


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:

Group Id	com.promineotech
Artifact Id	jeep-sales

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	com.promineotech
Artifact	jeep-sales
Name	jeep-sales
Description	Jeep Sales
Package name	com.promineotech
Packaging	Jar
Java	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.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.**
 - 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
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
JeepModel	model	JeepModel.WRANGLER
String	trim	"Sport"
String	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.  x

- 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.  x

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.  x

Screenshots of Code:

test class

```

1 package com.promineotech.jeepp.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
8 import org.springframework.boot.test.web.client.TestRestTemplate;
9 import org.springframework.boot.web.server.LocalServerPort;
10 import org.springframework.http.HttpStatus;
11 import org.springframework.http.ResponseEntity;
12 import org.springframework.test.context.ActiveProfiles;
13 import org.springframework.test.context.jdbc.Sql;
14 import org.springframework.test.context.jdbc.SqlConfig;
15 import com.promineotech.jeepp.controller.support.FetchJeepTestSupport;
16 import com.promineotech.jeepp.entity.Jeep;
17 import com.promineotech.jeepp.entity.JeepModel;
18 import lombok.Getter;
19
20 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
21 @ActiveProfiles("test")
22 @Sql(scripts = {
23     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
24     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
25     config = @SqlConfig(encoding = "utf-8"))
26
27 class FetchJeepTest extends FetchJeepTestSupport{
28     // 9. e. restTemplate & serverPort are in BaseTest. followed videos.
29     // 13. instead of uri in uri string, gets uri from BaseTest. followed videos.
30
31     @Test
32     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
33         //given: a valid model, trim, and URI
34         JeepModel model = JeepModel.WRANGLER;
35         String trim = "Sport";
36         String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);
37
38         //When: a connection is made to the URI
39         // 13. a. followed videos for this step
40         ResponseEntity<Jeep> response = getRestTemplate().getForEntity(uri, Jeep.class);
41
42         //Then: a success (OK - 200) status code is returned
43         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
44     }
45 }
46 }
47

```

base class for reference (followed videos, he used a base class for something instead of putting them in the test class)

```
1 package com.promineotech.jeepp.controller.support;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.boot.test.web.client.TestRestTemplate;
5 import org.springframework.boot.web.server.LocalServerPort;
6 import lombok.Getter;
7
8 public class BaseTest {
9     @LocalServerPort
10     private int serverPort;
11
12     @Autowired
13     @Getter
14     private TestRestTemplate restTemplate;
15
16     protected String getBaseUrl() {
17         return String.format("http://localhost:%d/jeeps", serverPort);
18     }
19
20 }
21
```

OpenAPI Doc and interface


```

1 package com.promineotech.jeepp.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 @RequestMapping("/jeeps")
21 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"),
22     servers = {@Server(url = "http://localhost:8080", description = "Local Server.")})
23
24 public interface JeepSalesController {
25     // API documentation
26     // @formatter:off
27     @Operation(
28         summary = "Returns a list of Jeep",
29         description = "Returns a list of Jeeps given an optional model and/or trim",
30         responses = {
31             @ApiResponse(
32                 responseCode = "200",
33                 description = "A list of jeeps is returned",
34                 content = @Content(
35                     mediaType = "application/json",
36                     schema = @Schema(implementation = Jeep.class))),
37             @ApiResponse(
38                 responseCode = "400",
39                 description = "The request parameters are invalid",
40                 content = @Content(
41                     mediaType = "application/json")),
42             @ApiResponse(
43                 responseCode = "404",
44                 description = "No Jeeps were found with the input",
45                 content = @Content(
46                     mediaType = "application/json")),
47             @ApiResponse(
48                 responseCode = "500",
49                 description = "An unplanned error occurred",
50                 content = @Content(
51                     mediaType = "application/json"))
52         },
53         parameters = {
54             @Parameter(
55                 name = "model",
56                 allowEmptyValue = false,
57                 required = false,
58                 description = "the model name (i.e., 'Wrangler')"),
59             @Parameter(
60                 name = "trim",
61                 allowEmptyValue = false,
62                 required = false,
63                 description = "the trim name (i.e., 'Sport')")
64         }
65     )

```

```

65     )
66     @GetMapping
67     @ResponseStatus(code = HttpStatus.OK)
68     List<Jeep> fetchJeeps(
69         @RequestParam(required = false)
70         JeepModel model,
71         @RequestParam(required = false)
72         String trim
73     );
74     // @formatter:on
75
76 }

```

Screenshots of Running Application:

Swagger

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": "WRANGLER", "trimLevel": "string", "numDoors": 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</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

URL to GitHub Repository:

<https://github.com/ANGRAYSON/Week-13-Jeep-Sales>