Universidad Don Bosco

FACULTAD DE INGENIERIA
ESCUELA DE COMPUTACIÒN
CICLO I– 2023

Asignatura:
Desarrollo de Software para Móviles

Docente:
Ing. Alexander Sigüenza

Integrantes:

Cesar Antonio Ramírez Barahona	RB191879
Nelson Ernesto Muñoz Barahona	MB192012

Fecha de entrega:
Sábado 29 de febrero del 2023

INDICE

INTRODUCCION

Con el tiempo en la comunidad se ha acostumbrado a la nueva manera de trabajar bajo los marcos de los patrones arquitectónicos esto les permite dar cabida a patrones más modulares, y que pueden ser testeados con mayor facilidad.

Los patrones arquitectónicos, o patrones de arquitectura, también llamados arquetipos ofrecen soluciones a problemas de arquitectura de software en ingeniería de software.

Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados.

conceptualización de una solución genérica y reutilizable, aplicable a un problema de diseño de software en un contexto determinado, satisfaciendo las necesidades del negocio.

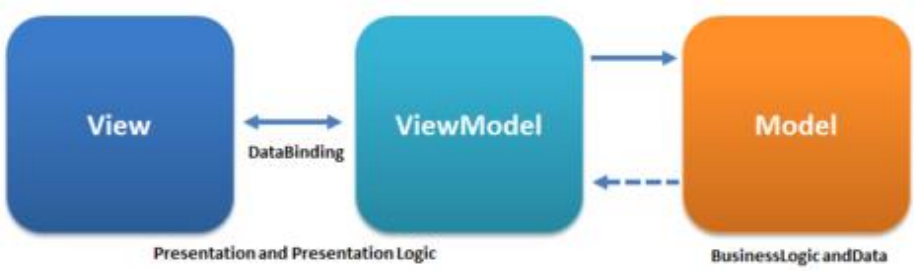
En esta investigación profundizaremos en el patrón arquitectónico: **MVVM**

¿Qué es el patrón MVVM?

¿Pero que es el patrón MVVM?

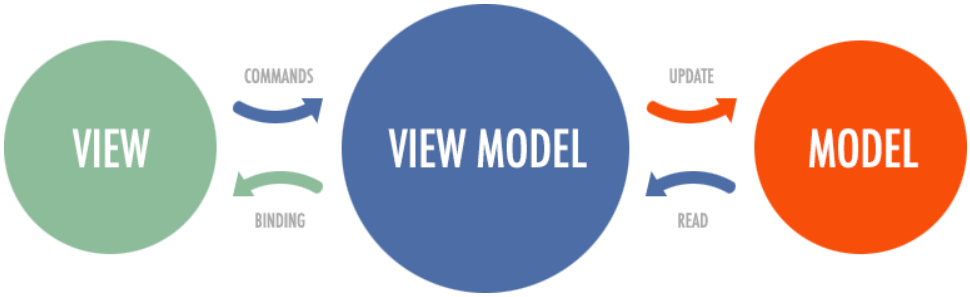
Este es un patrón de diseño utilizado en la capa de presentación de una aplicación, pero una aplicación no se compone solo de esta capa, sino que se complementa con otras capas como la de dominio y la de datos.

Este modelo de diseño tiene el objetivo para llevar a cabo la separación del apartado de la interfaz de usuario (View) de la parte lógica (Model) de esta manera genera diversas ventajas y facilidades para una mayor optimización de código.



Este modelo trabaja con una organización siguiendo el modelo de capas el cual encontramos:

- **View**
- **ViewModel**
- **Model**

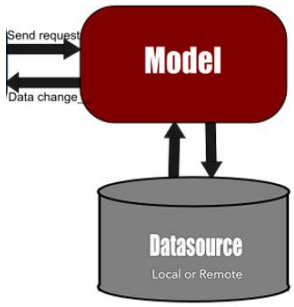


¿Cuáles son sus componentes principales?

