


# 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:

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

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>	com.promineotech
<b>Artifact</b>	jeep-sales
<b>Name</b>	jeep-sales
<b>Description</b>	Jeep Sales
<b>Package name</b>	com.promineotech
<b>Packaging</b>	Jar
<b>Java</b>	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.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.**
  - 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 "jeeep".
- 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.jeeep.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`.

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

- 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. 

```

1 package com.promineotech.jeep.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_JeepSchema.sql"},
10     config = @SqlConfig(encoding = "utf-8"))
11 class FetchJeepTest {
12     @LocalServerPort
13     private int serverPort;
14
15     @Autowired
16     private TestRestTemplate restTemplate;
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/models?model=%s&trim=%s",
24             serverPort, model, trim);
25
26         // When a connection is made to the URI...
27         ResponseEntity<List<Jeep>> response =
28             restTemplate.exchange(uri, HttpMethod.GET, null,
29                 new ParameterizedTypeReference<>() {});
30
31         // Then a success (OK - 200) status code is returned.
32         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
33     }
34 }

```

- 14) 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.
  - 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);
}

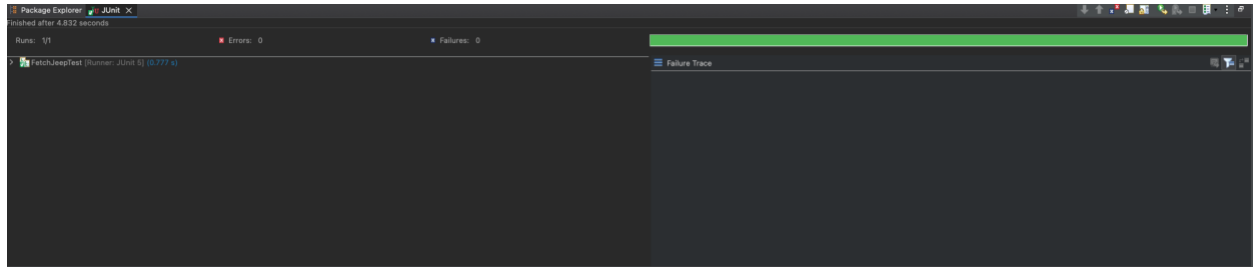
```

}

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

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. 🖥️



## Whitelabel Error Page

This application has no explicit mapping for `/error`, so you are seeing this as a fallback.

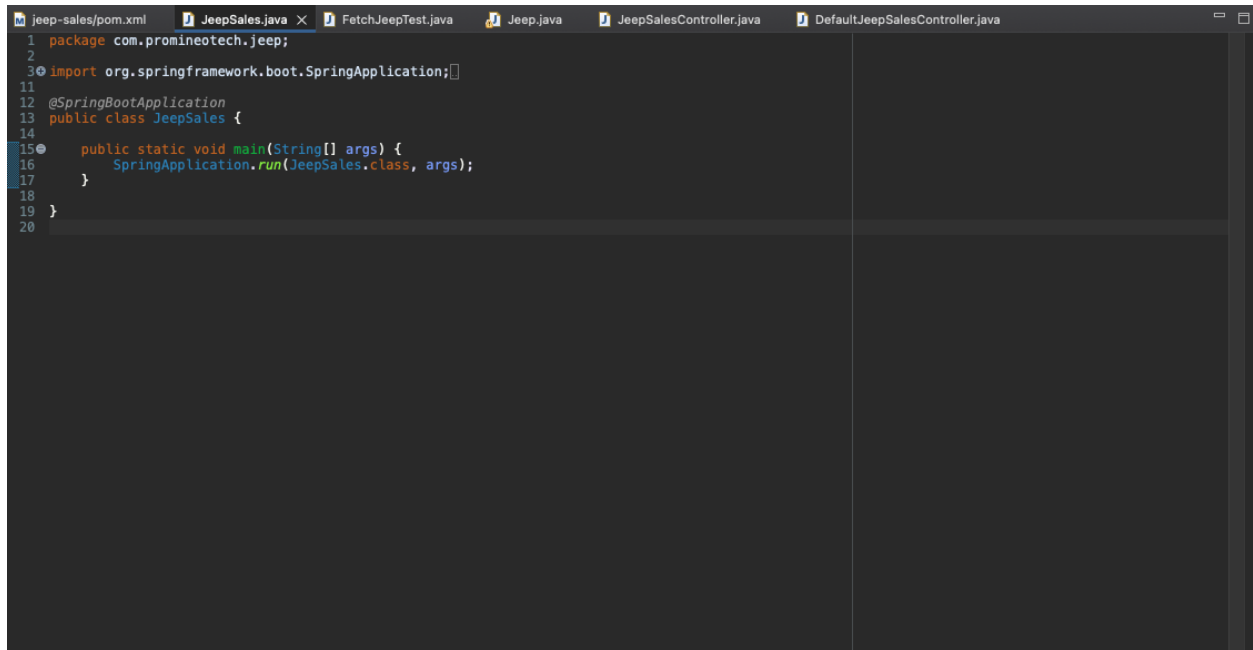
Fri Sep 23 22:22:36 EDT 2022

There was an unexpected error (type=Not Found, status=404).

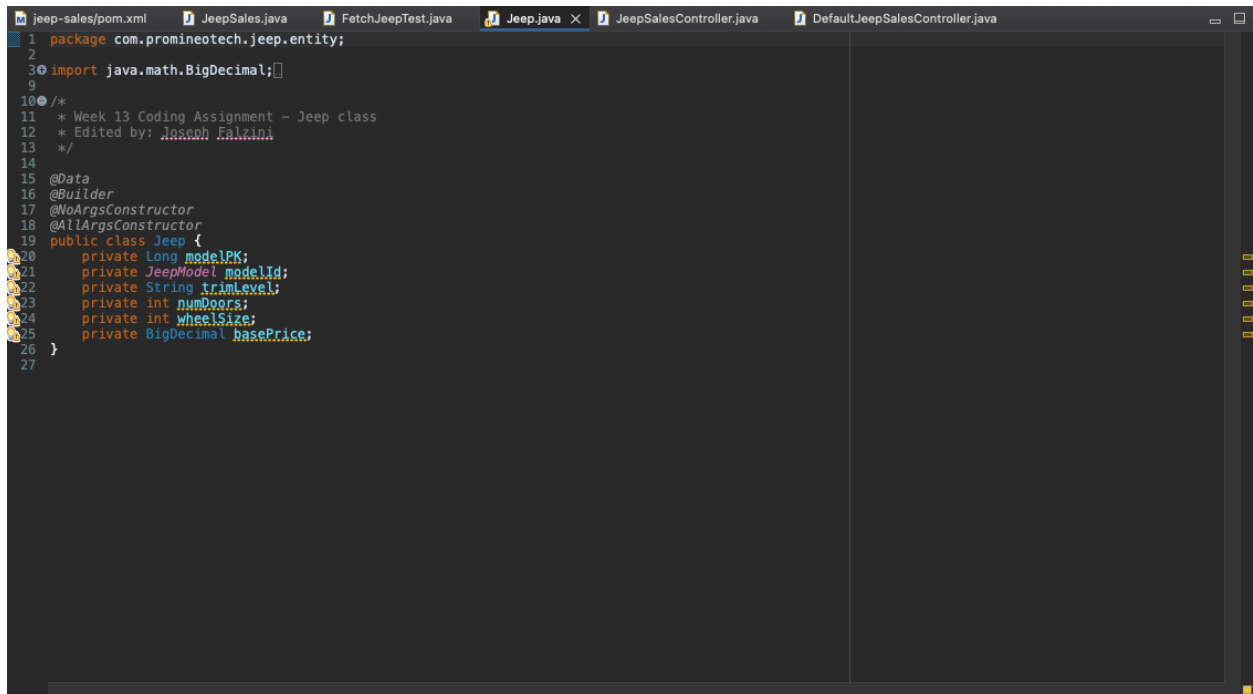
No message available

**\*MEETING WITH A MENTOR ON TUESDAY TO HOPEFULLY FIX THIS ERROR PAGE!!!**

## Screenshots of Code:



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



