


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). 

- 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. 
- 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. 
- 7) 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. 

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".

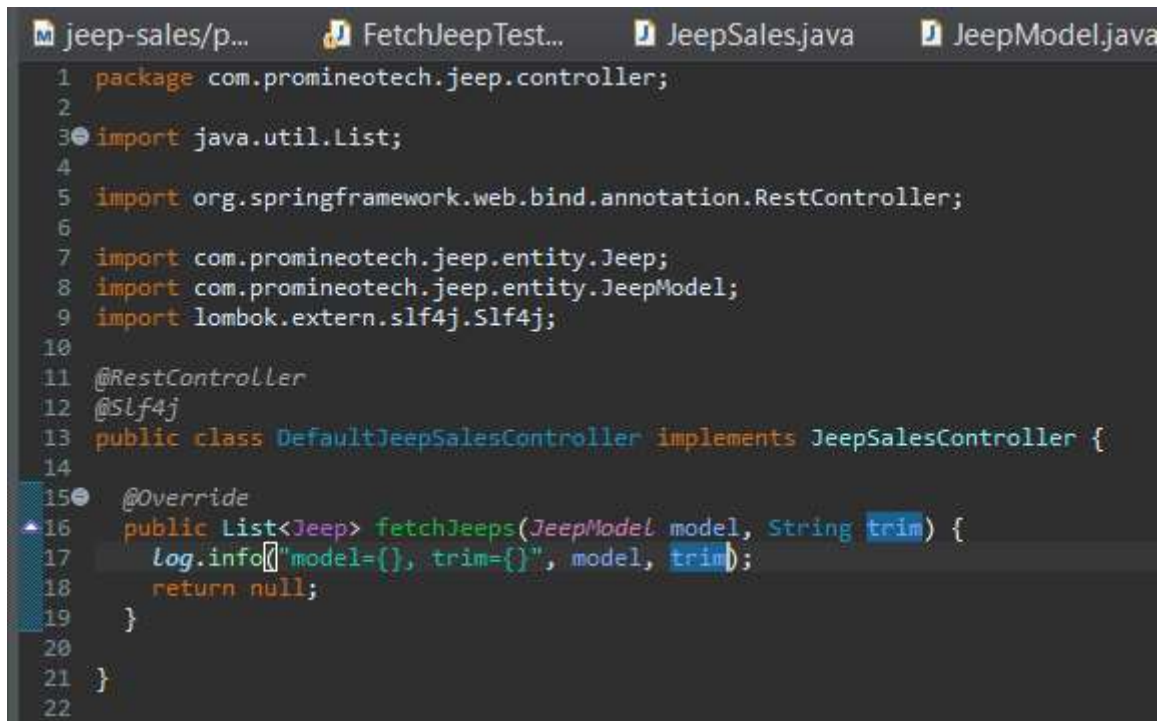
- 12) Create application-test.yaml in src/test/resources. Add the setting spring.datasource.url that points to the H2 database. It should look like this:

```
spring:
  datasource:
    url: jdbc:h2:mem:jeep
```

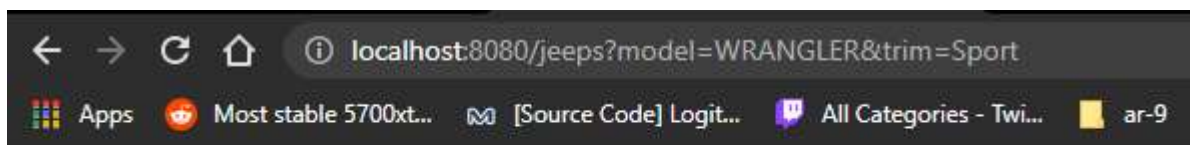
You do not need to set the username and password because the in-memory H2 database does not require them.

