

Web API Design with Spring Boot Week 2 Coding Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

Instructions: In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document, with your Java project code, to the repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

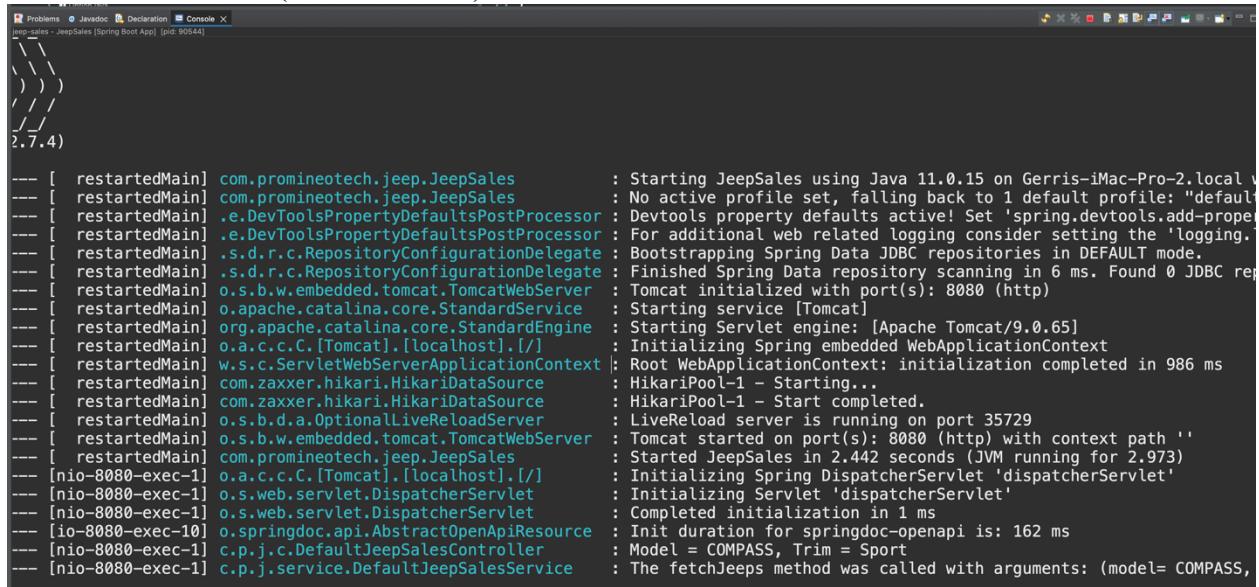
Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon: You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a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



```
jeep-sales - JeepSales [Spring Boot App] [pid: 90544]
\ \ \
) ) )
/ /
2.7.4)

--- [ restartedMain] com.promineotech.jeep.JeepSales      : Starting JeepSales using Java 11.0.15 on Gerris-iMac-Pro-2.local v
--- [ restartedMain] com.promineotech.jeep.JeepSales      : No active profile set, falling back to 1 default profile: "default"
--- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties=true' for additional web related logging consider setting the 'logging.level.web=DEBUG'
--- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
--- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 6 ms. Found 0 JDBC repositories
--- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
--- [ restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
--- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
--- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
--- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 986 ms
--- [ restartedMain] com.zaxxer.hikari.HikariDataSource    : HikariPool-1 - Starting...
--- [ restartedMain] com.zaxxer.hikari.HikariDataSource    : HikariPool-1 - Start completed.
--- [ restartedMain] o.s.b.d.a.OptionalAllaliveReloadServer : LiveReload server is running on port 35729
--- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
--- [ restartedMain] com.promineotech.jeep.JeepSales      : Started JeepSales in 2.442 seconds (JVM running for 2.973)
--- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
--- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   : Initializing Servlet 'dispatcherServlet'
--- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   : Completed initialization in 1 ms
--- [io-8080-exec-10] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 162 ms
--- [nio-8080-exec-1] c.p.j.c.DefaultJeepSalesController : Model = COMPASS, Trim = Sport
--- [nio-8080-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with arguments: (model= COMPASS,
```

- 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. 

