

# 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:

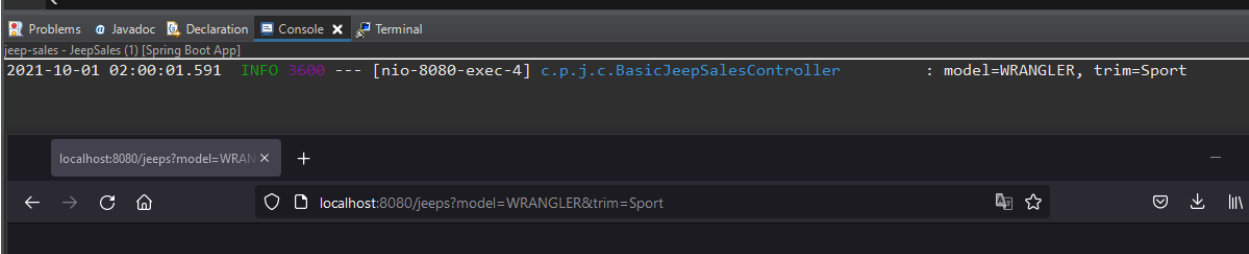
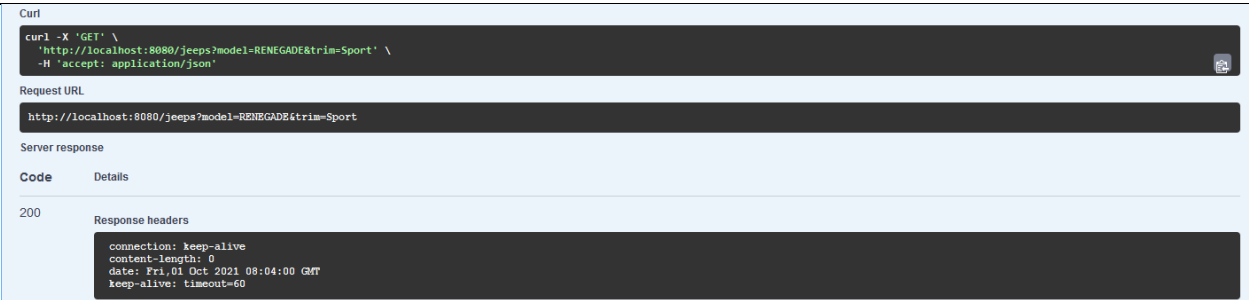
```
spring:
  datasource:
    url: jdbc:h2:mem:jeep
```

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:

## Screenshots of Running Application:

Q2	 <p>The screenshot shows an IDE with a console window displaying a log message: "2021-10-01 02:00:01.591 INFO 3600 --- [nio-8080-exec-4] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport". Below the console, a browser window is open at "localhost:8080/jeeps?model=WRANGLER&amp;trim=Sport".</p>
Q3	 <p>The screenshot shows a terminal window with a curl command: "curl -X 'GET' \ 'http://localhost:8080/jeeps?model=RENEGADE&amp;trim=Sport' \ -H 'accept: application/json'". Below the command, the request URL is shown: "http://localhost:8080/jeeps?model=RENEGADE&amp;trim=Sport". The server response is shown with a status code of 200 and response headers: "connection: keep-alive", "content-length: 0", "date: Fri, 01 Oct 2021 08:04:00 GMT", and "keep-alive: timeout=60".</p>

Q4

The screenshot displays the Eclipse IDE interface. The main editor shows the file `FetchJeepTest.java` with the following code:

```
1 package com.promineotech.jeepp.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
15
16 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
17 @ActiveProfiles("test")
18 @Sql(scripts = {
19     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
20     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
21     config = @SqlConfig(encoding = "utf-8"))
22
23 class FetchJeepTest extends FetchJeepTestSupport {
24
25     @Test
26     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
27         // fail("Not yet implemented");
28         // Martin Fowler, Given/When/Then
29         // Given: A valid model, trim, and URI
30         JeepModel model = JeepModel.WRANGLER;
31         String trim = "Sport";
32         String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);
33
34         System.out.println(uri);
35
36         // When: a connection is made to the URI
37         ResponseEntity<Jeep> response =
38             getRestTemplate().getForEntity(uri, Jeep.class);
39         // Then: a success (OK - 200) status code is returned
40         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
41     }
42
43 }
44
```

The left sidebar shows the 'Runs' tab with a green bar indicating successful execution. Below it, the 'Failure Trace' tab is visible. The bottom of the IDE features a 'Console' and 'Terminal' view. The terminal output shows the Spring Boot logo and the version number:

```
<terminated> FetchJeepTest [JUnit] C:\Users\Alicia\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justi.openjdk.h
:: Spring Boot :: (v2.5.5)
```

