Desarrollo avanzado de aplicaciones II

Tema Nº2:Spring Boot MVC.

Indicador de logro Nº2:Reconoce la estructura en un proyecto con Spring Boot, teniendo en cuenta el patrón de diseño MVC.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº2:**

Spring Boot MVC.

**Subtema 2.1:**

Introducción al patrón de diseño MVC.

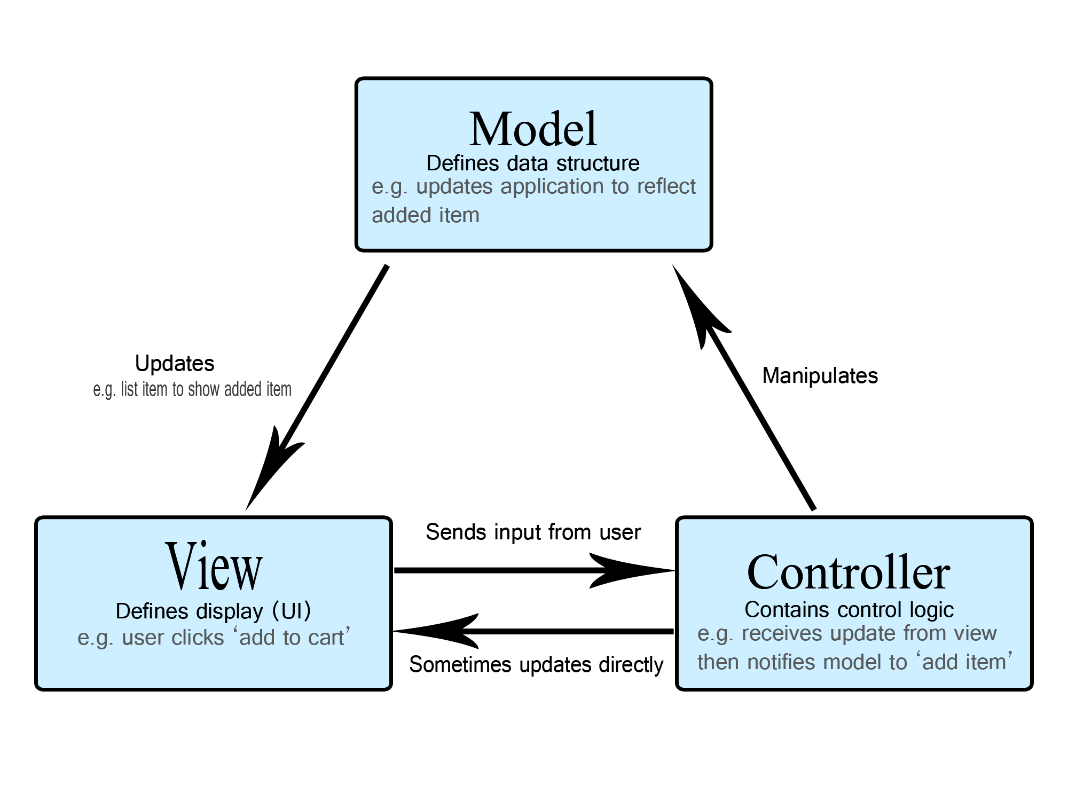
MVC (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo 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).

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

* Modelo: Maneja datos y lógica de negocios.
* Vista: Se encarga del diseño y presentación.
* Controlador: Enruta comandos a los modelos y vistas.

**Modelo Vista Controlador ejemplo**

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.



**Modelo**

El modelo define qué datos debe contener la aplicación. Si el estado de estos datos cambia, el modelo generalmente notificará a la vista (para que la pantalla pueda cambiar según sea necesario) y, a veces, el controlador (si se necesita una lógica diferente para controlar la vista actualizada).

Volviendo a nuestra aplicación de lista de compras, el modelo especificará qué datos deben contener los artículos de la lista (artículo, precio, etc.) y qué artículos de la lista ya están presentes.

**Vista**

La vista define cómo se deben mostrar los datos de la aplicación.

En nuestra aplicación de lista de compras, la vista definiría cómo se presenta la lista al usuario y recibiría los datos para mostrar desde el modelo.

