
Spring boot hola mundo

From zero to hero

Jorge López

2017-02-20



El presente documento nos enseñará a crear un API REST con Spring Boot en pocos minutos.

Contenidos

1 UD07-01.- Spring Boot Web ‘Hola mundo’.	2
1.1 Creación del proyecto base	2
2 Descripción del proyecto.	4
2.1 Directorio src/main/java	4
2.2 Clase UDxxAyyApplication.	5
2.3 Directorio src/main/resources	5
2.4 Fichero application.properties	6
2.5 Directorio src/test/java	6
2.6 Directorio src/test/resources	7
2.7 Fichero pom.xml	7
3 “Hola Mundo” Controller	7
4 Anexo I: código fuente:	10

1. UD07-01.- Spring Boot Web ‘Hola mundo’

1.1. Creación del proyecto base

Para crear un nuevo proyecto Spring Boot, para implementar servicios REST, debemos acceder a la página <https://start.spring.io/>, y rellenamos el formulario:

The screenshot shows the Spring Initializr web form. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.6.0 (M3)' selected. The 'Project Metadata' section has the following values: Group: ad, Artifact: ud07, Name: ud07, Package name: ad.ud07, Packaging: Jar, and Java: 11. The 'Dependencies' section shows 'Spring Web', 'H2 Database', and 'Spring Data JDBC' selected. The 'GENERATE' button is highlighted with a red box.

Figura 1: Formulario de creación de proyecto Spring

Donde:

Project	maven project	Seleccionamos maven como gestor de dependencias.
Lenguaje	Java	Seleccionamos Java como lenguaje de programación.
Spring boot	2.6.0 (M3)	Seleccionamos la versión de Spring Boot a utilizar.

Group	ad	Introducimos el GroupId del artefacto a generar.
Artifact	udxxaxx	Introducimos el ArtifactId del artefacto a generar.
Name	udxxaxx	Introducimos el nombre del artefacto a generar.
Package name	ad.udxx	Introducimos el paquete base del proyecto.
Packaging	jar	Seleccionamos el tipo de artefacto a generar.
Java	11	Seleccionamos la versión de Java a utilizar.

Añadimos las dependencias necesarias, pulsando el botón [Add dependencies](#), y seleccionando los paquetes que necesitamos para nuestra aplicación.

Por ejemplo, para un microservicio con acceso a datos utilizando JDBC, seleccionaremos:

- Spring Web → nos permite implementar servicios REST.
- H2 Database → nos permite utilizar la BBDD en memoria H2.
- Spring Data JDBC → nos permite acceder a datos utilizando JDBC.



Durante el curso será necesario generar proyectos con distintas dependencias, dependiendo de la unidad didáctica en la que nos encontremos, para probar las distintas tecnologías de acceso a datos, como jdbc y jpa.

Pulsamos el botón Generate, para descargar un archivo ZIP que contendrá el esqueleto de nuestro proyecto.

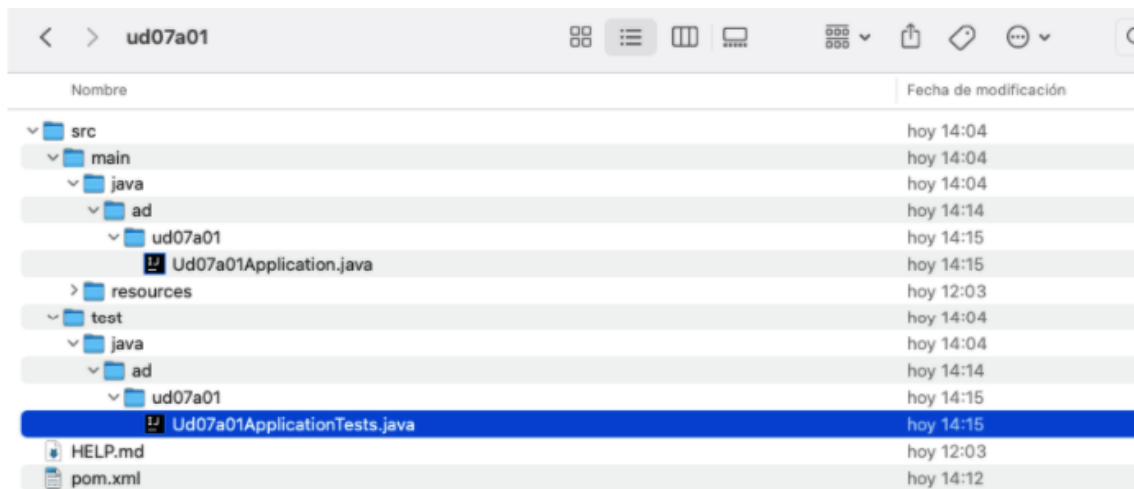


Figura 2: Estructura de proyecto descargado

Una vez descargado el proyecto base generado, y descomprimido, en caso de que existan, podemos borrar los siguientes ficheros:

