Lab 4 - Create a RESTful Service and Endpoint

One can edit `Application.java` under `src/main/java` and add a RESTful service implementation. The RESTful service is exactly the same as what was done in the previous project. Append the following code at the end of the `Application.java` file:

Copy

```
@RestController
class GreetingController{
  @RequestMapping("/")
  Greet greet(){
    return new Greet("Hello World!");
  }
}
class Greet {
  private String message;
public Greet() {}

  public Greet(String message) {
    this.message = message;
  }
//add getter and setter
}
```

To run, navigate to **Run As** | **Spring Boot App**. Tomcat will be started on the `8080` port:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.3.5.RELEASE)

2016-05-11 16:49:10.236  INFO 41130 --- [           main]
org.rvslab.chapter2.Application          : Starting Application on rvslab.local with
PID 41130 (/Users/rajeshrv/work/codebox/chapter2/chapter2.bootrest/target/classes
```

We can notice from the log that:

- Spring Boot get its own process ID (in this case, it is `41130`)

- Spring Boot is automatically started with the Tomcat server at the localhost, port `8080`.

Next, open a browser and point to `http://localhost:8080`. This will show the JSON response as shown in the following screenshot:

A key difference between the legacy service and this one is that the Spring Boot service is self-contained. To make this clearer, run the Spring Boot application outside STS. Open a terminal window, go to the project folder, and run Maven, as follows:

```
$ maven install
```

This will generate a fat JAR file under the target folder of the project. Running the application from the command line shows:

```
$java -jar target/bootrest-0.0.1-SNAPSHOT.jar
```

As one can see, `bootrest-0.0.1-SNAPSHOT.jar` is self-contained and could be run as a standalone application. At this point, the JAR is as thin as 13 MB. Even though the application is no more than just "Hello World", the Spring Boot service just developed, practically follows the principles of microservices.

Testing the Spring Boot microservice

There are multiple ways to test REST/JSON Spring Boot microservices. The easiest way is to use a web browser or a curl command pointing to the URL, as follows:

```
curl http://localhost:8080
```

There are number of tools available to test RESTful services, such as Postman, Advanced REST client, SOAP UI, Paw, and so on.

In this example, to test the service, the default test class generated by Spring Boot will be used.

Adding a new test case to `ApplicatonTests.java` results in:

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = Application.class)
@WebIntegrationTest
public class ApplicationTests {
  @Test
  public void testVanillaService() {
    RestTemplate restTemplate = new RestTemplate();
    Greet greet = restTemplate.getForObject("http://localhost:8080", Greet.class);
    Assert.assertEquals("Hello World!", greet.getMessage());
  }
}
```

Note that `@WebIntegrationTest` is added and `@WebAppConfiguration` removed at the class level. The `@WebIntegrationTest` annotation is a handy annotation that ensures that the tests are fired against a fully up-and-running server. Alternately, a combination of `@WebAppConfiguration` and `@IntegrationTest` will give the same result.

Also note that `RestTemplate` is used to call the RESTful service. `RestTemplate` is a utility class that abstracts the lower-level details of the HTTP client.

To test this, one can open a terminal window, go to the project folder, and run `mvn install`.