Produce a screenshot showing application-test.yaml. 🖥️

Screenshots of Code:



```
1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5 import org.springframework.web.bind.annotation.RestController;
6
7 import com.promineotech.jeep.entity.Jeep;
8 import com.promineotech.jeep.entity.JeepModel;
9 import lombok.extern.slf4j.Slf4j;
10
11 @RestController
12 @Slf4j
13 public class DefaultJeepSalesController implements JeepSalesController {
14
15     @Override
16     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
17         log.info("model={}, trim={}", model, trim);
18         return null;
19     }
20
21 }
22
```



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

Apps Most stable 5700xt... [Source Code] Logit... All Categories - Twi... ar-9

```

1 package com.promineotech.jeepp.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
22 @ActiveProfiles("test")
23 @Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
24               "classpath:flyway/migrations/V1.1__Jeep_data.sql"}, config = @SqlConfig(encoding = "utf-8"))
25 class FetchJeepTest {
26     @Autowired
27     private TestRestTemplate restTemplate;
28
29     @LocalServerPort
30     private int serverPort;
31
32     @Test
33     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
34         // given: a valid model, trim and URI
35         JeepModel model = JeepModel.WRANGLER;
36         String trim = "Sport";
37         String uri =
38             String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
39
40         // When: a connection is made to the URI
41         ResponseEntity<List<Jeep>> response =
42             restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
43
44         // Then: a success (OK - 200) status code is returned
45         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
46     }
47 }
48

```

Screenshots of Running Application:

The screenshot shows the Eclipse IDE's console window with the following logs:

```

jeep-sales - JeepSales [Spring Boot App] C:\Users\Pierce\Documents\STS\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v202202
2022-05-25 19:15:01.898 INFO 3764 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-05-25 19:15:01.898 INFO 3764 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-05-25 19:15:01.899 INFO 3764 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-05-25 19:15:01.902 INFO 3764 --- [nio-8080-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport

```

Swagger UI

localhost:8080/jeeps?model=W... x +

localhost:8080/swagger-ui/index.html#/default-jeep-sales-controller/fetchJeeps

Apps Most stable 5700xt... [Source Code] Logit... All Categories - Twi... ar-9 Other bookmarks

Responses

Curl

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

Request URL

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

Server response

Code	Details
200	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Thu, 26 May 2022 00:19:49 GMT keep-alive: timeout=60</pre>

Responses

Code	Description	Links
200	A list of Jeeps is returned	No links
400	The request parameters are invalid	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

