

# Web API Design with Spring Boot Week 14 Coding Assignment

Points possible: 75


URL to GitHub Repository: <https://github.com/DavidSteffen1/weekFourteenRepository/>

URL to Public Link of your Video: <https://youtu.be/9KupaB8Jkn8>


---

## 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.
-

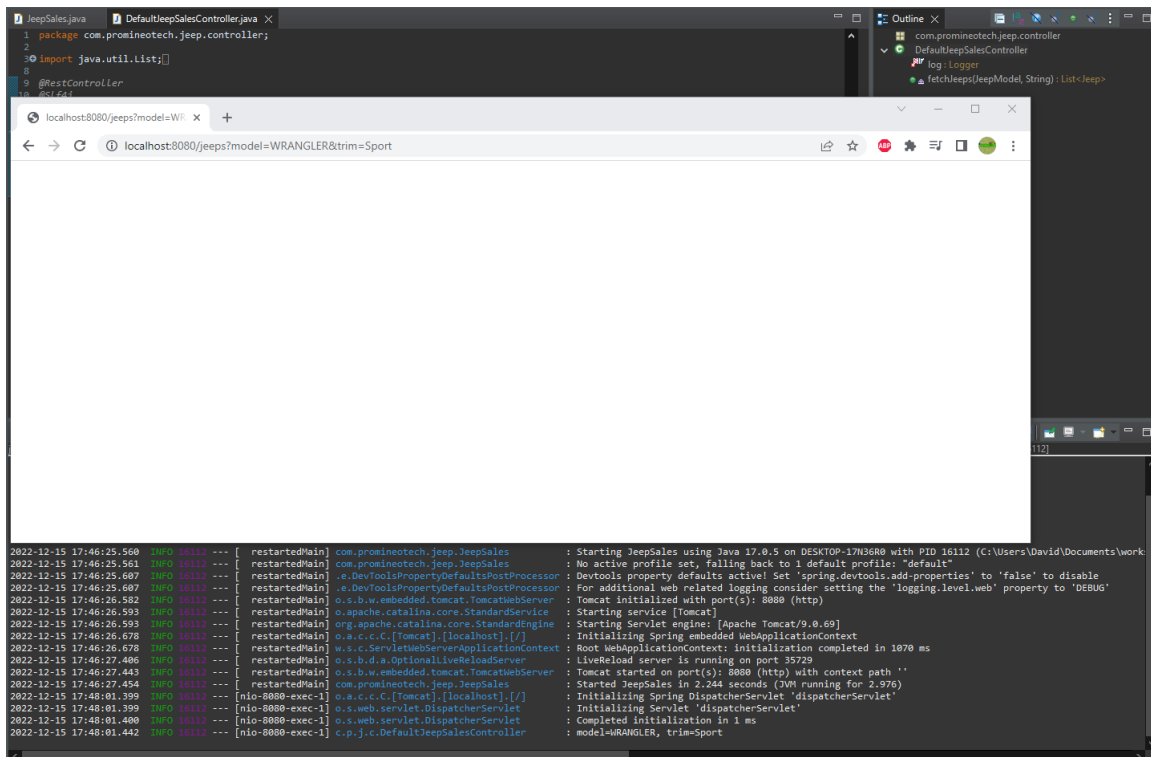
# Web API Design with Spring Boot Week 14 Coding Assignment

