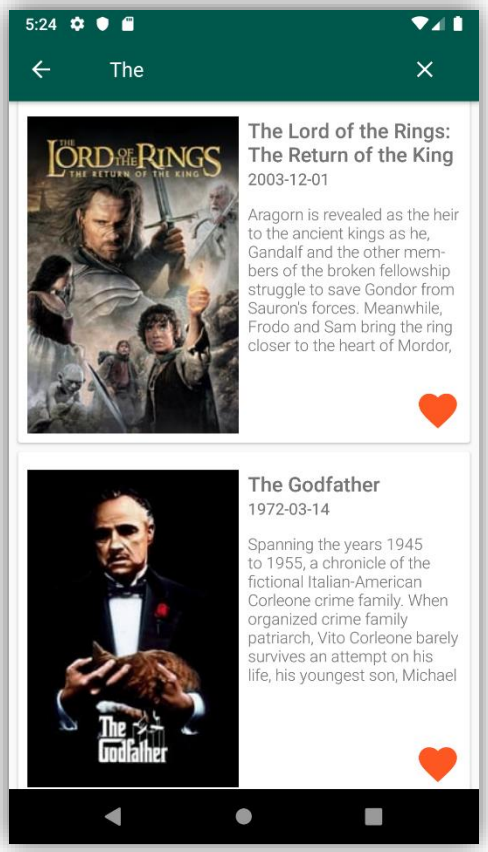

Ilustración 59: Buscar películas

7.4 Películas favoritas CU-03, 04

La representación con las capturas de pantalla del caso de uso CU-03 (Añadir película a favoritos) y CU-04 (Quitar película de favoritos).

Para mostrar la lista hace falta posicionarse en el menú desplegable principal y presionar sobre el elemento del menú “Favourite movies”.

Para añadir la película a favoritos basta con pulsar el objeto en forma de corazón en alguna de las listas anteriormente mencionadas o en la ventana de detalles de la película. Posteriormente se le notificará al usuario que la película ha sido añadida satisfactoriamente y el objeto se marcará en color rojo.

Para quitar película hace falta pulsar de nuevo en el objeto en forma del corazón en la lista de películas favoritas. Cabe destacar que por un tiempo determinado se puede deshacer el cambio de quitar película.

