

Web API Design with Spring Boot Week 13 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/DavidSteffen1/weekThirteenRepository>


URL to Public Link of your Video: <https://youtu.be/lmxHNX7x2yo>

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Create a new repository on GitHub for this week's assignment and push your completed code to this dedicated repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.


2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-

Web API Design with Spring Boot Week 13 Coding Assignment

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 Prompt 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>

Web API Design with Spring Boot Week 13 Coding Assignment

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	<code>com.promineotech</code>
Artifact Id	<code>jeep-sales</code>

a)

Click "Finish".

- 1) Navigate to the Spring Initializr (<https://start.spring.io/>).
 - d) Confirm the following settings:

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

- a) Add the dependencies from the Initializr:
 - d.i) Web

Web API Design with Spring Boot Week 13 Coding Assignment

- d.ii) Devtools
 - d.iii) Lombok
 - e) Click "Explore" at the bottom of the page.
 - f) Click "Copy" to copy the pom.xml generated by the Initializr to the clipboard.
- 1) 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.
 - 2) 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.
 - 3) Create a package in src/main/java named com.promineotech.jeeep. 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.jeeep;

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);
    }
}
```
 - 4) 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".

Web API Design with Spring Boot Week 13 Coding Assignment

- 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".
- 5) Using the MySQL Workbench or MySQL command line client (CLI), create a database named "jeep".
- 6) 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.
- 7) 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:
- 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;
```

Web API Design with Spring Boot Week 13 Coding Assignment

```
@LocalServerPort
```

```
private int serverPort;
```

- 8) Create a new package in `src/main/java` named `com.promineotech.jeeep.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`.
- 9) Create a `Jeep` class in the `com.promineotech.jeeep.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;  
}
```

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

Web API Design with Spring Boot Week 13 Coding Assignment

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

1)

- 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`.

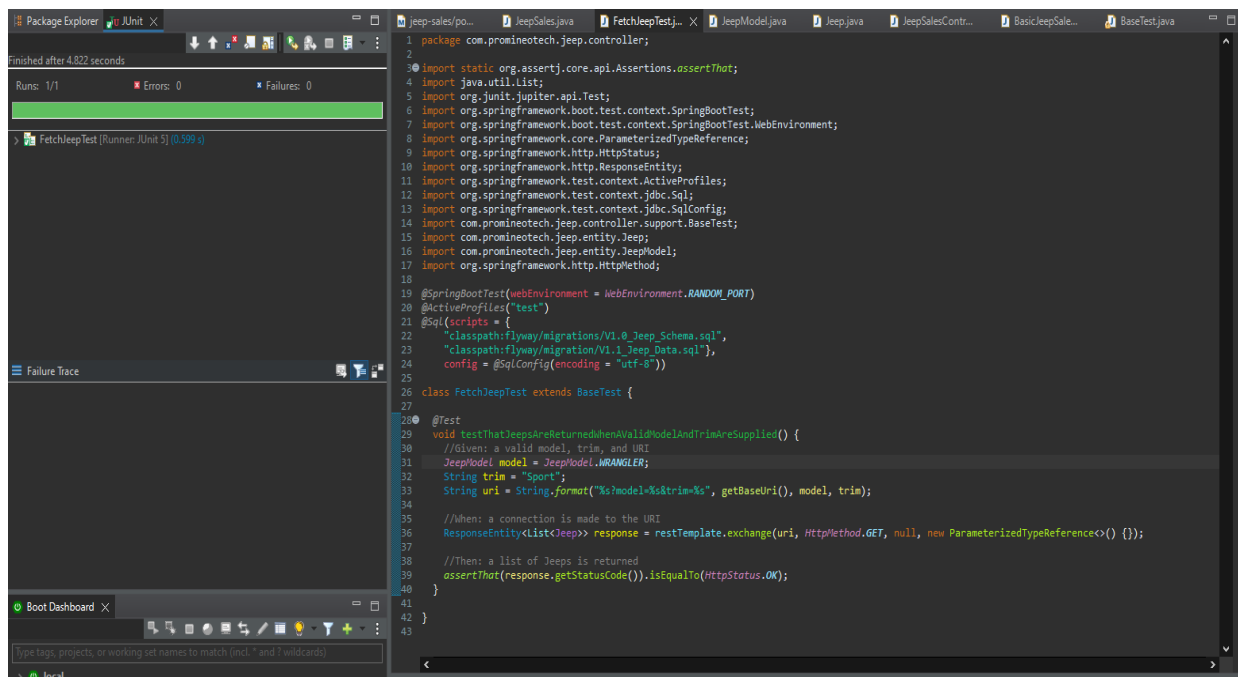
- b) Using [AssertJ](#), test that the response that comes back from the server is 200 (success) – or as is shown in the video: `HttpStatus.OK`. The code should look like this:

```
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. 



