Tabla de Contenidos

- Spring MVC y Spring Boot
  - Spring MVC y Pom.xml
- PersonaController y su funcionamiento
- Otros Artículos relacionados

Acabamos de ver como usar Java equals y hashcode

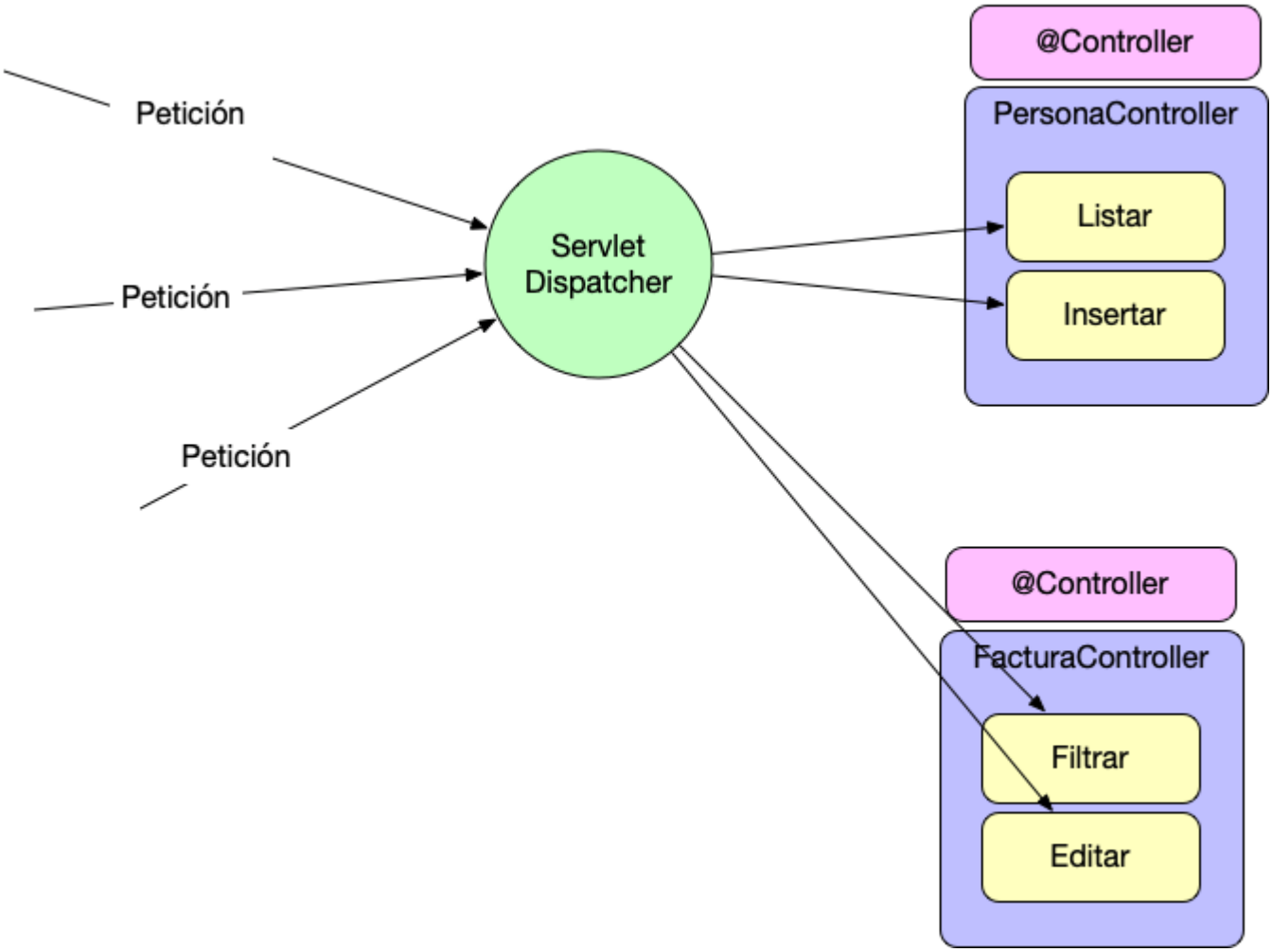
Spring MVC es quizás el framework Web más utilizado en el mundo Java y nos permite crear aplicaciones sobre modelo MVC que generen páginas HTML sencillas en las cuales nosotros podamos cargar los contenidos que necesitemos de forma sencilla pudiendo integrarse con otras tecnologías como jQuery , React o Vue a la hora de generar aplicaciones modernas y flexibles.

