

Web API Design with Spring Boot Week 1 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.

Here's a hint: make sure you are running a version of Java that is 11+. To get the version, open a Windows command window or a Mac Terminal window and type `java -version`. If you need to upgrade, go here: <https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/downloads-list.html>. Pick the .msi installer version (Windows) or the .pkg version (Mac).

Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

- 1) Create a Maven project named `JeepSales` as described in the video.

- a) In Spring Tool Suite, click the "File" menu. Select "New/Project...". In the popup, expand "Maven" and select "Maven Project". Click "Next".
- b) Check "Create a simple project (skip archetype selection)". Click "Next".
- c) Enter the following:

Group Id	com.promineotech
Artifact Id	jeep-sales

Click "Finish".

- 2) Navigate to the Spring Initializr (<https://start.spring.io/>).

- a) Confirm the following settings:

Project Maven Project	
Language	Java
Spring Boot	Select the latest stable version (not SNAPSHOT or RC)
Group	com.promineotech
Artifact	jeep-sales
Name	jeep-sales
Description	Jeep Sales
Package name	com.promineotech
Packaging	Jar
Java	11

- b) Add the dependencies from the Initializr:
 - i) Web
 - ii) Devtools
 - iii) Lombok
 - c) Click "Explore" at the bottom of the page.
 - d) Click "Copy" to copy the pom.xml generated by the Initializr to the clipboard.

- 3) In Spring Tool Suite, open pom.xml (in the project root directory). Select all the text in the editor and replace it with the XML copied to the clipboard in the prior step.
- 4) Navigate to <https://mvnrepository.com/>. Search for springdoc-openapi-ui. Select the latest version and add the entry to the POM file in the <dependencies> section.
- 5) Create a package in src/main/java named com.promineotech.jeep. In this package:
 - a) Create a Java class with a main method named JeepSales.
 - b) Add a class-level annotation: @SpringBootApplication and the import statement.
 - c) In the main() method, add a call to SpringApplication.run();. Use JeepSales.class as the first parameter, and the args parameter that was passed into the main() method as the second. The entire class should look like this:

```

package com.promineotech.jeep;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JeepSales {

    public static void main(String[] args) {
        SpringApplication.run(JeepSales.class, args);
    }
}

```
- 6) Refer to README.docx in the supplied project resources. Copy all files in the Files folder in the resources to your project as described in the README. **Do not copy the files in the Entity or Source folders at this time.**
 - a) Load the files that were added: right-click on the project in Package Explorer and select "Refresh".
 - b) Update the project with the new POM dependencies: right-click on the project in Package Explorer, select "Maven/Update Project". When the "Update Maven Project" panel appears, click "OK".
- 7) Using the MySQL Workbench or MySQL command line client (CLI), create a database named "jeep".
- 8) Using dBeaver, or the MySQL client of choice, load the supplied .sql files (v1.0_Jeep_Schema.sql, and v1.1_Jeep_Data.sql) into the MySQL database to create the tables and populate them with data. These files are found in the project folder src/test/resources/flyway/migrations.
- 9) Create a new package in src/test/java named com.promineotech.jeep.controller. Create a Spring Boot integration test named FetchJeepTest using the techniques shown in the video.

- a) Add the `@SpringBootTest`, `@ActiveProfiles`, and `@Sql` annotations as described in the video.
- b) The class must not be `public`. It should have package-level access (i.e., not `public`, `private`, or `protected`).
- c) The video extended `FetchJeepTestSupport`, but you don't need to do that for the homework. Just put everything in `FetchJeepTest`. It should look like this:

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@Sql/scripts = {
    "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
    "classpath:flyway/migrations/V1.1_Jeep_Data.sql"},
    config = @SqlConfig(encoding = "utf-8"))
class FetchJeepTest {
```

- d) Create a test method in `FetchJeepTest`. The method must have the following method signature:

```
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
```

- e) Inject a `TestRestTemplate` in the test class. Name the variable `restTemplate`. Inject the port used in the test using the `@LocalServerPort` annotation. Name the variable `serverPort`. The variables and annotations should look like this:

```
@Autowired
private TestRestTemplate restTemplate;

@LocalServerPort
private int serverPort;
```

- 10) Create a new package in `src/main/java` named `com.promineotech.jeep.entity`. In that package, create an enum named `JeepModel`. Add all the jeep models from the `model_id` column in the `models` table in the database. You can use this query in dBeaver: `SELECT DISTINCT model_id FROM models`.

Video 4 @25:30 says to add a class for `JeepModel` not enum.

- 11) Create a `Jeep` class in the `com.promineotech.jeep.entity` package. Add the columns from the `models` table into this class as instance variables. Annotate the class with the Lombok annotations `@Data`, `@Builder` (and optionally both `@NoArgsConstructor` and `@AllArgsConstructor`). Note that `modelId` should be of type `JeepModel` and `basePrice` should be of type `BigDecimal`. The class should look like this (remember to add the appropriate import statements):

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
```

```

public class Jeep {
    private Long modelPK;
    private JeepModel modelId;
    private String trimLevel;
    private int numDoors;
    private int wheelSize;
    private BigDecimal basePrice;
}

