EFA
MORATALAZ

2º CFGS Desarrollo de Aplicaciones Web

DESARROLLO WEB EN ENTORNO SERVIDOR

JESÚS SANTIAGO RICO

UT2 – SPRING MVC

EFA
MORATALAZ

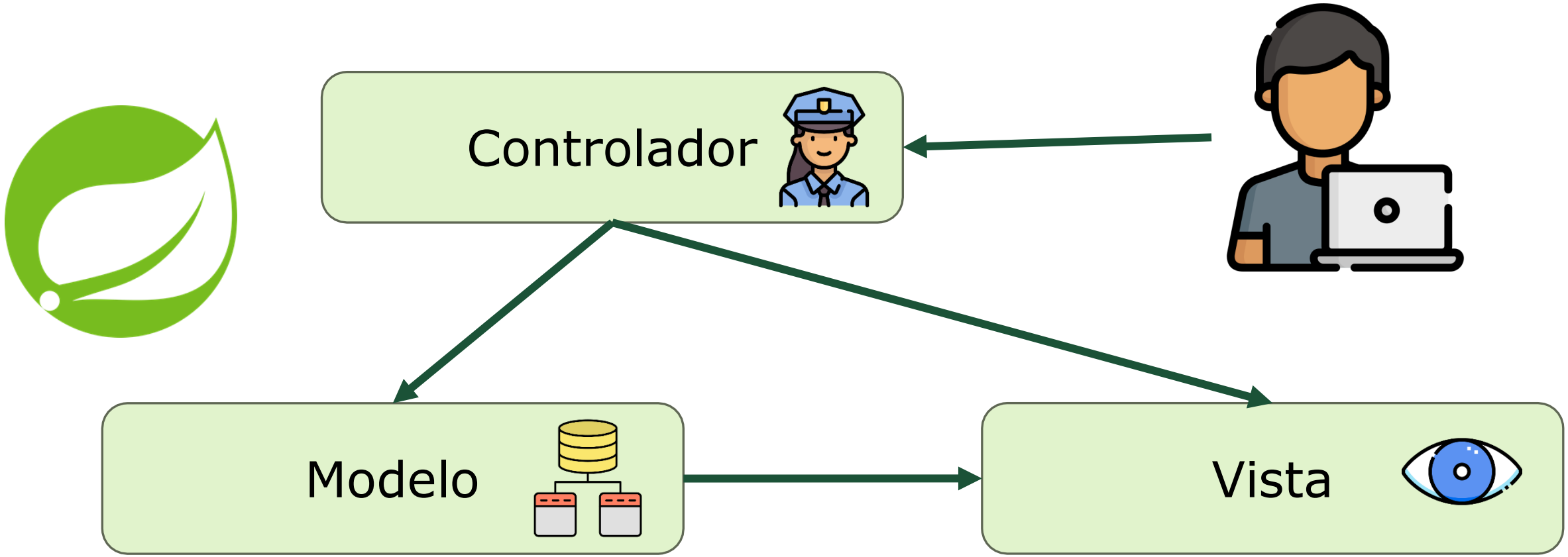
2º CFGS Desarrollo de Aplicaciones Web

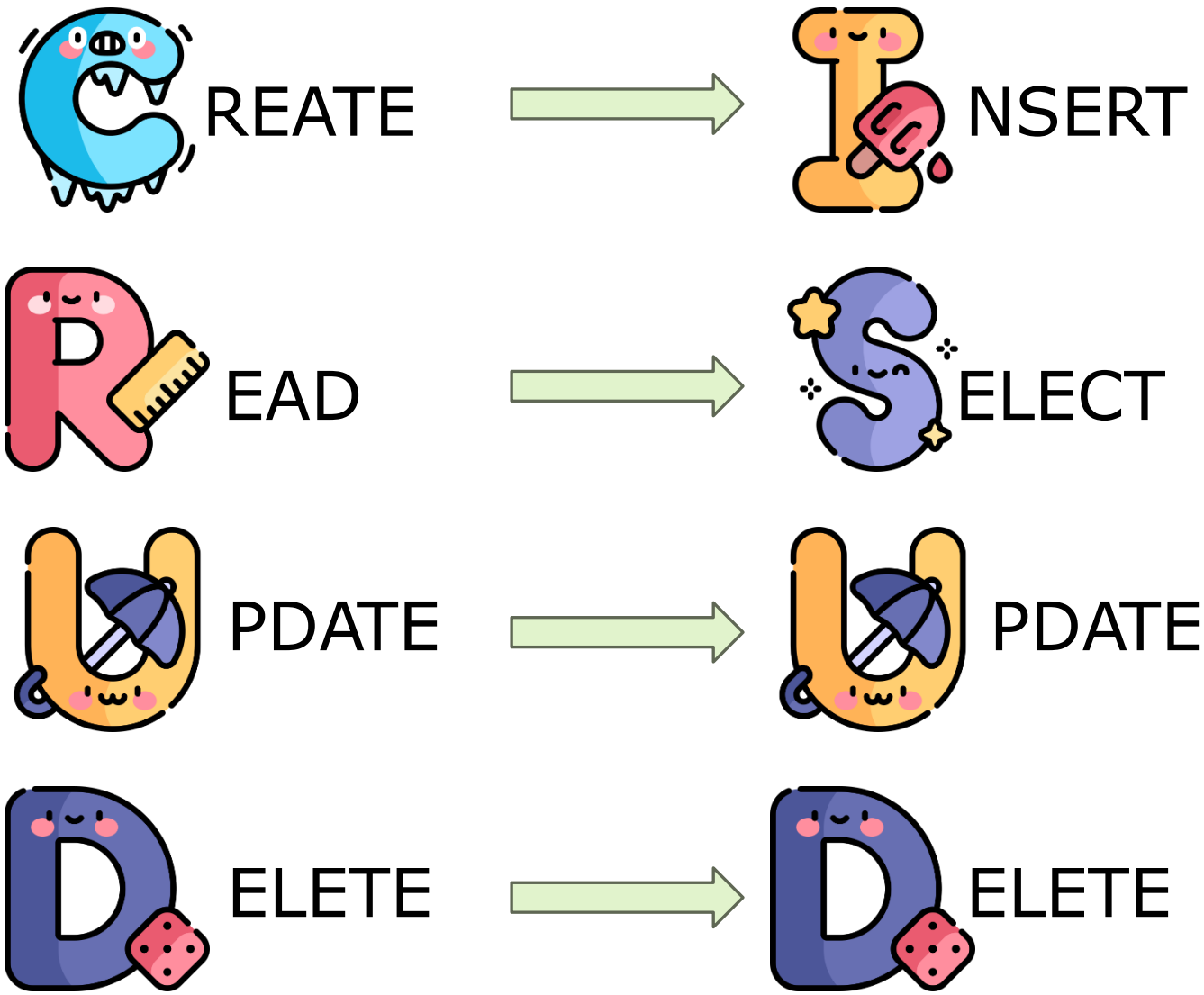
DESARROLLO WEB EN ENTORNO SERVIDOR

UT2 – SPRING MVC

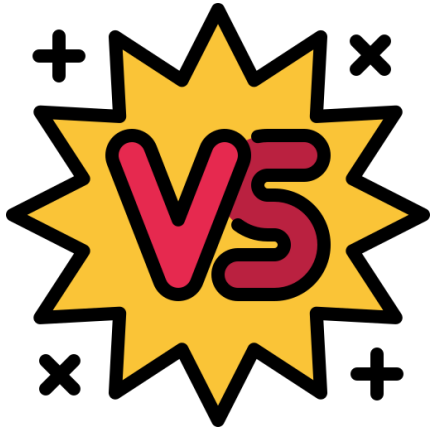
1. INTRODUCCIÓN
2. SPRING FRAMEWORK VS SPRING BOOT
3. CONTROLADORES Y VISTAS
4. COQUETEANDO CON THYMELEAF
5. ANOTACIÓN @MODELATTRIBUTE
6. ANOTACIÓN @REQUESTPARAM
7. ANOTACIÓN @PATHVARIABLE
8. USO REDIRECT Y FORWARD

INTRODUCCIÓN


Controlador
Modelo
Vista


REATE
NSERT
EAD
ELECT
PDATE
PDATE
ELETE
ELETE

SPRING FRAMEWORK VS SPRING BOOT


VS

Spring Framework

Spring es un marco de trabajo o Framework que nos permite desarrollar aplicaciones con el Lenguaje de Programación Java. Spring Framework cuenta con muchas características que hacen a los desarrolladores más productivos, como la inyección de dependencia y módulos listos como:

- Spring Test
- Spring ORM
- Spring JDBC
- Spring AOP
- Spring MVC
- Spring Security