default-jeep-sales-controller

GET /jeeps Returns a list of Jeeps

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

Parameters

Name Description

model string (query) COMPASS

trim string (query) Sport

Execute Clear

Responses

Curl

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

Request URL

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

Server response

Code Details

200 Response headers

```
connection: keep-alive
content-length: 0
date: Thu, 29 Sep 2022 14:59:19 GMT
keep-alive: timeout=60
```

Responses

Code Description Links

200 No links

- 4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar.  I am already done. Coding along with the videos. An red status at the end.
- 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:

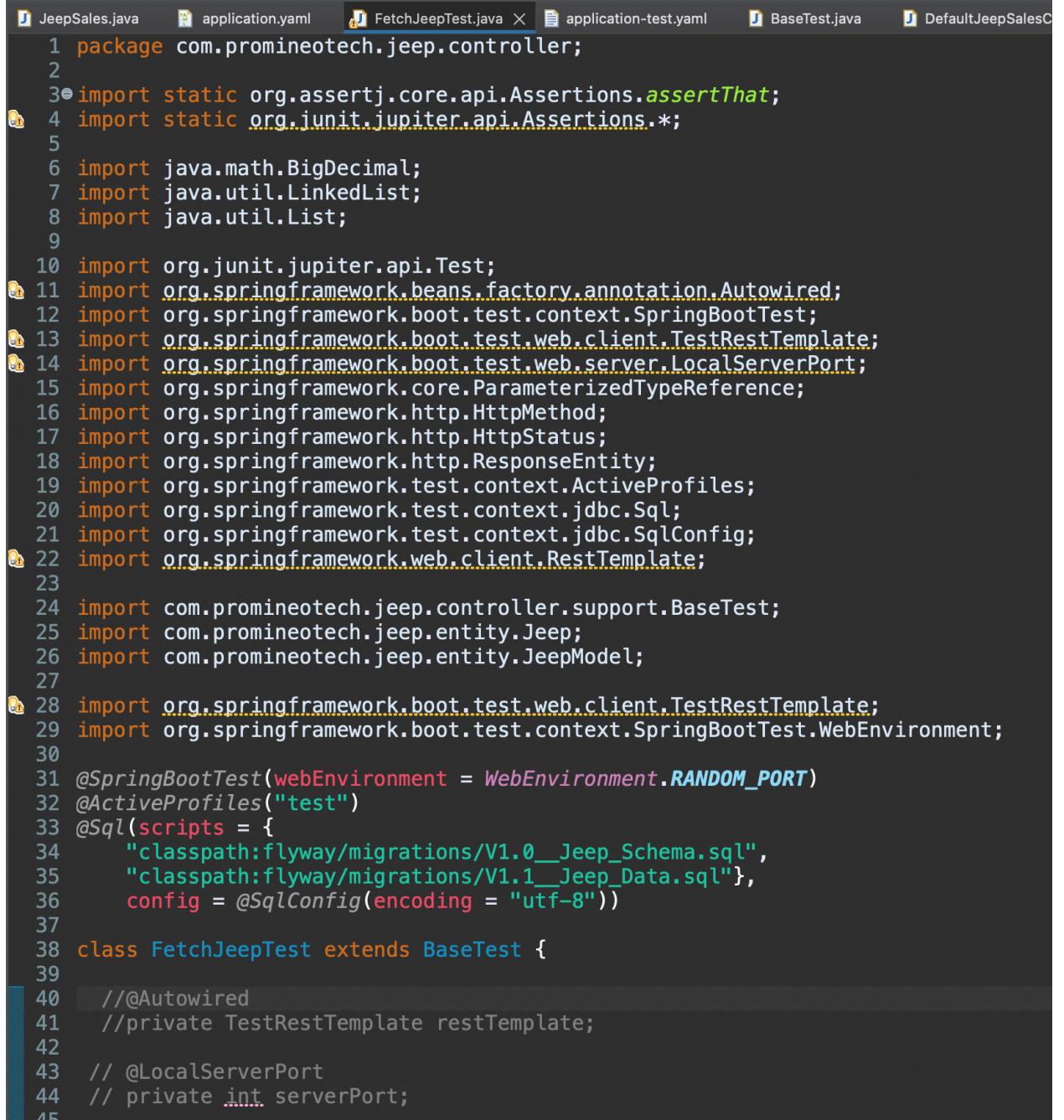
	Row 1	Row 2
Model ID	WRANGLER	WRANGLER

Trim Level	Sport	Sport
Num Doors	2	4
Wheel Size	17	17
Base Price	\$28,475.00	\$31,975.00

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.