- mvnw
- mvnw.cmd

Y ya podemos abrir el proyecto en nuestro entorno favorito, con soporte a proyectos Maven.



Durante el curso, las explicaciones se realizarán utilizando Eclipse IDE como entorno de desarrollo, aunque se puede utilizar cualquier entorno con soporte para proyectos Maven, como IntelliJ.

2. Descripción del proyecto.

2.1. Directorio src/main/java

El directorio src/main/java contendrá las clases (código fuente) de nuestro microservicio. En el podemos encontrar los paquetes de la aplicación.

En nuestro caso, el paquete principal de la aplicación será “ad.udxxay”, donde:

- xx se corresponde con el número de la unidad que estamos realizando.
- aa se corresponde con el número de la actividad que estamos implementando.

Por ejemplo, para la actividad 1 de la unidad 7, utilizaremos “ad.ud07a01”

2.2. Clase UDxxAyyApplication.

En el paquete principal, encontramos la clase UDxxAyyApplication, responsable de iniciar el microservicio, cuando se ejecuta el método main.

Si observamos el código fuente:

```
1 package ad.ud07a01;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Ud07a01Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Ud07a01Application.class, args);
11     }
12
13 }
```

- La clase incluye la anotación `@SpringBootApplication`, marcando la clase de inicio del microservicio.
- El método main incluye la llamada al método “`SpringApplication.run`”, responsable de iniciar el microservicio.

Al arrancar la aplicación, Spring Framework analiza el código fuente de la aplicación, en busca de anotaciones en el código, que van indicando al framework como debe comportarse la aplicación:

```
1 @SpringBootApplication
2 public class MiAplicacion
3
4 @Controller
5 public class MiControladorRest
6
7 @Repository
8 public class MiRepositorio
```

2.3. Directorio src/main/resources

El directorio `src/main/resources` contendrá los recursos estáticos (que no necesitan compilación), necesarios para el correcto funcionamiento de la misma.

Entre estos recursos estáticos podemos encontrar:

- Ficheros de configuración de datasources.

- Ficheros de configuración de la aplicación.
- Ficheros de datos.
- ...

2.4. Fichero application.properties

El fichero application.properties contiene los parámetros de configuración de nuestra aplicación REST. Podemos, entre otras cosas:

- Indicar el puerto al que queremos que responda el microservicio. Por ejemplo, el puerto 8081:

```
1 server.port 8081
```

- Indicar el contexto (path base) sobre el que responderá nuestro microservicio. Por ejemplo, la ruta '/api':

```
1 Server.servlet.context-path="/api"
```

- Configurar una base de datos en memoria, con h2. Por ejemplo, la base de datos 'empresa_ad', con usuario 'sa', y password 'password':

```
1 spring.datasource.url=jdbc:h2:mem:empresa_ad
2 spring.datasource.username=sa
3 spring.datasource.password=password
```

- Habilitar la consola de h2, para poder acceder a un cliente Web a través de un path del microservicio. Por ejemplo, sobre el path 'h2-console':

```
1 spring.h2.console.enabled=true
2 spring.h2.console.path=/h2-console
3 spring.h2.console.settings.trace=false
4 spring.h2.console.settings.web-allow-others=false
```

- Configurar un datasource sobre la base de datos h2 configurada:

```
1 spring.datasource.url=jdbc:h2:mem:empresa_ad
2 spring.datasource.username=sa
3 spring.datasource.password=password
```

2.5. Directorio src/test/java

El directorio src/test/java contendrá las clases (código fuente) de nuestros test unitarios. La estructura de este directorio es equivalente a la estructura del directorio src/main/java.

2.6. Directorio src/test/resources

El directorio src/test/resources contendrá los recursos estáticos necesarios para la correcta ejecución de los test unitarios, pero que no son necesarios para la ejecución de la aplicación. La estructura de este directorio es equivalente a la estructura del directorio src/main/resources.

