FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACIÓN

Asignatura: Desarrollo de Software para Móviles (DSM104)

CICLO ACADEMICO: 01-2023

Título:

"Desafío Práctico 03"

Docente:

Ing. Alexander Alberto Sigüenza Campos.

Presentado por:

Apellidos, Nombres	Carné	Carrera	Laboratorio
Granados González, Samuel Alejandro	GG150384	Ingeniería en Ciencias de la Computación	03L

Soyapango, 29 de abril de 2023.

INDICE

INTRODUCCIÓN

El patrón Model-View-ViewModel (MVVM) se ha convertido en uno de los patrones de arquitectura de software más populares en el desarrollo de aplicaciones móviles Android con Kotlin. Este patrón ayuda a separar la lógica de presentación de la lógica de negocio y de acceso a datos, lo que simplifica el proceso de desarrollo y mejora la calidad y mantenibilidad del código.

En esta investigación, se explorará el patrón MVVM en profundidad, incluyendo los componentes principales del patrón y cómo se implementa en Android con Kotlin. También se analizarán las ventajas y desventajas de utilizar este patrón en el desarrollo de aplicaciones móviles.

Al final de esta investigación, los desarrolladores de aplicaciones móviles podrán comprender el patrón MVVM y cómo pueden utilizarlo para crear aplicaciones móviles más eficientes y mantenibles para Android.

CONTENIDO

Patrón MVVM

Para establecer la definición del patrón MVVM, se segmentará en los siguientes apartados:

Patrón: Definición

Un patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se puede aplicar esta solución un millón de veces sin hacer lo mismo dos veces[1].

Patrón de diseño: Definición

En la ingeniería de software, un patrón de diseño es una solución repetible y general para problemas de ocurrencia cotidianos en el diseño de software. Es importante aclarar que un patrón de diseño no es un diseño de software terminado y listo para codificarse, más bien es una descripción o modelo de cómo resolver un problema que puede utilizarse en diferentes situaciones.

Por lo tanto, los patrones de diseño permiten agilizar el proceso de desarrollo de solución debido a que proveen un paradigma desarrollado y probado.

Patrón MVVM: Definición

MVVM, por sus siglas en ingles Model View ViewModel, es un patrón de diseño que tiene por finalidad separar la parte de la interfaz de usuario (Vista) de la parte de la lógica del negocio (Modelo), logrando así que la parte visual sea totalmente independiente. El otro componente es el ViewModel que es la parte que va a interactuar como puente entre la Vista y el Modelo[2].

El patrón MVVM ayuda a separar limpiamente la lógica de presentación y negocios de una aplicación de su interfaz de usuario (UI). Mantener una separación limpia entre la lógica de la aplicación y la interfaz de usuario ayuda a abordar numerosos problemas de desarrollo y facilita la prueba, el mantenimiento y la evolución de una aplicación. También puede mejorar significativamente las oportunidades de reutilización del código y permite a los desarrolladores y diseñadores de interfaz de usuario colaborar más fácilmente al desarrollar sus respectivas partes de una aplicación.

[1] Christopher Alexander
[2] Daniela Ortiz

Componentes principales del Patrón MVVM

Con este patrón, la aplicación tiene tres componentes principales:

View

Es la interfaz de usuario (IU), formada por las activities, los fragments, los archivos XML y elementos auxiliares para mostrar los datos como por ejemplo las clases adaptadoras para las listas. Estas clases deben ejecutar la menor lógica posible, siendo su mayor responsabilidad capturar eventos recibidos por parte del usuario y mostrar los datos, pero no la manera en la que se obtienen esos datos.

ViewModel

Es el encargado de obtener los datos y enviarlos a las vistas. Siguiendo el patrón observador, cada vista se suscribe a los datos deseados de su respectivo viewmodel, y estos cuando hay un cambio, lo notifican a la vista.

Model

Contiene la lógica de negocio. Es el componente en la que se almacena los datos en modelos o clases de datos, obtenidos al realizar llamadas a la base de datos o a un servicio web.

Para observar los datos obtenidos por los view models, se utiliza Live Data. Este es una clase contenedora de datos, observable y optimizada para los ciclos de vida de las actividades o fragmentos de la aplicación. Esta optimización asegura que los únicos datos que se actualizan para ser observados son los de los componentes de la aplicación cuyo ciclo de vida se encuentra en estado activo.

