Asignatura: Desarrollo de Software para Móviles

Grupo: G04L

Docente: Alexander Alberto Sigüenza Campos

Nombre	Apellido	Carné
Carlos Emerson	Portillo Martínez	PM191144
David Alexander	Aguilar Panameño	AP190139
Alejandro José	Castillo Pacheco	CP190719

Índice

Tabla de contenido

Introducción

En el mundo del desarrollo de software, la separación de responsabilidades es un principio fundamental que permite crear aplicaciones escalables, mantenibles y fáciles de entender. Para lograr este objetivo, existen diferentes patrones de arquitectura que ofrecen una guía para organizar el código de una manera eficiente y coherente.

Uno de los patrones de arquitectura más populares en el desarrollo de aplicaciones móviles y de escritorio es el patrón MVVM (Model-View-ViewModel). MVVM es una evolución del patrón MVC (Model-View-Controller) y se enfoca en la separación de responsabilidades entre la interfaz de usuario, la lógica de negocio y los datos de la aplicación.

En este trabajo de investigación, se explorará en profundidad el patrón MVVM, su estructura, su aplicación en kotlin y su implementación en diferentes plataformas. Además, se discutirán las ventajas y desventajas de utilizar este patrón en comparación con otros patrones de arquitectura y se analizarán algunos casos de uso en los que MVVM es especialmente adecuado.

El objetivo final de este trabajo es ofrecer una comprensión completa del patrón MVVM y su aplicación práctica en el desarrollo de aplicaciones móviles y de escritorio.

¿Qué es el patrón MVVM?

MVVM o por sus siglas **Model View ViewModel,** es una arquitectura utilizada para la creación productos de programación y mayormente para su utilización en la realización de aplicaciones móviles desde que Google convirtió dicho modelo en su arquitectura predilecta sobre la cual guiarse e iniciarse en el mundo del desarrollo Mobile.

Este modelo se encarga de realizar la separación lógica de la funcionalidad y la presentación entre la de manejo de información, esto realizado de manera que permita un rango de opciones representando una mejora en el desarrollo de aplicaciones, haciendo que el código empleado para la realización de este tenga mayores facilidades en sustentabilidad, calidad funcional y escalabilidad de este.

Es bien sabido que el desarrollo por medio de módulos es una de las mejores alternativas cuando se refiere a la calidad, mantenimiento y crecimiento de los proyectos, trabajar por secciones bien marcadas e independientes en mayor o menor medida una de otra permite poder realizar cambios sin la necesidad obligatoria de afectar a las demás partes que no se vean involucradas con esta implementación o corrección, permitiendo así mantener el correcto funcionamiento ininterrumpido de la aplicación para el mercado en el cual nos enfocamos. Poseer el conocimiento sobre la arquitectura necesaria para que uno de los módulos existentes en la aplicación nos extiende una vasta posibilidad para agregar y remover varios de estos a futuro, poseyendo amplia documentación tanta del modelo utilizado para la correcta creación de estos módulos como del funcionamiento de los mismos, manteniendo un esquema que se sostiene sin importar de la cantidad de módulos o de equipos que se vean inmiscuidos en el desarrollo de estos, pudiendo así asistir o encargarse de las actividades no importando si dicho apartado fue realizado por otro grupo.

