

Implementación de proyectos Spring MVC con Spring Boot

Spring Boot

- Nos permite crear rápidamente aplicaciones basadas en componentes de Spring, en este ejemplo será con Spring MVC, y posteriormente iremos añadiendo otros componentes como JPA, Hibernate, PostgreSQL, etc.
- Traen un servidor Tomcat embebido en la aplicación, por lo que pueden ser ejecutados sin necesidad de configurar uno externo
- Nuestras aplicaciones, a pesar de que son proyectos web, los ejecutaremos a partir del método **main** que nos crea Spring Boot, el cual instancia el servidor Tomcat embebido y levanta el aplicativo para que podamos accederlo mediante un navegador web.

Para poder crear proyectos con Spring Boot de manera rápida, existe una herramienta en línea que utilizaremos para generar nuestro proyecto Spring MVC con lo esencial. Dicha herramienta nos creará un proyecto tipo Maven que podremos importar en nuestro IDE para seguir trabajándolo.

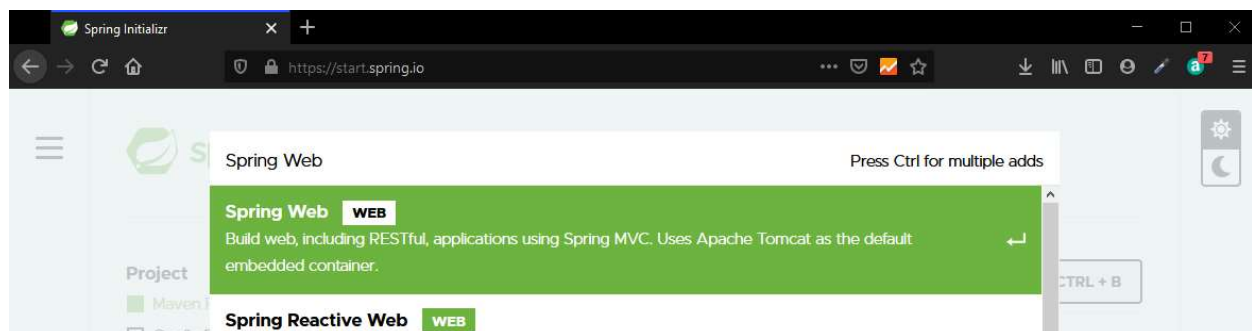
Pasos para crear nuestro proyecto Spring MVC utilizando Spring Boot

1. Ingresar a la URL <https://start.spring.io/>
2. Seleccionaremos las siguientes opciones (como en la imagen):
 - a. Project: **Maven Project**
 - b. Language: **Java**
 - c. Spring Boot: **2.2.6**
 - d. Project Metadata:
 - i. Group: **com.uca.capas**
 - ii. Artifact: **ejemplo**
 - iii. Name: **ejemplo**
 - iv. Description: **Proyecto Spring MVC con Spring Boot**
 - v. Package name: **com.uca.capas.ejemplo**
 - vi. Packaging: **Jar**
 - vii. Java: **8**



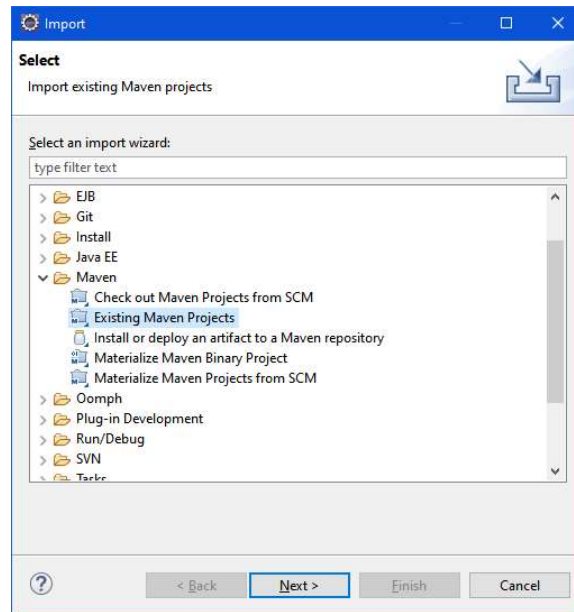
Project	Language
<input checked="" type="checkbox"/> Maven Project	<input checked="" type="checkbox"/> Java <input type="checkbox"/> Kotlin
<input type="checkbox"/> Gradle Project	<input type="checkbox"/> Groovy
Spring Boot	
<input type="checkbox"/> 2.3.0 M4 <input type="checkbox"/> 2.3.0 (SNAPSHOT) <input type="checkbox"/> 2.2.7 (SNAPSHOT)	
<input checked="" type="checkbox"/> 2.2.6 <input type="checkbox"/> 2.1.14 (SNAPSHOT) <input type="checkbox"/> 2.1.13	
Project Metadata	
Group	com.uca.capas
Artifact	ejemplo
Name	ejemplo
Description	Proyecto Spring MVC con Spring Boot
Package name	com.uca.capas.ejemplo
Packaging	<input checked="" type="checkbox"/> Jar <input type="checkbox"/> War
Java	<input type="checkbox"/> 14 <input type="checkbox"/> 11 <input checked="" type="checkbox"/> 8

