UNIVERSIDAD DON BOSCO

DESARROLLO DE SOFTWARE PARA ANDROID

Alumnos:

Javier Eliseo Gutiérrez Flores – GF220089

William Ernesto Ramos Valladares – RV200068

Kenya Elizabeth Parada Palma – PP220664

Axel Giovanni Ramirez Alfaro – RA160395

DOCENTE:

Alexander Siguenza

Índice

Contenido

Índice

La idea principal detrás del MVP es que los desarrolladores deben crear un producto lo suficientemente básico como para permitir a los usuarios probarlo, ofrecer retroalimentación y verificar si el producto resuelve un problema real para ellos. De esta manera, los desarrolladores pueden minimizar el riesgo de inversión y ahorrar tiempo y dinero, al centrarse solo en las funcionalidades que realmente importan.

El MVP es una estrategia de desarrollo que se enfoca en crear un producto simple, pero funcional, que permita a los desarrolladores recopilar información valiosa de los usuarios y ajustar el producto en consecuencia, lo que puede resultar en un producto final más sólido y exitoso.

¿Qué es el patrón MVP?

Modelo-vista-presentador

Es un formato de distribución para la puesta en práctica de la interfaz de usuario, se reduce la complejidad de uso de módulos, es un enfoque de creación de producto con mínimas características y funcionalidades esenciales, en lugar de invertir gran cantidad de tiempo en la creación de un producto completo el enfoque de MVP se centra en la construcción de producto básico y funcional, es comúnmente utilizado en startups y empresas que buscan reducir el riesgo de inversión. Al crear MVP se debe tener un entendimiento más profundo de las necesidades del cliente lo que ayuda a tomar decisiones más específicas y eficaces.

¿Cuáles son sus componentes principales y cómo se relacionan entre sí?

Modelo:

Es el componente del sistema que depende del acceso a la base de datos, lo más común es una memoria caché o conexiones a un web service pero siempre teniendo la característica de tener las conexiones y peticiones a servicios de base de datos.

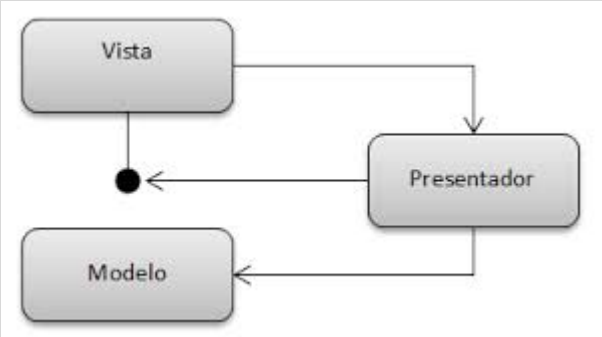
Vista:

Hace referencia a la visualización del diseño de la interfaz, lo que vemos e interactuamos, este componente del producto se encarga de mostrar el contenido dependiendo de las peticiones, algo simple para exhibir fragmentos o actividades.

Presentador:

Es el puente ente la Vista y el Modelo entre este se da la interacción de petición de información, así como su almacenamiento

Su relación va desde la primera generación de la petición del usuario a través de la Vista, luego gracias al Presentador esta solicitud llega al Modelo que ya dependiendo de la petición que se haya requerido se ejecutan las acciones y se muestra el contenido gracias al Presentador que manda la respuesta a la Vista.



¿Cómo se aplica el patrón MVP en Android con Kotlin?

El patrón MVP (Model-View-Presenter) es una arquitectura de software comúnmente utilizada en el desarrollo de aplicaciones móviles para separar las responsabilidades de la lógica de negocio y la interfaz de usuario. En Android con Kotlin, el patrón MVP se puede implementar de la siguiente manera:

1. Modelo (Model): La capa de Modelo es responsable de manejar la lógica de negocio y los datos de la aplicación. Esta capa puede incluir una o varias clases que se encarguen de la gestión de los datos, como la comunicación con una base de datos o una API.

2. Vista (View): La capa de Vista es responsable de mostrar los datos de la aplicación al usuario. Esta capa puede incluir una o varias actividades, fragmentos o vistas personalizadas que se encarguen de mostrar los datos de manera adecuada al usuario.