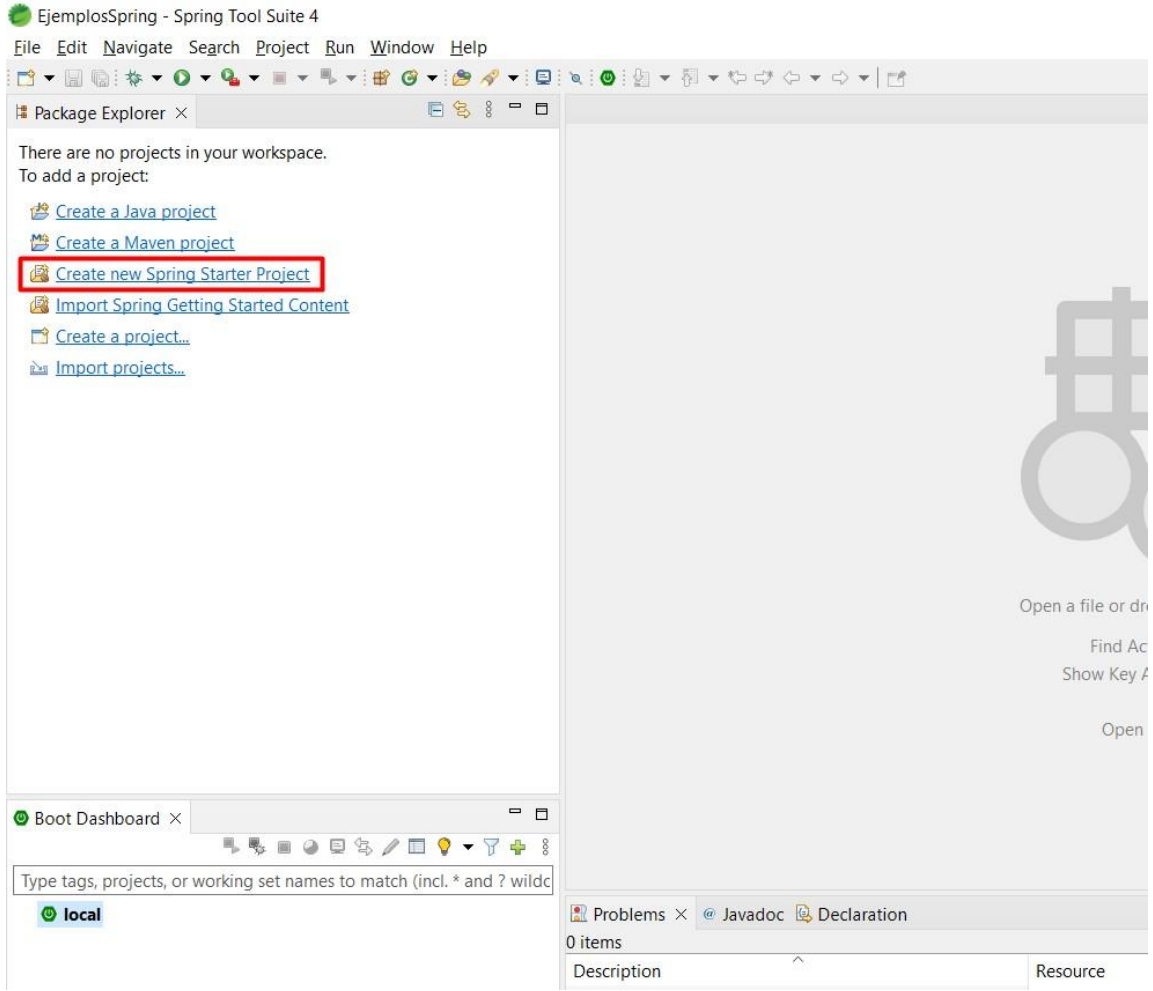
Spring boot

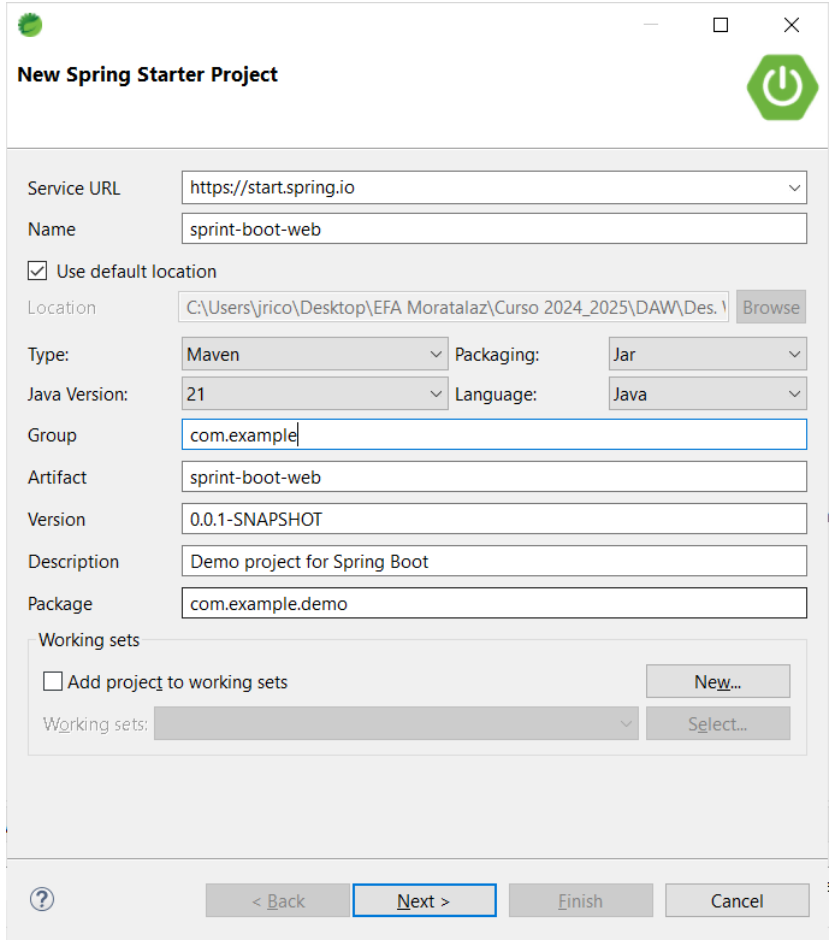
Spring Boot busca reducir la longitud del código y simplificar el desarrollo de aplicaciones web. Al aprovechar las anotaciones y la configuración repetitiva, Spring Boot reduce el tiempo que lleva desarrollar aplicaciones.

- Crea aplicaciones Spring independientes.
- Incrusta Tomcat, Jetty o Undertow directamente (no es necesario implementar archivos WAR).
