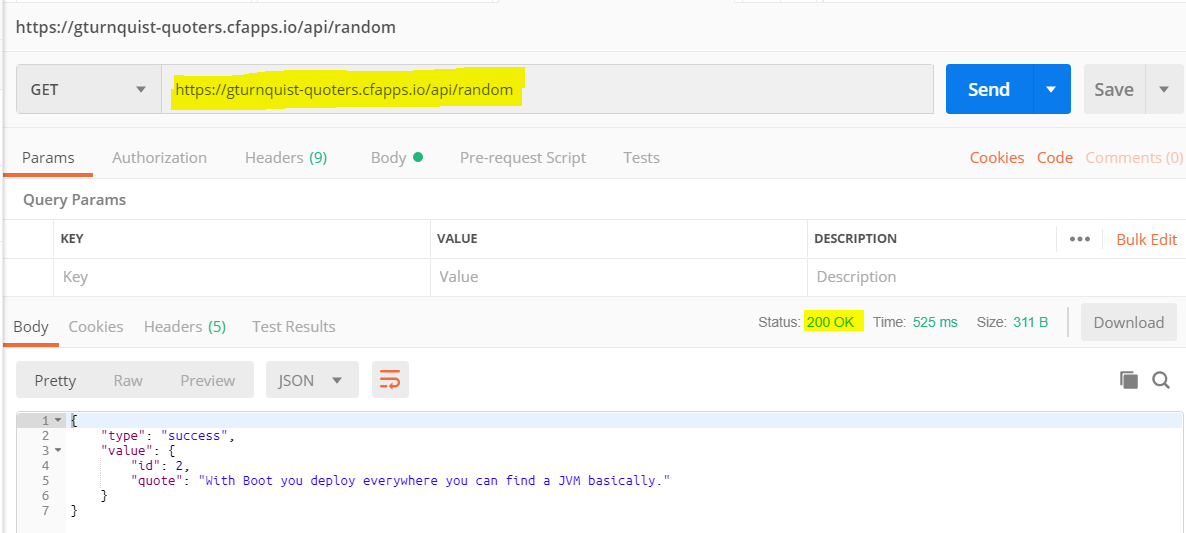
**CONSUMIR UN REST --- Spring.io**



**import com.fasterxml.jackson.annotation.JsonIgnoreProperties;**  
@JsonIgnoreProperties(ignoreUnknown = **true**)  
**public class Quote** {  
  
 **private** String **type**;  
 **private** Value **value**;  
  
 **public Quote**() { }  
 …  
 **@Override** **public** String **toString**() {  
 **return "Quote{"** + **"type='"** + **type** + **'\''** + **", value="** + **value** + **'}'**;  
 }  
}

…

**import com.fasterxml.jackson.annotation.JsonIgnoreProperties;**  
**@JsonIgnoreProperties(ignoreUnknown = true)  
public class Value** {  
  
 **private** Long **id**;  
 **private** String **quote**;  
  
 **public Value**() { }  
  
 **@Override  
 public String toString(){return "Value{" + "id=" + id +", quote='" + quote + '\'' + '}';** }  
}

…

**import** com.example.loggin.com.example.model.Quote;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.web.client.RestTemplate;  
  
**public class App** {  
  
 **private static final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(App.**class**);  
  
 **public static void main**(String[] args) {  
  
 **RestTemplate** restTemplate = **new** RestTemplate();  
 **Quote** forObject = **restTemplate**.**getForObject**(**"https://gturnquist-quoters.cfapps.io/api/random"**, Quote.**class**);  
 ***LOGGER*.info(forObject.toString());** }  
}

**======================= LO MISMO PERO EN REST**

**import** com.example.loggin.com.example.model.Quote;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.boot.CommandLineRunner;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.boot.web.client.RestTemplateBuilder;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** org.springframework.web.client.RestTemplate;  
  
**@RestController  
@SpringBootApplication  
public class LogginApplication {**  
 **private static final Logger *LOGGER* = LoggerFactory.*getLogger*(LogginApplication.class);**  
 **@RequestMapping("/")** **public** String **devuelve**() {

