Diseño arquitectónico de la aplicación

Simone Salvatore La Milia
Nizan Díaz Marcos
Alejandro Matías Cruz de la Cruz

1. Definición del dominio:

Nos disponemos a desarrollar una aplicación para Android que permitirá a los usuarios llevar un registro de las películas, series y programas de televisión que han visto o desean ver. Podrán buscar y filtrar entre los millones de registros que contiene la base de datos The Movie Database, a la cual accederá nuestra aplicación por medio de su API pública.

Las principales características de la aplicación son las siguientes:

- Búsqueda de películas series y programas de televisión por nombre.
- Filtrado de contenido según su tipo (película, serie, programa de televisión, documental, …)
- Filtrado de contenido según categorías (acción, drama, comedia, …)
- Se podrá guardar contenido en la sección de "por ver".
- Se podrá guardar contenido en la sección de "vistos".
- Las películas por ver y vistas se guardarán en la base de datos local del dispositivo, para que su información pueda ser accedida incluso sin conexión a Internet.

2. Identificación de los principales elementos de la arquitectura escogida:

La arquitectura que aplicaremos en el desarrollo de nuestra aplicación será la conocida MVVM (*Model-View-ViewModel* o Modelo-Vista-Modelo de vista), variación de la también popular MVP (*Model View Presenter* o Modelo-Vista-Presentador). Como ocurre en otras arquitecturas, el principal objetivo de la MVVM es desacoplar lo máximo posible la vista de la lógica de negocio. Para ello, la aplicación contará con tres partes muy diferenciadas entre sí:

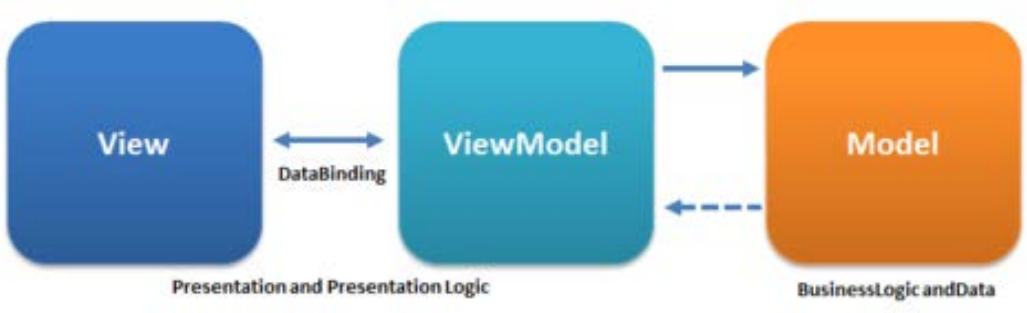
- Modelo. Representa la capa de datos y la lógica de negocio. Es completamente independiente de la vista y el modelo de vista (*ViewModel*) que lo manipula. Si bien es cierto que, de manera implícita, el modelo de vista observa los cambios en el modelo, esto no implica que el modelo sea dependiente, dado que no puede "ver" al modelo de vista que "observa" dichos eventos.
- Vista. Determina cómo se muestra en pantalla la información para que pueda ser interpretada por los usuarios. Se encarga también, por supuesto, de detectar la interacción del usuario con la aplicación y de delegar al *ViewModel* el manejo de la información introducida por el mismo. La vista depende fundamentalmente del modelo de vista.
- Modelo de vista (*ViewModel*). Es una abstracción de la vista. A diferencia del controlador en la arquitectura MVC (*Model-View-Controller* o Modelo-Vista-Controlador) y el presentador en la MVP, MVVM contiene implícitamente un *binder* o "enlazador", que automatiza la comunicación entre la vista y el *ViewModel*. Además, el *ViewModel* no cuenta con una referencia a la vista, como sucede en la MVP. En su lugar, la vista es enlazada mediante *data binding* con las propiedades declaradas en el *ViewModel* y puede acceder a ellas sin necesidad de escribir una gran cantidad de código.