- Configura automáticamente Spring y bibliotecas de terceros siempre que sea posible.
- Proporciona funciones listas para producción, como métricas, controles de estado y configuración externalizada.
- Absolutamente sin generación de código y sin requisitos para la configuración XML.

CREACIÓN Y ESTRUCTURA DE PROYECTO SPRING

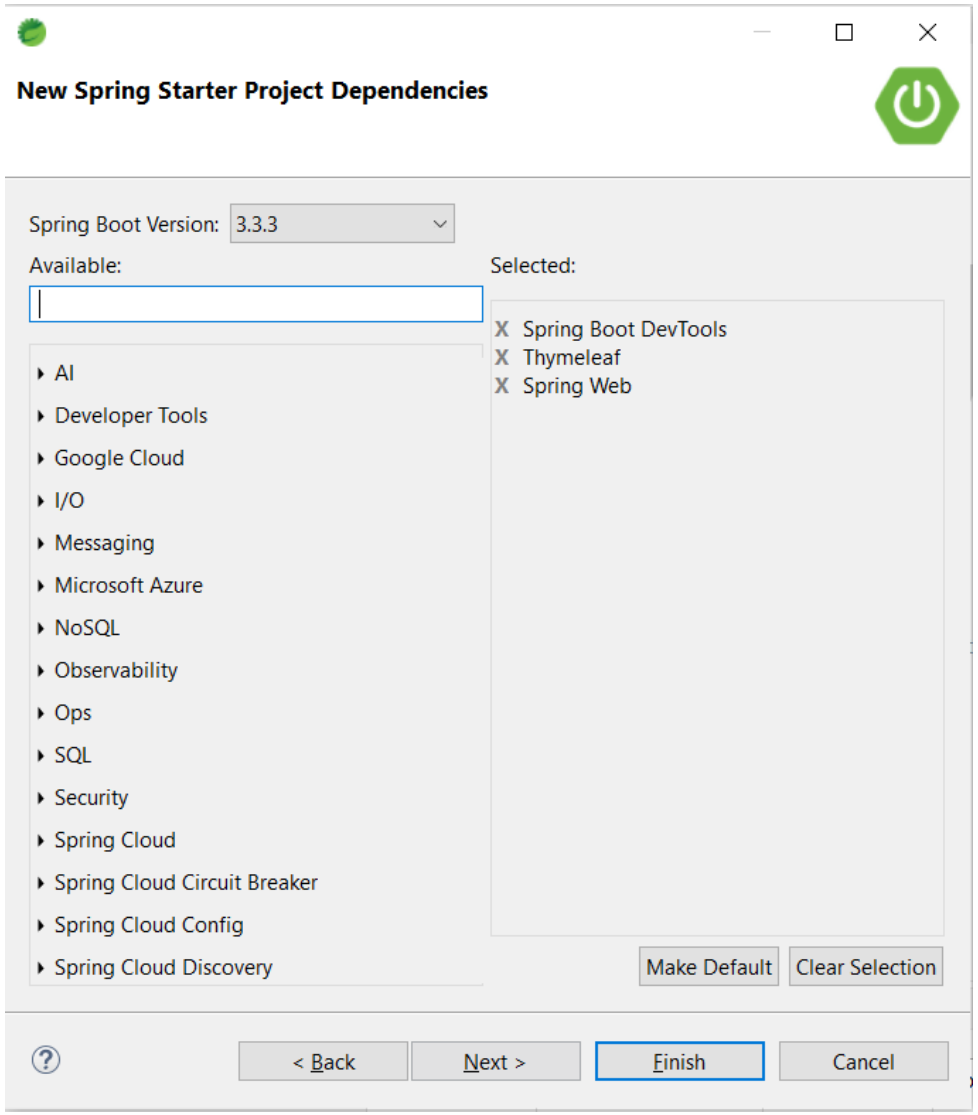
Para la creación del proyecto realizaremos los siguientes pasos:





