Web API Design with Spring Boot Week 14 Coding Assignment

Points possible: 75

URL to GitHub Repository: https://github.com/Brierre/SpringBoot-jeep-sales-repo

URL to Public Link of your Video: https://rumble.com/v1xcj6e-week-14-video-explanation.html

---

Instructions :

1. Follow the **Coding Steps** 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 screenshots into this Assignment Document indicated by: 🖥️
    - Create a video showcasing your work:
        - In this video: record and present your project verbally while showing the results of the working project.
        - 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 requested screenshots, indicated by: 🖥️
    - 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.

Web API Design with Spring Boot Week 14 Coding Assignment

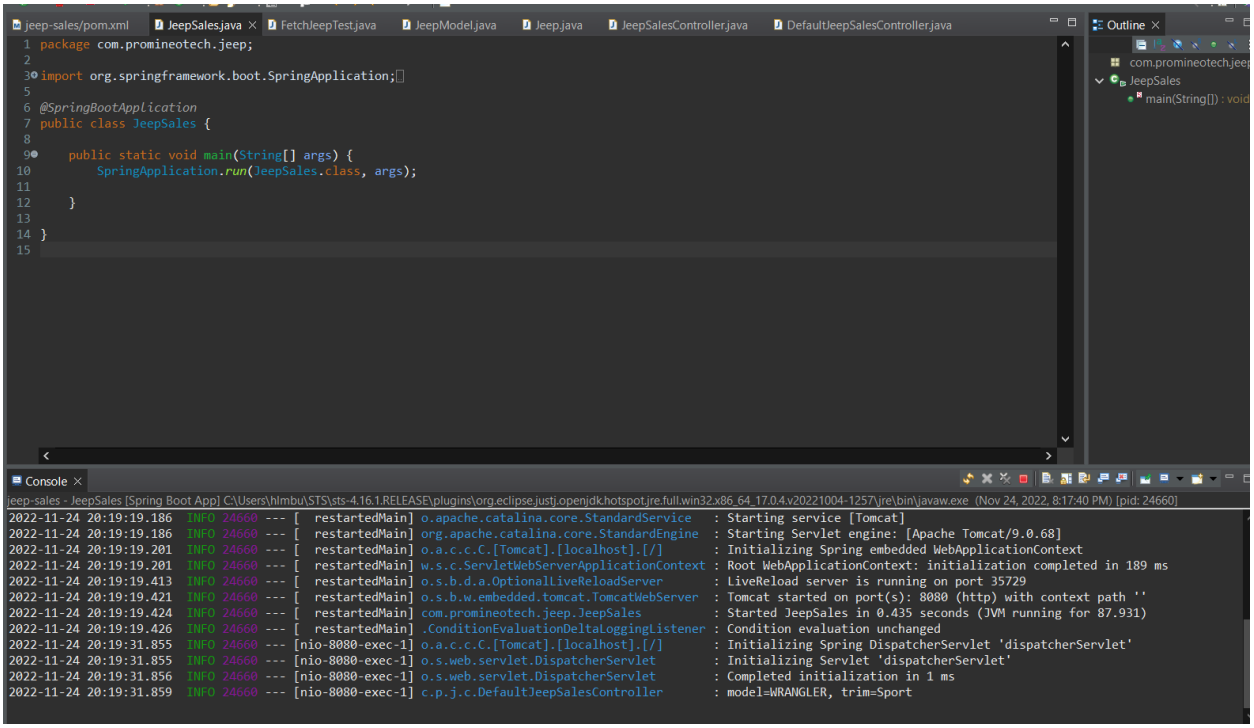
Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon: 💻 You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

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

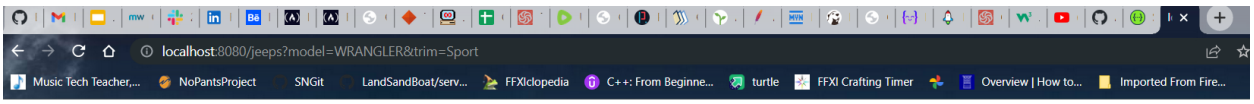
Coding Steps:

1) 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.

