

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.

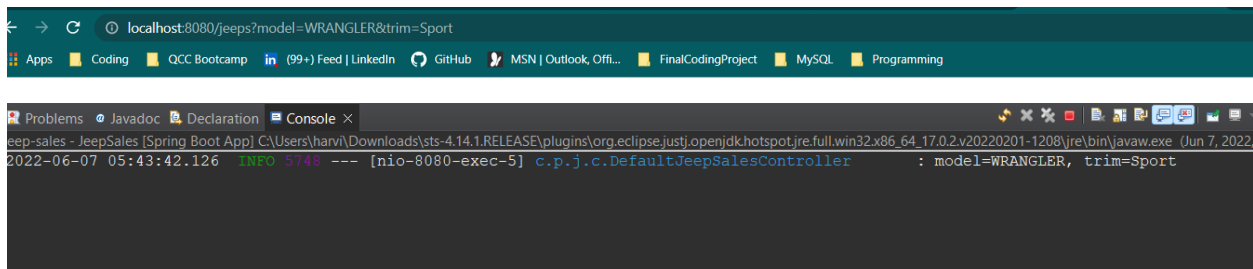
Produce a screenshot showing application-test.yaml. 

Screenshots of Code:

All the screenshots are provided below with the corresponding Step number.

Screenshots of Running Application:

Step #2:



Step #3:



Step #4:

Finished after 4.219 seconds

Runs: 1/1

✖ Errors: 0

✖ Failures: 0

✓ fetchJeepTest [Runner: JUnit 5] (0.640 s)
 ✓ testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

Step #6:

```
50  @Test
6   void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
7   //Given: valid Model, Trim
8
9       JeepModel model = JeepModel.WRANGLER;
10      String trim = "Sport";
11      String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s",
12                                serverPort, model, trim);
13      //When: an HTTP request is sent to the REST service passing in a JeepModel and trim as URI parameters
14
15      ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
16                        new ParameterizedTypeReference<>() {} );
17      //Then: assert that the response is as expected
18
19      assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
20      List<Jeep> expected = buildExpected();
21      assertThat(response.getBody()).isEqualTo(expected);
22
23
24  }
```

Package Explorer

Finished after 4.088 seconds

Runs: 1/1 ✖ Errors: 0 ✖ Failures: 1

fetchJeepTest [Runner: JUnit 5] (0.621 s)

testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() (0.621 s)

Failure Trace

org.opentest4j.AssertionFailedError:
expected:
[Jeep(modelId=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=1
Jeep(modelId=WRANGLER, trimLevel=Sport, numDoors=4, wheelSize=
but was:
null
at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:491)
at com.promineotech.jeep.controller.fetchJeepTest.testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied(fetchJeepTest.java:10)

```

65
66
67● protected List<Jeep> buildExpected() {
68     List<Jeep> newList = new LinkedList<>();
69
70     //@formatter:off
71     newList.add(Jeep.builder()
72         .modelId(JeepModel.WRANGLER)
73         .trimLevel("Sport")
74         .numDoors(2)
75         .wheelSize(17)
76         .basePrice(new BigDecimal("28475.00"))
77         .build());
78     newList.add(Jeep.builder()
79         .modelId(JeepModel.WRANGLER)
80         .trimLevel("Sport")
81         .numDoors(4)
82         .wheelSize(17)
83         .basePrice(new BigDecimal("31975.00"))
84         .build());
85
86     //@formatter:on
87     return newList;
88 }
89
90 }
91

