# Programación Móvil

Nombre: ***Adrian Eduardo Reynosa Flores***

**#2010455**

## Introducción al patrón MVVM

MVVM es un patrón arquitectónico que facilita la separación de preocupaciones en una aplicación, dividiendo los componentes en tres capas principales: Modelo (Model), Vista (View) y Modelo de Vista (ViewModel). Fue introducido por primera vez por Microsoft para el desarrollo de aplicaciones de escritorio y luego ganó popularidad en el desarrollo de aplicaciones web y móviles.

## Componentes del patrón MVVM

1. **Modelo (Model)**:
    - Representa los datos y la lógica de negocio de la aplicación.
    - No depende de la capa de la vista ni del modelo de vista.
    - Puede contener clases de datos, acceso a datos (como APIs o bases de datos), y lógica de validación y cálculo.
2. **Vista (View)**:
    - Es la interfaz de usuario visible para el usuario final.
    - No contiene lógica de negocio, solo se encarga de mostrar los datos y responder a las interacciones del usuario.
    - Idealmente, debería ser lo más pasiva posible y delegar la lógica de presentación al ViewModel.
3. **Modelo de Vista (ViewModel)**:
    - Actúa como un intermediario entre el Modelo y la Vista.
    - Prepara y maneja los datos que la Vista necesita para mostrar.
    - Generalmente, expone métodos y comandos que la Vista puede vincular y llamar.
    - No debe contener referencias directas a la Vista para mantener la separación de la lógica de la interfaz de usuario.

## Principios y Ventajas de MVVM

- **Separación de preocupaciones**: MVVM facilita la separación de la lógica de presentación (ViewModel) de la interfaz de usuario (Vista) y la lógica de negocio (Model), lo que mejora la modularidad y facilita el mantenimiento del código.
- **Facilita las pruebas unitarias**: Al separar la lógica de presentación del código de la interfaz de usuario, se facilita la escritura de pruebas unitarias automatizadas para el ViewModel sin la necesidad de simular la interacción del usuario.

- **Soporte para la reutilización de código**: Al reducir el acoplamiento entre la Vista y el ViewModel, es más fácil reutilizar el ViewModel en diferentes vistas o incluso en diferentes plataformas.
- **Mejora la escalabilidad**: MVVM es escalable para aplicaciones grandes ya que promueve la modularidad y la organización del código, lo que facilita la gestión de proyectos complejos a medida que crecen en tamaño y complejidad.

## Implementación de MVVM en Android

En Android, la implementación de MVVM generalmente implica el uso de componentes de Android Jetpack como LiveData, ViewModel y Data Binding:

- **LiveData**: Utilizado en el ViewModel para proporcionar actualizaciones observables a la Vista cuando los datos cambian.
- **ViewModel**: Mantiene datos relacionados con la UI que sobreviven a los cambios de configuración y gestiona la lógica de la UI.
- **Data Binding**: Facilita la conexión directa entre componentes de la UI y el ViewModel, lo que reduce la cantidad de código necesario para mantener la sincronización entre la Vista y el ViewModel.

## Ejemplo de Implementación Básica en Android

1. **Definición del Modelo** (por ejemplo, `User.java`):

```
public class User {
    private String name;
    private int age;

    // Constructor, getters y setters
}
```

2. **ViewModel** (por ejemplo, `UserViewModel.java`):

```
public class UserViewModel extends ViewModel {
    private MutableLiveData<User> userLiveData;

    public UserViewModel() {
        userLiveData = new MutableLiveData<>();
        // Inicializar datos o cargar desde una fuente
    }

    public LiveData<User> getUserLiveData() {
        return userLiveData;
    }

    public void setUser(User user) {
        userLiveData.setValue(user);
    }
}
```

3. **Vista** (por ejemplo, `MainActivity.java` con Data Binding):

```
public class MainActivity extends AppCompatActivity {
    private UserViewModel viewModel;
    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = DataBindingUtil.setContentView(this,
R.layout.activity_main);
        viewModel = new
ViewModelProvider(this).get(UserViewModel.class);

        viewModel.getUserLiveData().observe(this, user -> {
            // Actualizar la interfaz de usuario con los datos del
ViewModel
            binding.textViewName.setText(user.getName());

binding.textViewAge.setText(String.valueOf(user.getAge()));
        });

        // Simular la carga de datos o actualización del usuario
        User user = new User("John Doe", 30);
        viewModel.setUser(user);
    }
}
```

## Conclusión

MVVM es un patrón arquitectónico eficaz que promueve una estructura de código clara y modular en aplicaciones Android. Facilita la separación de preocupaciones, mejora la escalabilidad y la prueba de aplicaciones, y es compatible con las prácticas de desarrollo modernas como LiveData y Data Binding. Su adopción puede beneficiar significativamente el desarrollo de aplicaciones Android al mejorar la mantenibilidad y la claridad del código.