**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:

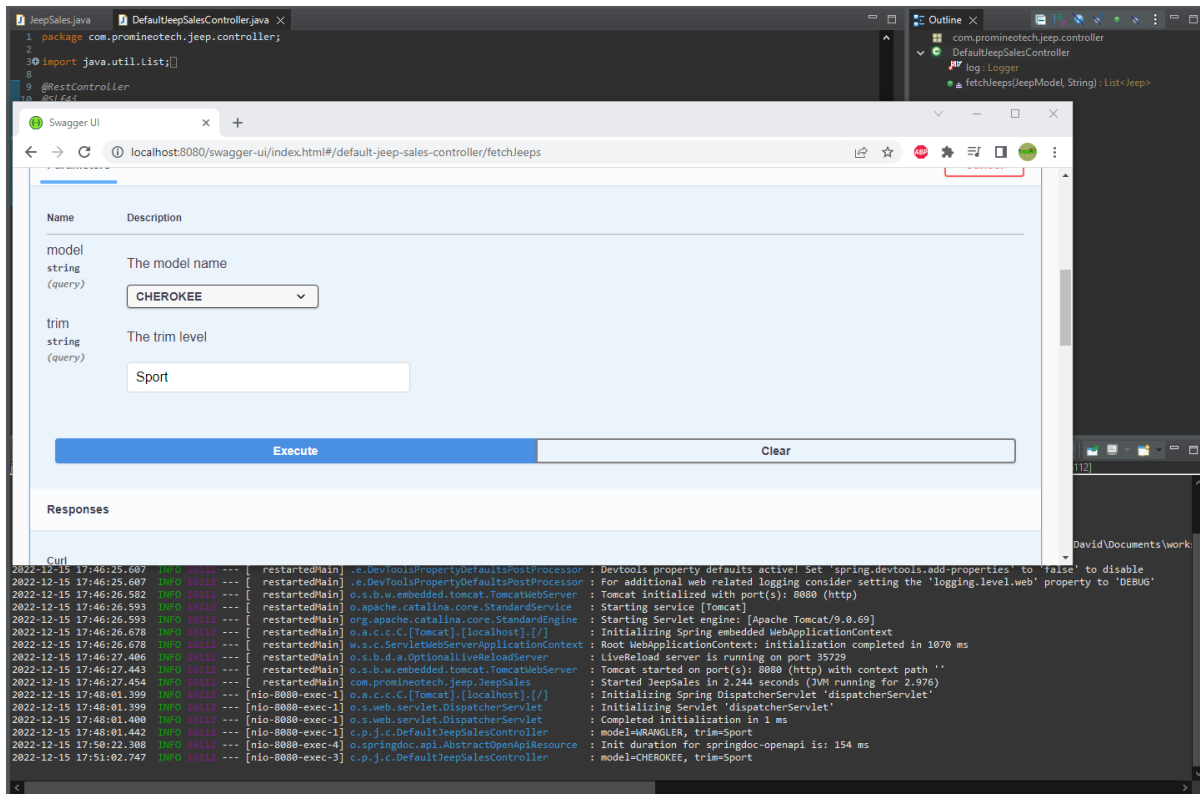
- 1) In the project you started last week, use Lombok to add an info-level logging statement in the controller implementation method that logs the parameters that were input to the method. Remember to add the `@Slf4j` annotation to the class.
- 2) Start the application (not an integration test). Use 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). 



The screenshot shows an IDE with two main windows. The top window displays the `DefaultJeepSalesController.java` file. The code includes the package `com.promineotech.jeepp.controller`, imports `java.util.List` and `@RestController`, and defines a class `DefaultJeepSalesController` with a `fetchJeeps` method. The bottom window shows a browser at `localhost:8080/jeeps?model=WRANGLER&trim=Sport`. The IDE console at the bottom shows the application's startup logs, including the message `Starting JeepSales using Java 17.0.5 on DESKTOP-17N36R8 with PID 16112` and the `fetchJeeps` method being invoked.

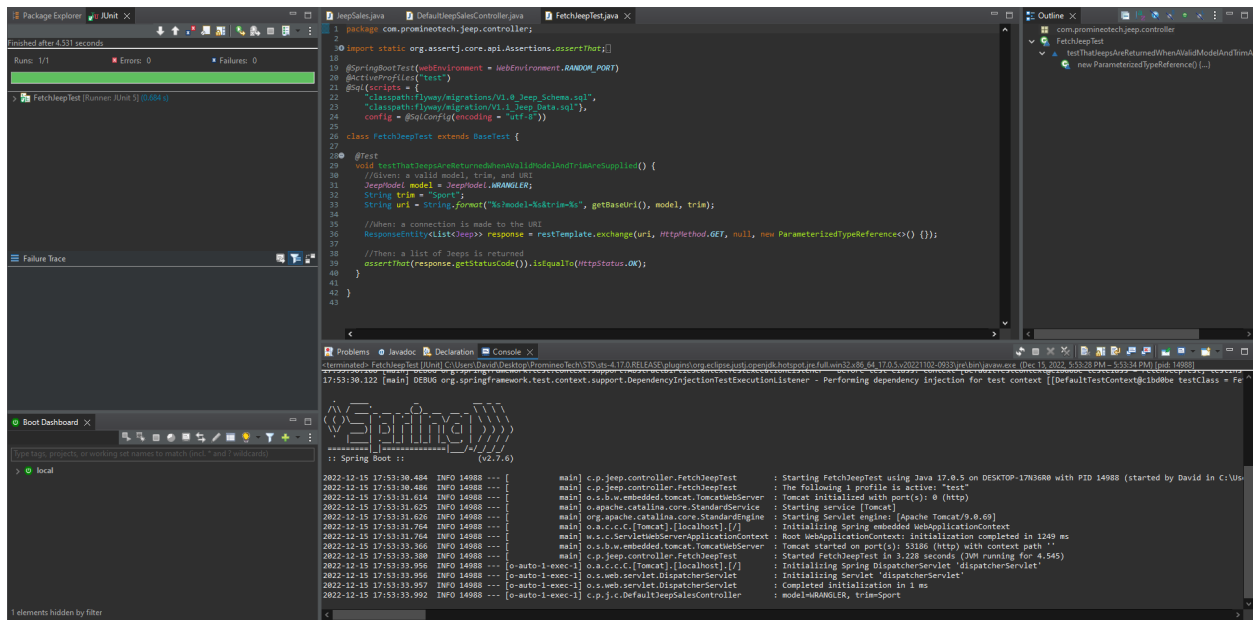
# Web API Design with Spring Boot Week 14 Coding Assignment