Spring MVC y Spring Boot

Para poder hoy en día desarrollar una aplicación sobre Spring MVC es suficiente con descargarnos un proyecto de Spring Boot que contenga dos de las dependencias más clásicas como Starters

- Spring Web
- Thymeleaf

Al tratarse de un framework MVC trabaja sobre todo con el concepto de Controlador Frontal (FrontController) que es el encargado de soportar todas las peticiones Web y redirigirlas a los componentes que sean necesarios . En este caso el controlador de Spring se denomina ServletDispatcher y viene configurado por defecto por Spring Boot.



Como podemos ver existe un único Servlet que recibe todas las peticiones (GET,POST etc) y según la información que recibe delega en el m´étodo adecuado de cada uno de los controladores. Vamos a ver un ejemplo inicial de como realizar estas operaciones . Lo primero que tenemos es que construir un proyecto de Spring Boot

adecuado que ejecuta la petición y devolverá unos resultados u otros .Spring MVC se apoya en este concepto.Vamos a montar como punto de partida una aplicación con Spring MVC que realice estas operaciones tan sencillas (Lista y Formulario). Para ello lo primero que necesitaremos es definir unas dependencias a nivel de Maven (pom.xml) con las librerias de Spring necesarias.

Spring MVC y Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.5.4</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.arquitecturajava</groupId>
        <artifactId>springweb</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>springweb</name>
        <description>Ejemplo Spring Web</description>
        <properties>
                <java.version>11</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
thymeleaf</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
```

```
web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
<groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

Una vez configurado todo correctamente nos encontraremos en una situación en la que tenemos una aplicación de Spring Boot prácticamente vacía . Es momento de generar un objeto de negocio y generar una lista de Items con él para poner a funcionar nuestro ejemplo de HolaMundo.

```
package com.arquitecturajava.springweb;

public class Persona {
```

```
	private String nombre;
	private String apellidos;
	private  int edad;
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.edad = edad;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
}
```

Acabamos de construir la clase .Es momento de diseñar el controlador que se encargará de mostrarnos un listado de Personas en una página html diseñada con el motor de plantillas de thymeleaf.

```
package com.arquitecturajava.springweb;

import java.util.Arrays;
import java.util.List;

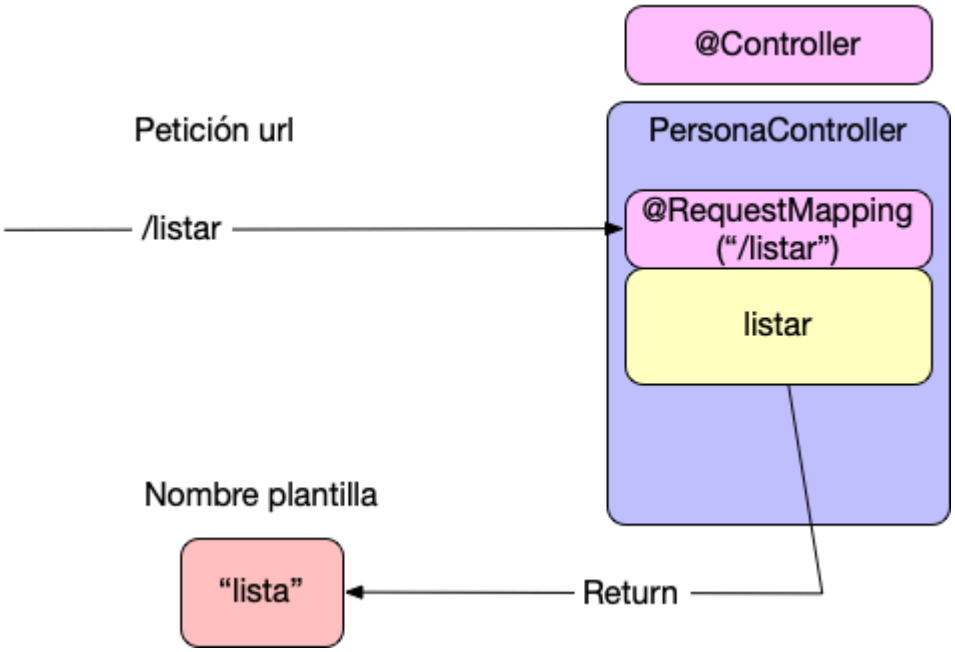
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;

@Controller
public class PersonaController {

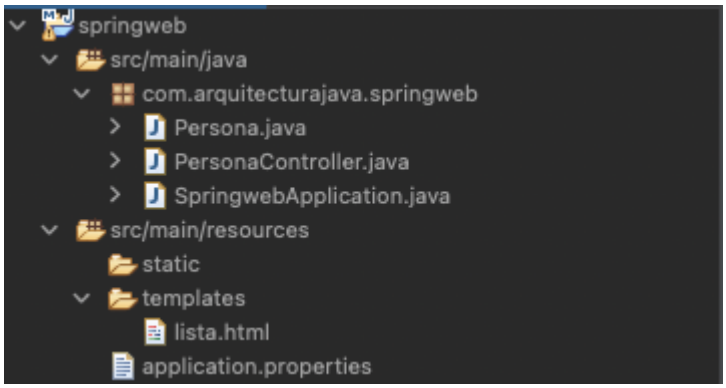
	@RequestMapping("/lista")
	public String lista(Model modelo) {
		Persona p1= new Persona("pedro","perez",20);
		Persona p2= new Persona("ana","gomez",30);
		Persona p3= new Persona("gema","alvarez",40);
		List<Persona> personas= Arrays.asList(p1,p2,p3);
		modelo.addAttribute("personas", personas);
		return "lista";
	}
}
```

