Spring Boot for Microservices

Lab Instructions

Your guide to completing the hand-on labs

Version 1.5.a

spring

Pivotal

Copyright Notice

Table of Contents

Chapter 1. [Optional] Spring Framework Lab

Review all the necessary steps required to create a simple web application with the **Spring Framework**.

Time: 25 minutes.

1.1. Requirements:

- Java 1.8 (recommended)
- Maven 3.x installed.
- Tomcat 8.5.x installed.

1.2. Simple Spring Web Application

1. To create a simple **Spring Web** application, open a terminal window and execute the following statement:

```
mvn archetype:generate -DgroupId=io.pivotal.workshop -DartifactId=simple-spring-webapp -Dversion=1.0-SNAPSHOT -DinteractiveMode=false -DarchetypeArtifactId=
```

> **Tip**
>
> If you have a **Maven** version below 3.x, you need to use the command **mvn archetype:create** instead.

2. Create the missing Java EE structure (ex: src/main/java, src/main/webapp/WEB-INF, etc.)
3. Create the web controller **SimpleController** class in **src/main/java/io/pivotal/workshop/web** folder.

   io.pivotal.workshop.web.SimpleController.java.

```
Unresolved directive in spring-framework.adoc - include::../../lab/spring-framework/simple-spring-webapp/src/main/java/io/pivotal/workshop/web/SimpleControl
```

4. Modify the **src/main/webapp/WEB-INF/web.xml** file.

   web.xml.

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >
```



```
<web-app>
  <display-name>Simple Spring Web Application</display-name>

  <servlet>
      <servlet-name>dispatcherServlet</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <servlet-name>dispatcherServlet</servlet-name>
      <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

5. Create the **src/main/webapp/WEB-INF/dispatcherServlet-servlet.xml** file.

   dispatcherServlet-servlet.xml.

```
Unresolved directive in spring-framework.adoc - include::../../lab/spring-framework/simple-spring-webapp/src/main/webapp/WEB-INF/dispatcherServlet-servlet.x
```

6. Modify the **src/main/webapp/index.jsp** file.

   index.jsp.

```
Unresolved directive in spring-framework.adoc - include::../../lab/spring-framework/simple-spring-webapp/src/main/webapp/index.jsp[]
```

7. Create the **src/main/webapp/WEB-INF/view/showMessage.jsp** file.

   showMessage.jsp.

```
Unresolved directive in spring-framework.adoc - include::../../lab/spring-framework/simple-spring-webapp/src/main/webapp/WEB-INF/view/showMessage.jsp[]
```

8. Modify the **pom.xml** file. Add the necessary dependencies.

   pom.xml.

```
Unresolved directive in spring-framework.adoc - include::../../lab/spring-framework/simple-spring-webapp/pom.xml[]
```

1.2.1. Building the Spring Web Application

To build the Spring Web application, execute the following command:

```
mvn clean package
```

the above command will generate the **target/simple-spring-webapp.war** .

1.2.2. Deploying the Spring Web Application

To deploy the application, install any application container that supports the Java Servlets specification.

For example, to deploy to **Apache Tomcat**, copy the **target/simple-spring-webapp.war** file into the **$TOMCAT-INSTALLATION/webapps/** directory and start the container. Then, go to your browser and hit the http://localhost:8080/simple-spring-webapp/ URL and click **Show** the message link.

You should see the text: **Simple Spring MVC Web App** .

Chapter 2. Spring Boot - Create a Simple Rest API

Get familiar with the Spring Initializr interface by creating a simple Spring Boot Web application.

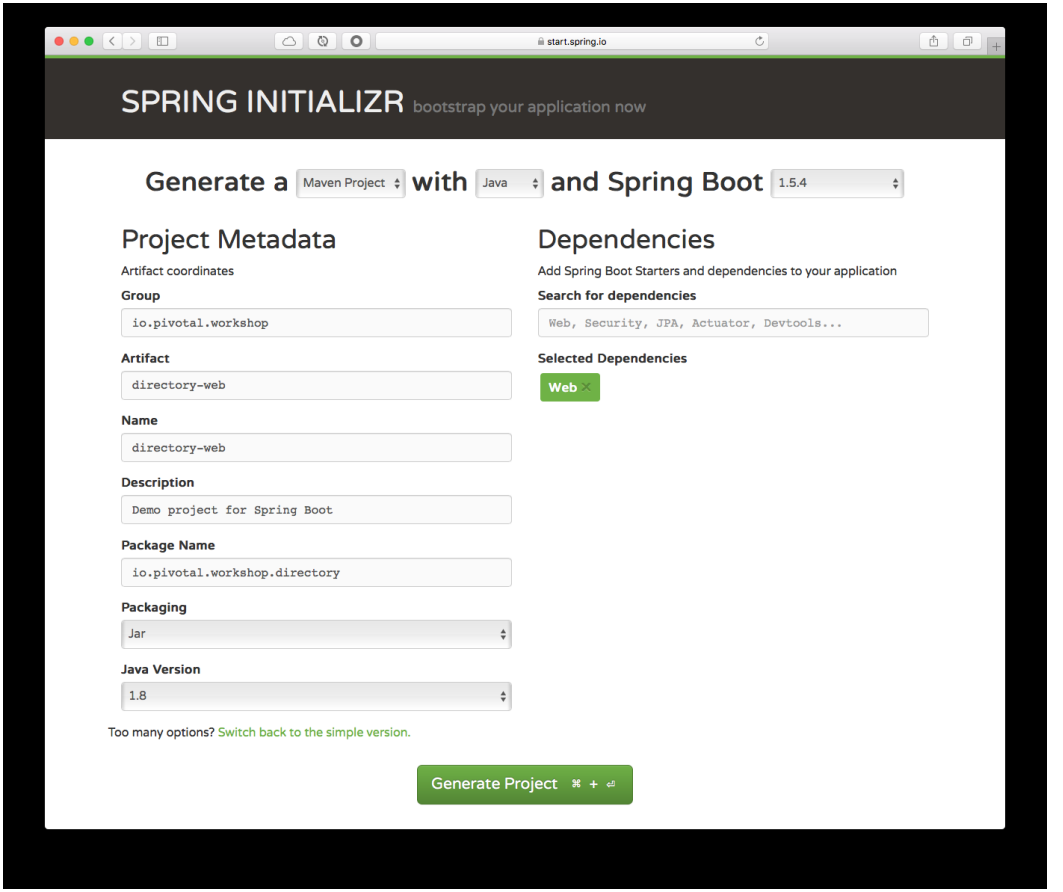
Time: 25 minutes.

2.1. Directory Web App

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the Directory Web App Project metadata with (See Figure 1.0):

Table 2.1. Directory Web App - metadata

Property	Value
Group:	*io.pivotal.workshop*
Artifact:	*directory-web*
Name:	*directory-web*
Package Name:	*io.pivotal.workshop.directory*
Dependencies:	*Web*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web** in the **Dependencies** field and press Enter.
5. Click the **Generate Project** button.
6. Unzip the file in any directory you want.
7. Import your project in any IDE you want.

8. Once imported, review the **pom.xml** file (or **build.gradle**).

maven - pom.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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>io.pivotal.workshop</groupId>
        <artifactId>directory-web-app</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>directory-web-app</name>
        <description>Demo project for Spring Boot</description>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.5.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
        </properties>

        <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>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>


</project>
```

Remember the **Spring Boot components**:

- **spring-boot-starter-parent**
- dependencies: **spring-boot-starter-web**
- plugin: **spring-boot-maven-plugin**

gradle - build.gradle.

```
buildscript {
        ext {
                springBootVersion = '1.5.4.RELEASE'
        }
        repositories {
```



```
                mavenCentral()
        }
        dependencies {
                classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
        }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'

version = '0.0.1-SNAPSHOT'
sourceCompatibility = 1.8

repositories {
        mavenCentral()
}


dependencies {
        compile('org.springframework.boot:spring-boot-starter-web')
        testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

9. Create the domain **io.pivotal.workshop.directory.domain.Person** class.

   io.pivotal.workshop.directory.domain.Person.java.

```
Unresolved directive in spring-boot-overview.adoc - include::../../lab/spring-boot-overview/directory-web/src/main/java/io/pivotal/workshop/directory/domain
```

10. Create the **io.pivotal.workshop.directory.repository.DirectoryRepository** class. This is you in-memory persistence.

    io.pivotal.workshop.directory.repository.DirectoryRepository.java.

```
package io.pivotal.workshop.directory.repository;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import io.pivotal.workshop.directory.domain.Person;

@Repository
public class DirectoryRepository {


        @SuppressWarnings("serial")
        private List<Person> directory = new ArrayList<Person>(){{
        add(new Person("john@email.com","John S","password","1985-11-10"));
                add(new Person("mike@email.com","Mike H","password","1984-12-02"));
                add(new Person("dan@email.com","Dan B","password","1983-03-07"));
                add(new Person("bill@email.com","Bill G","password","1983-06-12"));
                add(new Person("mark@email.com","Mark S","password","1986-02-22"));
        }};

        public Iterable<Person> findAll(){
                return this.directory;
        }

}
```

11. Create the **io.pivotal.workshop.directory.controller.DirectoryController** class. This will handle your REST API.

    io.pivotal.workshop.directory.controller.DirectoryController.java.


```
package io.pivotal.workshop.directory.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import io.pivotal.workshop.directory.domain.Person;
import io.pivotal.workshop.directory.repository.DirectoryRepository;

@RestController
public class DirectoryController {

        private DirectoryRepository repo;

        @Autowired
        public DirectoryController(DirectoryRepository repo){
                this.repo = repo;
        }

        @RequestMapping("/directory")
        public ResponseEntity<Iterable<Person>> findAll(){
                return ResponseEntity.ok(this.repo.findAll());
        }


}
```

12. Run your application and test the **/directory** endpoint by open a browser and hit: http://localhost:8080/directory

Tip

If you are not using any IDE, then you can run your application with maven: **./mvnw spring-boot:run** or if you are using gradle: **./gradlew bootRun**

2.2. Challenges

So far this is a very simple application, but still missing some of the HTTP request methods:

- Add a **index.html** page to be render as Home page (tip: **src/main/resources/static**).
- Add the **POST** method for adding a new person (tip: Use the @RequestBody annotation).
- Add the **PUT** method for updating a new person (tip: Use the @RequestBody annotation).
- Add the **DELETE** method for removing a person by id (tip: Use the @PathVariable annotation).
- Add a **GET** method for search by email (tip: Use the @RequestParam annotation).
- Add a **GET** method for finding by Id (tip: Use the @PathVariable annotation).

do any necessary changes to the classes.

2.2.1. [EXTRA - OPTIONAL] Challenges

- Mask the password when getting a directory JSON object.
- The **/directory** endpoint response as a JSON object and the dates are show as a long type. Modify it to get a formatted **'yyyy-MM-dd'** date.
- By default Spring Boot serialize automatically a JSON object, add a XML serialization too.

2.2.2. HOMEWORK

- Add validation to the Person object when doing POST or UPDATE.

Chapter 3. Spring Boot Internals

Get familiar with the Spring Boot Internals by using the **@Conditional** annotations.

Time: 35 minutes.

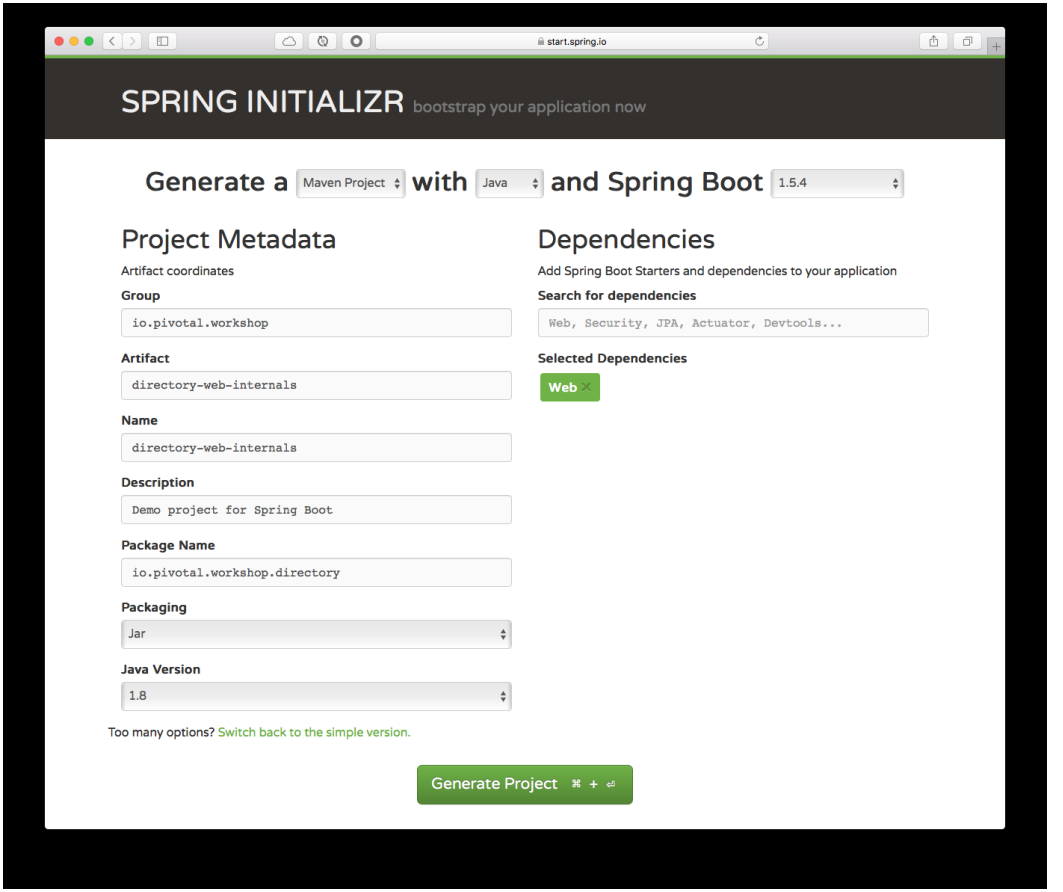
3.1. Directory Web Internals App

This application will use a custom **@Audit** annotation over any method and log the execution.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the Directory Web App Project metadata with (See Figure 1):

Table 3.1. Directory Web App - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*directory-web-internals*
Name:	*directory-web-internals*
Package Name:	*io.pivotal.workshop.directory*
Dependencies:	*Web*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web** in the **Dependencies** field and press Enter.
5. Click the **Generate Project** button.
6. Unzip the file in any directory you want.
7. Import your project in any IDE you want.

8. You can **copy all the code** from the previous lab (we are going to use it).

9. Add the following dependency to your **pom.xml** or **build.gradle**:

   maven - pom.xml.

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

   gradle - build.gradle.

```
compile('org.springframework.boot:spring-boot-starter-aop')
```

   we are going to use **AOP**, Aspect Oriented Programming.

10. Create the custom **@Audit** annotation by adding the **io.pivotal.workshop.directory.annotation.Audit** interface to the project.

```
package io.pivotal.workshop.directory.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Audit {
        Auditor value() default Auditor.NOTHING;
}
```

11. Create the **io.pivotal.workshop.directory.annotation.Auditor** enum, that is part of the **@Audit** annotation possible values.

```
package io.pivotal.workshop.directory.annotation;

public enum Auditor {
        BEFORE, AFTER, BEFORE_AND_AFTER, NOTHING
}
```

   the values in the **enum** above will be used later in the **challenges** section.

12. Create the **io.pivotal.workshop.directory.aop.SimpleAudit** class that will be used as **cross-cutting concern** for logging the execution of any method that has the **@Audit** annotation.

```
package io.pivotal.workshop.directory.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import io.pivotal.workshop.directory.annotation.Audit;

@Aspect
public class SimpleAudit {

        private static Logger log = LoggerFactory.getLogger("[AUDIT]");
```



```
        @Before("@annotation(audit)")
        public void audit(JoinPoint jp, Audit audit){
                log.info("[EXECUTING] " + jp.getSignature());
        }
}
```

13. In the **io.pivotal.workshop.directory.repository.DirectoryRepository** class add the **findByEmail** method (a challenge from the previous lab).

```
    @Audit
        public Optional<Person> findByEmail(String email){
                return findFirstBy( p -> p.getEmail().equals(email));
        }
```

as you can see, this method will have the **@Audit** annotation that will be log the execution of this method. If you missed the challenge from the previous lab, here it is, the **findFirstBy** code:

```
private Optional<Person> findFirstBy(Predicate<Person> findBy){
        return directory.stream()
                .filter(findBy)
                .findFirst();
}
```

14. Create a **io.pivotal.workshop.directory.config.DirectoryConfig** class that will create the aspect as Spring bean.

```
package io.pivotal.workshop.directory.config;

import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.pivotal.workshop.directory.aop.SimpleAudit;

@Configuration
public class DirectoryConfig {

        @ConditionalOnClass(name={"io.pivotal.workshop.directory.repository.PersonRepository"})
        @Bean
        public SimpleAudit simpleAudit(){
                return new SimpleAudit();
        }

}
```

is important to notice the usage of the **@ConditionalOnClass** annotation. Here the **simpleAudit** bean will be created **only** if the **PersonRepository** is on your classpath.

15. In the **io.pivotal.workshop.directory.controller.DirectoryController** class add the **searchByEmail** method (achallenge from the previous lab).

```
@RequestMapping("/directory/search")
public ResponseEntity<?> searchByEmail(@RequestParam String email) {
        Optional<Person> result = this.repo.findByEmail(email);

        if (result.isPresent()) {
                return ResponseEntity.ok(result.get());
        }

        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("{}");
}
```

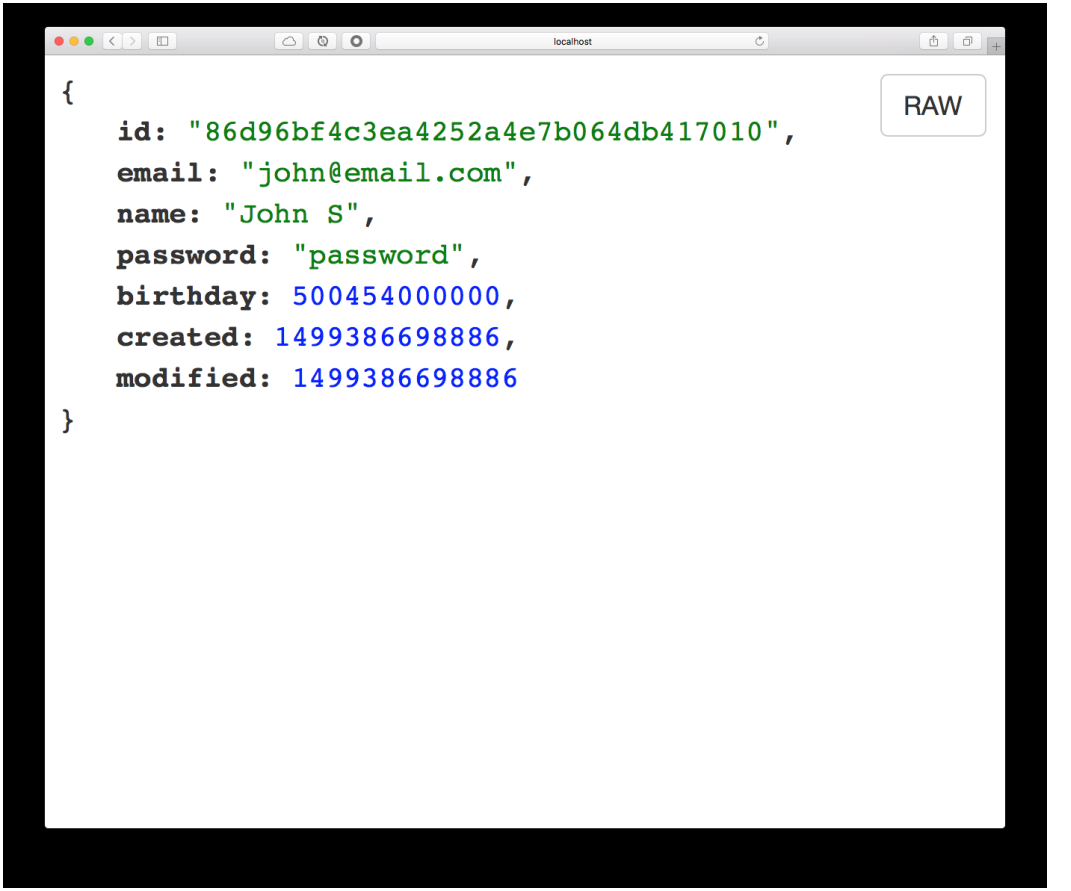

16. Run your application and test the **/directory/search** endpoint by open a browser and hit: **http://localhost:8080/directory/search?email=john@email.com**.

