


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

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	<code>com.promineotech</code>
Artifact	<code>jeep-sales</code>
Name	<code>jeep-sales</code>
Description	Jeep Sales
Package name	<code>com.promineotech</code>
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.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.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
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	<code>model</code>	<code>JeepModel.WRANGLER</code>
String	<code>trim</code>	<code>"Sport"</code>
String	<code>uri</code>	<code>String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);</code>

- 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. 

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(JeepModel model, String trim);  
}
```


```

List<Jeep> fetchJeeps(@RequestParam JeepModel model,
    @RequestParam String trim);
}

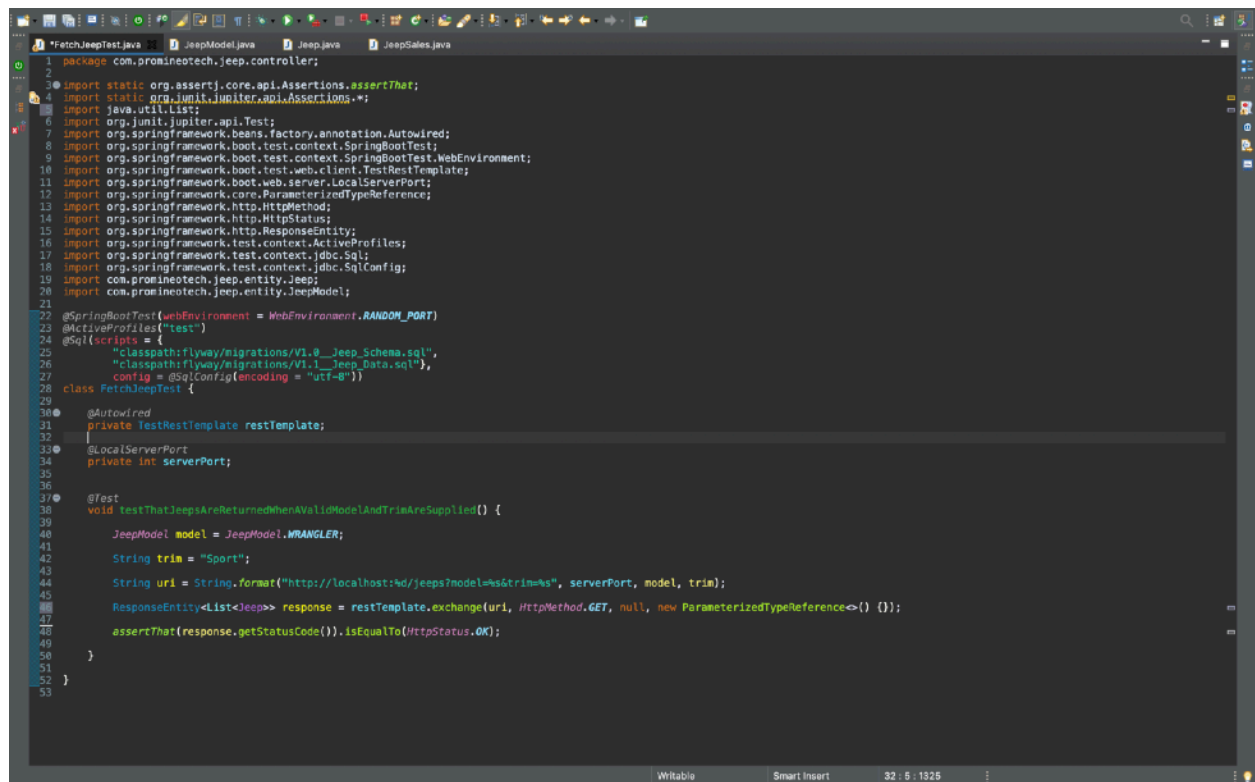
```

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. 

Screenshots of Code:



