TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería en Sistemas Computacionales

MATERIA:
Patrones de diseño

TÍTULO ACTIVIDAD:

Investigación patrón de arquitectura MVC

UNIDAD A EVALUAR:
Unidad 4 y 5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:
Ceballos Resendiz Veronica Sarahi 20211759

NOMBRE DEL MAESTRO (A):
Gabriela Lourdes Tapia González

PATRÓN DE ARQUITECTURA MVC

CONCEPTO

Es un patrón de arquitectura de software que separa una aplicación en tres componentes principales interconectados: el Modelo, que representa la lógica de negocio y los datos; la Vista, que maneja la presentación e interfaz de usuario; y el Controlador, que actúa como intermediario coordinando las interacciones entre ambos. Esta separación de responsabilidades facilita la organización del código, el mantenimiento y la escalabilidad de las aplicaciones, convirtiéndose en uno de los patrones más influyentes en el desarrollo de software moderno.

COMPONENTES

El Modelo: es el corazón de la aplicación y tiene la responsabilidad de gestionar los datos y la lógica de negocio. Esto incluye interactuar con la base de datos, implementar las reglas del dominio, validar datos antes de persistirlos y notificar a las vistas cuando los datos cambian. Una característica fundamental del Modelo es que debe ser completamente independiente de la interfaz de usuario, lo que permite su reutilización por diferentes vistas y garantiza que el estado crítico de la aplicación esté protegido y centralizado.

La Vista: Se encarga exclusivamente de la presentación y visualización de la información al usuario. Su trabajo consiste en mostrar los datos que recibe del Modelo a través del Controlador, presentar formularios y elementos de interacción, y actualizar la interfaz cuando los datos cambian. La vista debe ser lo más simple posible, sin contener lógica de negocio ni realizar consultas complejas. Una aplicación puede tener múltiples vistas para un mismo modelo, como una vista web, una vista móvil y una vista de impresión, todas mostrando los mismos datos de diferentes formas. La vista es esencialmente pasiva y no toma decisiones por sí misma.

El Controlador: Actúa como el orquestador de la aplicación, coordinando la interacción entre Modelo y Vista. Recibe las solicitudes del usuario, interpreta las entradas, invoca los métodos apropiados del Modelo para procesar datos, selecciona la Vista adecuada para mostrar y gestiona el flujo general de la aplicación. Es importante destacar que el Controlador maneja la lógica de control pero no la lógica de negocio, que debe permanecer en el Modelo.

FUNCIONAMIENTO

en una aplicación MVC comienza cuando el usuario interactúa con la interfaz, por ejemplo, haciendo clic en un botón o enviando un formulario. La Vista captura esta acción y la envía al Controlador correspondiente. El Controlador procesa la solicitud, determina qué operaciones deben realizarse y llama al Modelo apropiado. El Modelo ejecuta la lógica de negocio necesaria, actualiza su estado interno y, en algunos casos, notifica directamente a las Vistas interesadas sobre los cambios. El Controlador entonces selecciona la Vista apropiada para mostrar el resultado. La Vista consulta al Modelo para obtener los datos actualizados y renderizar la interfaz con esta nueva información. Finalmente, el usuario ve el resultado actualizado en su pantalla. Este ciclo se repite con cada interacción, manteniendo una separación clara de responsabilidades en todo momento.

VENTAJAS

- La separación de responsabilidades que proporciona MVC es quizás su mayor fortaleza. Cada componente tiene un propósito claro y definido, lo que facilita enormemente la comprensión del código, incluso en aplicaciones grandes y complejas.Esta separación también facilita el mantenimiento, ya que los cambios en la interfaz de usuario no afectan la lógica de negocio y viceversa.
- La reutilización de código es otro beneficio significativo. Un Modelo bien diseñado puede ser utilizado por múltiples Vistas y Controladores, lo que significa que puedes tener una aplicación web, una aplicación móvil y una API REST compartiendo la misma lógica de negocio.
- El desarrollo paralelo se vuelve mucho más eficiente con MVC. Diferentes equipos o desarrolladores pueden trabajar simultáneamente en el Modelo, la Vista y el Controlador sin interferir entre sí.
- Las pruebas se simplifican considerablemente porque cada componente puede ser probado de forma independiente. El Modelo, que contiene la lógica crítica del negocio, puede ser probado exhaustivamente sin necesidad de inicializar la interfaz de usuario.

