UNIVERSIDAD DON BOSCO

PATRÓN DE ARQUITECTURA MVVM

Desarrollo de Software para Móviles

CATEDRÁTICO: Ing. Alexander Alberto Sigüenza Campos

PRESENTADO POR:

Pacheco Guerrero, Gabriela Saraí PG180134
García López, Francisca Abigail GL180669

ENLACE A EJEMPLO MVVM:

https://github.com/Aby3005/PRACTICO3_DSM.git

ÍNDICE

INTRODUCCIÓN 3

¿QUÉ ES EL PATRÓN MVVM? 4

COMPONENTES PRINCIPALES 5

¿CÓMO SE APLICA EL PATRÓN MVVM? 6

VENTAJAS Y DESVENTAJAS DE UTILIZAR EL PATRÓN MVVM 7

ANEXOS 8

BIBLIOGRAFÍA 9

INTRODUCCIÓN

El patrón Modelo-Vista-Vista-Modelo, o MVVM por sus siglas en inglés, es un patrón de arquitectura de software que se ha popularizado en los últimos años gracias a su capacidad para separar claramente la lógica de negocio de la lógica de presentación en aplicaciones de escritorio y móviles. El patrón MVVM se basa en dos conceptos fundamentales: la separación de responsabilidades y la vinculación de datos bidireccional. Al separar las tareas de presentación y de lógica de negocio, el patrón MVVM facilita la mantenibilidad, la escalabilidad y la modularidad de las aplicaciones. A su vez, la vinculación de datos bidireccional permite que los cambios realizados en la vista se reflejen automáticamente en el modelo subyacente, y viceversa, lo que aumenta la eficiencia del desarrollo y la calidad final del producto. Esta investigación examina los aspectos teóricos y prácticos del patrón MVVM, sus ventajas y desventajas, sus componentes principales y como se relacionan entre sí, y un ejemplo de su uso en aplicaciones reales.

¿QUÉ ES EL PATRÓN MVVM?

El patrón de arquitectura MVVM, también conocido como Model View ViewModel, se refiere a un modelo de diseño que tiene el objetivo para llevar a cabo la separación del apartado de la interfaz de usuario (View) de la parte lógica (Model). Esto lo hace con el objetivo de que el aspecto visual sea completamente independiente.

El patrón Model-View-ViewModel (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.

Con el patrón MVVM, la interfaz de usuario de la aplicación y la presentación subyacente y la lógica de negocios se separan en tres clases independientes: la vista, que encapsula la lógica de la interfaz de usuario y la interfaz de usuario; el modelo de vista, que encapsula la lógica y el estado de la presentación; y el modelo, que encapsula la lógica de negocios y los datos de la aplicación.

¿CUÁLES SON SUS COMPONENTES PRINCIPALES Y CÓMO SE RELACIONAN ENTRE SÍ?

Hay tres componentes principales en el patrón MVVM: el modelo, la vista y el modelo de vista. Cada uno sirve para un propósito distinto.

- **El modelo**

Representa la capa de datos y/o la lógica de negocio, también denominado como el objeto del dominio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.

- **La vista**

La misión de la vista es representar la información a través de los elementos visuales que la componen. Las vistas en MVVM son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente. En Xamarin Forms podemos crear nuestras interfaces a través de código C# o XAML.

- **Modelo de vista (ViewModel)**

El ViewModel (modelo de vista) es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre la vista y el viewmodel se realiza por medio los enlaces de datos (binders).

¿CÓMO SE APLICA EL PATRÓN MVVM EN ANDROID CON KOTLIN?

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

1. **Crear un modelo:** Un modelo es un componente básico que representa los datos y la lógica empresarial de una aplicación. En Kotlin, podemos crear una clase de datos que contenga las propiedades y los métodos que necesita nuestro modelo.
2. **Crear una vista:** Una vista es la parte que representa la interfaz de usuario de una aplicación. En Android, podemos crear acciones o clips que implementen vistas. La apariencia de la interfaz se puede definir en el archivo XML asociado.
3. **Crear un modelo de vista:** El modelo de vista es la parte que actúa como intermediario entre el modelo y la vista. En Kotlin, podemos crear una clase que amplíe la clase ViewModel y contenga los datos y métodos que necesita nuestro modelo de vista.
4. **Conectar la vista con el modelo de vista:** Debemos crear una instancia de un modelo de vista en una actividad o fragmento y ver cómo cambian los datos con LiveData.

Estos pasos garantizan que la implementación del patrón MVVM en Android con Kotlin sea de manera efectiva, lo cual permite una mejor separación de responsabilidades y hace que la aplicación sea más fácil de mantener y probar.

¿CUÁLES SON LAS VENTAJAS Y DESVENTAJAS DE UTILIZAR EL PATRÓN MVVM EN EL DESARROLLO DE APLICACIONES MÓVILES?

El patrón MVVM (Model-View-ViewModel) tiene varias ventajas y desventajas al utilizarse en el desarrollo de aplicaciones móviles. Algunas de ellas son:

VENTAJAS:

- **Separación de responsabilidades:** El patrón MVVM permite una clara separación de responsabilidades entre el modelo, la vista y el view-model, lo que hace que el código sea más fácil de entender y mantener.
- **Testing:** El patrón MVVM facilita la realización de pruebas unitarias, ya que las funciones y métodos del ViewModel son independientes del resto del código.
- **Reutilización de código:** Al utilizar el patrón MVVM, el código se puede reutilizar en diferentes partes de la aplicación o en aplicaciones distintas.
- **Gestión del estado:** El patrón MVVM permite una gestión más sencilla del estado de la aplicación, ya que el viewModel se encarga de ello.