La siguiente tabla indica el significado de cada campo en la creación del proyecto.

Número	Campo	Descripción
1	Name	Nombre del proyecto web
2	Type	Tipo de administrador de dependencias, para generar nuestro proyecto **(Maven y Gradle)**
3	Packaging	Tipo de empaquetado del proyecto **(jar y war)**. Se recomienda el uso de .jar si se usa spring-boot que usa plantillas thymeleaf y también api rest. War servidor externo como por ejemplo JBOSS o vista JSP.
4	Java Version	Indica la versión de java a usar (recomendada la instalada en nuestra máquina)
5	Group	Nombre para indicar en nombre de espacios o paquetes Maven.
6	Package	Nombre del paquete raíz de Java, en donde se crearán todas las clases de Spring (controladores, clases de modelos, entities, DAOS)

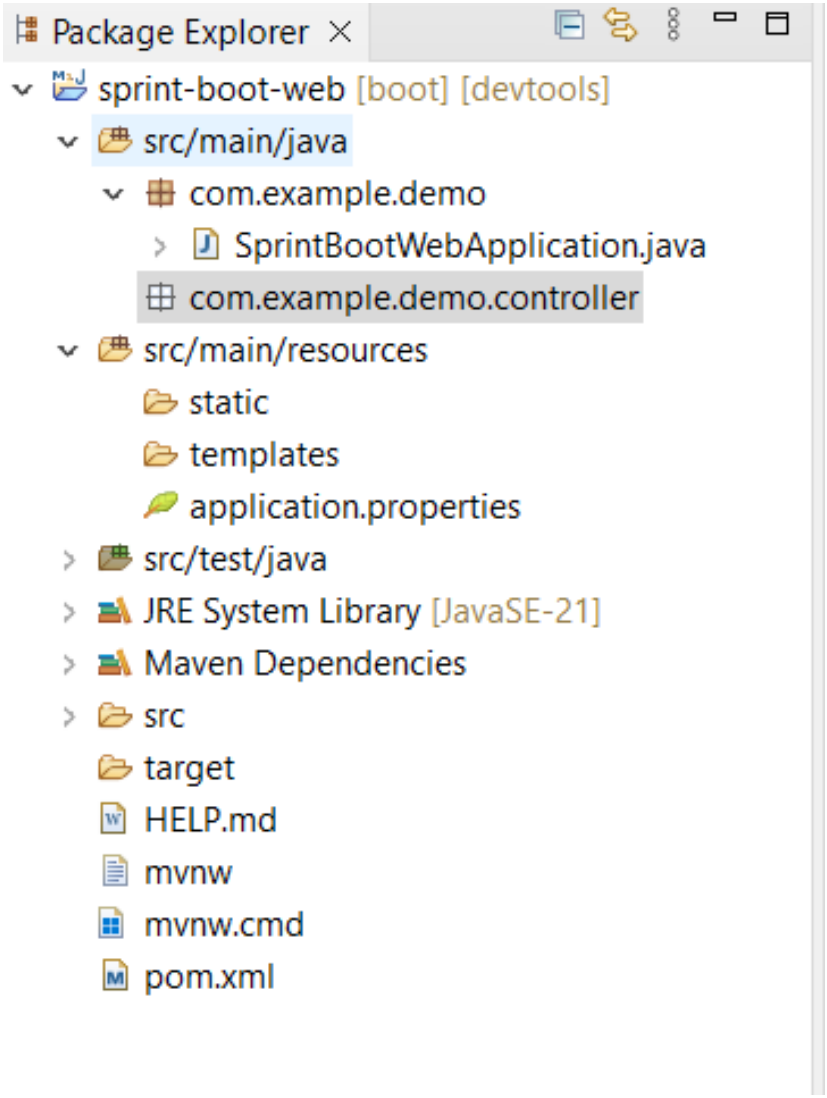


- **Spring boot Versión**, no debe seleccionarse una versión **M2 o SNAPSHOT** (versiones de desarrollo), debe usarse una versión estable.

- A continuación, se deben seleccionar las dependencias que usaremos en el proyecto:

  ✓ **Spring web**
  ✓ **Thymeleaf**
  ✓ **Spring Boot DevTools**

Estructura de proyecto



3. Creación y estructura proyecto Spring

sprint-boot-web/pom.xml

```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>3.3.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>sprint-boot-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>sprint-boot-web</name>
    <description>Demo project for Spring Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>21</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>
```

Estructura de proyecto

Número	Ruta	Descripción
1	Maven dependencias	Ruta en la se incluyen todas las librerías del proyecto y fueran seleccionadas en el wizard.