DESVENTAJAS

- Para aplicaciones pequeñas y simples, puede agregar complejidad innecesaria.
- El overhead de la separación estricta significa que tareas simples pueden requerir modificaciones en múltiples archivos.
- Las dependencias entre componentes pueden volverse complejas en aplicaciones grandes.

EVOLUCIONES DEL PATRÓN

El patrón MVC ha inspirado varias variantes que abordan diferentes necesidades y contextos. El patrón MVP (Model-View-Presenter) reemplaza el Controlador con un Presenter que contiene toda la lógica de presentación. En MVP, la Vista es aún más pasiva que en MVC tradicional, y el Presenter maneja toda la comunicación entre el Modelo y la Vista. Esta variante es particularmente popular en aplicaciones de escritorio y móviles porque facilita las pruebas unitarias, ya que el Presenter puede ser completamente independiente de la interfaz de usuario.

MVVM (Model-View-ViewModel) ha ganado enorme popularidad en aplicaciones modernas, especialmente con frameworks como Angular, Vue.js y WPF de Microsoft. El ViewModel actúa como un intermediario especializado entre el Modelo y la Vista, implementando data binding bidireccional. Esto significa que cuando los datos en el ViewModel cambian, la Vista se actualiza automáticamente, y viceversa. Este patrón es ideal para aplicaciones con interfaces ricas e interactivas donde los usuarios modifican datos constantemente.

HMVC (Hierarchical Model-View-Controller) extiende el concepto tradicional permitiendo estructuras jerárquicas de componentes MVC. En lugar de tener una única tríada de Modelo-Vista-Controlador para toda la aplicación, HMVC permite que cada módulo tenga su propio conjunto MVC completo.

IMPLEMENTACIÓN EN DIFERENTES TECNOLOGÍAS

- En el mundo del desarrollo web
- Ruby on Rails fue pionero en popularizar MVC con su filosofía de "convention over configuration"
- ASP.NET MVC de Microsoft trajo el patrón al ecosistema .NET con fuerte integración con Visual Studio y herramientas de desarrollo empresarial
- Laravel en PHP y Django en Python representan implementaciones modernas del patrón
- Laravel ofrece una sintaxis elegante y expresiva manteniendo los principios de MVC
- En aplicaciones móviles
- Apple históricamente ha promovido MVC como el patrón predeterminado para iOS
- En el desarrollo frontend moderno

USO

MVC es especialmente adecuado para aplicaciones web complejas con múltiples páginas donde la lógica de negocio es significativa. Sistemas empresariales que manejan procesos de negocio complejos, múltiples roles de usuario y flujos de trabajo elaborados se benefician enormemente de la estructura que proporciona MVC. Las aplicaciones que requieren múltiples interfaces, como una aplicación web, una app móvil y una API REST, pueden compartir el mismo Modelo, implementando solo diferentes Vistas y Controladores para cada plataforma.la separación clara de responsabilidades permite que diferentes miembros trabajen en paralelo sin conflictos constantes. Las aplicaciones que evolucionarán y escalaran con el tiempo encuentran en MVC una base sólida que facilita agregar nuevas funcionalidades sin romper las existentes. Los sistemas que requieren alto nivel de testing se benefician de la capacidad de probar cada componente independientemente. Sin embargo, MVC no es apropiado para todo. Aplicaciones muy simples como landing pages o sitios estáticos no necesitan la estructura completa de MVC. Prototipos rápidos donde la velocidad es más importante que la arquitectura a largo plazo pueden beneficiarse de enfoques más directos.