
Curso Spring Boot

Temario

- Aplicaciones Web MVC: Controlador
  - Introducción a Spring MVC
  - Creación de controladores
  - Gestión del controlador de entrada
  - Procesado de peticiones (Formularios)
- Aplicaciones Web MVC: Vista
  - Responsabilidades de la vista
  - Sintaxis JSP. Scope. Objetos predefinidos
  - Introducción a JSTL
  - Creación de plantillas con Thymeleaf
  - Atributos del modelo
  - Representación de errores
  - Internacionalización (i18n)
  - Introducción a AJAX

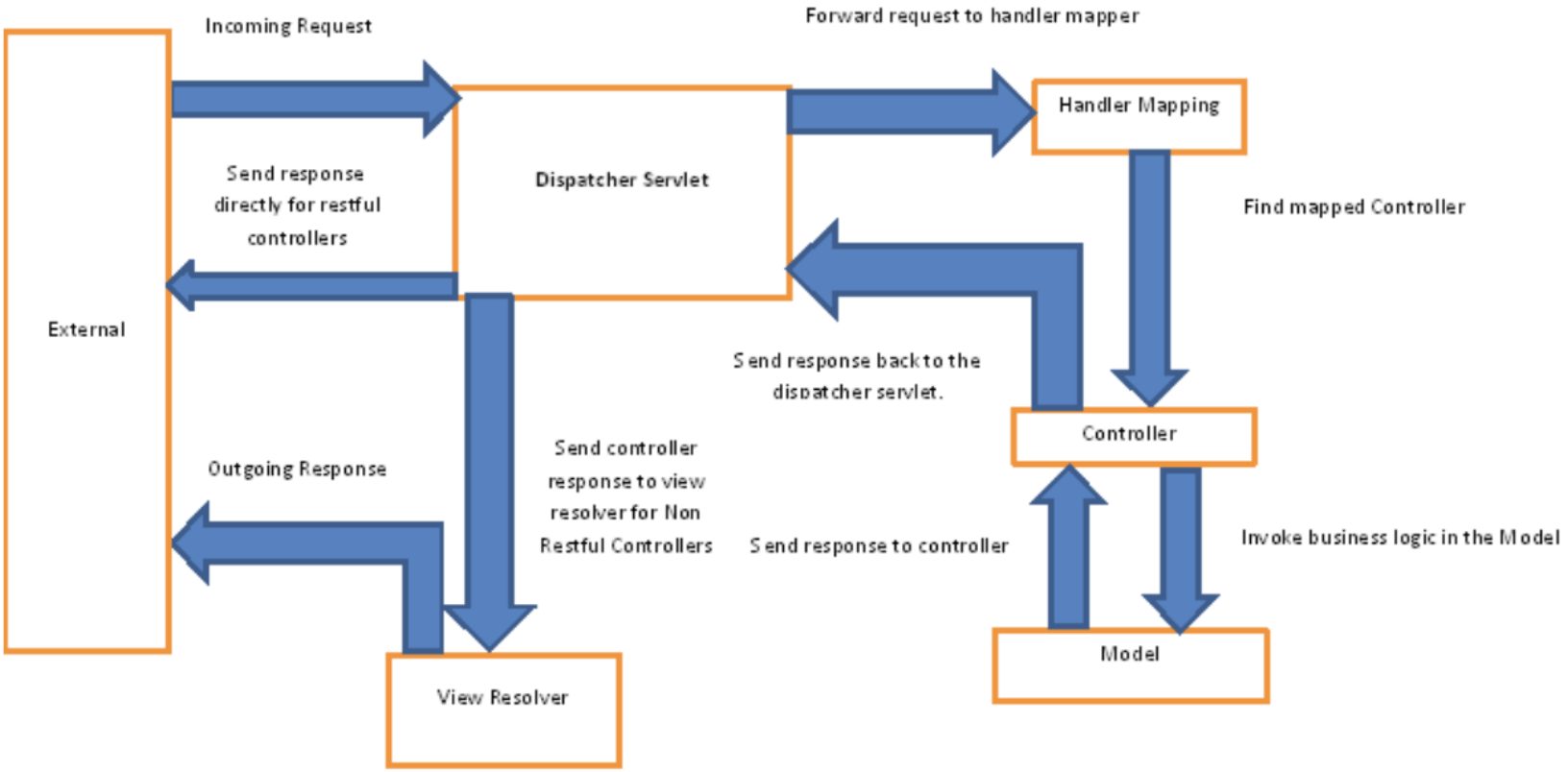
Spring MVC

- ¿Qué es?
  - En líneas generales, MVC es una propuesta de arquitectura del software utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, lo que ofrece beneficios diversos.
  - 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*, en inglés.