***LOGGER***.info(**"SOMEONE HIT!!!"**);  
 ***LOGGER***.error(**"error"**);  
 ***LOGGER***.warn(**"warn"**);  
 ***LOGGER***.info(**"info"**);

***LOGGER***.debug(**"debug"**); *// no debiera mostrarse* **return "Hello Daniel!"**;  
 }  
  
 **public static void main**(String[] args) {  
 SpringApplication.*run*(LogginApplication.**class**, args);

***LOGGER***.error(**"error"**);  
 ***LOGGER***.warn(**"warn"**);  
 ***LOGGER***.info(**"info"**);  
 ***LOGGER***.debug(**"debug"**);

}  
 **@Bean  
 public RestTemplate restTemplate(RestTemplateBuilder builder) {** **return** builder.build();  
 }  
 **@Bean  
 public CommandLineRunner run(RestTemplate restTemplate) throws Exception {** **return** args -> {  
 Quote quote = restTemplate.getForObject(**"https://gturnquist-quoters.cfapps.io/api/random"**, Quote.**class**);  
 ***LOGGER***.info(quote.toString());  
 };  
 }  
}

15**:**12 **[**main**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** Starting LogginApplication on DESKTOP**-**P3FK8PT with PID 3056 **(**C**:**\Users\Daniel\Downloads\loggin\out\production\classes started by Daniel in C**:**\Users\Daniel\Downloads\loggin**)**

15**:**12 **[**main**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** No active profile set**,** falling back to **default** profiles**:** **default**

15**:**12 **[**main**]** INFO o**.**s**.**b**.**w**.**e**.**tomcat**.**TomcatWebServer **\*\*** Tomcat initialized with port**(**s**):** 8080 **(**http**)**

15**:**12 **[**main**]** INFO o**.**a**.**coyote**.**http11**.**Http11NioProtocol **\*\*** Initializing ProtocolHandler **[**"http-nio-8080"**]**

15**:**12 **[**main**]** INFO o**.**a**.**catalina**.**core**.**StandardService **\*\*** Starting service **[**Tomcat**]**

15**:**12 **[**main**]** INFO o**.**a**.**catalina**.**core**.**StandardEngine **\*\*** Starting Servlet engine**:** **[**Apache Tomcat**/**9.0.21**]**

15**:**12 **[**main**]** INFO o**.**a**.**c**.**c**.**C**.[**Tomcat**].[**localhost**].[/] \*\* Initializing Spring embedded WebApplicationContext**

15**:**12 **[**main**]** INFO o**.**s**.**web**.**context**.**ContextLoader **\*\*** Root WebApplicationContext**:** initialization completed in 1639 ms

15**:**12 **[**main**]** INFO o**.**s**.**s**.**c**.**ThreadPoolTaskExecutor **\*\*** Initializing ExecutorService 'applicationTaskExecutor'

15**:**12 **[**main**]** INFO o**.**a**.**coyote**.**http11**.**Http11NioProtocol **\*\*** Starting ProtocolHandler **[**"http-nio-8080"**]**

15**:**12 **[**main**]** INFO o**.**s**.**b**.**w**.**e**.**tomcat**.**TomcatWebServer **\*\*** Tomcat started on port**(**s**):** 8080 **(**http**)** with context path ''

15**:**12 **[**main**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** Started LogginApplication in 2.623 seconds **(**JVM running **for** 3.228**)**

15**:**12 **[**main**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** Quote**{**type**=**'success'**,** value**=**Value**{**id**=**12**,** quote**=**'@springboot with @springframework is pure productivity! Who said in #java one has to write double the code than in other langs? #newFavLib'**}}**

15**:**12 **[**main**]** ERROR com**.**example**.**loggin**.**LogginApplication **\*\*** error

15**:**12 **[**main**]** WARN com**.**example**.**loggin**.**LogginApplication **\*\*** warn

15**:**12 **[**main**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** info