MVC significa Modelo Vista Controlador, porque en este patrón de diseño se separan los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. Cuando la lógica de negocio realiza un cambio, es necesario que ella sea la que actualiza la vista.

COMPONENTES:

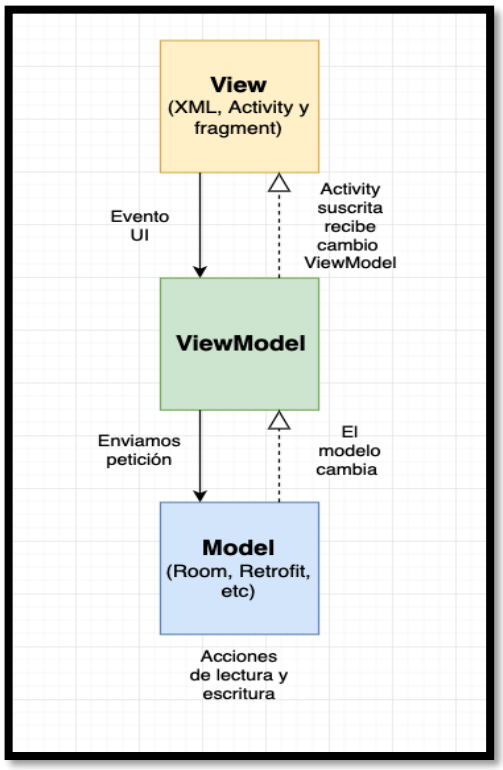
- La parte del modelo (Model), en la cual vamos a ver todo lo que serían los datos, dónde vamos a tener toda la lógica de datos.



- La vista modelo (View Models) que va a ser la encargada de interactuar tanto con el modelo como con la vista.

- La vista (Views) que va a ser la parte visual donde tendremos los diseños o vistas.

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



- En la capa **View** tendremos nuestras vistas XML layouts y diseño etc.

- La capa **viewModel** en Androd funcionara como intermediario de peticiones el cual conectara las vistas con las peticiones o eventos creados para poder hacer funcionar la app.

- En la capa **Model** recibirá toda acción dirigida a nuestra base de datos adicional tendremos las acciones de escrituras y lecturas en la cual nos servirá como intermediario hacia nuestra base de datos.

• View ejemplo en código

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="htt
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/viewContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/background"
    tools:context=".view.MainActivity">

    <TextView
        android:id="@+id/tvQuote"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="16dp"
        android:textColor="@color/white"
        android:textSize="24sp"
        android:textStyle="italic"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvAuthor"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="16dp"
        android:textColor="@color/white"
        android:textSize="24sp"
        android:textStyle="italic"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

• ViewModel ejemplo en código(Binding)

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}
```

• Model (Código en storage simulando BD)

```
class QuoteProvider {
    companion object {

        fun random(): QuoteModel {
            val position = (0..10).random()
            return quotes[position]
        }