2	src/main/resources/application.properties	Fichero en el que se puede modificar la configuración por defecto de spring, así como añadir las conexiones de BBDD, drivers, usuarios, dialectos de conexión de BBDD.
3	src/main/resources/static	Se crea de forma automática cuando trabajamos con Thymeleaf. Directorio que incluya los recursos estáticos de la aplicación, CSS, JS, imágenes, etc.
4	src/main/resources/templates	Se crea de forma automática cuando trabajamos con Thymeleaf. Guarda las vistas de los controladores.
5	src/main/java	Contiene todos los fuentes java.
6	target	Ruta en la que se deja el proyecto una vez que este es publicado.

Estructura de proyecto

Cambiamos el puerto (por ejemplo al 9090 para evitar conflictos con otras aplicaciones)

En el fichero application.properties → server.port=9090

CONTROLADORES Y VISTAS

¿Qué es una anotación?

Es una forma de añadir metadatos al código fuente Java que están disponibles para la aplicación en tiempo de ejecución o de compilación. Muchas veces se usa como una alternativa a la tecnología XML.

@Override

¿Qué es un controlador?

Es una clase java que contiene uno o más métodos, los cuales se corresponden con acciones o peticiones que se hacen desde la parte cliente y representan a las diferentes páginas de la aplicación. Para marcar una clase como un controlador de spring usamos la anotación:

@Controller

Al usar esta anotación se importa automáticamente la siguiente ruta:

org.springframework.stereotype.Controller

Métodos del controlador

Los métodos del controlador para poder ser usados deben ser mapeados con una determinada URL, para ellos podemos usar alguna de las siguientes anotaciones:

@GetMapping()
@PostMapping()
@RequestMapping()

Las tres anotaciones tienen como mínimo el parámetro **value**, el cual debe empezar con el carácter **"/"** o **""***.

