¿Qué es el MVC?

MVC es un patrón de diseño utilizado para desacoplar la interfaz de usuario (vista), los datos (modelo) y la lógica de aplicación (controlador). Este patrón ayuda a lograr la separación de las preocupaciones.

Usando el patrón MVC para sitios web, las solicitudes se enrutan a un controlador que es responsable de trabajar con el modelo para realizar acciones y / o recuperar datos. El Controlador elige la Vista para mostrar y le proporciona el Modelo. La vista representa la página final, en función de los datos del modelo.

Esta delineación de responsabilidades lo ayuda a escalar la aplicación en términos de complejidad porque es más fácil codificar, depurar y probar algo (modelo, vista o controlador) que tiene un solo trabajo. Es más difícil actualizar, probar y depurar código que tiene dependencias distribuidas en dos o más de estas tres áreas. Por ejemplo, la lógica de la interfaz de usuario tiende a cambiar con más frecuencia que la lógica de negocios. Si el código de presentación y la lógica empresarial se combinan en un solo objeto, se debe modificar un objeto que contenga lógica empresarial cada vez que se cambie la interfaz de usuario. Esto a menudo introduce errores y requiere la reevaluación de la lógica empresarial después de cada cambio mínimo de interfaz de usuario.

El Modelo en una aplicación MVC representa el estado de la aplicación y cualquier lógica de negocio u operaciones que deba realizar. La lógica de negocios debe encapsularse en el modelo, junto con cualquier lógica de implementación para mantener el estado de la aplicación. Las vistas fuertemente tipadas generalmente usan tipos de ViewModel diseñados para contener los datos que se mostrarán en esa vista. El controlador crea y completa estas instancias de ViewModel a partir del modelo.

Características

ASP.NET Core MVC incluye lo siguiente:

- Enrutamiento
- Modelo vinculante
- Modelo de validación
- Inyección de dependencia
- Filtros
- Areas
- API web
- Testabilidad
- Motor de vista de maquinilla de afeitar
- Vistas fuertemente tipadas
- Ayudantes de etiqueta
- Ver componentes

Historia

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.

MVC fue introducido por Trygve Reenskaug durante su visita a Xerox Parc en los años 70, seguidamente, en los años 80, Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80. Solo más tarde, en 1988, MVC se expresó como un concepto general en un artículo sobre Smalltalk-80.

En esta primera definición de MVC el controlador se definía como "el módulo que se ocupa de la entrada" (de forma similar a como la vista se ocupa de la salida). Esta definición no tiene cabida en las aplicaciones modernas en las que esta funcionalidad es asumida por una combinación de la vista y algún framework moderno para desarrollo. El controlador, en las aplicaciones modernas de la década de 2000, es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el modelo y la vista, y unifica la validación (utilizando llamadas directas para desacoplar el modelo de la vista en el modelo activo).

Ventajas

1. Podrás dividir la lógica de negocio del diseño, haciendo tu proyecto más escalable.
2. Te facilitará el uso de URL amigables, importantes para el SEO (Posicionamiento web), la mayoría de frameworks MVC lo controlan.
3. Muchos frameworks MVC ya incluyen librerías de Javascript como Jquery, lo que te facilitará validar formularios (Ej. Jquery.Validate) en el cliente y en el servidor.
4. Puedes utilizar abstracción de datos, como lo hace Ruby on Rails o con frameworks como **Hibernate para Java o NHibernate para ASP .NET MVC**, facilitando la realización de consultas a la base de datos.
5. La mayoría de frameworks controlan el uso de la memoria Caché, hoy en día muy importante para el posicionamiento web, ya que buscadores como google dan prioridad a las webs que tengan menor tiempo de descarga.
6. En el caso de proyectos donde hay varios desarrolladores, el seguir métodos comunes de programación, hace que el código sea más entendible entre estos, pudiendo uno continuar el trabajo de otro. En estos casos es conveniente utilizar herramientas de control de versiones como **Subversion**.
7. Los frameworks están creados para facilitar el trabajo de los desarrolladores, encontrarás clases para controlar fechas, URL's, Webservices, etc. lo que tiene una gran ventaja en cuanto a productividad. Inicialmente como es lógico habrá una curva de aprendizaje, pero luego tendrás muchos beneficios.

Desventajas

1. Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo.
2. La separación de conceptos en capas agrega complejidad al sistema.
3. La curva de aprendizaje del patrón de diseño es más alta usando otros modelos más sencillos.
4. La distribución de componentes obliga a crear y mantener un mayor número de ficheros.

Comparaciones con otros tipos de arquitecturas.

MVC significa Modelo Vista Controlador, porque en este patrón de diseño se separan los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. Cuando la lógica de negocio realiza un cambio, es necesario que ella sea la que actualiza la vista.

MVVM significa Modelo Vista - Vista Modelo, porque en este patrón de diseño se separan los datos de la aplicación, la interfaz de usuario, pero en vez de controlar manualmente los cambios en la vista o en los datos, estos se actualizan directamente cuando sucede un cambio en ellos, por ejemplo si la vista actualiza un dato que está presentando se actualiza el modelo automáticamente y viceversa.

"El Patrón Modelo-Vista-Presentador (**MVP**) surge para ayudar a realizar pruebas automáticas de la interfaz gráfica, para ello la idea es codificar la interfaz de usuario lo más simple posible, teniendo el menor código posible, de forma que no merezca la pena probarla. En su lugar, toda la lógica de la interfaz de usuario, se hace en una clase separada (que se conoce como Presentador), que no dependa en absoluto de los componentes de la interfaz gráfica y que, por tanto, es más fácil de realizar pruebas.