Desarrollo web en entorno servidor

UT1.2 – Programación web

6 – El patrón MVC – MVC en Spring / Spring Boot – Paso de datos a la vista – Selección de vista

IES Clara del Rey – Madrid

MVC

Model – View – Controller / Modelo – Vista – Controlador

Patrón de diseño o de sistemas, que se centra en la separación de responsabilidades o SoC (Separation of Concerns)

Indica como organizar los componentes de un sistema, en tres “capas”, cada una con una responsabilidad específica.

No es exclusivo de web, pero es el utilizado en la mayoría de frameworks de desarrollo web.

Su origen está en el desarrollo de aplicaciones Smalltalk (no web) en los años 70 / 80, pero se ha extendido a todo tipo de aplicaciones. Framewoks de desarrollo en servidor, aplicaciones móviles, aplicaciones web en cliente…

MVC

Propone la separación de los componentes del sistema en tres capas:

- Model (modelo): Responsable del dominio de la aplicación. Manipular datos, persistencia de datos, lógica de negocio y procesos, etc.
- View (vista): Responsable de la presentación de los datos al cliente. Genera la interfaz de la aplicación, que puede ser de distintos tipos dependiendo de la aplicación / del cliente que hace la petición.
- Controller (controlador): Responsable de atender las peticiones del usuario, y decidir qué hacer con ellas. Normalmente se remiten al modelo para su procesamiento. También es la capa responsable de preparar los datos para la vista, y pasárselos para que se genere la interfaz de usuario.

MVC – Modelo

En el Modelo de una aplicación MVC se puede encontrar:

- Clases que representan las entidades del dominio de la aplicación. Por ejemplo, en facturación serían las clases Factura, Cliente, etc.
- Lógica de negocio. Reglas, restricciones y acciones necesarias para que la aplicación realice su trabajo. Por ejemplo, crear una factura, enviar una factura al cliente, hacer un informe de facturación mensual, etc.
- Gestión del almacenamiento. Mecanismos que permiten que los objetos del dominio se almacenen y recuperen de la base de datos, o integración con servicios externos.
- Clases “auxiliares” específicas para la transferencia de datos, denominadas DTO (Data Transfer Objects).

MVC – Vista

Es la capa que contiene los componentes responsables de generar la interfaz de la aplicación. Suelen contener los elementos que permiten al usuario interactuar con la aplicación. En las aplicaciones web es la que contiene y genera el HTML de las páginas.

Puede considerarse que la vista es una representación de del estado del modelo en un momento concreto y en un contexto determinado.

Por ejemplo, la vista puede representar los datos de una factura en el contexto de validar la factura y marcarla como pagada.

Aunque se suele pensar en interacción con el usuario, no siempre es interactiva. Por ejemplo, en el sistema de facturación puede haber una vista que presente datos del modelo como un XML para que sea procesado por otro sistema.

MVC – Controlador

Los componentes de esta capa son los que hacen de intermediarios entre el usuario y el sistema.