```

- 12) In the supplied resources, copy all files in the Entities folder to the `src/main/java/com/- promineotech/jeep/entity` folder. **Do not copy anything from the Source folder at this time.**
- 13) Back in the test method that you were writing, create local variables for `JeepModel`, `trim`, and `uri`. Set them appropriately like this:

Variable Type	Variable Name	Variable Value
<code>JeepModel</code>	<code>model</code>	<code>JeepModel.WRANGLER</code>
<code>String</code>	<code>trim</code>	<code>"Sport"</code>
<code>String</code>	<code>uri</code>	<code>String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);</code>

- a) Send an HTTP request to the REST service that passes a `JeepModel` and `trim` level as URI parameters (as shown in the video). Use this method call:

```
ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri,
    HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
```

Make sure to use the `import java.util.List` and `org.springframework.http.HttpMethod`. – help!

- b) Using [AssertJ](#), test that the response that comes back from the server is 200 (success) – or as is shown in the video: `HttpStatus.OK`. The code should look like this:

```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
```

Use the import statements:

```
import static org.assertj.core.api.Assertions.assertThat;
```

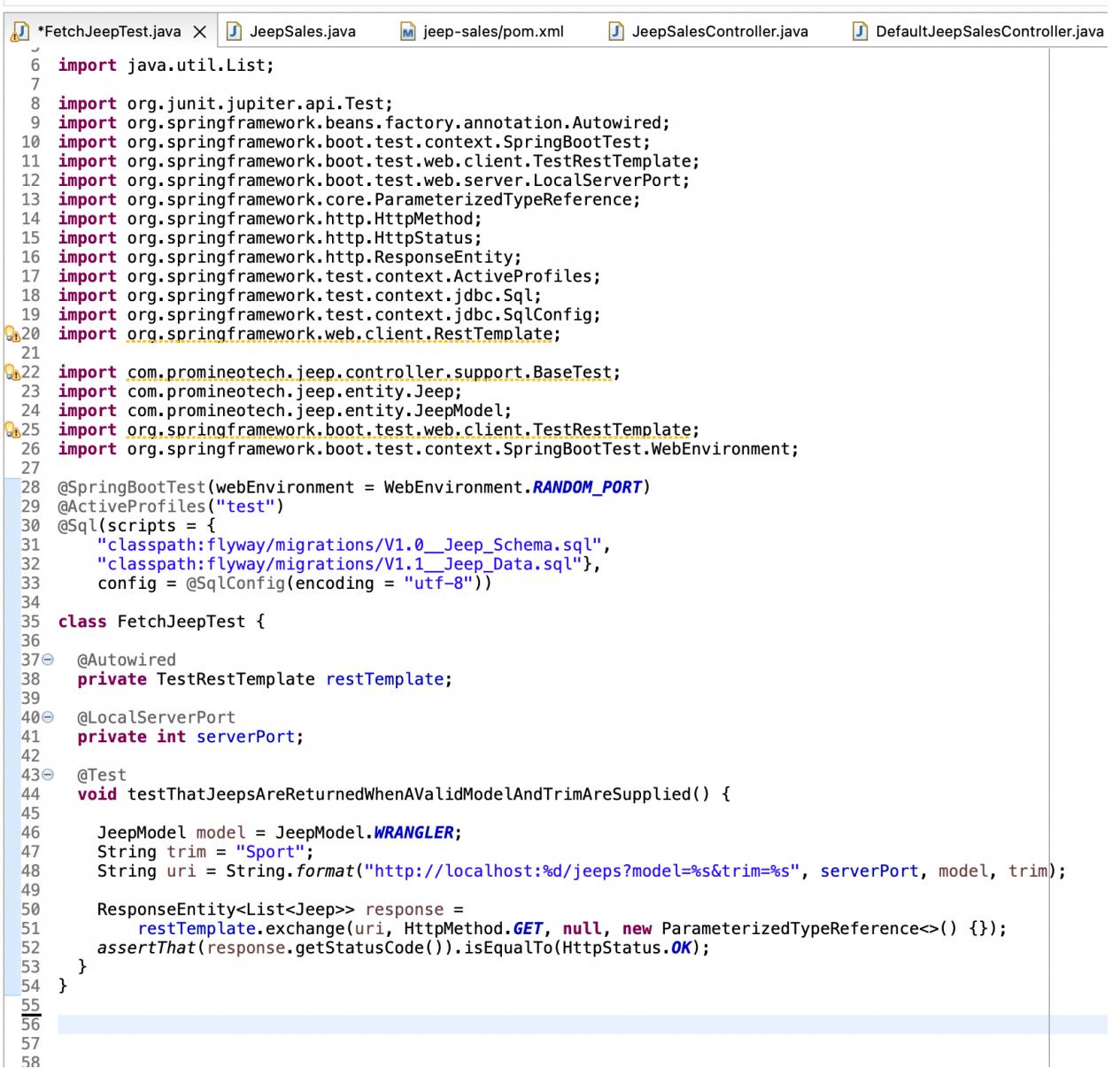
- c) Produce a screenshot showing the completed test class. 

```

    //Then: a success (OK - 200) status code is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
}

}

