

Programación de aplicaciones

MI. Ariana Gómez Contreras

Aplicación de Modelo Vista Controlador

**INTEGRANTES:**

Arredondo Flores Diana Vanessa

Arroyo Sánchez Brenda Jessica

Ramírez Morales Juan Armando

¿Qué es el modelo vista controlador?

Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

* El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
* La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
* El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El modelo es responsable de:

* Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
* Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
* Lleva un registro de las vistas y controladores del sistema.
* Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.).

El controlador es responsable de:

* Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
* Contiene 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)".

La vista es responsable de:

* Recibir datos del modelo y la muestra al usuario.
* Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
* Pueden 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).

El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Buenas practicas Modelo Vista Controlador

\*Declarar variables

Declaración de variables C# tipos definidos, ya que muchas veces usamos los tipos definidos en las clases de Framework de .NET

  int var1;  
string var2;  
object var3;

No usar

Int, String, Object, Boolean.

Métodos y propiedades

Evitar métodos y propiedades públicas, a menos que sea estrictamente necesario accederlas desde afuera de la clase. Utilizar el keyword internal si estos miembros deben ser accedidos desde dentro del mismo assembly.

Abrir y cerrar conexiones

Al abrir conexiones a bases de datos, Sockets, Streams, etc. siempre cerrarlos en los bloques finally. Esto asegurará que aún ante la eventualidad de una excepción, estos accesos serán cerrados. Se puede usar el Using en caso de no tener que capturar una excepción en particular dentro del catch.

Declaración de clases

Utilizar Pascal Casing para declarar nombres de clases.  
Utilizar el prefijo “I” con Pascal Casing para nomenclar Interfaces (Ejemplo: IUser).

Indentación, espaciado y agrupación de código

* Utilice TAB de 4 posiciones para indentar código. No utilice espacios.
* Los comentarios deben estar en el mismo nivel que el código, es decir que han de utilizar el mismo nivel de indentación.
* Las llaves de apertura y cierre deben estar en el mismo nivel que el código fuera de éstas.
* Las llaves deben estar en una línea independiente y no en la misma línea que if, for, etc.
* Utilizar un único espacio antes y después de cada operador, así como también entre términos utilizados durante la invocación de métodos.
* Mantener las variables de miembro, propiedades y métodos en la parte superior del archivo, y miembros públicos en la parte inferior del mismo.

Comentarios y notas de código.

* Corroborar ortografía, gramática y semántica de los comentarios, asegurándose de que la puntuación es utilizada correctamente.
* Agregar la documentación del encabezado de cada método (utilizando ///), especificando correctamente:  
  •     El propósito del método  
  •     Nombre, Tipo de datos y Contenido de cada parámetro recibido  
  •     El propósito de la información retornada  
  •  Esto es particularmente útil para generar documentación de forma automática, a partir de estos comentarios