



# Universidad Don Bosco

## FACULTAD DE INGENIERÍA

### ESCUELA DE INGENIERÍA EN COMPUTACIÓN

Desarrollo De Software Para Móviles

G04L

Ing. Alexander Alberto Sigüenza Campos

Integrante:

Reyes Vásquez, Fernando José

Numero de carnet:

RV180800

San Salvador, sábado 29 de abril del 2023

# Índice

Introducción.....	3
Modelo – Vista – Modelo de vista .....	4
¿Qué es el patrón MVVM?.....	4
¿Cómo se aplica el patrón MVVM en Android con Kotlin? .....	5
Ventajas .....	6
Desventajas .....	6
Anexos.....	7
Glosario de términos importantes .....	7
Vista del directorio principal de un proyecto que implementa MVVM.....	7
Bibliografía .....	8

# Introducción

A lo largo del ciclo hemos desarrollado numerosos proyectos en Android estudio, utilizando Java o Kotlin. Dichos proyectos han ido desde simples operaciones básicas, hasta una aplicación donde creamos un CRUD para manipular datos que están almacenados en una base de datos.

Android Studio es una herramienta muy flexible a la hora de trabajar con otras tecnologías que ayudan a los desarrolladores a crear aplicaciones más profesionales. Personalmente es mi primera experiencia en desarrollo de aplicaciones móviles, pero la flexibilidad de Android me a facilitado mucho mi trabajo a la hora de desarrollar una aplicación no tan simple.

Para entrar en materia hay que explicar de manera general que es un patrón de desarrollo y cómo funciona en Android. Pues en palabras simples un patrón de desarrollo es como una receta que debemos seguir paso a paso para poder preparar un platillo de calidad. En este caso el patrón MVVM son las indicaciones que debemos seguir para que nuestra aplicación sea todo lo nuestro usuario final necesita para estar satisfecho.

# Modelo – Vista – Modelo de vista

## ¿Qué es el patrón MVVM?

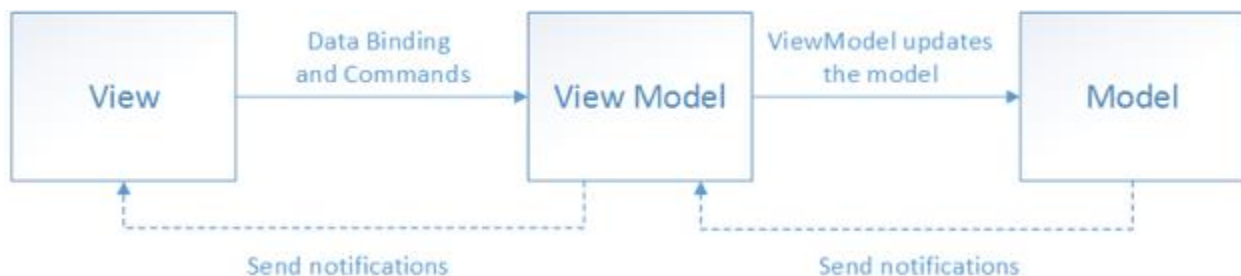
Antes de explicar que es el patrón MVVM debemos definir que es la lógica del negocio. La definición general sería: un conjunto de reglas o normas que deben ser implementadas dentro del código de la aplicación o sistema móvil. Para que esta última pueda funcionar como la empresa lo requiere. Generalmente se ve reflejada en cómo se manejan los datos que entran dentro del flujo de trabajo de la aplicación.

La lógica del negocio siempre es un aspecto cuidadosamente definido por una empresa, por lo tanto, los procesos, validaciones o mantenimientos deben ir en segundo plano para el usuario, que solo busca una aplicación intuitiva y funcional. En este punto entra el patrón MVVM ya que es el encargado de separar la lógica del negocio con la interfaz de usuario. Para los desarrolladores es muy practica porque se puede hacer un mantenimiento en la aplicación para mejorar su rendimiento sin que la interfaz de usuario se vea afectada ya que como mencionados anteriormente, trabajan separadas.

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

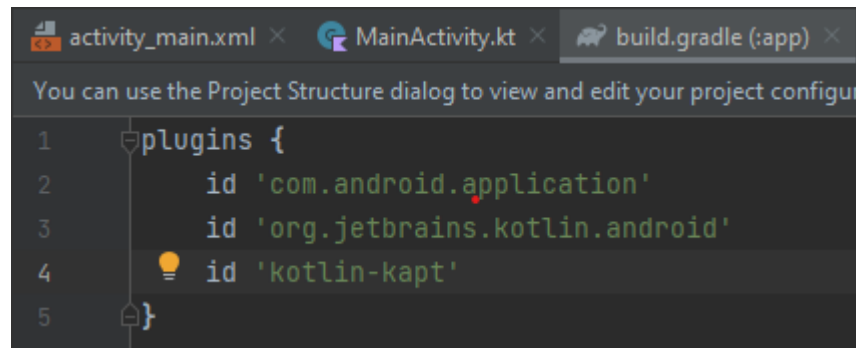
El patrón MVVM, consta de tres componentes que pueden ser visto de manera parecida a un diagrama de flujo, ya que los tres componentes están directa e indirectamente conectados.

