



URL to GitHub Repository: <https://github.com/tmccarthy91/Week-13-Assignments>

URL to Public Link of your Video: <https://youtu.be/vMXFs7KhwUA>

Instructions :

1. Follow the [Exercises](#) below to complete this assignment.


- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the functionality into your Video when you see: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project. Don't forget to include the requested functionality, indicated by: 
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.

2. In addition, please include the following in your Coding Assignment Document:

- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-


Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When something is required to be included in your video submission, look for the icon: 


Note: You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.


Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

In the project you started last week, use Lombok to add an info-level logging statement in the controller implementation method that logs the parameters that were input to the method. Remember to add the `@Slf4j` annotation to the class.

Start the application (not an integration test). Use a browser to navigate to the application passing the parameters required for your selected operation. (A browser, used in this manner, sends an HTTP GET request to the server.) In your video, show the browser navigation bar and the log statement that is in the IDE console showing that the controller method was reached (as in the video). 

With the application still running, use the browser to navigate to the OpenAPI documentation. Use the OpenAPI documentation to send a GET request to the server with a valid model and trim level. (You can get the model and trim from the provided `data.sql` file.) In your video, show the `curl` command, the request URL, and the response headers. 

Run the integration test and show that the test status is green. In your video, show the test class and the status bar. 

Add a method to the test to return a list of expected Jeep (`model`) objects based on the model and trim level you selected. You can get the expected list of Jeeps from

the file `src/test/resources/flyway/migrations/V1.1__Jeep_Data.sql`. So, for example, using the model Wrangler and trim level "Sport", the query should return two rows:

	Row 1	Row 2
Model ID	WRANGLER	WRANGLER
Trim Level	Sport	Sport
Num Doors	2	4
Wheel Size	17	17
Base Price	\$28,475.00	\$31,975.00


1)

The method should be named `buildExpected()`, and it should return a `List` of `Jeep`. The video put this method into a support superclass but you can include it in the main test class if you want.

- 1) Write an AssertJ assertion in the test to assert that the actual list of jeeps returned by the server is the same as the expected list. Run the test. Demonstrate in your video the...

The test with the assertion.

The JUnit status bar (should be red).

The method returning the expected list of Jeeps. 

Add a service layer in your application as shown in the videos:

Add a package named `com.promineotech.jeep.service`.

In the new package, create an interface named `JeepSalesService`.


In the same package (service), create a class named `DefaultJeepSalesService` that implements the `JeepSalesService` interface. Add the class-level annotation, `@Service`.

Inject the service interface into DefaultJeepSalesController using the @Autowired annotation. The instance variable should be private, and the variable should be named jeepSalesService.

Define the fetchJeeps method in the interface. Implement the method in the service class. Call the method from the controller (make sure the controller returns the list of Jeeps returned by the service method). The method signature looks like this:

```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```

Add a Lombok info-level log statement in the service implementation showing that the service was called. Print the parameters passed to the method. Let the method return null for now.

Run the test again. In your video, show your service class implementation, as well as the log line in the console. 


Add the database dependencies described in the video to the POM file (MySQL driver and Spring Boot Starter JDBC). To find them, navigate to <https://mvnrepository.com/>. Search for mysql-connector-j and spring-boot-starter-jdbc. In the POM file you don't need version numbers for either dependency because the version is included in the Spring Boot Starter Parent.

Create application.yaml in src/main/resources. Add the spring.datasource.url, spring.datasource.username, and spring.datasource.password properties to application.yaml. The url should be the same as shown in the video (jdbc:mysql://localhost:3306/jeep). The password and username should match your setup. If you created the database under your root user, the username is "root", and the password is the root user password. If you created a "jeep" user or other user, use the correct username and password.

Be careful with the indentation! YAML allows hierarchical configuration but it reads the hierarchy based on the indentation level. The keyword "spring" MUST start in the first column. It should look similar to this when done:

```
spring:
  datasource:
    username: username
    password: password
```

```
url: jdbc:mysql://localhost:3306/jeep
```

Start the application (the real application, not the test). In your video, show your finished `application.yaml` and the console showing that the application has started with no errors. 

Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".

Create `application-test.yaml` in `src/test/resources`. Add the setting `spring.datasource.url` that points to the H2 database. It should look like this:

```
spring:
```

```
  datasource:
```

```
    url: jdbc:h2:mem:jeep;mode=MYSQL
```

You do not need to set the username and password because the in-memory H2 database does not require them.

In your video, show your finished `application-test.yaml`. 