PersonaController y su funcionamiento

En este caso el controlador que estamos utilizando se encarga de mapear una URL de respuesta con la anotación @RequestMapping esta anotación liga un método de un controlador de Spring con una URL que un cliente necesite . En este caso la url de /lista queda mapeada a la ejecución del método lista del Controlador Persona.



Una vez tenemos claro cómo funcionan los controladores que hemos definido es momento de ver el contenido de los ficheros html que se van a encontrar en la carpeta de templates del proyecto de Spring Boot.



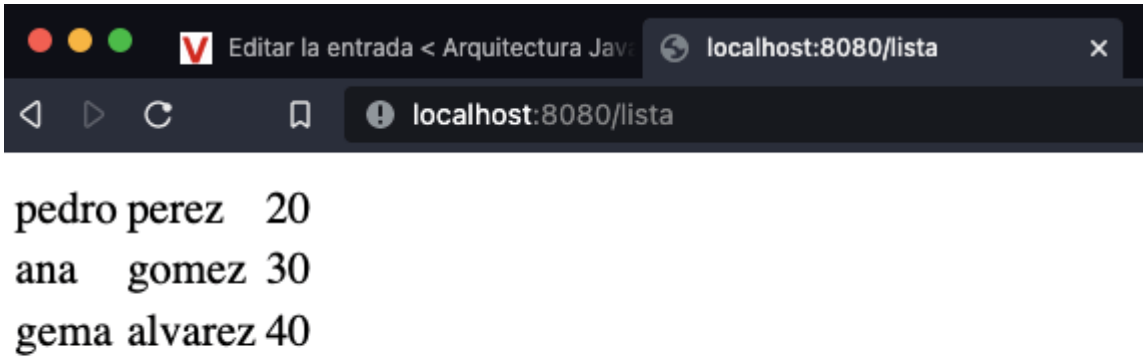
Nos queda por ver el código fuente de la plantilla que genera la lista:

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
<table>

<tr th:each="persona: ${personas}">
    <td th:text="${persona.nombre}" />
```

```
    <td th:text="${persona.apellidos}" />
    <td th:text="${persona.edad}" />
</tr>
</table>
</body>
</html>
```

Realizadas todas estas operaciones ya tenemos una aplicación de Spring MVC corriendo sobre Spring Boot de forma sencilla.



Otros Artículos relacionados

- ¿Qué es Spring Boot?
- Curso Spring REST
- Spring @Component , anotaciones y jerarquía
- Spring Boot JDBC y su configuración

  Acabamos de ver como usar Java equals y hashcode