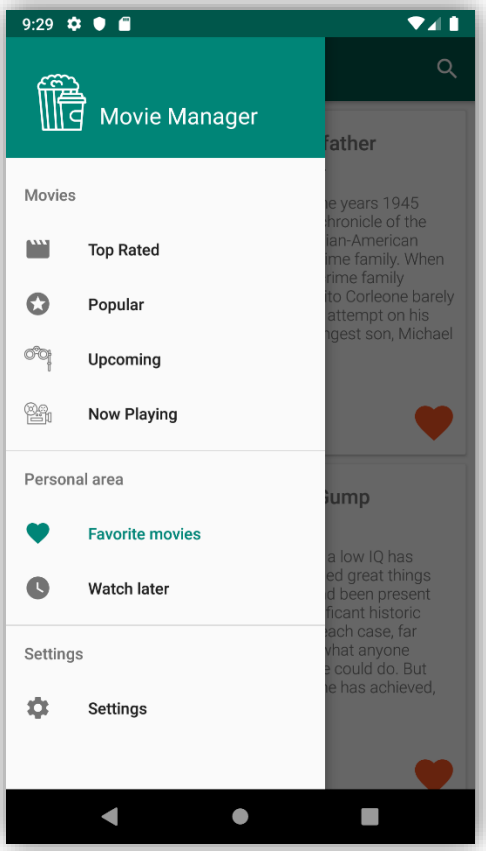

Ilustración 61: Películas favoritas - opción

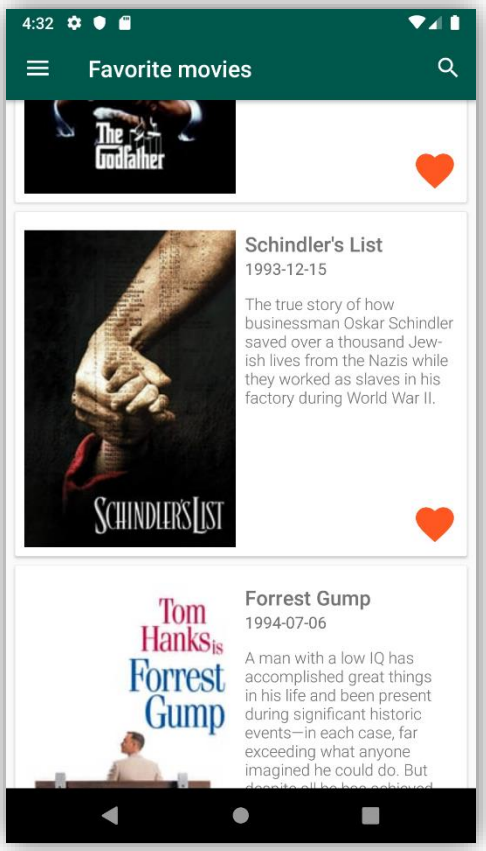

Ilustración 60: Películas favoritas - lista

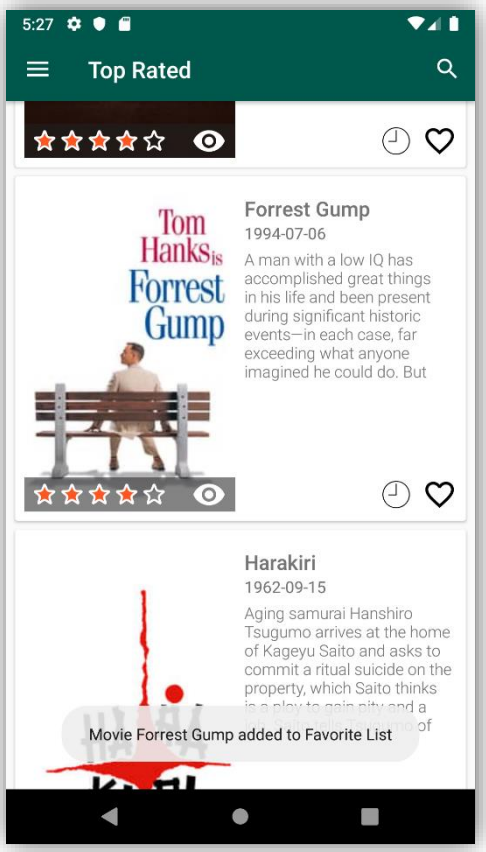

Ilustración 62: Películas favoritas - añadir

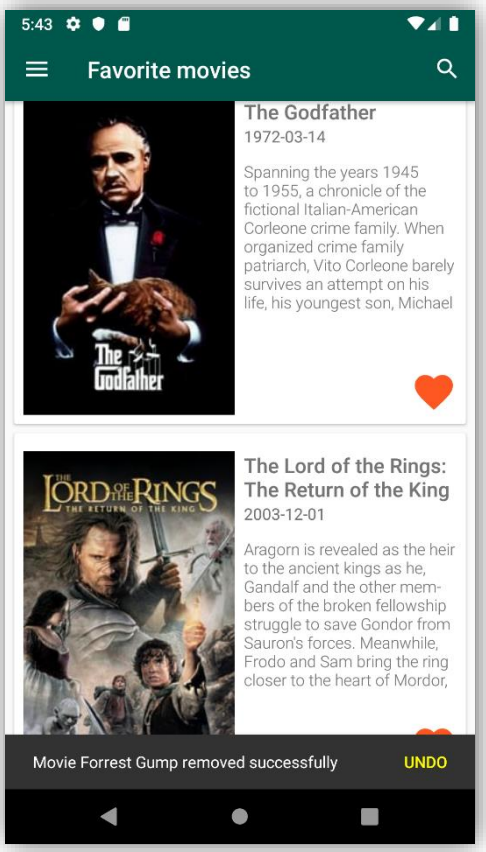

Ilustración 63: Películas favoritas - Quitar

7.5 Películas para ver CU-06, 07

La representación con las capturas de pantalla del caso de uso CU-06 (Añadir película a lista “para ver”) y CU-07 (Quitar película de lista “para ver”).

Para mostrar la lista hace falta posicionarse en el menú desplegable principal y presionar sobre el elemento del menú “Watch later”. Posteriormente se hará el procedimiento parecido al explicado anteriormente con la lista de las películas favoritas, por ello no se adjuntan más capturas.

Para añadir la película a la lista basta con pulsar el objeto en forma de reloj en alguna de las listas anteriormente mencionadas o en la ventana de detalles de la película.