Tip

If you are not using any IDE, then you can run your application with maven: **./mvnw spring-boot:run** or if you are using gradle: **./gradlew bootRun**

17. You should see the result.

Figure 2: Directory Web Internals App - http://localhost:8080/directory/search?email=john@email.com.



3.2. Questions

Q: Did you see the logs in the console window (or terminal)?

A: NO, because the **@ConditionalOnClass** annotation is looking for a **PersonRepository** class.

Modify the **DirectoryConfig** class and change the right class in the @ConditionalOnClass annotation and run the application again:

```
@ConditionalOnClass(name={"io.pivotal.workshop.directory.repository.DirectoryRepository"})
```

now you should see:

```
[AUDIT] : [EXECUTING] Optional io.pivotal.workshop.directory.repository.DirectoryRepository.findByEmail(String)
```

3.3. Challenges

This is a trivial example using the **@ConditionalOnClass** annotation, so let's add more features to our project and have a real use of the **Auditor** enum:

- Create a new **@Around** advice for using the **Auditor** enum, so it can log also:
  - the parameters values if **@Audit(Auditor.BEFORE)**.
  - the return value if **@Audit(Auditor.AFTER)**.
  - the parameters and return values if **@Audit(Auditor.BEFORE_AND_AFTER)**.
- Use the **@ConditionalOnProperty** to evaluate if the **directory.audit** is **on** or **off**. If is **on** then log, but if is **off** or not present then don't do anything.
- Add the logic to the Aspect to review the **directory.info** if is with value **short** to log just the name of the method being executed, or with value **long** to show the full name of the method being executed.

3.3.1. [HOMEWORK]

This lab is just simple enough to get started, but everything is in the same code. The challenge is now to create a new **audit project** and use it in the **directory-web-internals** by creating **@EnableAudit** annotation.

Chapter 4. Spring Boot Features

Get familiar with the main Spring Boot features.

Time: 25 minutes.

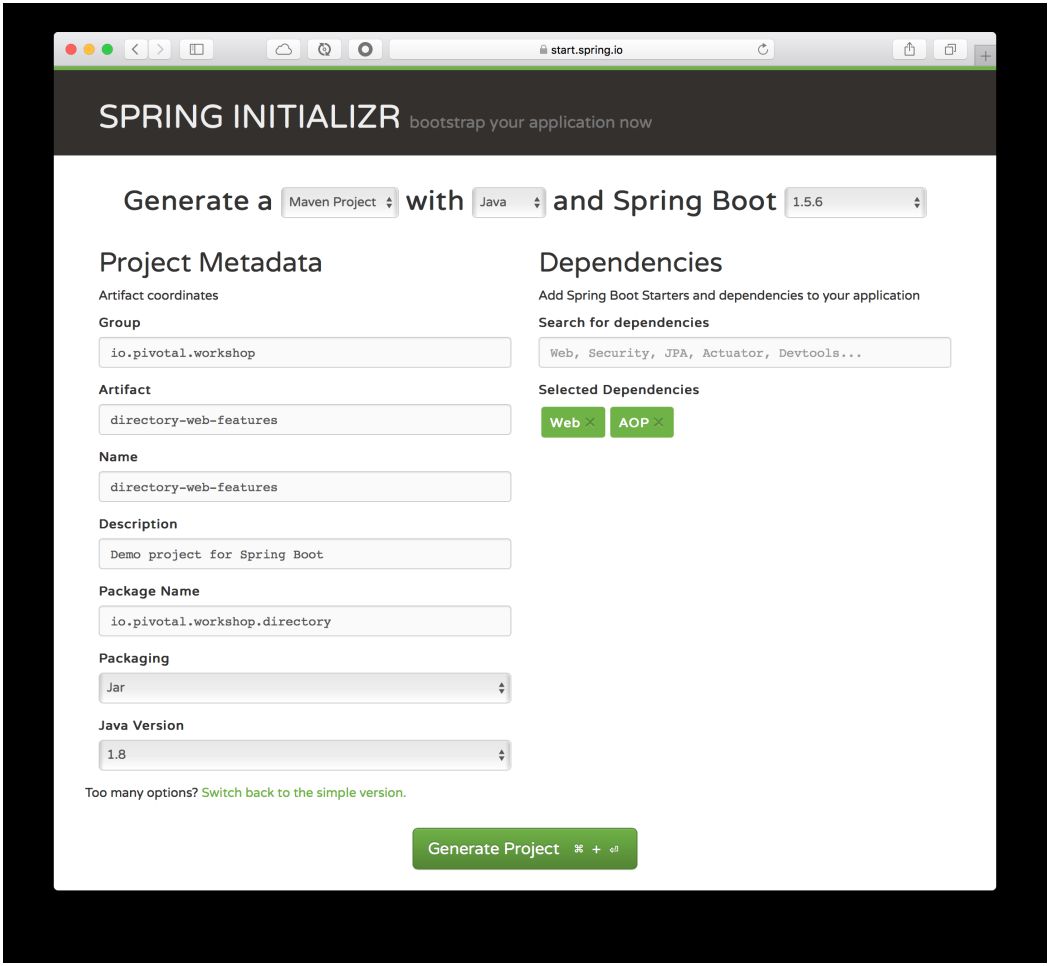
4.1. Directory Web Features

This lab will show you how to use some of the Spring Boot features. You are going to use the code from previous labs (**directory-web-internals** project)

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the Directory Web App Project metadata with (See Figure 1):

Table 4.1. Directory Web App - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*directory-web-features*
Name:	*directory-web-features*
Package Name:	*io.pivotal.workshop.directory*
Dependencies:	*Web, AOP*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web** and **AOP** in the **Dependencies** field and press Enter.
5. Click the **Generate Project** button.
6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. You can **copy all the code** from the previous lab (we are going to use it).

4.1.1. SpringBootApplication: Banner and Web Environment

Let's start with the **SpringBootApplication** class. In the main source code: **DirectoryWebFeaturesApplication** class, add the following code:

io.pivotal.workshop.directory.DirectoryWebFeaturesApplication.java.

```
package io.pivotal.workshop.directory;

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

@SpringBootApplication
public class DirectoryWebFeaturesApplication {

        public static void main(String[] args) {

                SpringApplication app = new SpringApplication(DirectoryWebFeaturesApplication.class);
                app.run(args);
        }
}
```

the **SpringApplication** will be use to turn on/off some features.

4.1.1.1. Challenges

- Add a **banner.txt** with some **ASCII ART** in the **src/main/resources** folder and run the application. You can choose some ascii art from: http://patorjk.com/software/taag
- Turn off the Banner using the **app** instance.
- Turn off the Web Environment using the **app** instance.
- You can turn on/off the Banner and Web Environment using the **application.properties**, find out how.

4.1.2. SpringBootApplication: CommandLineRunner / ApplicationRunner

In the main source code: **DirectoryWebFeaturesApplication** class, replace the class definition with the following code:

io.pivotal.workshop.directory.DirectoryWebFeaturesApplication.java.

```
package io.pivotal.workshop.directory;

import java.util.stream.Stream;

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.context.annotation.Bean;

@SpringBootApplication
public class DirectoryWebFeaturesApplication {

        public static void main(String[] args) {

                SpringApplication app = new SpringApplication(DirectoryWebFeaturesApplication.class);
                app.run(args);
        }

    private static final Logger log = LoggerFactory.getLogger("[ARGUMENTS]");

    @Bean
        public CommandLineRunner commandRunner() {
                return args -> {
                        Stream.of(args).forEach(s -> {
                                log.info("CommandLine: " + s);
                        });
                };
        }
}
```

the **CommandLineRunner** is use to execute code before your Spring Boot application start. The previous code will log all the arguments passed to the application.

Package and run the application with the following arguments: **--option=A,B,C --enable-audit=yes**

+ .create the JAR

```
./mvnw clean package -DskipTests=true
```

+ .run the application with some arguments

```
java -jar target/directory-web-features-0.0.1-SNAPSHOT.jar --option=A,B,C --enable-audit=yes
```

see the logs.

4.1.2.1. Challenges

- Use now the **ApplicationRunner** and get the **option** and **enable-audit** values, you should log something similar:

```
[ARGS]  : Option Name: enable-audit
[ARGS]  : Option Name: option
[ARGS]  : Found Value:[A,B,C]
[ARGS]  : Found Value:[yes]
[ARGS]  : Argument: --option=A,B,C
[ARGS]  : Argument: --enable-audit=yes
```

4.1.3. SpringBootApplication: External Configuration

If you didn't finish the previous lab (**directory-web-internals**) this is you chance! Create a

DirectoryProperties class that will hold the information about **audit** (**on/off**) and the **info** to be display (**long/short**) for the **AOP** audit.

Here you will use the **@ConfigurationProperties** annotation to enable a properties binding, you will use the **prefix directory** .

io.pivotal.workshop.directory.config.DirectoryProperties.java.

```
package io.pivotal.workshop.directory.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "directory")
public class DirectoryProperties {

        private String audit = "off";
        private String info = "long";

        public String getAudit() {
                return audit;
        }

        public void setAudit(String audit) {
                this.audit = audit;
        }

        public String getInfo() {
                return info;
        }

        public void setInfo(String info) {
                this.info = info;
        }

}
```

Modify the **application.properties** by adding the new properties:

src/main/resource/application.properties.

```
directory.audit=on
directory.info=short
```

Create the aspect **DirectoryAudit** and read the values to audit the method **findByEmail** from the **DirectoryRepository** class. (This is the challenge from the **directory-web-internals** lab).

io.pivotal.workshop.directory.aop.DirectoryAudit.java.

```
package io.pivotal.workshop.directory.aop;

import java.util.stream.IntStream;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import io.pivotal.workshop.directory.annotation.Audit;
import io.pivotal.workshop.directory.config.DirectoryProperties;

@Aspect
public class DirectoryAudit {

        private DirectoryProperties props;

        public DirectoryAudit(DirectoryProperties props){
                this.props = props;
        }
```



```
        private static Logger log = LoggerFactory.getLogger("[AUDIT]");

        @Around("execution(* *(..)) && @annotation(audit)")
        public Object audit(ProceedingJoinPoint jp, Audit audit) throws Throwable {
                Object[] args = jp.getArgs();

                this.printBar();
                this.print("[executing] " + (props.getInfo().toLowerCase().equals("short") ? jp.getSignature().getName() : jp.getSignature() ));

                switch (audit.value()) {
                case BEFORE:
                case BEFORE_AND_AFTER:
                        this.printArgs(args);
                default:
                        break;
                }

                Object obj = jp.proceed(args);

                switch (audit.value()) {
                case AFTER:
                case BEFORE_AND_AFTER:
                        this.print("[result] " + obj);
                default:
                        this.printBar();
                        break;
                }

                return obj;
        }

        private void print(Object obj) {
                log.info(obj.toString());
        }

        private void printArgs(Object[] args) {
                IntStream.range(0, args.length).forEach(idx -> {
                        log.info(String.format("[params] arg%d = %s", idx, args[idx]));
                });

        }

        private void printBar(){
                log.info("===========");
        }
}
```

In the **DirectoryRepository** add the **@Audit** annotation in the **findByEmail** method.

io.pivotal.workshop.directory.repository.DirectoryRepository.java - snippet.

```
@Audit
public Optional<Person> findByEmail(String email){
        return findFirstBy( p -> p.getEmail().equals(email));
}
```

Run you application and check out the logs.

Warning

The **on**/**off** feature only works if you did the challenge from the previous lab. You need to add the **@ConditionalOnProperty** to evaluate if the **directory.audit** is **on** or **off**.

4.1.3.1. Challenges

- Run you application by testing the **@Audit** annotation values: **Auditor.BEFORE**,**Auditor.AFTER**,**Auditor.BEFORE_AND_AFTER**
- Package your application and:
  - override the **audit** and **info** properties in the command line.
  - use environment variables to override the **audit** and **info** properties.
  - create a **application.yml** file in the current directory, add the **audit** and **info** properties and execute the program, what happen? did it worked?

Tip

Remember that, by adding an **application.properties** or **application.yml** to the current directory will override the values.

4.1.4. [HOMEWORK]

- Create profiles by:
  - Creating an **application-qa.properties** and **application-prod.properties**. Add the properties **audit** and **info** and test by enable the two profiles.
  - Creating an **application.yml** and add the profile section. Test by enable each profile.
- Instead of using the @Configuration properties, use **application arguments** to enable the **audit** and **info**.

Chapter 5. Spring Boot Web

Get familiar with the **Spring MVC** and the **Spring Boot Web applications**.

Time: 30 minutes.

5.1. Code Snippet Manager

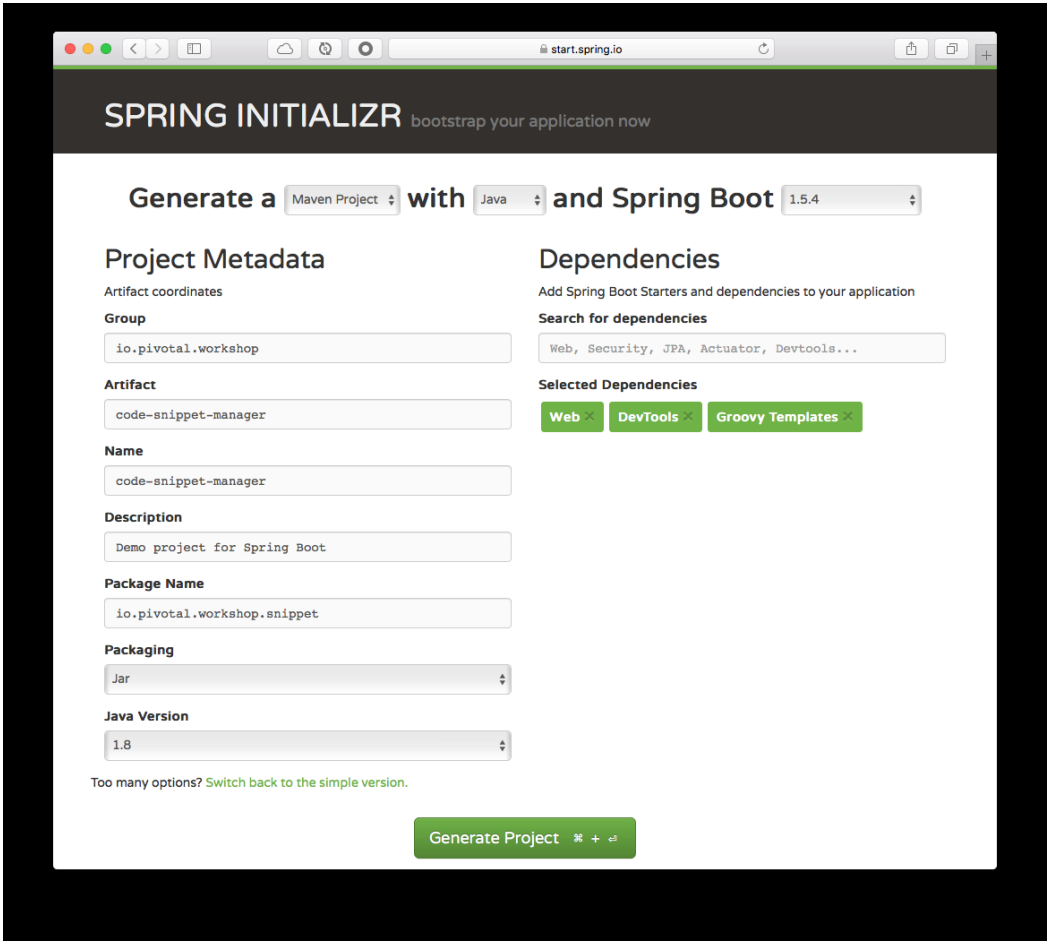
As developers normally we require to have some **code snippets** that help us to code fast and safe.

The main idea of this application is to create a site that can manage our **Code Snippets**. We are going to create a **RESTful API** for any external client.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager** Project metadata with (See Figure 1.0):

Table 5.1. Code Snippet Manager App - metadata

Property	Value
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager*
Name:	*code-snippet-manager*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **Groovy Templates**, in the **Dependencies** field and press Enter.
5. Click the **Generate Project** button.
6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. Let's start by defining the domain models. Create the following classes:

   io.pivotal.workshop.snippet.domain.Language.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/domain/Lang
```

   We can have multiple snippets in different programming languages, that's why we have this class. Take a look that also there is a **syntax** field; this field will be use later on for **syntax highlight** .

   io.pivotal.workshop.snippet.domain.Code.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/domain/Code
```

   This class will be use to hold the actual snippet code.

   io.pivotal.workshop.snippet.domain.Snippet.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/domain/Snip
```

   The above class will be the main reponse.

9. Next let's create a base interface that will be use as main **repository**. In this case we are going to hold all the data in **Memory** using **collections**:

   io.pivotal.workshop.snippet.repository.SimpleRepository.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/repository/
```

   as you can see it has just the most common actions.

10. Let's create now all the Repositories that will be implementing the **SimpleRepository** interface.

   io.pivotal.workshop.snippet.repository.LanguageRepository.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/repository/
```

   io.pivotal.workshop.snippet.repository.CodeRepository.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/repository/
```

   io.pivotal.workshop.snippet.repository.SnippetRepository.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/repository/
```


11.Next lets do the main controller that will expose the REST API:

io.pivotal.workshop.snippet.controller.SnippetController.java.

```
@RestController
public class SnippetController {


        private SnippetRepository snippetRepository;
        private LanguageRepository languageRepository;

        public SnippetController(SnippetRepository snippetRepository,LanguageRepository languageRepository){
                this.snippetRepository = snippetRepository;
                this.languageRepository = languageRepository;
        }

        @RequestMapping("/")
        public ModelAndView home(){
                assert snippetRepository != null;

                Map<String,Object> model = new HashMap<String,Object>();
                model.put("langs", languageRepository.findAll());
                model.put("snippets", snippetRepository.findAll());

                return new ModelAndView("views/home",model);
        }

        @RequestMapping("/snippets")
        public ResponseEntity<?> snippets(){
                assert snippetRepository != null;
                return ResponseEntity.ok(snippetRepository.findAll());
        }
}
```

12.Create a configuration class to initialize your repositories:

io.pivotal.workshop.snippet.config.SnippetConfiguration.java.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/src/main/java/io/pivotal/workshop/snippet/config/Snip
```

as you can see from the above code, we are building up our data.

If you take a close look, we are using the **Files.readAllBytes** helper class to read from a file, and this file is located in the **code/** folder.