        private val quotes = listOf(
            QuoteModel(
                quote = "It's not a bug. It's an undocumented feature!",
                author = "Anonymous"
            )
```

¿Ventajas de usar MVVM?

• Si una implementación de modelo existente encapsula la lógica de negocios existente, puede ser difícil o arriesgada cambiarla. En este escenario, el modelo de vista actúa como adaptador para las clases de modelo y evita que realice cambios importantes en el código del modelo.

• Los desarrolladores pueden crear pruebas unitarias para el modelo de vista y el modelo, sin usar la vista. Las pruebas unitarias para el modelo de vista pueden ejercer exactamente la misma funcionalidad que la vista.

• La interfaz de usuario de la aplicación se puede rediseñar sin tocar el modelo de vista y el código del modelo, siempre que la vista se implemente completamente en XML o Kotlin Por lo tanto, una nueva versión de la vista debe funcionar con el modelo de vista existente.

•Los diseñadores y desarrolladores pueden trabajar de forma independiente y simultánea en sus componentes durante el desarrollo. Los diseñadores pueden centrarse en la vista, mientras que los desarrolladores pueden trabajar en el modelo de vista y los componentes del modelo.

¿Desventajas de usar MVVM?

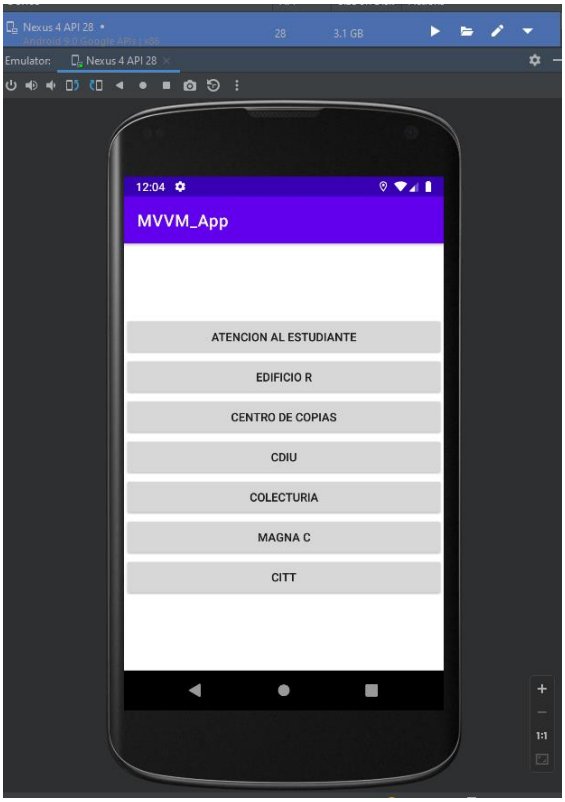
• Necesitas hacer ciertas cosas que se podrían hacer de manera más fácil.

• No es tan replicable como el MVP, en cada Activity o Fragment donde lo uses tienes que hacer algunas cosas diferentes.

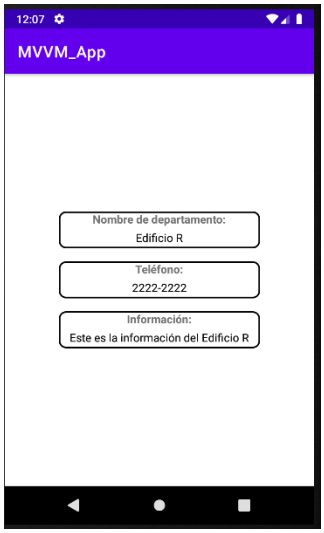
• Es difícil de adaptar a apps que ya están hechas.

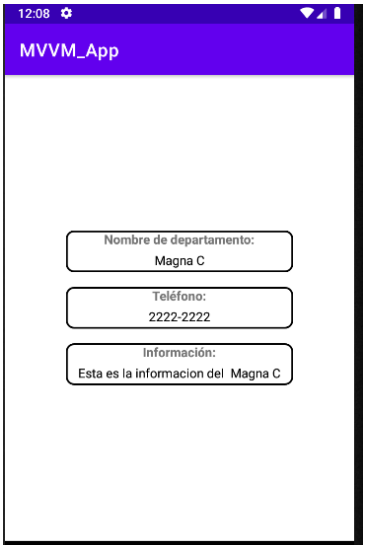
Ejemplo Practico

Nuestro ejemplo se basa en la simulación de un Menú para un directorio de la universidad Don Bosco, es un ejemplo básico pero que aplica bien la arquitectura MVVM

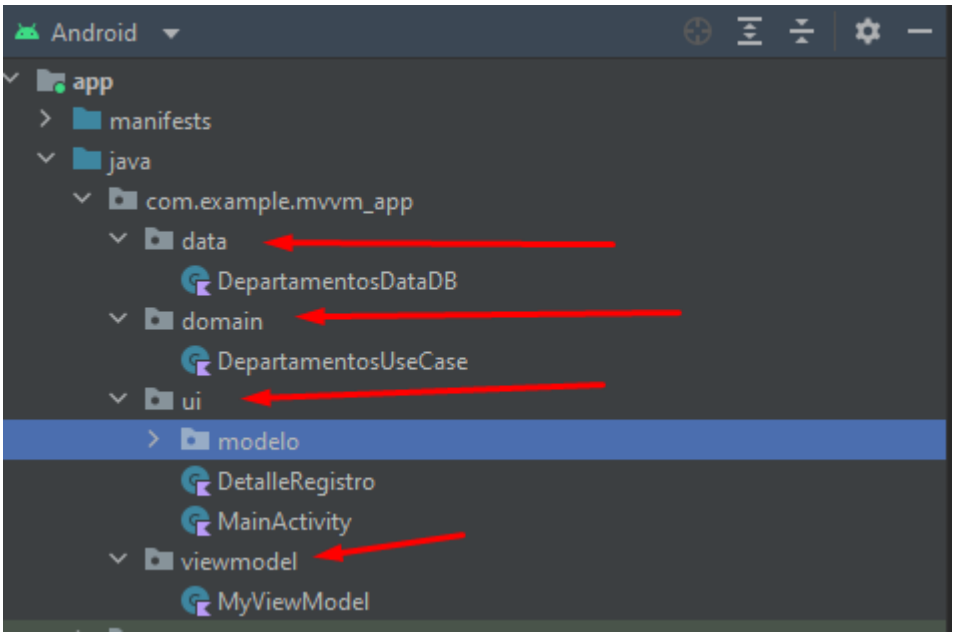


Funcionalidad:





Arquitectura de carpetas:



- **Data:** (Funciona como nuestro modelo, donde se tiene todo tipo de a la BD)
- **Domain:** (Nuestro binding o mas bien nuestro intermediario entre las peticiones hacia la BD del ViewModel)
- **UI:** (Obtiene todas las vistas de los departamentos con los directorios)
- **ViewModel** (Programación y peticiones)

CODIGO:

Data(Model): Simulando BD almacenando todo en local storage

