

Web API Design with Spring Boot Week 14 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/ThomasClifton/jeep-sales>


URL to Public Link of your Video: <https://youtu.be/1XIUFSLuVG8>

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

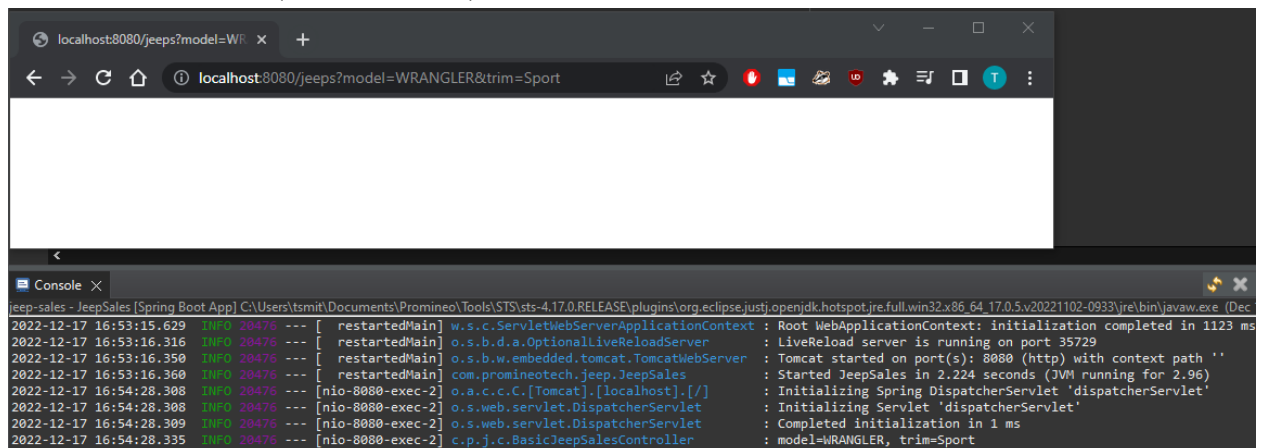
Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon:  You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a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

Web API Design with Spring Boot Week 14 Coding Assignment

headers.

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	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Sat, 17 Dec 2022 22:53:27 GMT keep-alive: timeout=60</pre>

- 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 a test run summary on the left and the test code on the right. The test run summary indicates that the test 'testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied()' passed successfully. The test code is a JUnit 5 integration test using Spring Boot Test and AssertJ.

```
1 package com.promineotech.jeeep.controller;
2
30 import static org.assertj.core.api.Assertions.assertThat;
31
32 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
33 @ActiveProfiles("test")
34 @Sql(scripts = {
35     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
36     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
37     config = @SqlConfig(encoding = "utf-8"))
38
39 class FetchJeepTest {
40
41     @Autowired
42     private TestRestTemplate restTemplate;
43
44     @LocalServerPort
45     private int serverPort;
46
47     @Test
48     void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
49         // GIVEN: a valid model, trim, url
50         JeepModel model = JeepModel.WRANGLER;
51         String trim = "Sport";
52         String url = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
53
54         // WHEN: a connection is made to the url
55         ResponseEntity<List<Jeep>> response = restTemplate.exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
56
57         // THEN: a success (OK - 200) code is returned
58         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
59     }
60 }
```

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...
- a) The test with the assertion.
 - b) The `JUnit` status bar (should be red).

Web API Design with Spring Boot Week 14 Coding Assignment

c) The method returning the expected list of Jeeps.

```
400 @Test
401 void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
402     // GIVEN: a valid model, trim, and
403     JeepModel model = JeepModel.WRANGLER;
404     String trim = "Sport";
405     String url = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
406
407     // WHEN: a connection is made to the API
408     ResponseEntity<List<Jeep>> response = restTemplate.exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<>());
409
410     // THEN: a success (OK - 200) code is returned
411     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
412
413     //And: the actual list is the same as the expected list
414     List<Jeep> expected = buildExpected();
415     assertThat(response.getBody()).isEqualTo(expected);
416 }
417
418 private List<Jeep> buildExpected() {
419     List<Jeep> list = new LinkedList<>();
420
421     // @formatter:off
422     list.add(Jeep.builder()
423         .modelId(JeepModel.WRANGLER)
424         .trimLevel("Sport")
425         .numDoors(2)
426         .wheelSize(17)
427         .basePrice(new BigDecimal("28475.00"))
428         .build());
429     list.add(Jeep.builder()
430         .modelId(JeepModel.WRANGLER)
431         .trimLevel("Sport")
432         .numDoors(4)
433         .wheelSize(17)
434         .basePrice(new BigDecimal("31975.00"))
435         .build());
436     // @formatter:on
437     return list;
438 }
```