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** INFO o**.**a**.**c**.**c**.**C**.[**Tomcat**].[**localhost**].[/] \*\* Initializing Spring DispatcherServlet 'dispatcherServlet'**

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** INFO o**.**s**.**web**.**servlet**.**DispatcherServlet **\*\*** Initializing Servlet 'dispatcherServlet'

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** INFO o**.**s**.**web**.**servlet**.**DispatcherServlet **\*\*** Completed initialization in 7 ms

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** SOMEONE HIT**!!!**

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** ERROR com**.**example**.**loggin**.**LogginApplication **\*\*** error

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** WARN com**.**example**.**loggin**.**LogginApplication **\*\*** warn

15**:**12 **[**http**-**nio**-**8080**-**exec**-**1**]** INFO com**.**example**.**loggin**.**LogginApplication **\*\*** info

===

**logback.xml**

*<?***xml version = "1.0" encoding = "UTF-8"***?>*<**configuration**>  
 <**appender name = "STDOUT" class = "ch.qos.logback.core.ConsoleAppender"**>  
 <**encoder**>  
 <**pattern**>%d{HH:mm} [%t] %-5level %logger{36} \*\* %msg%n</**pattern**>  
 </**encoder**>  
 </**appender**>  
 <**root level = "INFO"**>  
 <**appender-ref ref = "STDOUT"**/>  
 </**root**>  
</**configuration**>

===

**applition.properties**

#logging.level.root=info

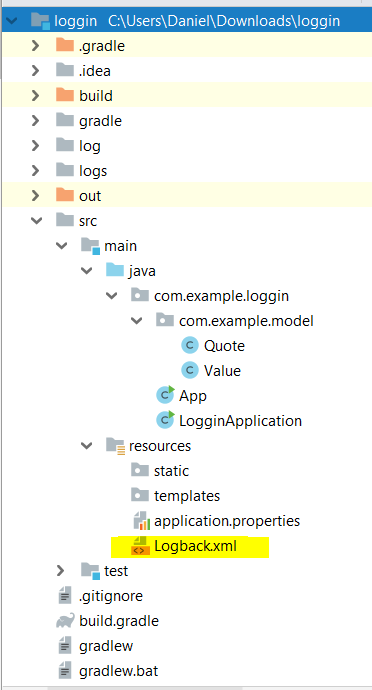
#logging.pattern.console=%d{HH:mm} [%t] %-5level %logger{36} \*\* %msg%n

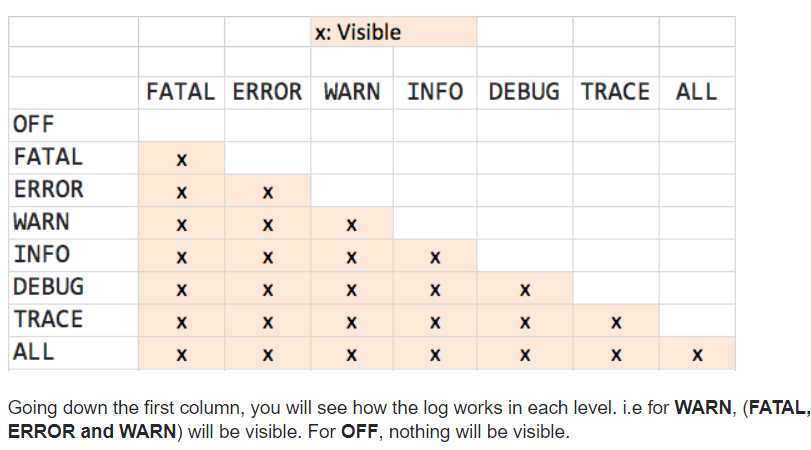
#logging.pattern.console.com.example.loggin.com.example.model.Quote=%d{HH:mm} [%t] %-5level %logger{36} \*\* %msg%nProtocol \*\* Initializing ProtocolHandler ["http-nio-8080"]

15:12 [main] INFO o.a.catalina.core.StandardService \*\* Starting service [Tomcat]