Para quitar una película hace falta pulsar de nuevo en el objeto en forma de reloj en la lista del caso de uso actual. Cabe destacar que por un tiempo determinado se puede deshacer el cambio de quitar película de la lista.

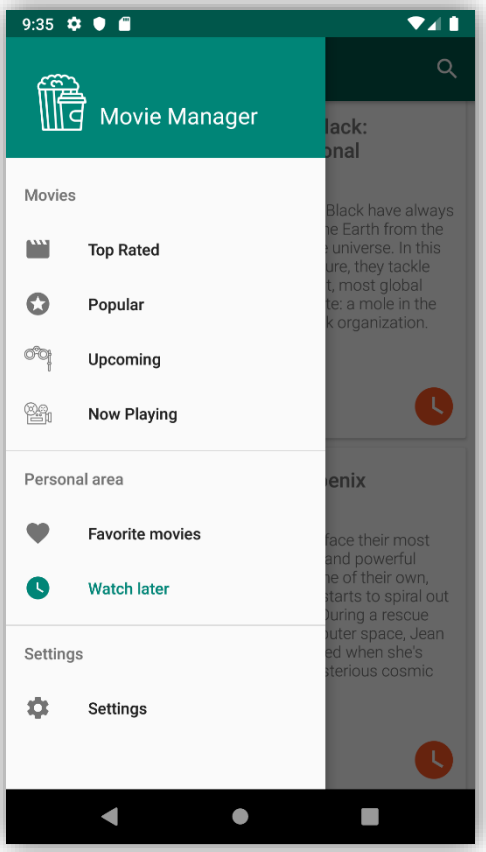

Ilustración 64: Películas para ver - opción

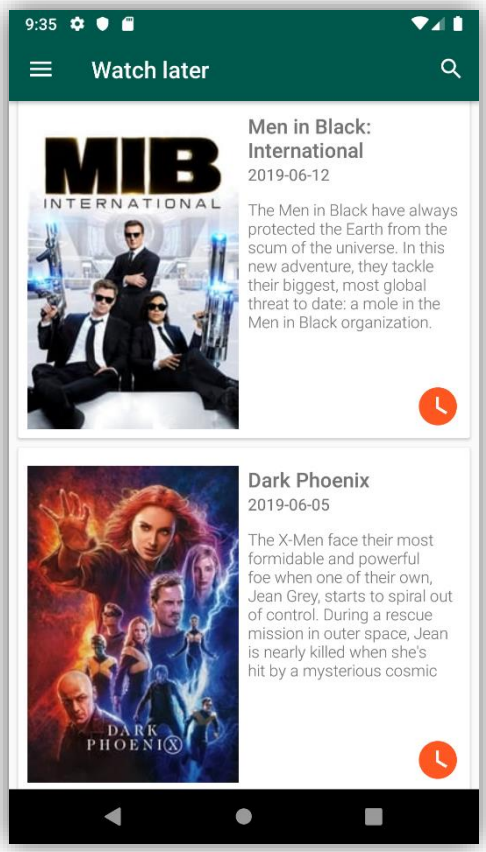

Ilustración 65: Películas para ver - lista

7.6 Actualizar la lista de películas CU-13

La representación con la captura de pantalla del caso de uso CU-13 (Actualizar películas).

Para actualizar la lista hace falta estar posicionado en el primer elemento de la lista y deslizar con el dedo hacia abajo hasta que aparezca el objeto en la pantalla con el circulo que simboliza la actualización, posteriormente quitar el dedo de la pantalla y la lista empezará a actualizarse.