- 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.



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

# Web API Design with Spring Boot Week 14 Coding Assignment




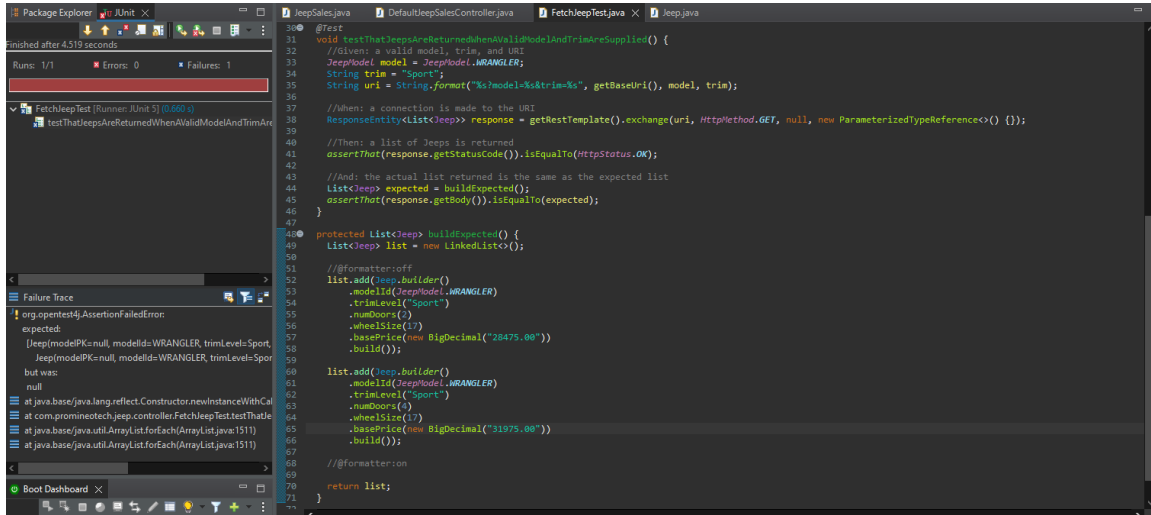
- 5) Add a method to the test to return a list of expected Jeep (model) objects based on the model and trim level you selected. You can get the expected list of Jeeps from the file `src/test/resources/flyway/migrations/V1.1__Jeep_Data.sql`. So, for example, using the model Wrangler and trim level "Sport", 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

- 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.
- 2) 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...

## Web API Design with Spring Boot Week 14 Coding Assignment

- The test with the assertion.
- The JUnit status bar (should be red).
- The method returning the expected list of Jeeps. 




```
30 @Test
31 void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
32     //Given: a valid model, trim, and URI
33     JeepModel model = JeepModel.WRANGLER;
34     String trim = "Sport";
35     String uri = String.format("%s?model=%s&trim=%s", getBaseUrl(), model, trim);
36
37     //When: a connection is made to the URI
38     ResponseEntity<List<Jeep>> response = getRestTemplate().exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
39
40     //Then: a list of Jeeps is returned
41     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
42
43     //And: the actual list returned is the same as the expected list
44     List<Jeep> expected = buildExpected();
45     assertThat(response.getBody()).isEqualTo(expected);
46 }
47
48 protected List<Jeep> buildExpected() {
49     List<Jeep> list = new LinkedList<>();
50
51     //formatter:off
52     list.add(Jeep.builder()
53         .modelId(JeepModel.WRANGLER)
54         .trimLevel("Sport")
55         .numDoors(2)
56         .wheelSize(17)
57         .basePrice(new BigDecimal("28475.00"))
58         .build());
59     list.add(Jeep.builder()
60         .modelId(JeepModel.WRANGLER)
61         .trimLevel("Sport")
62         .numDoors(4)
63         .wheelSize(17)
64         .basePrice(new BigDecimal("31975.00"))
65         .build());
66     //formatter:on
67
68     return list;
69 }
70
71 }
```