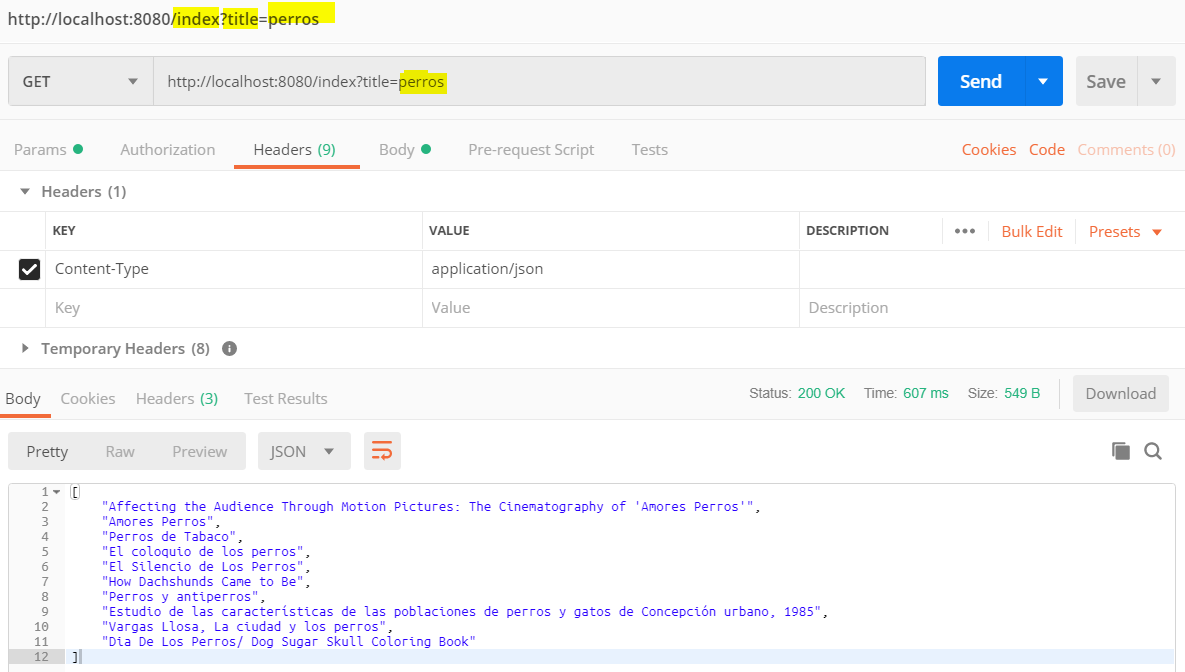
===

**build.gradle**

buildscript {  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath(**"org.springframework.boot:spring-boot-gradle-plugin:2.1.4.RELEASE"**)  
 }  
}  
  
*// Binary plugin  
/\*  
You apply plugins by their plugin id, which is a globally unique identifier, or name,   
for plugins. Core Gradle plugins are special in that they provide short names, such   
as 'java' for the core JavaPlugin  
\*/*plugins {  
 id **'java'**}  
  
*// Script plugin*apply **plugin**: **'java'**apply **plugin**: **'eclipse'**apply **plugin**: **'idea'**apply **plugin**: **'org.springframework.boot'**apply **plugin**: **'io.spring.dependency-management'**group **'com.example'**version **'1.0-SNAPSHOT'**sourceCompatibility = 1.8  
targetCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 compile(**"org.springframework.boot:spring-boot-starter"**)  
 compile(**"org.springframework:spring-web"**)  
 compile(**"com.fasterxml.jackson.core:jackson-databind"**)  
 testCompile(**"junit:junit"**)  
 testCompile(**"org.assertj:assertj-core:3.11.1"**)  
 *// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-test* testCompile **group**: **'org.springframework.boot'**, **name**: **'spring-boot-starter-test'**, **version**: **'2.1.4.RELEASE'** *// https://mvnrepository.com/artifact/org.slf4j/slf4j-simple* testCompile **group**: **'org.slf4j'**, **name**: **'slf4j-simple'**, **version**: **'1.6.1'**}  
  
