16_spring_boot

October 15, 2020

1 Spring Boot.

1.1 Preliminares.

Antes de empezar, es necesario instalar y configurar algunas herramientas básicas.

[ ]:
```
sudo apt update -y && sudo apt upgrade -y
```

[ ]:
```
sudo apt install ant ant-optional antlr groovy ivy junit4 openjdk-11-jdk -y
```

[ ]:
```
wget https://downloads.gradle-dn.com/distributions/gradle-6.1-bin.zip
```

[ ]:
```
sudo unzip gradle-6.1-bin.zip -d /opt/
```

[ ]:
```
sudo ln -sf /opt/gradle-6.1/bin/gradle /usr/local/bin/gradle
```

[ ]:
```
gradle -v
```

1.2 *Spring Framework.*

Spring Framework o *Spring* es un marco de trabajo de código abierto enfocado al desarrollo de aplicaciones empresariales basado en *Java*. La versión más reciente de este framework es *Spring 5*.

La arquitecture de *Spring* se basa en la Inversión de Control y la Inyección de dependencias, de tal modo que es posible crear aplicaciones muy complejas declarando sus componentes.

A lo largo del tiempo, *Spring* ha desarrollado diversos proyectos alrededor del su framework principal, siendo algunos de ellos:

- *Spring Cloud*.
- *Spring Boot*.
- *Spring Data*.
- *Spring Security*.
- *Spring Intergation*.
- *Spring Batch*.
- *Spring Mobile*.
- *Spring for Android*.
- *Spring AMQP*.

1.3 *Spring Boot.*

Spring Boot es una herramienta que forma parte de *Spring Framework*, la cual permite desarrollar y desplegar aplicaciones web de forma rápida y con muy pocas configuraciones.

- Es compatible con *Apache Maven* y *Gradle*.
- Permite utilizar código escrito en *Java*, *Groovy* y *Kotlin*.
- Cuenta con todo el stack de *Spring* para desarrollo de aplicaciones web.
- Cuenta con una interfaz de línea de comandos (CLI).
- Permite crear plantillas de proyectos de forma automatizada.

La documentación de referencia de Spring Boot puede ser consultada desde: https://docs.spring.io/spring-boot/docs/2.1.12.RELEASE/reference/html/

1.4 La *CLI* de *Spring Boot.*

Spring Boot permite ejecutar aplicaciones de forma rápida directamente desde una terminal.

La referencia de la *CLI* puede ser consultada desde:

https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-cli.html

1.4.1 Instalación y configuraciónde la *CLI.*

- A continuación se descargará, instala'ra y configurará la *CLI* de *Spring Boot*.

[ ]:
```
wget https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.
↪2.4.RELEASE/spring-boot-cli-2.2.4.RELEASE-bin.tar.gz
```

[ ]:
```
tar xvfz spring-boot-cli-2.2.4.RELEASE-bin.tar.gz -C ~/
```

[ ]:
```
chmod -R +rw ~/spring-2.2.4.RELEASE
```

[ ]:
```
mv ~/spring-2.2.4.RELEASE ~/spring
```

[ ]:
```
export PATH=$PATH:~/spring/bin
```

[ ]:
```
spring --version
```

1.4.2 El "*Hola, Mundo*" desde la *CLI* de *Spring Boot.*

El archivo src/17/hola_mundo/hola.groovy contiene el siguiente código, el cual desplegará un mensaje simple desde un servicio web de *Spring Boot*:

```
@Controller
class Ejemplo {
   @RequestMapping("/")
   @ResponseBody
   public String hola() {
      "Hola, Mundo"
   }
```

```
}
```

- La siguiente celda despelgará un servicio web en http://localhost:8080 a partir del archivo previo.

[ ]:
```
spring run src/17/hola_mundo/hola.groovy
```

1.4.3 *Spring Initializr.*

Para facilitar el desarrollo de plantillas para el desarrollo de aplicaciones a la medida de forma ágil, está disponible el servico en línea llamado *Spring Initializr*, mediante el cual es posible crear una estructura de archivos compatibles con *Apache Maven* y *Gradle* que incluyen los componentes y dependencias de un proyecto específico.

El servicio se encuentra en https://start.spring.io/

Al final se podrá descargar un archivo comp-rimido que contiene lo necesario para comenzar a desarrolla una aplicación.

1.5 El "*Hola, Mundo*" con una plantilla de *Spring Boot.*

El archivo src/17/demo.zip contiene una estructura de directorios y archivos creada mediante *Spring Initializr*.

- Se utilizará *Gradle* para construir un archivo *.jar*.
- Se seleccionó *Groovy* como el lemguaje por defecto de la aplicación.
- El grupo del proyecto es `tutorial`.
- El nombre del proyecto es `demo`.

- A continuación se desempaquetará el proyecto `demo`.

[ ]:
```
cd src/17
```

[ ]:
```
unzip demo.zip
```

- El archivo src/17/DemoApplication.groovy contiene el siguiente código:

```
package com.tutorial.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController

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

```
    @RequestMapping(value = "/")
    public String hola() {
        return "<h1>Hola, Mundo.</h1>";
    }
}
```

1.5.1 La anotación `@SpringBootApplication`.

Esta anotación indica que el archivo contiene a una aplicación de *Spring Boot*.

La refecia puede ser consultada en https://docs.spring.io/spring-boot/docs/2.1.12.RELEASE/reference/html/using-boot-using-springbootapplication-annotation.html

1.5.2 La anotación `@RestController`.

Esta anotación invoca a un controlador *MVC*, especializado en servicios *REST*.

1.5.3 La anotación `@RequestMapping`.

Esta anotación permite ligar una URI local con la definición de un método . El objeto que regrese dicho métodos será publicado en la URL.