```



```

*FetchJeepTest.java X  JeepSales.java  jeep-sales/pom.xml  JeepSalesController.java  DefaultJeepSalesController.java
1
2
3
4
5
6 import java.util.List;
7
8 import org.junit.jupiter.api.Test;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.boot.test.context.SpringBootTest;
11 import org.springframework.boot.test.web.client.TestRestTemplate;
12 import org.springframework.boot.test.web.server.LocalServerPort;
13 import org.springframework.core.ParameterizedTypeReference;
14 import org.springframework.http.HttpMethod;
15 import org.springframework.http.HttpStatus;
16 import org.springframework.http.ResponseEntity;
17 import org.springframework.test.context.ActiveProfiles;
18 import org.springframework.test.context.jdbc.Sql;
19 import org.springframework.test.context.jdbc.SqlConfig;
20 import org.springframework.web.client.RestTemplate;
21
22 import com.promineotech.jeep.controller.support.BaseTest;
23 import com.promineotech.jeep.entity.Jeep;
24 import com.promineotech.jeep.entity.JeepModel;
25 import org.springframework.boot.test.web.client.TestRestTemplate;
26 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
27
28 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
29 @ActiveProfiles("test")
30 @Sql scripts = {
31     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
32     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8")
33
34
35 class FetchJeepTest {
36
37     @Autowired
38     private TestRestTemplate restTemplate;
39
40     @LocalServerPort
41     private int serverPort;
42
43     @Test
44     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
45
46         JeepModel model = JeepModel.WRANGLER;
47         String trim = "Sport";
48         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
49
50         ResponseEntity<List<Jeep>> response =
51             restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
52         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
53     }
54 }
55
56
57
58

```

The screenshot shows the Eclipse IDE interface. At the top, the JUnit view displays "Runs: 1/1", "Errors: 0", and "Failures: 0". Below it, the Package Explorer shows a green progress bar indicating the test finished after 0.449 seconds. The code editor on the right contains two files: FetchJeepTest.java and JeepSales.java. FetchJeepTest.java includes annotations for @SpringBootTest, @WebEnvironment, and @ActiveProfiles("test"). It also contains @Sql scripts for Flyway migrations. The Java code defines a class FetchJeepTest. The output view at the bottom shows detailed logs from the application's main method, including Spring Boot initialization and Tomcat startup.

```

@ActiveProfiles("test")
@Sql(scripts = {
    "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
    "classpath:flyway/migrations/V1.1_Jeep_Data.sql"
}, config = @SqlConfig(encoding = "utf-8"))
class FetchJeepTest {
}

```

```

2022-09-23 23:43:07.397 INFO 28648 --- [main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 11.0.15 on Gerris-iMac-Pro-2.local with PID 28648
2022-09-23 23:43:07.398 INFO 28648 --- [main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-09-23 23:43:08.095 INFO 28648 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 0 (http)
2022-09-23 23:43:08.105 INFO 28648 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-23 23:43:08.105 INFO 28648 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-23 23:43:08.181 INFO 28648 --- [main] o.a.c.ServletContextListener : [localhost].[] : Initializing Spring embedded WebApplicationContext
2022-09-23 23:43:08.181 INFO 28648 --- [main] o.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 766 ms
2022-09-23 23:43:09.071 INFO 28648 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 51183 (http) with context path ''
2022-09-23 23:43:09.080 INFO 28648 --- [main] c.p.jeep.controller.FetchJeepTest : Started FetchJeepTest in 2.082 seconds (JVM running for 2.796)

```

14) In `src/main/java`, create a new package `com.promineotech.jeep.controller`. In this package, create an interface named `JeepSalesController`.