**Controlador**

El controlador contiene una lógica que actualiza el modelo y / o vista en respuesta a las entradas de los usuarios de la aplicación.

Entonces, por ejemplo, nuestra lista de compras podría tener formularios de entrada y botones que nos permitan agregar o eliminar artículos. Estas acciones requieren que se actualice el modelo, por lo que la entrada se envía al controlador, que luego manipula el modelo según corresponda, que luego envía datos actualizados a la vista.

Sin embargo, es posible que también se desee actualizar la vista para mostrar los datos en un formato diferente, por ejemplo, cambiar el orden de los artículos de menor a mayor precio o en orden alfabético. En este caso, el controlador podría manejar esto directamente sin necesidad de actualizar el modelo.

**MVC en la web**

Como desarrollador web, este patrón probablemente será 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

**Subtema 2.2:**

Conociendo la estructura de mi proyecto Spring MVC.

**Ejemplos:**

1. Directorio donde se podrá crear los paquetes, clases e interfaces java.
2. Directorio donde se podrá agregar recursos como imágenes, hojas de estilos, archivos javascript, páginas html y JSP.
3. Directorio para crear nuestras pruebas unitarias de la aplicación.
4. Directorio donde se encuentran todas las dependencias añadidas a nuestro proyecto.
5. Archivo donde podremos realizar la configuración de la aplicación y agregar dependencias a nuestro proyecto.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Subtema 2.3:**

Maven y dependencias.

Maven es una herramienta open-source, que se creó en 2001 con el objetivo de simplificar los procesos de build (compilar y generar ejecutables a partir del código fuente).

Antes de que Maven proporcionara una interfaz común para hacer builds del software, cada proyecto solía tener a alguna persona dedicada exclusivamente a configurar el proceso de build.

Además, los desarrolladores tenían que perder tiempo en aprender las peculiaridades de cada nuevo proyecto en el que participaban.

Si queríamos compilar y generar ejecutables de un proyecto, teníamos que analizar qué partes de código se debían compilar, qué librerías utilizaba el código, dónde incluirlas, qué dependencias de compilación había en el proyecto.

En el mejor de los casos, se empleaban unos pocos minutos para saber cómo hacer una build del proyecto. En el peor de los casos, el proceso de build era tan complejo que un desarrollador podía tardar horas en saber cómo compilar y generar los ejecutables a partir del código.

Por otra parte, con Maven la gestión de dependencias entre módulos y distintas versiones de librerías se hace muy sencilla. En este caso, solo tenemos que indicar los módulos que componen el proyecto, o qué librerías utiliza el software que estamos desarrollando en un fichero de configuración de Maven del proyecto llamado POM.

Además, en el caso de las librerías, no tienes ni tan siquiera que descargarlas a mano. Maven posee un repositorio remoto (Maven central) donde se encuentran la mayoría de librerías que se utilizan en los desarrollos de software, y que la propia herramienta se descarga cuando sea necesario.

<https://mvnrepository.com/>

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Subtema 2.4:**

Principales anotaciones de Spring

* @Controller: La anotación @Controller se usa para indicar que la clase es un controlador Spring. Esta anotación se puede utilizar para identificar controladores para Spring MVC o Spring WebFlux.
* @ModelAtributte: Permite realizar un binding de los datos que tenemos en un formulario Html con el controlador Java.
* @RequestMapping: Es una de las anotaciones más usada en Spring MVC, que nos redirecciona a un conjunto de formularios.
* @RequestParam: Permite extraer parámetros de consulta, parámetros de formulario e incluso archivos de la solicitud.
* @PathVariable: Maneja variables de plantilla en la asignación de URI de solicitud y las establece como parámetros de método.

**Subtema 2.5:**

Interacción entre controlador y formularios HTML.

**Ejemplos:**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Actividad:**

a) CUESTIONARIO TÉCNICO

* Defina cada elemento del MVC.
* En que directorio de nuestro proyecto Spring se agregan las vistas.
* Para que sirve la anotación @Controller.
* Para que sirve la anotación @RequestMapping.
* Cree un proyecto con Spring Boot y agregue 2 controladores, cada controlador tendrá 2 acciones que invocarán a sus respectivas vistas.

b) CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_