13.Add some code snippets in the **code/** folder.

code/html-code.txt - HTML.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/code/html-code.txt[]
```

code/cs-code.txt - C#.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/code/cs-code.txt[]
```

code/pas-code.txt - Pascal.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/code/pas-code.txt[]
```

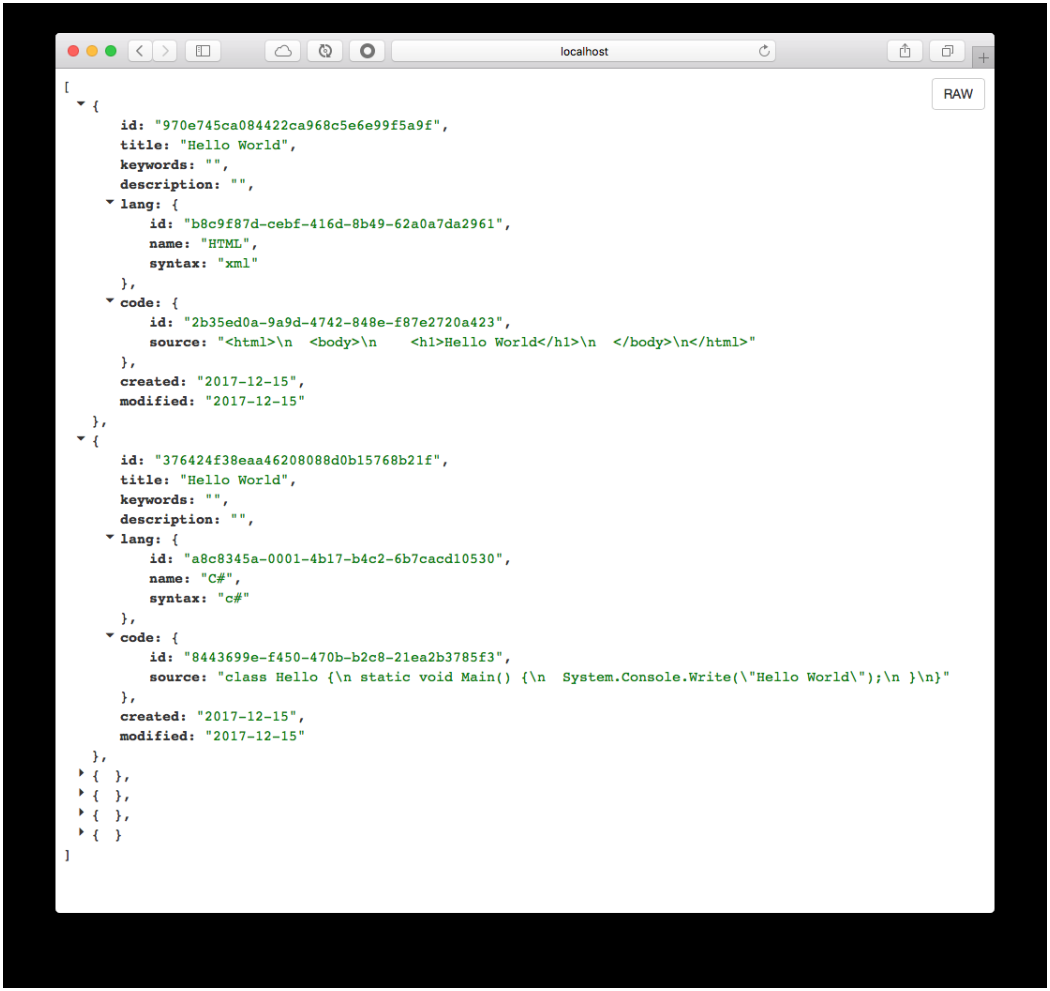

code/erl-code.txt - Erlang.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/code/erl-code.txt[]
```

code/js-code.txt - JavaScript.

```
Unresolved directive in spring-boot-web.adoc - include::../../lab/spring-boot-web/code-snippet-manager/code/js-code.txt[]
```

14. A lot of code, right? Well now its time to run the application. You can use your IDE or command line. Once running you application, go to your browser and hit: http://localhost:8080/snippets and you should get the a response like Figure 2.0.



5.2. Challenges

5.2.1. Adding a UI - Home Page

You can see that the ??? has the **home** () method, and it returns the view: **views/home** and the **model** , a map that contains the languages and the snippets: Add the following missing dependencies to your **pom.xml**

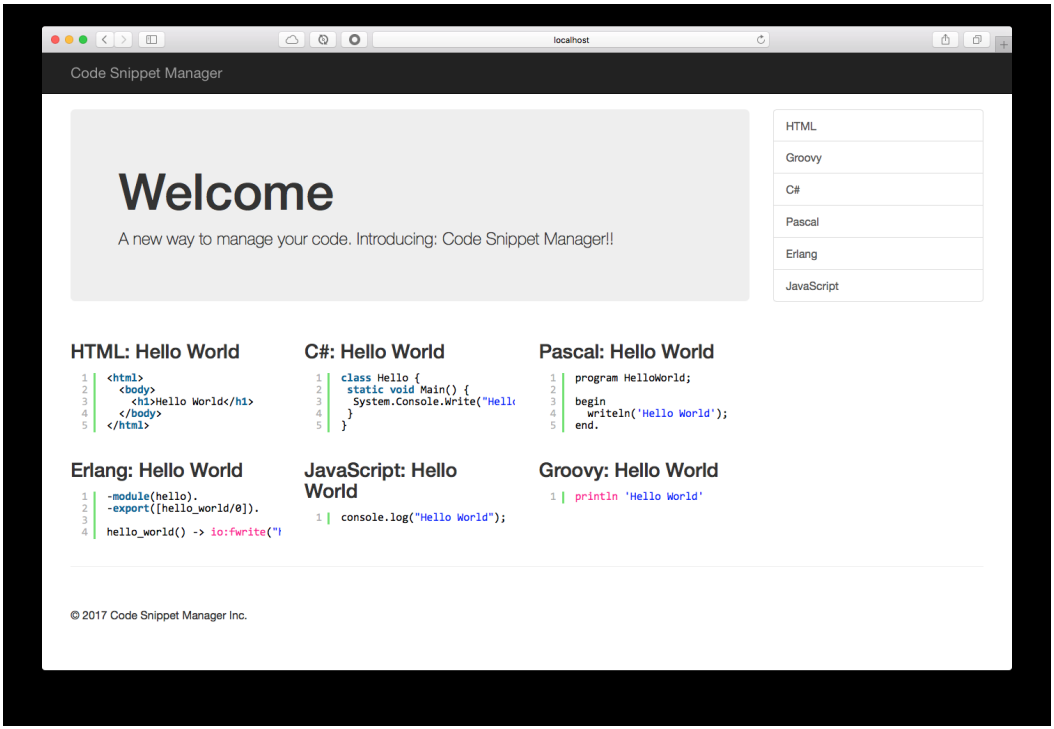
because you will need them:

```
<dependency>
    <groupId>org.webjars</groupId>
        <artifactId>jquery</artifactId>
        <version>2.2.4</version>
</dependency>
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>3.3.6</version>
</dependency>
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>angularjs</artifactId>
        <version>1.5.7</version>
</dependency>
```

Add the following libraries in the **src/main/resources/static/css**: theme.css and offcanvas.css and in the **src/main/resources/static/js** folder add the syntaxhighlighter.js script.

- Create the necessary layout and home page to have the same as Figure 3.0.

Figure 3.0: Snippet Code Manager - http://localhost:8080 .



5.2.2. REST API

- Complete the RESTful API:
  - provide the **/snippets/{id}** path endpoint to search by snippet Id.
  - provide methods to handle the: **POST**, **PUT** for adding a new snippet and updating the snippet.
- Make changes to repose either **JSON** or **XML**.
- When doing a XML response, the snippet code should be in a **CDATA** tag. Modify the code to show a **CDATA** for every source code. (**tip**: **@JacksonXmlCData**)

5.3. [HOMEWORK]

- Add validation to the **POST** and **PUT** methods (**tip**: *@Valid*).
- Finish up any missing **UI** or **Controller** methods.

Chapter 6. Spring Boot Data

Get familiar with the **Spring Data** and the **Spring Boot Data** features.

Time: 60 minutes (20 minutes per section).

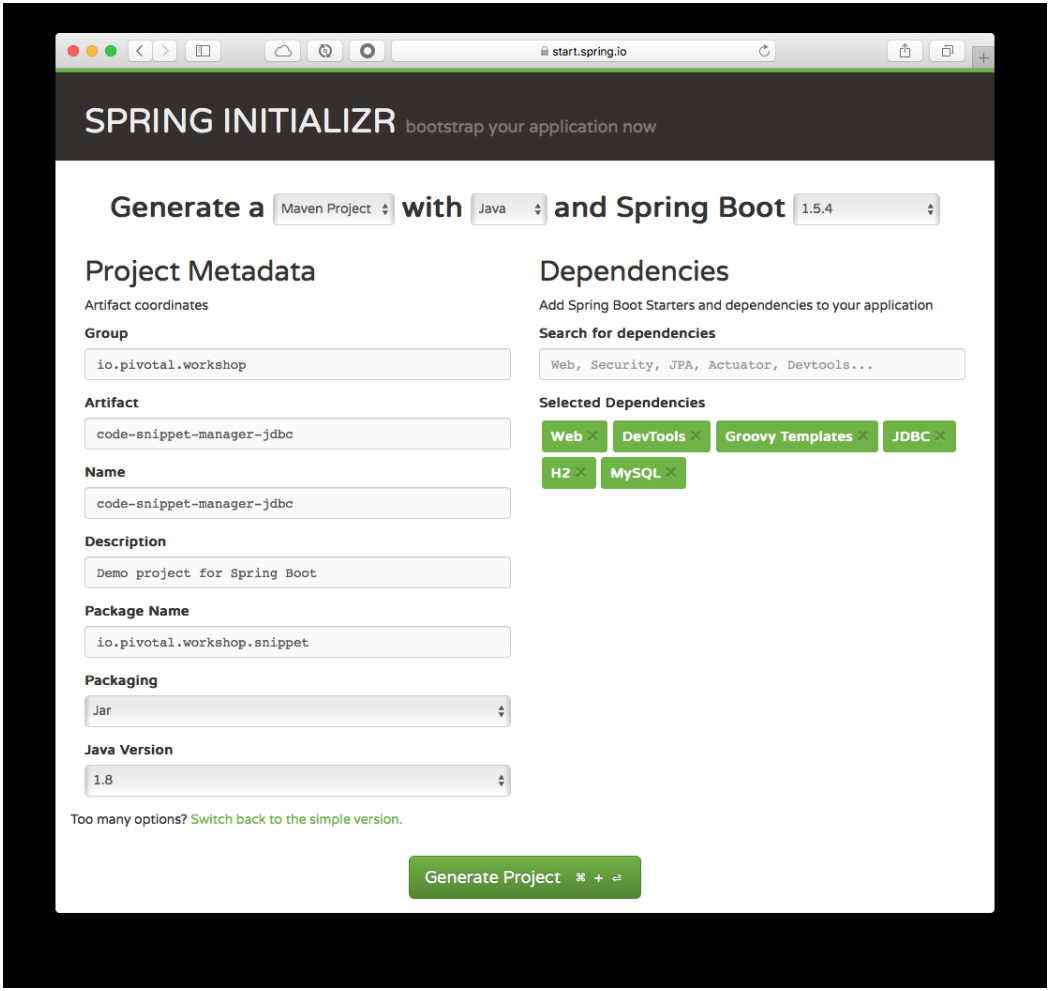
6.1. Code Snippet Manager JDBC

You will continue with the **Code Snippet Manager** code, but this time using a persistence engine. You will reuse the code from previous labs.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager JDBC** Project metadata with (See Figure 1.0):

Table 6.1. Code Snippet Manager JDBC App - metadata

Property	Value
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-jdbc*
Name:	*code-snippet-manager-jdbc*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates, JDBC, H2, MySQL*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **Groovy Templates**, **JDBC**, **H2**, and **MySQL** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.
7. Import your project in any IDE you want.
8. Copy all the packages (with code) into the new project.
9. The classes in **io.pivotal.workshop.snippet.domain**: **Code** , **Language** and **Snippet** should be the same.
10. Create a new class **CrossSnippetLanguageCode** in the **domain** package:

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/doma
```

this class will hold the relationship between the **Code** and **Language** within the **Snippet** class.

11. Because you will reuse the **SimpleRepository** interface, is necessary modify all the implementations. Modify the code accordingly:

io.pivotal.workshop.snippet.repository.LanguageRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

io.pivotal.workshop.snippet.repository.CodeRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

io.pivotal.workshop.snippet.repository.SnippetRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

Every single class is using the **JdbcTemplate** (with some methods like: **queryForObject**, **query** and **update**) and the **RowMapper** , analyze the code and review the **SQL** statements.

12. As you already guess from the code above, the **SnippetRepository** class has on its constructor the **CrossSnippetLanguageCodeRepository** reference. Create this class:

io.pivotal.workshop.snippet.repository.CrossSnippetLanguageCodeRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

Have you noticed that the **CrossSnippetLanguageCodeRepository** is using a declared **Java 8 lambda** notation for the **RowMapper**?

13. Talking about **RowMapper**, all the above code is using it to map the **ResultSet** with the domain class. Create the following mappers in the **io.pivotal.workshop.snippet.repository.mapper** package:

io.pivotal.workshop.snippet.repository.mapper.LanguageRowMapper.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

io.pivotal.workshop.snippet.repository.mapper.CodeRowMapper.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

io.pivotal.workshop.snippet.repository.mapper.SnippetRowMapper.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/java/io/pivotal/workshop/snippet/repo
```

14. The **SnippetController** and the **SnippetConfiguration** classes must be the same. Practically there is no change on any of them, they should work.

15. Next add the **schema.sql** in the **src/main/resources/** folder.

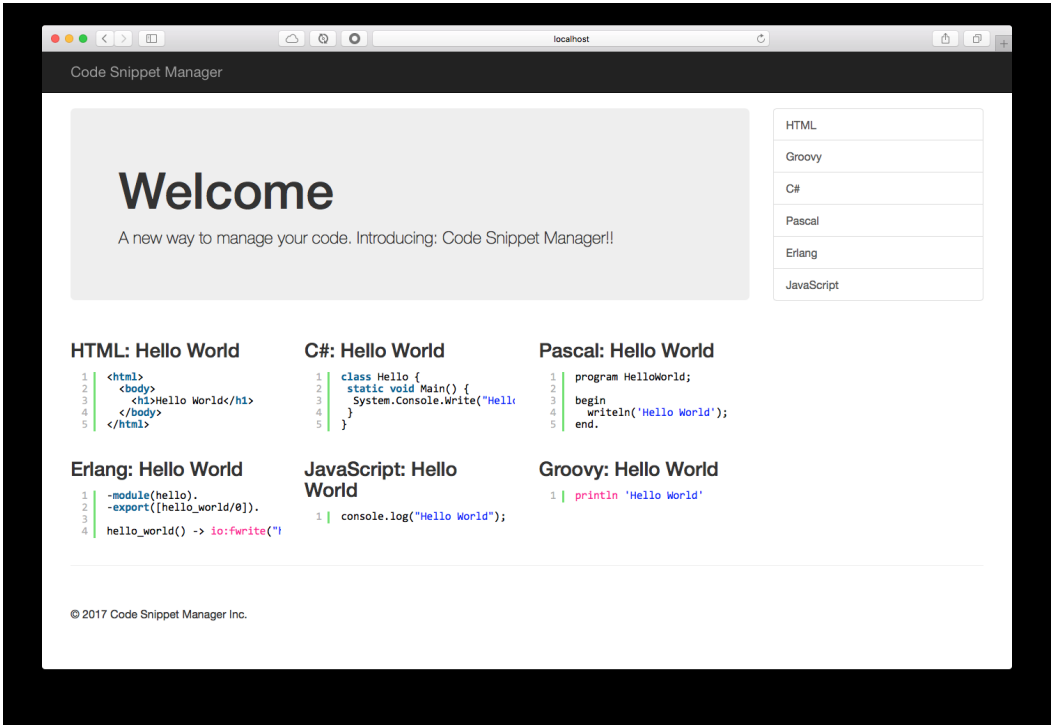
```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jdbc/src/main/resources/schema.sql[]
```

Tip

Before you run your application make sure to have copied **resources/static** and **resources/templates** files from the previous labs.

6.1.1. Challenges

- Run the application and make sure it works.
- Do any necessary change to have the home page working.



6.2. Code Snippet Manager JPA

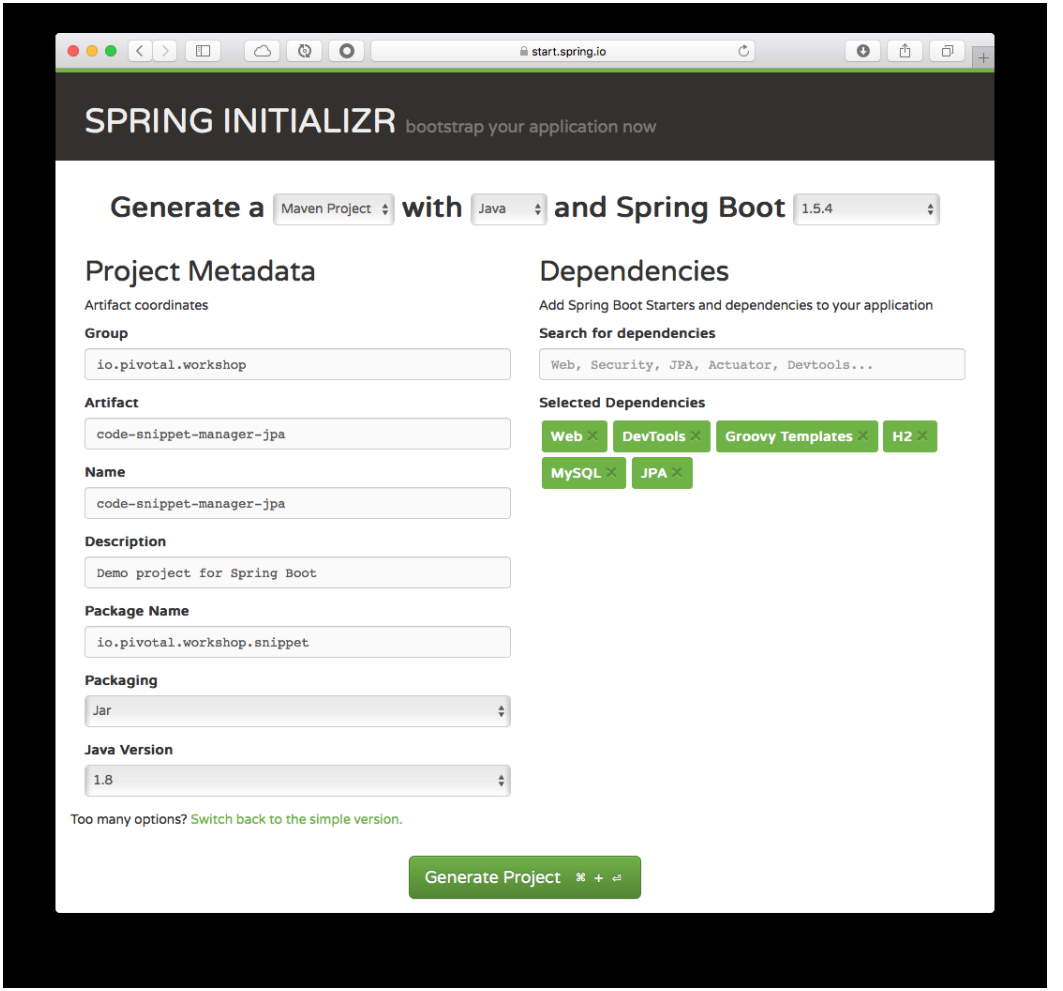
You will continue with the **Code Snippet Manager** code, but this time using **JPA**. You will reuse the code from previous labs.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager JPA** Project metadata with (See Figure 2.0):

Table 6.2. Code Snippet Manager JPA App - metadata

Property	Value

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-jpa*
Name:	*code-snippet-manager-jpa*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates, JPA, H2, MySQL*



Tip

You can choose either **Maven** or **Gradle** project types.

4. type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **H2**, and **MySQL** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. Copy all the packages (with code) into the new project.

9. The classes in **io.pivotal.workshop.snippet.domain**: **Code** , **Language** and **Snippet** will change because you will use the **JPA** features. Create/modify the following classes:

   io.pivotal.workshop.snippet.domain.Language.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/domai
```

   io.pivotal.workshop.snippet.domain.Code.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/domai
```

   io.pivotal.workshop.snippet.domain.Snippet.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/domai
```

   See that you are using now the **JPA** standard annotations: **@Entity**, **@Id**, **@OneToOne**, etc.

10. The repositories will change becasue you will use the power of the **spring-data** and **spring-data-jpa** projects. Create/Modify the following classes:

   io.pivotal.workshop.snippet.repository.LanguageRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/repos
```

   io.pivotal.workshop.snippet.repository.CodeRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/repos
```

   io.pivotal.workshop.snippet.repository.SnippetRepository.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa/src/main/java/io/pivotal/workshop/snippet/repos
```

   As you can see from the code above, you don't need to implement anything but just extend from the **CrudRepository<T,ID>** interface, which uses generics, meaning that you need to pass the domain class and the unique identifies, in this case what is marked as **@Id**. All the operation from the **CrudRepository<T,ID>** interface will be implemented by the **spring-data** classes.