2.7. Fichero pom.xml

El fichero pom.xml contiene la meta-información del proyecto:

- Proyectos de los que hereda componentes (parent).
- Grupo, artefacto, nombre y versión (groupId, artifactId, name, version).
- Versión de java (properties.java.version).
- Dependencias, o librerías necesarias para la compilación y ejecución (dependencies) .

3. “Hola Mundo” Controller

Vamos a crear nuestro primer controlador, que permitirá realizar una petición GET al recurso /hola.

1. Creamos el paquete ‘ad.udxxayy.controllers’ en nuestra aplicación.
2. Creamos la clase HolaMundoController dentro del paquete que hemos creado.
 1. Le añadimos la anotación @RestController.
 2. Le añadimos la anotación @RequestMapping(“/hola”).
3. Añadimos el método public String holaMundo():
 1. Le añadimos la anotación @GetMapping(“”)
 2. El método debe devolver la cadena ‘Hola mundo’.
4. Añadimos el método public String holaPersona(@PathVariable String nombre):
 1. Le añadimos la anotación @GetMapping(“/{nombre}”)
 2. El método debe devolver la cadena ‘Hola’ + nombre.

Quedando así:

```
1 package ad.ud07a01.controllers;  
2  
3 import org.springframework.web.bind.annotation.GetMapping;  
4 import org.springframework.web.bind.annotation.PathVariable;  
5 import org.springframework.web.bind.annotation.RequestMapping;  
6 import org.springframework.web.bind.annotation.RestController;
```



```
7
8 @RestController
9 @RequestMapping("/hola")
10 public class HolaMundoController {
11
12     @GetMapping("")
13     public String holaMundo() {
14         return "hola mundo";
15     }
16
17     @GetMapping("/{nombre}")
18     public String holaPersona(@PathVariable String nombre) {
19         return "hola " + nombre;
20     }
21 }
```

Tras implementar el método, arrancamos la aplicación, ejecutando la clase UdxxayyApplication.

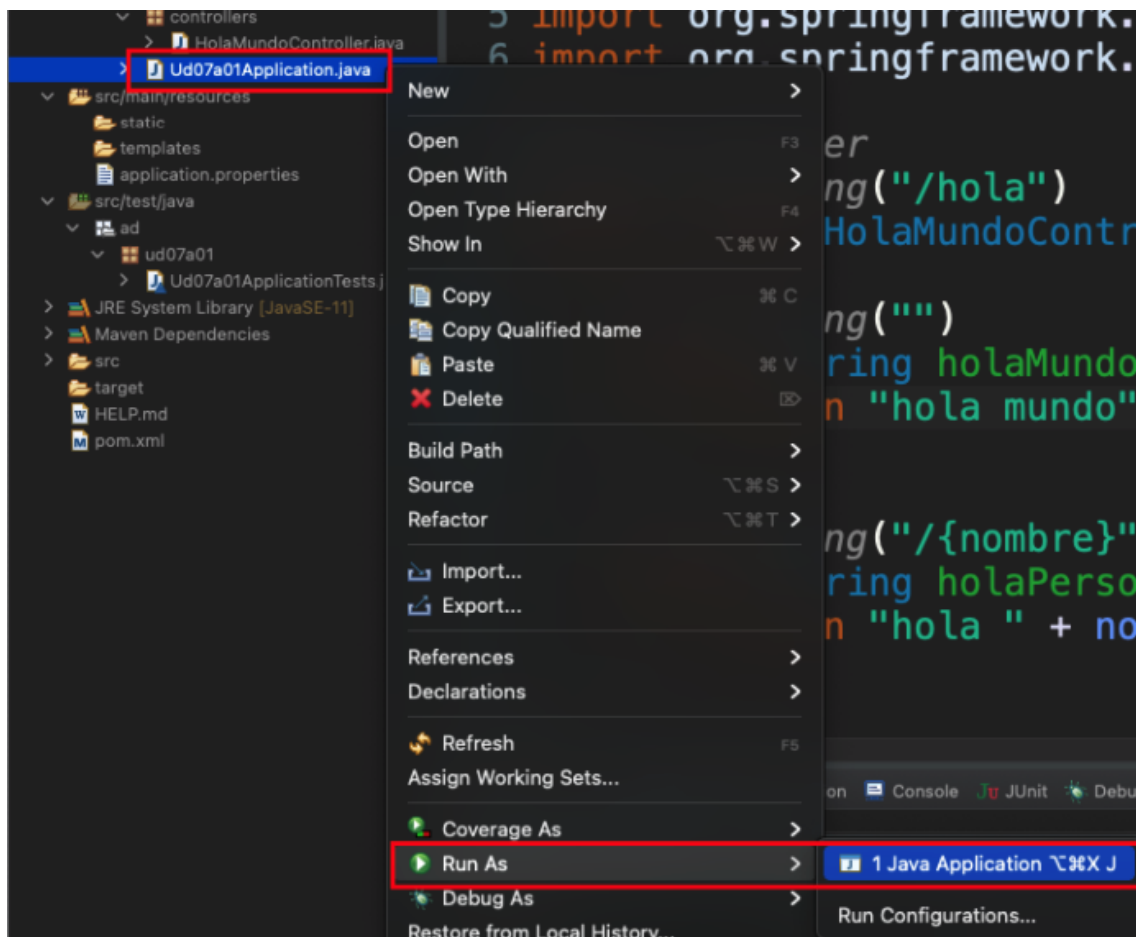
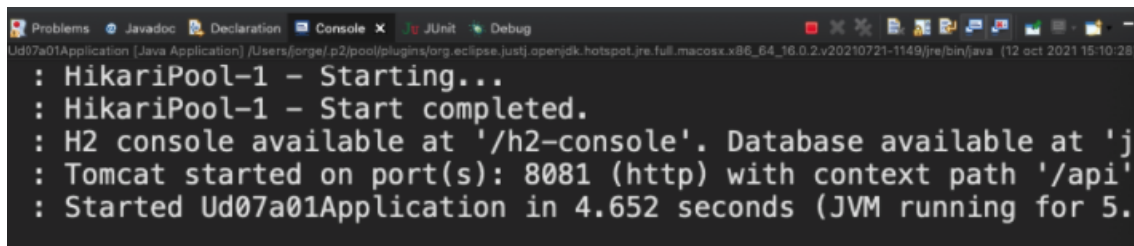


