

Web API Design with Spring Boot Week 14 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/admmoore21/JeepSalesAPI>


URL to Public Link of your Video: https://www.youtube.com/watch?v=-V4oTB6_pj4

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.
-

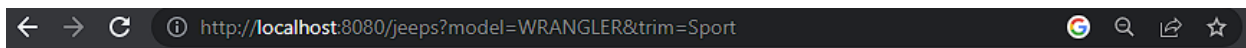
Web API Design with Spring Boot Week 14 Coding Assignment

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). 



```
2023-01-09 17:05:58.987 INFO 24544 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-01-09 17:05:58.994 INFO 24544 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Started JeeppSales in 1.702 seconds (JVM running for 2.443)
2023-01-09 17:08:17.490 INFO 24544 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-01-09 17:08:17.490 INFO 24544 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-01-09 17:08:17.491 INFO 24544 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2023-01-09 17:08:17.524 INFO 24544 --- [nio-8080-exec-2] c.p.j.c.DefaultJeeppSalesController : model=WRANGLER, trim=Sport
```

- 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

GET

/jeeps Returns a list of Jeeps

⌵

Returns a list of Jeeps given an optional model and/or trim

Parameters

Cancel

Name	Description
model <small>* required</small> string <small>(query)</small>	The model name (i.e., 'WRANGLER') <div>WRANGLER</div>
trim <small>* required</small> string <small>(query)</small>	The trim level (i.e., 'Sport') <div>Sport</div>

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/jeeps?model=WRANGLER&trim=Sport' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8080/jeeps?model=WRANGLER&trim=Sport
```

Server response

Code	Details
200	<div>Response headers</div> <div><pre>connection: keep-alive content-length: 0 date: Mon, 09 Jan 2023 22:54:19 GMT keep-alive: timeout=60</pre></div>

Responses

Code	Description	Links
200	<div>A list of Jeeps is returned</div> <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "modelId": 0, "modelId": "WRANGLER", "trimLevel": "string", "numDoors": 0, "wheelSize": 0, "basePrice": 0 }</pre></div>	No links
400	<div>The request parameters are invalid</div> <div>Media type</div> <div>application/json</div>	No links
404	<div>No Jeeps were found with the input criteria</div> <div>Media type</div> <div>application/json</div>	No links
500	<div>An unplanned error occurred</div> <div>Media type</div> <div>application/json</div>	No links

Schemas

⌵

Jeep >

Web API Design with Spring Boot Week 14 Coding Assignment

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

The screenshot shows an IDE with the JUnit test runner on the left and the Java source code on the right. The test runner indicates that the test 'FetchJeepTest' passed successfully after 3.601 seconds, with 1/1 runs, 0 errors, and 0 failures. The status bar at the bottom is green. The Java code defines a test class 'FetchJeepTest' that uses Spring Boot's test annotations and a REST client to verify that a valid model and trim return a 200 OK response.