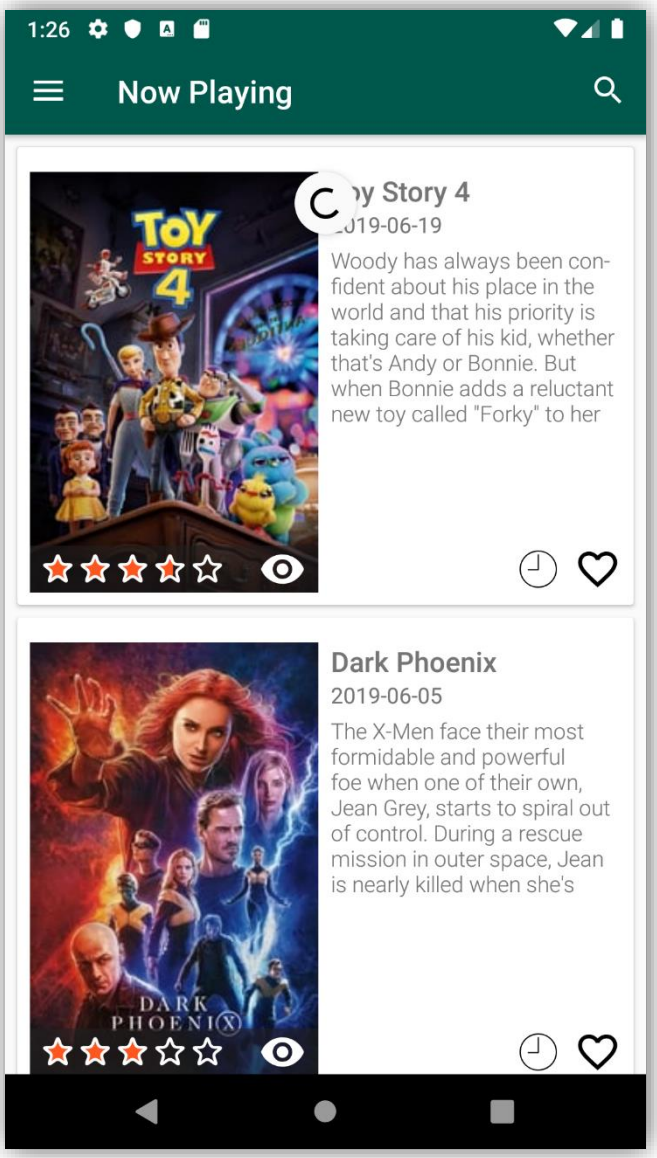

Ilustración 66: Actualizar la lista de películas

7.7 Detalle de película CU-09,12

La representación con las capturas de pantalla del caso de uso CU-09 (Marcar película como vista) y CU-12 (Desmarcar película como vista).

Para marcar la película como vista lo único que hace falta es pulsar sobre el objeto en forma de ojo que se encuentra en la posición superior de la caratula de la película. Posteriormente se notificará al usuario de la acción y se marcará el objeto de color naranja.

Para desmarcar se sigue el mismo procedimiento explicado anteriormente con la diferencia de que el objeto tomará su estado original, es decir el color del objeto se pondrá en blanco.