Spring MVC

- ¿Qué es?
  - **Modelo:** este componente se encarga de manipular, gestionar y actualizar los datos. Si se utiliza una base de datos aquí es donde se realizan las consultas, búsquedas, filtros y actualizaciones.
  - **Vista:** este componente se encarga de mostrarle al usuario final las pantallas, ventanas, páginas y formularios; el resultado de una solicitud. Desde la perspectiva del programador este componente es el que se encarga del *frontend*; la programación de la interfaz de usuario si se trata de un aplicación de escritorio, o bien, la visualización de las páginas web (CSS, HTML, HTML5 y Javascript).
  - **Controlador:** este componente se encarga de gestionar las instrucciones que se reciben, atenderlas y procesarlas. Por medio de él se comunican el modelo y la vista: solicitando los datos necesarios; manipulándolos para obtener los resultados; y entregándolos a la vista para que pueda mostrarlos.

Spring MVC


Incoming Request
Forward request to handler mapper
Handler Mapping
Send response directly for restful controllers
Dispatcher Servlet
Find mapped Controller
External
Send response back to the dispatcher servlet.
Controller
Outgoing Response
Send controller response to view resolver for Non Restful Controllers
Send response to controller
Invoke business logic in the Model
Model
View Resolver

Spring MVC Controlador

- @Controller, @RestController
  - La capa del controlador es la de más alto nivel y se encarga de exponer los servicios de la aplicación
  - En esta capa se debe implementar únicamente las tareas relacionadas con el formato y gestión de los datos de entrada-salida hacia-desde nuestra aplicación, quedando relegada la implementación de negocio a la capa de servicios
  - Esta capa se encargará además de la validación de la entrada así como de negociar con el cliente los datos y tipo de los mismos
  - @RequestMapping(method="GET", path="api/productos")
  - @GetMapping: Servicio de solo lectura de datos. No se deberían modificar ninguno de los datos en una llamada GET
  - @PostMapping: Servicio encargado de añadir un nuevo recurso
  - @PutMapping: Servicio encargado de actualizar un recurso dado
  - @DeleteMapping: Servicio encargado de eliminar un recurso dado

Spring MVC Controlador

- Errores HTTP
  - 400 Bad Request:
    - Cuando el cliente envía una petición que el servidor no puede comprender, aparece el error 400. Suele sucede cuando los datos enviados por el navegador no cumplen con las normas del protocolo HTTP.
  - 401 Unauthorization
    - Si el cliente ha solicitado una página web que se encuentra protegida por una contraseña, el servidor devuelve el error 401
  - 403 Forbidden
    - Este tipo de error se presenta cuando el servidor comprende la solicitud de cliente, pero por alguna razón no puede completarla
  - 404 Not found
    - Este código indica que el servidor no ha encontrado nada en la ubicación especificada por el cliente, porque la URL no se ha ingresado correctamente
  - 500 Internal Server Error
    - El código 500 es uno de los más conocidos cuando se trata de errores por parte del servidor. Se emplea cada vez que el servidor se encuentra con algún tipo de fallo que no le permite completar la solicitud del cliente.