3. En la sección **Dependencies** dar clic al botón **Add Dependencies** y buscaremos "Spring Web":

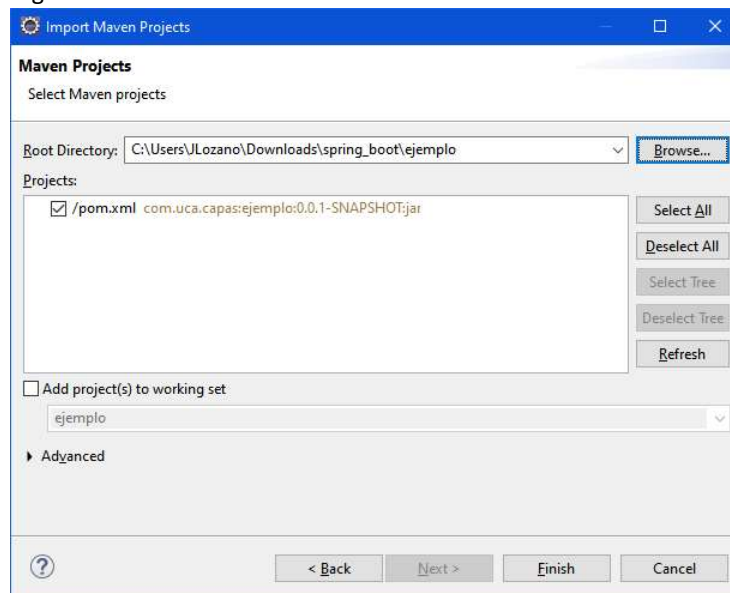


Damos clic en él y nos aparecerá ya en nuestras dependencias seleccionadas.

- Ahora daremos clic en el botón **Generate** (o podemos presionar las teclas Control + Enter) y descargaremos el proyecto (comprimido) con la configuración que hemos realizado.
- Descomprimos el proyecto y lo guardamos en una carpeta local.
- Ahora, con nuestro IDE (en este caso Eclipse) procedemos a importar el proyecto Maven.
- Para eso, damos clic derecho en el Project Explorer y seleccionamos Import -> Existing Maven Projects:



- Seleccionamos la carpeta que acabamos descomprimir y nos reconocerá el archivo pom.xml de nuestro proyecto Maven descargado.



- Damos clic en Finish y tendremos nuestro proyecto importado.



Revisemos el pom.xml de nuestro proyecto y específicamente el nodo **dependencies** observamos:

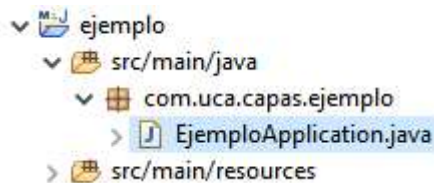
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Básicamente al agregar la dependencia **Spring Web** nos genera un pom con dos dependencias:

- Spring-boot-starter-web: El cual contiene las librerías necesarias para utilizar Spring MVC
- Spring-boot-starter-test: Esta dependencia no es necesaria para el funcionamiento de Spring MVC, sin embargo se incluye para la realización de pruebas unitarias

En cuanto a los paquetes, vemos que solo encontramos uno:



```
ejemplo
├── src/main/java
│   └── com.uca.capas.ejemplo
│       └── EjemploApplication.java
└── src/main/resources
```

El cual contiene una clase llamada EjemploApplication.java, que contiene lo siguiente:

```
package com.uca.capas.ejemplo;

import org.springframework.boot.SpringApplication;

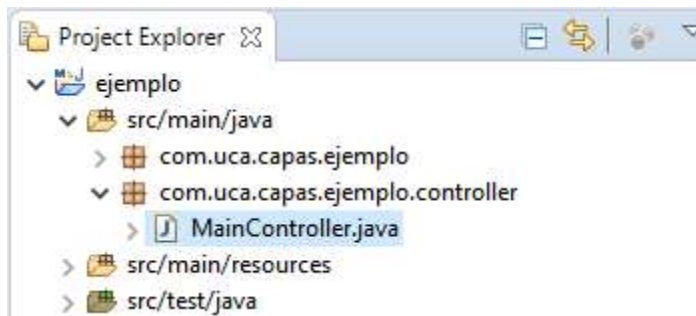
@SpringBootApplication
public class EjemploApplication {

    public static void main(String[] args) {
        SpringApplication.run(EjemploApplication.class, args);
    }

}
```