*/\*  
Script plugins are additional build scripts that further configure the build and usually   
implement a declarative approach to manipulating the build. They are typically used within   
a build although they can be externalized and accessed from a remote location. Binary plugins   
are classes that implement the Plugin interface and adopt a programmatic approach to manipulating   
the build. Binary plugins can reside within a build script, within the project hierarchy or externally   
in a plugin jar.\*/*



**================================= CON NODES CUANDO NO CONOCEMOS LO QUE HAY ABAJO**



{

"kind": "books#volumes",

"totalItems": 479,

"**items**": [

{

"kind": "books#volume",

"id": "1TMLRbolbZ8C",

"etag": "r1CWxrGq+H4",

"selfLink": "https://www.googleapis.com/books/v1/volumes/1TMLRbolbZ8C",

"**volumeInfo**": {

"**title**": "Affecting the Audience Through Motion Pictures: The Cinematography of 'Amores Perros'",

"authors": [

"Susanne Schwarz"

],

"publisher": "GRIN Verlag",

"publishedDate": "2009-10",

"description": …

**=============================== SIN ARRAY**

{

"**type**":"success",

"**value**":{

"**id**":9,

"**quote**":"So easy it is to switch container in #springboot."

}

}

**import** com**.**fasterxml**.**jackson**.**databind**.**JsonNode**;**

**import** com**.**fasterxml**.**jackson**.**databind**.**node**.**ObjectNode**;**

**import** org**.**slf4j**.**Logger**;**

**import** org**.**slf4j**.**LoggerFactory**;**

**import** org**.**springframework**.**web**.**client**.**RestTemplate**;**

public class **App** **{**

public static final **Logger** logger **=** **LoggerFactory.getLogger(App.**class**);**

public static void **main(**String**[]** args**)** **{**

**RestTemplate** restTemplate **=** **new** RestTemplate**();**

String title **=** "Quotas de google!"**;**

logger**.**info**(**"Se buscan libros con el titulo de: {}"**,** title**);**

ObjectNode data **=** **restTemplate.**getForObject**(**"https://gturnquist-quoters.cfapps.io/api/**2**"**,** ObjectNode**.**class**);**

JsonNode value **=** data**.**get**(**"**value**"**);**

String id **=** value**.**get**(**"**id**"**).**asText**();**

String quote **=** value**.**get**(**"**quote**"**).**asText**();**

logger**.**info**(**id**+** " - " **+** quote**);**

**}**

**}**

…

20:02:17.972 [main] INFO com.example.ddroer.unRest.App - Se buscan libros con el titulo de: Quotas de google!

20:02:18.300 [main] DEBUG org.springframework.web.client.RestTemplate - **HTTP GET https://gturnquist-quoters.cfapps.io/api/2**

20:02:18.316 [main] DEBUG org.springframework.web.client.RestTemplate - Accept=[application/json, application/\*+json]

20:02:18.925 [main] DEBUG org.springframework.web.client.RestTemplate - Response 200 OK

20:02:18.925 [main] DEBUG org.springframework.web.client.RestTemplate - Reading to [com.fasterxml.jackson.databind.node.ObjectNode]

20:02:18.941 [main] INFO com.example.ddroer.unRest.App - **2** - **With Boot you deploy and find a JVM basically**.

**============================ SOLO CONTROLLER**

**import** com.fasterxml.jackson.databind.JsonNode;  
**import** com.fasterxml.jackson.databind.node.ArrayNode;  
**import** com.fasterxml.jackson.databind.node.ObjectNode;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RequestParam;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** org.springframework.web.client.RestTemplate;  
**import** java.util.ArrayList;  
**import** java.util.Collection;  
  
**@RestController  
public class Application** {  
  
 ***// Yo creo que cuando uno es feliz puede ser cualquier cosa***Logger **logger** = LoggerFactory.*getLogger*(Application.**class**);  
  
 **@RequestMapping("/index")** **public** Collection<String> getVolumnes(@RequestParam String title) {  
  
 RestTemplate **restTemplate** = **new** RestTemplate();  
  