The screenshot shows an IDE with a test class `FetchJeepTest` and its execution results. The test class is located in `com.promineotech.jeeptest` and extends `BaseTest`. It contains a test method `testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied()` which uses `restTemplate` to send an HTTP GET request to the server and asserts that the response status is `HttpStatus.OK`. The test class is annotated with `@SpringBootTest` and `@ActiveProfiles("test")`. The execution results show that the test passed successfully.


```
1 package com.promineotech.jeeptest;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import java.util.List;
5 import org.junit.jupiter.api.Test;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
8 import org.springframework.core.ParameterizedTypeReference;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.http.ResponseEntity;
11 import org.springframework.test.context.ActiveProfiles;
12 import org.springframework.test.context.jdbc.Sql;
13 import org.springframework.test.context.jdbc.SqlConfig;
14 import com.promineotech.jeeptest.support.BaseTest;
15 import com.promineotech.jeeptest.entity.Jeep;
16 import com.promineotech.jeeptest.entity.JeepModel;
17 import org.springframework.http.HttpMethod;
18
19 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
20 @ActiveProfiles("test")
21 @Sql(scripts = {
22     "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
23     "classpath:flyway/migration/V1.1_Jeep_Data.sql"},
24     config = @SqlConfig(encoding = "utf-8"))
25
26 class FetchJeepTest extends BaseTest {
27
28     @Test
29     void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
30         //Given: a valid model, trim, and URI
31         JeepModel model = JeepModel.WRANGLER;
32         String trim = "Sport";
33         String uri = String.format("%s?model=%s&trim=%s", getBaseUrl(), model, trim);
34
35         //When: a connection is made to the URI
36         ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
37
38         //Then: a list of Jeeps is returned
39         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
40     }
41 }
42
43
```

Web API Design with Spring Boot Week 13 Coding Assignment

- 2) In `src/main/java`, create a new package `com.promineotech.jeeep.controller`. In this package, create an interface named `JeepSalesController`.
- a) Add the class-level annotation `@RequestMapping("/jeeeps")`.
 - b) 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`.
 - c) 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.
 - d) Add the parameter annotations in the OpenAPI documentation to describe the `model` and `trim` parameters.
 - e) Add the `@GetMapping` annotation and the `@ResponseStatus(code = HttpStatus.OK)` annotation as method-level annotations to the `fetchJeeps` method.
 - f) Add the `@RequestParam` annotations to the parameters as described in the video. The interface should look like this (omitting the OpenAPI annotations):

```
@RequestMapping("/jeeeps")  
  
public interface JeepSalesController {  
    @GetMapping  
    @ResponseStatus(code = HttpStatus.OK)  
    List<Jeep> fetchJeeps(@RequestParam JeepModel model,  
        @RequestParam String trim);  
}
```
 - g) Produce a screenshot showing the interface and OpenAPI documentation. 

Web API Design with Spring Boot Week 13 Coding Assignment

```
20 @RequestMapping("/jeeps")
21
22 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
23     @Server(url = "http://localhost:8080", description = "Local server.")})
24
25 public interface JeepSalesController {
26     // @formatter:off
27     @Operation(
28         summary = "Returns a list of Jeeps",
29         description = "Returns a list of Jeeps given an optional model and/or trim",
30         responses = {
31             @ApiResponse(responseCode = "200",
32                 description = "A list of Jeeps is returned",
33                 content = @Content(mediaType = "application/json",
34                     schema = @Schema(implementation = Jeep.class))),
35             @ApiResponse(responseCode = "400",
36                 description = "The request parameters are invalid",
37                 content = @Content(mediaType = "application/json")),
38             @ApiResponse(responseCode = "404",
39                 description = "No Jeeps were found with the input criteria",
40                 content = @Content(mediaType = "application/json")),
41             @ApiResponse(responseCode = "500",
42                 description = "An unplanned error occurred.",
43                 content = @Content(mediaType = "application/json"))
44         },
45         parameters = {
46             @Parameter(name = "model",
47                 allowEmptyValue = false,
48                 required = false,
49                 description = "The model name"),
50             @Parameter(name = "trim",
51                 allowEmptyValue = false,
52                 required = false,
53                 description = "The trim level")
54         }
55     )
56
57     @GetMapping
58     @ResponseStatus(code = HttpStatus.OK)
59     List<Jeep> fetchJeeps(@RequestParam(required = false) JeepModel model, @RequestParam(required = false) String trim);
60     // @formatter:on
61 }
```

- 12) Add the controller implementation class named DefaultJeepSalesController. Don't forget the @RestController annotation.
- 13) 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. 

Web API Design with Spring Boot Week 13 Coding Assignment

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

(query)

trim

Responses

Code	Description	Links
200	A list of Jeeps is returned	No links
<p>Media type</p> <p>application/json</p> <p>Controls Accept header</p> <p>Example Value Schema</p> <pre>{ "modelPK": 0, "modelId": "WRANGLER", "trimLevel": "string", "numDoors": 0, "wheelSize": 0, "basePrice": 0}</pre>		
400	The request parameters are invalid	No links
<p>Media type</p> <p>application/json</p>		
404	No Jeeps were found with the input criteria	No links
<p>Media type</p> <p>application/json</p>		
500	An unplanned error occurred.	No links
<p>Media type</p> <p>application/json</p>		