- 6) 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.



The screenshot shows a Java IDE interface with a code editor containing Java test code. The code is annotated with line numbers from 1 to 45. The imports section includes various Spring Boot and JUnit dependencies. The class definition starts with a test annotation and extends a base test class. It contains several blank lines and two commented-out sections, likely representing code that was either removed or is part of a larger function.

```
1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 import java.math.BigDecimal;
7 import java.util.LinkedList;
8 import java.util.List;
9
10 import org.junit.jupiter.api.Test;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.boot.test.context.SpringBootTest;
13 import org.springframework.boot.test.web.client.TestRestTemplate;
14 import org.springframework.boot.test.web.server.LocalServerPort;
15 import org.springframework.core.ParameterizedTypeReference;
16 import org.springframework.http.HttpMethod;
17 import org.springframework.http.HttpStatus;
18 import org.springframework.http.ResponseEntity;
19 import org.springframework.test.context.ActiveProfiles;
20 import org.springframework.test.context.jdbc.Sql;
21 import org.springframework.test.context.jdbc.SqlConfig;
22 import org.springframework.web.client.RestTemplate;
23
24 import com.promineotech.jeep.controller.support.BaseTest;
25 import com.promineotech.jeep.entity.Jeep;
26 import com.promineotech.jeep.entity.JeepModel;
27
28 import org.springframework.boot.test.web.client.TestRestTemplate;
29 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
30
31 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
32 @ActiveProfiles("test")
33 @Sql(scripts = {
34     "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
35     "classpath:flyway/migrations/V1.1_Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
36
37
38 class FetchJeepTest extends BaseTest {
39
40     //@Autowired
41     //private TestRestTemplate restTemplate;
42
43     // @LocalServerPort
44     // private int serverPort;
```

```

11 // private static SERVER;
45
46● @Test
47 void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
48
49     JeepModel model = JeepModel.WRANGLER;
50     String trim = "Sport";
51     String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);
52
53     ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri,
54     HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
55     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
56     List<Jeep> expected = buildExpected();
57     assertThat(response.getBody()).isEqualTo(expected);
58
59     // ResponseEntity<List<Jeep>> response = getRestTemplate().exchange(uri,
60     // HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
61     // assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
62     // List<Jeep> expected = buildExpected();
63
64     // ResponseEntity<Jeep> response =
65     //     getRestTemplate().getForEntity(uri, Jeep.class);
66     // assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
67
68     // List<Jeep> expected = buildExpected();
69     // assertThat(response.getBody()).isEqualTo(expected);
70
71 }
72
73
74● List<Jeep> buildExpected(){
75     List<Jeep> list = new LinkedList<>();
76
77     list.add(Jeep.builder()
78             .modelId(JeepModel.WRANGLER)
79             .trimLevel("Freedom")
80             .numDoors(2)
81             .wheelSize(17)
82             .basePrice(new BigDecimal ("28474.00"))
83             .build());
84
85     list.add(Jeep.builder().modelId(JeepModel.WRANGLER)
86             .trimLevel("Freedom")
87             .numDoors(4)
88             .wheelSize(17)
89             .basePrice(new BigDecimal ("31975.00"))
90             .build());
91
92     return list;
93
94 }
95

```

7) 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`.

- 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)

- g) 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.