1. La vista: Como se diría en desarrollo web, la vista es todo aquello con lo que usuario va a interactuar: cajas de texto, botones, tablas, notificaciones, entre otros. Todo lo que sea envidado desde la vista no llegara a la base de datos de golpe, si no que llegara a la vista del modelo.
2. Vista del modelo: En términos técnicos es el que se encarga de ejecutar todas las operaciones de la lógica del negocio, con los datos que recibe de la vista. También es el encargado de mantener actualizados todos los datos reflejando los cambios que puedan realizarse en cualquier momento. Por lo tanto, su tarea es más que ser un simpe intermediario entre la vista del usuario y el modelo.
3. Modelo: Es la clase que nos sirve como plantilla para crear nuestros objetos de datos. El modelo también es el encargado de trabajar directamente con los datos, realizando todos los procesos necesarios para mantener el orden en los datos.



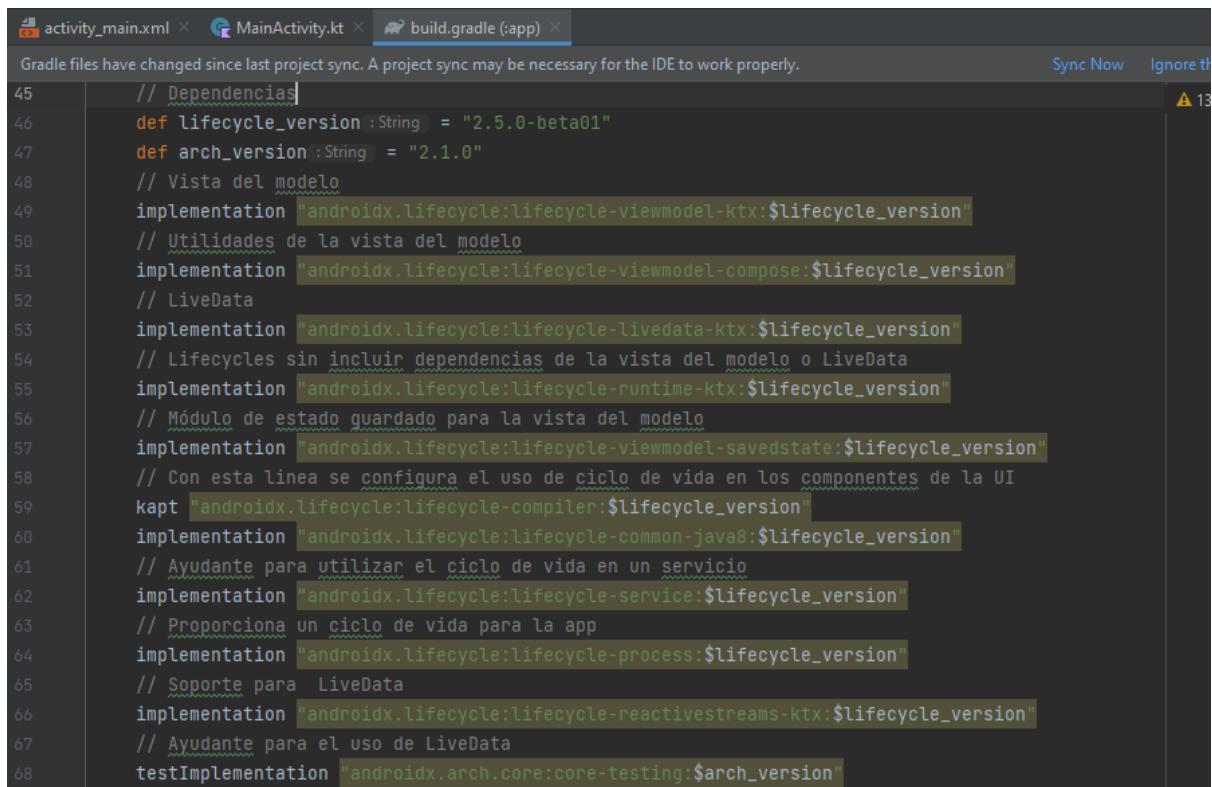
## ¿Cómo se aplica el patrón MVVM en Android con Kotlin?