11. The **SnippetController** and the **SnippetConfiguration** classes must be the same. Practically there is no change on any of them, they should work.

12. This time there is no need for any **SQL** schema, this time you will use the **DDL auto-creation** feature. In the **src/main/resources/application.properties** file add the following content:

src/main/resources/application.properties.

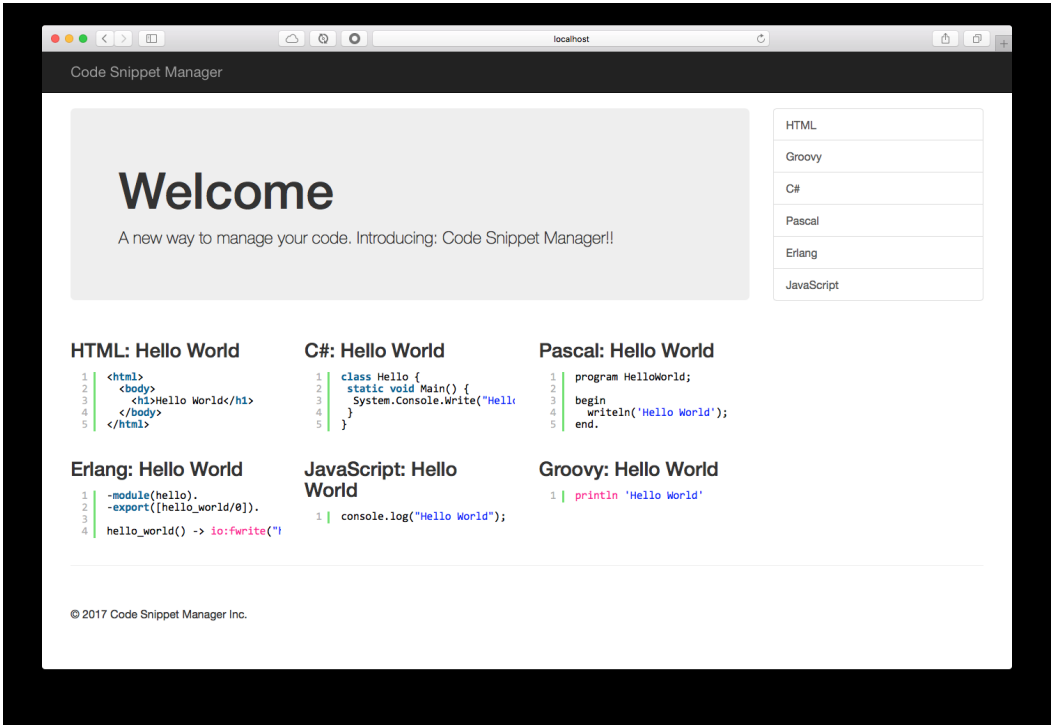
```
## JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
```

Tip

Before you run your application make sure to have copied **resources/static** and **resources/templates** files from the previous labs.

6.2.1. Challenges

- Run the application and make sure it works.
- Do any necessary change to have the home page working.



6.3. Code Snippet Manager JPA REST

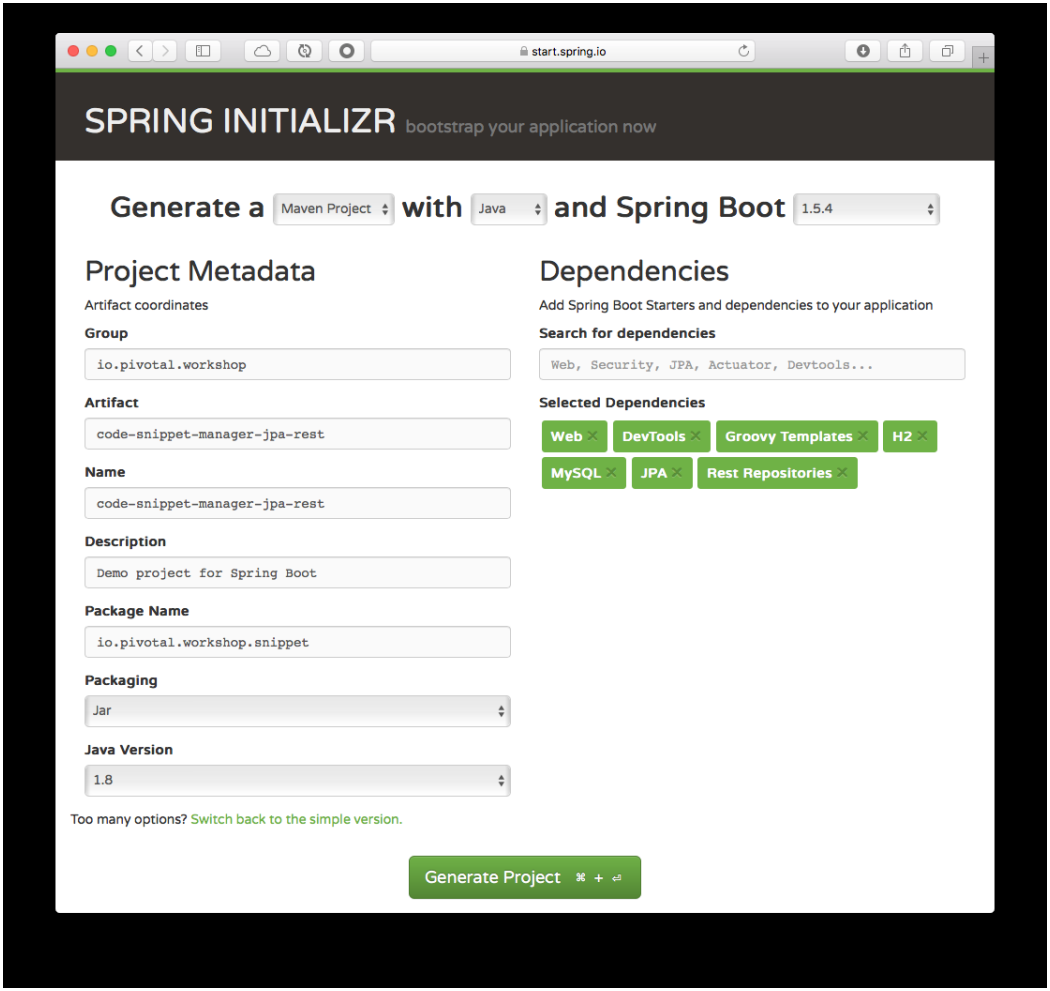
You will continue with the **Code Snippet Manager** code, but this time using **Data Rest** module. You will reuse the code from previous labs.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager JPA REST** Project metadata with (See Figure 3.0):

Table 6.3. Code Snippet Manager JPA REST App - metadata

Property	Value

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-jpa-rest*
Name:	*code-snippet-manager-jpa-rest*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates, JPA, Rest Repositories, H2, MySQL*



Tip

You can choose either **Maven** or **Gradle** project types.

4. type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **Rest Repositories**, **H2**, and **MySQL** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.
7. Import your project in any IDE you want.
8. Copy all the packages (with code) into the new project.
9. The only class that will change will be the **SnippetController**. Create/modify the folowing class:

   io.pivotal.workshop.snippet.controller.SnippetController.java.

```
Unresolved directive in spring-boot-data.adoc - include::../../lab/spring-boot-data/code-snippet-manager-jpa-rest/src/main/java/io/pivotal/workshop/snippet/
```

   As you can see, only is necessary the **home** method, and this is because the **spring-data-rest** module will take care of creating all the web controllers for accepting any request regarding to the domain repositories.

10. When using the **spring-data-rest** libraries, the default path for the controllers is the root: /, but in this case you are using the root as home page, that's why is necessary to override the **spring-data-rest** context path defaults, in the **src/main/resources/application.properties** file add the following:

   src/main/resources/application.properties.

```
## JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop

## REST
spring.data.rest.base-path=api
```

   Now you should be able to use the http://localhost:8080/api to access the REST repositories.

Tip

Before you run your application make sure to have copied **resources/static** and **resources/templates** files from the previous labs.

6.3.1. Challenges

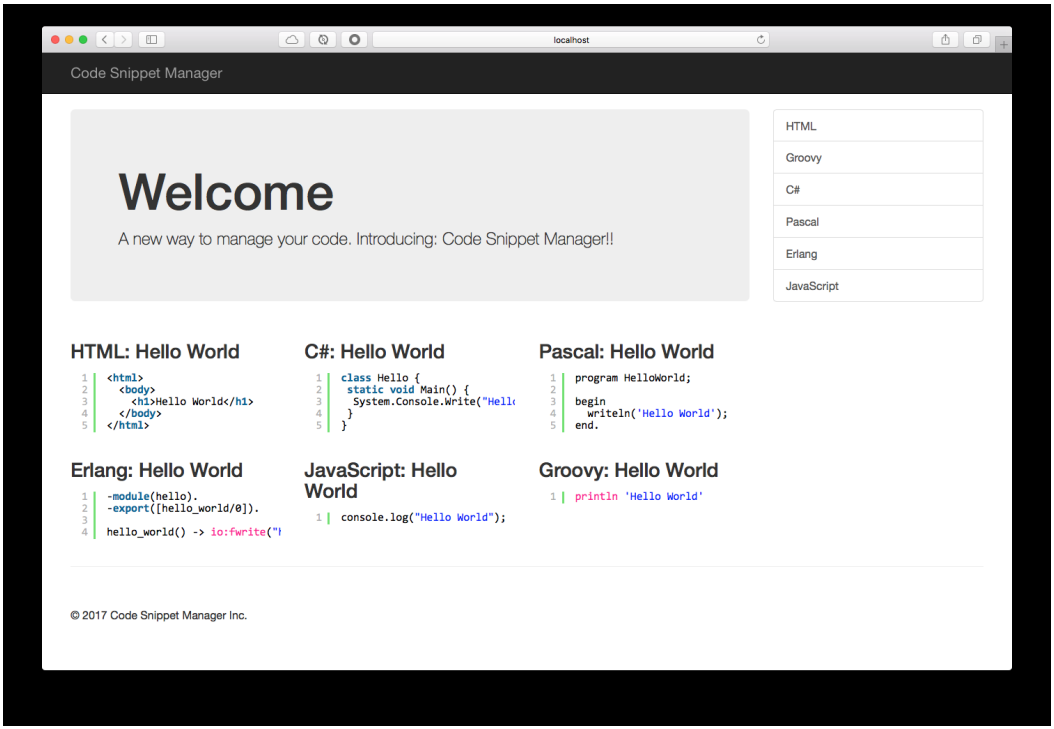
- Run the application and make sure it works.

Tip

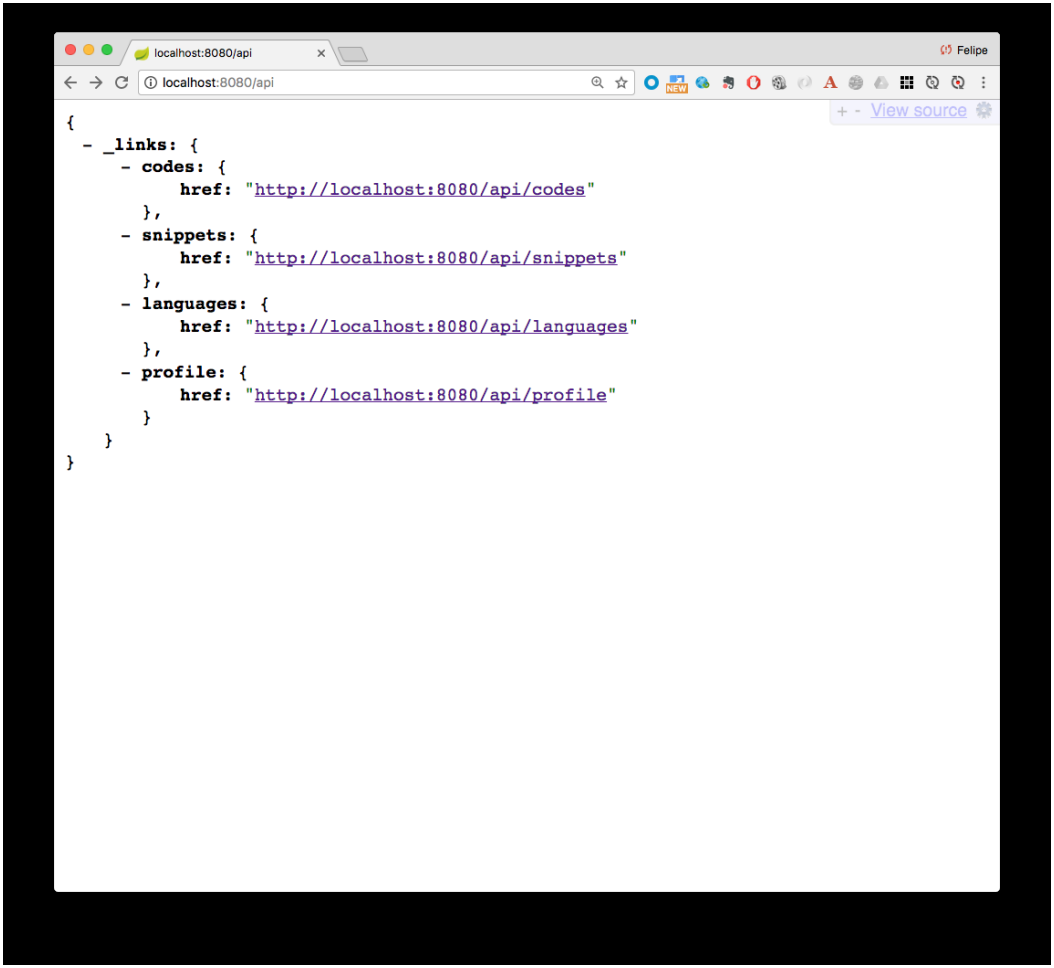
See the logs and take a look at all the mapping that is being generated.

- Do any necessary change to have the home page working.

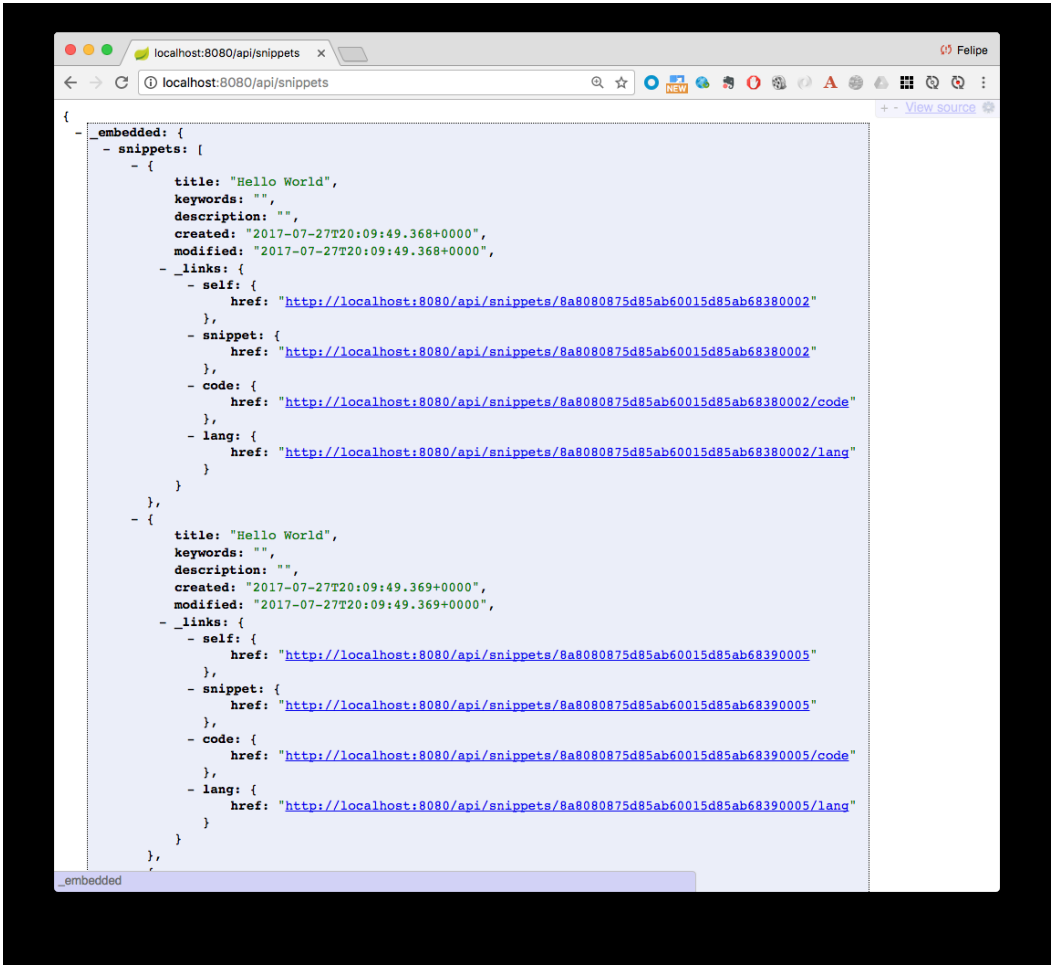
Figure 3.1: Code Snippet Manager JPA REST - http://localhost:8080/ .



- Go to the http://localhost:8080/api, you should see the following image:



and if you click the http://localhost:8080/api/snippets you should see the following image:



- Review all the links and see that the all the information is being translated into a **JSON/HAL**.
- So far we are using the **H2** engine, do the necessary changes to use now **MySQL**, how can you accomplish this?

6.4. HOMEWORK

- Use multiple databases like MySQL and MongoDB.

Chapter 7. Spring Boot Testing

Get familiar with the **Spring Testing** and the **Spring Boot Testing** features.

Time: 20 minutes.

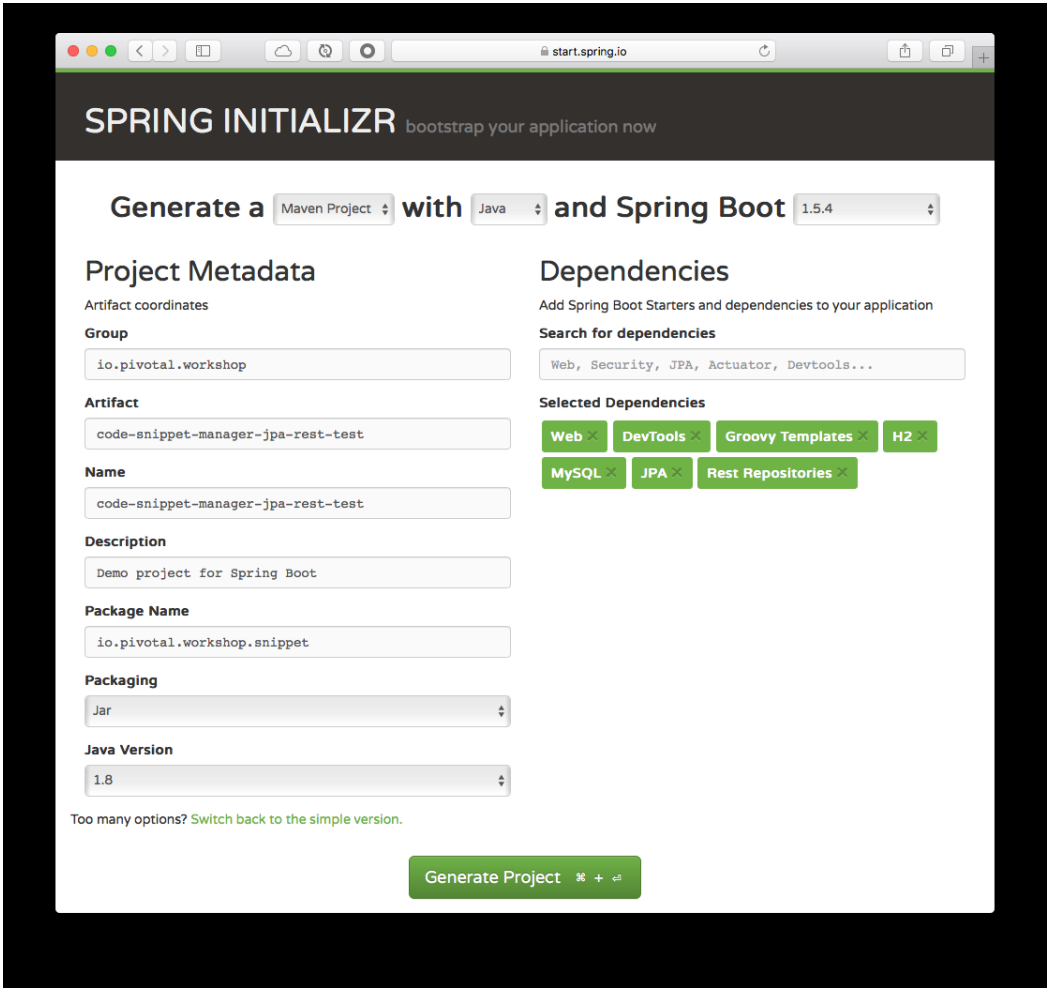
7.1. Code Snippet Manager JPA REST Testing

In this lab you will use the code from the **Code Snippet Manager JPA Rest** project. You will adding some tests.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager JPA REST Test** Project metadata with (See Figure 1.0):

Table 7.1. Code Snippet Manager JPA REST Test App - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-jpa-rest-test*
Name:	*code-snippet-manager-jpa-rest-test*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates, JPA, Rest Repositories, H2, MySQL*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **Rest Repositories**, **H2**, and **MySQL** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. Copy all the packages (with code) into the new project. None of the code will be changed.