```
@RequestMapping(value = "<URL>")
  public String <metodo>() {
     return <contenido>;
  }
```

La siguiente liga hace referencia a unn artículo que describe la anotación https://springframework.guru/spring-requestmapping-annotation/

- A continuación se copiará el archivo `DemoApplication.groovy` en le directorio `demo/src/main/groovy/com/tutorial/demo/DemoApplication.groovy`.

[ ]:
```
cp DemoApplication.groovy demo/src/main/groovy/com/tutorial/demo/
↪DemoApplication.groovy
```

[ ]:
```
cd demo
```

- Se construirá el archivo `build/libs/demo-0.0.1-SNAPSHOT.jar` usando *Gradle*.

[ ]:
```
gradle clean build
```

- Se ejecutará dicho archivo, el cual levantará un servicio que podrá ser consultado en http://localhost:8080

[ ]:
```
java -jar build/libs/demo-0.0.1-SNAPSHOT.jar
```

- Se regresará al directorio superior.

[ ]:
```
cd ..
```

1.6 Despliegue de un proyecto de *API REST* simple.

El archivo `demo-rest.zip` contiene una estructura de directorios y archivos creada mediante *Spring Initializr*.

- Se utilizará *Gradle* para construir un archivo *.jar*.
- Se seleccionó *Groovy* como el lemguaje por defecto de la aplicación.
- El grupo del proyecto es `tutorial`.
- El nombre del proyecto es `demo-rest`.
- La aplicación utiliza una estructura *HashMap* para almacenar objetos que contienen los atributos:
  - `id`
  - `name`

1.6.1 Defincón del proyecto.

- La *API REST* corresponde a un sistema *CRUD* para objetos instanciados de una clase `Product`.
- El endpoint `localhost:8080/products` tiene habilitados los métodos:
  - `GET` para obtener un listado de los productos en formato *JSON*.
  - `POST` el cual dará de alta un nuevo producto al recibir datos en formato *JSON* con la estructura `{"id":"<número>", "name":"<nombre>"}`.
- El endpoint `localhost:800/products/<id>`, donde `<id>` es un número de identificación, tiene habilitados los métodos.
  - `PUT` el cual modificará a un producto al recibir datos en formato *JSON* con la estructura `{"name":"<nombre>"}`.
  - `DELETE` el cual eliminará al producto con el valor `<id>` correspondiente.
- Los datos serán almacenados y gestionados en una estructura `HashMap`.
- A continuación se desempaquetará el proyecto `demo-rest`.

[ ]:
```
unzip demo-rest.zip
```

1.6.2 El archivo `DemoRestApplication.groovy`.

```
package com.tutorial.demorest

import org.springframework.boot.SpringApplication
import org.springframework.boot.autoconfigure.SpringBootApplication

@SpringBootApplication
class DemoRestApplication {

    static void main(String[] args) {
        SpringApplication.run(DemoRestApplication, args)
    }

}
```

1.6.3 El archivo `ProductServiceController.java`.

Este archivo es el encargado de la parte del controlador de *MVC*, realziando operaciones específicas de un *endpoint* en función del métido utilizado para acceder a este.

```
package com.tutorial.demorest.controller;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.tutorial.demorest.model.Product;

@RestController
public class ProductServiceController {
   private static Map<String, Product> productRepo = new HashMap<>();
   static {
      Product leche = new Product();
      leche.setId("1");
      leche.setName("Leche");
      productRepo.put(leche.getId(), leche);

      Product pan = new Product();
      pan.setId("2");
      pan.setName("Pan de caja");
      productRepo.put(pan.getId(), pan);
   }

   @RequestMapping(value = "/products/{id}", method = RequestMethod.DELETE)
   public ResponseEntity<Object> delete(@PathVariable("id") String id) {
      productRepo.remove(id);
      return new ResponseEntity<>("Producto eliminado.", HttpStatus.OK);
   }

   @RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
   public ResponseEntity<Object> updateProduct(@PathVariable("id") String id, @RequestBody Proc
      productRepo.remove(id);
      product.setId(id);
      productRepo.put(id, product);
      return new ResponseEntity<>("Producto actualizado.", HttpStatus.OK);
   }
```

```
    @RequestMapping(value = "/products", method = RequestMethod.POST)
    public ResponseEntity<Object> createProduct(@RequestBody Product product) {
        productRepo.put(product.getId(), product);
        return new ResponseEntity<>("Producto creado.", HttpStatus.CREATED);
    }

    @RequestMapping(value = "/products")
    public ResponseEntity<Object> getProduct() {
        return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);
    }
}
```

1.6.4 El paquete `org.springframework.http`.

Este paquete contiene herramientas par la gestión de peticiones y respuestas *HTTP*.

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/

La clase `org.springframework.http.ResponseEntity`. https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html

La clase `org.springframework.http.HttpStatus`. https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html

1.7 El archivo *Product.java*.

```
package com.tutorial.demorest.model;

public class Product {
    private String id;
    private String name;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

- Se copiarán los archivos a `demo-rest/src/main/groovy/com/tutorial/demorest/`.

[ ]:
```
cp DemoRestApplication.groovy Product.java ProductServiceController.java␣
↪demo-rest/src/main/groovy/com/tutorial/demorest/
```

- Se creará el archivo *.jar*.

[ ]:
```
cd demo-rest
```

[ ]:
```
gradle clean build
```

- Se ejecutará dicho archivo, el cual levantará un servicio que podrá ser consultado en http://localhost:8080/products

[ ]:
```
java -jar build/libs/demo-rest-0.0.1-SNAPSHOT.jar
```