Exercise 1. Create a "Thymeleaf MVC with JPA Repository" Spring Boot Program

Overview	
In this exercise you will write a basic Spring Boot program that allows for Bootstrapping Object's into an in-memory database (H2) then view that data through Thymeleaf Page. The objectives of this exercise are: 1. Add additional starters to a Spring Boot application. 2. Become confident how to bootstrap data on Application start-up. 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-mvc-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-mvc-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-mvc-lab**
4. Click the **Finish** button.

You should now have the **spring-boot-mvc-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 dependencies for additional 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)**.**

Presented by:
Mick Knutson
https://www.baselogic.com

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

6. Look for the ***<dependencies>*** section of the `pom.xml`.
7. Add the **`spring-boot-starter-actuator`** inside the ***<dependencies>*** section.
8. Add the **`spring-boot-starter-actuator-docs`** inside the ***<dependencies>*** section.
9. Add the **`spring-boot-starter-thymeleaf`** inside the ***<dependencies>*** section.
10. Add the **`spring-boot-starter-web`** inside the ***<dependencies>*** section.
11. Make sure to **save** all files.

Step 2: Create the `RepositoryBootstrapper` 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 **`RepositoryBootstrapper`** and click **Finish**.
14. This class must be annotated with the `@Component` annotation to allow it to be component scanned.
15. Edit this class to implement `CommandLineRunner` to enable this class as a start-up Bean.
16. Auto wire in a variable called `UserRepository repository` to be able to add data to this repository at start up.
17. Override the ***`run(String... args)`*** method :

```
@Override public void run(String... args) throws Exception {...}
```

18. Inside the run method, we want to add a few **`User`**'s to the **`UserRepository`** such as the following:

```
System.out.println("*** RepositoryBootstrapper ***");
repository.save(new User("Mick", "Knutson"));
repository.save(new User("Mick", "Knutson"));
repository.save(new User("Dan", "Corsberg"));
repository.save(new User("Peter", "Schmitz"));
repository.save(new User("Chuck", "Norris"));
```

Hint: feel free to experiment with adding additional data.

19. Make sure to save your work.

Step 3: Create the `WelcomeController` class

20. Right click on the **`com.springclass.boot`** package, located in projects **`~/src/main/java`** directory, and select **New** -> **Package**.
21. Name the new package **`com.springclass.boot.web`** and click **Finish**.
22. Right click on the **`com.springclass.boot.web`** package and select **New** -> **Class**.
23. Name the new class **`WelcomeController`** and click **Finish**.
24. Annotate the **`WelcomeController`** class with **`@Controller`** stereotype annotation.
25. Add variable for **`UserRepository`** and annotate the variable with **`@Autowired`**.

26. Add a welcome method to map to the URI "/" that will return a Map containing a single User from the database

Hint: The `RepositoryBootstrapper` adds several User objects to the database. We only need to pull a single user for display.

Hint: The method should look something like:

```
@RequestMapping("/welcome")
public String welcome(Map<String, Object> model) {
    model.put("user", repository.findOne(1L));
    return "welcome";
}
```

Step 4: Create the `Welcome.html` view file

27. Right click on the **`~/src/main/resources/templates`** directory, and select **New -> File**.
28. Name the new file **`welcome.html`** and click **Finish**.
29. Add the following header information for this view file:

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Spring Boot Thymeleaf Mvc Excercise - Welcome</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" th:href="@{/css/main.css}"
   href="../../css/main.css" />
</head>
```

30. Add the following ***<body>*** information for this view file:

```
<body><div>
        <div class="starter-template">
              <h1>Spring Boot Thymeleaf Mvc Excercise</h1>
              <h2>Single User from database</h2>
        </div>
        <table>
              <tr>
                    <th>ID</th>
                    <th>First Name</th>
                    <th>Last Name</th>
              </tr>
              <tr>
                    <td th:text="${user.id}">-1</td>
```

```
                        <td th:text="${user.firstName}">Chuck</td>
                        <td th:text="${user.lastName}">Norris</td>
                  </tr>
            </table>
</div></body></html>
```

31. Inspect both `<head>` and `<body>` elements and notice there is zero JSP scriptlets, and no HTML styling. That has been pushed into the CSS for clear separation.
32. Make sure to save your work.

Step 6: Run the Spring Boot Application

33. Run the **`SpringBootMvcApplication`** program. You can execute **`SpringBootMvcApplication`** 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**.

34. After application startup, inspect the Console logs, and you should see the *`RepositoryBootstrapper`* printing to the logs:

```
... c.s.boot.RepositoryBootstrapper: *** RepositoryBootstrapper ***
```

Step 7: Open a web browser and navigate to the application

35. Open a web browser and navigate to **`http://localhost:8080`** and you should see the index page stating **"Spring Boot Thymeleaf Mvc Exercise".**
36. Now navigate to **`http://localhost:8080/welcome`** and you should see the page stating **"Single User from database"** followed by a table containing a single User retrieved from the database.