9. In this lab you will use also a 3rd Party library called: Rest Assured. Add the following dependency to your **pom.xml** or **build.gradle**

   pom.xml.

```
<dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>spring-mock-mvc</artifactId>
        <version>3.0.0</version>
        <scope>test</scope>
</dependency>
```

   build.gradle.

```
testCompile('io.rest-assured:spring-mock-mvc:3.0.0')
```

10. Add in the **src/test/java** folder in the package **io.pivotal.workshop.snippet** the following integration test:

   io.pivotal.workshop.snippet.IntegrationTest.java.

```
package io.pivotal.workshop.snippet;

import static io.restassured.module.mockmvc.RestAssuredMockMvc.when;
import static org.assertj.core.api.Assertions.assertThat;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.Collection;
import java.util.Collections;
import java.util.stream.StreamSupport;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.hateoas.Resource;
import org.springframework.hateoas.Resources;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import io.pivotal.workshop.snippet.domain.Snippet;
import io.restassured.module.mockmvc.RestAssuredMockMvc;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class IntegrationTest {

        @Autowired
        private TestRestTemplate restTemplate;

        @Autowired
        private WebApplicationContext context;

        @Before
```



```
        public void setUp() {
                RestAssuredMockMvc.mockMvc(MockMvcBuilders.webAppContextSetup(context).build());
        }

        @Test
        public void homePageTest() {
                String body = this.restTemplate.getForObject("/", String.class);
                assertThat(body).contains("Hello World");
        }

        @Test
        public void linksTests() {
                when().get("/api/snippets").then().assertThat(status().isOk()).body("_links.self.href",
                                equalTo("http://localhost/api/snippets"));
        }
}
```

Take a look at the code and see that is exposing the **RestAssuredMockMvc** and also the usage of the **TestRestTemplate**.

7.1.1. Challenges: Integration Tests

- Analyze the code and run the test, either using your IDE, maven or gradle.
- Add a new test method **restControllerTest** that asserts the **HAL/JSON** response.

Tip

Use the **ResponseEntity<Resources<Resource<Snippet>>>** instance and the **restTemplate.exchange** method.

7.1.2. Challenges: Slices

- Add a new **JsonTest** class and use the **@JsonTest** annotation. You will use the **JacksonTester<T>** class and the **assertj** library to do the assertions.
- Add a new **JpaTests** class and use the **@DataJpaTest** annotation. You will use the **TestEntityManager** class and the **hamcrest** library to do the assertions.

Chapter 8. Spring Boot Actuator

Get familiar with the **Spring Boot Actuator** and its features.

Time: 35 minutes.

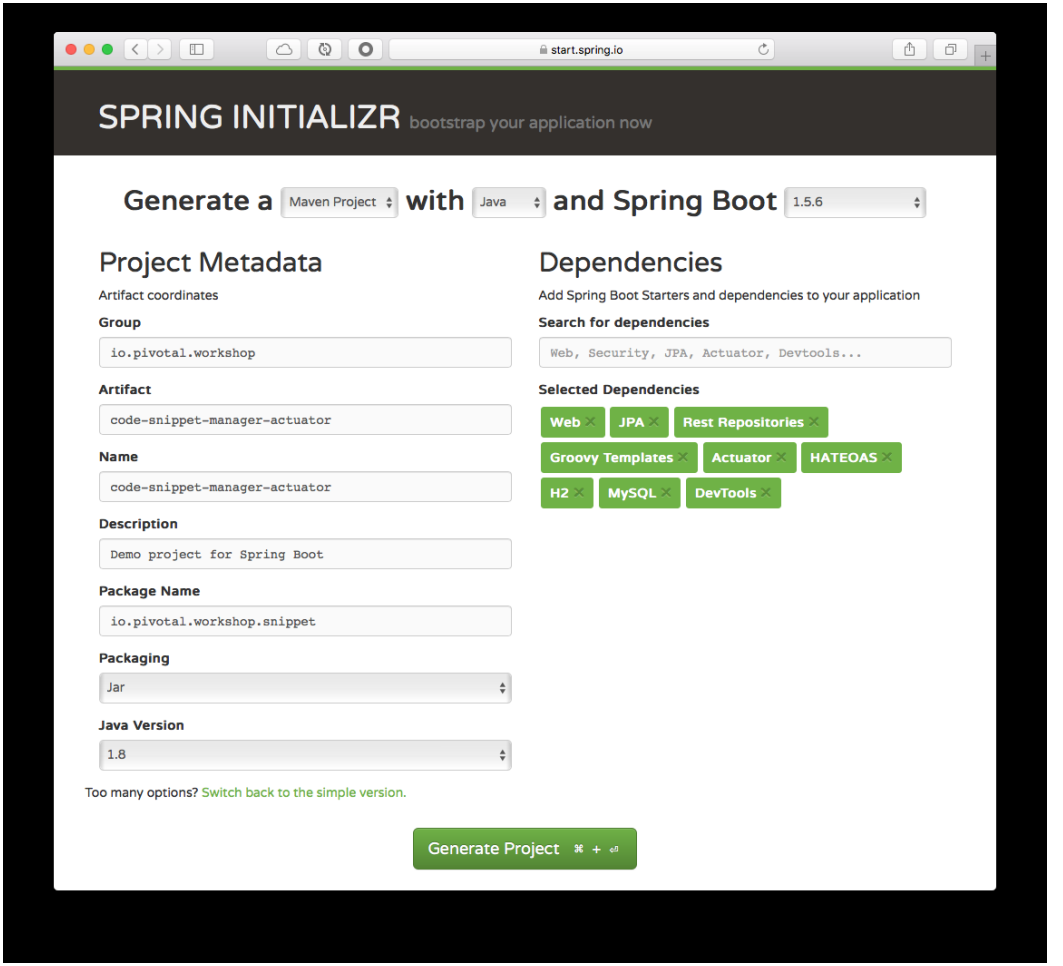
8.1. Code Snippet Manager Actuator

You will continue with the **Code Snippet Manager** code. For this lab you will need the **snippet-code-manager-jpa-rest** project code. You will reuse the code from previous labs.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Code Snippet Manager Actuator** Project metadata with (See Figure 1.0):

Table 8.1. Code Snippet Manager Actuator - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-actuator*
Name:	*code-snippet-manager-actuator*
Package Name:	*io.pivotal.workshop.snippet*
Dependencies:	*Web, DevTools, Groovy Templates, JPA, Rest Repositories, H2, MySQL, Actuator, HATEOAS*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **Rest Repositories**, **H2**, **MySQL**, **Actuator**, **HATEOAS** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.
7. Import your project in any IDE you want.
8. Copy all the packages (with code) into the new project.
9. The packages: **domain**, **repository** and **config** will remain with no change.

8.1.1. Spring Boot Actuator: Metrics

1. In the **SnippetController** add a **CounterService** so we can have a statistic of the number of visits to the home page.

   io.pivotal.workshop.snippet.controller.SnippetController.java.

```
Unresolved directive in spring-boot-actuator.adoc - include::../../lab/spring-boot-actuator/code-snippet-manager-actuator/src/main/java/io/pivotal/workshop/
```

   Take a look that the **CounterService** is being used and the key is: **homepage** .

2. Remember that the **spring-boot-actuator** module since spring boot 1.5 is secured? Also, remember that the actuator uses as default the root for the endpoint, so let's override that as well. Create/modify the **src/main/resources/application.properties** file to look like this:

   src/main/resources/application.properties.

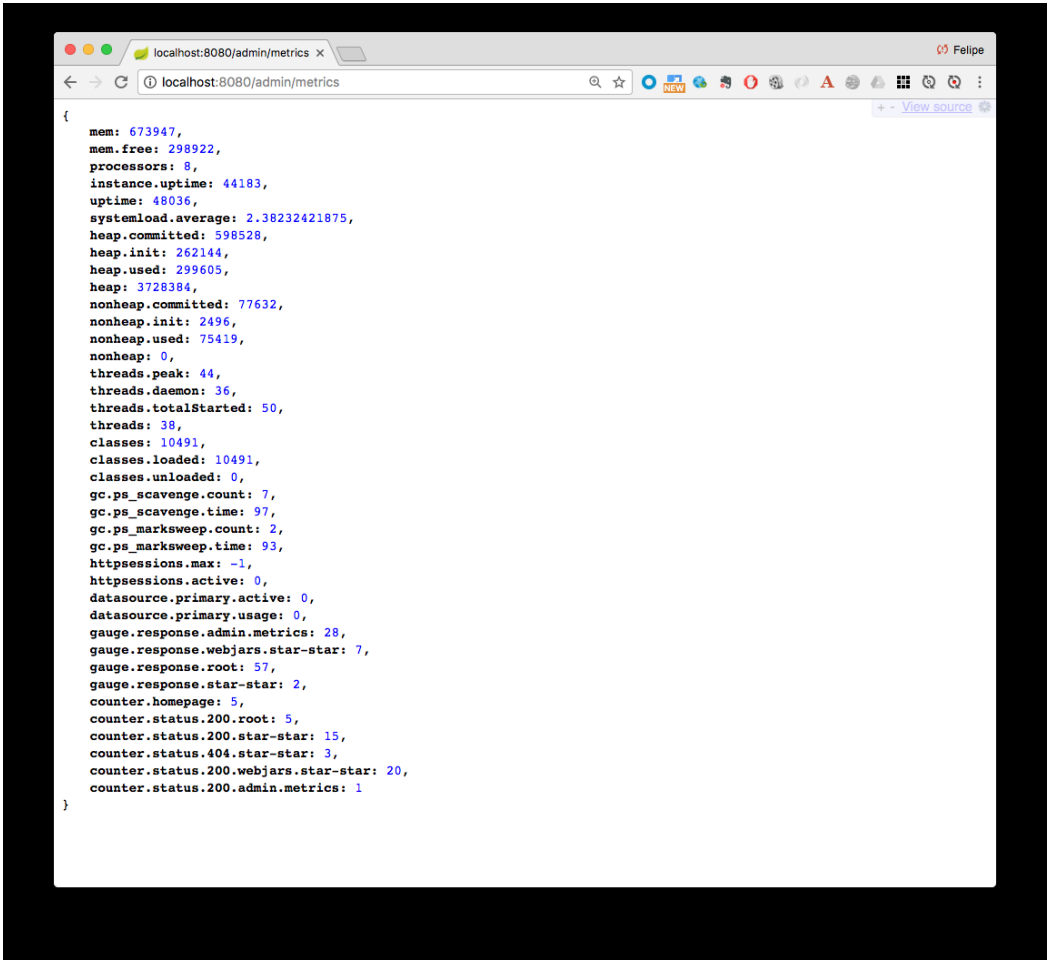
```
## JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop

## REST
spring.data.rest.base-path=api

## ACTUATOR
management.context-path=/admin
management.security.enabled=false
```

   See that we are using the **management.security.enabled=false** to disable the security and we are changing the default context path, so all the actuator endpoints will live in: **/admin**.

3. Run the application, and refresh several times the homepage: http://localhost:8080
4. Go to http://localhost:8080/admin/metrics and you should see the **counter.homepage** key displayed. See the



8.1.2. Spring Boot Actuator: HealthIndicator

Using the same code, imagine that we are going to use a FileSystem to save some information. A specific path, by using the **snippet.path** property. We need to know:

- If the path doesn't exists, report **out of service** .
- If the path exists but doesn't have writing permissions, report **system down**.

- If the path exists and it has write permissions, report **up**.

1 Let's create a properties holder class. Add the **SnippetProperties** class in the **io.pivotal.workshop.snippet.config** package:

io.pivotal.workshop.snippet.config.SnippetProperties.java.

```
Unresolved directive in spring-boot-actuator.adoc - include::../../lab/spring-boot-actuator/code-snippet-manager-actuator/src/main/java/io/pivotal/workshop/
```

The above code will hold the information about the path with the property: **snippet.path**.

2 Create the **SnippetHealthCheck** class in the **io.pivotal.workshop.snippet.actuator** package:

io.pivotal.workshop.snippet.actuator.SnippetHealthCheck.java.

```
Unresolved directive in spring-boot-actuator.adoc - include::../../lab/spring-boot-actuator/code-snippet-manager-actuator/src/main/java/io/pivotal/workshop/
```

Take a look at the constructor. If the **snippet.path** is not found, it will use the **/tmp** as default path.

3 In the **src/main/resources/application.properties** file add the following property:

```
## Snippet
snippet.path=/tmp/snippets
```

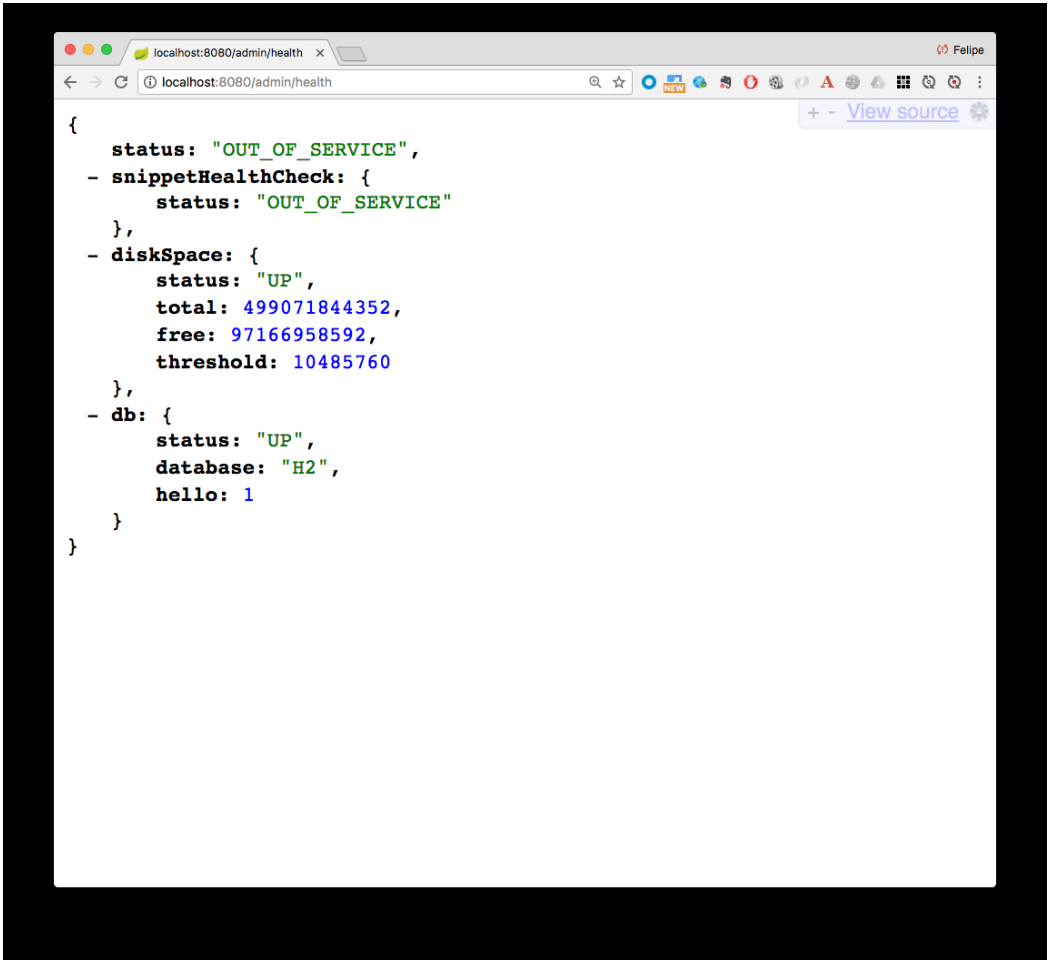
Tip

Do not create this folder yet, we will make sure our custom health indicator works.

Tip

If you are using Windows try to use the **C:/tmp/snippets** or **C:\\tmp\\snippets** format.

4 If you run the application and go to the http://localhost:8080/admin/health, you should have something like the following Figure 3.0:



Here you can see the **SnippetHealthCheck** value: **OUT_OF_SERVICE**.

5 Create the **/tmp/snippets** (or **C:\tmp\snippets** for Windows) path but add/modify **only read** access, and then you can refresh the page, you should see the same as Figure 4.0:

Here you can see the **SnippetHealthCheck** value: **DOWN**.

6 Modify the path to write access and refresh, you should have the following, see Figure 5.0.

Here you can see the **SnippetHealthCheck** value: **UP**.

8.1.3. Challenge

You created a metrics for the number of visits at the home page, now:

- Create a custom endpoint **/snippet-metric** that shows all the statistics for the Snippet repository for every JPA Rest endpoint, in this case for every GET, POST, PUT, DELETE. Something like Figure 6.0:

Figure 6.0: Spring Boot Actuator - http://localhost:8080/admin/snippet-metric .

Chapter 9. Spring Boot Security

Get familiar with **Spring Security** and the **Spring Boot Security** features.

Time: 35 minutes.

