

# Web API Design with Spring Boot Week 13 Coding Assignment

Points possible: 75


URL to GitHub Repository: <https://github.com/Bwcash/Web-API-Spring-Boot.git>

URL to Public Link of your Video: <https://youtu.be/PYCq3Gh50X4>


---

## 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:

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

a)

Click "Finish".

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

- a) Confirm the following settings:

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

## Web API Design with Spring Boot Week 13 Coding Assignment

- a) Add the dependencies from the Initializr:
  - i) Web
  - ii) Devtools
  - iii) Lombok
- b) Click "Explore" at the bottom of the page.
- c) 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.**

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

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. 

# Web API Design with Spring Boot Week 13 Coding Assignment

```
*FetchJeepTest.java x JeepModel.java Jeep.java
1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import static org.junit.jupiter.api.Assertions.*;
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.context.SpringBootTest.WebEnvironment;
12 import org.springframework.boot.test.web.client.TestRestTemplate;
13 import org.springframework.boot.test.web.server.LocalServerPort;
14 import org.springframework.core.ParameterizedTypeReference;
15 import org.springframework.http.HttpMethod;
16 import org.springframework.http.HttpStatus;
17 import org.springframework.http.ResponseEntity;
18 import org.springframework.test.context.ActiveProfiles;
19 import org.springframework.test.context.jdbc.Sql;
20 import org.springframework.test.context.jdbc.SqlConfig;
21
22 import com.promineotech.jeep.entity.Jeep;
23 import com.promineotech.jeep.entity.JeepModel;
24 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
25 @ActiveProfiles("test")
26 @Sql(scripts = {
27     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
28     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
29     config = @SqlConfig(encoding = "utf-8"))
30 class FetchJeepTest {
31     @Autowired
32     private TestRestTemplate restTemplate;
33     @LocalServerPort
34     private int serverPort;
35
36     @Test
37     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
38         // given: a valid model, trim, and URI
39         JeepModel model = JeepModel.WRANGLER;
40         String trim = "Sport";
41         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
42
43         // when: a connection is made to the URI
44         ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
45
46         // then: a success (OK - 200) status code is returned
47         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
48     }
49 }
50
```


- 2) In src/main/java, create a new package com.promineotech.jeep.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.



## Web API Design with Spring Boot Week 13 Coding Assignment


- 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);  
}
```

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

# Web API Design with Spring Boot Week 13 Coding Assignment

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

- 3) Add the controller implementation class named DefaultJeepSalesController. Don't forget the @RestController annotation.
- 4) 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

The screenshot displays a web browser window with the Swagger UI interface for a Spring Boot application. The browser's address bar shows the URL: `localhost:8080/swagger-ui/index.html#/default-jeep-sales-controller/fetchJeeps`. The interface lists several API endpoints with their status codes, descriptions, and media types.

Status Code	Description	Media Type	Links
200	A list of Jeeps is returned	<input type="text" value="application/json"/>	No links
400	The request parameters are invalid	<input type="text" value="application/json"/>	No links
404	No Jeeps were found with the input criteria	<input type="text" value="application/json"/>	No links
500	An unplanned error occurred.	<input type="text" value="application/json"/>	No links

The 200 response section includes an "Example Value" field showing a JSON object:

```
{  "modelPk": 0,  "modelId": "WRANGLER",  "trimLevel": "string",  "numDoors": 0,  "wheelSize": 0,  "basePrice": 0}
```

The browser's taskbar at the bottom shows the system time as 11:20 AM on 12/15/2022, along with various application icons and a search bar.