 **logger**.info(**"Se buscan libros con el titulo de: {}"**, title);  
**ObjectNode** data = restTemplate.getForObject(**"https://www.googleapis.com/books/v1/volumes?q=intitle:"** + title,

**ObjectNode**.**class**);  
 Collection<String> bookTitles = **new** ArrayList<>();  
  
 ArrayNode items = (ArrayNode) data.get(**"items"**);  
  
 **for** (**int** i = 0; i < items.size(); i++) {  
 JsonNode jsonNode = items.get(i);  
 String s = jsonNode.get(**"volumeInfo"**).get(**"title"**).asText();  
 bookTitles.add(s);  
 **logger**.info(s);  
 }

**return bookTitles;** }  
}

…

21:21 [main] INFO o.a.coyote.http11.Http11NioProtocol \*\* Starting ProtocolHandler ["http-nio-8080"]

21:21 [main] INFO o.s.b.w.e.tomcat.TomcatWebServer \*\* Tomcat started on port(s): 8080 (http) with context path ''

21:21 [main] INFO com.example.loggin.LogginApplication \*\* Started LogginApplication in 2.508 seconds (JVM running for 3.108)

21:21 [http-nio-8080-exec-1] INFO o.a.c.c.C.[Tomcat].[localhost].[/] \*\* Initializing Spring DispatcherServlet 'dispatcherServlet'

21:21 [http-nio-8080-exec-1] INFO o.s.web.servlet.DispatcherServlet \*\* Initializing Servlet 'dispatcherServlet'

21:21 [http-nio-8080-exec-1] INFO o.s.web.servlet.DispatcherServlet \*\* Completed initialization in 8 ms

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Se buscan libros con el titulo de: perros

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Affecting the Audience Through Motion Pictures: The Cinematography of 'Amores Perros'

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Amores Perros

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Perros de Tabaco

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* El Silencio de Los Perros

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* How Dachshunds Came to Be

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Perros y antiperros

21:21 [http-nio-8080-exec-1] INFO c.e.loggin.controller.Application \*\* Vargas Llosa, La ciudad y los perros

**import** java**.**util**.**Stack**;**

public class **Parentheses** **{**

public static void **main(**String**[]** args**)** **{**

// Scanner sc = new Scanner(System.in);

String brackets **=** "{}}"**;** // sc.nextLine();

Stack**<**Character**>** stack **=** **new** Stack**<>();**//CREATING THE stack FOR THE BRACKETS

boolean isBalanced **=** **true;**//CHECKING WHETHER THE BRACKETS ARE BALANCED

**for** **(**int i **=** 0**;** i **<** brackets**.**length**();** i**++)** **{**//GETTING THROUGH ALL THE ELEMENTS FROM THE INPUT

char bracket **=** brackets**.**charAt**(**i**);**//GETTING ONE SINGLE BRACKET AND PUTTING IT INTO THE char bracket

**if** **(**bracket **==** '{' **||** bracket **==** '[' **||** bracket **==** '('**)** **{**//CHECKING WHETHER THE BRACKET TYPE IS THE OPENING ONE

stack**.**push**(**bracket**);**//ADDING THE OPENING BRACKET INTO THE stack

**}** **else** **{**

**if** **(**bracket **==** '}'**)** **{**

bracket **=** '{'**;**

**}** **else** **if** **(**bracket **==** ')'**)** **{**

bracket **=** '('**;**

**}** **else** **if** **(**bracket **==** ']'**)** **{**

bracket **=** '['**;**

**}**

**if** **(**stack**.**empty**())** **{**//CHECKING WHETHER THE stack IS EMPTY

isBalanced **=** **false;**

**break;**

**}**

**if** **(**bracket **==** stack**.**peek**())** **{**

stack**.**pop**();**

**}** **else** **{**

isBalanced **=** **false;**

**break;**

**}**

**}**

**}**

**if** **(**isBalanced**)** **{**

System**.**out**.**println**(**"YES"**);**

**}** **else** **{**

System**.**out**.**println**(**"NO"**);**

**}**

**}**

**}**

**…**

NO

SPRING BOOT

*<?***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 http://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**>2.1.5.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
 <**groupId**>com.pluralsight</**groupId**>  
 <**artifactId**>fundamentals</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>fundamentals</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
  
 </**dependencies**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
</**project**>

**Spring Boot Gradle Plugin**

The Spring Boot Gradle Plugin provides Spring Boot support in Gradle, letting you package executable jar or war archives, run Spring Boot applications, and use the dependency management provided by spring-boot-dependencies.