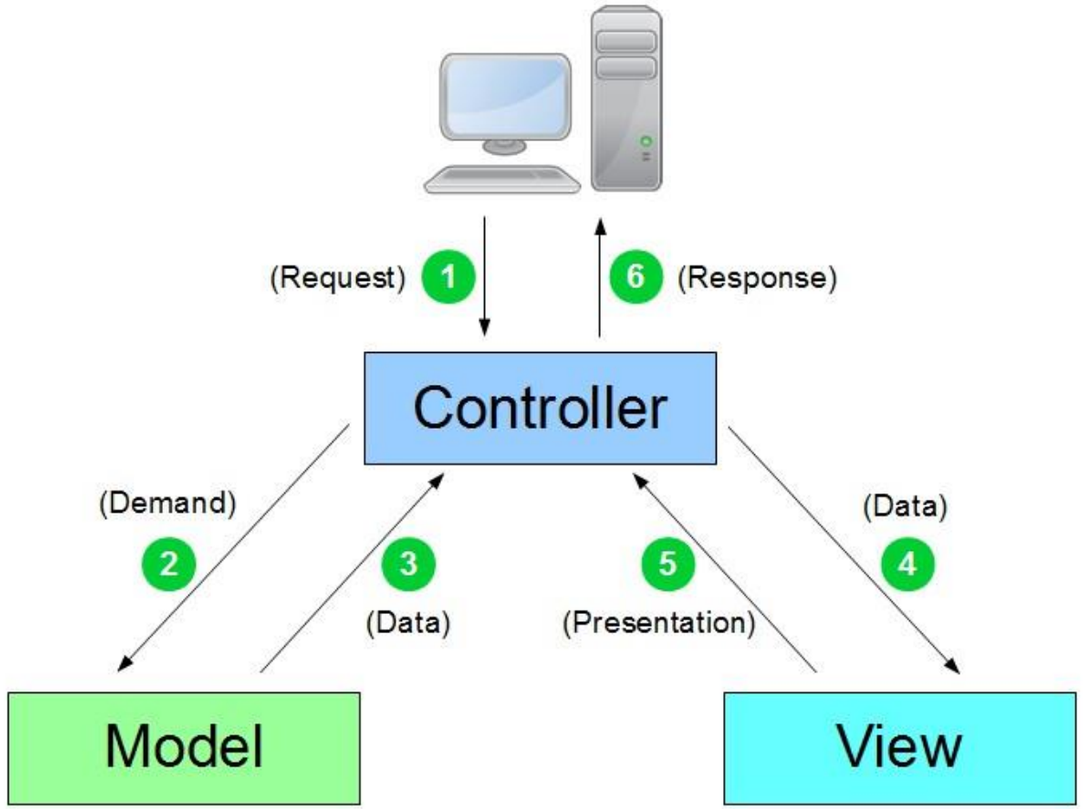
Capturan las acciones que el usuario realiza en la vista, y:

- Las analizan para determinar qué se debe hacer para atenderlas
- Solicitan al modelo que realice las acciones necesarias
- Si es necesario, recuperan datos del modelo
- Trasladan los datos a la vista, que será la responsable de presentarlos.

El controlador se podría considerar como un coordinador general del sistema, responsable de recibir las peticiones de del usuario, interactuar con el modelo, y devolver a la vista los datos necesarios para actualizarse.

MVC – Proceso habitual

1. El controlador recibe la acción / petición del usuario. Normalmente como respuesta a una acción en un componente del UI.
2. El controlador pide al modelo que realice acciones, que devuelven datos.
3. El controlador pasa los datos a la vista, que genera el UI (HTML en web)
4. La vista devuelve al controlador el UI generado, que contendrá componentes para acciones adicionales.
5. El controlador envía / muestra el UI al usuario.



MVC – Ampliación de la estructura

En aplicaciones complejas o grandes, de nivel empresarial, el concepto "modelo", tal y como lo define el patrón MVC, puede ser algo ambiguo.

Engloba demasiadas cosas: las clases los datos que el controlador pasa a la vista, procesos de negocio, persistencia de datos, validación, etc.

Se puede ampliar la estructura, dividiendo el modelo en varios bloques o capas, cada una con unas responsabilidades más concretas:

- Modelo: se reserva a los datos que el controlador pasa a la vista, o que el controlador recibe junto a algunas peticiones.
- Servicios: engloban la mayor parte de la lógica de la aplicación.
- Repositorios: se encargan de la persistencia de datos.

MVC en Spring – Proyecto con Spring Boot

Para poder desarrollar una aplicación Web MVC en Spring, se debe incluir la dependencia “Spring Web”.

Esta dependencia introduce en Maven o Gradle la siguiente dependencia:

org.springframework.boot.spring-boot-starter-web

Al incluir esta dependencia, en la aplicación se dispone de un servidor Apache Tomcat embebido, que se lanza automáticamente al ejecutar la aplicación.

Por defecto, el servidor escucha en el puerto 8080, pero puede cambiarse con la propiedad *server.port* en el fichero *application.properties*, que se encuentra en la carpeta “resources” del proyecto.

MVC en Spring

Para implementar MVC en Spring se usan elementos como:

- En controladores:
  - Anotaciones @Controller (web/HTML) o @RestController (API) para identificar los controladores, que serán beans.
  - Anotaciones @RequestMapping, o las más especializadas @GetMapping, @PostMapping, etc., para asociar métodos de controlador a las URL de las peticiones de los clientes.
  - Anotaciones como @PathParam o @RequestBody, entre otras, para extraer datos de la petición.