- Add the class-level annotation `@RequestMapping("/jeeps")`.
- Add the `fetchJeeps` method in a controller interface with the following signature:

```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```

Make sure you use the `List` from `java.util.List`.

- Add OpenAPI documentation to document the four possible outcomes: 200 (success), 400 (bad input), 404 (not found) and 500 (unplanned error) as shown in the video.
- Add the parameter annotations in the OpenAPI documentation to describe the `model` and `trim` parameters.
- Add the `@GetMapping` annotation and the `@ResponseStatus(code = HttpStatus.OK)` annotation as method-level annotations to the `fetchJeeps` method.
- Add the `@RequestParam` annotations to the parameters as described in the video. The interface should look like this (omitting the OpenAPI annotations):

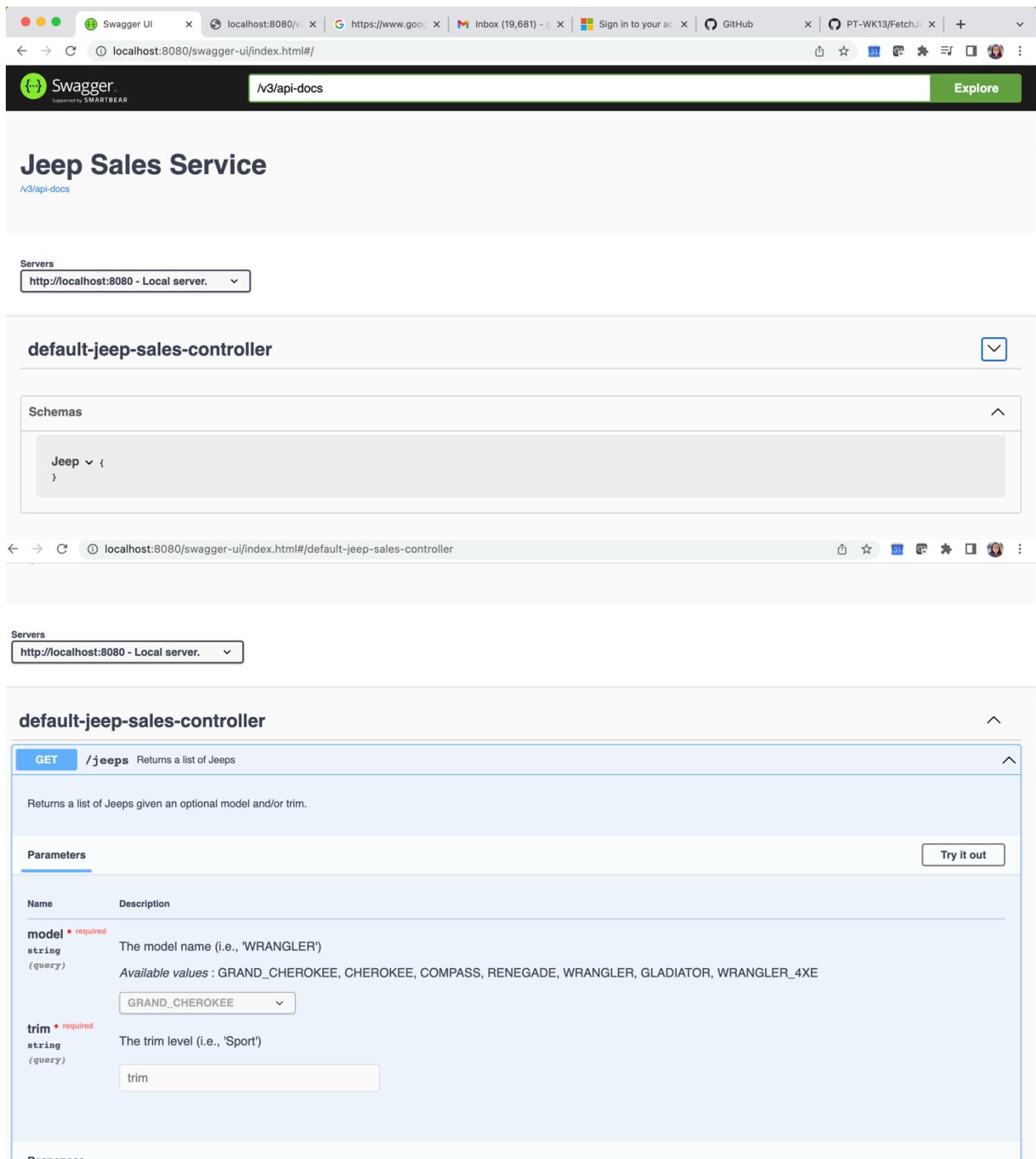
```
@RequestMapping("/jeeps")
public interface JeepSalesController {
```

