Pablo González Pérez - Jericó Luzardo Miranda
Fecha: 15/10/2024

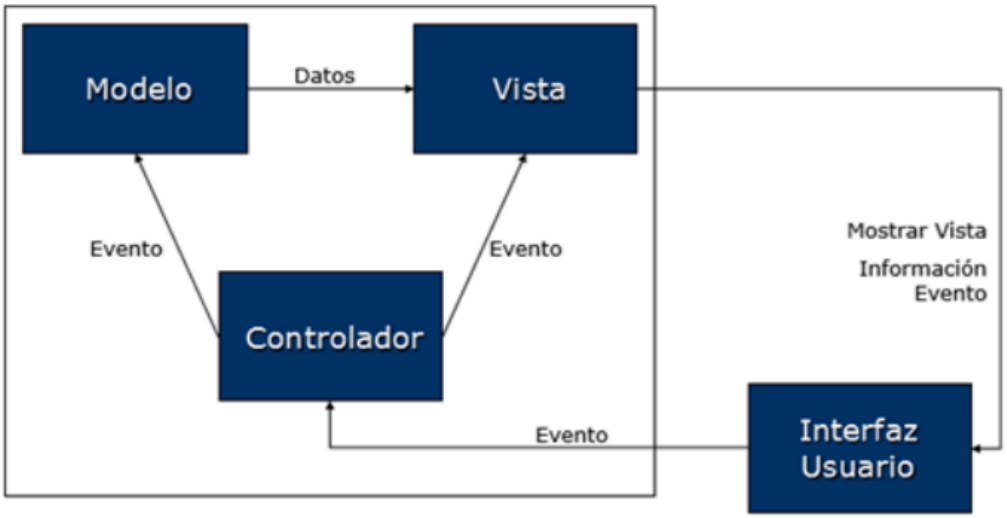
Ejercicio
Comparación de Arquitecturas de Software

Parte 1: Estudio Comparativo

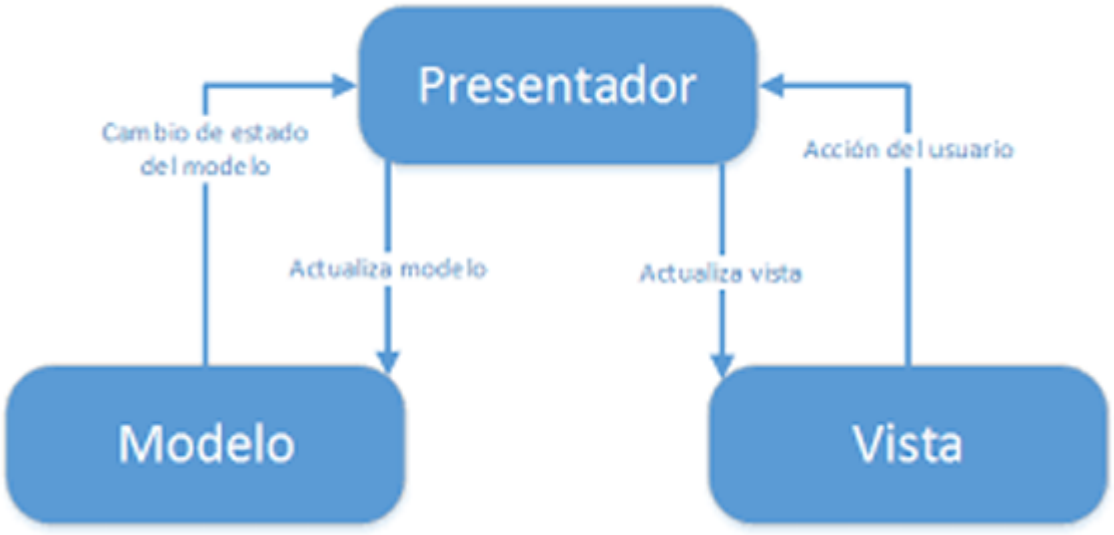
Deben realizar un análisis teórico de las arquitecturas de software MVC, MVP, y MVVM. Esto incluye:

1. **Describir la estructura básica de cada arquitectura.**

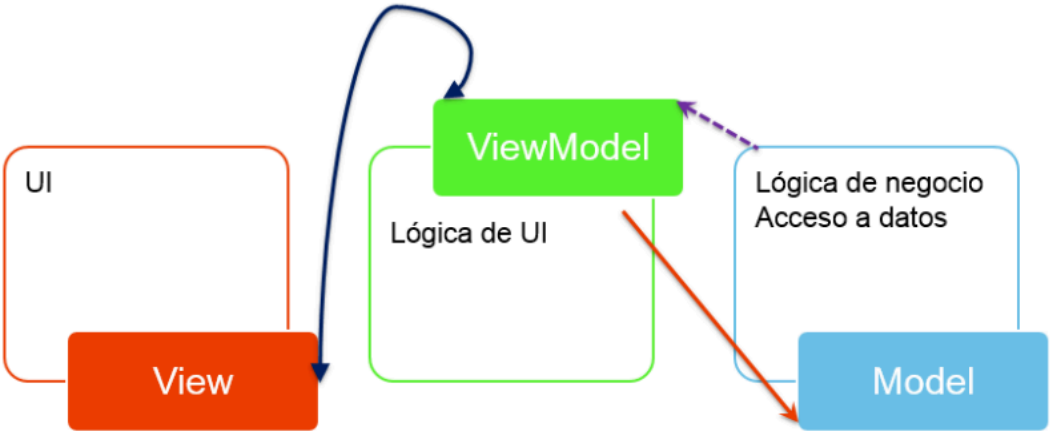
- **MVC:** Divide la aplicación en 3 partes: Modelo (datos), Vista (UI) y Controlador (lógica que comunica ambos mediante eventos).



- **MVP:** Es bastante similar a la arquitectura MVC, pero en vez de Controlador tiene un Presentador (gestiona la lógica y comunica ambos mediante interfaces). A diferencia de MVC, si el modelo quiere enviar datos a la vista debe pasar por el presentador.



- **MVVM:** Utiliza la misma separación que MVP, pero el ViewModel gestiona los datos y la lógica, exponiéndolos directamente a la vista (View) a través de la vinculación de datos.



2. **Definir los roles y responsabilidades de cada componente (Modelo, Vista, Controlador/Presentador/ViewModel).**

    - **Modelo:** Gestiona los datos y la lógica de negocio.
    - **Vista:** Muestra los datos al usuario.
    - **Controlador:** Recibe la entrada del usuario y actualiza el modelo o la vista.
    - **Presentador:** Intermediario entre el modelo y la vista.
    - **ViewModel:** Gestiona la lógica y expone los datos a la vista a través de la vinculación de datos.

3. **Comparar el nivel de acoplamiento y cohesión entre los componentes.**

- **<u>Acoplamiento</u>**
    - **MVC:** El acoplamiento entre el **Controlador** y la **Vista** suele ser más fuerte en comparación con otras arquitecturas, ya que el controlador está estrechamente ligado a la vista, manejando tanto la lógica como las respuestas de la interfaz de usuario. Esto hace que haya más dependencia entre ambos.

    - **MVP:** El acoplamiento se reduce porque el **Presentador** no interactúa directamente con la **Vista**, sino que lo hace a través de una interfaz. Esto disminuye la dependencia entre la Vista y el Presentador, permitiendo que los componentes se cambien más fácilmente sin afectar al otro.

    - **MVVM:** Suele tener el acoplamiento más bajo, ya que el **ViewModel** no tiene referencias directas a la **Vista**. En su lugar, usa la vinculación de datos (data-binding) para actualizar automáticamente la Vista. Esto resulta en una arquitectura donde los componentes están muy desacoplados y es fácil cambiar la vista sin alterar el modelo ni la lógica.

- **<u>Cohesión</u>**
    - **MVC**: La cohesión es moderada. El **Modelo**, **Vista** y **Controlador** tienen responsabilidades claras, pero el **Controlador** puede estar sobrecargado porque maneja tanto la entrada del usuario como la lógica de aplicación. Esto puede hacer que su cohesión sea un poco menor que en otras arquitecturas.

    - **MVP**:La cohesión es mayor en comparación con **MVC**, ya que el **Presentador** tiene una única responsabilidad: manejar la lógica y la comunicación entre el Modelo y la Vista. Esto separa claramente la lógica de la presentación y reduce la sobrecarga de responsabilidades en el Presentador.

    - **MVVM** tiene una cohesión muy alta, especialmente en el **ViewModel**, ya que este solo se encarga de la lógica de presentación y no tiene ninguna responsabilidad relacionada con la interfaz de usuario. El **ViewModel** expone los datos y comandos necesarios para la Vista, manteniendo una responsabilidad única y clara, lo que facilita su mantenimiento y evolución.

