DESARROLLO DE SOFTWARE

PROGRAMACION 3

CONSULTA

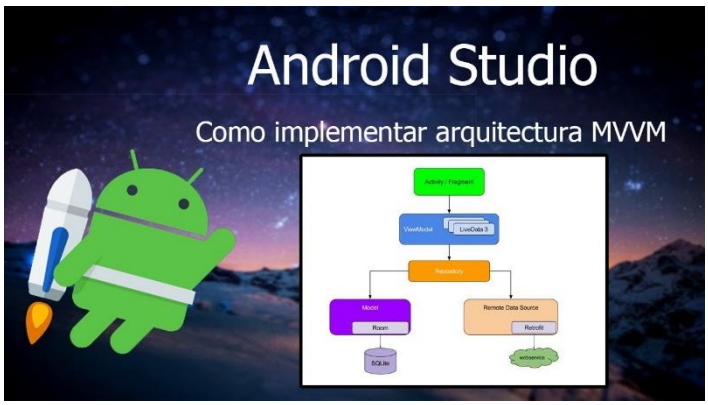
Cómo implementar una Arquitectura MVVM en Android Studio

Patrones de diseño

Andrés Sebastián Cajas López

Tercer Semestre TSDS

Cómo implementar una Arquitectura MVVM en Android Studio



La arquitectura MVVM (Modelo-Vista-ViewModel) es uno de los patrones de diseño más utilizados en el desarrollo de aplicaciones Android modernas. Su principal ventaja es que permite separar de manera clara la lógica de negocio de la interfaz de usuario, lo que facilita la mantenibilidad y escalabilidad de la aplicación. Implementar MVVM en Android Studio ayuda a organizar mejor el código, haciendo que sea más fácil de mantener y expandir a medida que la aplicación crece.

Pasos para implementar MVVM en Android Studio

1. **Crea el modelo (Model)**: Este componente maneja los datos de tu aplicación. Puede ser una clase que interactúe con la base de datos o APIs para obtener o enviar información. El modelo no debe preocuparse de cómo se muestran los datos en la pantalla, solo se encarga de gestionarlos.
2. **Crea la vista (View)**: Es la interfaz de usuario, todo lo que el usuario ve e interactúa en la app. En Android, la vista son tus actividades o fragmentos. La vista es responsable de mostrar los datos, pero no debe manejar la lógica de negocio.
3. **Crea el ViewModel**: Este componente actúa como intermediario entre la vista y el modelo. Toma los datos del modelo y los prepara para que la vista los muestre. También maneja la lógica de presentación, pero no se involucra directamente con los datos o la interfaz.

4. **Conecta todo en tu actividad o fragmento**: En tu actividad o fragmento, creas una instancia del ViewModel y lo conectas con la vista. La vista observa el ViewModel para actualizarse cuando los datos cambian.

Patrones de diseño

Los patrones de diseño provienen de Christopher Alexander, quien los aplicó al diseño urbano. En 1995, Erich Gamma y otros adaptaron esta idea a la programación orientada a objetos en su libro Patrones de diseño, donde presentaron 23 patrones para resolver problemas comunes en software, popularizando su uso en la programación.

Características:

- Representa una solución probada para problemas comunes.
- Ofrece una estructura general que se puede adaptar al proyecto.
- A diferencia de un algoritmo, que define pasos exactos, un patrón se enfoca en la estructura general de la solución.

Estructura de los Patrones de Diseño:

- Propósito: Define el problema y la solución.
- Motivación: Explica cómo lo resuelve.
- Estructura de clases: Muestra la organización de clases.
- Ejemplo de código: Cómo implementarlo.

Ventajas:

- Ofrecen soluciones comprobadas.
- Mejoran la comunicación en equipos

Desventajas:

- A veces se usan en lenguajes sin las herramientas adecuadas.
- Pueden ser ineficientes si no se implementan bien.
- Los principiantes tienden a usarlos innecesariamente.

