

A thick dark blue vertical bar is positioned on the left side of the page. Below it, several thin, curved lines in shades of blue and grey sweep upwards from the bottom left corner.

2 de abril de 2022

Trabajo de investigación

Desarrollo de Software para Móviles

Anthony Alexander Vásquez Iraheta

VI180187

Luis Felipe Coto Arias

CA180315

Grupo de teoría: 03

Docente: Ing. Mario Alvarado

Índice

Objetivo general	1
Objetivos específicos	1
Introducción.....	2
MVP	3
Modelo.....	3
Vista	3
Presentador.....	4
Principales diferenciadores del MVP	4
Ventajas de la arquitectura MVP	4
Desventajas de la arquitectura MVP	5
MVC.....	5
Modelo.....	5
Vista	5
Controlador.....	5
Ventajas.....	6
Desventajas.....	7
Conclusión	8
Sitiografía	9
Anexos	10
Video explicativo	10
Repositorio	10

OBJETIVO GENERAL

- Exponer una selección de patrones de diseño para el desarrollo de aplicaciones en Android.

OBJETIVOS ESPECÍFICOS

- Ejemplificar las ventajas y desventajas de la utilización de los patrones.
- Comparar los patrones presentado.
- Explicar el objetivo y funcionamiento de los patrones.

INTRODUCCIÓN

El desarrollo de software y sistemas tecnológicos ha presentado un incremento exponencial en los últimos años, es por ello que muchas personas se han visto envueltas en este sector tan crucial para las sociedades.

El ingreso de un considerable volumen de personas al mundo tecnológico ha afectado de forma positiva el área, ya que se han podido desarrollar, diseñar y crear una amplia variedad de herramientas que agilizan e innovan la forma de vivir de nuestro día a día.

Por otro lado, el aumento de individuos interesados en el sector conlleva que existan una cantidad considerable de diferentes maneras de diseñar los sistemas que se implementan, y también implica una amplia variación en la forma en como se codifican estas herramientas/software, lo cual deriva en problemas y limitantes de agilidad para el desarrollo y mantenimiento de los proyectos.

Dado que la agilidad es una característica esencial para el desarrollo, era necesario que se diseñaran estándares que definieran un patrón a seguir, el cual a la larga facilitaría garantizar la escalabilidad, mantenimiento y corrección de los sistemas. La existencia de esta problemática dio lugar al nacimiento de los patrones de diseño/desarrollo de software, los cuales otorgan una serie de pasos y reglas a seguir que permiten optimizar el avance de los proyectos informáticos, ayudando también a potenciar el uso correcto de las herramientas de desarrollo.

MVP

MVP (Model-View-Provider) surgió como una alternativa a la arquitectura tradicional MVC (Model-View-Controller). Al usar MVC como arquitectura de software, los desarrolladores enfrentan los siguientes desafíos:

- La mayor parte de la lógica comercial básica reside en el controlador. Durante la vida útil de la aplicación, este archivo crece y el código se vuelve difícil de mantener.
- Dado que la interfaz de usuario y el mecanismo de acceso a los datos están estrechamente vinculados, la capa del controlador y la capa de visualización residen en la misma actividad o segmento. Esto genera problemas al cambiar la funcionalidad de la aplicación.
- Es muy difícil probar las diferentes clases, ya que la mayoría de las partes para probar requieren componentes del SDK de Android.

El patrón MVP supera los desafíos de MVC y proporciona una manera fácil de estructurar el código del proyecto. La razón por la que MVP es tan ampliamente aceptado es que proporciona modularidad, capacidad de prueba y una base de código más limpia y fácil de mantener. Incluye los siguientes tres componentes:

Modelo

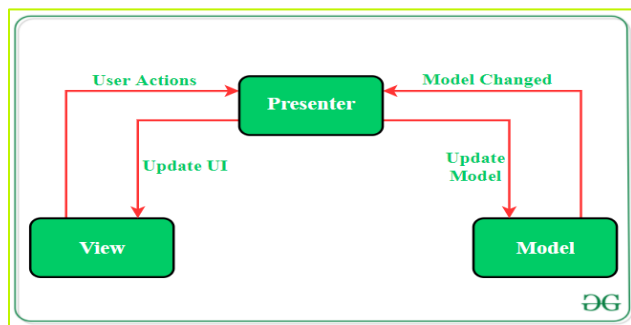
La capa de almacenamiento de datos. Es responsable de administrar la lógica del dominio (reglas comerciales del mundo real) y la comunicación con la base de datos y las capas de red.