```

@GetMapping
@ResponseBody(code = HttpStatus.OK)
List<Jeep> fetchJeeps(@RequestParam JeepModel model,
                      @RequestParam String trim);
}

```

- g) Produce a screenshot showing the interface and OpenAPI documentation. 



The screenshot shows the Swagger UI interface for the "Jeep Sales Service". The top navigation bar includes tabs for "Swagger UI", "localhost:8080", "https://www.google.com", "Inbox (19,681)", "Sign in to your account", "GitHub", and "PT-WK13/Fetch.js". The main title is "Jeep Sales Service" under the path "/v3/api-docs". A "Servers" dropdown is set to "http://localhost:8080 - Local server". The "default-jeep-sales-controller" section is expanded, showing a "Schemas" panel containing a "Jeep" schema definition. Below it, the "GET /jeeps" operation is detailed, showing its purpose ("Returns a list of Jeeps given an optional model and/or trim."), parameters (model and trim), and responses. The "model" parameter is a required string query parameter with available values: GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, WRANGLER, GLADIATOR, WRANGLER_4XE. The "trim" parameter is a required string query parameter with available values: GRAND_CHEROKEE, Sport.



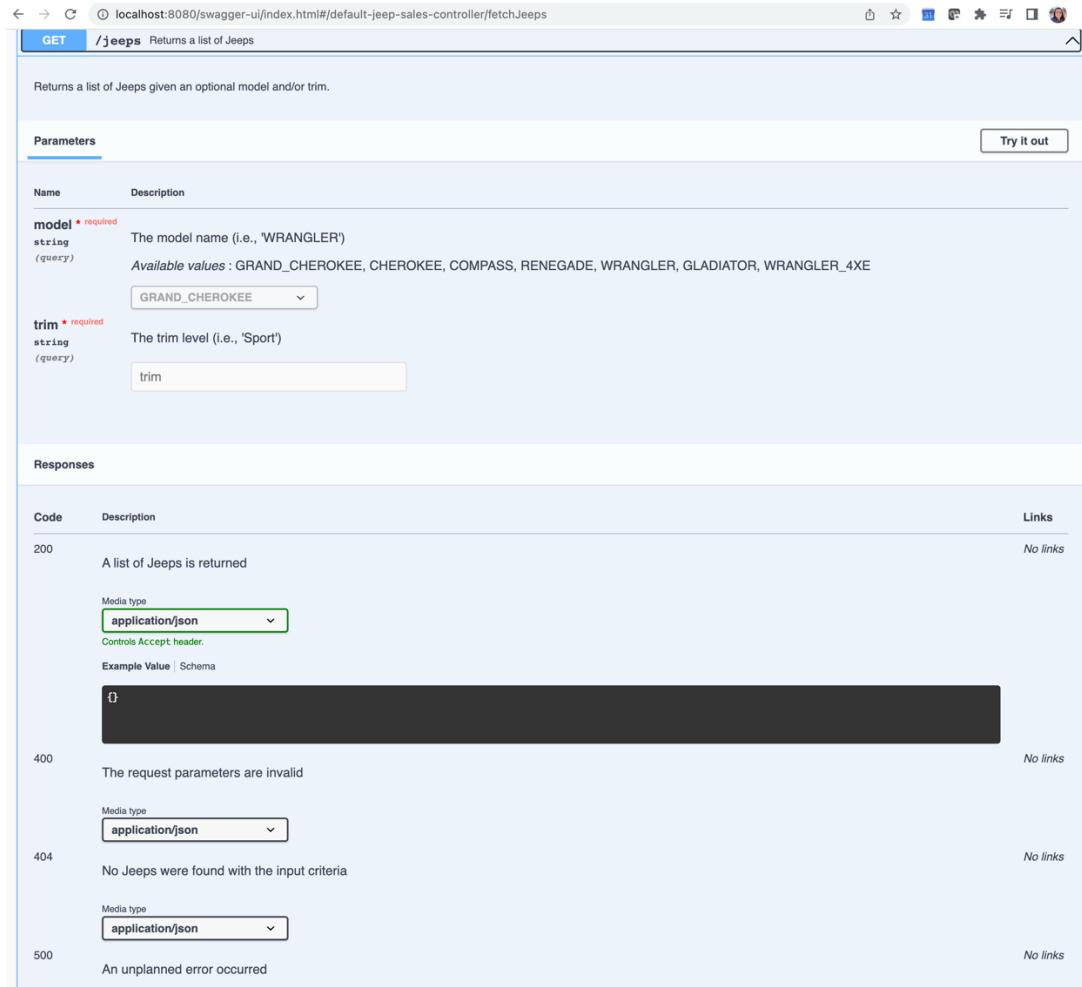
:: Spring Boot :: (v2.7.4)

