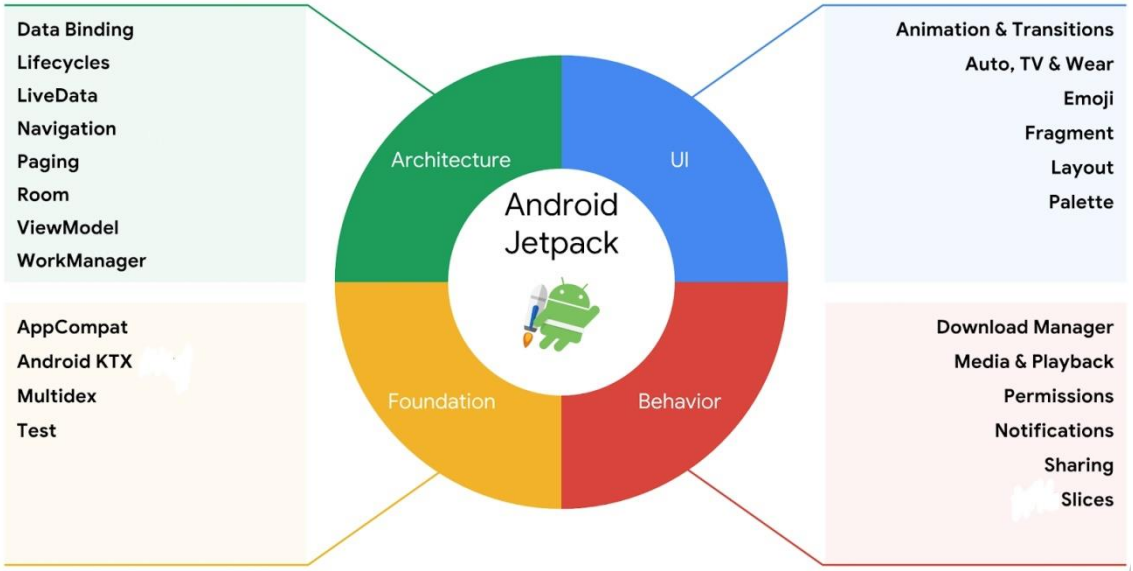
Jetpack


Data Binding
Lifecycles
LiveData
Navigation
Paging
Room
ViewModel
WorkManager
AppCompat
Android KTX
Multidex
Test
Architecture
UI
Android
Jetpack
Foundation
Behavior
Animation & Transitions
Auto, TV & Wear
Emoji
Fragment
Layout
Palette
Download Manager
Media & Playback
Permissions
Notifications
Sharing
Slices

Patrón de arquitectura de software

Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado.

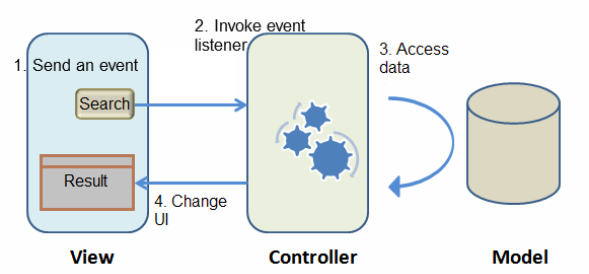
MVC – MVP – MVVM

Son patrones de arquitectura de software muy utilizados.

El patrón MVC (Modelo-Vista-Controlador) fue una de las primeras ideas (1978) en el campo de las interfaces gráficas de usuario. Algunos aspectos del patrón MVC han evolucionado dando lugar a ciertas variantes entre ellas MVP (Modelo-Vista-Presentador) y MVVM (Modelo-Vista Vista-Modelo).

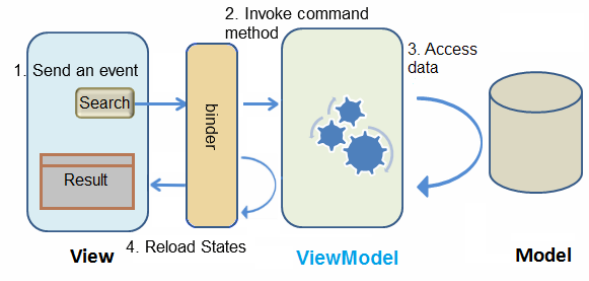
MVC

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.



MVVM

MVVM significa Modelo Vista VistaModelo, porque en este patrón de diseño se separan los datos de la aplicación, la interfaz de usuario pero en vez de controlar manualmente los cambios en la vista o en los datos, estos se actualizan directamente cuando sucede un cambio en ellos, por ejemplo si la vista actualiza un dato que está presentando se actualiza el modelo automáticamente y viceversa.



Las principales diferencias entre MVC y MVVM son que en MVVM el «Controller» cambia a «ViewModel» y hay un «binder» que sincroniza la información en vez de hacerlo un controlador «Controller» como sucede en MVC.