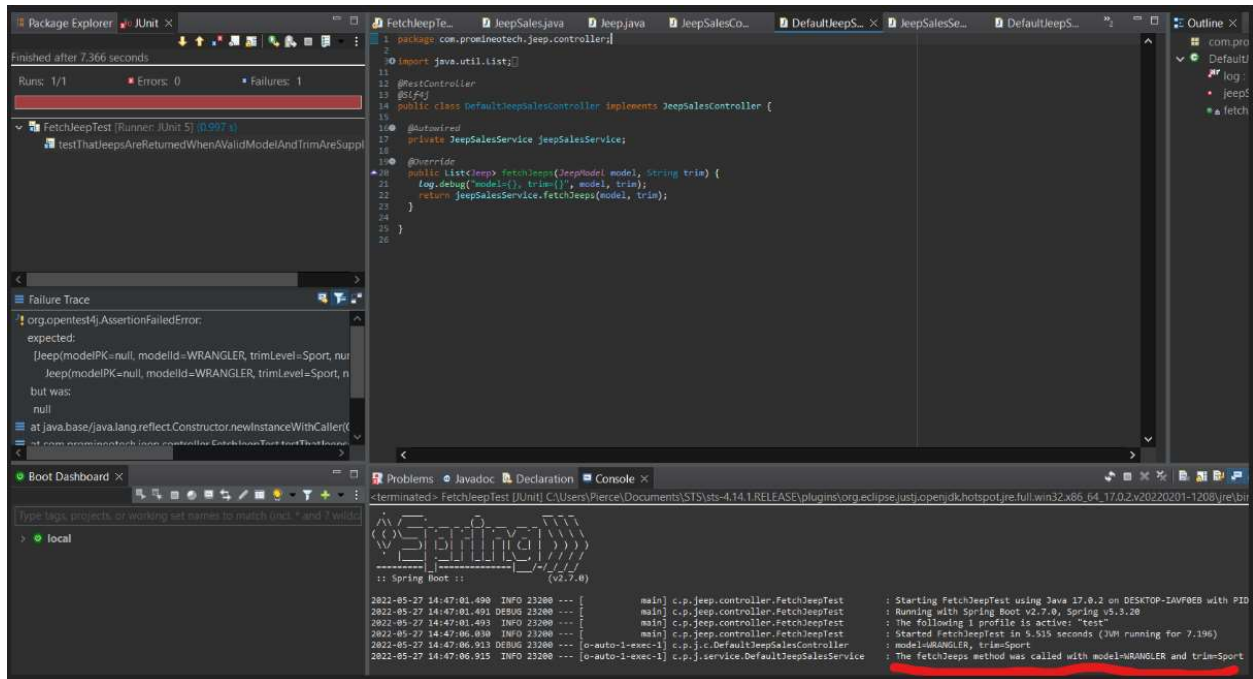
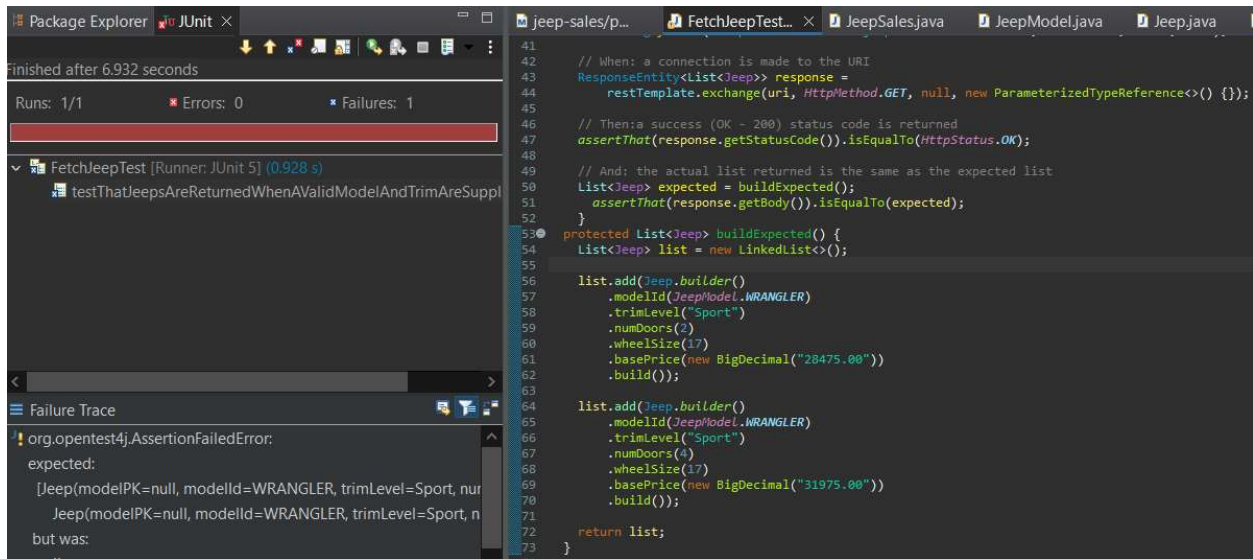
```
{
  "modelId": 0,
  "modelId": "GRAND_CHEROKEE",
  "trimLevel": "string",
  "numDoors": 0,
  "wheelSize": 0,
  "basePrice": 0
}
```

Finished after 7.164 seconds

Runs: 1/1 ✖ Errors: 0 * Failures: 0

FetchJeepTest [Runner: JUnit 5] (1.091 s)

Failure Trace



The screenshot shows the Eclipse IDE with several tabs open: 'jeep-sales/p...', 'FetchJeepTe...', 'JeepSales.java', 'DefaultJeepS...', 'DefaultJeepS...', and 'application...'. The 'application...' tab is active, displaying a Spring Boot configuration file with the following content:

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

Below the code editor, the 'Console' tab is active, showing the following output:

```
jeep-sales - JeepSales [Spring Boot App] C:\Users\Pierce\Documents\STS\sts-4.14.1.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\java.exe
15:34:21.620 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.
[Spring Boot] (v2.7.0)
:: Spring Boot ::
2022-05-27 15:34:22.280 INFO 17304 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 17.0.2 on DESKTOP-IAVF0EB with PID 17304 (
2022-05-27 15:34:22.281 DEBUG 17304 --- [ restartedMain] com.promineotech.jeep.JeepSales : Running with Spring Boot v2.7.0, Spring v5.3.20
2022-05-27 15:34:22.281 INFO 17304 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default profile: "default"
2022-05-27 15:34:25.933 INFO 17304 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 4.297 seconds (JVM running for 5.404)
```

The screenshot shows a code editor with the following Spring Boot configuration:

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

URL to GitHub Repository:

<https://github.com/PierceIsaacson/week-14-Spring-week2>