DESVENTAJAS:

- **Curva de aprendizaje:** El patrón MVVM puede ser un poco difícil de entender al principio, lo que puede ser un obstáculo para desarrolladores nuevos en el trabajo con el patrón.
- **Mayor complejidad:** El patrón MVVM añade una capa adicional de complejidad al código de la aplicación, lo que puede aumentar el tiempo necesario para el desarrollo de la aplicación.
- **Alto consumo de memoria:** Algunos desarrolladores han comentado que el patrón MVVM puede consumir bastante memoria, especialmente si la aplicación es grande y compleja.
- **Conexión entre la vista y el ViewModel:** La conexión entre la vista y el ViewModel puede ser complicada en ocasiones, lo que puede dificultar el proceso de depuración y mantenimiento del código.

En general, el patrón MVVM puede ser muy beneficioso en el desarrollo de aplicaciones móviles debido a la separación de responsabilidades y la gestión del estado de la aplicación que ofrece, pero también puede presentar algunos desafíos en términos de complejidad y curva de aprendizaje.

ANEXOS

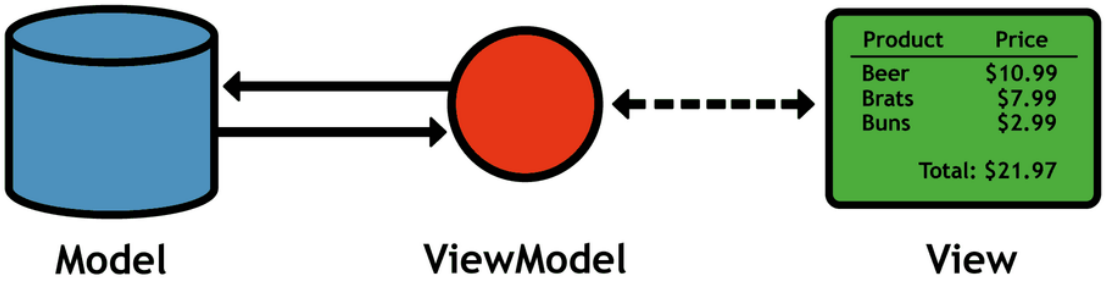

Imagen 1. Componentes principales del patrón MVVM

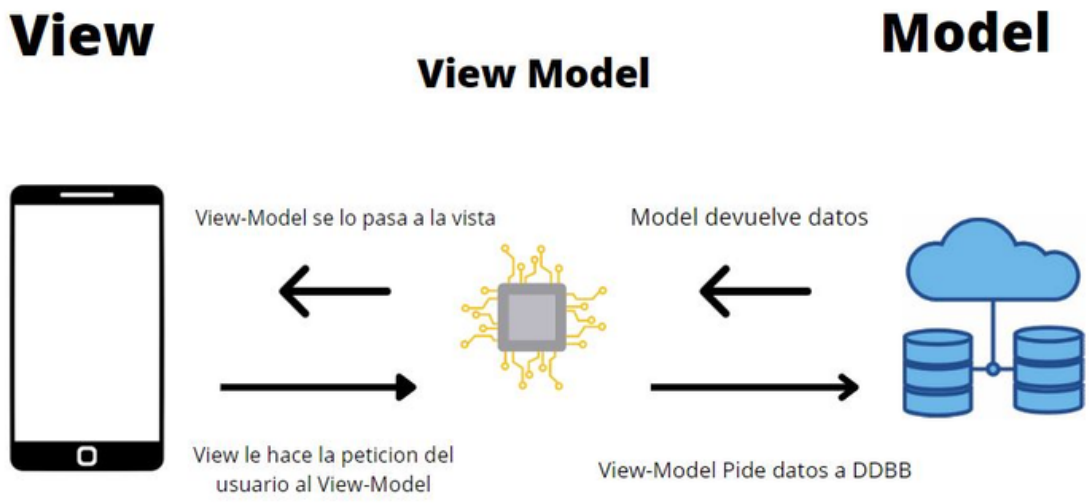

Imagen 2. Implementación del patrón MVVM en Android con Kotlin

BIBLIOGRAFÍA

https://www.facebook.com/grokkeepcoding. (2022, August 29). Pasos para la implementación de MVVM en Android. KeepCoding Bootcamps. https://keepcoding.io/blog/pasos-para-la-implementacion-de-mvvm-en-android/

Reyes, L. (2018, December 21). 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

Camero, S. (2020, September 24). Arquitecturas de Software. MVC y MVVM - Abatic Soluciones Tecnológicas. Abatic Soluciones Tecnológicas. https://www.abatic.es/arquitecturas-de-software-mvc-y-mvvm/

Andres Felipe Ocampo. (2021, September 28). VIPER Vs MVVM, el patrón Arquitectónico lo es todo. Medium; Medium. https://andresfelipeocampo.medium.com/viper-vs-mvvm-el-patr%C3%B3n-arquitect%C3%B3nico-lo-es-todo-79162ac3d6d1

Team, K. (2023, April 21). ¿Qué es el patrón de arquitectura MVVM? KeepCoding Bootcamps. https://keepcoding.io/blog/que-es-el-patron-de-arquitectura-mvvm/#:~:text=El%20patr%C3%B3n%20de%20arquitectura%20MVVM%2C%20tambi%C3%A9n%20conocido%20como%20Model%20View,la%20parte%20l%C3%B3gica%20(Model).

▷ Xamarin Forms: el patrón MVVM【 2020 】. (n.d.). https://softwarecrafters.io/xamarin/patron-mvvm-xamarin-forms