¿Por qué usar MVVM en lugar de MVC?

La vista en MVC depende del modelo y del controlador para obtener datos o modificarlos. Si hay un cambio en la vista que implique un cambio en el modelo necesita al controlador para hacerlo mientras que si hay datos nuevos, depende del modelo para traerlos. En el caso del MVVM, la vista solo depende del modelo de la vista para obtener datos o modificarlos, además nadie tiene que decirle que actualice los datos como haría el controlador en MVC pasivo, ya que está observando cambios en el modelo directamente.

El modelo en MVC puede tener demasiadas responsabilidades como obtener los datos de fuentes de datos, informar al controlador sobre cambios en esos datos y prepararlos para que se puedan mostrar en la vista. En MVVM el modelo queda totalmente desacoplado de la vista y solo contiene información, nunca acciones para manipularla.

En MVC no queda muy claro quién se debería encargar de la lógica de presentación ya que el modelo solo debería enviar datos en crudo a la vista y por otro lado, llevar esta lógica a la vista complicaría algunos tests. En MVVM, esta responsabilidad es muy clara y cae en manos del modelo de la vista.

MVC en Android

El **modelo** contiene la información con la que el sistema trabaja, proporcionándosela a la vista para que pueda mostrarla y permitiendo realizar cambios en ella desde el controlador.

> En los ejercicios hechos hasta ahora serían nuestras clases POJO llamadas por ejemplo Item. Cuando veamos BD, también serán las clases que definen la conexión, las consultas,… a dicha base de datos.

La **vista** presenta al usuario la información del modelo.

> En los ejercicios hechos hasta ahora serían los ficheros XML hechos para los "layouts" (y las clases Adapter para enlazar los datos con las vistas de listas)

El **controlador** responde a acciones del usuario, modificando el modelo cuando sea necesario. Además, se comunica con la vista para que se actualice con los últimos cambios del modelo.

> En los ejercicios hechos hasta ahora serían las clases Activity que implementan los escuchadores a eventos sobre botones, ítems, etc.

Uno de los errores más comunes a la hora de utilizar MVC en Android ha sido y aún sigue siendo utilizar una Activity como controlador, haciéndola responsable de tareas de las que no debería ocuparse, como llamar directamente al modelo para realizar una modificación de algún dato, cuando esto debería ser tarea del controlador. Esto es claramente una violación del principio de responsabilidad única y al usar una Activity como controlador le estamos haciendo dependiente de la plataforma, por lo que su código podría verse afectado cuando la Activity sea destruida por el sistema.

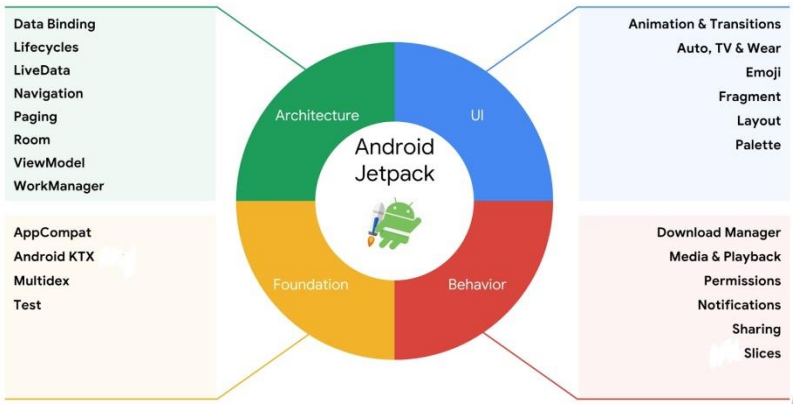
Las activities y fragments solo deberían ocuparse de mostrar los datos y notificar los eventos del usuario al controlador, siendo los únicos componentes ligados a la plataforma. Los controladores y modelos deberían ser clases separadas sin ninguna dependencia con Android

MVVM en Android

MVVM es el estándar para implementar aplicaciones de Android desde que Google lanzó su Guía de arquitectura de aplicaciones (eso no quiere decir que otros patrones no sean válidos). Para usar este patrón arquitectónico en Android hay que usar los componentes de arquitectura de Android Jetpack.

Android Jetpack

Jetpack es un conjunto de bibliotecas, herramientas y guías para ayudar a los desarrolladores a escribir apps de alta calidad de forma más sencilla. Está formado por las bibliotecas de paquetes de androidx.*.



Componentes de Android Jetpack

Los componentes de Android Jetpack son una colección de bibliotecas que se pueden usar de manera individual, aunque fueron diseñados para funcionar en conjunto al mismo tiempo que aprovechas las características del lenguaje Kotlin que te permiten aumentar la productividad. Úsalos todos o combínalos como más te convenga.

