ARQUITECTURA MODELO VISTA CONTROLADOR

AUTOR:

Caterin Martinez Badillo

TÍTULO A OBTENER: Tecnologa en Analisis y Desarrollo de Software

INSTRUCTOR:

Alvaro Perez Niño

Servicio de Aprendizaje SENA

Centro de Gestión Empresarial

2501259: Analisis y Desarrollo de Software

Antioquia, Medellín

19 de Septiembre del 2022

¿QUE ES MVC?

El Modelo Vista Controlador (MVC) 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. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo y una mejora de mantenimiento. Algunos otros patrones de diseño se basan en MVC, como MVVM (Modelo-Vista-modelo de vista), MVP (Modelo-Vista-Presentador) y MVW (Modelo-Vista-Whatever).

PORQUE SE USA MVC?

Se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos. Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés. En este artículo estudiaremos con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando lo desarrollen. El MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web.

PARTES DE MVC?

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

- ➔ Modelo: Se encarga de manejar el sistema de datos, su lógica de negocio, y sus mecanismos de persistencia.
- ➔ Vista: Se encarga del diseño y presentación, que compone la información que se le envía al cliente y los mecanismos de interacción con éste, también se le conoce como interfaz de usuario.
- ➔ Controlador: Se encarga de actuar como intermediario entre el Modelo y la Vista, gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

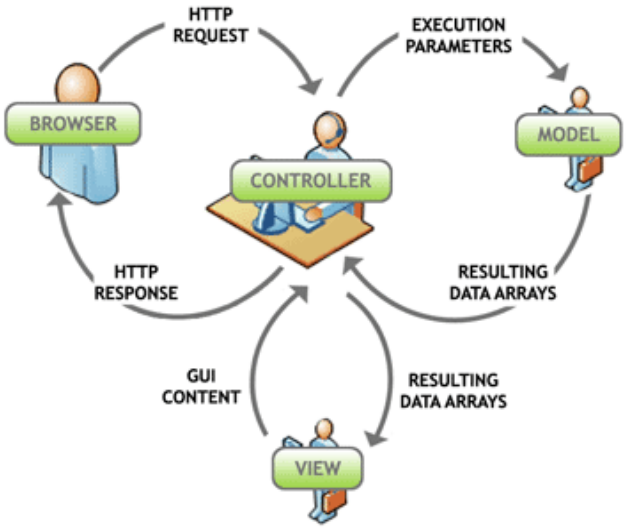
PORQUE SE PREFIERE MVC?

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y el mantenimiento.

Los ingenieros del software se dedican a estudiar de qué manera se pueden mejorar los procesos de creación de software y una de las soluciones a las que han llegado es la arquitectura basada en capas que separan el código en función de sus responsabilidades o conceptos. Por tanto, cuando estudiamos MVC lo primero que tenemos que saber es que está ahí para ayudarnos a crear aplicaciones con mayor calidad.

PASO A PSO DE FLUJO MVC

1. El usuario realiza una solicitud a nuestro sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.
2. El controlador se comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
3. Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación; por eso esa flecha la hemos coloreado de otro color.
4. Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.



MVC EN LA WEB

Como desarrollador web, este tipo de modelo probablemente se considerara bastante familiar, incluso si nunca lo has usado conscientemente antes. Su modelo de datos probablemente esté contenido en algún tipo de base de datos (ya sea una base de datos tradicional del lado del servidor como MySQL, o una solución del lado del cliente como IndexedDB). El código de control de su aplicación probablemente esté escrito en HTML / JavaScript , y su interfaz de usuario probablemente esté escrita usando HTML / CSS / o lo que sea. Esto se parece mucho a MVC, pero MVC hace que estos componentes sigan un patrón más rígido.

En los primeros días de la Web, la arquitectura MVC se implementó principalmente en el lado del servidor, con el cliente solicitando actualizaciones a través de formularios o enlaces, y recibiendo vistas actualizadas para mostrar en el navegador. Sin embargo, en estos días, mucha de la lógica se enviaba al cliente con la llegada de los almacenes de datos del lado del cliente, y XMLHttpRequest permitía actualizaciones parciales de la página según era necesario. Los frameworks de desarrollo web como AngularJS y Ember.js implementan una arquitectura MVC, aunque de maneras ligeramente diferentes.

