Building a Spring Boot Application

Learning Outcomes

After completing the lab, you will be able to:

1. Describe how to create runnable Spring Boot application
2. Describe how to create a controller that responds to HTTP requests
3. Use gradle to run gradle tasks

The pages codebase contains a local Git repository with the starting points and the solutions for all the labs in this unit. Download the linked zip file and extract the codebase in the ~/workspace directory. The extracted directory will contain a single text file as well as the (hidden) Git files. You will be building up the code in this directory bit by bit, and we have provided reference implementations at each stage identified by tags in the Git repository. Take some time to navigate through the tags and branches using the following command:

```
git log --graph --decorate --oneline --all
```

Create a repository called pages in your GitHub account. Add this repository as a remote called origin of your local repository.Keep everything default, while creating the repository, don't change anything other than default. You will push all of your work to this repository during the next few labs.

We will start by pushing the initial commit to GitHub, complete with the start and solutions tags.

```
git push origin master --tags
```

We can then navigate to GitHub and view the solution tags. This is handy when you get stuck during a lab and need a little help.

1. Create a build.gradle file with following content

```
plugins {
        id 'org.springframework.boot' version '2.3.1.RELEA
SE'
        id 'io.spring.dependency-management' version '1.0.
9.RELEASE'
        id 'java'
}
```

```
group = 'com.example'
sourceCompatibility = '11'

repositories {
        mavenCentral()
}

dependencies {
        implementation 'org.springframework.boot:spring-bo
ot-starter-web'
        testImplementation('org.springframework.boot:sprin
g-boot-starter-test') {
                exclude group: 'org.junit.vintage', modul
e: 'junit-vintage-engine'
        }
}

test {
        useJUnitPlatform()
}
```

2. Create the gradle ecosystem by using the following commands

```
gradle wrapper --gradle-version 6.4.1 --distribution-type
all
```

3. Open the project in Intellij
4. Right-click on the project, select the option to crate new directory and create `src/main/java` and `src/test/java`
5. Create two packages `org.dell.kube.pages` and `org.dell.kube.pagesapi` under src/test/java
6. Create a test class called `HomeControllerTest` within package `org.dell.kube.pages` with below content

```
package org.dell.kube.pages;

import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class HomeControllerTest {
    private final String message = "YellowPages";
```

```
    @Test
    public void itSaysYellowPagesHello() throws Exception
{
        HomeController controller = new HomeController();

        assertThat(controller.getPage()).contains(messag
e);
    }


}
```

7. Create a Test class called `HomeApiTest` under the package `org.dell.kube.pagesapi` with below content

```
package org.dell.kube.pagesapi;

import org.dell.kube.pages.PageApplication;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowi
red;
import org.springframework.boot.test.context.SpringBootTes
t;
import org.springframework.boot.test.web.client.TestRestTe
mplate;

import static org.assertj.core.api.Assertions.assertThat;
import static org.springframework.boot.test.context.Spring
BootTest.WebEnvironment.RANDOM_PORT;

@SpringBootTest(classes = PageApplication.class, webEnviro
nment = RANDOM_PORT)
public class HomeApiTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Override
    protected Object clone() throws CloneNotSupportedExcep
tion {
        return super.clone();
    }

    @Test
    public void readTest() {
        String body = this.restTemplate.getForObject("/",
```

```
String.class);
        assertThat(body).contains("YellowPages");
    }
}
```

8. Create a `settings.gradle` file in the root project directory with below content

```
rootProject.name = 'pages'
```

9. Create package `org.dell.kube.pages` within `src/main/java`
10. Create class `PageApplication` within the `package org.dell.kube.pages`
11. `PageApplication.java` - Example code snippet

```
package org.dell.kube.pages;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootAp
plication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class PageApplication {

        public static void main(String[] args) {
                SpringApplication.run(PageApplication.clas
s, args);
        }
}
```

12. Create class `HomeController` within the package `org.dell.kube.pages`
13. `HomeController.java` - Example code snippet

```
package org.dell.kube.pages;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapp
ing;
import org.springframework.web.bind.annotation.RestControl
ler;

@RestController
@RequestMapping("/")
public class HomeController {
```

```
    @GetMapping
    public String getPage(){
        return "Hello from page : YellowPages";
    }


}
```

14. Add `application.properties` in both test and src
15. Add the below content in both the properties files

```
spring.application.name=pages
```

16. Build & test the application

```
./gradlew clean build
```

17. Run the application

```
./gradlew bootRun
```

18. Access your application

```
Browse to http://localhost:8080
```

19. Commit your code to your github repository

```
git add .
git commit -m "commit message"
git push -u origin master
```