Primero agregamos esta línea en el apartado de construcción de Gradle para poder incorporar nuestras dependencias en nuestro proyecto:



```
1 plugins {  
2     id 'com.android.application'  
3     id 'org.jetbrains.kotlin.android'  
4     id 'kotlin-kapt'  
5 }
```

Para el apartado de dependencias agregamos estas líneas de código:



```
45 // Dependencias  
46 def lifecycle_version :String = "2.5.0-beta01"  
47 def arch_version :String = "2.1.0"  
48 // Vista del modelo  
49 implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"  
50 // Utilidades de la vista del modelo  
51 implementation "androidx.lifecycle:lifecycle-viewmodel-compose:$lifecycle_version"  
52 // LiveData  
53 implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"  
54 // Lifecycles sin incluir dependencias de la vista del modelo o LiveData  
55 implementation "androidx.lifecycle:lifecycle-runtime-ktx:$lifecycle_version"  
56 // Módulo de estado guardado para la vista del modelo  
57 implementation "androidx.lifecycle:lifecycle-viewmodel-savedstate:$lifecycle_version"  
58 // Con esta línea se configura el uso de ciclo de vida en los componentes de la UI  
59 kapt "androidx.lifecycle:lifecycle-compiler:$lifecycle_version"  
60 implementation "androidx.lifecycle:lifecycle-common-java8:$lifecycle_version"  
61 // Ayudante para utilizar el ciclo de vida en un servicio  
62 implementation "androidx.lifecycle:lifecycle-service:$lifecycle_version"  
63 // Proporciona un ciclo de vida para la app  
64 implementation "androidx.lifecycle:lifecycle-process:$lifecycle_version"  
65 // Soporte para LiveData  
66 implementation "androidx.lifecycle:lifecycle-reactivestreams-ktx:$lifecycle_version"  
67 // Ayudante para el uso de LiveData  
68 testImplementation "androidx.arch.core:core-testing:$arch_version"
```

## Ventajas

- Proporciona orden y claridad, ya que todo se trabaja de manera separada, sin que los cambios realizados al modelo signifiquen un conflicto para la interfaz de usuario.
- Tenemos todas las herramientas necesarias para poder crear una interfaz que de la mejor experiencia a nuestros usuarios finales.
- El uso de este patrón permite que la aplicación siga evolucionando.
- Al trabajar con Android hemos dejado claro que el aprendizaje es bastante productivo por su flexibilidad, por lo tanto, una herramienta como MVVM nos ayuda de una gran manera a que nuestra aplicación sea más profesional y con un código ordenado.

## Desventajas

- Al ser un desarrollador novato en aplicaciones móviles, ha sido un poco complicado comprender el funcionamiento del patrón MVVM, viendo desde el código.
- Se necesitan muchas dependencias para poder configurar este patrón, generalmente pueden existir errores por incompatibilidad de versiones, lo que genera una inversión más de tiempo en poder solucionar los futuros conflictos en la compilación.
- El patrón MVVM puede introducir una sobrecarga en el rendimiento debido a la necesidad de traducir los datos entre el modelo, la vista y la vista del modelo.

# Anexos

## Glosario de términos importantes

**Lógica del negocio o lógica empresarial:** Son todos los procesos, operaciones, objetivos, misiones, validaciones, límites perfectamente ordenados y estructurados que toda empresa tiene para generar ingresos eficientemente. Toda lógica del negocio puede cambiar o evolucionar si la empresa logra un crecimiento a largo o corto plazo.

**Patrón:** Una o varias técnicas de desarrollo que permiten resolver un problema aparentemente largo, de forma simplificada.

**Arquitectura de software:** Conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software.

**Ciclo de vida:** Son los diferentes estados por los que pasa una actividad (Activity) o un fragmento (Fragment) de una aplicación mientras se ejecuta en un dispositivo móvil.

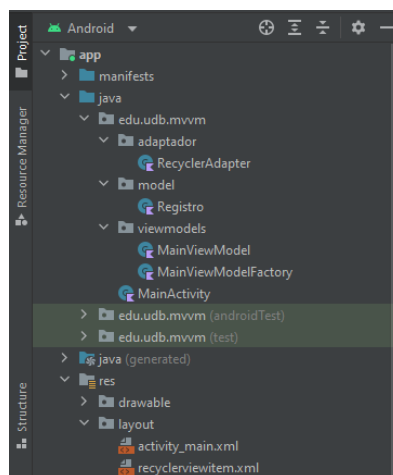
**LiveData:** Es una clase que se utiliza para crear y manejar objetos que son observables en una aplicación de Android. Se integra bien con los ciclos de vida de los componentes de la aplicación y solo actualiza los observadores registrados cuando la activity o el fragmento están en el estado activo.

**Kotlin-kapt:** Plugin que permite utilizar procesadores de anotaciones Java en código Kotlin.

**Adaptador:** Es una clase que sirve de intermediario entre los datos y una vista. Muy utilizado para mostrar datos mediante una RecyclerView o un ListView.

**RecyclerView:** Es un widget de UI que sirve para mostrar datos en una lista personalizable, es bastante utilizada por lo eficiente que es y por su gran flexibilidad a la hora de mostrar contenido como registros, imágenes o botones.

Vista del directorio principal de un proyecto que implementa MVVM.



## Bibliografía

Andorid Developers. (3 de Abril de 2023). *Documentacion oficial de Android*. Obtenido de <https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es-419>

Barasa, M. (17 de Noviembre de 2020). *Implementing MVVM architecture in Android using Kotlin*. Obtenido de <https://www.section.io/engineering-education/implementing-mvvm-architecture-in-android-using-kotlin/>

Microsoft Developer Division, .. a. (11 de Marzo de 2022). *learn microsoft*. Obtenido de <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>