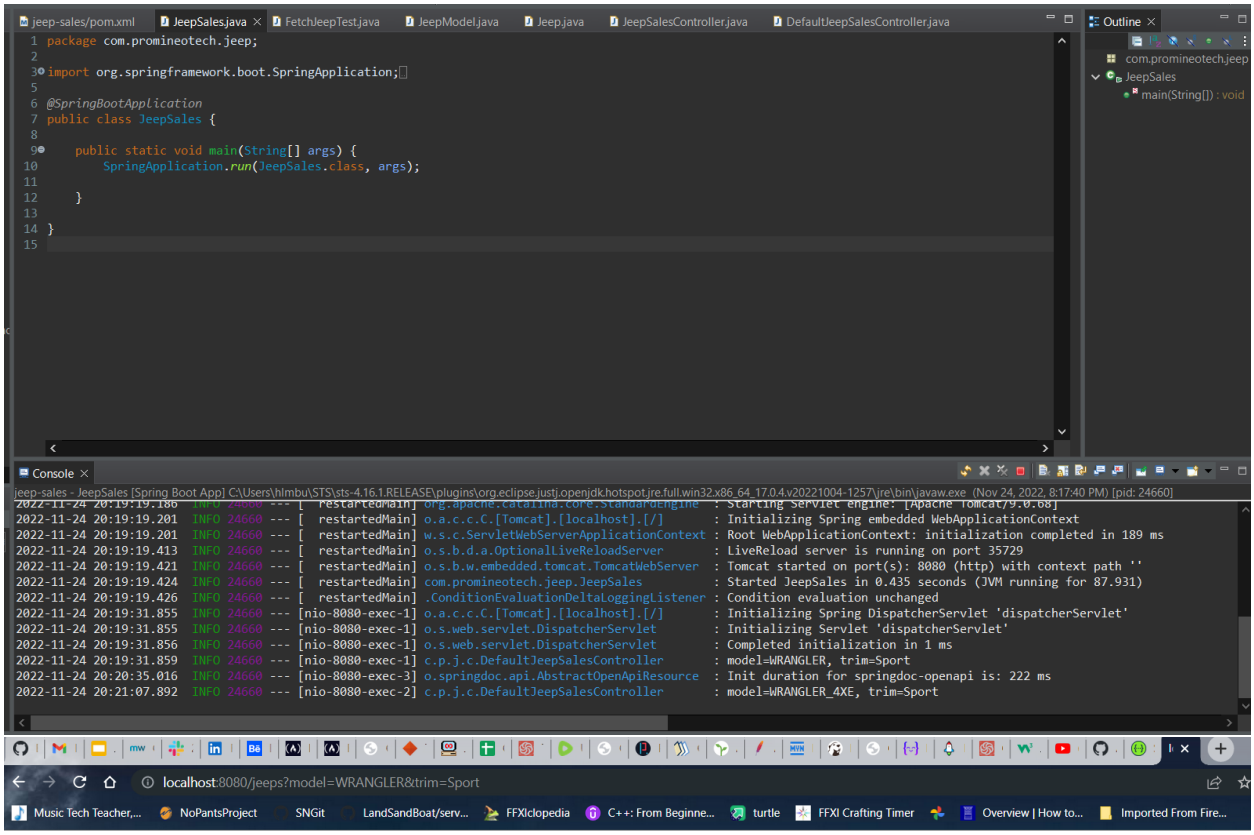
2) Start the application (not an integration test). Use a browser to navigate to the application passing the parameters required for your selected operation. (A browser, used in this manner, sends an HTTP GET request to the server.) Produce a screenshot showing the browser navigation bar and the log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

