Exercise 1. CREATE A "REST JPA REPOSITORY" SPRING BOOT PROGRAM

Overview	
In this exercise you will write a basic Spring Boot program that allows for the adding and finding of a "User" Object from an in-memory database (H2) via REST API's. The objectives of this exercise are: 1. Learn how to configure and run a Spring Boot program. 2. Become confident that you understand how "AutoConfiguration" works. 3. Gain familiarity with the basics of the Spring-Data Repository configuration.	
Objective	Familiarity with creating and running Spring Boot programs.
Builds on Previous Labs	Standalone. However, all the code for this exercise is directly covered in the course.
Time to Complete	30 minutes

Start your IDE and import "spring-boot-rest-service-lab" Project

Prior to starting on this exercise, if you have not already done so, please work through the tutorial on setting up your IDE for this course. Take your time working through that tutorial, making sure you understand what you are being asked to do because you will be working with these tools and utilities throughout the remainder of the course.

1. Import an "***Existing Maven Projects***" called **spring-boot-rest-service-lab**. You can perform this operation from a variety of starting points (right-click in the Project Explorer and select **Import->import**).
2. Select an import source of "**Existing Maven Projects**" and click **Next**.
3. For the Root Directory, select the lab-code directory: **~/StudentWork/Labs/spring-boot-rest-service-lab**
4. Click the **Finish** button.

You should now have the **spring-boot-rest-service-lab** project created that includes a sub-section called ***Maven Dependencies*** that includes all the required libraries for this project.

Add Maven build support for Spring Boot.

Step 1: Add Maven Plugin and Spring Boot Starters.

5. Open **`pom.xml`**, located in project root **`~/`** directory, and switch to XML view (far right-hand tab in Eclipse file view)**.**
6. Look for the ***`<plugins>`*** section of the `pom.xml`.
7. Add the spring boot maven plugin inside the ***`<plugins>`*** section:

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

We're now going to add two spring boot starters to the maven project as dependencies.

8. Look for the ***<dependencies>*** section of the `pom.xml`.
9. Add the **spring-boot-starter-data-rest** inside the ***<dependencies>*** section.
10. Add the **spring-boot-starter-data-jpa** inside the ***<dependencies>*** section.
11. Make sure to **save** all files.

Step 2: Create the `SpringBootRestServiceApplication` class

12. Right click on the **`com.springclass.boot`** package, located in projects **`~/src/main/java`** directory, and select **New -> Class**.
13. Name the new class `SpringBootRestServiceApplication` and click **Finish**.
14. Annotate the Class with `@SpringBootApplication` to enable Auto Configuration.
15. Add a ***`main`*** method and run the Spring application `SpringApplication.run(SpringBootRestServiceApplication.class, args);`

Step 3: Create the `User` class

16. Right click on the **`com.springclass.boot`** package, located in projects **`~/src/main/java`** directory, and select **New -> Package**.
17. Name the new package **`com.springclass.boot.domain`** and click **Finish**.
18. Right click on the **`com.springclass.boot.domain`** package and select **New -> Class**.
19. Name the new class **`User`** and click **Finish**.
20. Annotate the **`User`** class with **`@javax.persistence.Entity`** .
21. Add Variables for **`long id, String firstName, String lastName`**, and add setters and getters.
22. Annotate variable **`id`** with **`@Id`** and **`@GeneratedValue(strategy = GenerationType.AUTO)`** .

Step 4: Create the `UserRepository` class

23. Right click on the **`com.springclass.boot.domain`** package, located in projects **`~/src/main/java`** directory, and select **New -> Interface**.
24. Name the new interface **`UserRepository`** and click **Finish**.
25. Annotate the **`UserRepository`** interface with **`@RepositoryRestResource`** .
26. Add an **`extends`** declaration for this interface to extend **`CrudRepository<User, Long>`**.
27. Make sure to save your work.

Step 5: Review the `TestClient` class

28. Open the **`com.springclass.boot.client.TestClient.java`** file, located in projects **`~/src/test/java`** directory.
29. Inspect the **`main()`** method to see how the **`RestTemplate`** is used to interact with our local REST service.

Step 6: Run the Spring Boot Application

30. Run the **`SpringBootRestServiceApplication`** program. You can execute **`SpringBootRestServiceApplication`** by right-clicking on it and selecting **Run As -> Java Application**.

Hint: If your IDE has Spring Boot support, you can also run the application by right-clicking on it and selecting **Run As -> Spring Boot App**.

Step 7: Run the program and inspect the output

31. Run the **`TestClient`** program. You can execute **`TestClient`** by right-clicking on it and selecting **Run As -> Java Application**.

In the console you may see some output of the logging system, followed by ***"*** Successfully created a User through REST ***"***

32. Optionally, open the **`application.properties`** file located in the **`~/src/main/resources/`** directory, and add an entry for **`debug=true`** to turn on DEBUG logging, which is available from the **`devtools`** starter.