MVC en Spring

Para implementar MVC en Spring se usan elementos como (cont.):

- En servicios:
  - Anotación @Service, que amplía @Component, para identificar los beans con la lógica de negocio.
- En repositorios:
  - Anotación @Repository (Spring Data), en beans de acceso a datos
- En vista:
  - Motores de plantillas como Thymeleaf o JSP

MVC en Spring

En el modelo (los datos):

- Clases POJO, records e interfaces para representar los datos.
- Objetos de las clases Model, ModelMap o ModelAndView, entre otras, para transferir los datos del controlador a la vista.

Spring MVC – **DispatcherServlet**

El DispatcherServlet es el componente central para coordinar el trabajo entre el modelo (servicios), la vista y el controlador. Sigue estos pasos:

1. Intercepta todas las solicitudes entrantes
2. Analiza la solicitud y determina el método de controlador apropiado, a partir de los mappings definidos en los controladores.
3. Llama al método del controlador, pasando datos si es necesario. El controlador obtiene datos del modelo, y determina la vista que debe procesar estos datos.
4. Recoge los datos del controlador, y los pasa a la vista que este indique.
5. Recoge el resultado de procesar la vista (y datos), y lo envía al cliente.

Spring MVC – **Mappings**

El DispatcherServlet, para identificar el controlador y método que deben procesar una petición de un cliente, utiliza los mappings.

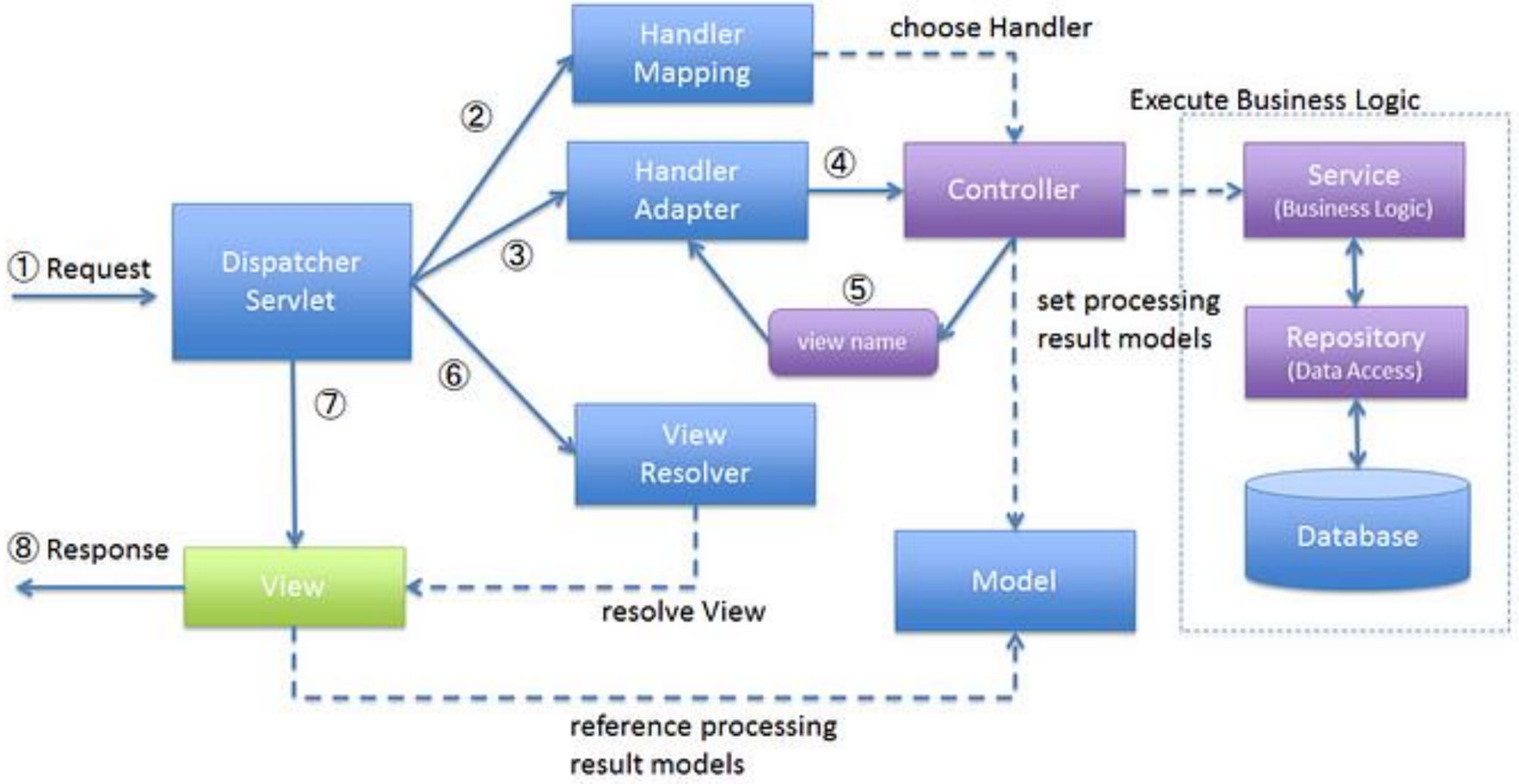
Un mapping es una asociación entre la combinación de URL, método HTTP, y otros criterios, y el método de controlador que debe procesarla.