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

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

finished after 4.41 seconds

Runs: 1/1 Errors: 0 Failures: 1

FetchJeepTest [Runner: JUnit 5] (0.740 s)

testThatJeepsAreReturnedWhenAValidModelIdIsProvided

Failure Trace

```
J! org.opentest4j.AssertionFailedError:  
expected:  
  Jeep(modelPK=null, modelId=WRANGLER, trimLevel=  
    Jeep(modelPK=null, modelId=WRANGLER, trimLevel  
but was:  
  null  
  at com.promineotech.jeep.controller.FetchJeepTest  
  at java.base/java.util.ArrayList.forEach(ArrayList.java:  
  at java.base/java.util.ArrayList.forEach(ArrayList.java:
```

Problems Javadoc Declaration Console

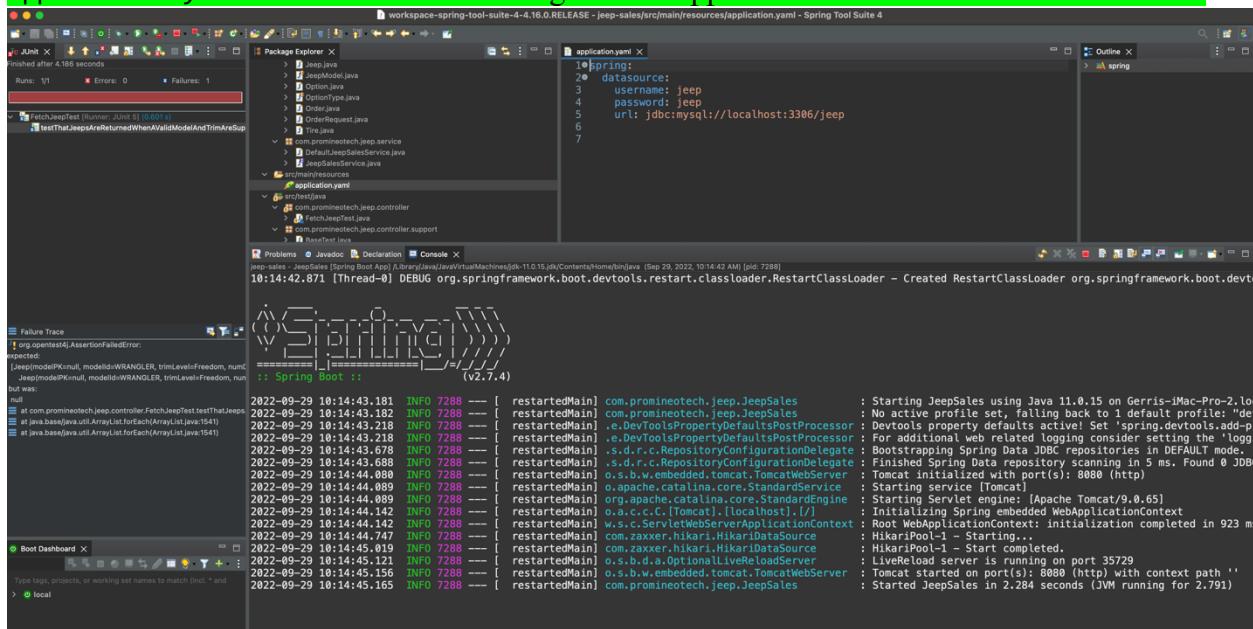
```
terminated> FetchJeepTest [JUnit] /Library/Jenkins/jobs/promineotech-jeep-test/workspace/target/test-classes/testThatJeepsAreReturnedWhenAValidModelIdIsProvided [Sep 29, 2023, 09:14:49 AM] [ID: 169] [Anil [pid: 4494]  
0:11:50.501 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Neither @ContextConfiguration nor @ContextHierarchy found for test class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.504 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.504 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.504 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource locations for test class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.619 [main] INFO org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate component class: file [users/gerricancanelli/Documents/workspace/FetchJeepTest/src/main/java/com/promineotech/jeep/controller/FetchJeepTest.java] (id: 1)  
0:11:50.629 [main] INFO org.springframework.test.context.SpringBootTestBootstrapper - Found @SpringBootTest annotated test class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.703 [main] DEBUG org.springframework.boot.test.context.SpringBootTestBootstrapper - @TestExecutionListeners is not present for class [com.promineotech.jeep.controller.FetchJeepTest]  
0:11:50.720 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Loaded default TestExecutionListener class names from location [META-INF/spring.factories]: [org.springframework.boot.test.context.SpringBootTestExecutionListener]  
0:11:50.724 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTestContext@3e7634b9 testClass = FetchJeepTest, testName = testThatJeepsAreReturnedWhenAValidModelIdIsProvided]  
0:11:50.734 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Performing dependency injection for test context [DefaultTestContext@3e7634b9 testClass = FetchJeepTest, testName = testThatJeepsAreReturnedWhenAValidModelIdIsProvided]  
  
Spring Boot :: (v2.7.4)  
  
09-29 10:11:51.063 INFO 4494 --- [ main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 11.0.15 on Gerris-iMac-Pro-2.local with PID 4494 (started by g  
09-29 10:11:51.064 INFO 4494 --- [ main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"  
09-29 10:11:51.568 INFO 4494 --- [ main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.  
09-29 10:11:51.581 INFO 4494 --- [ main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 7 ms. Found 0 JDBC repository interfaces.  
09-29 10:11:52.089 INFO 4494 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8 (http)  
09-29 10:11:52.090 INFO 4494 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 52764 (http) with context path ''  
09-29 10:11:52.090 INFO 4494 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Starting Servlet engine: [Apache Tomcat/9.0.65]  
09-29 10:11:52.187 INFO 4494 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext : Root WebApplicationContext: initialization completed in 1101 ms  
022-09-29 10:11:52.187 INFO 4494 --- [ main] o.w.s.c.ServletWebServerApplicationContext : HikariPool-1 - Starting..  
022-09-29 10:11:53.452 INFO 4494 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.  
022-09-29 10:11:53.710 INFO 4494 --- [ main] com.zaxxer.hikari.HikariDataSource : Tomcat started on port(s): 52764 (http) with context path ''  
022-09-29 10:11:53.913 INFO 4494 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Started FetchJeepTest in 3.167 seconds (JVM running for 4.018)  
022-09-29 10:11:53.923 INFO 4494 --- [ main] c.p.jeep.controller.FetchJeepTest : Initializing Spring DispatcherServlet 'dispatcherServlet'  
022-09-29 10:11:54.593 INFO 4494 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : DispatcherServlet 'dispatcherServlet' : Completed initialization in 1 ms  
022-09-29 10:11:54.594 INFO 4494 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Model = WRANGLER, Trim = Sport  
022-09-29 10:11:54.639 INFO 4494 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : The fetchJeeps method was called with arguments: (model= WRANGLER, trim= Sport)  
022-09-29 10:11:54.758 INFO 4494 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated..  
022-09-29 10:11:54.763 INFO 4494 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

- 8) 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.
- 9) 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
```