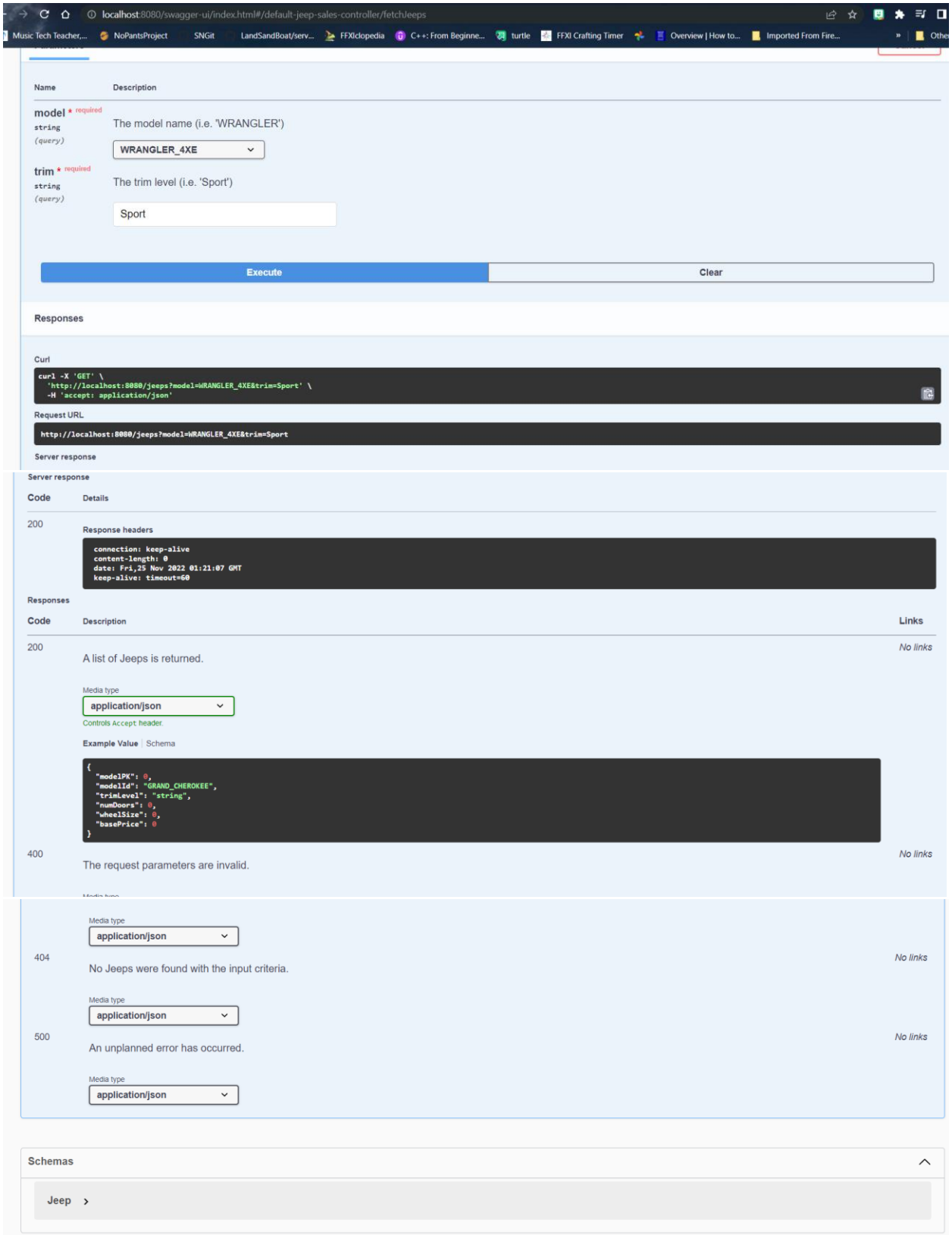


3) With the application still running, use the browser to navigate to the OpenAPI documentation. Use the OpenAPI documentation to send a GET request to the server with a valid model and trim level. (You can get the model and trim from the provided `data.sql` file.) Produce a screenshot showing the `curl`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