```
FetchJeepTest.java  JeepSales.java  jeep-sales/pom.xml  JeepSalesController.java  DefaultJeepSalesController.java  Help
```

```
1 package com.promineotech.jeep.controller;
2
3
4 import java.util.List;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RequestParam;
9 import org.springframework.web.bind.annotation.ResponseStatus;
10 import com.promineotech.jeep.entity.Jeep;
11 import com.promineotech.jeep.entity.JeepModel;
12 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
13 import io.swagger.v3.oas.annotations.Operation;
14 import io.swagger.v3.oas.annotations.Parameter;
15 import io.swagger.v3.oas.annotations.info.Info;
16 import io.swagger.v3.oas.annotations.media.Content;
17 import io.swagger.v3.oas.annotations.media.Schema;
18 import io.swagger.v3.oas.annotations.responses.ApiResponse;
19 import io.swagger.v3.oas.annotations.servers.Server;
20
21 @RequestMapping("/jeeps")
22 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
23     @Server(url = "http://localhost:8080", description = "Local server.")})
24
25
26 public interface JeepSalesController {
27
28     @Operation(
29         summary = "Returns a list of Jeeps",
30         description = "Returns a list of Jeeps given an optional model and/or trim.",
31         responses = {
32             @ApiResponse(responseCode = "200",
33                 description = "A list of Jeeps is returned",
34                 content = @Content(mediaType = "application/json",
35                     schema = @Schema(implementation = Jeep.class))),
36             @ApiResponse(responseCode = "400",
37                 description = "The request parameters are invalid",
38                 content = @Content(mediaType = "application/json")),
39             @ApiResponse(responseCode = "404",
40                 description = "No Jeeps were found with the input criteria",
41                 content = @Content(mediaType = "application/json")),
42             @ApiResponse(responseCode = "500",
43                 description = "An unplanned error occurred",
44                 content = @Content(mediaType = "application/json"))
45         },
46         parameters = {
47             @Parameter(name = "model",
48                 allowEmptyValue = false,
49                 required = false,
50                 description = "The model name (i.e., 'WRANGLER')"),
51             @Parameter(name = "trim",
52                 allowEmptyValue = false,
53                 required = false,
54                 description = "The trim level (i.e., 'Sport')")
55         }
56     )
57
58     @GetMapping
59     @ResponseStatus(code = HttpStatus.OK)
60     List<Jeep> fetchJeeps(
61         @RequestParam JeepModel model,
62         @RequestParam String trim);
63 }
```

15) Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.

16) Run the application within the IDE and show the resulting OpenAPI (Swagger) documentation produced in the browser. Produce a screenshot of the documentation showing all four possible outcomes. 



The screenshot shows the Swagger UI interface for a REST API. At the top, the URL is `localhost:8080/swagger-ui/index.html#/default-jeep-sales-controller/fetchJeeps`. The main section is for the `GET /jeeps` endpoint, which returns a list of Jeeps. It includes a description: "Returns a list of Jeeps given an optional model and/or trim." Below this is a "Parameters" table:

Name	Description
<code>model</code> * required <code>string</code> (query)	The model name (i.e., 'WRANGLER') Available values : GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, WRANGLER, GLADIATOR, WRANGLER_4XE GRAND_CHEROKEE
<code>trim</code> * required <code>string</code> (query)	The trim level (i.e., 'Sport') trim

Below the parameters is a "Responses" table:

Code	Description	Links
200	A list of Jeeps is returned Media type application/json Controls Accept header. Example Value Schema [redacted]	No links
400	The request parameters are invalid Media type application/json	No links
404	No Jeeps were found with the input criteria Media type application/json	No links
500	An unplanned error occurred Media type application/json	No links

Screenshots of Code:



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

- Project Explorer:** Shows the project structure with 'JUN1' as the active workspace.
- Console:** Displays the output of the 'mvn clean package' command, indicating successful compilation and deployment to Tomcat 9.0.45.
- Java Editor:** The file 'JeepSales.java' is open, showing the main method and annotations for a Spring Boot application.
- Server View:** Shows the 'Tomcat v9.0 (Apache Tomcat/9.0.45)' server is running.
- Run View:** Shows 1 run, 0 errors, and 0 failures.
- FetchJeepTest:** A JUnit test runner with one test case (0.474 s).

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  ⊕  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.4</version>
    <relativePath/> <!-- lookup parent from repository --&gt;
  &lt;/parent&gt;

  &lt;groupId&gt;com.promineotech&lt;/groupId&gt;
  &lt;artifactId&gt;jeep-sales&lt;/artifactId&gt;
  &lt;version&gt;0.0.1-SNAPSHOT&lt;/version&gt;

  &lt;name&gt;jeep-sales&lt;/name&gt;
  &lt;description&gt;Jeep Sales&lt;/description&gt;

  ⊕  &lt;properties&gt;
    &lt;java.version&gt;11&lt;/java.version&gt;
  &lt;/properties&gt;

  ⊕  &lt;dependencies&gt;
    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
      &lt;artifactId&gt;spring-boot-starter-web&lt;/artifactId&gt;
    &lt;/dependency&gt;

    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
      &lt;artifactId&gt;spring-boot-devtools&lt;/artifactId&gt;
      &lt;scope&gt;runtime&lt;/scope&gt;
      &lt;optional&gt;true&lt;/optional&gt;
    &lt;/dependency&gt;

    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.projectlombok&lt;/groupId&gt;
      &lt;artifactId&gt;lombok&lt;/artifactId&gt;
      &lt;optional&gt;true&lt;/optional&gt;
    &lt;/dependency&gt;

    &lt;!-- OpenAPI dependency =====&gt;
    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.springdoc&lt;/groupId&gt;
      &lt;artifactId&gt;springdoc-openapi-ui&lt;/artifactId&gt;
      &lt;version&gt;1.6.11&lt;/version&gt;
    &lt;/dependency&gt;

    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
      &lt;artifactId&gt;spring-boot-starter-test&lt;/artifactId&gt;
      &lt;scope&gt;test&lt;/scope&gt;
    &lt;/dependency&gt;

    ⊕  &lt;dependency&gt;
      &lt;groupId&gt;org.springdoc&lt;/groupId&gt;
      &lt;artifactId&gt;springdoc-openapi-ui&lt;/artifactId&gt;
      &lt;version&gt;1.6.11&lt;/version&gt;
    &lt;/dependency&gt;

  &lt;/dependencies&gt;

  ⊕  &lt;build&gt;
    ⊕  &lt;plugins&gt;
      ⊕  &lt;plugin&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-maven-plugin&lt;/artifactId&gt;
        ⊕  &lt;configuration&gt;
          ⊕  &lt;excludes&gt;
            ⊕  &lt;exclude&gt;
              &lt;groupId&gt;org.projectlombok&lt;/groupId&gt;
              &lt;artifactId&gt;lombok&lt;/artifactId&gt;
            &lt;/exclude&gt;
          &lt;/excludes&gt;
        &lt;/configuration&gt;
      &lt;/plugin&gt;
    &lt;/plugins&gt;
  &lt;/build&gt;

&lt;/project&gt;
</pre>