Q6

Package Explorer: JUnit X

Finished after 3.604 seconds

Runs: 1/1 Errors: 0 Failures: 1

FetchJeepTest (Runner: JUnit 5) (0.564 s)

testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

Failure Trace

org.opentest4j.AssertionFailedError: expected: [Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoo=2, wheelSize=17, baseBut was: null]

at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)

at com.promineotech.jeeptest.FetchJeepTest.testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Boot Dashboard

Type tags, projects, or working set names to match (incl. \* and ? wildcards)

local

```

1 package com.promineotech.jeeptest.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 class FetchJeepTest extends FetchJeepTestSupport {
12
13     @Test
14     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
15         // fail("Not yet implemented");
16         // Martin Fowler, Given/When/Then
17         // Given: A valid model, trim, and URI
18         JeepModel model = JeepModel.WRANGLER;
19         String trim = "Sport";
20         String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);
21
22         // When: a connection is made to the URI
23         ResponseEntity<List<Jeep>> response =
24             getRestTemplate().exchange(uri, HttpMethod.GET, null,
25                 new ParameterizedTypeReference<>() {});
26         // Then: a success (OK - 200) status code is returned
27         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
28
29         // And: the actual list returned is the same as the expected list
30         List<Jeep> expected = buildExpected();
31         assertThat(response.getBody()).isEqualTo(expected);
32     }
33 }

```

Q7

Package Explorer: JUnit X

Finished after 3.717 seconds

Runs: 1/1 Errors: 0 Failures: 1

FetchJeepTest (Runner: JUnit 5) (0.579 s)

testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

Failure Trace

org.opentest4j.AssertionFailedError: expected: [Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoo=2, wheelSize=17, baseBut was: null]

at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)

at com.promineotech.jeeptest.FetchJeepTest.testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Boot Dashboard

Type tags, projects, or working set names to match (incl. \* and ? wildcards)

local

```

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

```

Problems | Javadoc | Declaration | Console | Terminal

terminated: FetchJeepTest [Junit] C:\Users\Alice\AppData\Local\Microsoft\Windows\Start Menu\Programs\STU-4.11.1 RELEASE\plugin\org.eclipse.jdt.launcher\org.eclipse.jdt.launcher.exe [Oct 1, 2021, 6:18:40:12.188 [main] DEBUG org.springframework.core.env.StandardEnvironment - Activating profiles [test]

18:40:12.193 [main] DEBUG org.springframework.test.context.support.TestPropertySourceUtils - Adding inlined properties to environment: {spring.jmx.en

Spring Boot (v2.5.5)

2021-10-01 18:40:12.546 INFO 19580 --- [main] c.p.jeeptest.controller.FetchJeepTest : Starting FetchJeepTest using Java 16.0.2 on lili

2021-10-01 18:40:12.546 DEBUG 19580 --- [main] c.p.jeeptest.controller.FetchJeepTest : Running with Spring Boot v2.5.5, Spring v5.3.10

2021-10-01 18:40:12.547 INFO 19580 --- [main] c.p.jeeptest.controller.FetchJeepTest : The following profiles are active: test

2021-10-01 18:40:14.798 INFO 19580 --- [main] c.p.jeeptest.controller.FetchJeepTest : Started FetchJeepTest in 2.593 seconds (JVM runn

2021-10-01 18:40:15.301 DEBUG 19580 --- [o-auto-1-exec-1] c.p.jeeptest.controller.FetchJeepTest : model=WRANGLER, trim=Sport

2021-10-01 18:40:15.302 INFO 19580 --- [o-auto-1-exec-1] c.p.jeeptest.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRAN

Q10

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure for 'jeep-sales', including 'src/main/java' and 'src/test/java'.
- application.yml:** The file is open, showing the following configuration:

```
spring:
  datasource:
    password: jeep
    username: jeep
    url: jdbc:mysql://localhost:3306/jeep
  logging:
    level:
      root: warn
      '[com.promineotech]': debug
```
- Console:** Displays the Spring Boot startup logs, including the Spring Boot logo and the following messages:

```
2021-10-01 19:37:07.932 INFO 21336 --- [ restartedMain] com.promineotech.jee... : Starting JeepSales using Java 16.0.2 on Lilith with PID 21336 (C:\Users\Alicia...
2021-10-01 19:37:07.935 DEBUG 21336 --- [ restartedMain] com.promineotech.jee... : Running with Spring Boot v2.5.5, Spring v5.3.10
2021-10-01 19:37:07.935 INFO 21336 --- [ restartedMain] com.promineotech.jee... : No active profile set, falling back to default profiles: default
2021-10-01 19:37:09.773 INFO 21336 --- [ restartedMain] com.promineotech.jee... : Started JeepSales in 2.176 seconds (JVM running for 2.938)
```

Q12

The screenshot shows an IDE with the following components:

- File Explorer:** Shows the project structure for 'jeep-sales'.
- application.yml:** The file is open, showing the following 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/allobrandt/Spring-Boot-Wk2>