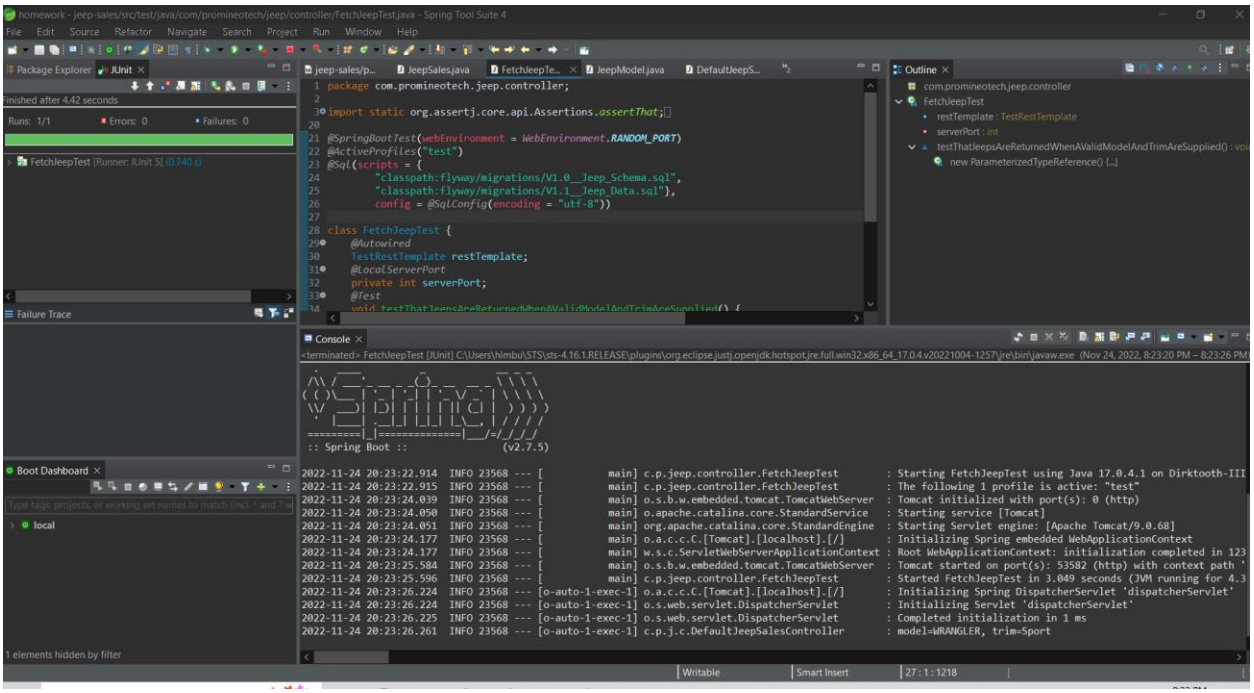


Web API Design with Spring Boot Week 14 Coding Assignment




localhost:8080/swagger-ui/index.html#/default-jeep-sales-controller/fetchJeeps
Name
Description
model * required
string
(query)
The model name (i.e. 'WRANGLER')
WRANGLER_4XE
trim * required
string
(query)
The trim level (i.e. 'Sport')
Sport
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8080/jeeps?model=WRANGLER_4XE&trim=Sport' \
-H 'accept: application/json'
Request URL
http://localhost:8080/jeeps?model=WRANGLER_4XE&trim=Sport
Server response
Server response
Code
Details
200
Response headers
connection: keep-alive
content-length: 0
date: Fri,25 Nov 2022 01:21:07 GMT
keep-alive: timeout=60
Responses
Code
Description
Links
200
A list of Jeeps is returned.
No links
Media type
application/json
Controls Accept header.
Example Value
Schema
{
"modelPK": 0,
"modelId": "GRAND_CHEROKEE",
"trimLevel": "string",
"numDoors": 0,
"wheelSize": 0,
"basePrice": 0
}
400
The request parameters are invalid.
No links
Media type
application/json
404
No Jeeps were found with the input criteria.
No links
Media type
application/json
500
An unplanned error has occurred.
No links
Media type
application/json
Schemas
Jeep

4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar.



5) 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

6)

