


# 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:

- Create a Maven project named `JeepSales` as described in the video.

- In Spring Tool Suite, click the "File" menu. Select "New/Project...". In the popup, expand "Maven" and select "Maven Project". Click "Next".
- Check "Create a simple project (skip archetype selection)". Click "Next".
- Enter the following:

Group Id	com.promineotech
Artifact Id	jeep-sales

Click "Finish".

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

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

- Add the dependencies from the Initializr:
  - Web
  - Devtools
  - Lombok
- Click "Explore" at the bottom of the page.
- Click "Copy" to copy the pom.xml generated by the Initializr to the clipboard.

- 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.
- 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.
- Create a package in src/main/java named com.promineotech.jeepp. In this package:
  - Create a Java class with a main method named JeepSales.
  - Add a class-level annotation: @SpringBootApplication and the import statement.
  - 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);
```

```
    }
```

```
}
```

- 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.**
  - Load the files that were added: right-click on the project in Package Explorer and select "Refresh".
  - 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".
- Using the MySQL Workbench or MySQL command line client (CLI), create a database named "jeepp".

- 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.
- Create a new package in src/test/java named com.promineotech.jeeptest.controller. Create a Spring Boot integration test named FetchJeepTest using the techniques shown in the video.
  - Add the @SpringBootTest, @ActiveProfiles, and @Sql annotations as described in the video.
  - The class must not be public. It should have package-level access (i.e., not public, private, or protected).
  - 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 {
}
```

- Create a test method in FetchJeepTest. The method must have the following method signature:
 

```
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
```
- 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;
```
- Create a new package in src/main/java named com.promineotech.jeeptest.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`.

- 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;
}
```

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

- 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`.

- 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;
```

- Produce a screenshot showing the completed test class. 
- In `src/main/java`, create a new package `com.promineotech.jeep.controller`. In this package, create an interface named `JeepSalesController`.
  - Add the class-level annotation `@RequestMapping("/jeeps")`.
  - 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`.
  - 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.
  - Add the parameter annotations in the OpenAPI documentation to describe the `model` and `trim` parameters.
  - Add the `@GetMapping` annotation and the `@ResponseStatus(code = HttpStatus.OK)` annotation as method-level annotations to the `fetchJeeps` method.
  - 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);
}
```

- Produce a screenshot showing the interface and OpenAPI documentation. 🖥️
- Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.
- 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. 🖥️

## Screenshots of Code:

### Jeepsales class

```
10 /**  
4  package com.promineotech.jeep;  
5  
60 import org.springframework.boot.SpringApplication;  
8  
90 /**  
10  * @author north  
11  *  
12  */  
13  @SpringBootApplication  
14  public class JeepSales {  
15  
160  /**  
17  * @param args  
18  */  
190  public static void main(String[] args) {  
20      SpringApplication.run(JeepSales.class, args);  
21  
22  }  
23  |  
24  }  
25
```

### FetchJeepTest class

```

18 /**
19  *
20  * @author north
21  *
22  */
23
24 @SpringBootTest(classes = JeepSales.class, 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 extends FetchJeepTestSupport {
31
32     @Test
33     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
34         JeepModel model = JeepModel.WRANGLER;
35         String trim = "Sport";
36         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
37         System.out.println(uri);
38
39         ResponseEntity<Jeep> response = getRestTemplate().getForEntity(uri, Jeep.class);
40
41         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
42     }
43 }
44 }
45

```

**BaseTest class**



```

10 /**
4  package com.promineotech.jeepp.controller;
5
6  import org.springframework.beans.factory.annotation.Autowired;
10
11 /**
12  * @author north
13  *
14  */
15 public class BaseTest {
16  @LocalServerPort
17  int serverPort;
18
19  @Autowired
20  @Getter
21  private TestRestTemplate restTemplate;
22
23  String getBaseUri() {
24      return String.format("http://localhost:%d/jeeps", serverPort);
25  }
26 }
27