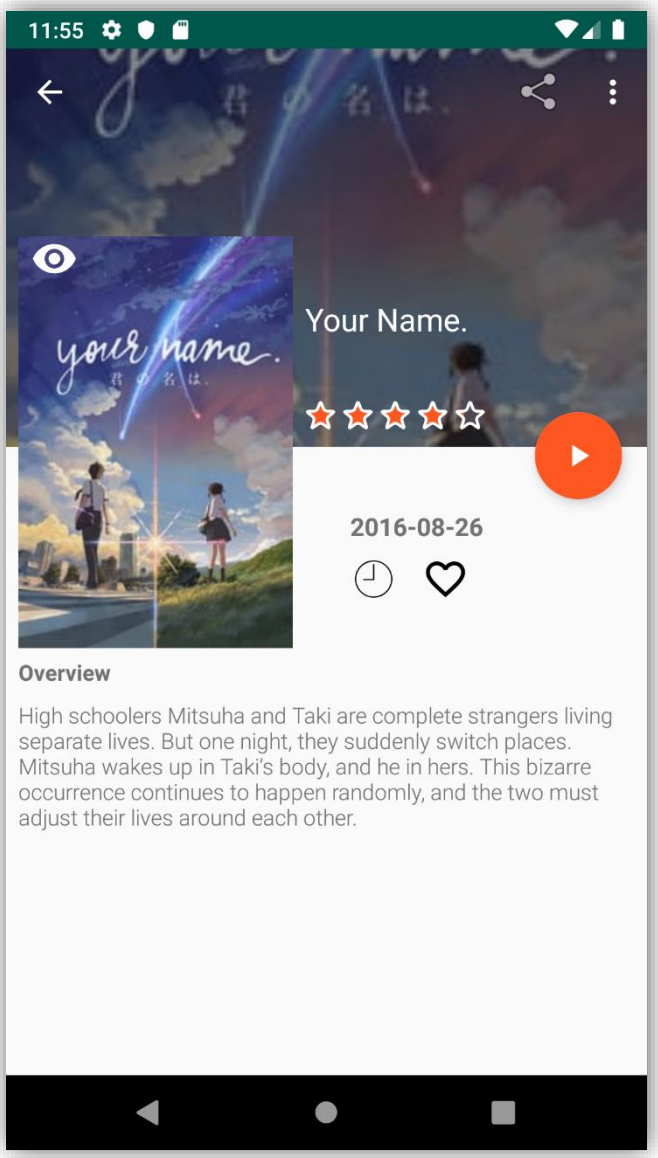

Ilustración 67: Detalle de película

8. Conclusiones y futuros trabajos

En este apartado se describe una serie de conclusiones a las que llega el autor una vez terminado el trabajo. Así como también se describen los trabajos que se van a realizar de la dicha aplicación en el futuro para darle más funcionalidad.

8.1 Conclusiones técnicas

Como bien se puede recordar el objetivo de este proyecto era desarrollar una aplicación móvil bajo el sistema operativo Android, que permita a los usuarios poder gestionar las películas.

Después de un conjunto de los tests funcionales sobre la aplicación se ha construido la primera versión de la aplicación. La misma ha sido testeada por distintos tipos usuarios que encontraron diversos errores en la aplicación y en el diseño que posteriormente se corrigieron. Después de un trabajo de corrección de errores surgidos por los tests, se ha generado nuevamente la aplicación en su segunda versión y se ha testeado nuevamente con los mismos procedimientos. En esta segunda versión la aplicación ha generado buenas sensaciones y como conclusión se puede comentar que los objetivos pretendidos por el autor han sido alcanzados satisfactoriamente.

Si de nuevo nos referimos a la tabla MoSCow sin contar los requisitos *Could have* y *Won't have* podemos observar el nivel de completitud de la aplicación.

Prioridad	Requisitos	Se cumple
Debe tener *(Must have)*	Una lista de películas disponibles en los cines.	Sí
	Una lista de películas favoritas.	Sí
	Una búsqueda.	Sí
	Calificación de películas.	Sí
	Detalle de película.	Sí
	Actualización de listas.	Sí
	Una lista de películas populares.	Sí
	Una lista de películas mejor valoradas.	Sí
Debería tener (*Should have)*	Una lista de próximos estrenos.	Sí
	Un filtro.	Parcialmente
	Ordenar por.	Parcialmente
	Una lista con películas para ver.	Sí
	Ajustes.	Parcialmente
	Una interfaz intuitiva.	Sí

Tabla 23: MoSCow

En las siguientes dos tablas se pueden ver los requisitos no funcionales y casos de uso propuestos y alcanzados por el autor. Así como algunos objetivos pospuestos para versiones futuras de la aplicación.

Tipo de requisito	Comentarios	Alcanzado
Usabilidad	El usuario puede con facilidad utilizar la aplicación. Interfaz amigable. Operatividad. Protección de errores.	Sí
Eficiencia	Utilización eficiente de los recursos. Aplicación no ocupa mucho espacio.	Sí
Confiabilidad	Aplicación se recupera de fallos perfectamente y se encuentra disponible en todo momento.	Sí
Seguridad	No se utiliza ningún dato del usuario, ni tampoco de la cuenta ya que esta funcionalidad se va a desarrollar para versiones posteriores de la aplicación.	Sí
Mantenimiento	Utilizando la arquitectura adecuada, testeando la aplicación y siguiendo el modelo SOLID la aplicación es fácilmente mantenible.	Sí
Compatibilidad	La aplicación es compatible con el mayor número dispositivos a partir de la API 19 inclusive.	Sí