```

```
*JeepSalesController.java X DefaultJeepSalesController.java Jeep.java JeepModel.java
1 package com.promineotech.jeep.controller;
2
3
4 import java.util.List;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RequestParam;
9 import org.springframework.web.bind.annotation.ResponseStatus;
10 import com.promineotech.jeep.entity.Jeep;
11 import com.promineotech.jeep.entity.JeepModel;
12 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
13 import io.swagger.v3.oas.annotations.Operation;
14 import io.swagger.v3.oas.annotations.Parameter;
15 import io.swagger.v3.oas.annotations.info.Info;
16 import io.swagger.v3.oas.annotations.media.Content;
17 import io.swagger.v3.oas.annotations.media.Schema;
18 import io.swagger.v3.oas.annotations.responses.ApiResponse;
19 import io.swagger.v3.oas.annotations.servers.Server;
20
21 @RequestMapping("/jeeps")
22 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
23     @Server(url = "http://localhost:8080", description = "Local server.")})
24
25
26 public interface JeepSalesController {
27
28     @Operation(
29         summary = "Returns a list of Jeeps",
30         description = "Returns a list of Jeeps given an optional model and/or trim.",
31         responses = {
32             @ApiResponse(responseCode = "200",
33                 description = "A list of Jeeps is returned",
34                 content = @Content(mediaType = "application/json",
35                     schema = @Schema(implementation = Jeep.class))),
36             @ApiResponse(responseCode = "400",
37                 description = "The request parameters are invalid",
38                 content = @Content(mediaType = "application/json")),
39             @ApiResponse(responseCode = "404",
40                 description = "No Jeeps were found with the input criteria",
41                 content = @Content(mediaType = "application/json")),
42             @ApiResponse(responseCode = "500",
43                 description = "An unplanned error occurred",
44                 content = @Content(mediaType = "application/json"))
45         },
46         parameters = {
47             @Parameter(name = "model",
48                 allowEmptyValue = false,
49                 required = false,
50                 description = "The model name (i.e., 'WRANGLER'))",
51             @Parameter(name = "trim",
52                 allowEmptyValue = false,
53                 required = false,
54                 description = "The trim level (i.e., 'Sport'))"
55         }
56     )
57
58     @GetMapping
59     @ResponseStatus(code = HttpStatus.OK)
60     List<Jeep> fetchJeeps(
61         @RequestParam JeepModel model,
62         @RequestParam String trim);
63 }
64
65 }
```