Métodos del controlador: GetMapping()

```java
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class IndexController {

    @GetMapping(value = "/")
    public String index() {
        return "index";
    }

    @GetMapping(value = "/mensaje")
    public String indexMensaje(Model modelo) {
        modelo.addAttribute("mensaje", "Estamos probando un mensaje desde la /mensaje");
        return "mensaje";
    }

    @GetMapping(value = "/mensajeCompleto")
    public String indexMensajeCompleto(Model modelo) {
        modelo.addAttribute("mensaje", "Estamos probando un mensaje desde /mensajeCompleto");
        return "mensajeCompleto";
    }

}
```

Métodos del controlador: PostMapping()

```java
package com.example.demo.controller;

import java.util.ArrayList;

@Controller
public class PostMController {

    private List<LibroDTO> listaLibros = new ArrayList<>();

    @GetMapping("/lista_libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", listaLibros);
        return "mostrarNuevoLibro";
    }

    // Mostrar el formulario para agregar un nuevo libro
    @GetMapping("/nuevo_libro")
    public String mostrarFormulario(Model model) {
        model.addAttribute("libro", new LibroDTO());
        return "nuevo_libro";
    }

    // Manejar la solicitud POST para guardar el libro
    @PostMapping("/guardar_libro")
    public String guardarLibro(@ModelAttribute LibroDTO libro, Model model) {
        listaLibros.add(libro);
        model.addAttribute("libros", listaLibros);
        return "mostrarNuevoLibro";
    }
}
```

Otra de las funciones del controlador es definir una ruta por defecto para un controlador, por lo tanto, para poder usar un método del controlador deberemos escribir previamente la ruta por defecto y posteriormente el método al que queremos invocar, para definir la ruta por defecto usamos la anotación:

@RequestMapping()

A continuación, se lista un ejemplo con una ruta por defecto.

```java
package com.example.demo.controller;

import org.springframework.stereotype.Controller;

@Controller
@RequestMapping("/info")
public class RequestMController {

    @GetMapping(value = "/mensaje")
    public String indexMensaje(Model modelo) {
        modelo.addAttribute("mensaje", "Estamos probando un mensaje desde info/mensaje");
        return "mensaje";
    }

    @GetMapping(value = "/mensajeCompleto")
    public String indexMensajeCompleto(Model modelo) {
        modelo.addAttribute("mensaje", "Estamos probando un mensaje desde info/mensajeCompleto");
        return "mensajeCompleto";
    }

}
```

Además de marcar una clase como controlador, configurar todos los métodos y definir una ruta por defecto, el controlador puede pasar datos a la vista y para ello podremos usar alguna de las siguientes opciones.

- **Interfaz Model (Por debajo se usa una Map)**
- **ModelMap**
- **Interfaz Map**
- **ModelAndView**

A continuación, se listan ejemplos de métodos con las distintas opciones para pasar datos a las vistas.

Interfaz model

```
@RequestMapping(value="/index1", method = RequestMethod.GET)
public String index1(Model modelo) {
    modelo.addAttribute("titulo", "Hola mundo en spring en función index1");
    return "index";
}
```

ModelMap

```
@GetMapping(value="/index2")
public String index2(ModelMap modelo) {
    modelo.addAttribute("titulo", "Hola mundo en spring en función index2");
    return "index";
}
```

Interfaz Map

```
@GetMapping(value="/index3")
    public String index3(Map<String, Object> modelo) {
    modelo.put("titulo", "Hola mundo en spring en función index3");
    return "index";
}
```

ModelAndView

```
@GetMapping({"/home", "/", "/inicial"})
public ModelAndView index4(ModelAndView modelo) {
    modelo.addObject("titulo", "Hola mundo en spring en función index3");
    modelo.setViewName("index");
    return modelo;
}
```

Transferencia de datos a la vista mediante un objeto

```
@GetMapping("/biblioteca/libro")
public String mostrarLibro(Model modelo) {
    LibroDTO libro = new LibroDTO("2024", "En agosto nos vemos", "Gabriel Garcia Marquez");
    modelo.addAttribute("titulo", "Ver libro");
    modelo.addAttribute("informacion", libro);
    return "mostrarLibro";
}
```

COQUETEANDO CON THYMELEAF

Adaptación de la vista

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title th:text="${titulo}"></title>
</head>
<body>
    <h1 th:text="${informacion}"></h1>
</body>
</html>
```

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title th:text="${titulo}"></title>
</head>
<body>
    <h1 th:text="${titulo}"></h1>
    <span th:text="${informacion.publicadoEn}"></span><br>
    <span th:text="${informacion.titulo}"></span><br>
    <span th:text="${informacion.autor}"></span><br>
</body>
</html>
```