En el caso de nuestra aplicación, el *data binding* lo llevará a cabo Android automáticamente, gracias a la clase *ViewModel*, a partir de la cual se pueden crear modelos de vista fácilmente enlazables a la vista.

Por otra parte, el modelo consistirá en una serie de *data classes* de Kotlin, que definirán la estructura de los datos que se consumirán de la API externa, así como el formato de los registros en la base de datos local.

Por último, la vista se implementará utilizando *Jetpack Compose*, el kit de herramientas que actualmente se utiliza para desarrollar aplicaciones Android. Este permite dividir la vista en componentes llamados *composables*, que consisten en funciones de Kotlin.

3. Diseño de la arquitectura:



4. Caso de uso:

ID: 1
Descripción: Añadir el audiovisual seleccionado a la sección de “vistos”.
Precondiciones: • El usuario se encuentra en la pantalla de detalles de un audiovisual. • Dicho audiovisual no se encuentra en la sección de “vistos”.
Postcondiciones: • El sistema muestra una confirmación indicando que el audiovisual se ha añadido a la sección de “vistos”. • El audiovisual se almacena en la base de datos local del dispositivo. • El audiovisual se mostrará cuando el usuario acceda a la pantalla de “vistos”.
Flujo normal: 1. El usuario introduce en el buscador palabras clave para buscar una película, serie, programa de televisión, documental, … 2. El sistema muestra una lista con los resultados que se ajustan al texto introducido por el usuario. 3. El usuario selecciona uno de los resultados. 4. El sistema muestra la pantalla de detalles del resultado seleccionado.

5. El usuario pulsa el botón para añadir el audiovisual a la sección de “vistos”.
6. El sistema almacena el audiovisual en la base de datos local y confirma la acción.

Implementación en arquitectura MVVM:
La funcionalidad se divide en tres partes: modelo, vista y modelo de vista. En este caso, el modelo estaría compuesto por dos clases, una que define el formato de los datos que se consumen de la API externa y otra que determina el formato de los datos que se almacenan en la base de datos local.

La vista, por otra parte, está formada por la pantalla de detalles, en la cual se encuentra el usuario, además de, a su vez, todos los *composables* que la integran, incluyendo el botón para añadir a la sección de “vistos”. También formará parte de la vista de este caso de uso la pantalla de “vistos” y todos sus componentes. Adicionalmente, la vista contará con elementos que darán respuesta al usuario cuando este solicite realizar la acción descrita.

Por último, el modelo de vista se encargará de enlazar los datos, cuya forma la dicta el modelo, con la representación gráfica dispuesta por la vista. Por ello, se encargará de llevar a cabo los procedimientos necesarios para que, cuando el usuario pulse el botón para añadir a la sección de “vistos”, la información del audiovisual seleccionado se guarde en la base de datos local y se muestre cuando el usuario navegue a la pantalla de “vistos”. Además, el modelo de vista hará que la vista comunique al usuario que la acción que este solicitó se ha realizado correctamente.

5. Conclusión:

Tal y como se mencionó anteriormente, la principal ventaja de utilizar MVVM es la gran cantidad de herramientas que ofrece Android para trabajar con este tipo de arquitectura. Gracias al *ViewModel* y el enlace de datos, es realmente sencillo enlazar la vista con la lógica de la aplicación en pocas líneas de código.

Por otra parte, no nos hemos decantado por arquitecturas más complejas y escalables, como la arquitectura hexagonal, debido a que nuestro propósito es dedicar las fases iniciales del proyecto en su totalidad al desarrollo de la funcionalidad. Dichas arquitecturas requieren de una cantidad considerable de preparación y de una muy cuidada planificación previa, con el fin de desacoplar la estructura de la aplicación de los *frameworks* y herramientas utilizados para desarrollarla, mientras que nuestro proyecto sacará el máximo partido de dichas herramientas, sacrificando algo de independencia en el proceso.