La configuración de mapeo se define con anotaciones en los métodos del controlador:

- @RequestMapping: La forma más flexible de definir mapeos.
- “Versiones” específicas de @RequestMapping: @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping. Cada una para un método HTTP específico.

Más información: https://www.baeldung.com/spring-requestmapping

Spring MVC – **Procesado de peticiones**



Fuente: medium.com

Spring MVC – **Paso de datos a la vista**

El controlador, una vez procesada la petición, puede pasar datos a la vista de muchas formas diferentes (más de diez), pero las más habituales son:

- Utilizando "Model". Permite agregar atributos que estarán disponibles en la vista.
- Utilizando Map. Spring trata el objeto Map como Model.
- Utilizando "ModelMap". Map que también tiene los métodos de Model.
- Utilizando ModelAndView. Combina datos y vista. Permite devolver en un solo objeto la vista y la información necesaria para generar el resultado.
- Utilizando un objeto RedirectAttributes (para redirecciones). Junto a "flash attributes" permiten pasar datos entre métodos de controlador.

Spring MVC – **Paso de datos a la vista**

Otras formas menos habituales o más avanzadas y de más bajo nivel son:

- Devolviendo directamente un objeto. Esto sólo para servicios REST, en los que la vista no usa una plantilla, sino que es XML o JSON.
- Usando ResponseEntity para añadir metadatos HTTP en la respuesta. Más habitual en servicios REST.
- Utilizando anotaciones como @SessionAttributes, @ModelAttribute, @RequestScope, @SessionScope, @ApplicationScope.
- Usando BindingResult (para validaciones)

Y algunas más…

Spring MVC – **Paso de datos – Model**

Los datos se pasan a la vista en forma similar a un diccionario, muy similar a una colección "Map" de Java.

La interfaz "Model" define los métodos necesarios para añadir y consultar información del modelo, de los datos que el controlador pasa a la vista.

Los datos se añaden en la forma "clave-valor". Algunos métodos:

- Añadir: addAttribute, addAllAttributes, mergeAttributes.
- Consultar: containsAttribute, getAttribute, asMap.
- No tiene métodos para retirar atributos del modelo.

Para más información:

Referencia de Model en docs.spring.io

Spring MVC – **Paso de datos – Map**

Para determinar los pares clave-valor, se puede usar directamente un Map<String, Object>.

Spring interpreta cada Map.Entry<String, Object> como un atributo del modelo, al que accederá por su clave.

Permite manejar el modelo como si fuera un diccionario, con métodos como put, remove, isEmpty, containsKey, keySet, values, etc.

En este caso sí que se pueden eliminar atributos del modelo.

Spring MVC – **Paso de datos – ModelMap**

Combinación de Map y Model.

Hereda de LinkedHashMap<String, Object>. Tiene todos los métodos de Map<String. Object>, y además tiene los mismos métodos que Model.