```
1 package com.promineotech.jeepp.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
6 @ActiveProfiles("test")
7 @Sql(scripts = {
8     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
9     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"
10 }, config = @SqlConfig(encoding = "utf-8"))
11
12 class FetchJeepTest {
13     @Autowired
14     private TestRestTemplate restTemplate;
15     @LocalServerPort
16     private int serverPort;
17
18     @Test
19     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
20         //Given: A valid model trim and URI
21         JeepModel model = JeepModel.WRANGLER;
22         String trim = "Sport";
23         String uri =
24             String.format("http://localhost:%d/jeeps?model=%s&trim=%s",
25                 serverPort, model, trim);
26
27         //When: A connection is made to the URI
28         ResponseEntity<List<Jeep>> response =
29             restTemplate.exchange(uri, HttpMethod.GET, null,
30                 new ParameterizedTypeReference<>() {});
31
32         //Then: a success (OK - 200) is returned
33         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
34     }
35 }
```

- 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

Web API Design with Spring Boot Week 14 Coding Assignment

Base Price	\$28,475.00	\$31,975.00
------------	-------------	-------------


6)

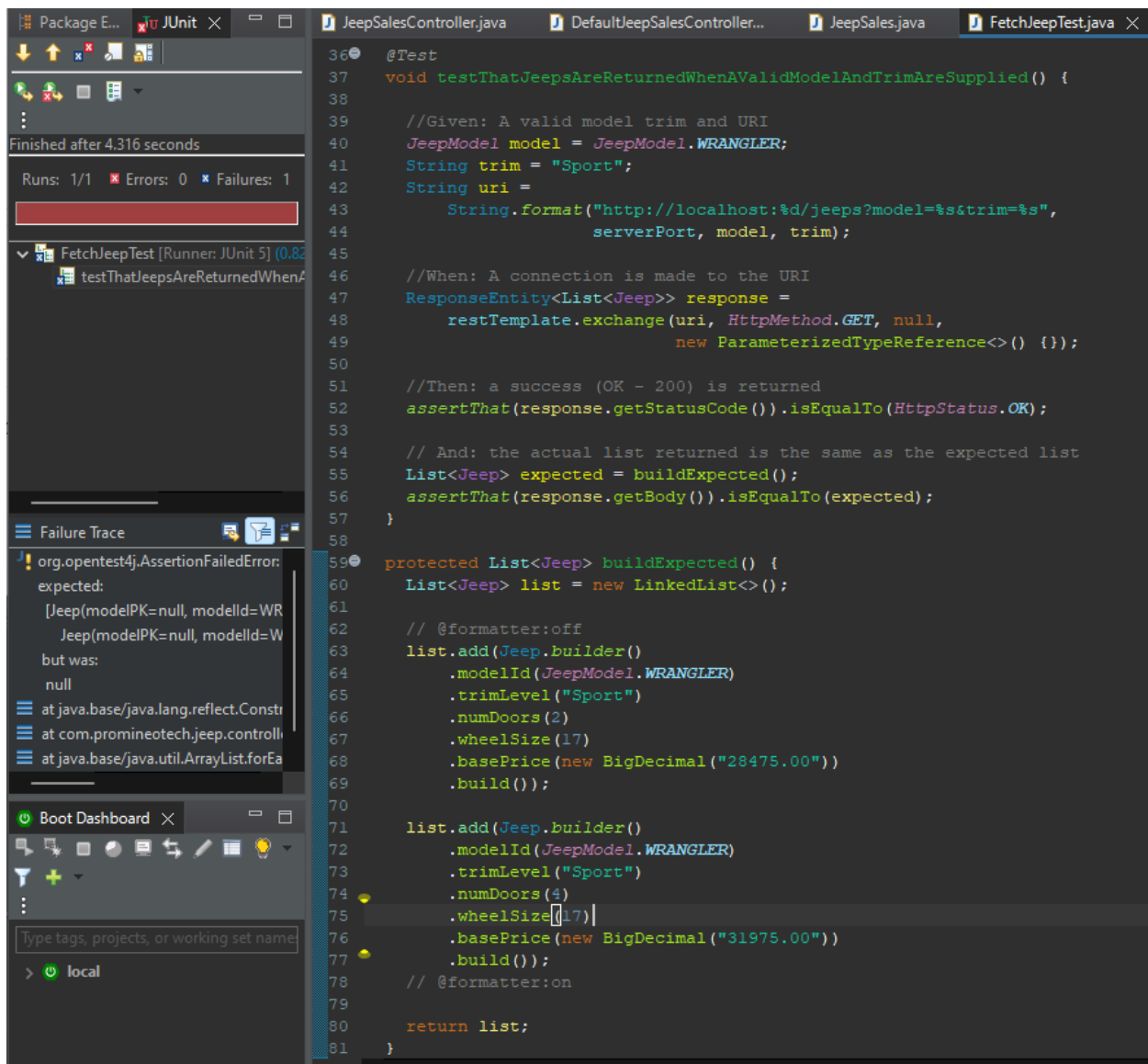
The method should be named `buildExpected()`, and it should return a `List` of `Jeep`. The video put this method into a support superclass but you can include it in the main test class if you want.

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. 