Spring MVC Controlador

- Validaciones
  - @NotNull
  - @NotBlank
  - @Email
  - @Size, @Min, @Max
  - @Past, @Future
- Control de Errores (@ControllerAdvice)

```
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(MethodArgumentNotValidException.class)
public Map<String, String> handleValidationExceptions(
  MethodArgumentNotValidException ex) {
    Map<String, String> errors = new HashMap<>();
    ex.getBindingResult().getAllErrors().forEach((error) -> {
        String fieldName = ((FieldError) error).getField();
        String errorMessage = error.getDefaultMessage();
        errors.put(fieldName, errorMessage);
    });
    return errors;
}
```

Spring MVC Controlador

- Creación de una validación
  - Crear la clase que implementa la lógica de validación (IpAddressValidator)

```
class IpAddressValidator implements ConstraintValidator<IpAddress, String> {

  @Override
  public boolean isValid(String value, ConstraintValidatorContext context) {
```

  - Crear la anotación (@IpAddress)

```
@Target({ FIELD })
@Retention(RUNTIME)
@Constraint(validatedBy = IpAddressValidator.class)
@Documented
public @interface IpAddress {
```

Spring REST

- ¿Qué es?
  - REST es un acrónimo de Representational State Transfer o transferencia de estado representacional, le agrega una capa muy delgada de complejidad y abstracción a HTTP. Mientras que HTTP es transferencia de archivos, REST se basa en la transferencia de recursos.
  - Una API RESTful es una API diseñada con los conceptos de REST:
    - Recurso: todo dentro de una API RESTful debe ser un recurso.
    - URI: los recursos en REST siempre se manipulan a partir de la URI, identificadores universales de recursos.
    - Acción: todas las peticiones a tu API RESTful deben estar asociadas a uno de los verbos de HTTP

Spring REST

- Características
  - Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
  - Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
  - Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.
  - Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
  - Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
  - Uso de hipermedios: hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Spring REST

- Ventajas
  - **Separación entre el cliente y el servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
  - **Visibilidad, fiabilidad y escalabilidad**. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
  - **La API REST siempre es independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Spring REST

- Buenas Prácticas
  - Utililizar JSON como formato de datos
  - Usar nombres para los recursos. Las acciones serán definidas por los verbos HTTP (PUT, POST, DELETE, GET)
  - Utilizar plurales para los nombres de recursos siempre que exista una colección de los mismos
  - Mantener la jerarquía de recursos en la definición del PATH (“/users/{id}/tasks”)
  - Gestión de errores SIEMPRE transformando los mismos en códigos de error HTTP (4XX, 5XX, etc)
  - Utilizar filtros, ordenación, paginación y selección de campos en forma de Query Params (“/users?limit=5&page=3&sort=name_ASC&fields=id,name”)
  - Versionado. Diferentes formatos (“/api/2.1.0/”, “/api/v2/”. “/api/2021-10-01/”)
  - Documentación. Swagger, RESTDocs
  - Utilización de SSL/TLS

Spring REST

- HATEOAS
  - Es la abreviación de *Hypermedia as the Engine of Application State* (hipermedia como motor del estado de la aplicación). Es una restricción de la arquitectura de la aplicación **REST** que lo distingue de la mayoría de otras arquitecturas.
  - Cuando el servidor nos devuelva la representación de un recurso parte de la información devuelta serán identificadores únicos en forma de hipervínculos a otros recursos asociados.
  - La restricción HATEOAS desacopla cliente y el servidor de una manera que permite la funcionalidad del servidor para evolucionar de forma independiente.
- Spring HATEOAS
  - RepresentationModel, Link, and WebMvcLinkBuilder

Spring REST

- API First
  - **API First**: Es un enfoque para el desarrollo de software donde todo gira en torno a la idea de que las APIs son la primera y la más importante manera de acceder e interactuar con el producto principal. Enfocadas en aplicaciones cliente, independientes de la implementación, implican contratos
  - **API Documentation**: Manual de referencia para el API dirigido al usuario. Nos indica cómo utilizar el API
  - **Specification**: Define un estándar independiente del lenguaje para la descripción de APIs que permite tanto a humanos como a máquinas descubrir y entender las capacidades de un servicio sin requerir acceso al código fuente (Open API Specification)
  - **API Definition**: Define cómo está organizada y cómo funciona el API. Está dirigida al consumo por máquinas. Sirve como base para las herramientas automatizadas que generan documentación partiendo del código fuente del API, o código fuente partiendo de la documentación
  - **Swagger**: Conjunto de herramientas opensource para el trabajo con APIs. Permite definir, diseñar, documentar, probar y escalar APIs siguiendo los estándares anteriormente definidos