```

1 package com.promineotech.jeeptest;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import static org.junit.jupiter.api.Assertions.*;
5 import java.util.List;
6 import org.junit.jupiter.api.Test;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.boot.test.web.client.TestRestTemplate;
10 import org.springframework.boot.web.server.LocalServerPort;
11 import org.springframework.core.ParameterizedTypeReference;
12 import org.springframework.http.HttpMethod;
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.jeeptest.entity.Jeeptest;
19 import com.promineotech.jeeptest.entity.JeeptestModel;
20
21
22 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
23 @ActiveProfiles("test")
24 @Sql(scripts = {
25     "classpath:flyway/migrations/V1.0__Jeeptest.Schema.sql",
26     "classpath:flyway/migrations/V1.1__Jeeptest.Data.sql"},
27     config = @SqlConfig(encoding = "utf-8"))
28 class FetchJeepTest {
29
30     @Autowired
31     private TestRestTemplate restTemplate;
32
33     @LocalServerPort
34     private int serverPort;
35
36
37     @Test
38     void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
39
40         JeeptestModel model = JeeptestModel.WRANGLER;
41
42         String trim = "Sport";
43
44         String url = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
45
46         ResponseEntity<List<Jeep>> response = restTemplate.exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
47         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
48     }
49
50 }
51
52
53

```

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

Screenshots of Running Application:

The screenshot displays the Swagger UI for the Jeep Sales Service. The browser address bar shows the URL: `www.localhost:8080/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config#/default-jeep-sales-controller/fetchJeeps`. The Swagger logo is visible in the top left corner, and the text "v3/api-docs" is in the top right corner. The main heading is "Jeep Sales Service" with the subtext "v3/api-docs". Below this, the "Servers" section shows "http://localhost:8080 - Local server." with a dropdown arrow. The "default-jeep-sales-controller" section is expanded, showing the "GET /jeeps" endpoint. The description states: "Returns a list of Jeeps given an optional model and/or trim". The "Parameters" section lists two query parameters: "model" (string, required = false, description: "The model name (i.e., 'WRANGLER')") and "trim" (string, required = false, description: "The trim level (i.e., 'Sport')"). The "model" parameter has a dropdown menu showing "Available values: WRANGLER, GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, GLADIATOR, WRANGLER_4XE". A "Try it out" button is located to the right of the parameters section.

Swagger UI

www.localhost:8080/swagger-ui/index.html?configUrl=js/api-docs/swagger-config#/default-jeep-sales-controller/fetchJeeps

trim
string
(query)

The trim level (i.e., 'Sport')

trim

Responses

Code	Description	Links
200	A list of Jeeps is returned.	No links
Media type application/json		
Controls Accept header.		
Example Value Schema		
<pre>{ "modelPK": 0, "modelId": "WRANGLER", "trimLevel": "string", "numDoors": 0, "wheelSize": 0, "basePrice": 0}</pre>		
400	The request parameters are invalid.	No links
Media type application/json		
404	No Jeeps were found with the input criteria.	No links

Swagger UI

www.localhost:8080/swagger-ui/index.html?configUrl=js/api-docs/swagger-config#/default-jeep-sales-controller/fetchJeeps

Media type
application/json

400

The request parameters are invalid.

No links

Media type
application/json

404

No Jeeps were found with the input criteria.

No links

Media type
application/json

500

An unplanned error occurred.

No links

Media type
application/json

Schemas

```
Jeep {  modelPK      integer($int64)  modelId      string  Enum:  > Array [ 7 ]  trimLevel    string  numDoors     integer($int32)  wheelSize    integer($int32)  basePrice    number}
```

```
2021-09-18 17:04:42.210 INFO 8165 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Starting JeepSales using Java 16.0.2 on Johnathans-MacBook-Pro.local with PID 8165 (/Users/jc
2021-09-18 17:04:42.212 INFO 8165 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : No active profile set, falling back to default profiles: default
2021-09-18 17:04:42.246 INFO 8165 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2021-09-18 17:04:42.247 INFO 8165 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2021-09-18 17:04:42.989 INFO 8165 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-09-18 17:04:42.916 INFO 8165 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-09-18 17:04:42.916 INFO 8165 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-09-18 17:04:42.950 INFO 8165 --- [ restartedMain] o.s.c.C.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-09-18 17:04:42.950 INFO 8165 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 713 ms
2021-09-18 17:04:43.491 INFO 8165 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2021-09-18 17:04:43.427 INFO 8165 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-18 17:04:43.436 INFO 8165 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Started JeepSales in 1.433 seconds (JVM running for 7.02)
2021-09-18 17:06:09.527 INFO 8165 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-18 17:06:09.527 INFO 8165 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-09-18 17:06:09.528 INFO 8165 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2021-09-18 17:06:19.613 INFO 8165 --- [nio-8080-exec-6] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 137 ms
```

```
Finished after 8.07 seconds
Runs: 1/1 Errors: 0 Failures: 0
FetchJeepTest (Runner: JUnit 5) (0.459 s)
```

URL to GitHub Repository:

<https://github.com/johnbarts/springbootweek1>