1. Overview

The Spring Boot Gradle plugin helps us manage Spring Boot dependencies, as well as package and run our application when using Gradle as a build tool.

In this tutorial, we’ll discuss how we can add and configure the plugin, and then we’ll see how to build and run a Spring Boot project.

2. Build File Configuration

First, we need to add the Spring Boot plugin to our build.gradle file by including it in our plugins section:

plugins {

id "org.springframework.boot" version "2.0.1.RELEASE"

}

If we’re using a Gradle version earlier than 2.1 or we need dynamic configuration, we can add it like this instead:

buildscript {

ext {

springBootVersion = '2.0.1.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath(

"org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

apply plugin: 'org.springframework.boot'

3. Packaging our Application

We can package our application to an executable archive (jar or war file) by building it using the build command:

./gradlew build

As a result, the generated executable archive will be placed in the build/libs directory.

If we want to generate an executable jar file, then we also need to apply the java plugin:

apply plugin: 'java'

On the other hand, if we need a war file, we’ll apply the war plugin:

apply plugin: 'war'

Building the application will generate executable archives for both Spring Boot 1.x and 2.x. However, for each version, Gradle triggers different tasks.

Next, let’s have a closer look at the build process for each Boot version.

3.1. Spring Boot 2.x

In Boot 2.x, the bootJar and bootWar tasks are responsible for packaging the application.

The bootJar task is responsible for creating the executable jar file. This is created automatically once the java plugin is applied.

Let’s see how we can execute the bootJar task directly:

./gradlew bootJar

Similarly, bootWar generates an executable war file and gets created once the war plugin is applied.

We can execute the bootWar task using:

./gradlew bootWar

Note that for Spring Boot 2.x, we need to use Gradle 4.0 or later.

We can also configure both tasks. For example, let’s set the main class by using the mainClassName property:

bootJar {

mainClassName = 'com.baeldung.Application'

}

Alternatively, we can use use the same property from the Spring Boot DSL:

springBoot {

mainClassName = 'com.baeldung.Application'

}

3.2. Spring Boot 1.x

With Spring Boot 1.x, bootRepackage is responsible for creating the executable archive (jar or war file depending on the configuration.

We can execute the bootRepackage task directly using:

./gradlew bootRepackage

Similar to the Boot 2.x version, we can add configurations to the bootRepackage task in our build.gradle:

bootRepackage {

mainClass = 'com.example.demo.Application'

}

We can also disable the bootRepackage task by setting the enabled option to false:

bootRepackage {

enabled = false

}

4. Running our Application

After building the application, we can just run it by using the java -jar command on the generated executable jar file:

java -jar build/libs/demo.jar

Spring Boot Gradle plugin also provides us with the bootRun task which enables us to run the application without the need to build it first:

./gradlew bootRun

The bootRun task can be simply configured in build.gradle.

For example, we can define the main class:

bootRun {

main = 'com.example.demo.Application'

}

5. Relation with Other Plugins

5.1. Dependency Management Plugin

For Spring Boot 1.x, it used to apply the dependency management plugin automatically. This would import the Spring Boot dependencies BOM and act similar to dependency management for Maven.