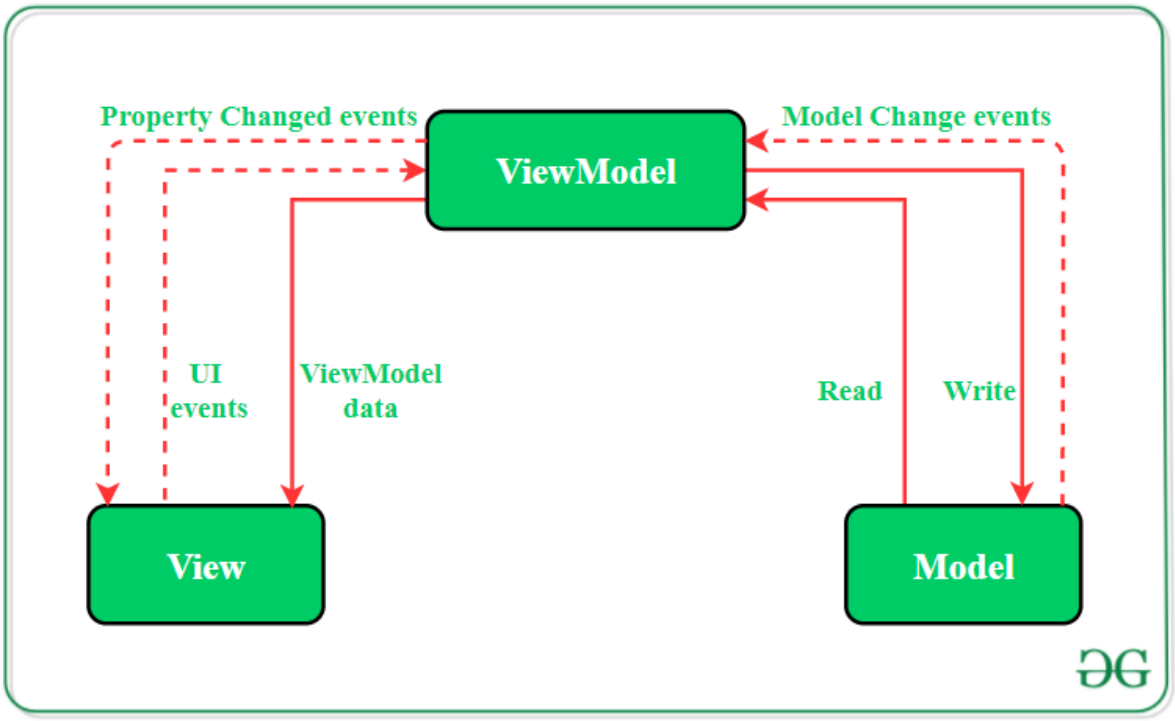
Es por esto y muchas más facilidades que brinda el esquema MVVM que es uno de los mayormente utilizados en la actualidad, esto por sobre otras arquitecturas como pueden ser la MVP o MVC, donde a pesar de las claras ventajas que estas también poseen, la arquitectura predilecta de Google ha sabido posicionarse de mejor manera en el mercado atrayendo de esta manera las vistas de las organizaciones que desean tomar parte importante en un mercado tan discutido como el que tenemos hoy en día.

¿Cuáles son sus componentes principales y cómo se relacionan entre sí?

Este modelo se basa en tres elementos a la hora de diseñar el funcionamiento de la aplicación, estos son:

- **Model**, dicha capa es la que se encarga del manejo y almacenamiento de datos, las distintas fuentes de información que se puedan utilizar a lo largo del desarrollo de nuestro producto van a ser guardados dentro de esta sección.
- **View,** este apartado del modelo esta pensado para la parte visual, el UI, siendo estos representados por medio de las activities, screens, XML y cualquier medio que permita al usuario visualizar e interactuar con la información recopilada por el Model.
- **ViewModel,** se podría decir que esta es la capa más importante de las 3 que forman el modelo, como su nombre lo índica es el nexo entre ambas capas anteriormente dichas, las Views se encargan de enlazarse a los respectivos ViewModels y estos se conectan con el Model para realizar cambios a la misma y devolverlos a la vista.

A continuación podemos visualizar un esquema donde se nos presenta la relación que estas tres capas forman entre si al momento de interactuar entre la información necesaria para la correcta experiencia del usuario y la visualización y posterior interacción que el mismo tendrá con dichos datos:



Como podemos apreciar, es necesaria el correcto funcionamiento de cada una de las capas con respecto a la función que esta posee dentro del desempeño de una aplicación, esto por el constante intercambio de acciones e información que tienen entre ellas donde en caso de llegar a fallar una de las capas, la otra se verá imposibilitada de realizar su función de manera satisfactoria, derivando así en una mala experiencia entregada para el usuario y su posterior perdida dentro de nuestro mercado optando por un competidor.