Failure Trace

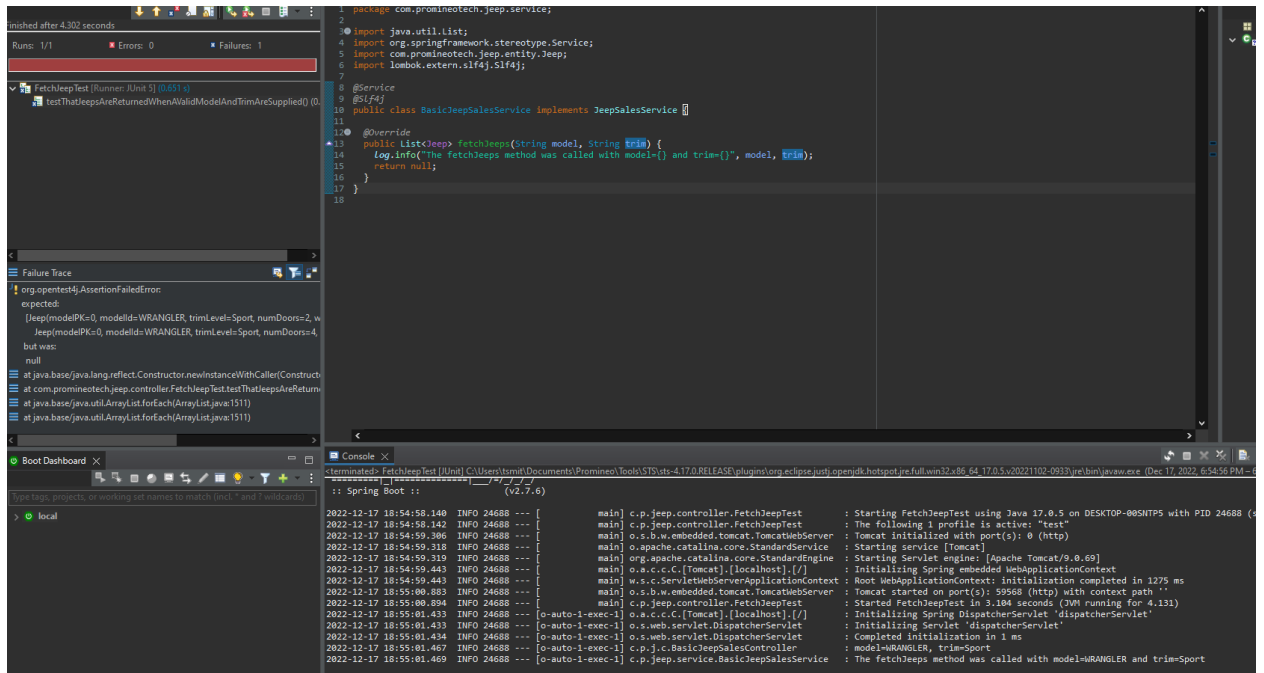
```
org.opentest4j.AssertionFailedError:
expected:
[Jeep(modelPK=0, modelId=WRANGLER, trimLevel=Sport, numDoors=2, w
Jeep(modelPK=0, modelId=WRANGLER, trimLevel=Sport, numDoors=4,
but was:
null
at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:490)
at com.promineotech.jeeptest.controller.FetchJeepTest.testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied(FetchJeepTest.java:151)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

3) 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.

Web API Design with Spring Boot Week 14 Coding Assignment

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



- 4) 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.
- 5) 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:

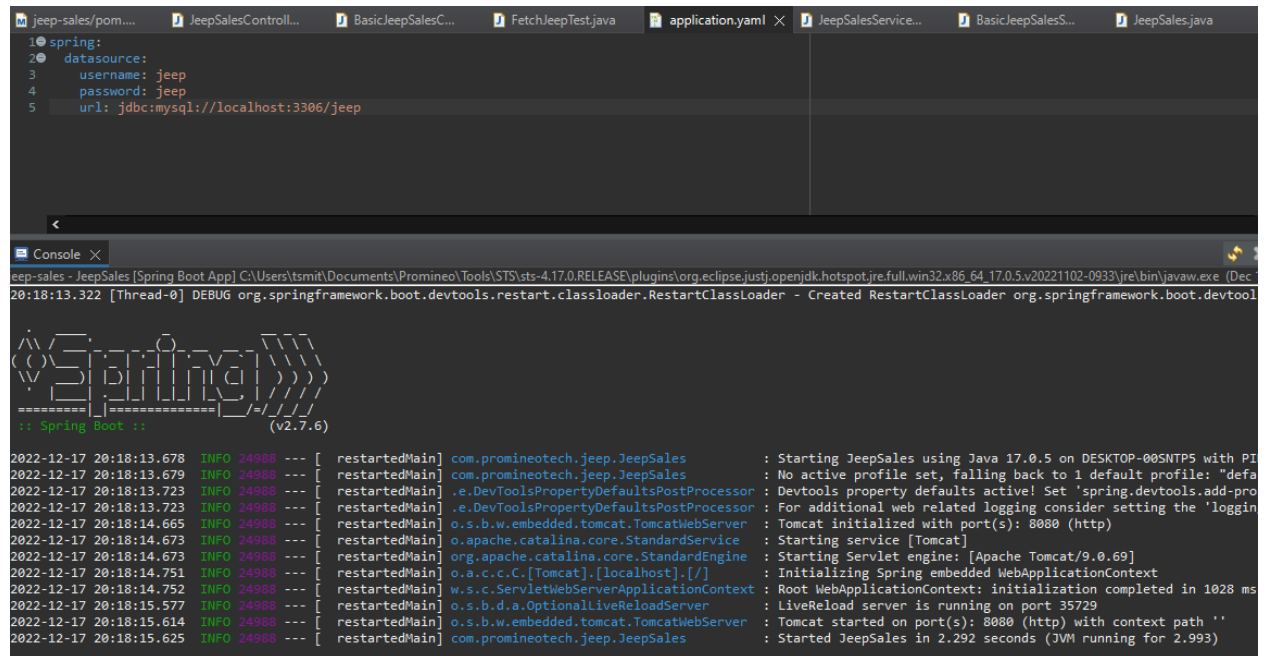
username: username

password: password

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

Web API Design with Spring Boot Week 14 Coding Assignment

- 6) 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 several tabs. The 'application.yaml' tab is active, displaying the following configuration:

```
1 spring:
2   datasource:
3     username: jeep
4     password: jeep
5     url: jdbc:mysql://localhost:3306/jeep
```

The console window below shows the application startup logs. The logs indicate that the application started successfully with no errors. The logs include the following information:

- Restarted Main: com.promineotech.jeepp.JeeppSales
- Restarted Main: com.promineotech.jeepp.JeeppSales
- Restarted Main: .e.DevToolsPropertyDefaultsPostProcessor
- Restarted Main: .e.DevToolsPropertyDefaultsPostProcessor
- Restarted Main: o.s.b.w.embedded.tomcat.TomcatWebServer
- Restarted Main: o.apache.catalina.core.StandardService
- Restarted Main: org.apache.catalina.core.StandardEngine
- Restarted Main: o.s.c.c.c.[Tomcat].[localhost].[/]
- Restarted Main: w.s.c.ServletWebServerApplicationContext
- Restarted Main: o.s.b.d.a.OptionalLiveReloadServer
- Restarted Main: o.s.b.w.embedded.tomcat.TomcatWebServer
- Restarted Main: com.promineotech.jeepp.JeeppSales

The logs also show the following messages:

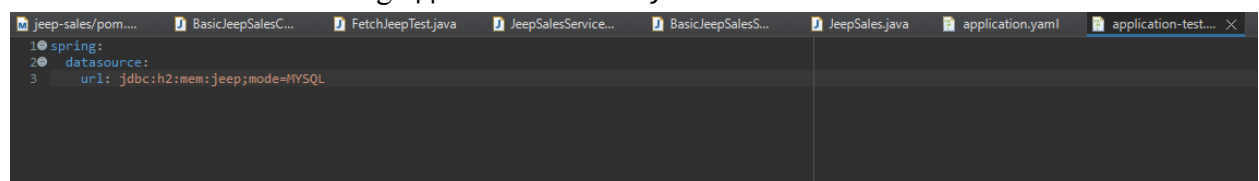
- Starting JeeppSales using Java 17.0.5 on DESKTOP-00SNTP5 with PI
- No active profile set, falling back to 1 default profile: "defa
- Devtools property defaults active! Set 'spring.devtools.add-pro
- For additional web related logging consider setting the 'login
- Tomcat initialized with port(s): 8080 (http)
- Starting service [Tomcat]
- Starting Servlet engine: [Apache Tomcat/9.0.69]
- Initializing Spring embedded WebApplicationContext
- Root WebApplicationContext: initialization completed in 1028 ms
- LiveReload server is running on port 35729
- Tomcat started on port(s): 8080 (http) with context path ''
- Started JeeppSales in 2.292 seconds (JVM running for 2.993)

- 7) 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".
- 8) 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 an IDE with several tabs. The 'application-test.yaml' tab is active, displaying the following configuration:

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