29-4-2023

Universidad Don Bosco

Trabajo de investigación

Desarrollo de software para móviles

03L

Alumno:

Martínez Sanabria, David Ezequiel | MS180761

Alexander Alberto Siguenza Campos

UNIVERSIDAD DON BOSCO

Contenido

Introducción

El patrón de arquitectura MVVM (Model-View-ViewModel) es una de las opciones más populares para desarrollar aplicaciones en Android. Este patrón se enfoca en separar la lógica de presentación de los datos de la aplicación, lo que resulta en una estructura de código más clara y mantenible. Además, Google recomienda su uso para desarrollar aplicaciones en Android, lo que lo hace aún más relevante en el ecosistema Android. Con MVVM, el código se divide en tres componentes principales: el modelo, la vista y el modelo de vista, lo que permite un mejor manejo de los datos y una separación clara de las responsabilidades.

Patrón MVVM

¿Qué es?

Patrón de arquitectura que ayuda a separar limpiamente la lógica de presentación y negocios de una aplicación de su interfaz de usuario (UI); Esto ayuda a abordar numerosos problemas de desarrollo y facilita la prueba, el mantenimiento y la evolución de una aplicación. El patrón consta de 3 tres componentes principales:

- View (vista)
- View Model (modelo de la vista)
- Model (modelo)

Componentes de la Arquitectura

El patrón MVVM está compuesto principalmente por 3 capas: View-ViewModel-Model; la capa View se comunica con la capa Model a través de la capa ViewModel

View

Es la capa responsable de mostrar la apariencia de lo que el usuario ve e interactúa en la pantalla, en esta parte de la capa se pueden ubicar todas las *Activities* y *Fragments* de una aplicación de Android. Esta capa se comunica con la capa *ViewModel* mediante eventos realizados por los componentes de la UI (cuando se le da clic a un botón, cuando se tiene pulsado un botón, cuando se tecla en un input, entre otros) y esta recibe la respuesta de dichos eventos por parte del ViewModel.

Model

Es la capa que se encargará de proveer los datos a nuestra aplicación sin importar la fuente de origen de estas (base de datos local, consumo de una graphql-API o restAPI, share-preferences, entre otros). Las clases de *Model* se suelen usar junto con servicios o repositorios que encapsulan el acceso a datos y el almacenamiento en caché.

ViewModel

Es la capa que permite realizar la comunicación entre la capa *View* y la capa *Model*. La capa *View* le notifica a *ViewModel* que el usuario a hecho una interacción, *ViewModel* solicita la información a la capa *Model* y cuando el *Model* ya tiene los datos solicitados, *ViewModel* se los envía a *View* para que la pantalla en la que se encuentra el usuario sea actualizada y pueda reflejar los cambios (permitiendo al usuario que su acción efectivamente tuvo un efecto en la aplicación).

¿Cómo se aplica en Android con Kotlin?

Para poder utilizar este patrón de arquitectura se necesita principalmente viewModel de la biblioteca lifecycle (dentro build.gradle de nivel de módulo).

```
val lifecycle_version = "2.5.1"

// ViewModel
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version ")
```

Además, necesita agregar el paquete *LiveData* que facilita la unión entre la capa *View* y la capa *ViewModel*

```
// LiveData
implementation("androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version")
```

También se recomienda la siguientes librerías que permite trabajar cómodamente en activities o fragments

```
// Fragment
implementation "androidx.fragment:fragment-ktx:1.5.5"
// Activity
implementation "androidx.activity:activity-ktx:1.6.1"
```

Además de eso se recomienda enormemente que se utilice el *viewBinding* para una mejor experiencia.

Teniendo esto, se recomienda ordenar el proyecto en paquetes donde se pueda reflejar las utilidades de las clases (en anexo se agrega un ejemplo al respecto).

Ventajas y Desventajas

Ventajas

- Facilita la creación de pruebas unitarias y las pruebas de integración.
- Separa la lógica del negocio y la presentación de la interfaz del usuario; facilitando el mantenimiento y la escalabilidad de la aplicación.
- Mejora la reusabilidad del código, ya que promueve la modularidad de los componentes.

Desventajas

- La curva de aprendizaje puede ser alta para los desarrolladores que están acostumbrados a trabajar con otros patrones
- Puede aumentar la complejidad en la estructura de archivos y carpetas; es peor una mala abstracción que no tenerla
- Para proyectos pequeños puede resultar una sobrecargar innecesaria; la complejidad del patrón puede retrasar el desarrollo de la aplicación.