- 10) 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.



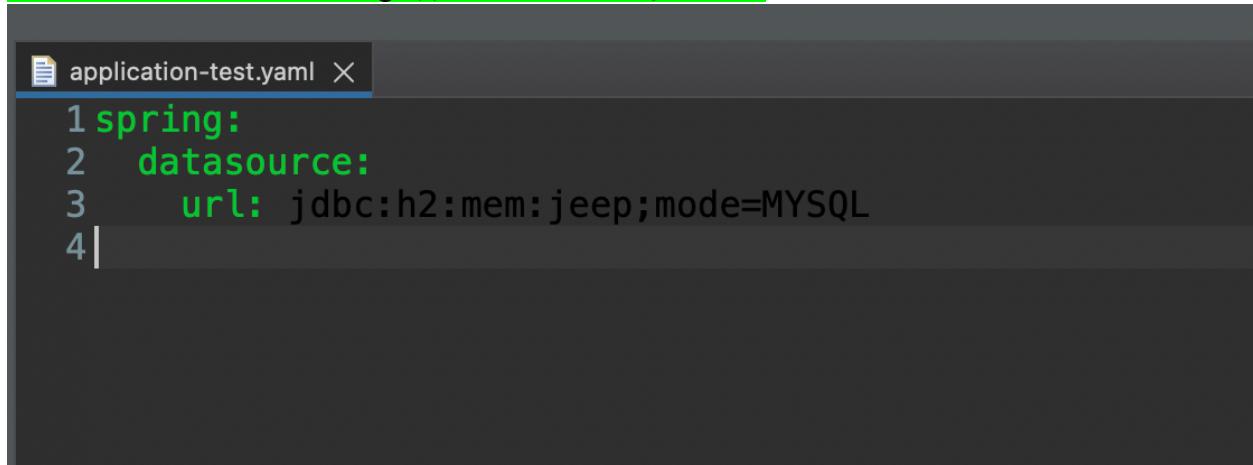
- 11) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".

