

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
```

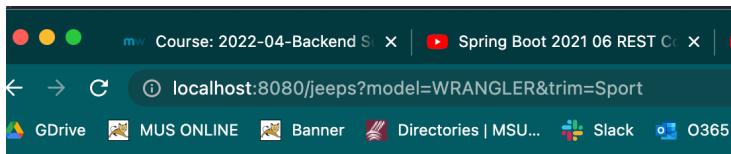
- 10) Start the application (the real application, not the test). Produce a screenshot that shows `application.yaml` and the console showing that the application has started with no errors. 
- 11) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. 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:



The screenshot shows a Swagger API documentation page for the `default-jeep-sales-controller`. The page includes:

- GET /jeeps**: Returns a list of Jeeps.
- Parameters** section:
 - model** (string, query): The model name (i.e., 'WRANGLER'). Value: GRAND_CHEROKEE
 - trim** (string, query): The trim name (i.e., 'Sport'). Value: Sport
- Execute** button.
- Responses** section:
 - 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 200):

```
Response headers
connection: keep-alive
content-length: 0
date: Tue, 09 Aug 2022 22:34:19 GMT
keep-alive: timeout=60
```

Screenshots of Running Application:

```

Console X
jeep-sales - JeepSales [Spring Boot App] [pid: 77618]
2022-08-09 16:26:08.503 INFO 77513 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-09 16:26:08.503 INFO 77513 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-09 16:26:08.539 INFO 77513 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-09 16:26:08.539 INFO 77513 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 581 ms
2022-08-09 16:26:08.913 INFO 77513 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-08-09 16:26:08.933 INFO 77513 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-09 16:26:08.940 INFO 77513 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 1.194 seconds (JVM running for 1.714)
2022-08-09 16:26:08.953 INFO 77513 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet
2022-08-09 16:26:08.954 INFO 77513 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-09 16:26:08.954 INFO 77513 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
2022-08-09 16:26:08.954 INFO 77513 --- [nio-8080-exec-4] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 113 ms
2022-08-09 16:28:24.739 INFO 77513 --- [nio-8080-exec-7] c.p.j.c.DefaultJeepSalesController : model = WRANGLER, trim = Sport

```

```

Console X
jeep-sales - JeepSales [Spring Boot App] [LibraryJava/JavaVirtualMachines/jdk-11.0.14.jdk/Contents/Home/bin/java] (Aug 9, 2022, 4:33:01 PM) [pid: 78024]
2022-08-09 16:34:19.238 INFO 78024 --- [nio-8080-exec-6] c.p.j.c.DefaultJeepSalesController : model = GRAND_CHEROKEE, trim = Sport

```

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Shows the title "STS - jeep-sales/src/test/java/com/promineotech/jeep/controller/FetchJeepTe".
- Left Sidebar:** "Package Explorer" shows a single file "FetchJeepTest.java" under "com.promineotech.jeep.controller". Below it, "JUnit" shows "Runs: 1/1", "Errors: 0", and "Failures: 0".
- Central Area:** The code editor displays the "FetchJeepTest.java" file. The code is a JUnit test for a REST API endpoint. It imports assertions from JUnit Jupiter and uses Spring Boot Test annotations like @SpringBootTest and @Test. The test checks if Jeeps are returned when valid model and trim are supplied.
- Bottom Left:** "Failure Trace" view is visible.

```

STS - jeep-sales/src/test/java/com/promineotech/jeep/controller/FetchJeepTe
FetchJeepTest.java

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

```

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Shows various icons for file operations, search, and navigation.
- Left Sidebar:** "Package Explorer" and "JUnit" tabs. The JUnit tab shows "Finished after 2.292 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 1".
- Central Area:** Four tabs are open: "Jeep.java", "DefaultJeepSalesController.java", "JeepSalesController.java", and "FetchJeepTest.java".
- FetchJeepTest.java Content:**