Anexo

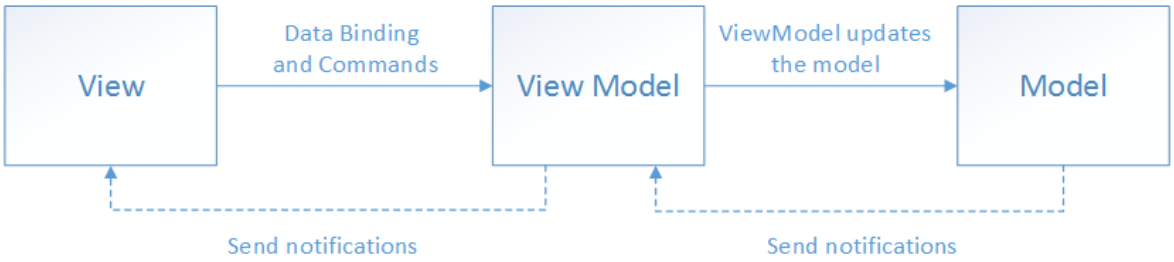

Diagrama que muestra las relaciones que tiene cada capa en el patrón MVVM

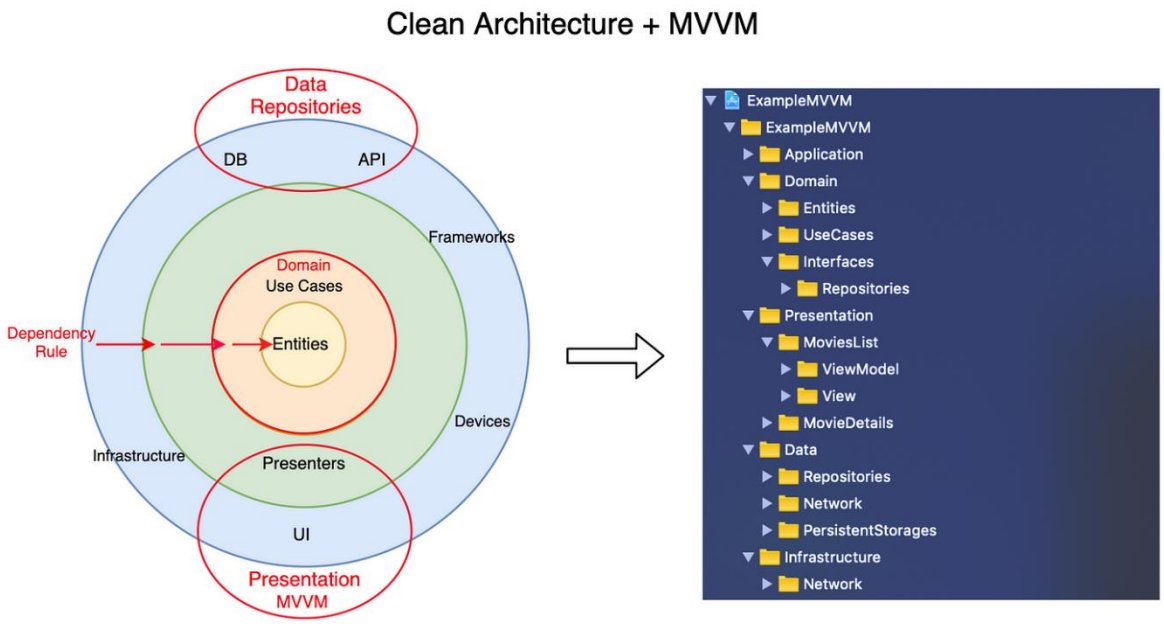

Diagrama y estructura de carpeta que reflejan el patrón MVVM junto con el patrón Clean Architecture

Bibliografía

Stonis, M., [michaelstonis], Jain, T., [erjain], & Pine, D., [IEvangelist]. (2022, 28 noviembre). Model-View-ViewModel (MVVM). Microsoft Learn. Recuperado 22 de abril de 2023, de https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm

Tyagi, A. (2018). Better Android Apps Using MVVM With Clean Architecture. Toptal Engineering Blog. https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture

Reyes, L. (2021, 7 diciembre). Aplicando el patrón de diseño MVVM - Leomaris Reyes - Medium. Medium. https://medium.com/@reyes.leomaris/aplicando-el-patr%C3%B3n-de-dise%C3%B1o-mvvm-d4156e51bbe5