Como Map, permite manejar el modelo como si fuera un diccionario, con métodos como put, remove, isEmpty, containsKey, keySet, values, etc.

Se pueden eliminar atributos del modelo, y esta es quizá la diferencia fundamental con Model. Esto hace que ModelMap sea útil en escenarios más complejos, con una lógica más elaborada.

Para más información sobre métodos:

Referencia de ModelMap en docs.spring.io

Spring MVC – **Paso de datos – ModelAndView**

Es una combinación de modelo, junto a la información de la vista.

Los métodos de controlador pueden devolver distintos valores para referenciar la vista, y un objeto de este tipo es uno de ellos.

Para modificar el modelo, se puede usar el método getModel, que devuelve un Map<String, Object> en el que se pueden añadir o eliminar entradas, que son a los atributos del modelo.

Tiene métodos y atributos asociados a la vista, que permiten personalizar más la vista que se tiene que generar.

Para más información sobre ModelAndView:

Referencia de ModelAndView en docs.spring.io

Spring MVC – **Paso de datos – ModelAndView**

Es una combinación de modelo, junto a la información de la vista.

Un método de controlador puede devolver distintos valores para indicar la vista, y un objeto de este tipo es uno de ellos. Tiene métodos y atributos asociados a la vista, que permiten especificarla de distintas formas.

Para modificar el modelo, se puede usar el método getModel, que devuelve un Map<String, Object> en el que se pueden añadir o eliminar entradas, que son a los atributos del modelo.

Útil para los métodos que no reciben modelo, pero deben generarlo. Ejemplo: listados, detalles de elemento, borrado de elemento...

Para más información sobre ModelAndView:

Referencia de ModelAndView en docs.spring.io

Spring MVC – @ModelAttribute

Esta anotación permite crear atributos en el modelo de una vista de forma más o menos automática. Se puede usar para crear ciertos atributos en los modelos de todos los métodos de un controlador:

```
@ModelAttribute
private <tipo> crearParteDelModelo(){
        return <variable del tipo indicado>;
}

@ModelAttribute
private void crearOtraParteDelModelo(Model model){
        model.addAttribute("atributo", <valor del atributo>);
}
```

El modelo tendrá un atributo por cada uno de los métodos. Se puede personalizar el nombre del atributo con @ModelAttribute(name="...")

Spring MVC – @ModelAttribute

También se puede usar para que un parámetro de un método del controlador se añada automáticamente al modelo de la vista:

```
@GetMapping("{id}/{nombre}")
public ModelAndView metodoDeControlador(
                @ModelAttribute @PathVariable int id,
                @ModelAttribute @PathVariable String nombre) {
        ModelAndView model = new ModelAndView();
        model.addObject("titulo", "Prueba de @ModelAttribute");
        model.setViewName("test");
        return model;
}
```

El modelo tendrá un atributo por cada uno de los parámetros. También se puede personalizar el nombre con @ModelAttribute(name="...")

Spring MVC – **Selección de la vista**

Un método de controlador puede devolver la vista de diferentes formas:

- Con String. Devuelve el nombre de la vista. Spring buscará una vista que se llame igual, pero con la extensión configurada. Por defecto .html. Las vistas pueden estar organizadas en subcarpetas, y en ese caso, se incluye las subcarpetas (ej. return "admin/user/detail").
- Con ModelAndView. Devuelve una combinación de modelo y vista. La vista se puede especificar como un string (equivalente a devolver String), pero permite escenarios más avanzados, como generar vistas dinámicamente, a partir, por ejemplo, de plantillas almacenadas en una base de datos.

Spring MVC – **Selección de la vista**

Se puede devolver la vista de diferentes formas (continuación):

- Con void. Spring intentará obtener la vista a partir de la URL de la petición.
- Con RedirectView / String "redirect: otra-url". Redirige a otra URL. Los datos recibidos no se reenvían.
- Con String "forward: otra-url". Reenvía la solicitud a otra URL, incluidos los datos que se recibieron.
- Con ResponseEntity o devovler directamente un objeto. Para servicios REST.

Aún hay alguna otra forma de devolver la vista, además de las indicadas.