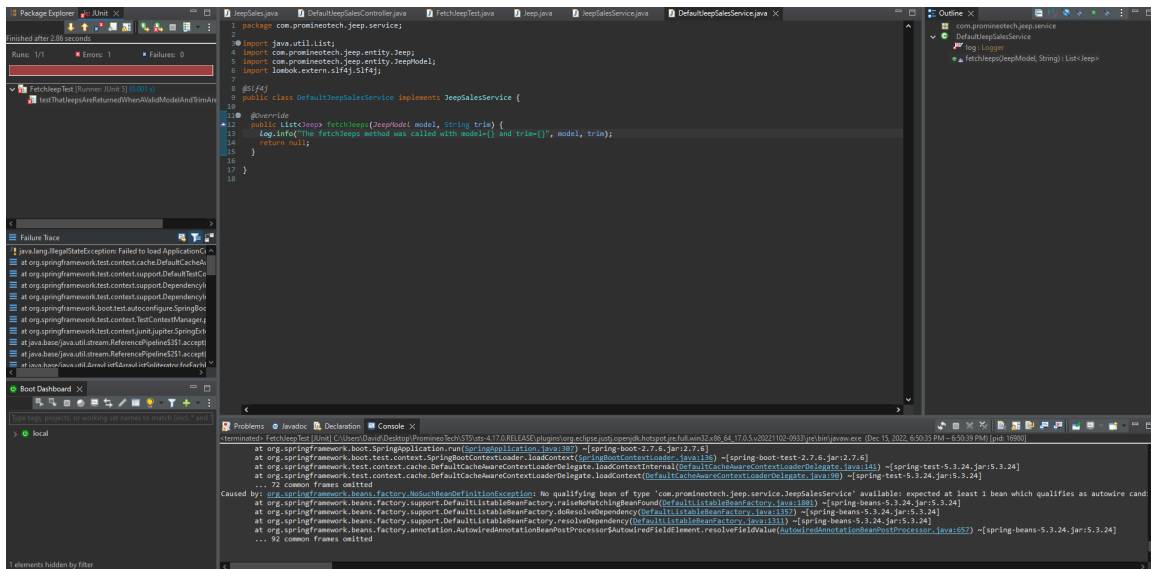
Failure trace:

```
org.opentest4j.AssertionFailedError:
  expected:
    JeepModelPK=null, modelId=WRANGLER, trimLevel=Sport
  but was:
    null
  at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:491)
  at com.promineotech.jeeptest.FetchJeepTestThatLe...
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

- Add a service layer in your application as shown in the videos:
  - Add a package named `com.promineotech.jeeptest.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. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar. 

# Web API Design with Spring Boot Week 14 Coding Assignment



- 7) 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.
- 8) 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:

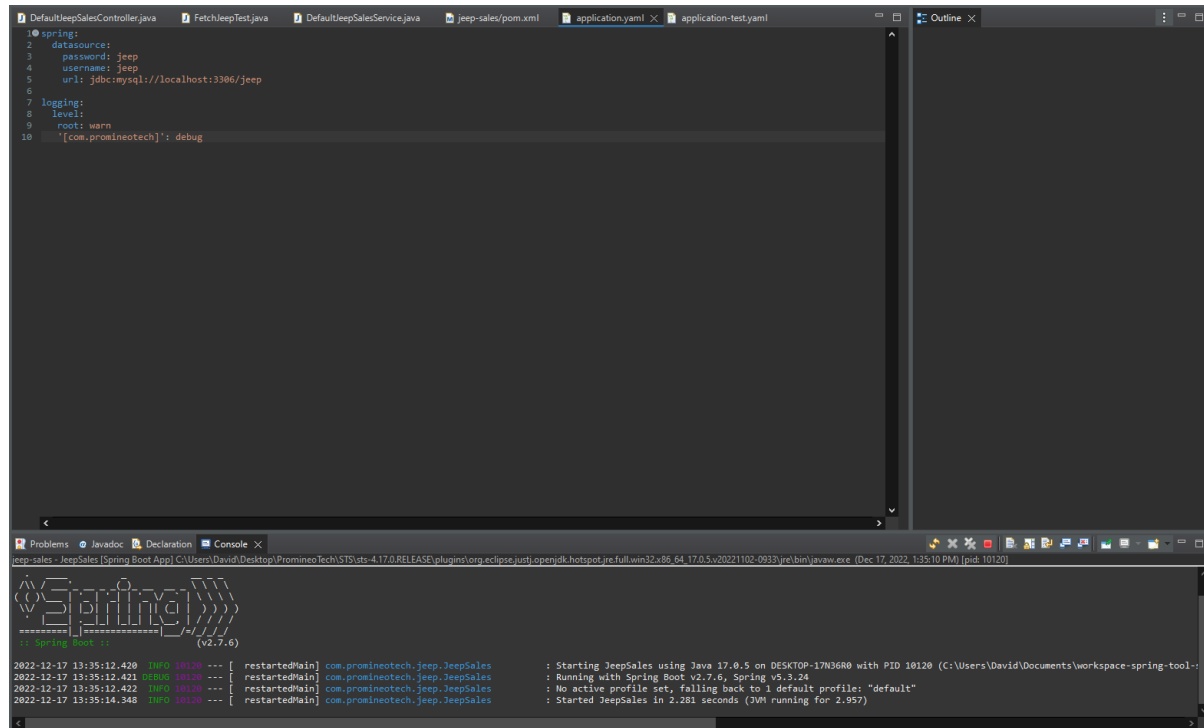
    username: username

    password: password

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

# Web API Design with Spring Boot Week 14 Coding Assignment

- 9) 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. 🖥️



The screenshot shows an IDE with the following files open: DefaultJeepSalesController.java, FetchJeepTest.java, DefaultJeepSalesService.java, jeep-sales/pom.xml, application.yaml, and application-test.yaml. The application.yaml file is selected and shows the following content:

```
1 spring:
2   datasource:
3     password: jeep
4     username: jeep
5     url: jdbc:mysql://localhost:3306/jeep
6
7 logging:
8   level:
9     root: warn
10    '[com.promineotech]': debug
```

The console output at the bottom shows the application starting successfully:

```
2022-12-17 13:35:12.420 [INFO] 10120 --- [ restarted@main] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 17.0.5 on DESKTOP-17N36R0 with PID 10120 (C:\Users\David\Documents\workspace-spring-tool-
2022-12-17 13:35:12.421 [INFO] 10120 --- [ restarted@main] com.promineotech.jeep.JeepSales : Running with Spring Boot v2.7.6, Spring v5.3.24
2022-12-17 13:35:12.422 [INFO] 10120 --- [ restarted@main] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default profile: "default"
2022-12-17 13:35:14.348 [INFO] 10120 --- [ restarted@main] com.promineotech.jeep.JeepSales : Started JeepSales in 2.281 seconds (Tom running for 2.957)
```

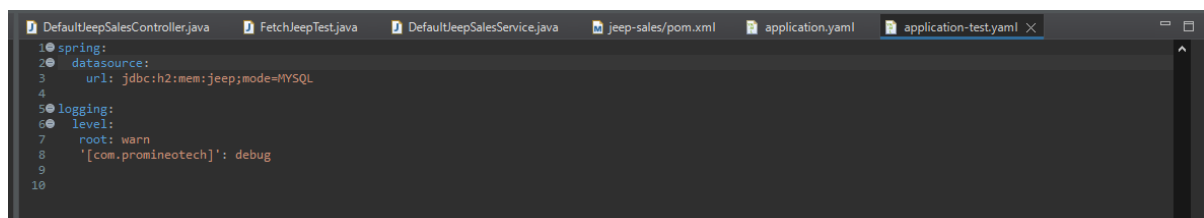
- 10) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".

- 11) 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. 🖥️



The screenshot shows the application-test.yaml file in the IDE. The content is as follows:

```
1 spring:
2   datasource:
3     url: jdbc:h2:mem:jeep;mode=MYSQL
4
5 logging:
6   level:
7     root: warn
8     '[com.promineotech]': debug
9
10
```