

# Web API Design with Spring Boot Week 13 Coding Assignment

Points possible: 75

URL to GitHub Repository:


[https://github.com/DracaQueen/ProminenoTech\\_SpringbootProject](https://github.com/DracaQueen/ProminenoTech_SpringbootProject)

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


---

## 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>

- d)  
Click "Finish".

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

Project	Maven Project
<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

- 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.jeepp. 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.jeepp;

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>

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;
```

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

- 15) 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.

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

- 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);

}
```

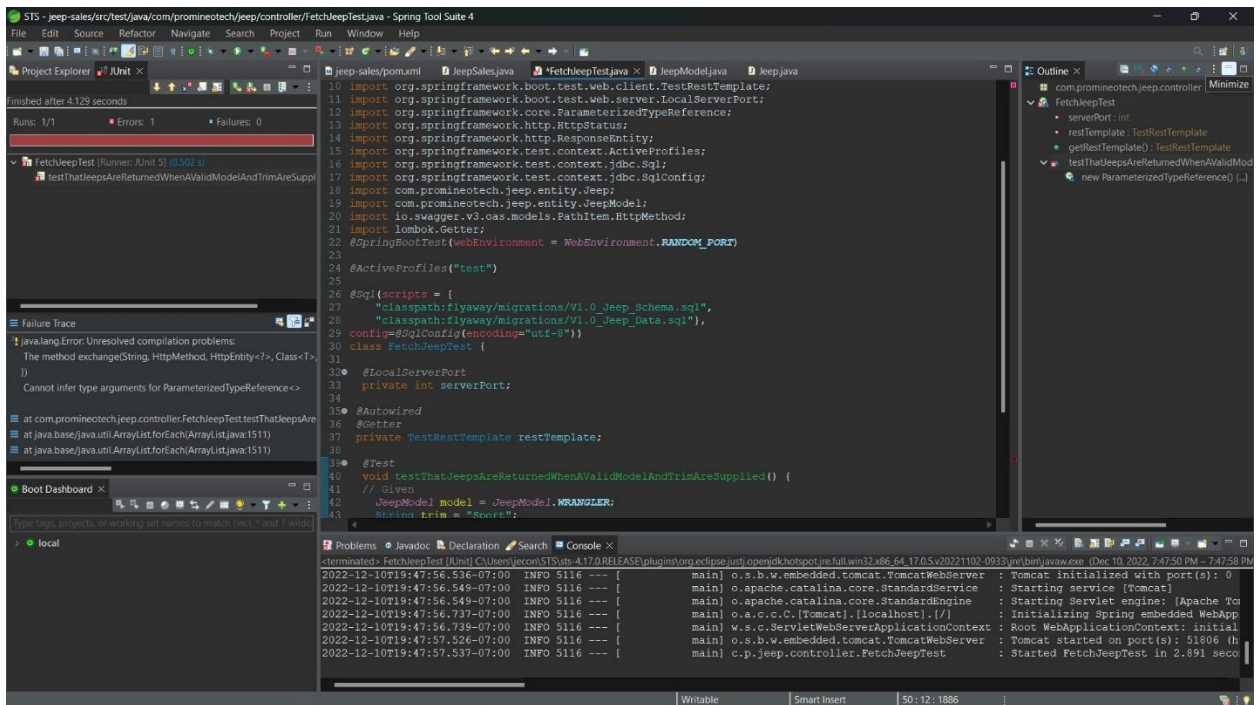
- g) Produce a screenshot showing the interface and OpenAPI documentation. 
- 16) Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.
- 17) Run the application within the IDE and show the resulting OpenAPI (Swagger) documentation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