```
36 @Test
37 void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
38
39     //Given: A valid model trim and URI
40     JeepModel model = JeepModel.WRANGLER;
41     String trim = "Sport";
42     String uri =
43         String.format("http://localhost:%d/jeeps?model=%s&trim=%s",
44             serverPort, model, trim);
45
46     //When: A connection is made to the URI
47     ResponseEntity<List<Jeep>> response =
48         restTemplate.exchange(uri, HttpMethod.GET, null,
49             new ParameterizedTypeReference<>() {});
50
51     //Then: a success (OK - 200) is returned
52     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
53
54     // And: the actual list returned is the same as the expected list
55     List<Jeep> expected = buildExpected();
56     assertThat(response.getBody()).isEqualTo(expected);
57 }
58
59 protected List<Jeep> buildExpected() {
60     List<Jeep> list = new LinkedList<>();
61
62     // @formatter:off
63     list.add(Jeep.builder()
64         .modelId(JeepModel.WRANGLER)
65         .trimLevel("Sport")
66         .numDoors(2)
67         .wheelSize(17)
68         .basePrice(new BigDecimal("28475.00"))
69         .build());
70
71     list.add(Jeep.builder()
72         .modelId(JeepModel.WRANGLER)
73         .trimLevel("Sport")
74         .numDoors(4)
75         .wheelSize(17)
76         .basePrice(new BigDecimal("31975.00"))
77         .build());
78     // @formatter:on
79
80     return list;
81 }
```

Finished after 4.316 seconds

Runs: 1/1 Errors: 0 Failures: 1

Failure Trace

```
org.opentest4j.AssertionFailedError:
expected:
  [Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=17, basePrice=28475.00),
   Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoors=4, wheelSize=17, basePrice=31975.00)]
but was:
  null
at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:490)
at com.promineotech.jeeptest.FetchJeepTest.testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied(FetchJeepTest.java:56)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

Boot Dashboard

Type tags, projects, or working set names

> local

Web API Design with Spring Boot Week 14 Coding Assignment

- 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.

The screenshot shows an IDE with several tabs open: `DefaultJeepSalesController.java`, `JeepSales.java`, `FetchJeepTest.java`, `Jeep.java`, `JeepSalesService.java`, and `DefaultJeepSalesService.java`. The `DefaultJeepSalesService.java` tab is active, showing the following code:

```
1 package com.promineotech.jeep.service;
2
3 import java.util.List;
4 import org.springframework.stereotype.Service;
5 import com.promineotech.jeep.entity.Jeep;
6 import com.promineotech.jeep.entity.JeepModel;
7 import lombok.extern.slf4j.Slf4j;
8
9 @Service
10 @Slf4j
11 public class DefaultJeepSalesService implements JeepSalesService {
12
13     @Override
14     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
15         log.info("The fetchJeeps method was called with model={} and trim={}",
16             model, trim);
17         return null;
18     }
19 }
20
21
```

The console output at the bottom shows the test run results:


```
31004 --- [main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.5 on AlienAdam with PID 31004 (started by wrest in
31004 --- [main] c.p.jeep.controller.FetchJeepTest : Running with Spring Boot v2.7.6, Spring v5.3.24
31004 --- [main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"
31004 --- [main] c.p.jeep.controller.FetchJeepTest : Started FetchJeepTest in 2.749 seconds (JVM running for 3.584)
31004 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
31004 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim=Sport
```

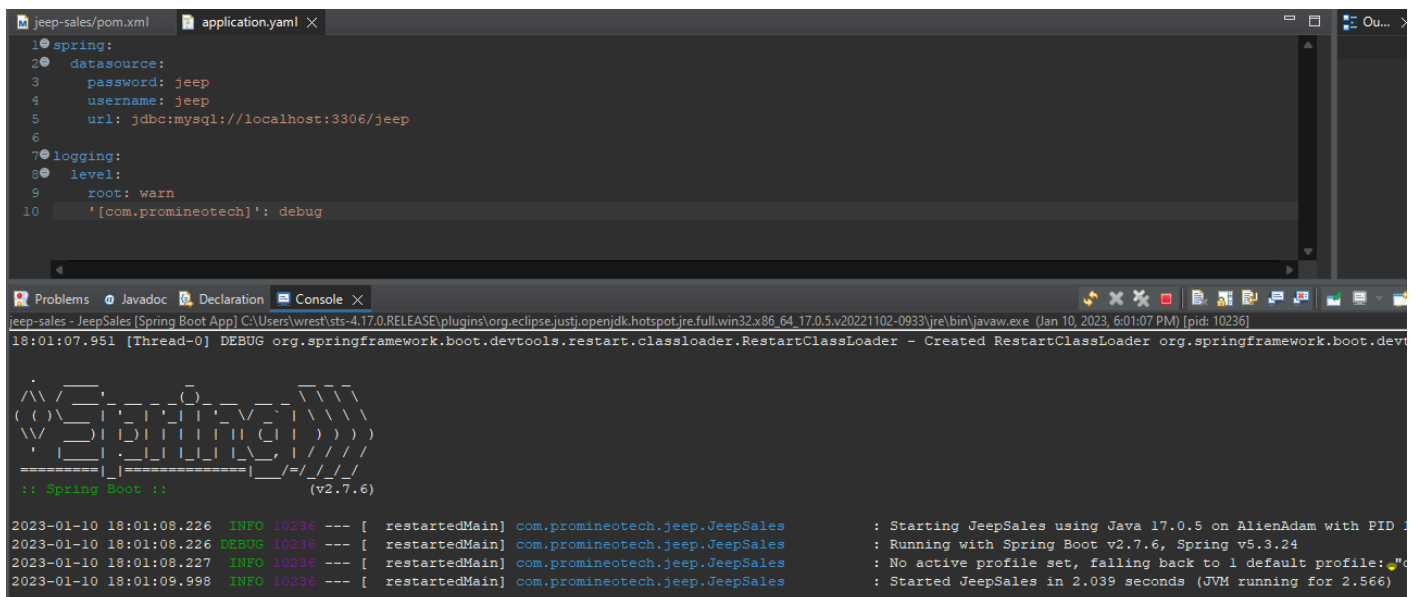
Web API Design with Spring Boot Week 14 Coding Assignment