Jeep.java X JeepModel.java

```
1+ /**
2  package com.promineotech.jeep.entity;
3
4  import java.math.BigDecimal;
5
6  import lombok.AllArgsConstructor;
7  import lombok.Builder;
8  import lombok.Data;
9  import lombok.NoArgsConstructor;
10
11
12
13 /**
14  * @author gerriciancanelli
15  *
16  */
17
18 @Data
19 @Builder
20 @NoArgsConstructor
21 @AllArgsConstructor
22 public class Jeep {
23     private Long modelPK;
24     private JeepModel modelId;
25     private String trimLevel;
26     private int numDoors;
27     private int wheelSize;
28     private BigDecimal basePrice;
29 }
```

DefaultJeepSalesController.java X Jeep.java JeepModel.java

```
1 package com.promineotech.jeep.controller;
2 import java.util.List;
3
4
5
6
7 @RestController
8 public class DefaultJeepSalesController implements JeepSalesController {
9
10    @Override
11    public List<Jeep> fetchJeeps(JeepModel model, String trim) {
12        return null;
13    }
14
15 }
```

```
1+ package com.promineotech.jeep.entity;
2
3
4  public enum JeepModel {
5      GRAND_CHEROKEE,
6      CHEROKEE,
7      COMPASS,
8      RENEGADE,
9      WRANGLER,
10     GLADIATOR,
11     WRANGLER_4XE
12 }
13
14
15
16
17
18
19
20
```

Screenshots of Running Application:

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

GET /jeeps Returns a list of Jeeps

Returns a list of Jeeps given an optional model and/or trim.

Parameters

Name **Description**

model * required string (query) The model name (i.e., 'WRANGLER') Available values : GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, WRANGLER, GLADIATOR, WRANGLER_4XE

GRAND_CHEROKEE

trim * required string (query) The trim level (i.e., 'Sport')

trim

Try it out

Responses

Code	Description	Links
200	A list of Jeeps is returned	No links
	Media type application/json	
	Controls Accept header.	
	Example Value Schema	
	{}	
400	The request parameters are invalid	No links
	Media type application/json	
404	No Jeeps were found with the input criteria	No links
	Media type application/json	
500	An unplanned error occurred	No links

JUnit X | Package Explorer | Console | JUnit Test [Runner: JUnit 5] (0.474 s)

Finished after 2.909 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

Java - Java Virtual Machine [jdk-11.0.15] (Sep 25, 2022, 11:06:17 PM) [pid: 17449]

23:06:18.232 [thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.classloader.RestartClassLoader@77

Spring Boot :: (v2.7.4)

```

2022-09-25 23:06:18.467 INFO 17449 --- [ restartedMain] com.promineotech.jeep.JeepSales           : Starting Jeepsales using Java 11.0.15 on Gerric-Mac-Pro-2.local with PID 17449 (/Users/gerriccananelli/Desktop)
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] com.promineotech.jeep.JeepSales           : No active profile set, falling back to default profile: "default"
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active. Set 'spring.devtools.add-properties' to 'false' to disable
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8088 (http)
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-25 23:06:18.500 INFO 17449 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : ContextListener: contextInitialized()
2022-09-25 23:06:19.152 INFO 17449 --- [ restartedMain] w.a.S.DefaultWebServerApplicationContext : Root WebApplicationContext: initialization completed in 652 ms
2022-09-25 23:06:19.596 INFO 17449 --- [ restartedMain] o.s.b.d.OptionalLiveReloadServer        : LiveReload server is running on port 35729
2022-09-25 23:06:19.622 INFO 17449 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8088 (http) with context path ''
2022-09-25 23:06:19.621 INFO 17449 --- [ restartedMain] com.promineotech.jeep.JeepSales           : Started Jeepsales in 1.391 seconds (JVM running for 1.857)

```

FetchJeepTest [Runner: JUnit 5] (0.474 s)

FetchJeepTest.java | JeepSales.java | jeep-sales/pom.xml | JeepSalesController.java | DefaultJeepSalesController.java | Jeep.java | JeepModel.java

```

1 package com.promineotech.jeep;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class JeepSales {
8
9     public static void main(String[] args) {
10         SpringApplication.run(JeepSales.class, args);
11     }
12 }
13
14
15

```

URL to GitHub Repository:

<https://github.com/geraldinedepaul17/BackEndClass/tree/main/Week%2013%20Homework>