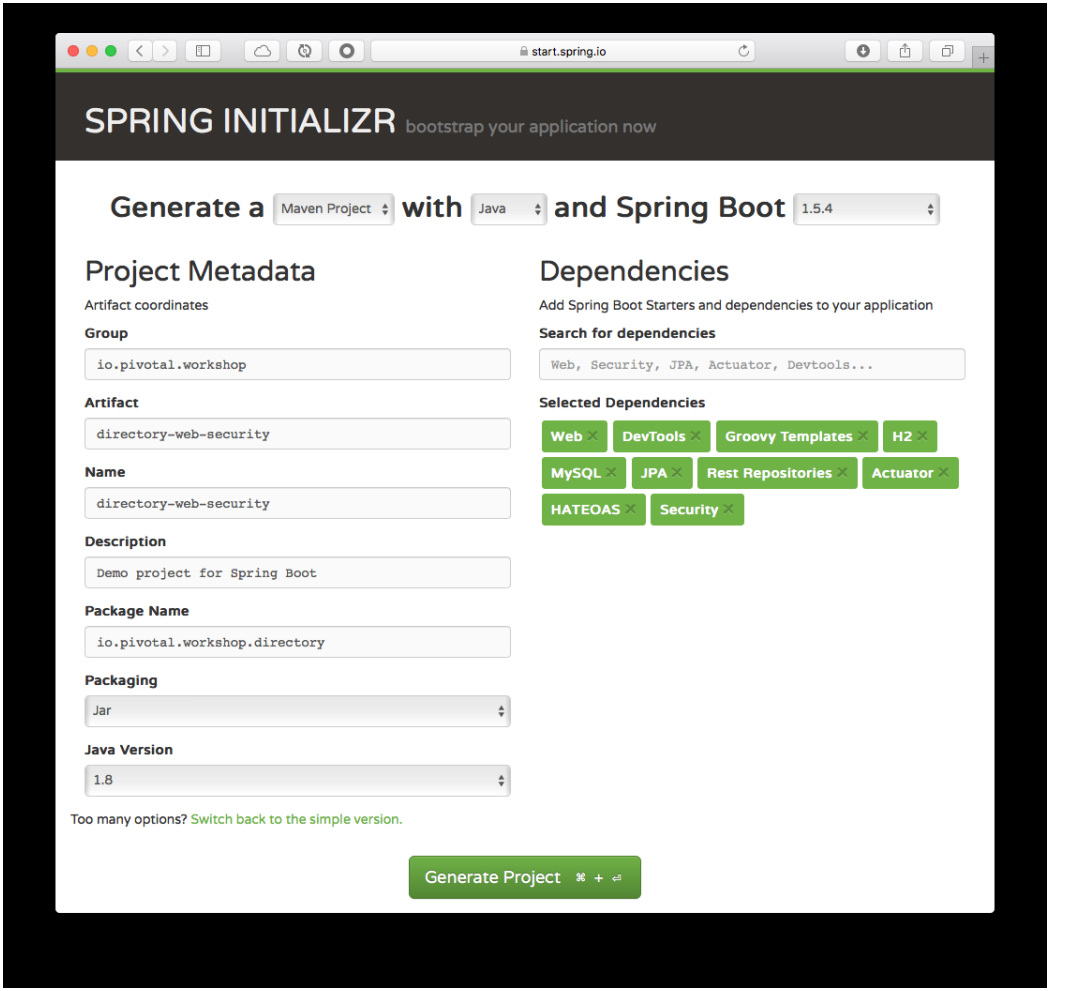
9.1. Directory Web Security App

Remember this lab? Where we have a persons directory? We are going to re-take part of the code and make this project more secure. You saw in the demo how easy is to set up the security using the JDBC but it was using a pre-configured schema (**users** and **authorities**). In this lab we will use our own schema (our own data) so you see how easy is to implement **spring-security** in a web project.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the Directory Web App Project metadata with (See Figure 1.0):

Table 9.1. Directory Web Security App - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*directory-web-security*
Name:	*directory-web-security*
Package Name:	*io.pivotal.workshop.directory*
Dependencies:	*Web, DevTools, H2, MySQL, Security, JPA, Rest Repositories, Actuator, HATEOAS, Groovy Templates*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **H2**, **MySQL**, **Security**, **JPA**, **Rest Repositories**, **Actuator**, **HATEOAS** and **Groovy Templates** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. You can copy the code from the first labs (**Spring Boot Overview**).

9. Because we are using **JPA** and **Rest Repositories** dependencies, lets convert the **Person** as entity. Create/Modify the **Person** class:

   io.pivotal.workshop.directory.domain.Person.java.

```
Unresolved directive in spring-boot-security.adoc - include::../../lab/spring-boot-security/directory-web-security/src/main/java/io/pivotal/workshop/directo
```

   See that we are using the **@Entity** and **@Id** annotations from **JPA**. What is new in this class is the two new fields: **role** and **enabled**, that we are going to use later on.

10. Next, create/modify the **PersonRepository** class:

   io.pivotal.workshop.directory.repository.PersonRepository.java.

```
Unresolved directive in spring-boot-security.adoc - include::../../lab/spring-boot-security/directory-web-security/src/main/java/io/pivotal/workshop/directo
```

   This is part of the **spring-data** project, where only by extending from the **CrudRepository<T,ID>** interface we get all the persistence functionality. Also take a look that we are defining a **findBy** named method, that will be also implemented for us.

11. Next, let create a configuration that will initialize our database:

   io.pivotal.workshop.directory.config.DirectoryConfig.java.

```
@Configuration
public class DirectoryConfig extends WebMvcConfigurerAdapter {

        @Override
        public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/").setViewName("views/home");
        }

        @Bean
        public CommandLineRunner directoryProcess(PersonRepository repo) {
                return args -> {
                        repo.save(new Person("john@email.com", "John C.", "simplepwd", "1980-08-03", "ADMIN", true));
                        repo.save(new Person("mike@email.com", "Mike H.", "simplepwd", "1980-04-10", "USER", true));
                        repo.save(new Person("mark@email.com", "Mark S.", "simplepwd", "1981-10-08", "USER", true));
                };
        }
}
```

   As you can see we are extending from **WebMvcConfigurerAdapter** and the purpose of this is to configure our home page (or view) by overriding the **addViewControllers** method (this is another way to configure a web controller).

12. We need to add our own security based on the **Person** class. Let's add the security configuration. Create the

DirectorySecurityConfig class:

io.pivotal.workshop.directory.config.DirectorySecurityConfig.java.

```
Unresolved directive in spring-boot-security.adoc - include::../../lab/spring-boot-security/directory-web-security/src/main/java/io/pivotal/workshop/directo
```

As you can see we are extending from **WebSecurityConfigurerAdapter** and it give us a way to override some methods, in this case the **configure(HttpSecurity)** (that provides an easy way to configure the request access) and **configure(AuthenticationManagerBuilder** (where we are adding our custom secured service, in this case the **UserDetailsService**).

13. Next, create the **DirectoryUserDetailsService** class that will have our custom access to our own schema:

io.pivotal.workshop.directory.security.DirectoryUserDetailsService.java.

```
Unresolved directive in spring-boot-security.adoc - include::../../lab/spring-boot-security/directory-web-security/src/main/java/io/pivotal/workshop/directo
```

In this class we are including the **PersonRepository** and we are using the **findByEmail** method. See that we are implementing the **UserDetailsService** interface and we are implementing the **loadUserByUsername** that returns a **UserDetails**.

14. Next, open the **src/main/resources/application.properties** file and add/modify it to look like the following:

src/main/resources/application.properties.

```
## REST
spring.data.rest.base-path=api

## ACTUATOR
management.context-path=/admin

## JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
```

As you can see, all these properties are well known from previous labs. The Rest repository is exposed in the **/api** endpoint and the **spring-boot-actuator** endpoint at the **/admin** context-path.

15. Add the necessary UI, remember where? Here are the files you need:

- theme.css
- offcanvas.css
- main.tpl
- home.tpl

16.Don’t forget to add the necessary dependencies in your **pom.xml** or **build.gradle**.

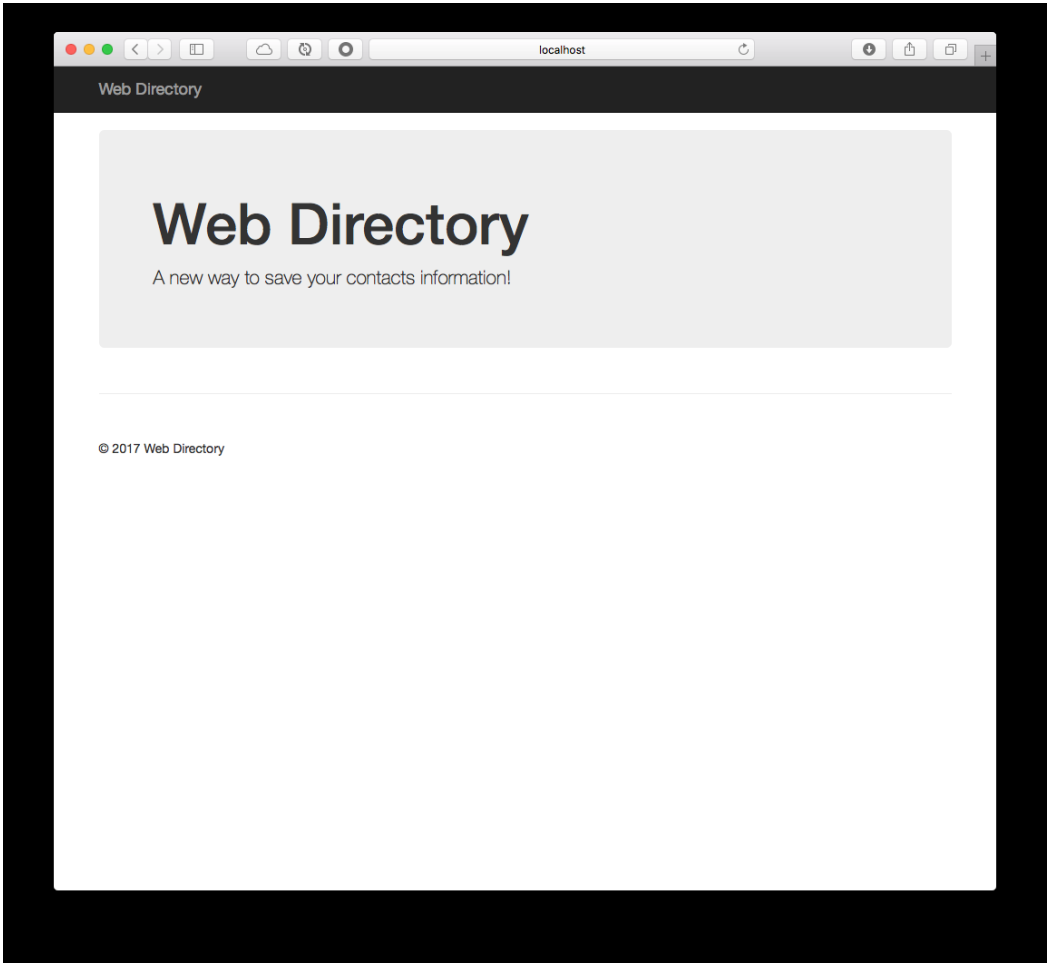
pom.xml.

```
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>jquery</artifactId>
        <version>2.2.4</version>
</dependency>
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>3.3.6</version>
</dependency>
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>angularjs</artifactId>
        <version>1.5.7</version>
</dependency>
```

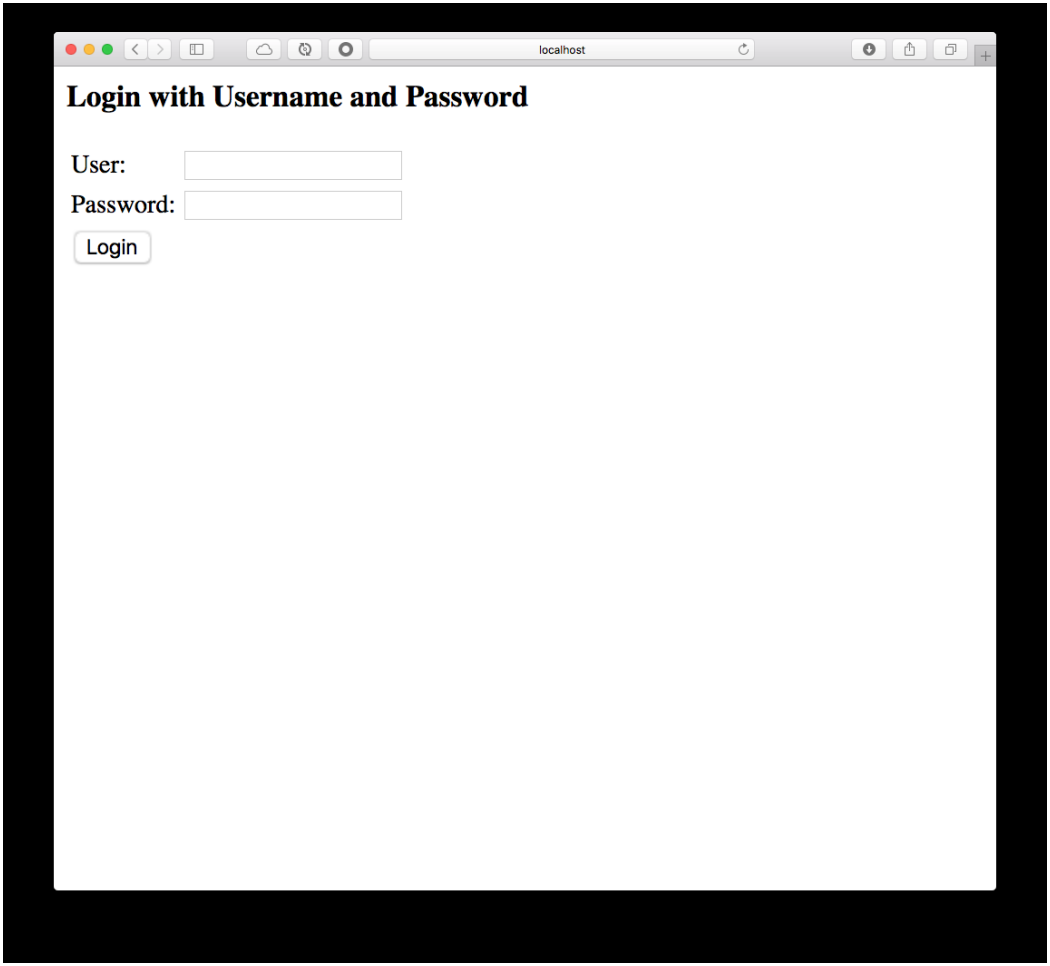
build.gradle.

```
compile('org.webjars:jquery:2.2.4')
compile('org.webjars:bootstrap:3.3.6')
compile('org.webjars:angularjs:1.5.7')
```

17.Run the application, either command line or through your IDE. If you go to the http://localhost:8080 in your browser, you should get the same as the following Figure 2.0:



18.If you try to go to the http://localhost:8080/api, you should get the following Figure 3.0:



You can now use one of the persons we added in the configurations, for example use: **john@email.com** and **simplepwd** as password, and you should get now the **Person Repository Rest API** response.

9.1.1. Challeges

- Normally you will use the REST API through a programmatic client, not directly in the browser, right? Use a terminal window and use the **cURL** command to access the REST API.

- You can use a client like Postman. Access the REST api using this tool.

9.2. Directory Web Security App with OAuth2

In this lab we are going to add the OAuth2 to our project. This project will be both: Authorization Server and Resource Server.

1. First add the necessary dependencies in your **pom.xml** or **build.gradle**.

   pom.xml.

```
<dependency>
        <groupId>org.springframework.security.oauth</groupId>
        <artifactId>spring-security-oauth2</artifactId>
</dependency>
```

   build.gradle.

```
compile('org.springframework.security.oauth:spring-security-oauth2')
```

2. Create the **DirectoryOAuth2Config** class that will be used as configuration for the **OAuth2** features.

   io.pivotal.workshop.directory.config.DirectoryOAuth2Config.java.

```
Unresolved directive in spring-boot-security.adoc - include::../../lab/spring-boot-security/directory-web-security/src/main/java/io/pivotal/workshop/directo
```

   See that here we are using **@EnableAuthorizationServer** (so it will expose the **/oauth/*** endpoints) and the **@EnableResourceServer** (that will be securing the Rest API through **OAuth2**). See also that we are extending from **ResourceServerConfigurerAdapter** and overriding the **configure(HttpSecurity)** method to provide the configuration for the **/api** endpoint.

3. In the **src/main/resources/application.properties** file append the following configuration:

```
## OAuth2
security.oauth2.client.client-id=workshop
security.oauth2.client.client-secret=springboot

security.oauth2.client.scope=read,write
security.oauth2.client.authorized-grant-types=client_credentials,password
security.oauth2.authorization.check-token-access=isAuthenticated()

security.oauth2.resource.filter-order=3
```

   The first part is to create a **client-id** and **client-secret**, you can omit these properties but then Spring Boot will generate (every time you start the app) them and you need to copy them. The second part is to define the scope and grant access for the client. The last part is necessary so the **OAuth2** mechanism will be the first to be configured and it can work with our configuration.

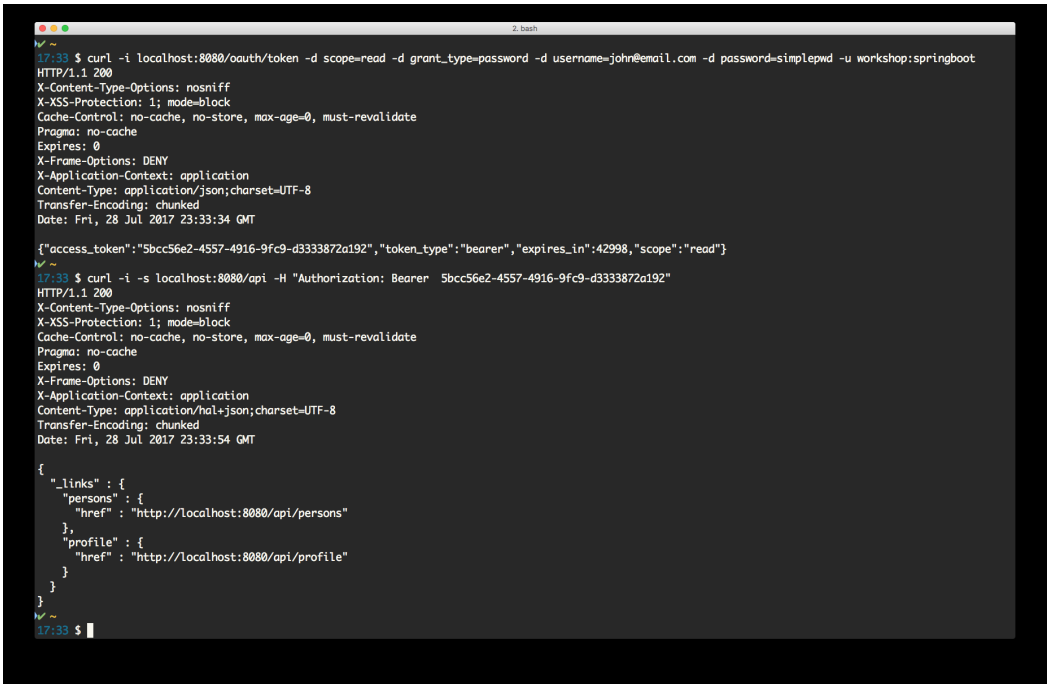
4. Open a new Terminal window and use the **cUrl** command to access to the **/api** endpoint using **OAuth2**:

```
$ curl -i localhost:8080/oauth/token -d scope=read -d grant_type=password -d username=john@email.com -d password=simplepwd -u workshop:springboot
```

5. Use the previous **access_token** and access the **/api** endpoint, see Figure 4.0.

```
$ curl -i -s localhost:8080/api -H "Authorization: Bearer <access_token>"
```

Figure 4.0: OAuth2 - Using cUrl.



Tip

If you are a Windows user, you can use Postman.

9.2.1. Challeges

- Use Postman to access the REST api using **OAuth2**.

- You already have a working application, the **Code Snippet Manager**, use this project as **Resource Server**, and the main **Authorization** will be the **Directory Web Security App**:
  - Modify the **Code Snippet Manager** project to use the **spring-oauth2** security as **Resource Server**.

9.3. HOMEWORK

- Add SSL to both projects.

Chapter 10. Spring Boot AMQP

Get familiar with the **RabbitMQ** and **Spring Boot AMQP** and its features.

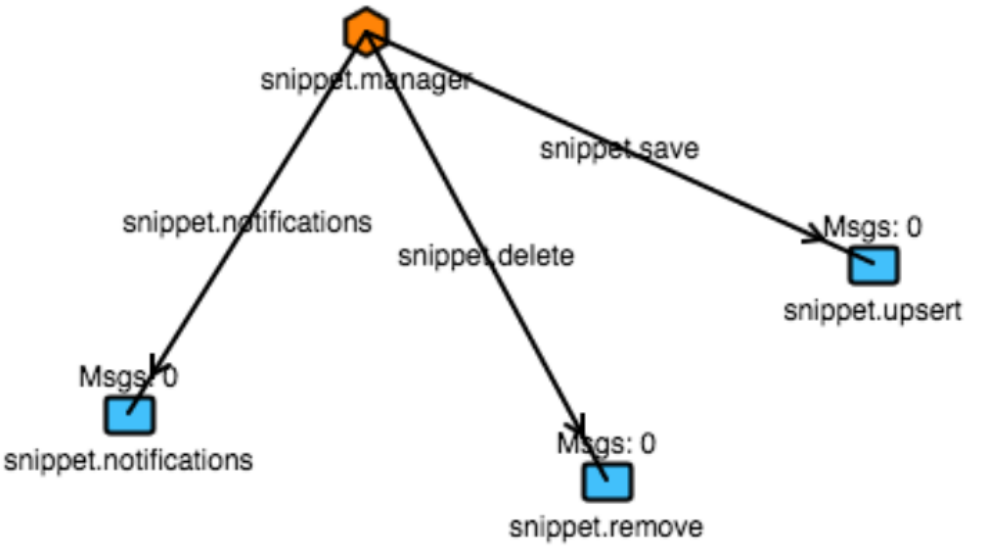
Time: 20 minutes.

