

Web API Design with Spring Boot Week 13 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/Xelyen/Week-13/tree/main/jeep-sales>


URL to Public Link of your Video: <https://www.youtube.com/watch?v=K2xKzbnXRg>

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>

- d)
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	<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)

Web API Design with Spring Boot Week 13 Coding Assignment

- 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.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);
```

```
    }
```

```
}
```

- 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.**

Web API Design with Spring Boot Week 13 Coding Assignment

- 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
```

Web API Design with Spring Boot Week 13 Coding Assignment

```
private TestRestTemplate restTemplate;
```

```
@LocalServerPort
```

```
private int serverPort;
```

- 10) 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.
- 11) 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;  
}
```

- 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.**

Web API Design with Spring Boot Week 13 Coding Assignment

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

14)

- 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:

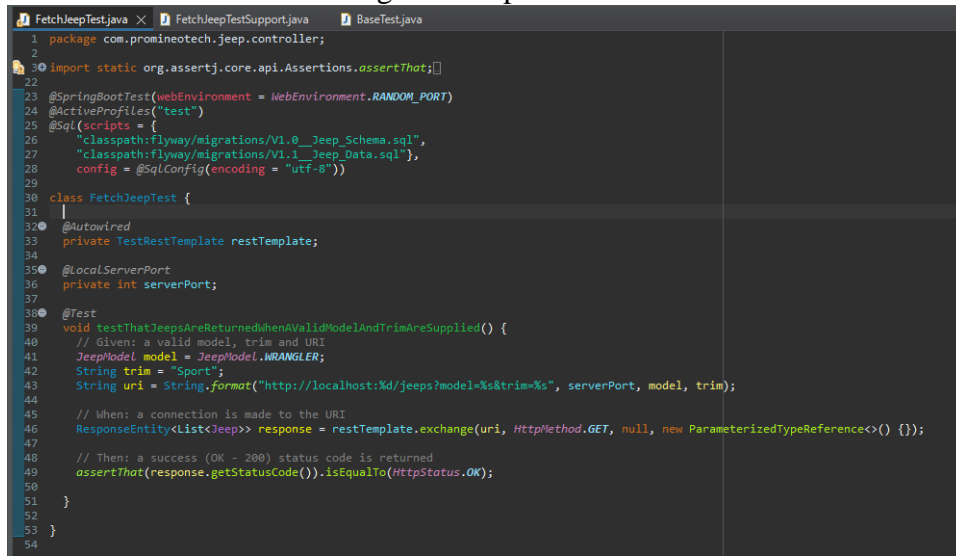
```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
```

Use the import statements:

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

Web API Design with Spring Boot Week 13 Coding Assignment

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



```
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 {
12     @Autowired
13     private TestRestTemplate restTemplate;
14
15     @LocalServerPort
16     private int serverPort;
17
18     @Test
19     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
20         // Given: a valid model, trim and URI
21         JeepModel model = JeepModel.WRANGLER;
22         String trim = "Sport";
23         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
24
25         // When: a connection is made to the URI
26         ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
27
28         // Then: a success (OK - 200) status code is returned
29         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
30     }
31 }
32
33 }
```

- 15) In src/main/java, create a new package com.promineotech.jeeptest.controller. In this package, create an interface named JeepSalesController.

- a) Add the class-level annotation `@RequestMapping("/jeeps")`.
- 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("/jeeps")
public interface JeepSalesController {
    @GetMapping
    @ResponseStatus(code = HttpStatus.OK)
    List<Jeep> fetchJeeps(@RequestParam JeepModel model,
        @RequestParam String trim);
}
```


Web API Design with Spring Boot Week 13 Coding Assignment

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

```
FetchJeepTest.java  JeepSalesController.java X
30 import java.util.List;
18
19 @RequestMapping("/jeeps")
20 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
21     @Server(url = "http://localhost:8080", description = "Local server.")
22 })
23 public interface JeepSalesController {
24     // @formatter:off
25     @Operation(
26         summary = "Returns a list of Jeeps",
27         description = "Returns a list of Jeeps given an optional model and/or trim",
28         responses = {
29             @ApiResponse(
30                 responseCode = "200",
31                 description = "A list of Jeeps is returned",
32                 content = @Content(
33                     mediaType = "application/json",
34                     schema = @Schema(implementation = Jeep.class)),
35             @ApiResponse(
36                 responseCode = "400",
37                 description = "The request parameters are invalid",
38                 content = @Content(mediaType = "application/json")),
39             @ApiResponse(
40                 responseCode = "404",
41                 description = "No Jeeps were found with the input criteria",
42                 content = @Content(mediaType = "application/json")),
43             @ApiResponse(
44                 responseCode = "500",
45                 description = "An unplanned error occurred.",
46                 content = @Content(mediaType = "application/json"))
47         },
48         parameters = {
49             @Parameter(
50                 name = "model",
51                 allowEmptyValue = false,
52                 required = false,
53                 description = "The model name (i.e., 'WRANGLER')"),
54             @Parameter(
55                 name = "trim",
56                 allowEmptyValue = false,
57                 required = false,
58                 description = "The trim level (i.e., 'Sport')")
59         }
60     )
61     @GetMapping
62     @ResponseStatus(code = HttpStatus.OK)
63     List<Jeep> fetchJeeps(
64         @RequestParam(required = false)
65         String model,
66         @RequestParam(required = false)
67         String trim);
68     // @formatter:on
69 }
```

- 16) Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.
- 17) Run the application within the IDE and show the resulting OpenAPI (Swagger) documentation produced in the browser. Produce a screenshot of the documentation showing

Web API Design with Spring Boot Week 13 Coding Assignment

all four possible outcomes. 

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