```
1 package com.promineotech.jeep.controller;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
6 @ActiveProfiles("test")
7 @Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
8 "classpath:flyway/migrations/V1.1__Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
9
10 class FetchJeepTest {
11
12     @Autowired
13     private TestRestTemplate restTemplate;
14
15     @LocalServerPort
16     private int serverPort;
17
18     @Test
19     void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
20         // Given a valid model, trim and URI
21         JeepModel model = JeepModel.WRANGLER;
22         String trim = "Sport";
23         String uri =
24             String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
25
26         // When a connection is made to the URI
27         ResponseEntity<List<Jeep>> response =
28             restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>());
29
30         // Then a success (200) status code is returned.
31         assertEquals(response.getStatusCode(), HttpStatus.OK);
32
33         // And : the actual list returned is the same as the expected list
34         List<Jeep> expected = buildExpected();
35         assertEquals(response.getBody(), expected);
36     }
37
38     /**
39      * @return
40      */
41     protected List<Jeep> buildExpected() {
42         List<Jeep> list = new LinkedList<>();
43
44         // @formatter:off
45         list.add(Jeep.builder()
46             .modelId(JeepModel.WRANGLER)
47             .trimLevel("Sport")
48             .numDoors(2)
49             .wheelSize(17)
50             .basePrice(new BigDecimal("28475.00"))
51             .build());
52
53         list.add(Jeep.builder()
54             .modelId(JeepModel.WRANGLER)
55             .trimLevel("Sport")
56             .numDoors(4)
57             .wheelSize(17)
58             .basePrice(new BigDecimal("31975.00"))
59             .build());
60         // @formatter:on
61
62         return list;
63     }
64
65 }
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88 }
```
- Bottom Left:** "Failure Trace" section showing an assertion failure:

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

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Shows various icons for file operations, search, and navigation.
- Left Sidebar:** "Package Explorer" and "JUnit" tabs. The JUnit tab shows "Finished after 2.197 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 1".
- Central Area:** Five tabs are open: "JeepSales.java", "DefaultJeepSalesCon...", "JeepSalesService.java", "DefaultJeepSalesSer...", and "JeepSalesController.java".
- DefaultJeepSalesController.java Content:**

```
1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultJeepSalesController implements JeepSalesController {
8
9     @Autowired
10    private JeepSalesService jeepSalesService;
11
12    @Override
13    public List<Jeep> fetchJeeps(JeepModel model, String trim) {
14        log.info("model = {}, trim = {}", model, trim);
15        return null;
16    }
17
18    //Rob returned jeepSalesService.fetchJeeps(model, trim);
19
20
21
22
23
24 }
```
- Bottom Left:** "Failure Trace" section showing an assertion failure:

```
! org.opentest4j.AssertionFailedError:  
expected:  
[Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoors=2  
Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoors=2  
but was:  
null  
at com.promineotech.jeep.controller.FetchJeepTest.testThatJeepsAreR  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)
```

The screenshot shows an IDE interface with several tabs open, including `JeepSales.java`, `DefaultJeepSalesCo...`, `JeepSalesService.java`, `DefaultJeepSalesSer...`, `JeepSalesController.j...`, `jeep-sales/pom.xml`, `application.yaml`, and `Outline`. Below the tabs, a terminal window displays the following log output:

```

2022-08-09 22:05:49.558 INFO 87998 --- [ restartedMain] com.prominotech.jeep.JeepSales           : Starting JeepSales using Java 11.0.14 on CH1049-1624.local with PID 87998 (/Users/f55m018/Documents/Promi
2022-08-09 22:05:49.558 INFO 87998 --- [ restartedMain] com.prominotech.jeep.JeepSales           : No active profile is defined. Falling back to default profile "default"
2022-08-09 22:05:49.583 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-08-09 22:05:49.583 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 4 ms. Found 0 JDBC repository interfaces.
2022-08-09 22:05:49.583 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-08-09 22:05:49.916 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-08-09 22:05:49.916 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 4 ms. Found 0 JPA repository interfaces.
2022-08-09 22:05:49.923 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repositories in DEFAULT mode.
2022-08-09 22:05:49.923 INFO 87998 --- [ restartedMain] o.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 4 ms. Found 0 MongoDB repository interfaces.
2022-08-09 22:05:50.236 INFO 87998 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-08-09 22:05:50.236 INFO 87998 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : StandardEngine [Tomcat/9.0.65]
2022-08-09 22:05:50.236 INFO 87998 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-09 22:05:50.236 INFO 87998 --- [ restartedMain] o.apache.catalina.core.StandardService : Initializing Spring embedded WebApplicationContext
2022-08-09 22:05:50.236 INFO 87998 --- [ restartedMain] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 689 ms
2022-08-09 22:05:50.273 INFO 87998 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : HikariPool-1 - Starting...
2022-08-09 22:05:50.685 INFO 87998 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-08-09 22:05:51.035 INFO 87998 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : LiveReload server is running on port 35729
2022-08-09 22:05:51.038 INFO 87998 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-09 22:05:51.049 INFO 87998 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Started JeepSales in 1.724 seconds (JVM running for 2.238)
2022-08-09 22:05:51.057 INFO 87998 --- [ restartedMain] com.prominotech.jeep.JeepSales           : Started JeepSales in 1.724 seconds (JVM running for 2.238)

```

```

1 spring:
2   datasource:
3     url: jdbc:h2:mem:jeep

```

URL to GitHub Repository:

<https://github.com/lindsey406/week14homework.git>