- 12) 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
```

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. 



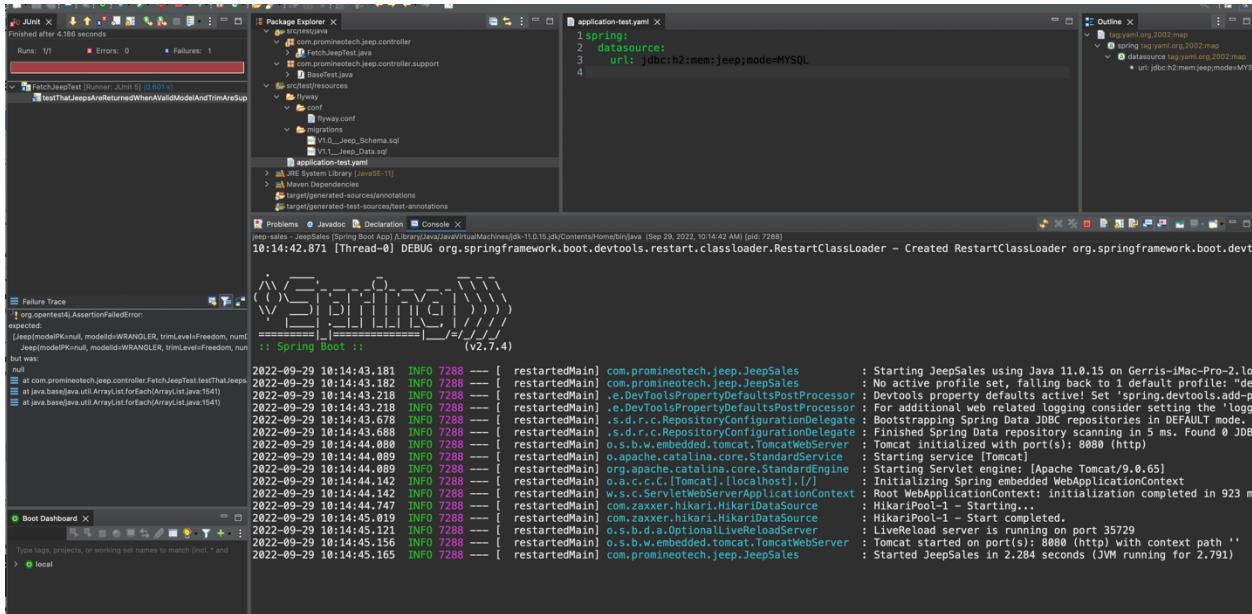
The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "application-test.yaml X". The code itself is a single-line configuration entry:

```
1 spring:  
2   datasource:  
3     url: jdbc:h2:mem:jeep;mode=MYSQL  
4 |
```

Screenshots of Code:

See above

Screenshots of Running Application:



model
string
(query)

The model name (i.e., 'WRANGLER')

trim
string
(query)

The trim level (i.e., 'Sport')

Execute **Clear**

Responses

Curl

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

Request URL

<http://localhost:8080/jeeps?model=COMPASS&trim=Sport>

Server response

Code	Details	Links
200	Response headers	
	<pre>connection: keep-alive content-length: 0 date: Thu, 29 Sep 2022 14:59:19 GMT keep-alive: timeout=60</pre>	
	Responses	
200	Description	No links
	A list of Jeeps is returned	
	Media type	
	<input type="text" value="application/json"/>	
	Controls Accept header.	
	Example Value Schema	
	<pre>{ "modelPK": 0, "modelId": "GRAND_CHEROKEE", "trimLevel": "string", "numDoors": 0, "wheelSize": 0, "basePrice": 0 }</pre>	

URL to GitHub Repository:

<https://github.com/geraldinedepaul17/BackEndClass/tree/main/Week%2014>