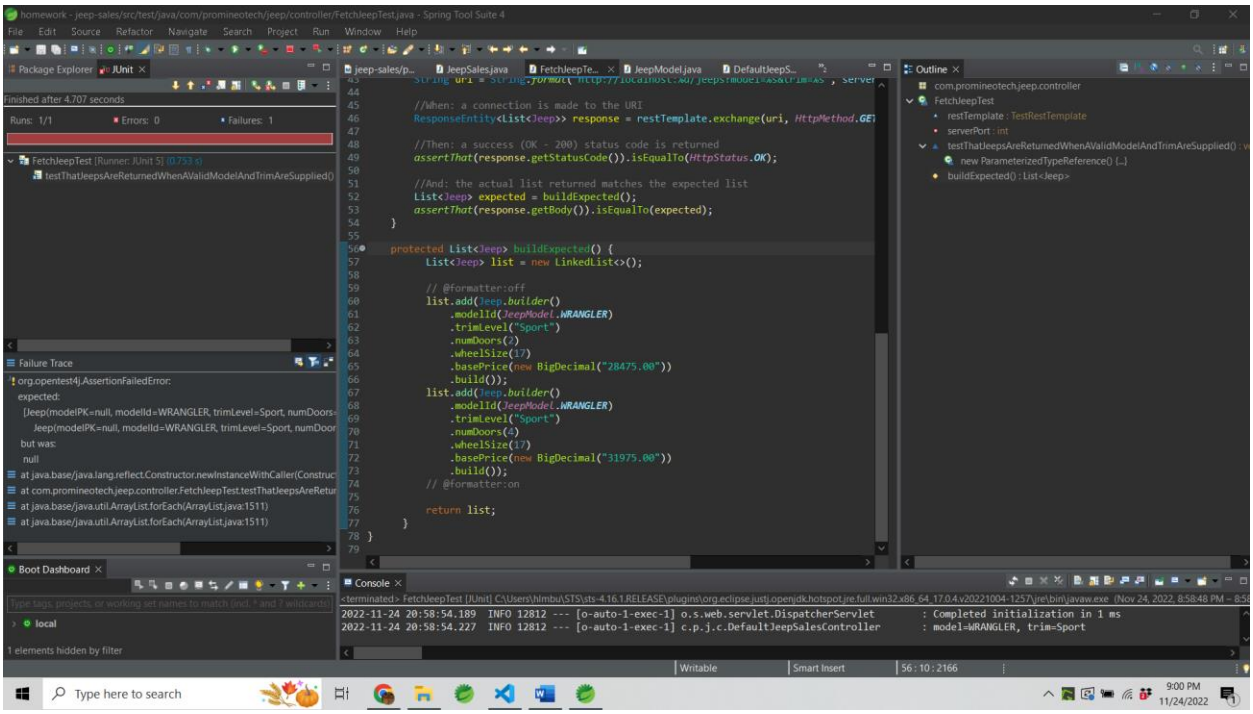
The method should be named `buildExpected()`, and it should return a `List` of `Jeep`. The video put this method into a support superclass but you can include it in the main test class if you want.

7) 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. Produce a screenshot showing…

   a) The test with the assertion.

   b) The JUnit status bar (should be red).

   c) The method returning the expected list of Jeeps.



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

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

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

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

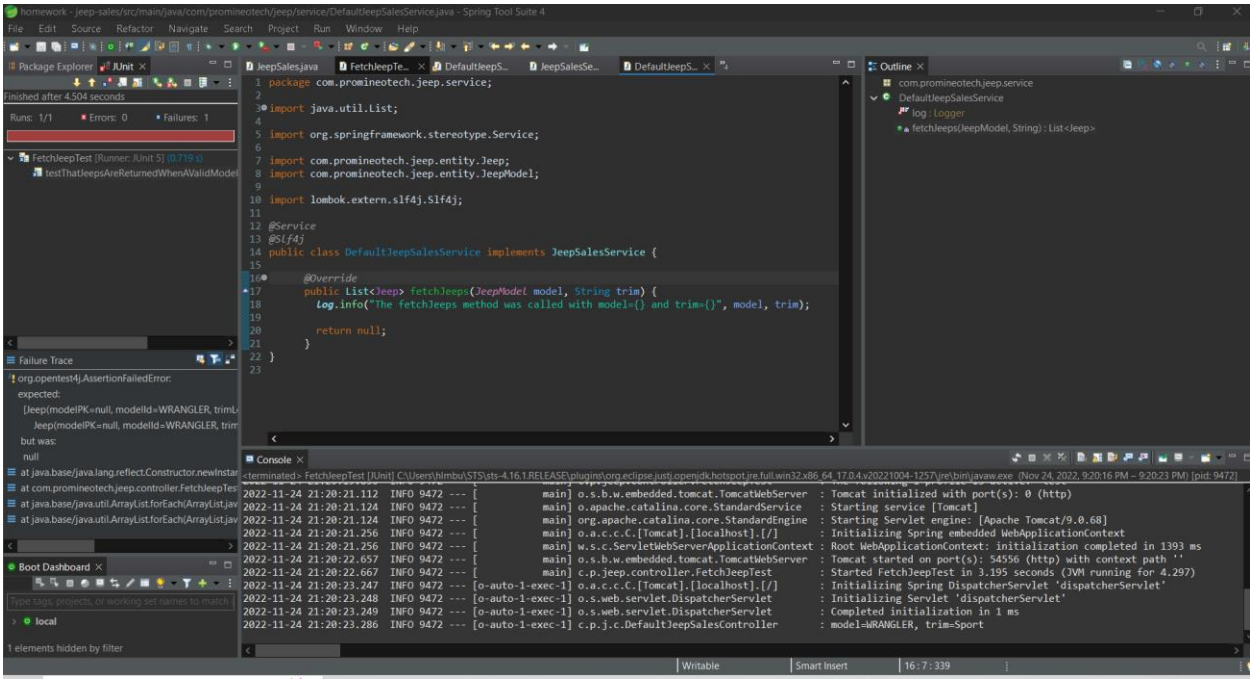
d) Inject the service interface into `DefaultJeepSalesController` using the `@Autowired` annotation. The instance variable should be `private`, and the variable should be named `jeepSalesService`.