```
import com.example.mvvm_app.ui.modelo.Departamentos

class DepartamentosDataDB {

    fun createListDepartamentos():List<Departamentos>{
        return listOf(
            Departamentos("Atencion al Estudiante","2222-2222","Este es el departamento de estudiantes"),
            Departamentos("Edificio R", "2222-2222","Este es la información del Edificio R"),
            Departamentos("Centro de Copias","2222-2222","Este es la informacion del centro de copias"),
            Departamentos("CDIU","2222-2222","Este es la informacion del edificio CDIU"),
            Departamentos("Colecturia","2222-2222","Esta es la informacion del departamento de Colecturia"),
            Departamentos("Magna C", "2222-2222","Esta es la informacion del  Magna C"),
            Departamentos("CITT","2222-2222", "Esta es la informacion del CITT")
        )
    }
}
```

Domain: (Binding ,instanciamos los datos de la BD)

DepartamentosDataDB.kt | DepartamentosUseCase.kt

```
package com.example.mvvm_app.domain

import ...

class DepartamentosUseCase {

    val departamentosDataDB= DepartamentosDataDB()

    fun getListaDespartamentos():List<Departamentos>{
        return departamentosDataDB.createListDepartamentos()
    }
}
```

ViewModel :Seteamos y obtenemos datos transportados al MainActivity

MyViewModel.kt

```
package com.example.mvvm_app.viewmodel

import ...

class MyViewModel:ViewModel() {

    private val listData = MutableLiveData<List<Departamentos>>()

    val Departamentosusecase = DepartamentosUseCase()

    init {
        getListaDepartamentos()
    }


    fun setListDataDepartamentos(ListaDepartamentos:List<Departamentos>){
        listData.value = ListaDepartamentos
    }

    fun getListaDepartamentos(){
        setListDataDepartamentos(Departamentosusecase.getListaDespartamentos())
    }

    fun getListaDepartamentosLiveData():LiveData<List<Departamentos>>{
        return listData
```

Anexos

- **Microsoft página educativa oficial:** [Enlace]

- **Adictos al trabajo :** [Enlace]

- **Open webinar (Free source) :** [Enlance]