Figura 3: Arranque de la aplicación.

Y observamos el log de la aplicación, ahsta que arranque:



```
Ud07a01Application [Java Application] /Users/jorge/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.macosx.x86_64_16.0.2.v20210721-1149/jre/bin/java (12 oct 2021 15:10:28)
: HikariPool-1 - Starting...
: HikariPool-1 - Start completed.
: H2 console available at '/h2-console'. Database available at 'jdbc:hs:net://localhost:1521/ud07a01'
: Tomcat started on port(s): 8081 (http) with context path '/api'
: Started Ud07a01Application in 4.652 seconds (JVM running for 5.711s)
```

Figura 4: Log del arranque de la aplicación.

Cuando la aplicación indique que ha arrancado, podemos hacer una petición a los servicios implementados. Para conocer la ruta de los servicios, tenemos que tener en cuenta:

- Los parametros del fichero application.properties:

```
1 server.port 8081
2 server.servlet.context-path=/api
```

- Los parámetros de la clase Controller:

```
1 @RestController
2 @RequestMapping("/hola")
3 public class HolaMundoController {
4
5     @GetMapping("")
6     public String holaMundo() {
7         ...
8     }
9
10    @GetMapping("/{nombre}")
11    public String holaPersona(@PathVariable String nombre) {
12        ...
13    }
14 }
```

Teniendo en cuenta lo anterior, tenemos dos servicios:

- `http://localhost:8081/api/hola`
- `http://localhost:8081/api/hola/{nombre}` -> donde {nombre} es una variable.

Abre un navegador web, y prueba las rutas:



Figura 5: Prueba de nuestro servicio con el navegador web.

Al realizar la petición a `/api/hola`, Spring trata de buscar la ruta del controlador que casa con la petición, y la encuentra en el `@controller` con `@RequestMapping('/hola')`, en concreto con el método `@GetMapping("")`.

Al realizar la petición a `/api/hola/pepe`, Spring trata de buscar la ruta del controlador que casa con la petición, y la encuentra en el `@controller` con `@RequestMapping('/hola')`, en concreto con el método `@GetMapping('/nombre')`.

4. Anexo I: código fuente:

[Spring Boot Web 'Hola mundo'.](#)