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

```
File Edit Source Refactor Navigate Search Project Run Window Help
jeep-sales/pom.xml JeepSales.java *FetchJeepTest.java x JeepModel.java Jeep.java
10 import org.springframework.boot.test.web.client.TestRestTemplate;
11 import org.springframework.boot.test.web.server.LocalServerPort;
12 import org.springframework.core.ParameterizedTypeReference;
13 import org.springframework.http.HttpStatus;
14 import org.springframework.http.ResponseEntity;
15 import org.springframework.test.context.ActiveProfiles;
16 import org.springframework.test.context.jdbc.Sql;
17 import org.springframework.test.context.jdbc.SqlConfig;
18 import com.promineotech.jeepp.entity.Jeep;
19 import com.promineotech.jeepp.entity.JeepModel;
20 import io.swagger.v3.oas.models.PathItem.HttpMethod;
21 import lombok.Getter;
22 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
23
24 @ActiveProfiles("test")
25
26 @Sql(scripts = {
27     "classpath:flyaway/migrations/V1.0_Jeep_Schema.sql",
28     "classpath:flyaway/migrations/V1.0_Jeep_Data.sql"},
29     config=@SqlConfig(encoding="utf-8"))
30 class FetchJeepTest {
31
32     @LocalServerPort
33     private int serverPort;
34
35     @Autowired
36     @Getter
37     private TestRestTemplate restTemplate;
38
39     @Test
40     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
41         // Given
42         JeepModel model = JeepModel.WRANGLER;
43         String trim = "Sport";
44         String uri =
45             String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
46
47         // When
48         ResponseEntity<List<Jeep>>response =
49             restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>(){});
50         // Then
51         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
52     }
53 }
54
55
```

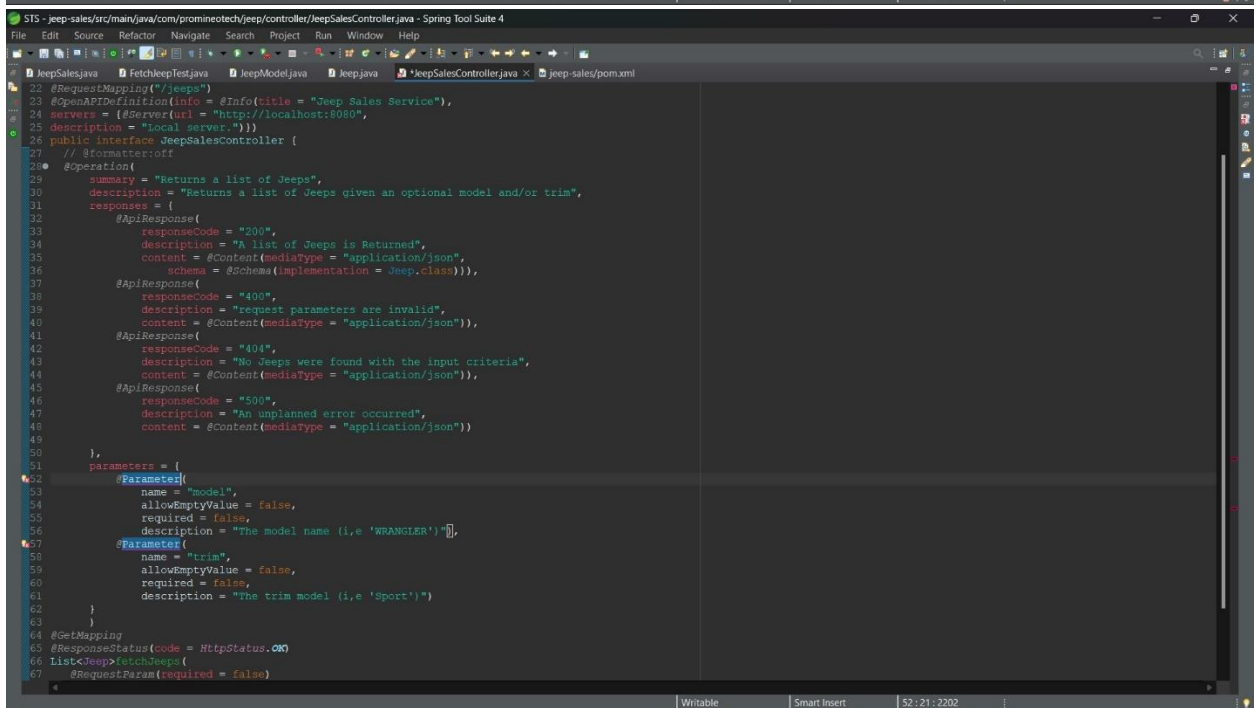
# Web API Design with Spring Boot Week 13 Coding Assignment