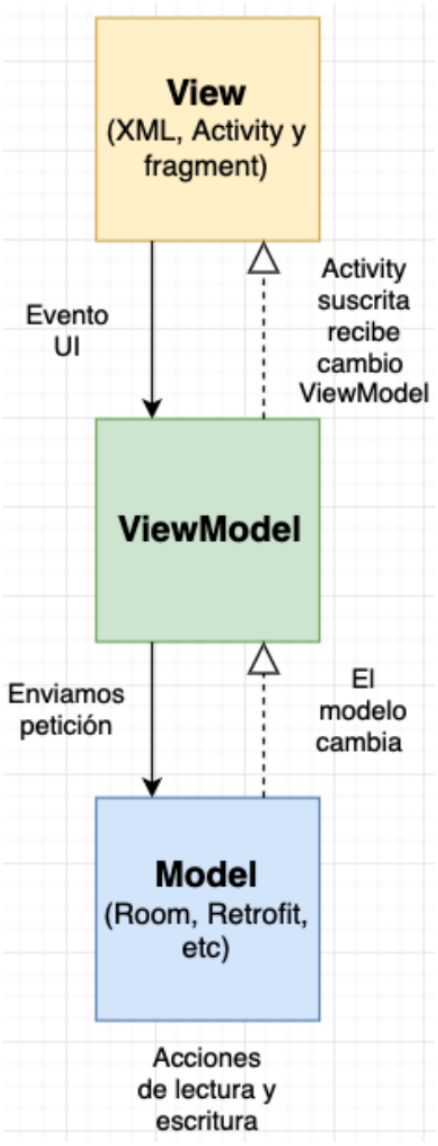

Ilustración 1: Relación entre los componentes del patrón MVVM

Patrón MVVM usando Android y Kotlin

Para implementar el patrón MVVM en Android con Kotlin, se pueden seguir los siguientes pasos:

- Crear el modelo:

Crear una clase que represente los datos y la lógica de negocio de la aplicación.

- Crear el ViewModel:

Crear una clase que extienda la clase ViewModel y se encargue de manejar los datos y la lógica de presentación. El ViewModel debe exponer los datos como objetos Observable, que pueden ser observados por la vista.

- Crear la vista:

Crear una actividad, un fragmento o una vista personalizada que se encargue de mostrar la interfaz de usuario al usuario final. La vista debe observar los cambios en los objetos Observable expuestos por el ViewModel y actualizar la interfaz de usuario en consecuencia.

- Conectar la vista con el ViewModel:

La vista debe conectarse con el ViewModel para obtener los datos y la lógica de presentación. Esto se puede hacer mediante la inyección de dependencias o mediante la creación de una instancia del ViewModel en la vista.

- Actualizar el modelo:

Cuando el usuario realiza una acción en la vista, como hacer clic en un botón, la vista debe actualizar el modelo mediante el ViewModel. El ViewModel se encarga de procesar la acción y actualizar los datos en consecuencia.

Otros elementos que pueden ayudar dentro del proyecto son los siguientes:

- LiveData:

Una forma común de exponer los datos del ViewModel a la vista es mediante el uso de LiveData. LiveData es una clase observable que se actualiza automáticamente cuando cambian los datos en el ViewModel. Esto significa que la vista puede observar los cambios en los datos y actualizarse en consecuencia.

- ViewModelFactory:

La clase ViewModelFactory se utiliza para crear instancias de ViewModel en la vista. Al utilizar una ViewModelFactory, se puede inyectar dependencias en el ViewModel y asegurarse de que se mantengan durante el ciclo de vida de la vista.

- DataBinding:

DataBinding es una biblioteca que permite conectar directamente los elementos de la interfaz de usuario en la vista con los datos del ViewModel. Esto reduce la cantidad de código necesario para actualizar la interfaz de usuario y hace que la aplicación sea más fácil de mantener.

- Repository:

Es una buena práctica tener un repositorio que se encargue de interactuar con los datos del modelo. El repositorio se encarga de la lógica de acceso a los datos, lo que significa que el ViewModel no tiene que preocuparse por el almacenamiento y recuperación de los datos. El repositorio también puede actuar como una capa de abstracción entre el ViewModel y los datos subyacentes, lo que permite una fácil modificación de la fuente de datos sin afectar el ViewModel.