But since Spring Boot 2.x, we need to apply it explicitly in our build.gradle if we need this functionality:

apply plugin: 'io.spring.dependency-management'

5.2. Java Plugin

When we apply the java plugin, the Spring Boot Gradle plugin takes multiple actions like:

creating a bootJar task, which we can use to generate an executable jar file

creating a bootRun task, which we can use to run our application directly

disabling jar task

5.3. War Plugin

Similarly, when we apply the war plugin, that results in:

creating the bootWar task, which we can use to generate an executable war file

disabling the war task

=============================== UN MODELO DE PROJECTO SPRING BOOT MVC

0- ROOT

@SpringBootApplication  
**public class** FundamentalsApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.run(FundamentalsApplication.**class**, args);  
 }  
}

1- CONTROLLER

**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.PathVariable;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** org.springframework.web.server.ResponseStatusException;  
  
**import** java.util.List;  
  
@RestController  
@RequestMapping(**"/tza"**)  
**public class** TzaController {  
 **private** ApplicationService **applicationService**;  
 **private** TicketService **ticketService**;  
  
 @Autowired  
 **public void** setApplicationService(ApplicationService applicationService) { **this**.**applicationService** = applicationService; }  
  
 @Autowired  
 **public void** setTicketService(TicketService ticketService) {  
 **this**.**ticketService** = ticketService;  
 }  
  
 @GetMapping(**"/tickets"**)  
 **public** ResponseEntity<List<Ticket>> getAllTickets() {  
 List<Ticket> list = **ticketService**.listTickets();  
 **return new** ResponseEntity<List<Ticket>>(list, HttpStatus.OK);  
 }  
  
 @GetMapping(**"/applications"**)  
 **public** ResponseEntity<List<Application>> getAllApplications() {  
 List<Application> list = **applicationService**.listApplications();  
 **return new** ResponseEntity<List<Application>>(list, HttpStatus.OK);  
 }  
  
 @GetMapping(**"/application/{id}"**)  
 **public** ResponseEntity<Application> getApplication(@PathVariable(**"id"**) **long** id) {  
 **try** {  
 **return new** ResponseEntity<Application>(**applicationService**.findApplication(id), HttpStatus.OK);  
 } **catch** (ApplicationNotFoundException exception) {  
 **throw new** ResponseStatusException(HttpStatus.NOT\_FOUND, **"Application Not Found"**);  
 }  
 }  
}

2- SERVICE

**public interface** ApplicationService {  
 List<Application> listApplications();  
 Application findApplication(**long** id);  
}

…

**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.List;  
**import** java.util.Optional;  
  
@Service  
**public class** ApplicationServiceImpl **implements** ApplicationService {  
 @Autowired  
 **private** ApplicationRepository **applicationRepository**;  
  
 @Override  
 **public** List<Application> listApplications() {  
 **return** (List<Application>) **applicationRepository**.findAll();  
 }  
  
 @Override  
 **public** Application findApplication(**long** id) {  
 Optional<Application> optionalApplication = **applicationRepository**.findById(id);  
  
 **if**(optionalApplication.isPresent())  
 **return** optionalApplication.get();  
 **else  
 throw new** ApplicationNotFoundException(**"Application Not Found"**);  
 }  
}

3- REPOSITORY

**import** org.springframework.data.repository.CrudRepository;  
  
**public interface** ApplicationRepository **extends** CrudRepository<Application, Long> {  
}

4- MODEL

**import** javax.persistence.\*;  
  
@Entity  
**public class** Application {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name=**"application\_id"**)  
 **private** Long **id**;  
  
 @Column(name = **"app\_name"**, nullable = **false**)  
 **private** String **name**;  
  
 @Column(length = 2000)  
 **private** String **description**;  
 **private** String **owner**;  
  
 **public** Application() {  
 }  
  
 **public** Application(String name, String owner,  
 String description) {  
 **this**.**name** = name;  
 **this**.**owner** = owner;  
 **this**.**description** = description;  
 }

5- EXCEPTION

**public class** ApplicationNotFoundException **extends** RuntimeException {  
  
 **public** ApplicationNotFoundException(String exception) {  
 **super**(exception);  
 }  
}

6- application.properties

**logging.level.org.springframework**: **DEBUG***# H2***spring.h2.console.enabled**=**true  
spring.h2.console.path**=**/h2  
spring.datasource.url**=**jdbc:h2:mem:bugtracker**

RESPONSES (ResponseEntity.java)