```

## JeepModel Interface

```

10 /**
4  package com.promineotech.jeepp.entity;
5
6  /**
7  * @author north
8  *
9  */
10 public enum JeepModel {
11  WRANGLER, GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, GLADIATOR, WRANGLER_4XE
12 }
13

```

## Jeep Class

```

1  /**
2   *
3   */
4  package com.promineotech.jee.entity;
5
6  import java.math.BigDecimal;
7
8  /**
9   * @author north
10  *
11  */
12  @Data
13  @Builder
14  @NoArgsConstructor
15  @AllArgsConstructor
16  public class Jeep {
17      private Long modelPK;
18      private JeepModel modelId;
19      private String trimLevel;
20      private int numDoors;
21      private int wheelSize;
22      private BigDecimal basePrice;
23  }
24
25

```

## JeepSalesController class

```

1  /**
2   *
3   */
4  package com.promineotech.jee.controller;
5
6  import java.util.List;
7
8  /**
9   * @author north
10  *
11  */
12  @RequestMapping("/jeeps")
13  @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
14      @Server(url = "http://localhost:8080", description = "Yo Local Server")})
15  public interface JeepSalesController {
16
17      @Operation(
18          summary = "Summary of Jeep",
19          description = "Yo description of jeep",
20          responses = {
21              @ApiResponse(responseCode = "200", description = "Good", content = @Content(mediaType = "application/json", schema = @Schema(implementation = Jeep.class))),
22              @ApiResponse(responseCode = "400", description = "Invalid", content = @Content(mediaType = "application/json")),
23              @ApiResponse(responseCode = "404", description = "Not Found", content = @Content(mediaType = "application/json")),
24              @ApiResponse(responseCode = "500", description = "error", content = @Content(mediaType = "application/json"))
25          },
26          parameters = {
27              @Parameter(name = "model", allowEmptyValue = false, required = false, description = "Model name"),
28              @Parameter(name = "trim", allowEmptyValue = false, required = false, description = "Trim")
29          }
30      )
31      @GetMapping
32      @ResponseStatus(code = HttpStatus.OK)
33      List<Jeep> fetchJeeps(@RequestParam(required = false) String model, @RequestParam(required = false) String trim);
34  }
35