10.1. Part 1: Code Snippet Manager AMQP

You will continue with the **Code Snippet Manager** code. For this lab you will need the **snippet-code-manager-actuator** project code.

There are new requirements for the **Code Snippet Manager**:

- Receive code snippets using RabbitMQ:
  - The communication should be as RPC (emulate synchronous communication) from/to a client.
  - Accept messages for **save**, **update** and **delete** code snippets.
  - Create Listeners for two queues: **snippet.upsert** and **snippet.remove**.
  - For every message that process, send a **Snippet Notification** to the **snippet.notifications** queue.
- Should you be able to create the next topology:



1 Open a browser and hit the url: http://start.spring.io

2 Click the **Switch to the full version** link.

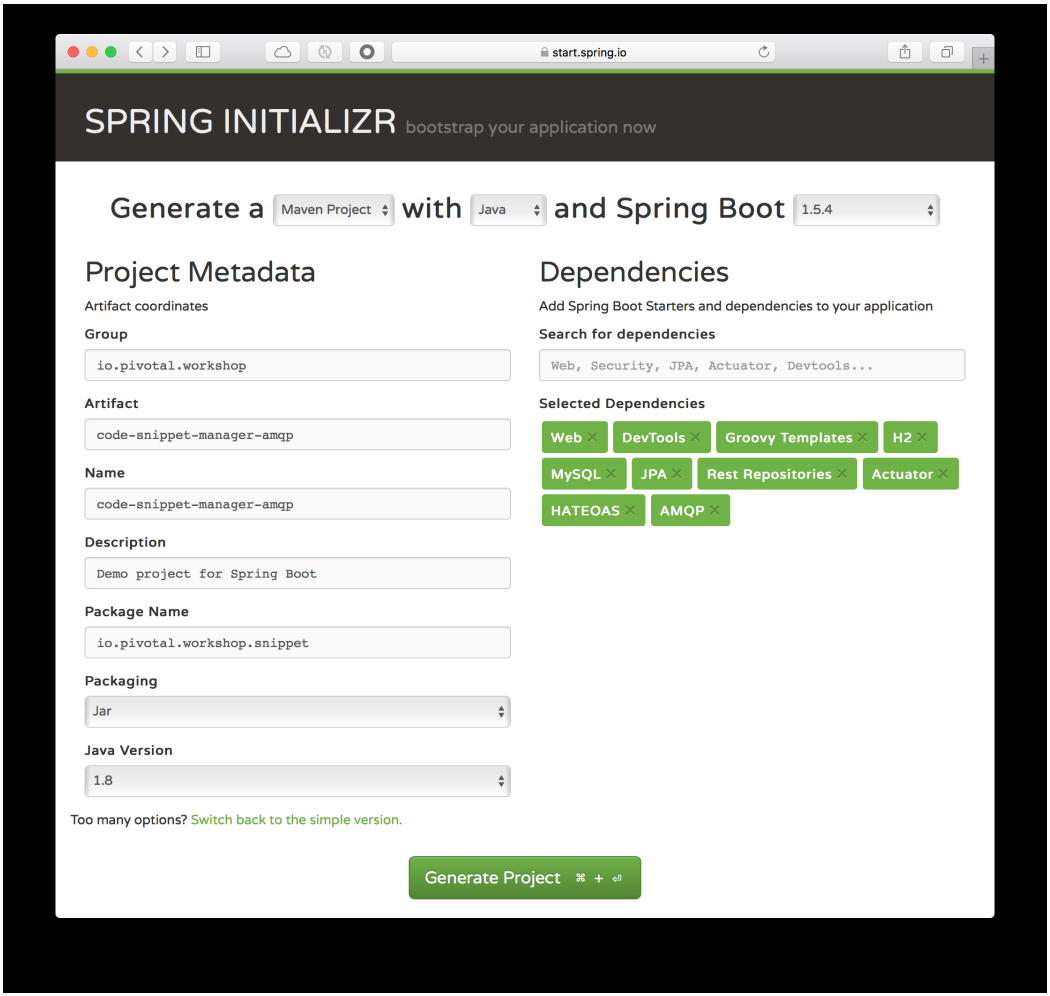
3 Fill out the **Code Snippet Manager AMQP** Project metadata with (See Figure 1.0):

Table 10.1. Code Snippet Manager AMQP - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*code-snippet-manager-amqp*
Name:	*code-snippet-manager-amqp*
Package Name:	*io.pivotal.workshop.snippet*

Property	Value
Dependencies:	*Web, DevTools, Groovy Templates, JPA, Rest Repositories, H2, MySQL, Actuator, HATEOAS, AMQP*

Figure 1.0: Spring Initializr - http://start.spring.io/ .



Tip

You can choose either **Maven** or **Gradle** project types.

4 Type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **Rest Repositories**, **H2**, **MySQL**, **Actuator**, **HATEOAS** and **AMQP** in the **Dependencies** field and press Enter.

5 Click the **Generate Project** button.

6 Unzip the file in any directory you want.

7 Import your project in any IDE you want.

8 Copy all the packages (with code) into the new project.

9 The packages: **domain**, **actuator**, **controller**, **repository** and **config** will remain with no change.

10 Next, start by adding the *Listeners* that will be using the **queues**, create the **SnippetAmqpListener** class in the **io.pivotal.workshop.snippet.amqp** package:

io.pivotal.workshop.snippet.amqp.SnippetAmqpListener.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/code-snippet-manager-amqp/src/main/java/io/pivotal/workshop/sn
```

In the code above we are using the **@RabbitListener** and **@SendTo** annotations. There are two methods that handle the messages (receiving a **Snippet**) from the two queues. Also notice that both methods return a new **SnippetNotification**. Actually the **@SendTo** annotation will be the mechanism to reply that notification to the client.

11 Create in the **domain** package the **SnippetNotification** and **SnippetError** classes:

io.pivotal.workshop.snippet.domain.SnippetNotification.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/code-snippet-manager-amqp/src/main/java/io/pivotal/workshop/sn
```

io.pivotal.workshop.snippet.domain.SnippetError.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/code-snippet-manager-amqp/src/main/java/io/pivotal/workshop/sn
```

12 Next let's create the configuration that will be use to convert the incoming messages into a **Snippet** instances and also let's wire up the **topology** we are going to need. Create the **SnippetAmqpConfig** class in the **io.pivotal.workshop.snippet.config** package:

io.pivotal.workshop.snippet.config.SnippetAmqpConfig.java.

```
package io.pivotal.workshop.snippet.config;

import org.springframework.amqp.core.AmqpAdmin;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.Binding.DestinationType;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SnippetAmqpConfig {

        private final String SNIPPET_EXCHANGE = "snippet.manager";

        @Bean
        public SimpleRabbitListenerContainerFactory rabbitListenerContainerFactory(ConnectionFactory connectionFactory){
                SimpleRabbitListenerContainerFactory container = new SimpleRabbitListenerContainerFactory();
                container.setConnectionFactory(connectionFactory);
                container.setMessageConverter(new Jackson2JsonMessageConverter());
                return container;
        }

        @Bean
        public DirectExchange directExchange(){
                return new DirectExchange(SNIPPET_EXCHANGE, true, false);
        }

        @Bean
        public Queue upsert(){
                return new Queue("snippet.upsert");
        }

        @Bean
        public Queue remove(){
                return new Queue("snippet.remove");
        }

        @Bean
        public CommandLineRunner queuesAndBindings(AmqpAdmin admin){
                return args -> {
                    admin.declareBinding(new Binding("snippet.upsert", DestinationType.QUEUE, SNIPPET_EXCHANGE, "snippet.save", null));
                        admin.declareBinding(new Binding("snippet.remove", DestinationType.QUEUE, SNIPPET_EXCHANGE, "snippet.delete", null));
                        admin.declareBinding(new Binding("snippet.upsert", DestinationType.QUEUE, SNIPPET_EXCHANGE, "snippet.update", null));
                };
        }
}
```

In the above code we need to create a **SimpleRabbitListenerContainerFactory** so it can any **JSON** format into our **Snippet** class. In other words, the queues will receive a snippet as **JSON** object. Also we are creating the **Exchange** and the **Queues**, and after the application is about to start we **Bind** the exchange with all the queues by specifying the **Routing Keys**.

13 Now you can run the code and take a look at the **RabbitMQ Web Console** by going to http://localhost:15672 and see the exchange and queues created and the consumers listening to the queues.

Tip

The username/password for the RabbitMQ console is: **guest**/**guest**.

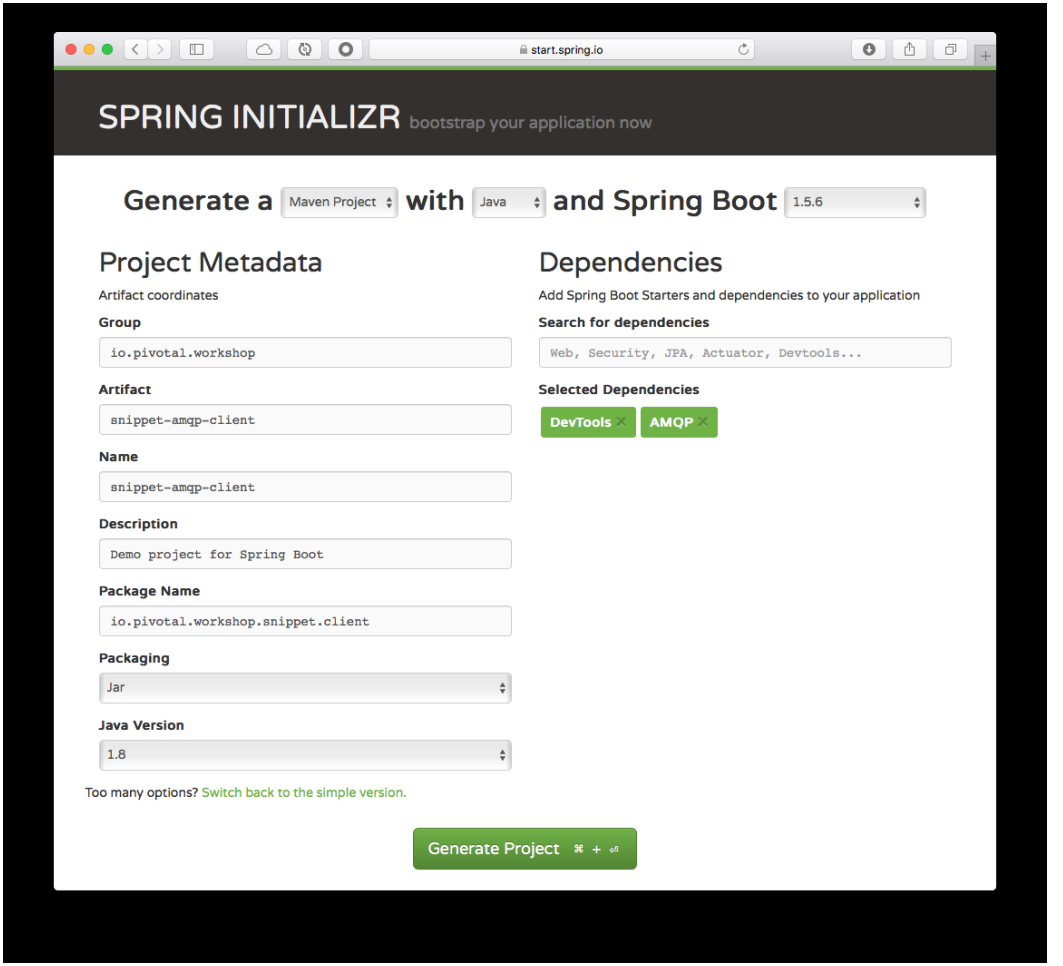
10.2. Part 2: Snippet AMQP Client

The idea of this application is be able to send snippets to be saved or delete, and receive a notification.

1. Open a browser and hit the url: http://start.spring.io
2. Click the **Switch to the full version** link.
3. Fill out the **Snippet AMQP Client** Project metadata with (See Figure 2.0):

Table 10.2. Snippet AMQP Client - metadata

Property	**Value**
Group:	*io.pivotal.workshop*
Artifact:	*snippet-amqp-client*
Name:	*snippet-amqp-client*
Package Name:	*io.pivotal.workshop.snippet.client*
Dependencies:	*DevTools, AMQP*



Tip

You can choose either **Maven** or **Gradle** project types.

4. Type **Web**, **DevTools**, **Groovy Templates**, **JPA**, **Rest Repositories**, **H2**, **MySQL**, **Actuator**, **HATEOAS** and **AMQP** in the **Dependencies** field and press Enter.

5. Click the **Generate Project** button.

6. Unzip the file in any directory you want.

7. Import your project in any IDE you want.

8. Lets start by creating the domains will use, create the following classes in the **io.pivotal.workshop.snippet.client.domain** package:

   io.pivotal.workshop.snippet.client.domain.Code.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

   io.pivotal.workshop.snippet.client.domain.Language.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

   io.pivotal.workshop.snippet.client.domain.Snippet.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

   io.pivotal.workshop.snippet.client.domain.SnippetError.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

   io.pivotal.workshop.snippet.client.domain.SnippetNotification.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

9. Next, lets create a producer that will send the messages to the exchange, create the **SnippetProducer** class in the **io.pivotal.workshop.snippet.client.amqp** package:

   io.pivotal.workshop.snippet.client.amqp.SnippetProducer.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

   Here we are using the **RabbitTemplate** and it's method **convertSendAndReceive** (here we are doing the RPC - emulating the **synchronous** communication).

10. Because we are doing an **RPC**, we are getting back from the **Code Snippet Manager AMQP** app the **SnippetNotification** (as **JSON**), so we need to configure the **RabbitTemplate** to support this conversion. Create the **SnippetClientAmqpConfig** in the **io.pivotal.workshop.snippet.client.config** package:

    io.pivotal.workshop.snippet.client.config.SnippetClientAmqpConfig.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```


Its important to notice that we are using the same converter as before, the **Jackson2JsonMessageConverter**, that practically will map the **JSON** object to the **SnippetNotification**. Here we are adding a little trick, a special header: **TypeId** that will be use as reference for the mapping, our own **SnippetNotification**.

Tip

If you want to know more about this solution, ask the instructor to explain a little further. Also you can take a look at the documentation: http://docs.spring.io/spring-amqp/reference/html/_reference.html#message-converters

11. Now its time to send some **Snippet**. Create the **SnippetClientConfig** class in the **io.pivotal.workshop.snippet.client.config** package:

io.pivotal.workshop.snippet.client.config.SnippetClientConfig.java.

```
Unresolved directive in spring-boot-messaging.adoc - include::../../lab/spring-boot-messaging/snippet-amqp-client/src/main/java/io/pivotal/workshop/snippet/
```

Here we are using the **Snippet** producer to send the **Snippet** message.

12. Now you can run the application, and you should get your response back, a **SnippetNotification** object.

10.3. Challenges

As you already know we havent implemented some of the requirements:

- Create the **snippet.notifications** queue and bind it to the **snippet.manager** exchange.
- The **Code Snippet Manager AMQP** app needs to send a notification every time it receives a **Snippet**:
  - Without modifying the **SnippetAmqpListener** add the behavior to send a **SnippetNotification** to the **snippet.notifications** queue.

Tip

Use **AOP** for this particular concern and use the **RabbitTemplate** to send a **asynchronous** notification.

- Make the **Snippet AMQP Client** app listen for any notification.

Chapter 11. Microservices with Spring Boot

Create Microservices with Spring Boot and deploy them into Cloud Foundry

Time: 35 minutes.

11.1. Cloud Foundry

This lab is just to get familiar with some of the common Cloud Foundry command using the CF CLI.

1. Make sure you have an account in PWS if not, create a trial account in Pivotal Web Services: https://run.pivotal.io
2. Install Cloud Foundry CLI. You can get it from: https://github.com/cloudfoundry/cli#installers-and-compressed-binaries
3. Login into Cloud Foundry

   cf login.

```
$ cf login -a api.run.pivotal.io
```

11.1.1. Useful CF CLI commands

cf help - Show help.

```
$ cf help -a
```

marketplace - List available offerings in the marketplace.

```
$ cf marketplace
```

services - List all service instances in the target space.

```
$ cf services
```

create-service - Create a service instance.

```
$ cf create-service cloudamqp lemur rabbitmq
```

bind-service - Bind a service instance to an app.


```
$ cf bind-service myapp rabbitmq
```

push - Push a new app or sync changes to an existing app.

```
$ cf push myapp
```

set-env - Set an env variable for an app.

```
$ cf set-env myapp MY_VARIABLE ITS_VALUE
```

11.2. Deploy Spring Boot Microservices to Cloud Foundry

For this lab you will deploy two projects, the **directory-web-security** and the **code-snippet-manager-security**, both must be completed with **OAuth2** implementation. Remember that the **directory-web-security** is the **Authorization Server** and the **code-snippet-manager-security** is the **Resource Server**. If you haven't finish the challenges for both projects, the it's time to do it.

11.2.1. Deploying directory-web-security project

1. Open a terminal window in the **directory-web-security** project and create the JAR file.

   maven.

```
./mvnw clean package -DskipTests=true
```

   gradle.

```
./gradlew build -x test
```

2. Deploy the **directory-web-security** JAR.

   If you used Maven, it creates the JAR in the target/ folder. If you used Gradle, the JAR is in the build/libs folder.

```
cf push directory-web -p target/directory-web-security-0.0.1-SNAPSHOT.jar --random-route -b java_buildpack
```

3. Once deployed make sure is working by accessing the **/api** using the **cUrl** command:

   This is an example. Change the URL accordingly to your random-route.

```
curl -i http://directory-web-unapprehended-pluralism.cfapps.io/oauth/token -d scope=read -d grant_type=password -d username=john@email.com -d password=simpl
```


Use the access_token from the previous command.

```
curl -i -s http://directory-web-unapprehended-pluralism.cfapps.io/api -H "Authorization: Bearer  <access_token>"
```

11.2.2. Deploying code-snippet-manager-security project

1. Open your **code-snippet-manager-security** project because we need to do a small modification in the code. Open the **SnippetConfiguration** class. Modify it to look like the following code:

```
package io.pivotal.workshop.snippet.config;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.actuate.metrics.CounterService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.web.servlet.handler.MappedInterceptor;

import io.pivotal.workshop.snippet.domain.Code;
import io.pivotal.workshop.snippet.domain.Language;
import io.pivotal.workshop.snippet.domain.Snippet;
import io.pivotal.workshop.snippet.interceptor.ApiInterceptor;
import io.pivotal.workshop.snippet.repository.SnippetRepository;

@Configuration
public class SnippetConfiguration {

        @Bean
        public MappedInterceptor apiInterceptor(CounterService counterService) {
            return new MappedInterceptor(new String[]{"/api/**"},new ApiInterceptor(counterService));
        }

    @Bean
        @Profile("local")
        public CommandLineRunner runner(SnippetRepository snippetRepo) {
                return args -> {
                        @SuppressWarnings("serial")
                        List<Snippet> snippets = new ArrayList<Snippet>() {
                                {
                                        add(new Snippet("Hello World", new Language("HTML", "xml"),new Code(new String(Files.readAllBytes(Paths.get("code/ht
                                        add(new Snippet("Hello World", new Language("C#", "c#"),new Code(new String(Files.readAllBytes(Paths.get("code/cs-co
                                        add(new Snippet("Hello World", new Language("Pascal", "py"),new Code(new String(Files.readAllBytes(Paths.get("code/p
                                        add(new Snippet("Hello World", new Language("Erlang", "erl"),new Code(new String(Files.readAllBytes(Paths.get("code/
                                        add(new Snippet("Hello World", new Language("JavaScript", "js"),new Code(new String(Files.readAllBytes(Paths.get("co
                                        add(new Snippet("Hello World", new Language("Groovy", "groovy"),new Code("println 'Hello World'")));
                                }
                        };

                        snippetRepo.save(snippets);

                };
        }

    @Bean
        @Profile("cloud")
        public CommandLineRunner runnerCloud(SnippetRepository snippetRepo) {
                return args -> {
                        @SuppressWarnings("serial")
                        List<Snippet> snippets = new ArrayList<Snippet>() {{
                                add(new Snippet("Hello World", new Language("JavaScript", "js"),new Code("console.log(\"Hello World\");")));
                                add(new Snippet("Hello World", new Language("Groovy", "groovy"),new Code("println 'Hello World'")));
                        }};
                        snippetRepo.save(snippets);
                };
        }
}
```


As you can see we are using the **@Profile** annotation, and we adding it to the **runner** method, making it a *local* profile, but also we are adding a new method **runnerCloud** and using the *cloud* profile. Why do we need to do this? well we are not deploying the **code/** folder, so there are no initial snippets.

Tip

By default Cloud Foundry activates the **cloud** profile.

2. Open a terminal window in the **code-snippet-manager-security** project and create the JAR file.

   maven.