La anotación **@SpringBootApplication**, el cual afecta a la clase, especifica que este es un proyecto Spring Boot, y de esta manera realiza todas las configuraciones necesarias de manera automática para que (en este caso) Spring MVC funcione correctamente.

Ahora crearemos un paquete llamado **com.uca.capas.ejemplo.controller**, el cual contendrá nuestras clases Java que tendrán métodos mapeados a una URL (como los servlets) y que serán ejecutados al momento de llamarlos desde el navegador. Dentro de este paquete crearemos la clase **MainController.java**:



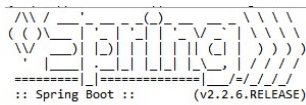
La clase **MainController.java** tendrá lo siguiente:

```
1 package com.uca.capas.ejemplo.controller;
2
3 import java.util.Calendar;
4
5 import org.springframework.stereotype.Controller;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
9 @Controller
10 public class MainController {
11
12     @RequestMapping("/ejemplo")
13     public @ResponseBody String ejemplo() {
14         String texto = "Hola Mundo desde Spring MVC! La hora actual es: ";
15         return texto.concat(Calendar.getInstance().getTime().toString());
16     }
17
18 }
```

Anotaciones:

- **@Controller (línea 9):** Esta anotación le indica a Spring que esta clase (y sus métodos) actúan como controladores. Al momento de iniciar la aplicación (al levantar el servidor) Spring escanea las clases en busca de estas anotaciones, cuando encuentra una clase con esta anotación recorre sus métodos para mapear la URL asociada (con la anotación **@RequestMapping**) al nombre del método. Con esto el **DispatcherServlet**, cuando recibe una petición (un submit de un form), sabrá a que método redirigirla.
- **@RequestMapping (línea 12):** Esta anotación sirve para asociar una URL a un método específico, el cual es el que será ejecutado cuando el **DispatcherServlet** reciba una petición, y la redirija a dicho método. Esta anotación recibe de parámetro la URL a la que estará asociado el método.
- **@ResponseBody (línea 13):** Esta anotación convierte el dato que se está retornando en el cuerpo de la respuesta HTTP, para que sea interpretada por el navegador.

Ahora iniciaremos el aplicativo, ejecutando el método **main** de la clase **EjemploApplication.java** (lo ejecutaremos como una aplicación Java normal)



```
2020-04-15 13:36:03.765 INFO 17040 --- [main] c.uca.capas.ejemplo.EjemploApplication : Starting EjemploApplication on DRML17039 with PID 17040 (C:\Users\JLozano
2020-04-15 13:36:03.769 INFO 17040 --- [main] c.uca.capas.ejemplo.EjemploApplication : No active profile set, falling back to default profiles: default
2020-04-15 13:36:05.144 INFO 17040 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-04-15 13:36:05.153 INFO 17040 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-04-15 13:36:05.153 INFO 17040 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]
2020-04-15 13:36:05.237 INFO 17040 --- [main] o.a.c.c.c.[Tomcat].[/] : Initializing Spring embedded WebApplicationContext
2020-04-15 13:36:05.237 INFO 17040 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1425 ms
2020-04-15 13:36:05.386 INFO 17040 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-04-15 13:36:05.525 INFO 17040 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-04-15 13:36:05.528 INFO 17040 --- [main] c.uca.capas.ejemplo.EjemploApplication : Started EjemploApplication in 2.093 seconds (JVM running for 2.517)
```

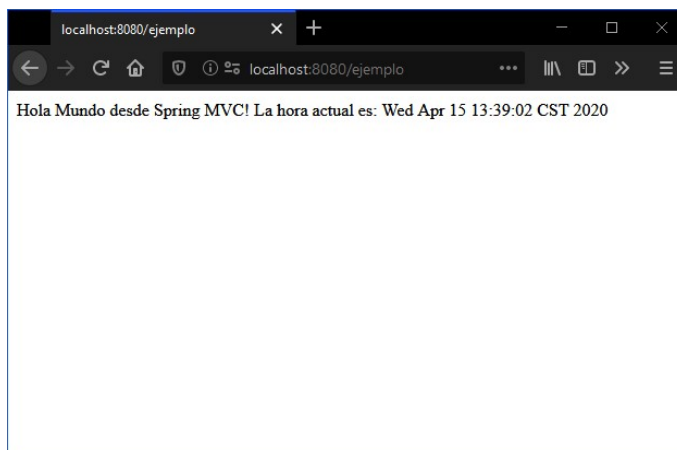
Cuando la consola llegue a este punto sabremos que ha iniciado correctamente.