Base

Los componentes de base ofrecen funcionalidades interrelacionadas, como pruebas, compatibilidad con versiones anteriores y lenguaje Kotlin.

Android KTX
Escribe código Kotlin más idiomático y conciso

AppCompat
Degradación elegante a versiones anteriores de Android

Auto
Componentes que ayudan a desarrollar apps para Android Auto

Benchmark
Obtén rápidamente comparativas de tu código basado en Kotlin o en Java dentro de Android Studio

Multidex
Brinda compatibilidad con apps con varios archivos DEX

Seguridad
Lee y escribe archivos encriptados y preferencias compartidas siguiendo prácticas recomendadas de seguridad

Test
Marco de trabajo de Android para pruebas de IU de unidades y tiempo de ejecución

TV
Componentes que ayudan a desarrollar apps para Android TV

Wear OS by Google
Componentes que ayudan a desarrollar apps para Wear

Arquitectura

Los componentes de arquitectura ayudan a diseñar apps sólidas, que puedan someterse a pruebas y admitan mantenimiento.

Vinculación de datos
Vincula datos observables a elementos de IU de manera declarativa

Ciclos de vida
Administra los ciclos de vida de tu actividad y tu fragmento

LiveData
Notifica las vistas cuando la base de datos subyacente cambia

Navegación
Administra todo lo necesario para la navegación desde la app

Paginación
Carga información de tu fuente de datos de manera gradual según la demanda

Room
Acceso fluido a la base de datos SQLite

ViewModel
Administra los datos relacionados con la IU de manera optimizada para los ciclos de vida

WorkManager
Administra tus tareas de Android en segundo plano

Comportamiento

Los componentes de comportamiento ayudan a tu app a integrarse con servicios estándar de Android, como notificaciones, permisos, uso compartido y el Asistente.

CameraX
Agrega fácilmente funciones de cámara a tus apps

Administrador de descargas
Programa y administra descargas de gran tamaño

Contenido multimedia y reproducción
API compatibles con versiones anteriores para la reproducción y el enrutamiento de contenido multimedia (incluido Google Cast)

Notificaciones
Proporciona una API de notificación compatible con versiones anteriores que admite Wear y Auto

Permisos
API de compatibilidad para verificar y solicitar permisos de apps

Preferencias
Crea pantallas de configuración interactivas

Opciones para compartir
Proporciona una acción compartida apta para la barra de acción de una app

Secciones
Crea elementos de IU flexibles que pueden mostrar datos de la app fuera de ella

IU

Los componentes de IU ofrecen widgets y asistentes para crear una app que sea fácil y agradable de usar. Obtén información acerca de Jetpack Compose, que ayuda a simplificar el desarrollo de IU.

Animaciones y transiciones
Mueve widgets y alterna entre pantallas

Emoji
Habilita una fuente de emoji actualizada en una plataforma anterior

Fragmento
Unidad básica de IU componible

Diseño
Implementa widgets con diferentes algoritmos

Paleta
Obtén información útil de las paletas de colores

Ya conocemos algunas de ellas (por ejemplo, Fragment de UI, Preferencias de comportamiento, etc). Dependiendo del tipo será necesario añadir la dependencia en el fichero build.gradle (ej: implementation 'androidx.preference:preference:1.1.0')

Componentes de la arquitectura de Android que forman parte de Android Jetpack

Estos componentes lanzados por el equipo de Android hace ya un par de años nos facilitan mucho la vida a la hora de implementar un patrón MVVM en Android, además de hacer nuestras aplicaciones más robustas, mantenibles y fáciles de probar.

LiveData:

Objetos que notifican a la vista cuando hay cambios, por ejemplo, en la base de datos.

Son conscientes de los ciclos de vida, por lo que nos ayudan a evitar crashes cuando una Activity se detiene por ejemplo.

ViewModel:

Clases responsables de preparar y manejar los datos de una Activity o Fragment.

Permite que los datos sobrevivan a cambios de configuración como rotaciones de pantalla, minimizando el uso de onSaveInstanceState()para guardar y recuperar datos.

Expone la información a través de un LiveData observado desde la vista.

Room:

Librería ORM (Object-Relational mapping): convierte objetos SQLite a objetos Java/Kotlin automáticamente.

Validación SQL en tiempo de compilación.

Devuelve objetos LiveData para observar cambios en la base de datos.

Navigation

Se refiere a las interacciones que permiten a los usuarios navegar a través, dentro y fuera de las diferentes piezas de contenido de la app. El componente Navigation de Android Jetpack permite implementar la navegación, desde simples clics de botones hasta patrones más complejos, como las barras de apps y los paneles laterales de navegación. El componente Navigation también garantiza una experiencia del usuario coherente y predecible.

Los iremos conociendo y utilizando a partir de ahora.