```

## DefaultJeepSalesController class

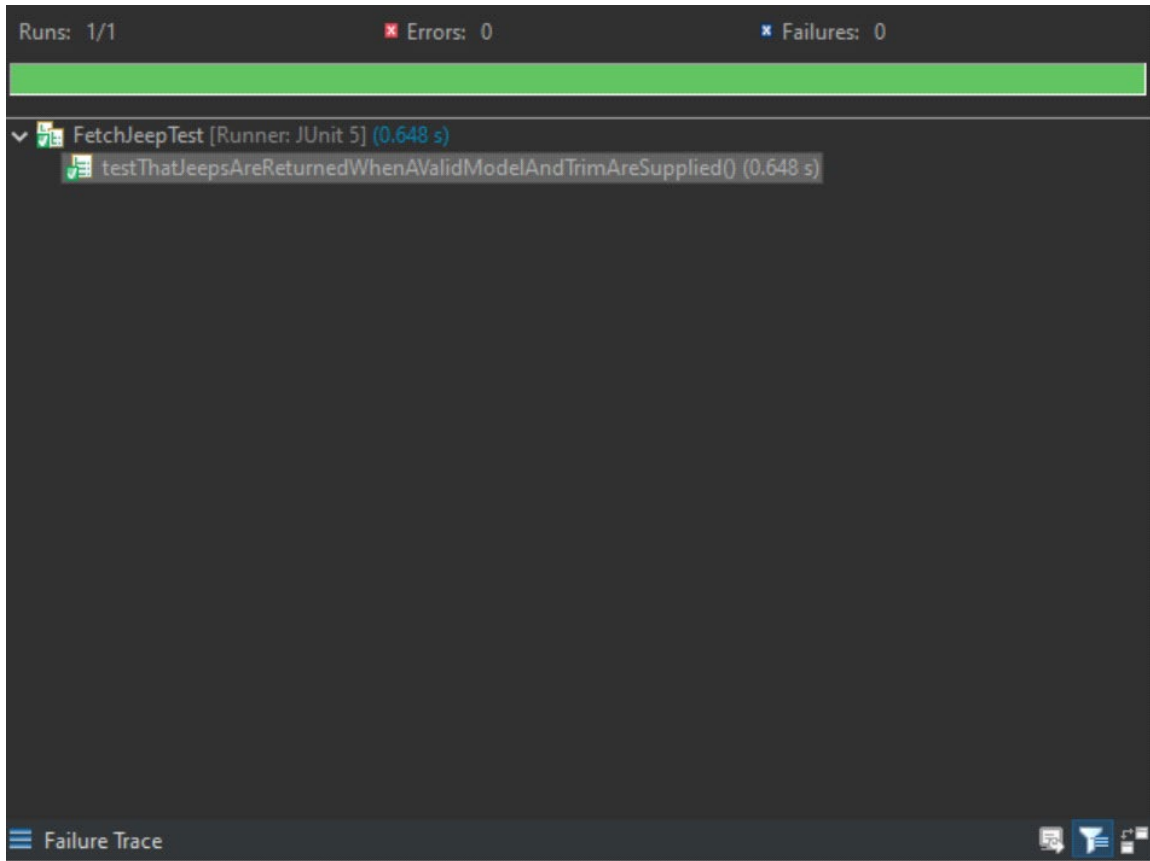
```

1  /**
2   *
3   */
4  package com.promineotech.jee.controller;
5
6  import java.util.List;
7
8  /**
9   * @author north
10  *
11  */
12  @RestController
13  public class DefaultJeepSalesController implements JeepSalesController {
14
15      @Override
16      public List<Jeep> fetchJeeps(String model, String trim) {
17          return null;
18      }
19  }
20

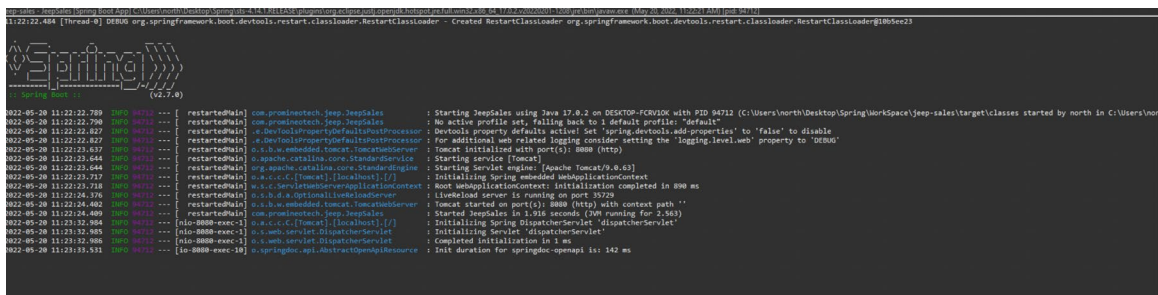
```

## Screenshots of Running Application:

## JUnit pass



## SpringBootApplication running



## OpenApi

The top screenshot displays the Swagger UI for the "Jeep Sales Service" at the endpoint `GET /jeeps`. It includes a "Try it out" button and a table of query parameters:

Name	Description
model	Model name
trim	Trim

The bottom screenshot shows the response details for the `200` status code. It includes a "Try it out" button and a table of response details:

Code	Description	Links
200	Good	No links
400	Invalid	No links
404	Not Found	No links
500	error	No links

The JSON example for the `200` response is:

```
{  "model": "Jeep",  "modelId": "BRANDLER",  "trimLevel": "string",  "trim": "Jeep",  "trimId": "Jeep",  "trimPrice": "Jeep",  "trimPrice": "Jeep"}
```

The schema definition for the `Jeep` object is:

```
Jeep {  modelId: Integer(5int64)  trimLevel: string  trim: Enum  trimPrice: Array [ 7 ]  trimPrice: string}
```

URL to GitHub Repository: <https://github.com/Starssk1ttles/Week1SpringBoot>