Tabla 24: Conclusiones técnicas - requisitos no funcionales

Caso de uso	Comentarios	Alcanzado
CU-01	La aplicación permite la búsqueda de películas.	Sí
CU-02	Debido a que la funcionalidad está aún en desarrollo se ha decidido no incorporar el filtro de las películas en la primera versión.	Parcialmente
CU-03	La aplicación permite añadir la película a la lista de favoritos.	Sí
CU-04	La aplicación permite quitar película de favoritos.	Sí
CU-05	Debido a que la funcionalidad se está aún en desarrollo se ha decidido no incorporar notificaciones en la primera versión.	Parcialmente
CU-06	La aplicación permite añadir la película a la lista “para ver”.	Sí
CU-07	La aplicación permite quitar película de la lista “para ver”.	Sí
CU-08	Debido a que la funcionalidad ordenar películas está aún en desarrollo se ha decidido no incorporar dicha funcionalidad en la primera versión.	Parcialmente
CU-09	La aplicación permite marcar película como vista.	Sí
CU-10	Debido a que la funcionalidad está aún en desarrollo se ha decidido no incorporar notificaciones en la primera versión.	Parcialmente
CU-11	La aplicación permite mostrar lista de películas.	Sí
CU-12	La aplicación permite desmarcar una película como vista.	Sí
CU-13	La aplicación permite actualizar la lista de las películas.	Sí

Tabla 25: Conclusiones técnicas - requisitos funcionales

Al analizar las tablas se puede nuevamente concluir que casi todos los objetivos, sin contar los que van a ser incorporados en las futuras versiones de la aplicación, han sido alcanzados.

Para mayor aclaración con los casos de uso 02, 05, 08 y 10 el autor ha tomado la decisión de añadir estas funcionalidades en las siguientes versiones de la aplicación debido a que se necesita invertir una cantidad de tiempo importante para su funcionamiento.

8.2 Conclusiones personales

El objetivo que pretendía conseguir el autor desarrollando dicha aplicación era ampliar sus conocimientos en el área de desarrollo de aplicaciones para dispositivos móviles Android. Concretamente aprender a utilizar la arquitectura MVVM, así como también las librerías que se utilizan mucho hoy en día en el sector laboral.

Los objetivos descritos anteriormente no han sido solo cumplidos, sino que también gracias a la labor se ha aprendido a utilizar el entorno de desarrollo Android Studio, se ha aprendido programar en el lenguaje Kotlin y utilizar las tecnologías que sirven de ayuda extra al programador.

Como conclusión personal se puede comentar que desde el punto de vista del autor el proyecto le ha motivado a seguir trabajando en el área de desarrollo de aplicaciones para móviles y le ha ayudado a la compresión de las distintas tecnologías.

8.3 Relación con los estudios cursados

Durante los cuatro cursos que forman el grado, el autor ha cursado una serie de asignaturas que le han sido de utilidad para la realización del proyecto actual.

A continuación, se presentan una serie de asignaturas que tienen una relación directa con el proyecto:

- Estructuras de datos
- Programación
- Bases de datos
- Redes
- Lenguajes, tecnologías y paradigmas de la programación
- Ingeniería del software
- Gestión de proyectos
- Calidad de software
- Diseño de software
- Análisis y especificación de requisitos
- Análisis, depuración y validación de software
- Integración e interoperabilidad
- Proceso de software
- Proyecto de ingeniería de software

8.4 Futuros trabajos

Como bien se ha comentado con anterioridad, aparte de las mejoras que se pueden realizar y extender la funcionalidad de la aplicación, se pretende incorporar los casos de usos parcialmente implementados durante las dos siguientes versiones.

De tal modo que en la siguiente versión se van a incorporar los casos de uso CU-02 (Filtrar películas) y CU-08 (Ordenar películas). Y posteriormente en la siguiente versión se implementarán los casos de uso CU-05 (Activar notificación) y CU-10 (Desactivar

notificación). El diseño de la pantalla de los últimos dos casos de uso se puede ver en la siguiente figura, ya que los casos de uso se encuentran en el proceso del desarrollo.