¿Cómo se aplica el patrón MVVM en Android con Kotlin?

Existe un par de alternativas para aplicar el patrón MVVM par aplicaciones realizadas con el lenguaje de programación de Kotlin:

- Haciendo utilización de la librería de DataBinding que provee Google.
- Utilizando herramientas que permitan esta posibilidad como lo puede ser RxJava.

Para el proyecto que se nos presenta actualmente se hará hincapie en la utilización de la primera alternativa de las mencionadas, esto ya que consideramos que al Google haberla seleccionado como la predilecta cuando se refiere a instruír en el mundo de las creaciones de aplicaciones moviles lo mejor sería seguir haciendo uso de las recomendaciones que esta empresa realiza al posicionarse como las alternativas a la vanguardia, tal cual le ocurrió a Kotlin con Java.

Podemos observer que en la web para desarrolladores por parte de google, tenemos mucha información para hacer aprender a darle un uso correcto a la librería anteriomente mencionada, lo siguiente es un pequeño ejemplo sobre las facilidades que ofrece:

- El siguiente trozo de codigo cumple la función de asignar una cadena de texto desde el viewModel a un componente TextView dentro de nuestro framework de UI:

```
findViewById<TextView>(R.id.sample_text).apply {
    text = viewModel.userName
}
```

Es facil notar la utilización de un método que permite aplicar a la propiedad del texto que posee el TextView un dato accediendo directamente a uno de los valores del ViewModel, la librería que nos presenta google permite realizar esto directamente de manera más facil dentro del archivo XML:

```
<TextView
        android:text="@{viewmodel.userName}" />
```

Con el llamado de una propiedad inherente de TextView podemos asignar en forma de variable un dato accediendo directamente al ViewModel sin recurrir a la utilización de una porción de código más extensa.

La biblioteca proporciona al usuario las funciones necesarias para realizar importaciones, variables y elementos para su posterior uso dentro de los diseños, esto permite mayor versatilidad y eficiencia sobre el diseño realizado y su posterior conexión con la capa que almacena los datos, cabe destacar que todo esto ocurre sin tener alguna repercusión inesperada con el diseño ya realizado, estas coexisten de manera correcta.

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto">
        <data>
            <variable
                name="viewmodel"
                type="com.myapp.data.ViewModel" />
        </data>
        <ConstraintLayout... /> <!-- UI layout's root element -->
    </layout>
```

Partiendo de los datos anteriormente dados, procederemos a especificar de manera más puntual el proceso de implementación del modelo para las aplicaciones moviles realizadas con Kotlin.

- En primer lugar crear debemos crear la estructura de carpetas que nos permita trabajar correctamente bajo el modelo, esto es posible mediante la creación de los apartados “model”, “views” y “viewmodels” dentro de nuestra carpeta inicial de Activitys y posteriomente crear las actividades en el interior de cada categoría correspondiente.

- Ahora es momento de agregar la librería mencionada con anterioridad dentro de nuestro archivo de configuraciones de gradle, más especificamente build.gradle, donde especificaremos que la librería debe de ser habilitada.

```
android {

    dataBinding {
        enabled = true
    }
}
```

- Antes de salir de dicho archivo debemos de igual manera agregar la dependencia que nos permitirá acceder a las capacidades de las que más tarde haremos uso en el desarrollo de la aplicación.

```
implementation 'android.arch.lifecycle:extensions:1.1.0'
```

- Codificaremos la estructura de codigo que formará parte de nuestro archivo Model, hay que recordar que es aquí donde se recibirán y almacenarán los datos que a futuro serán distribuídos para la consiguiente visualización e interacción de los usuarios, esta información puede ser recogida de varias maneras como puede ser base de datos, servicios web tales como API’s, formularios, entre las muchas maneras de consumo de información existente.

- Ahora bien es momento de realizar el diseño de la UI que permita al usuario visualizar la información previamente recolectada por medio del Model, ya sea que esta sea data inicial pensada para mostrarse sin importar el contexto o que la misma se encuentre moldeada en base a las preferencias del usuario anteriormente elegidas, hay que recordar hacer uso del apartado <data> al cual la librería nos abre posibilidad de uso.