Directiva IF

Directiva FOR

```
<head>
    <meta charset="UTF-8">
    <title>Lista de Libros</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <div class="container">
        <h1 class="mt-4">Lista de Libros</h1>

        <!-- Comprobamos si hay libros en la lista -->
        <div th:if="${libros.isEmpty()}">
            <p>No hay libros disponibles.</p>
        </div>

        <!-- Mostramos la tabla solo si hay libros -->
        <div th:if="${not libros.isEmpty()}">
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Autor</th>
                        <th>Titulo</th>
                        <th>Publicado en</th>
                    </tr>
                </thead>
                <tbody>
                    <!-- Iteramos sobre la lista de libros -->
                    <tr th:each="libro : ${libros}">
                        <td th:text="${libro.autor}">Autor del libro</td>
                        <td th:text="${libro.titulo}">Titulo del libro</td>
                        <td th:text="${libro.publicadoEn}">Publicado en</td>
                    </tr>
                </tbody>
            </table>
        </div>

        <!-- Enlace para agregar un nuevo producto -->
```

ANOTACIÓN @MODELATTRIBUTE

Se trata de otra forma de mandar datos a la vista. La diferencia de esta notación es que los datos que se setean en el modelo, son visibles desde cualquier método del controlador, siempre y cuando la vista pueda mostrarlos.

```
@ModelAttribute
public void titulo(Model model) {
    model.addAttribute("titulo", "Mostrar Libros");
}
```

También se utiliza para pasar objetos como parámetro de entrada de cualquier método:

```
@PostMapping("/guardar_libro")
public String guardarLibro(@ModelAttribute LibroDTO libro, Model model) {
    listaLibros.add(libro);
    model.addAttribute("libros", listaLibros);
    return "mostrarNuevoLibro";
}
```

ANOTACIÓN @REQUESTPARAM

¿Qué es un parámetro?

Un parámetro, es un valor que se pasa en una solicitud web mediante la URL y con el que se puede realizar diferentes tipos de acciones. La forma de indicar en spring que usaremos un parámetro es con la anotación:

@RequestParam()

Esta anotación se indica en los diferentes parámetros de los métodos de un controlador y tienen las opciones que se indican en la siguiente diapositiva.

Opciones @RequestParam

Opción	Descripción
name	Esta opción indica el nombre del parámetro que se pasa por la URL, pero este campo puede ser optativo, la condición para que esta opción sea optativa es que el parámetro que contenga la anotación se llame exactamente igual que el parámetro de la URL.
value	Alias del campo anterior, tiene el mismo comportamiento
defaultValue	Asigna un valor por defecto en caso de que no se informe el parámetro configurado.
required	Indica si el parámetro es necesario informarlo o no y los posibles valores son true o false.

localhost:9090/pruebaParametros?titulo=Don%20Quijote&autor=Cervantes

```java
@GetMapping("/pruebaParametros")
public String infoLibroParam(@RequestParam(name = "titulo") String titulo,
        @RequestParam(name = "autor", required = false, defaultValue = "Sin autor") String autor, Model model) {
    model.addAttribute("titulo", titulo);
    model.addAttribute("autor", autor);

    return "pruebaParametros";
```

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Prueba parametros</title>
</head>
<body>
    <h1>Prueba parametros</h1>
    <p>Titulo: <span th:text="${titulo}"></span></p>
    <p>Autor: <span th:text="${autor}"></span></p>
</body>
</html>
```

HttpServletRequest

```java
@GetMapping("/pruebaRecogerParametros")
public String infoLibroParamServlet(HttpServletRequest request, Model model) {
    String titulo = request.getParameter("titulo");
    String autor = request.getParameter("autor");

    model.addAttribute("resultado", "El titulo recuperado es: "+titulo+ " y el autor recuperado es: " +autor);
    model.addAttribute("titulo", titulo);
    model.addAttribute("autor", autor);

    return "pruebaParametrosServlet";
}
```

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Prueba parametros HttpServlet</title>
</head>
<body>
    <h1>Prueba parametros HttpServlet</h1>
    <p><span th:text="${resultado}"></span></p>
    <p>Titulo: <span th:text="${titulo}"></span></p>
    <p>Autor: <span th:text="${autor}"></span></p>
</body>
</html>
```

ANOTACIÓN @PATHVARIABLE

```java
@GetMapping(value = "/infoLibro/{nombre}")
public String infoLibroPathV(@PathVariable String nombre, Model modelo) {
    modelo.addAttribute("mensaje", "El nombre del libro es: " + nombre);
    return "pruebaParametrosPath";
}
```

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Prueba parametros PathVaribale</title>
</head>
<body>
    <h1>Prueba parametros PathVaribale</h1>
    <p>Nombre: <span th:text="${mensaje}"></span></p>
</body>
</html>
```