The screenshot shows an IDE window titled "STS - jeep-sales/src/test/java/com/promineotech/jeep/controller/JeepControllerTest.java - Spring Tool Suite 4". The main editor displays the test class code, which includes imports for Spring Boot test utilities, a database configuration, and a test method. The test method is annotated with `@SpringBootTest` and `@ActiveProfiles("test")`. It uses `@Sql` to execute SQL scripts before the test. The test method itself is annotated with `@Test` and `@Autowired`, and it uses `JeepModel.WRANGLER` and `JeepModel.Sport` as test data.

```
10 import org.springframework.boot.test.web.client.TestRestTemplate;
11 import org.springframework.boot.test.web.server.LocalServerPort;
12 import org.springframework.core.ParameterizedTypeReference;
13 import org.springframework.http.HttpStatus;
14 import org.springframework.http.ResponseEntity;
15 import org.springframework.test.context.ActiveProfiles;
16 import org.springframework.test.context.jdbc.Sql;
17 import org.springframework.test.context.jdbc.SqlConfig;
18 import com.promineotech.jee.entity.Jee;
19 import com.promineotech.jee.entity.JeeModel;
20 import io.swagger.v3.oas.models.PathItem.HttpMethod;
21 import lombok.Getter;
22 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
23
24 @ActiveProfiles("test")
25
26 @Sql(scripts = {
27     "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
28     "classpath:flyway/migrations/V1.0_Jeep_Data.sql"),
29 config=@SqlConfig(encoding="utf-8"))
30 class FetchJeepTest {
31
32     @LocalServerPort
33     private int serverPort;
34
35     @Autowired
36     @Setter
37     private TestRestTemplate restTemplate;
38
39     @Test
40     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
41         // Given
42         JeepModel model = JeepModel.WRANGLER;
43         String trim = "Sport";
```

The left sidebar shows the "Problems" view with a list of errors, including "java.lang.Error: Unresolved compilation problems: The method exchange(String, HttpMethod, HttpEntity?)>, Class<T> is not applicable for the argument types (String, HttpMethod, HttpEntity?) and the bounds of the argument list". The bottom sidebar shows the "Console" view with the output of the test execution, including the startup of the Tomcat server and the execution of the test method.



The screenshot shows an IDE window titled "STS - jeep-sales/src/main/java/com/promineotech/jeep/controller/JeepSalesController.java - Spring Tool Suite 4". The main editor displays the controller class code, which includes a `RequestMapping` annotation, a `GetMapping` annotation, and a `GetMapping` annotation. The controller class is annotated with `@RequestMapping` and `@GetMapping`, and it uses `JeepModel.WRANGLER` and `JeepModel.Sport` as test data.

```
22 @RequestMapping("/jeeps")
23 @GetMapping
24 public interface JeepSalesController {
25     // @PostMapping
26     // @PostMapping
27     // @PostMapping
28     // @PostMapping
29     // @PostMapping
30     // @PostMapping
31     // @PostMapping
32     // @PostMapping
33     // @PostMapping
34     // @PostMapping
35     // @PostMapping
36     // @PostMapping
37     // @PostMapping
38     // @PostMapping
39     // @PostMapping
40     // @PostMapping
41     // @PostMapping
42     // @PostMapping
43     // @PostMapping
44     // @PostMapping
45     // @PostMapping
46     // @PostMapping
47     // @PostMapping
48     // @PostMapping
49     // @PostMapping
50     // @PostMapping
51     // @PostMapping
52     // @PostMapping
53     // @PostMapping
54     // @PostMapping
55     // @PostMapping
56     // @PostMapping
57     // @PostMapping
58     // @PostMapping
59     // @PostMapping
60     // @PostMapping
61     // @PostMapping
62     // @PostMapping
63     // @PostMapping
64     // @PostMapping
65     // @PostMapping
66     // @PostMapping
67     // @PostMapping
68     // @PostMapping
69     // @PostMapping
70     // @PostMapping
71     // @PostMapping
72     // @PostMapping
73     // @PostMapping
74     // @PostMapping
75     // @PostMapping
76     // @PostMapping
77     // @PostMapping
78     // @PostMapping
79     // @PostMapping
80     // @PostMapping
81     // @PostMapping
82     // @PostMapping
83     // @PostMapping
84     // @PostMapping
85     // @PostMapping
86     // @PostMapping
87     // @PostMapping
88     // @PostMapping
89     // @PostMapping
90     // @PostMapping
91     // @PostMapping
92     // @PostMapping
93     // @PostMapping
94     // @PostMapping
95     // @PostMapping
96     // @PostMapping
97     // @PostMapping
98     // @PostMapping
99     // @PostMapping
100    // @PostMapping
```