```

Step #7.g:

```

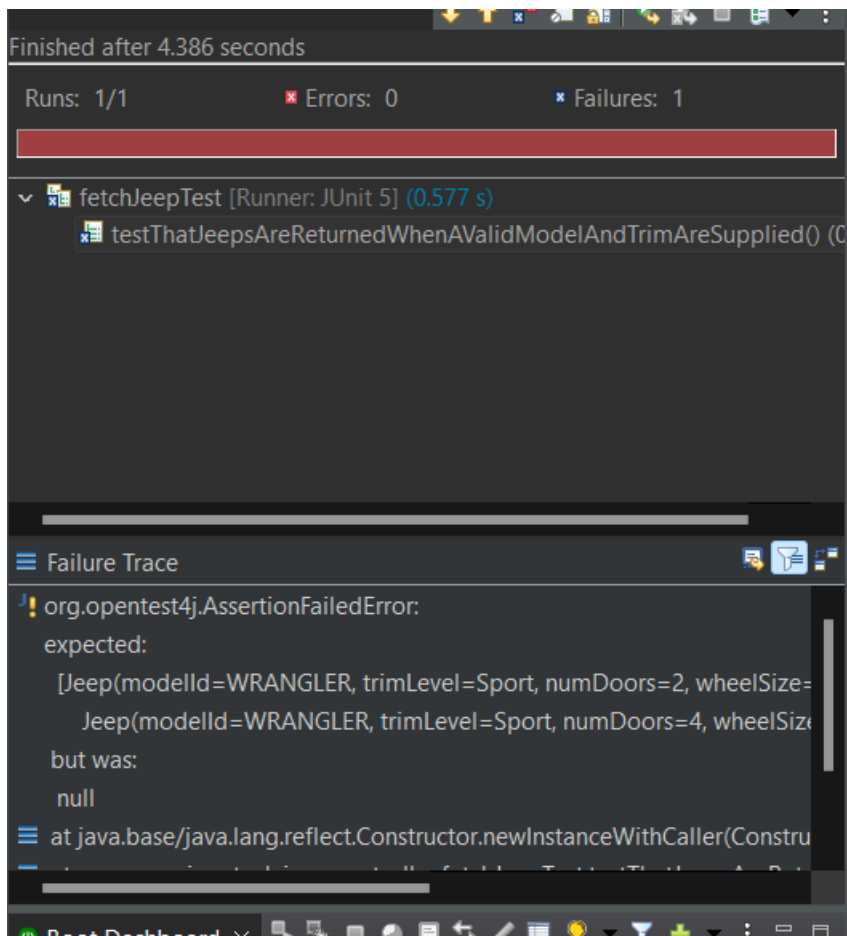
JeepSales.java  fetchJeepTe...  Jeep.java  DefaultJeepS...  JeepSalesSe...  DefaultJeepS... × 1/2
1● /**
2
3
4 package com.promineotech.jeep.service;
5
6● import java.util.List;
7 import org.springframework.stereotype.Service;
8 import com.promineotech.jeep.entity.Jeep;
9 import com.promineotech.jeep.entity.JeepModel;
10 import lombok.extern.slf4j.Slf4j;
11
12● /**
13  * @author harvi
14  *
15  */
16 @Service
17 @Slf4j
18 public class DefaultJeepSalesService implements JeepSalesService {
19
20
21● public List<Jeep> fetchJeeps(JeepModel model, String trim){
22     log.info("The service was called for model={} and the trim={}", model, trim);
23     return null;
24 }
25
26
27 }
28

```

```
terminated> fetchJeepTest [JUnit] C:\Users\harvi\Downloads\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 (Jun 7, 2022, 7:50:45 AM - 7:51:02 AM) [pid: 17204]
07:50:58.556 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.web
07:50:58.566 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTestContext@7966b
07:50:58.583 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Performing dependency injection for test context [Defa

=====
:: Spring Boot :: (v2.7.0)

2022-06-07 07:50:59.101 INFO 17204 --- [main] c.p.jeep.controller.fetchJeepTest : Starting fetchJeepTest using Java 17.0.2 on LAPTOP-D03PNF5C with
2022-06-07 07:50:59.102 INFO 17204 --- [main] c.p.jeep.controller.fetchJeepTest : The following 1 profile is active: "test"
2022-06-07 07:51:00.236 INFO 17204 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2022-06-07 07:51:00.249 INFO 17204 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-06-07 07:51:00.250 INFO 17204 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.63]
2022-06-07 07:51:00.363 INFO 17204 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-06-07 07:51:00.363 INFO 17204 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1238 ms
2022-06-07 07:51:01.711 INFO 17204 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 52569 (http) with context path ''
2022-06-07 07:51:01.721 INFO 17204 --- [main] c.p.jeep.controller.fetchJeepTest : Started fetchJeepTest in 3.106 seconds (JVM running for 4.307)
2022-06-07 07:51:02.240 INFO 17204 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-06-07 07:51:02.240 INFO 17204 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-06-07 07:51:02.241 INFO 17204 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-06-07 07:51:02.269 INFO 17204 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
2022-06-07 07:51:02.270 INFO 17204 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : The service was called for model=WRANGLER and the trim=Sport
```



Step #10:

The screenshot shows an IDE with several tabs open: `JeepSales.java`, `fetchJeepTe...`, `JeepModel.java`, `Jeep.java`, `DefaultJeepS...`, `JeepSalesSe...`, `DefaultJeepS...`, and `application...`. The `application...` tab is active, showing a Spring configuration class with the following code:

```
1 Spring:
2   datasource:
3     password: root
4     username: a3873
5     url: jdbc:mysql://localhost:3306/Jeep
```

Below the code, the console output is visible, showing the application's startup logs. The logs indicate that the application is running on Java 17.0.2 on a laptop with PID 3532. The logs also show that the application is using the default profile and that the DevTools property defaults are active. The application is initialized with port(s) 8080 (http) and started on port(s) 8080 (http) with context path ''.

Step #12:

The screenshot shows an IDE with several tabs open: `JeepSales.java`, `fetchJeepTe...`, `Jeep.java`, `DefaultJeepS...`, `JeepSalesSe...`, `DefaultJeepS...`, and `application...`. The `application...` tab is active, showing a Spring configuration class with the following code:

```
1 Spring:
2   datasource:
3     url: jdbc:hs:mem:jeep
```

URL to GitHub Repository:

<https://github.com/harvisd/Week14SpringAssignment>