Clasificación:

1.- Por complejidad:

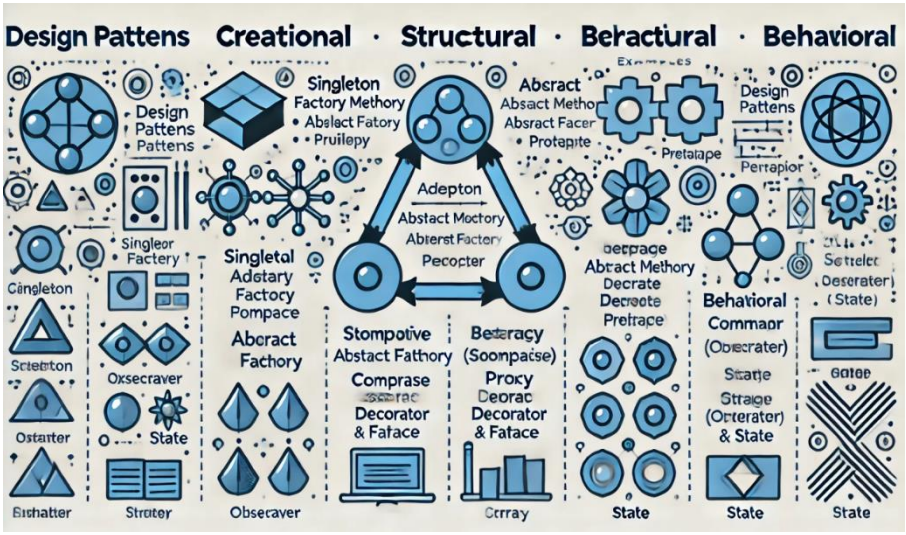
- **Idioms**: Patrones básicos y específicos para un lenguaje de programación.
- **Patrones de arquitectura**: Más generales, aplicables a cualquier lenguaje, y usados para diseñar la arquitectura de una aplicación completa

2.- Por propósito:

Creacionales: Se enfocan en la creación flexible de objetos, dentro de los cuales tenemos:

Factory Method	Proporciona una interfaz para crear objetos, permitiendo que las subclases cambien el tipo de objeto.
Abstract Factory	Crea familias de objetos sin especificar sus clases concretas.
Builder	Facilita la construcción de objetos complejos paso a paso.
Prototype	Copia objetos existentes sin depender de sus clases concretas.
Singleton	Asegura que una clase tenga una única instancia.

Estructurales: Definen cómo organizar objetos y clases para mantener la flexibilidad y eficiencia, dentro de estos tenemos:



Adapter	Permite que objetos con interfaces incompatibles trabajen juntos.
Bridge	Divide una clase grande en dos jerarquías (abstracción e implementación).
Composite	Agrupa objetos en estructuras de árbol tratándolos como objetos individuales.
Decorator	Añade funcionalidades a un objeto envolviéndolo en otro objeto
Facade	Ofrece una interfaz simplificada para un conjunto complejo de clases.
Flyweight	Optimiza el uso de memoria compartiendo partes comunes de estado.
Proxy	Proporciona un sustituto para un objeto, controlando su acceso.

De comportamiento: Se centran en la comunicación y asignación de responsabilidades entre objetos, dentro de estos tenemos:

Chain of Responsibility	Pasa solicitudes a lo largo de una cadena de manejadores.
Command	Convierte una solicitud en un objeto independiente, permitiendo su parametrización.
Iterator	Permite recorrer una colección sin exponer su estructura interna.
Mediato	Reduce dependencias entre objetos al hacer que se comuniquen solo a través de un mediador.
Memento	Guarda y restaura el estado de un objeto sin revelar su implementación.
Observer	Notifica a varios objetos sobre eventos que ocurren en el objeto observado.
State	Cambia el comportamiento de un objeto cuando su estado interno cambia.
Strategy	Define una familia de algoritmos intercambiables.
Template Method	Define un algoritmo en la superclase, permitiendo modificaciones en subclases.
Visitor	Separa algoritmos de los objetos sobre los que operan.

Bibliografía:

(Alexander, REFACTORING GURU, s.f.)https://refactoring.guru/

(Gaerste, 2023)https://dianagaerste.com/patrones-de-comportamiento/

(Luis, 2024)https://keepcoding.io/blog/pasos-para-la-implementacion-de-mvvm-en-android/