Aquitectura Escogida Y porque

Software basado en Componentes

**Reutilización del software.** Nos lleva a alcanzar un mayor nivel de reutilización de software.

**Simplifica las pruebas.** Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

**Simplifica el mantenimiento del sistema.** Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

**Mayor calidad.** Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

BENEFICIOS

Los siguientes son los principales beneficios del estilo de arquitectura basado en componentes:

• **Facilidad de Instalación**. Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.

• **Costos reducidos**. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.

• **Facilidad de desarrollo**. Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.

• **Reusable**. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.

• **Mitigación de complejidad técnica**. Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones.

Modelo vista controlador

**Modelo vista controlador**

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software utilizado en muchas aplicaciones web el cual separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos (modelo, vista y controlador).6

**Modelo:** Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Aquí se operan los datos y las reglas de negocio asociadas al sistema, incluyendo el análisis sintáctico y el procesamiento de los datos de entrada y de los datos de salida.

El Modelo es el responsable de:

Acceder a la capa de almacenamiento de datos. Siendo ideal que el modelo sea independiente del sistema de almacenamiento.

Definir las reglas de negocio (la funcionalidad del sistema).

Llevar registro de las vistas y controladores del sistema.

Si se está ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero *batch* que actualiza los datos, un temporizador que desencadena una inserción, etc.). Un ejemplo de MVC con un modelo pasivo (aquel que no notifica cambios en los datos) es la navegación web, que responde a las entradas del usuario, pero no detecta los cambios en datos del servidor.

**Vista:** Esta presenta el modelo, usualmente la interfaz de usuario. La vista es la capa de la aplicación que ve el usuario en un formato adecuado para interactuar, en otras palabras, es la interfaz gráfica.

Las vistas son responsables de:

Recibir datos del modelo y mostrarlos al usuario.

Llevar un registro de su controlador asociado.

Dar el servicio de “Actualización ()”, para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

**Controlador:** El Controlador es la capa que controla todo lo que puede realizar la aplicación. Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Está compuesto por acciones que se representan con funciones en una clase. 43

El controlador es responsable de:

Recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.).

Contener reglas de gestión de eventos, del tipo “SI Evento Z, entonces Acción W”. Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método “Actualizar()”.Una petición al modelo puede ser “Obtener\_tiempo\_de\_entrega(nueva\_orden\_de\_venta)”

**Ventajas y desventajas:** La popularidad de este diseño se debe más que todo a que es mucho más fácil organizar aplicaciones grandes.

**Las ventajas:**

Clara separación entre interfaz, lógica de negocio y de presentación.

Sencillez para crear distintas representaciones de los mismos datos.

Facilidad para la realización de pruebas unitarias de los componentes, así como de aplicar desarrollo guiado por pruebas.

Reutilización de los componentes.

Simplicidad en el mantenimiento de los sistemas.

Facilidad para desarrollar prototipos rápidos.

Los desarrollos suelen ser más escalables.

**Las desventajas:**

Tener que ceñirse a una estructura predefinida, lo que a veces puede incrementar la complejidad del sistema. Hay problemas que son más difíciles de resolver respetando el patrón MVC.

La curva de aprendizaje para los nuevos desarrolladores se estima mayor que la de modelos más simples como *Webforms*.

La distribución de componentes obliga a crear y mantener un mayor número de ficheros.7

**De una manera más simplificada se tiene que:**

**El Modelo:** se encarga de interactuar con la base de datos y también ejecuta las reglas de negocio.

**La Vista:** es el código HTML que se muestra al usuario, con la información proveniente del controlador.

**El Controlador:** procesa las peticiones de la página web (vista), y envía estos datos a la capa modelo, para que esta le devuelva la información adecuada para mostrarla en la capa vista.8