```
./mvnw clean package -DskipTests=true
```

   gradle.

```
./gradlew build -x test
```

3. Deploy the **code-snippet-manager-security** JAR but don't start the application just yet.

   If you used Maven, it creates the JAR in the target/ folder. If you used Gradle, the JAR is in the build/libs folder.

```
cf push code-snippet-manager -p target/code-snippet-manager-security-0.0.1-SNAPSHOT.jar --random-route -b java_buildpack --no-start
```

4. If you completed the challenge for **code-snippet-manager-security** project, you should added the **security.oauth2.resource.tokenInfoUri** pointing to the **Authorization Server**. In this case we are going to use the Cloud Foundry environment variables to set this property for the **code-snippet-manager** app.

```
cf set-env code-snippet-manager SECURITY_OAUTH2_RESOURCE_TOKENINFOURI http://directory-web-unapprehended-pluralism.cfapps.io/oauth/check_token
```

5. Start the **code-snippet-manager** application.

```
cf start code-snippet-manager
```

6. Once started make sure is working by accessing the **code-snippet-manager** app **/api** using the **cUrl** command:

   This is an example. Change the URL accordingly to your random-route. Remember this is the directory-web app, our Authorization Server .

```
curl -i http://directory-web-unapprehended-pluralism.cfapps.io/oauth/token -d scope=read -d grant_type=password -d username=john@email.com -d password=simpl
```


Use the access_token from the previous command. Here we are using the code-snippet-manager url and use the token to access the /api endpoint.

```
curl -i -s http://code-snippet-manager-nonanesthetic-ceruse.cfapps.io/api -H "Authorization: Bearer <access_token>"
```

11.3. Challenges

- Add the code from the **code-snippet-manager-amqp** to use a RabbitMQ service.
- Instead of using an **H2** as persistence engine, use a **MySQL** service, what changes do you need to do in your code?

Appendix A. Spring XML Configuration Tips

A.1. Bare-bones Bean Definitions

```
<bean id="rewardNetwork" class="rewards.internal.RewardNetworkImpl">
</bean>

<bean id="accountRepository" class="rewards.internal.account.JdbcAccountRepository">
</bean>

<bean id="restaurantRepository" class="rewards.internal.restaurant.JdbcRestaurantRepository">
</bean>

<bean id="rewardRepository" class="rewards.internal.reward.JdbcRewardRepository">
</bean>
```

Figure A.1. Bare-bones bean definitions

A.2. Bean Class Auto-Completion

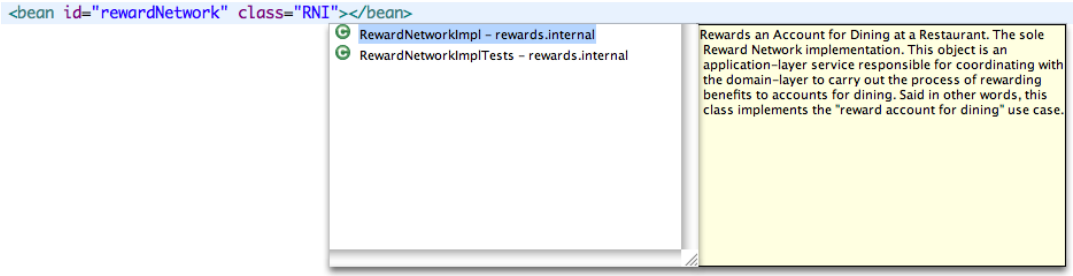


Figure A.2. Bean class auto-completion

A.3. Constructor Arguments Auto-Completion

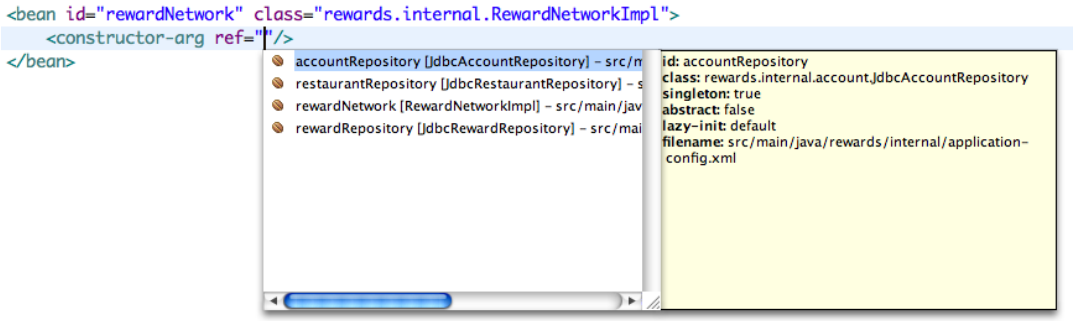


Figure A.3. Constructor argument auto-completion

A.4. Bean Properties Auto-Completion

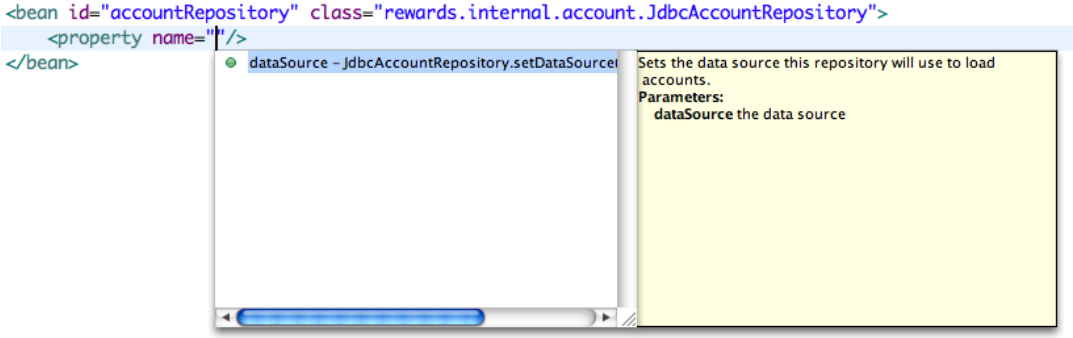


Figure A.4. Bean property name completion

Appendix B. Eclipse Tips

B.1. Introduction

This section will give you some useful hints for using Eclipse.

B.2. Package Explorer View

Eclipse's Package Explorer view offers two ways of displaying packages. Flat view, used by default, lists each package at the same level, even if it is a subpackage of another. Hierarchical view, however, will display subpackages nested within one another, making it easy to hide an entire package hierarchy. You can switch between hierarchical and flat views by selecting the menu inside the package view (represented as a triangle), selecting either Flat or Hierarchical from the Package Presentation submenu.

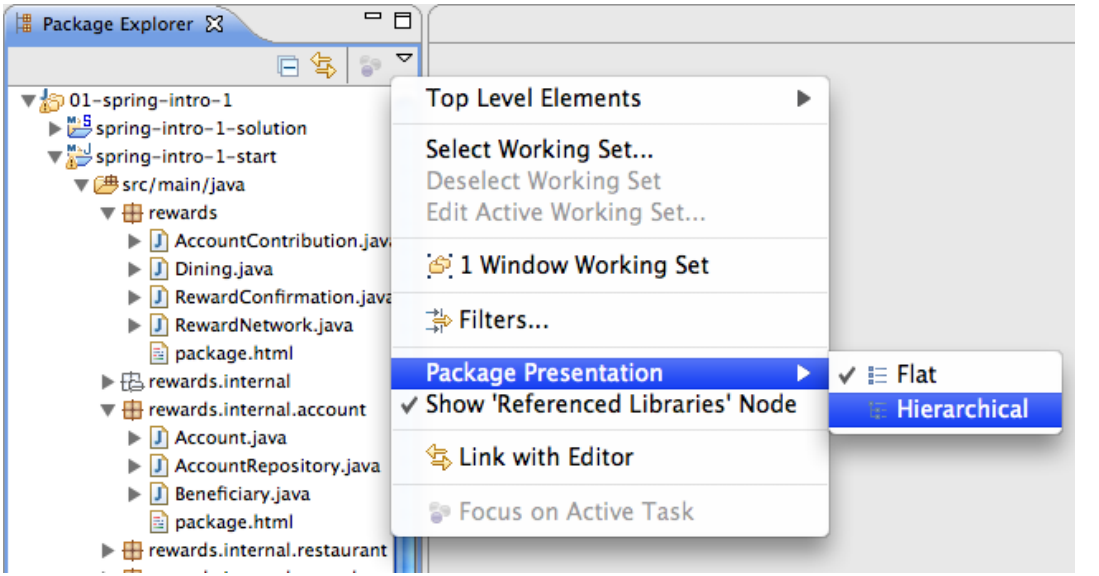


Figure B.1. Swtiching Views

Switch between hierarchical and flat views by selecting the menu inside the package view (represented as a

triangle), selecting either Flat or Hierarchical from the Package Presentation submenu

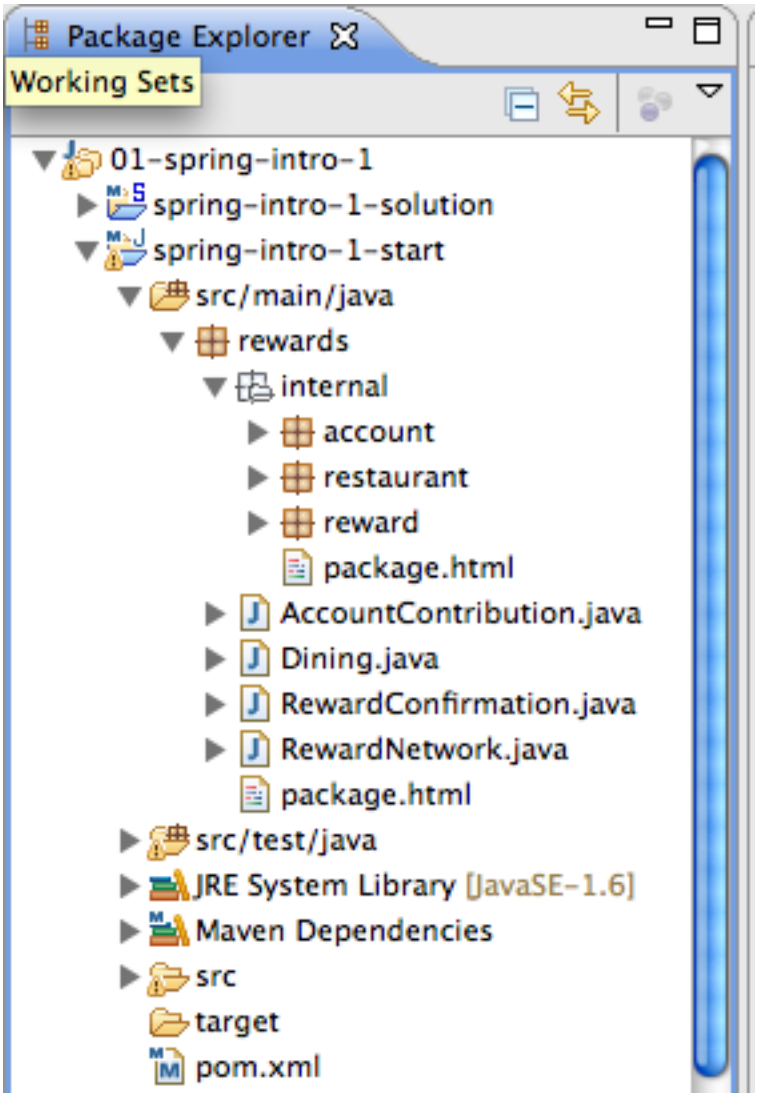


Figure B.2. The hierarchical view shows nested packages in a tree view

B.3. Add Unimplemented Methods

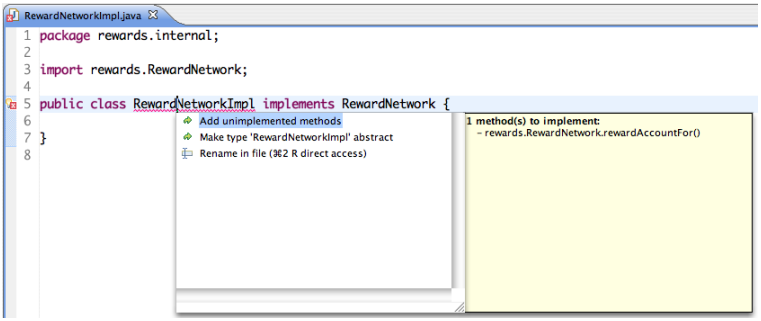


Figure B.3. Add unimplemented methods" quick fix

B.4. Field Auto-Completion

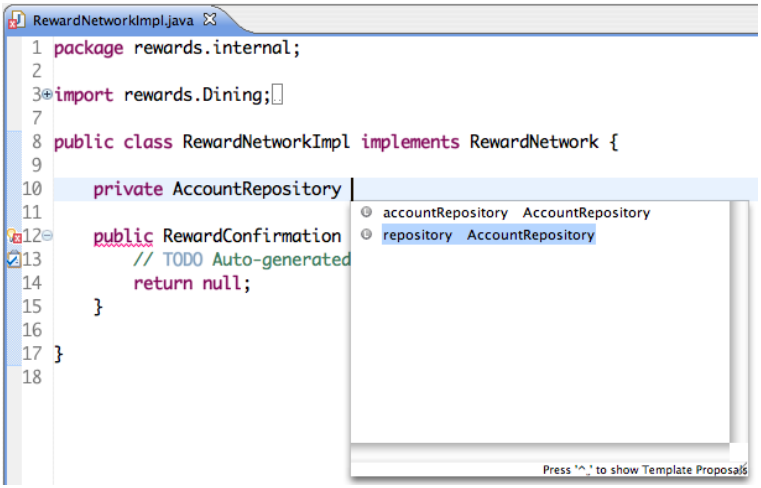


Figure B.4. Field name auto-completion

B.5. Generating Constructors From Fields

You can "Generate a Constructor using Fields" using the Source Menu (ALT + SHIFT + S)

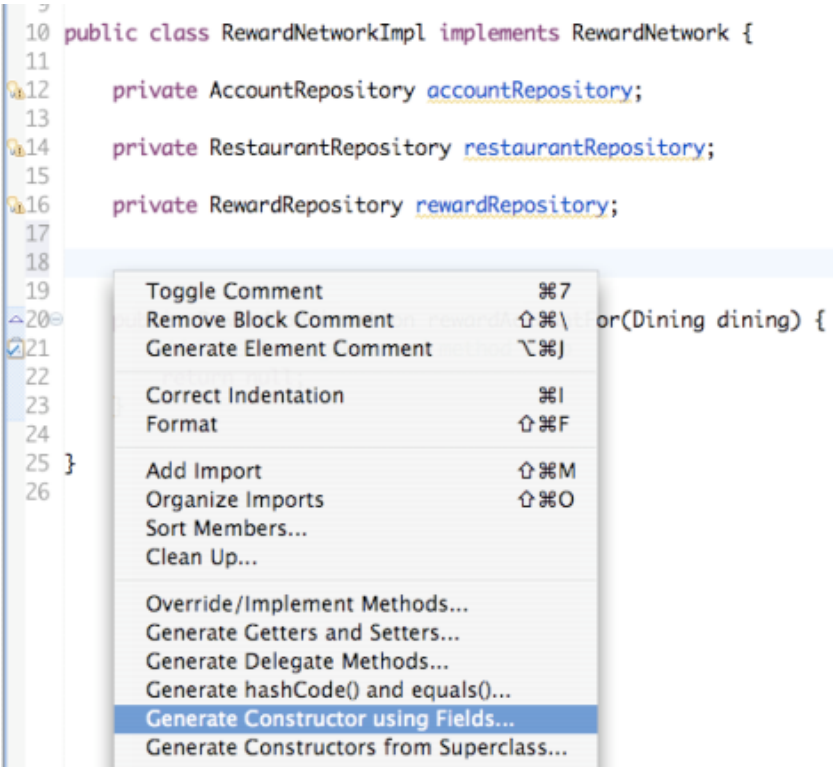


Figure B.5. Generating Constructors

B.6. Field Naming Conventions

A field's name should describe the role it provides callers, and often corresponds to the field's type. It should not describe implementation details. For this reason, a bean's name often corresponds to its service interface. For example, the class JdbcAccountRepository implements the AccountRepository interface. This interface is what callers work with. By convention, then, the bean name should be accountRepository.

B.7. Tasks View

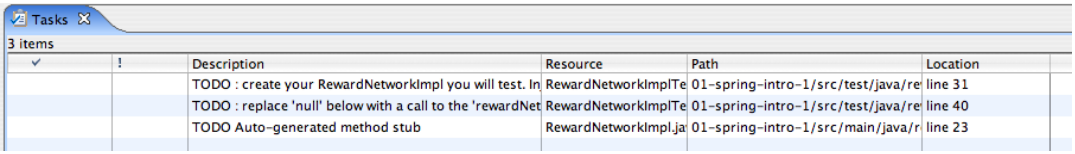


Figure B.6. The tasks view in the bottom right page area

You can configure the Tasks View to only show the tasks relevant to the current project. In order to do this, open the dropdown menu in the upper-right corner of the tasks view (indicated by the little triangle) and select 'Configure Content...'. Now select the TODO configuration and from the Scopes, select 'On any element in same project'. Now if you have multiple project opened, with different TODOs, you will only see those relevant to the current project.

B.8. Rename a File

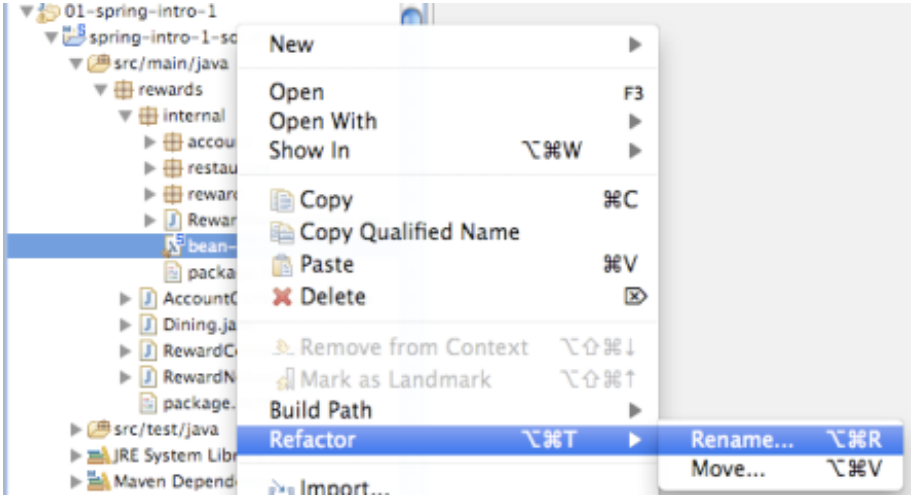


Figure B.7. Renaming a Spring configuration file using the Refactor command