```
1 package com.promineotech.jeepp.entity;
2
3 import java.math.BigDecimal;
4
5 /*
6  * Week 13 Coding Assignment - Jeep class
7  * Edited by: Joseph Falchini
8  */
9
10 @Data
11 @Builder
12 @NoArgsConstructor
13 @AllArgsConstructor
14 public class Jeep {
15     private Long modelPK;
16     private JeepModel modelId;
17     private String trimLevel;
18     private int numDoors;
19     private int wheelSize;
20     private BigDecimal basePrice;
21 }
```



```
jeep-sales/pom.xml JeepSales.java FetchJeepTest.java Jeep.java JeepSalesController.java X DefaultJeepSalesController.java
1 package com.promineotech.jeeo.controller;
2
3 import java.util.List;
4
5 // Week 13 Coding Assignment - JeepSalesController Interface
6 // Edited by: Jasson Jfalzini
7
8
9 @RequestMapping("/jeeps")
10 @RequestMappingInfo({info = @RequestMappingInfo(servers = {
11     @Server(url = "http://localhost:8080", description = "Local Server")
12 })
13 })
14 public interface JeepSalesController {
15     // @formatter:off
16     @Operation(summary = "Returns a list of Jeeps",
17         description = "Returns a list of Jeeps given an option model and/or trim",
18         responses = {
19             @ApiResponse(responseCode = "200",
20                 description = "A list of Jeeps is returned",
21                 content = @Content(mediaType = "application/json",
22                     schema = @Schema(implementation = Jeep.class))),
23             @ApiResponse(responseCode = "400",
24                 description = "The request parameters are invalid",
25                 content = @Content(mediaType = "application/json")),
26             @ApiResponse(responseCode = "404",
27                 description = "No Jeeps were found with the input criteria",
28                 content = @Content(mediaType = "application/json")),
29             @ApiResponse(responseCode = "500",
30                 description = "An unplanned error occurred",
31                 content = @Content(mediaType = "application/json"))
32         },
33         parameters = {
34             @Parameter(name = "model",
35                 allowEmptyValue = false,
36                 required = false,
37                 description = "The model name (i.e., 'MUNGLER')",
38             ),
39             @Parameter(name = "trim",
40                 allowEmptyValue = false,
41                 required = false,
42                 description = "The trim level (i.e., 'Sport')")
43         }
44     )
45     // @formatter:on
46     @GetMapping
47     @ResponseStatus(code = HttpStatus.OK)
48     List<Jeep> fetchJeeps(@RequestParam(required = false) JeepModel model, @RequestParam(required = false) String trim);
49 }
50
```

```
jeep-sales/pom.xml JeepSales.java FetchJeepTest.java Jeep.java JeepSalesController.java X DefaultJeepSalesController.java X
1 package com.promineotech.jeeo.controller;
2
3 import java.util.List;
4
5 import org.springframework.web.bind.annotation.RestController;
6
7 import com.promineotech.jeeo.entity.Jeep;
8 import com.promineotech.jeeo.entity.JeepModel;
9
10 @RestController
11 public class DefaultJeepSalesController implements JeepSalesController {
12
13     // @formatter:off
14     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
15         // TODO Auto-generated method stub
16         return null;
17     }
18 }
19
20
```

**Screenshots of Running Application:** \*Same as last screenshot of Whitelabel error page and JUnit Test Case results.

**URL to GitHub Repository:**

**<https://github.com/jfalzini99/jeep-sales>**