Ahora, en un navegador, ingresaremos a la siguiente URL:

<http://localhost:8080/ejemplo>

Notemos que no hemos especificado el nombre de la aplicación como al ejecutar los servlets (<http://localhost:8080/aplicacion/ejemplo>), ya que al estar ejecutando un servidor Tomcat embebido no tendremos más que una aplicación ejecutándose, por lo que no tiene sentido especificarlo (aunque si es posible).

Al ingresar a la URL veremos:



De igual forma, los métodos de los controladores pueden recibir los parámetros **HttpServletRequest** y **HttpServletResponse** para manejo adicional de funcionalidades:

```
@RequestMapping("/parametro")
public @ResponseBody String parametro(HttpServletRequest request) {
    Integer param = Integer.parseInt(request.getParameter("dia"));
    String dia;
    switch (param) {
        case 1: dia = "Lunes";
        break;
        case 2: dia = "Martes";
        break;
        case 3: dia = "Miercoles";
        break;
        case 4: dia = "Jueves";
        break;
        case 5: dia = "Viernes";
        break;
        case 6: dia = "Sabado";
        break;
        case 7: dia = "Domingo";
        break;
        default: dia = "Ningun dia seleccionado";
        break;
    }

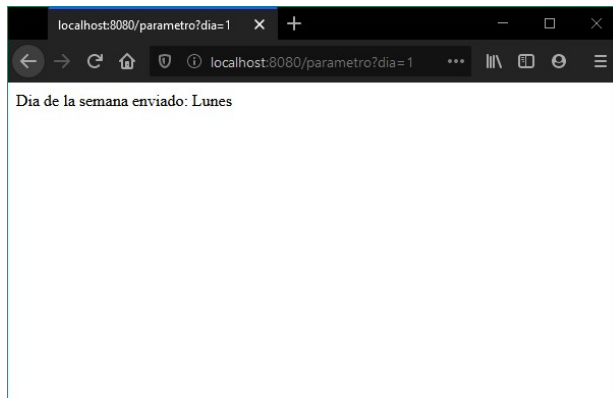
    return "Dia de la semana enviado: " + dia;
}
```

En este ejemplo, el método recibe de parámetro el objeto **HttpServletRequest**, el cual utilizamos para recibir un parámetro de nombre **día** que es enviado por la URL, justamente como lo hacíamos con los Servlets.

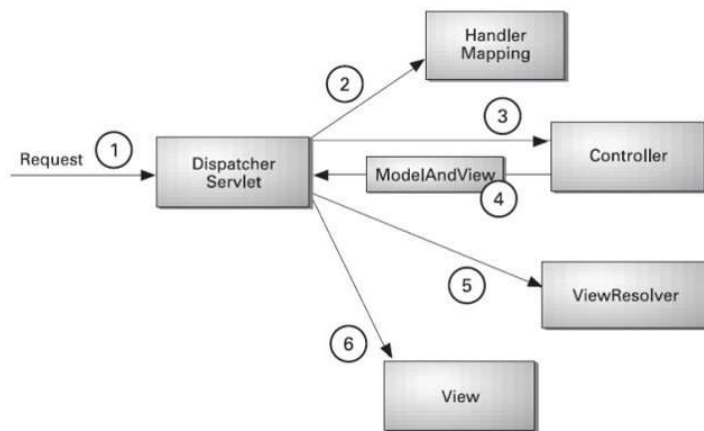
Entonces al ingresar a la URL:

<http://localhost:8080/parametro?dia=1>

Nos devolverá:



Recordando el flujo de funcionamiento de Spring MVC:



1. El Dispatcher Servlet (DS) recibe la petición al ingresar la URL y presionar Enter.
2. Busca en el Handler Mapping el método al que está asociada la URL
3. Cuando la encuentra lo redirige al controlador (método) correspondiente (en este caso **ejemplo()**)
4. En este caso no hemos utilizado un ModelAndView para retornar, sino un tipo de dato String que es devuelto al DispatcherServlet, y este es devuelto al usuario en una respuesta HTTP.

En la siguiente clase veremos un ejemplo con ModelAndView

Repositorio git del proyecto: <https://github.com/jlozano86/springmvc.git>