**MVC – Modelo Vista Controlador**

Patrón de diseño para crear aplicaciones Web utilizado en la mayoría de frameworks de desarrollo.

Las aplicaciones se dividen en tres partes o módulos principales:

* **Controlador:** Recoge las peticiones del usuario, las procesa y realiza una consulta al modelo. Cuando el modelo le devuelve la información requerida, el controlador la envía a una vista para que finalmente el usuario reciba la información.
* **Modelo:** Almacena los datos y la información. En la mayoría de los casos se trata de una Base de Datos. El modelo se encarga de proporcionar al controlado la información que éste le requiere.
* **Vista:** Es la interfaz o pagína web que muestra el resultado de la petición que ha realizado el usuario. Es decir, es donde el controlador envía la respuesta a su petición. El usuario recibe dicha información a través de la vista.

Ventajas:

* Favorece el desarrollo al dividir la aplicación en partes.
* Favorece el mantenimiento, por la misma razón.
* Favorece la depuración
* Favorece la escalabilidad (ampliación de funcionalidades)

Con Spring:

* El controlador se corresponde siempre con un Servlet.
* El modelo se corresponde, en la mayoría de los casos, con una base de datos; aunque también puede utilizarse una colección.
* La vista puede construirse con diversas tecnologías pero en la mayoría de los casos se construye con JSP (Java Service Pages).

**INTERFAZ MODEL**

Agrega información al modelo. Tiene un método principal sobrecargado:

**addAtribute(Object attributeValue)**

**addAtribute(String attributeName, Object attributeValue)**

**ANOTACIONES SPING**

@RequestMapping (/urlPeticion): Indica el método que se devuelve con la petición asociada a esa URL.

**API**

Interfaz de Programación de Aplicaciones: Conjunto de reglas, funcionalidades o recursos que permite a diferentes aplicaciones externas interactuar con una determinada aplicación. Define como se pueden solicitar y compartir los datos y funcionalidades entre diferentes sistemas de software (aplicaciones web, m´voil, de escriotrio, videojuegos…).

Permite separar la fuente de datos (generalmente asociado al back-end) de la presentación de dichos datos al usuario final (usualmente llamado front-end). Ello permite comunicar dos tecnologías totalmente distintas sin necesidad de conocer como funciona cada una de ellas; basta con conocer las funcionalidades de la API para ponerlas en conexión. Todo ello se traduce en independencia y bajo acoplamiento entre componente y software.

**REST**

Representational State Transfer: Es un estilo de arquitectura de software que permite realizar una comunicación cliente-servidor; se trata de una interfaz que conecta sistemas o aplicaciones diferentes basándose en el protocolo HTTP. Los mensajes que se envían o reciben pueden estar en formatos como XML o JSON. JSON es comúnmente utilizado por defecto. No hay un estándar oficial de REST.

Alternativa a SOAP.

PINCIPIOS DE REST

* Interfaz uniforme: Rest está basado en recursos. Se opera sobre recursos y no sobre servicios o métodos.
* Para realizar las operaciones, emplea los métodos clásicos del protocolo HTTP:
  + GET: Obtiene recursos del servidor.
  + POST: Inserta un recurso nuevo en el servidor.
  + PUT: Actualiza un recurso completo en el servidor.
  + PATCH: Actualiza parcialmente un recurso en el servidor, solo algunos de sus campos.
  + DELETE: Borra un recurso en el servidor.
  + HEAD
  + CONNECT
  + OPTIONS
  + TRACE
* Cada recurso se maneja mediante su URL, a la que se puede aplicar las las operaciones CRUD (Create, Read, Update, Delete).
* Stateless: Rest no mantiene ningún tipo de estado entre peticiones distintas de forma que cada una de ellas es totalmente independiente de la siguiente. Entre dos llamadas cualesquiera, el servicio pierde todos los datos, no los “recuerda” en la siguiente petición. Es el cliente el que debe pasar el estado en cada llamada. Sin embargo, sí se puede usar una caché para reducir el tiempo de espera en consultas.
* Operaciones específicas: Cada operación tiene un único propósito sin existir multifuncionalidad dentro de la misma operación.
* Cliente-Servidor: El servidor es quein realiza el procesamiento de la API y expone los recursos a los Clientes. Cliente y Servidor son totalmente independientes y la comunicación entre ellos se realiza a través de la API.

VENTAJAS:

* Permite separar el front (cliente) del back (servidor). Se pueden desarrollar con tecnologías o framework diferentes.
* REST es independiente de la plataforma. Se puede montar en diferentes S.O.
* Aporta visibilidad y escalabilidad.

**SOAP**

Simple Object Acces Protocol: Arquitectura dividida por niveles empleada antiguamente para desarrollar un servicio que implicaba cierta complejidad en su utilización.

**RESTFUL**

Es la API que implementa REST como estilo de arquitectura. La aplicación que implementa los principios REST

**SPRING FRAMEWORK - SPRING BOOT – SPRING INITIALIZER**

Spring Framwork proporciona una infraestructura de trabajo, una serie de herramientas y funcionalidades que facilita el desarrollo de aplicaciones Java.

SpringBoot es una extensión de Spring Framework (una herramienta que simplifica la aplicación de Spring Framework) que proporciona unos modelos o plantillas, unas autoconfiguraciones para empezar a desarrollar una aplicación basada en Spring.

Proporciona una serie de “paquetes” de dependencias predefinidos en función del tipo de aplicación que se desea desarrollar, por lo que no habría que seleccionar las dependencias o librerías JAR una a una.

Spring Initializer es el asistente de Boot.