- Patrón Observer:

En el patrón MVVM, la vista observa los cambios en los datos del ViewModel. Esto se puede lograr utilizando el patrón Observer. La vista se suscribe a los objetos LiveData expuestos por el ViewModel y se actualiza automáticamente cuando cambian los datos.

- Dependency Injection:

El uso de Dependency Injection (DI) es una práctica recomendada en Android. Permite una fácil configuración y gestión de las dependencias de la aplicación. Puede usar la biblioteca Dagger 2 para implementar la inyección de dependencias en su aplicación.

- Separar la lógica de la vista y el ViewModel:

Es importante que la lógica de la vista y el ViewModel se mantengan separados. La vista debe ser responsable de mostrar los datos, mientras que el ViewModel debe ser responsable de manejar los datos y la lógica de presentación. Esto permite una fácil modificación y mantenimiento de la aplicación.

Ventajas y desventajas de utilizar el patrón MVVM en el desarrollo de aplicaciones móviles

Ventajas

- Está recomendado por Google y debido a su estructura de implementación prácticamente está hecho para Android.
- Posee herramientas para manejar el lifecycle de los activities y fragments.
- Logra la separación de intereses dentro del aplicativo.
- Tanto el Model como el ViewModel son partes comprobables por unidad.
- Resuelve el problema del MVC cuando hay exceso de código en el Modelo o la Vista.
- Los desarrolladores pueden crear pruebas unitarias para el modelo de vista y el modelo, sin usar la vista. Las pruebas unitarias del modelo de vista pueden ejercer exactamente la misma funcionalidad que la vista.
- Los diseñadores y desarrolladores pueden trabajar de forma independiente y simultánea en sus componentes durante el proceso de desarrollo. Los diseñadores pueden centrarse en la vista, mientras que los desarrolladores pueden trabajar en el modelo de vista y los componentes del modelo.
- Se obtiene un código más limpio y organizado.

Desventajas

- No es tan replicable como el MVP, en cada Activity o Fragment donde se utilice, se tienen que contemplar cambios que en otros patrones no son necesarios.
- Se debe adaptar a una estructura predefinida, lo cual puede incrementar la complejidad del sistema.
- La distribución de componentes obliga a crear y mantener un mayor número de ficheros.
- La curva de aprendizaje para los desarrolladores que no han trabajado con este patrón se estima mayor que la de otros patrones.

ANEXOS

Enlace al repositorio:

https://github.com/ggsgranados/Desafio3DSM

Enlace a infografía resumen:

https://www.canva.com/design/DAFhIsDYvtM/U47dSWdfT5I7PrBDfC_tww/view?utm_content=DAFhIsDYvtM&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink

BIBLIOGRAFÍA

Crespo, D. (s.f.). *DESARROLLO DE UNA APLICACIÓN ANDROID PARA CROSSROADS 2.0, UN JUEGO EDUCATIVO PARA CONCIENCIAR SOBRE EL CAMBIO CLIMÁTICO*. https://uvadoc.uva.es/bitstream/handle/10324/57250/TFG-5793.pdf?sequence=1&isAllowed=y

Loor, C. (2015). *Desarrollo e implementación de un sistema para la gestión y control de los recursos utilizados en proyectos de investigación de naturaleza estadística*. https://dspace.ups.edu.ec/bitstream/123456789/10327/1/UPS-GT001236.pdf

KeepCoding. (29 de agosto de 2022). *Pasos para la implementación de MVVM en Android*. KeepCoding Bootcamps. https://keepcoding.io/blog/pasos-para-la-implementacion-de-mvvm-en-android/

Reyes, L. (21 de diciembre de 2018). *Aplicando el patrón de diseño MVVM - Leomaris Reyes - Medium*. Medium; Medium. https://medium.com/@reyes.leomaris/aplicando-el-patr%C3%B3n-de-dise%C3%B1o-mvvm-d4156e51bbe5

michaelstonis. (28 de noviembre de 2022). *Modelo-Vista-Modelo de vista*. Microsoft.com. https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm

Patrón MVVM. (2023). Scribd. https://es.scribd.com/document/560697209/Patron-MVVM#

Nakayama, A., & Solano, J. A. (s.f.). *Guía práctica de estudio 13: Patrones de diseño.* http://profesores.fi-b.unam.mx/annkym/LAB/poo_p13.pdf