Vista

Clase de interfaz de usuario. Proporciona visualización de datos y mantiene registros de las acciones de los usuarios para informar al Presentador.

Presentador

Obtiene los datos del modelo y aplica la lógica de la interfaz de usuario para decidir qué mostrar. Administra el estado de la vista y toma medidas en función de la entrada del usuario desde la vista.



Principales diferenciadores del MVP

- La comunicación entre Presentador-Vista y Presentador-Modelo se realiza a través de una interfaz (también llamada contrato).
- La clase de presentador ejecuta una vista a la vez, es decir, existe una relación 1-1 entre el presentador y la vista.
- Las clases Modelo y Vista no tienen conocimiento de la existencia de la otra.

Ventajas de la arquitectura MVP

- No existe una relación conceptual en los componentes de Android.
- Probar y mantener el código es fácil porque el modelo de aplicación, la capa de la vista y la capa de presentador están separados.

Desventajas de la arquitectura MVP

- Si el desarrollador no sigue el principio de la responsabilidad única de descifrar el código, la capa de presentador tiende a convertirse en una clase gigante que lo sabe todo.

MVC

El patrón MVC (Modelo-Vista-Controlador) consiste en seccionar el código de nuestras aplicaciones en 3 partes diferentes: modelo, vista y controlador.

Modelo

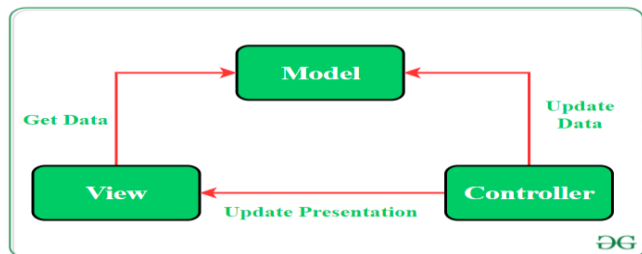
Esta sección se utiliza para almacenar todo lo relacionado con los datos utilizados en la aplicación. Se caracteriza por ser el encargado de organizar y manejar la lógica de las reglas comerciales de la aplicación, y de la comunicación que ésta posee con las bases de datos y capas de red. Es importante aclarar, que todo lo anteriormente mencionado se realiza sin tener conocimiento previo sobre la interfaz de la aplicación.

Vista

Esta sección le corresponde a la interfaz de usuario (IU), la cual contiene todos los elementos visibles en la pantalla de nuestra aplicación. En esta parte, se presentan los datos almacenados en la sección de Modelo y se brinda interacción con el usuario.

Controlador

Esta sección ayuda a construir la relación entre las dos secciones anteriores (modelo y vista), aportando la lógica de la aplicación central e informando de la interacción del usuario. Además, actualiza el Modelo como sea necesario.



Al aplicar el MVC es recomendado adoptar un enfoque de aplicación, el cual nos ayudará a mantener cierta relación entre cada una de las capas de código. Existen dos enfoques esenciales, donde el primero consiste en definir que las actividades y fragmentos puedan hacer la función de controlador y ser a la vez los encargados de actualizar la interfaz; y por otra parte el segundo sugiere que las actividades y fragmentos se utilicen como vistas y controlador, en tanto que el modelo será una clase separada que no extiende ninguna clase de Android.