```
<data>

    <variable
        name="viewModel"
        type="com.journaldev.androidmvvmbasics.viewmodels.LoginViewModel" />
</data>
```

- Dentro de nuestro diseño realizado en el paso anterior es importante aprender a hacer buen uso de las capacidades que nos habilita la utilización de MVVM por medio de la librería de google, como poder hacer referencia la información almacenada dentro del modelo de manera directa y optimizada, ya que de lo contrario no le estaremos dando un correcto uso y en su lugar estaremos malgastando el esfuerzo invertido en la investigación.

- Ahora con la creación de ambos modulos Model y View, podemos desarrollar el funcionamiento por medio del nexo entre mabos, el ViewModel, es aquí donde se estará haciendo una escucha constante de las acciones del usuario dentro de la UI para cuando el mismo realice una interacción que necesite actualizar la misma, sea el ViewModel el que se encargue de gestionar dicho cambio formalizando la petición entre View y Model para evitar tiempos de carga innecesario y brindarle la información con la mayor prontitud al usuario que se encuentre haciendo uso de la aplicación.

- Luego de realizar correctamente y con detalles menores los pasos anteriormente listados tendríamos lista nuestra aplicación con bases en la arquitectura MVM en funcionamiento para el publico, teniendo la facilidad de agregar, eliminar o modificar modulos para hacer del proyecto uno más robusto capaz de solventar hasta las mayores necesidades que el mercado actual puedan presentar, esto manteniendo siempre un estandar de calidad con respecto a la limpieza, optimización y escalabilidad del código empleado.

¿Cuáles son las ventajas y desventajas de utilizar el patrón MVVM en el desarrollo de aplicaciones móviles?

- Ventajas:
    - Separación de responsabilidades: MVVM permite una separación clara de responsabilidades entre la interfaz de usuario, la lógica de negocio y los datos de la aplicación, lo que facilita la organización y mantenimiento del código.

    - Testeabilidad: MVVM facilita las pruebas unitarias, ya que separa la lógica de negocio de la interfaz de usuario, lo que permite probar la lógica de negocio de forma aislada.

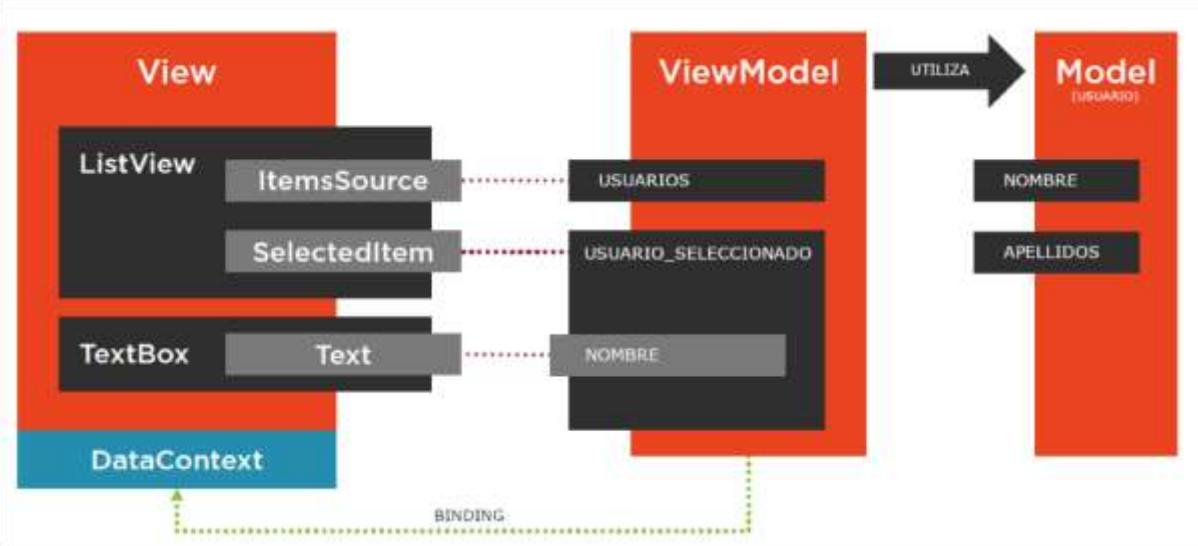