3. Presentador (Presenter): La capa de Presentador actúa como intermediario entre la capa de Modelo y la capa de Vista. Esta capa puede incluir una o varias clases que se encarguen de la comunicación entre ambas capas, recibiendo los datos del modelo y proporcionándolos a la vista. El presentador también puede ser responsable de manejar eventos de usuario y actualizar el modelo en consecuencia.

En la implementación de MVP en Android con Kotlin, se pueden seguir algunos pasos como:

1. Crear una interfaz para la vista, que define los métodos que el presentador utilizará para comunicarse con la vista.
2. Crear una clase para la vista que implemente la interfaz de la vista y defina la lógica para mostrar los datos en la interfaz de usuario.
3. Crear una clase para el presentador que se encargue de manejar los datos de la aplicación y la comunicación entre la vista y el modelo. El presentador debe implementar la interfaz de la vista para poder comunicarse con ella.
4. Crear una clase para el modelo que maneje los datos de la aplicación y su almacenamiento.
5. Conectar las capas de la aplicación: el presentador debe comunicarse con el modelo para obtener los datos y actualizar la vista para mostrarlos al usuario.

En resumen, el patrón MVP se puede aplicar en Android con Kotlin mediante la separación de responsabilidades entre la capa de Modelo, Vista y Presentador, lo que ayuda a organizar y mantener una aplicación escalable, mantenible y fácilmente testeable.

¿Cuáles son las ventajas y desventajas de utilizar el patrón MVP en el desarrollo de aplicaciones móviles?

El patrón MVP (Modelo-Vista-Presentador) es un enfoque popular en el desarrollo de aplicaciones móviles que separa la lógica de negocio de la interfaz de usuario. A continuación, se presentan algunas ventajas y desventajas de utilizar este patrón en el desarrollo de aplicaciones móviles:

Ventajas:

1. Separación clara de responsabilidades: El patrón MVP separa la lógica de negocio de la interfaz de usuario, lo que hace que el código sea más fácil de entender y mantener.

2. Pruebas más fáciles: Como la lógica de negocio se separa de la interfaz de usuario, es más fácil probar la aplicación de manera unitaria.

3. Flexibilidad: MVP permite una mayor flexibilidad en el desarrollo de aplicaciones, ya que los desarrolladores pueden cambiar la vista y el presentador sin afectar el modelo.

4. Mejora la escalabilidad: MVP ayuda a mejorar la escalabilidad de la aplicación al permitir que la interfaz de usuario y la lógica de negocio se desarrollen y cambien de manera independiente.

Desventajas:

1. Aumento de la complejidad: MVP puede aumentar la complejidad del código, ya que se necesitan más clases para implementar la separación de la lógica de negocio y la interfaz de usuario.

2. Mayor tiempo de desarrollo: Debido a la separación de responsabilidades, MVP puede llevar más tiempo para desarrollar que otros patrones.

3. Curva de aprendizaje: MVP requiere una comprensión clara de los roles de la vista, el presentador y el modelo, lo que puede requerir una curva de aprendizaje para los nuevos desarrolladores.

4. Requiere más código: Debido a la separación de responsabilidades, se necesita más código para implementar MVP en comparación con otros patrones.

En general, el patrón MVP puede ser beneficioso en el desarrollo de aplicaciones móviles en términos de modularidad y escalabilidad, pero puede aumentar la complejidad y el tiempo de desarrollo. Depende del equipo de desarrollo determinar si el uso del patrón MVP es apropiado para su proyecto y cómo puede ajustarse a sus necesidades específicas.

Bibliografía

¿Qué es el Modelo-Vista-Presentador (MVP)? (keepcoding.io)

https://medium.com/@ricardo.ramos/domina-el-patr%C3%B3n-de-dise%C3%B1o-modelo-vista-presentador-mvp-android-d0d5ef419e7

https://learn.microsoft.com/es-es/archive/msdn-magazine/2011/december/mvpvm-design-pattern-the-model-view-presenter-viewmodel-design-pattern-for-wpf