Dentro de la arquitectura Modelo-Vista-Controlador, el controlador ayuda a actualizar todos los datos de la aplicación y la vista los recibe. Debido a que el Modelo no tiene conocimiento sobre la interfaz, este puede ser probado independientemente de los demás. En el caso de que la vista de la aplicación cumpla con el conocido como principio de responsabilidad única, entonces su función será mantener actualizado al controlador dentro de cada uno de los eventos y solo mostrar datos del Modelo, evitando el tener que aplicar alguna regla de lógica comercial. Solo en este caso extraordinario es que las pruebas de la interfaz de usuario son suficientes como para abordar todas las funcionalidades necesarias de la vista.

Ventajas

- Gracias a la división de responsabilidades en secciones, permite mantener el código de nuestras aplicaciones más limpio, simple y fácil de darle mantenibilidad.
- Es óptimo para desarrollar en equipo, ya que al poder delegar cada una de las tareas en paralelo, se puede aumentar la velocidad de producción.
- Se pueden generar diferentes vistas bajo un mismo modelo, permitiendo aprovechar mejor lo desarrollado con anterioridad y garantizando que exista cierta consistencia entre cada una de ellas.
- Otorga cierta facilidad para realizar pruebas unitarias.

Desventajas

- Requiere de una arquitectura inicial sobre la cual posteriormente se construyen las clases e interfaces que dan lugar a la comunicación y modificación de los módulos de la aplicación.
- Cierta dependencia entre las capas a pesar de ser aplicado correctamente.
- Debido a que es un patrón orientado a objetos, su implementación representa una dificultad añadida cuando se aplica en lenguajes que no siguen ese paradigma.
- Carencia de un parámetro que dicte cómo mostrar correctamente los datos.

MVC VS MVP	
MVC	MVP
La capa del controlador y la de la vista se encuentran en la misma actividad/fragmento.	La comunicación entre el Vista-Presentador y el Presentador-Modelo se realiza a través de una interfaz.
Las entradas del usuario son manejadas por el Controlador que instruye al modelo para las operaciones posteriores.	Las entradas del usuario son manejadas por View, que instruye al presentador para que llame a las funciones apropiadas.
El Controlador es el encargado general ya que crea la Vista apropiada e interactúa con el Modelo de acuerdo a la solicitud del usuario.	La Vista es la responsable general en este esquema ya que la Vista llama a los métodos del Presentador que a su vez dirige el Modelo.
Apoyo limitado a las pruebas unitarias.	Las pruebas unitarias cuentan con un gran apoyo. ¹

Según las encuestas y las preferencias de los internautas dedicados al desarrollo de aplicativos para Android, es preferible implementar el patrón MVP ya que facilita y mejora la modularidad de los sistemas, y gracias a su funcionalidad bidireccional, favorece la implementación de pruebas unitarias para validar la calidad de los sistemas y detectar fallos en los mismos.

¹ Fuente: Geeks for Geeks

CONCLUSIÓN

Los patrones de diseño juegan un papel crucial en la realización de sistemas informáticos de alto nivel, ya que, gracias a sus paradigmas de flujo de información y codificación, promueven el correcto uso de herramientas designadas a la creación de software. Además, proveen una guía avanzada para el aseguramiento de la eficiencia, agilidad y limpieza de los procesos de producción de aplicativos, la cual varía levemente según el enfoque que plantea cada uno de los diferentes patrones, en donde la selección de que patrón implementar dependerá de los objetivos de los proyectos y capacidades de los equipos de desarrollo.

SITIOGRAFÍA

(s.f.). Obtenido de Junta de andalucia:

<https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122>

(29 de Octubre de 2020). Obtenido de Geeks for geeks:

<https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/>

NGuerrero. (5 de Febrero de 2021). Obtenido de Programa en linea:

<https://www.programaenlinea.net/que-es-el-patron-mvc/>

ANEXOS

Video explicativo

<https://drive.google.com/file/d/10XVtfdQmWVTbPwMWyDNdvgSviyfHzWTA/view>

Repositorio

https://github.com/luiscotoo/Trabajo_Investigacion_2_CA180315_VI180187

Usuarios de Github

Anthony Alexander Vásquez Iraheta - **AnthonyVas**

Luis Felipe Coto Arias - **luiscotoo**