- 9) 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.
- 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. 



The screenshot shows an IDE with two tabs: `jeep-sales/pom.xml` and `application.yaml`. The `application.yaml` file contains the following configuration:

```
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 shows the application starting successfully:

```
jeep-sales - JeepSales [Spring Boot App] C:\Users\wrest\sts-4.17.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.5.v20221102-0933\jre\bin\javaw.exe (Jan 10, 2023, 6:01:07 PM) [pid: 10236]
18:01:07.951 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.classloader.RestartClassLoader
[INFO] Starting Spring Boot 2.7.6
[INFO] Starting JeepSales using Java 17.0.5 on AlienAdam with PID 10236
[INFO] Running with Spring Boot v2.7.6, Spring v5.3.24
[INFO] No active profile set, falling back to 1 default profile: "default"
[INFO] Started JeepSales in 2.039 seconds (JVM running for 2.566)
```

Web API Design with Spring Boot Week 14 Coding Assignment

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:

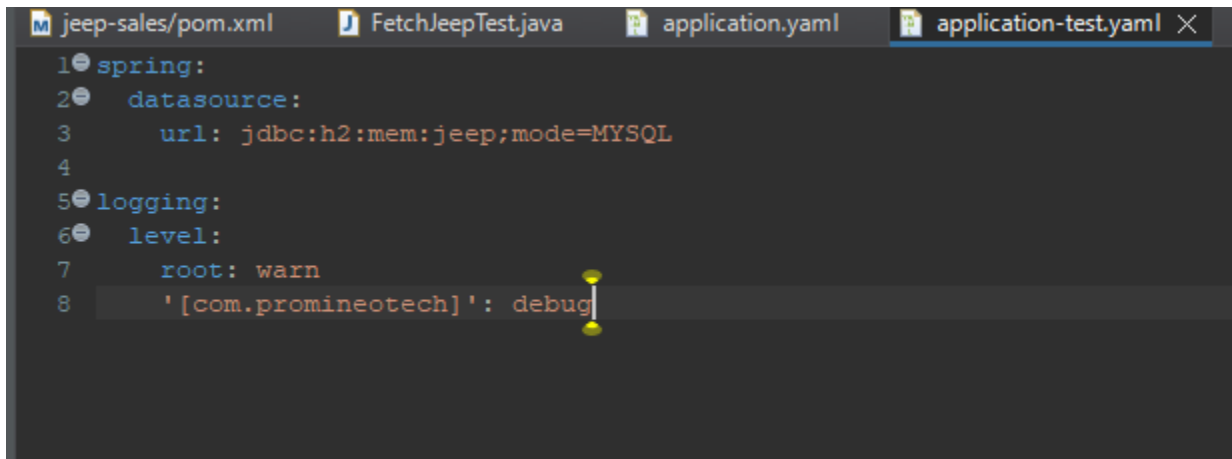
spring:

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. 🖥️



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