e) 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);
```

f) 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.

g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar.



9) Add the database dependencies described in the video to the POM file (MySQL driver and Spring Boot Starter JDBC). To find them, nagivate 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.

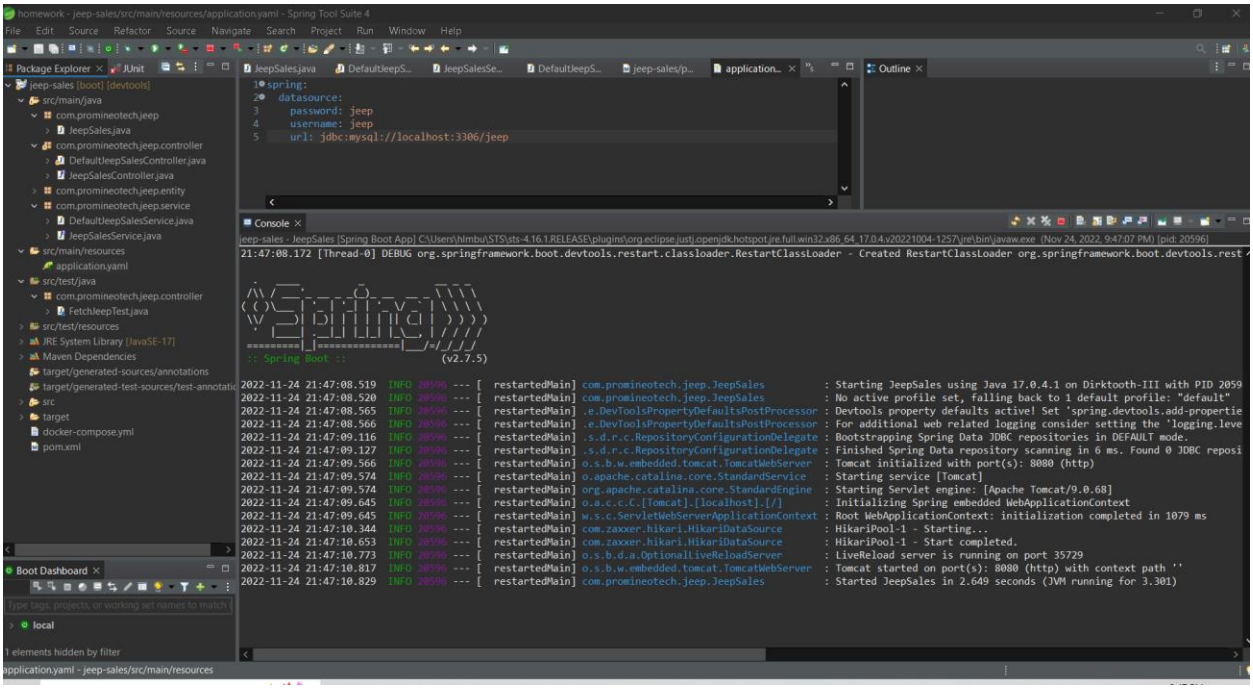
10) Create `application.yaml` in `src/main/resources`. Add the `spring.datasource.url`, `spring.datasource.username`, and `spring.datasource.password` properties to `application.yaml`. The url should be the same as shown in the video (`jdbc:mysql://localhost:3306/jeep`). The password and username should 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
```

11) Start the application (the real application, not the test). Produce a screenshot that shows `application.yaml` and the console showing that the application has started with no errors.



12) 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".

13) 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.

Produce a screenshot showing `application-test.yaml`.