    - Flexibilidad: MVVM permite una mayor flexibilidad en el desarrollo, ya que permite la creación de múltiples vistas para un mismo ViewModel, lo que permite la reutilización de código y un mayor modularidad.

    - Flexibilidad: MVVM permite una mayor flexibilidad en el desarrollo, ya que permite la creación de múltiples vistas para un mismo ViewModel, lo que permite la reutilización de código y un mayor modularidad.

En general, el patrón MVVM ofrece numerosas ventajas en el desarrollo de aplicaciones móviles con Kotlin, pero también puede presentar algunos desafíos y desventajas. La elección del patrón de arquitectura adecuado dependerá de los requisitos específicos de cada proyecto y de las preferencias del equipo de desarrollo.

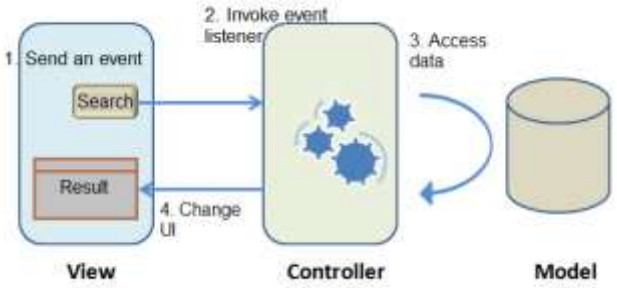
Anexos

Esquema de la arquitectura MVVM

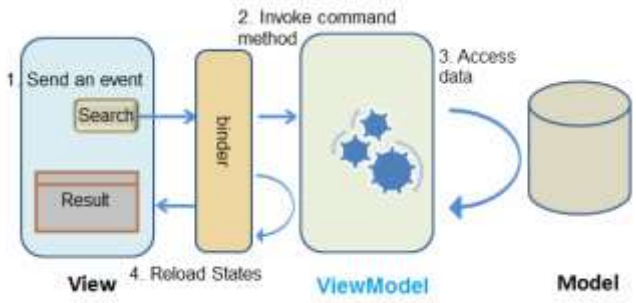


Comparación entre MVC y MVVM

- MVC



- MVVM



MVC (Model-View-Controller) y MVVM (Model-View-ViewModel) son dos patrones de arquitectura de software que se utilizan comúnmente en el desarrollo de aplicaciones móviles y web, la principal

diferencia entre MVC y MVVM radica en la forma en que se separan las responsabilidades y en cómo se comunica la vista con el modelo y la lógica de presentación. MVVM es una evolución de MVC que se enfoca más en la separación de responsabilidades y en la testabilidad de la lógica de presentación

Bibliografía

Coding Ninjas. (s. f.). *Code Studio*. https://www.codingninjas.com/codestudio/library/android-mvvm-model-view-viewmodel-architecture

Chugh, A. (2022). Android MVVM Design Pattern. *DigitalOcean*. https://www.digitalocean.com/community/tutorials/android-mvvm-design-pattern

GeeksforGeeks. (2022). MVVM Model View ViewModel Architecture Pattern in Android. *GeeksforGeeks*. https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

Aris. (2023). MVVM en Android con Kotlin, LiveData y View Binding – Android Architecture Components. *Curso Kotlin Para ANDROID*. https://cursokotlin.com/mvvm-en-android-con-kotlin-livedata-y-view-binding-android-architecture-components/

Camero, S. (2020, 25 septiembre). *Arquitecturas de Software. MVC y MVVM - Abatic Soluciones Tecnológicas*. Abatic Soluciones Tecnológicas. https://www.abatic.es/arquitecturas-de-software-mvc-y-mvvm/