4. **Evaluar la facilidad de realizar pruebas unitarias en cada arquitectura.**

    - **MVC:** Puede ser más difícil de probar debido a la relación cercana entre Vista y Controlador.

    - **MVP:** Más fácil de probar, ya que el Presentador puede probarse de forma aislada.

    - **MVVM:** Facilita aún más las pruebas, especialmente del ViewModel, que no depende directamente de la Vista.

5. **Analizar la curva de aprendizaje y complejidad.**

- **MVC:** Es la arquitectura más básica y tiene una curva de aprendizaje baja, pero su implementación en aplicaciones complejas puede volverse más difícil de manejar.

- **MVP:** Tiene una curva de aprendizaje moderada por el uso del Presentador, pero es fácil de mantener una vez comprendido.

- **MVVM:** Tiene una curva de aprendizaje más alta debido a la complejidad del Data Binding y la interacción entre la Vista y el ViewModel, pero es muy efectiva en aplicaciones que requieren interfaces de usuario complejas.

6. **Comparar la escalabilidad de las arquitecturas.**

   La escalabilidad de una estructura se relaciona directamente con su nivel de cohesión y acoplamiento. Cuanto mejor cohesión y acoplamiento posea una estructura mayor será su escalabilidad. Por tanto:

   - **MVC**: Escalable, pero puede volverse difícil de manejar en aplicaciones grandes.
   - **MVP**: Ofrece mejor escalabilidad gracias a su separación de responsabilidades.
   - **MVVM**: Es la más escalable y adecuada para aplicaciones con interfaces de usuario complejas.

7. **Presentar ejemplos de casos de uso y aplicaciones prácticas donde cada arquitectura sería más adecuada.**

   - **MVC:** Es ideal para aplicaciones web tradicionales como sistemas de gestión de contenido o aplicaciones CRUD (crear, leer, actualizar, eliminar) que necesitan una estructura clara entre datos y presentación como plataformas de comercio electrónico o blogs. Por lo cual, funciona bien en juegos de navegador donde el servidor controla el estado del juego.

   - **MVP:** Es útil para las aplicaciones móviles, especialmente en Android, ya que facilita la separación de la lógica de presentación y mejora el rendimiento. También, es ideal para aplicaciones de escritorio que necesitan interfaces más complejas..

   - **MVVM:** Es perfecto para las aplicaciones con interfaces dinámicas que necesitan actualizarse en tiempo real como redes sociales. Para ello, se utiliza mucho en aplicaciones móviles avanzadas y en aplicaciones web modernas que necesitan una interfaz rápida como herramientas de videollamadas.

Parte 2: Análisis de un Escenario Práctico

El escenario práctico propuesto es el diseño de una aplicación de red social. Deben elegir una de las arquitecturas y justificar por qué es la más adecuada para este tipo de aplicación. El análisis puede incluir:

Para una aplicación de red social, consideramos que la mejor opción sería MVVM.

1. **¿Cómo el MVVM maneja las interacciones del usuario con la interfaz?**

   En una red social, los usuarios interactúan mucho con listas de amigos, publicaciones y notificaciones que cambian constantemente. Por tanto, MVVM facilita todo esto gracias al Data Binding, que permite que la interfaz se actualice automáticamente en tiempo real sin que los desarrolladores tengan que intervenir tanto.

2. **¿Cómo se facilita el mantenimiento del código al usar MVVM?**

   Con MVVM, el código está muy bien organizado, ya que separa completamente la lógica de modelo de la vista mediante el uso del ViewModel como intermediario, es decir, si es necesario actualizar la interfaz o cambiar algo en el sistema, se puede hacer sin afectar otras partes del código, lo que hace que la aplicación sea más fácil de mantener y escalar.

3. **Consideraciones de rendimiento de la arquitectura.**

   Consideramos que el Data Binding ayuda a mejorar el rendimiento, ya que reduce la cantidad de código para que la interfaz se actualice. Además, MVVM es muy productivo en aplicaciones como las redes sociales, necesitan que la interfaz sea dinámica y se actualice en tiempo real.

Parte 3: Reflexión Personal

Finalmente, deben reflexionar sobre cuál consideran que es la arquitectura más adecuada para proyectos móviles a gran escala y por qué.

Consideramos que MVVM es la arquitectura más adecuada porque tiene buena escalabilidad, cohesión y bajo acoplamiento entre los componentes. Básicamente, es perfecto para un proyecto donde se añaden continuamente nuevas funcionalidades, y si se necesita cambiar algo puntual, no es necesario modificar todo, gracias al bajo acoplamiento.