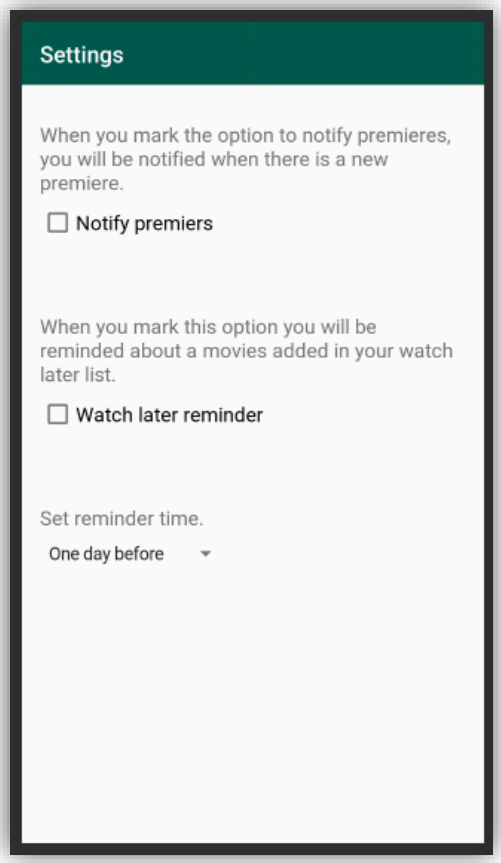

Ilustración 68: Ajustes

En futuras versiones el autor tiene pensado incorporar funcionalidades como por ejemplo ver los *trailers*, si están disponibles, añadir cuenta de usuario y diseñar las vistas para el modo apaisado, actualmente en desarrollo como se puede ver en la figura siguiente.

Ilustración 69: Pantalla en modo apaisado

Otra funcionalidad importante desde el punto de vista del autor, después de incorporar las cuentas de usuario, sería poder comentar la película y valorarla.

Cabe destacar que otras mejoras y funcionalidades dependerán también de las opiniones y sugerencias de los usuarios de la aplicación y por lo tanto tendrán prioridad unas respecto a otros.

9. Referencias

[1] DataArt. (9 de Diciembre de 2015). *Model-View-ViewModel Pattern*. Obtenido de Habr: https://habr.com/en/company/dataart/blog/272737/ [Última consulta: 15 de Marzo de 2019]

[2] Developers, A. (15 de Junio de 2019). *Documentación para desarrolladores de apps*. Obtenido de https://developer.android.com/docs [Última consulta: 25 de Junio de 2019]

[3] Developers, A. (s.f.). *Sugerencias sobre el rendimiento by Android Developers*. Obtenido de https://developer.android.com/training/articles/perf-tips [Última consulta: 28 de Abril de 2019]

[4] Google. (2018). *Google Samples*. Obtenido de GitHub: https://github.com/googlesamples/android-architecture-components [Última consulta: 15 de Junio de 2019]

[5] Gorman, L. O. (6 de Febrero de 2019). *Movie Manager*. Obtenido de https://play.google.com/store/apps/details?id=com.leogorman.moviemanager [Última consulta: 03 de Marzo de 2019]

[6] Griffiths, D., & Griffiths, D. (2017). *Head First Android Development*. O'Reilly Media. [Última consulta: 1 de Junio de 2019]

[7] Jemerov, D., & Isakova, S. (2016). *Kotlin in Action*. Manning Publications. [Última consulta: 29 de Abril de 2019]

[8] *Kotlin Programming Language*. (s.f.). Obtenido de https://kotlinlang.org/docs/reference/ [Última consulta: 23 de Abril de 2019]

[9] *LiveData*. (28 de Diciembre de 2017). Obtenido de StartAndroid: https://startandroid.ru/ru/courses/dagger-2/27-course/architecture-components/525-urok-2-livedata.html [Última consulta: 20 de Marzo de 2019]

[10] Moqups. (s.f.). *Moqups*. Obtenido de https://moqups.com [Última consulta: 7 de Abril de 2019]

[11]Muntenescu, F. (21 de Mayo de 2018). *Android Jetpack: Paging*. Obtenido de YouTube: https://www.youtube.com/watch?v=QVMqCRs0BNA [Última consulta: 24 de Abril de 2019]

[12] Murthy, A. (02 de Julio de 2018). *7 steps to implement Paging library in Android*. Obtenido de Medium: https://proandroiddev.com/8-steps-to-implement-paging-library-in-android-d02500f7fffe [Última consulta: 2 de Junio de 2019]

[13] *Patron de diseño Observador*. (s.f.). Obtenido de Programación reactiva: https://reactiveprogramming.io/books/design-patterns/es/catalog/observer [Última consulta: 1 de Abril de 2019]

[14] ReactiveX. (s.f.). *ReactiveX*. Obtenido de http://reactivex.io/documentation [Última consulta: 5 de Junio de 2019]

[15] Training, M. f. (1 de Noviembre de 2017). *Movie Manager*. Obtenido de https://play.google.com/store/apps/details?id=noh.android.moviemanager [Última consulta: 03 de Marzo de 2019]