Trabajo de investigación

1 de abril de 2022

Desarrollo de Software para Móviles

Anthony Alexander Vásquez Iraheta	VI180187
Luis Felipe Coto Arias	CA180315

Grupo de teoria: 03

Docente: Ing. Mario Alvarado

Indice

OBJETIVO GENERAL

- Exponer los diferentes patrones de diseño para sistemas para android

OBJETIVOS ESPECÍFICOS

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

INTRODUCCION

El desarrollo de software y sistemas tecnológicos no tenia mas que crecimiento 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 es específicamente al desarrollo de software ha sido muy beneficio para el sector ya que sean podido desarrollar, diseñar y crear herramientas que agilizan y innovan la forma de vivir de nuestro día a día.

Por el otro lado, al tener un gran volumen de personas ingresando al sector conlleva a que hayan variables muchas variables de los sistemas que se implementan y también una amplia variación en la forma en como se codifican estas herramientas/software, lo cual deriva en problemas y limitantes de agilidad para la implementación y desarrollo de los proyectos.

Claramente si se quería mantener la agilidad del desarrollo, debía diseñarse estándares que obligaran a seguir un patrón que a la larga facilitaría la escalabilidad, manteamiento y corrección de los sistemas, dado esta necesidad nacen los patrones o arquitecturas de diseño/desarrollo de software que viene a dictarnos ciertos patrones que debemos seguir para optimizar el desarrollo de los sistemas y 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-Provider). 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

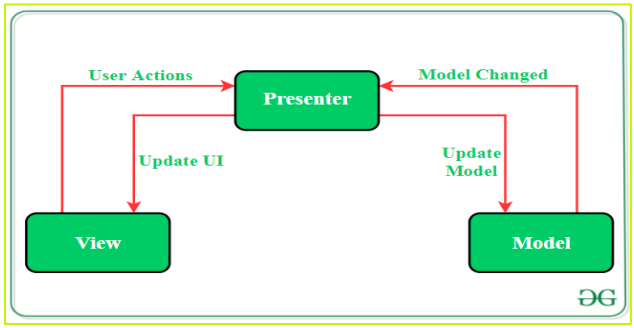
La capa de almacenamiento de datos. Es responsable de administrar la lógica del dominio (reglas comerciales del mundo real) y la comunicación con la base de datos y las capas de red.

Vista

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

Presentador

Obtiene los datos del modelo y aplique la lógica de la interfaz de usuario para decidir qué mostrar. Administra el estado de la vista y tome 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 un 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

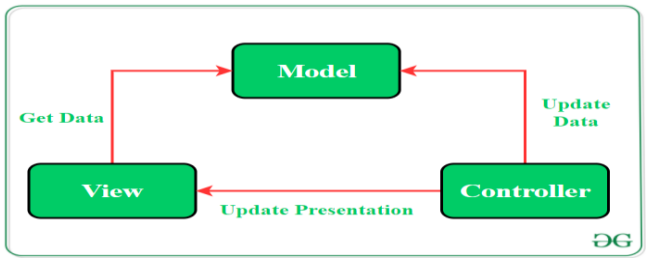
Está sección se utiliza para almacenar todo lo relacionado con los datos utilizados en la aplicación. Se caracteriza por ser el encargado de organizar y manejar la lógica de las reglas comerciales de la aplicación, y de la comunicación que ésta posee con las bases de datos y capas de red. Es importante 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 View-Presenter y el Presenter-Model 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 facilita la implementación de pruebas unitarias para validar la calidad de los sistemas y detectar fallos en los mismos.

[1] Fuente: Geeks for Geeks

CONCLUSION

Los patrones de diseño juegan un papel crucial a la hora que se requiere realizar sistemas informáticos de alto nivel ya gracias a sus paradigmas de flujo de información y codificación facilitan el correcto uso de las herramientas para creación de software y además da herramientas para crear sistemas eficientes y a la medida de las necesidades de los solicitantes.

SITIOGRAFIA

(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