VENTAJAS DEL MVC

- ➔ Aunque no tenga nada que ver, comencemos con algo tan sencillo como son el HTML y las CSS. Al principio, en el HTML se mezclaba tanto el contenido como la presentación. Es decir, en el propio HTML tenemos etiquetas como "font" que sirven para definir las características de una fuente, o atributos como "bgcolor" que definen el color de un fondo. El resultado es que tanto el

contenido como la presentación estaban juntos y si algún día pretendíamos cambiar la forma con la que se mostraba una página, estábamos obligados a cambiar cada uno de los archivos HTML que componen una web, tocando todas y cada una de las etiquetas que hay en el documento. Con el tiempo se observó que eso no era práctico y se creó el lenguaje CSS, en el que se separó la responsabilidad de aplicar el formato de una web.

- ➔ Al escribir programas en lenguajes como PHP, cualquiera de nosotros comienza mezclando tanto el código PHP como el código HTML (e incluso el Javascript) en el mismo archivo. Esto produce lo que se denomina el "Código Espagueti". Si algún día pretendemos cambiar el modo en cómo queremos que se muestre el contenido, estamos obligados a repasar todas y cada una de las páginas que tiene nuestro proyecto. Sería mucho más útil que el HTML estuviera separado del PHP.
- ➔ Si queremos que en un equipo intervengan perfiles distintos de profesionales y trabajen de manera autónoma, como diseñadores o programadores, ambos tienen que tocar los mismos archivos y el diseñador se tiene necesariamente que relacionar con mucho código en un lenguaje de programación que puede no serle familiar, siendo que a éste quizás solo le interesan los bloques donde hay HTML. De nuevo, sería mucho más fácil la separación del código.
- ➔ Durante la manipulación de datos en una aplicación es posible que estemos accediendo a los mismos datos en lugares distintos. Por ejemplo, podemos acceder a los datos de un artículo desde la página donde se muestra éste, la página donde se listan los artículos de un manual o la página de backend donde se administran los artículos de un sitio web. Si un día cambiamos los datos de los artículos (alteramos la tabla para añadir nuevos campos o cambiar los existentes porque las necesidades de nuestros artículos varían), estamos obligados a cambiar, página a página, todos los lugares donde se consumían datos de los artículos. Además, si tenemos el código de acceso a datos disperso por decenas de lugares, es posible que estemos repitiendo las mismas sentencias de acceso a esos datos y por tanto no estamos reutilizando código.

EJEMPLO SOBRE MVC

Imagine una sencilla aplicación de lista de compras. Todo lo que queremos es una lista del nombre, la cantidad y el precio de cada artículo que necesitamos comprar esta semana. A continuación describiremos cómo podríamos implementar parte de esta funcionalidad usando MVC.

MVC EN JAVA

Es más frecuente en aplicaciones Web que en aplicaciones de escritorio, sin embargo es aplicable también a este, sin ningún problema, Java ya contaba hace rato con Observer y Observable, herramientas que nos ayudan a la interacción entre

la interfaz y el modelo, sin embargo , el ejemplo que dejamos a continuación no hace uso de estas herramientas.

EJEMPLO DE MVC EN JAVA

1. Crea un nuevo proyecto en netbeans, para este ejemplo, el proyecto se llama "jc_mvc_demo". Crea una estructura de paquetes (Controller, Model, View), hacemos esto para separar cada componente, ser más organizados.
2. Empecemos creando la lógica de la aplicación, crea una nueva clase en el paquete Model, llamalo "modelo.java" y añade el código del link.
3. Diseñemos ahora la interfaz de usuario, nuestra VISTA. agregue un JFrame al paquete VIEW, llamelo "vista.java", OJO los recuadros marcados con rojo, son los componentes que tendrán interacción con el modelo, así que para evitar confusiones, es mejor renombrarlos, coloca los nombres tal como se ven en la imagen de abajo.
4. Ahora continuamos con el CONTROLADOR de nuestra aplicación, crea una clase "controlador.java" en el paquete CONTROLLER, el código que debes colocar, es en qué le llegue al link.
5. Para finalizar debe crear el Main y copiar el último código del link.

http://jc-mouse.blogspot.com/2011/12/patron-mvc-en-java-con-netbeans.html

REFERENCIAS:

- https://developer.mozilla.org/es/docs/Glossary/MVC
- https://desarrolloweb.com/articulos/que-es-mvc.html
- https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html