

Web API Design with Spring Boot Week 13 Coding Assignment

Points possible: 75


URL to GitHub Repository: <https://github.com/oalinares/Week13AndOnSpringBoot>

URL to Public Link of your Video:


https://www.youtube.com/watch?v=iPC-ZI-TFfc&ab_channel=OscarLinares

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:

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

a)

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 (or whatever your version is)

Web API Design with Spring Boot Week 13 Coding Assignment

- a) Add the dependencies from the Initializr:
 - i) Web
 - ii) Devtools
 - iii) Lombok
- b) Click "Explore" at the bottom of the page.
- c) 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.**

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
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>

1)

- 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. 

2) 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)

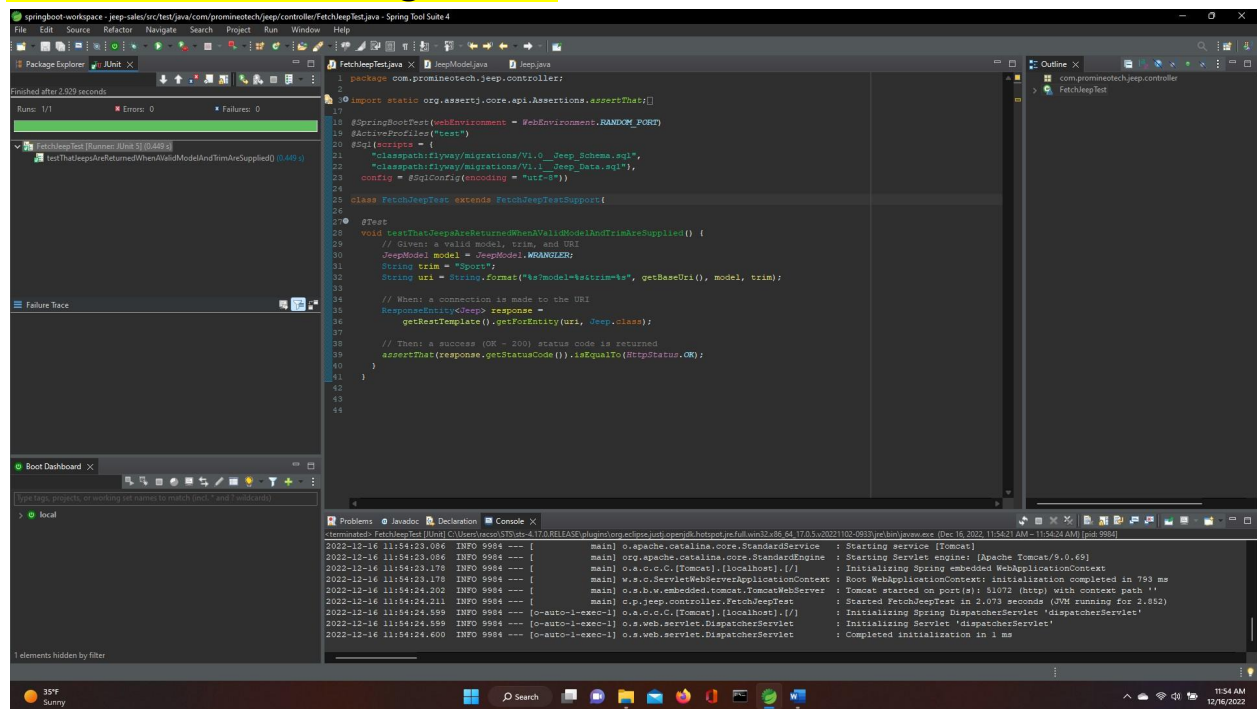
    List<Jeep> fetchJeeps(@RequestParam JeepModel model,
        @RequestParam String trim);

}
```

- g) Produce a screenshot showing the interface and OpenAPI documentation. 
- 3) Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.
 - 4) 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

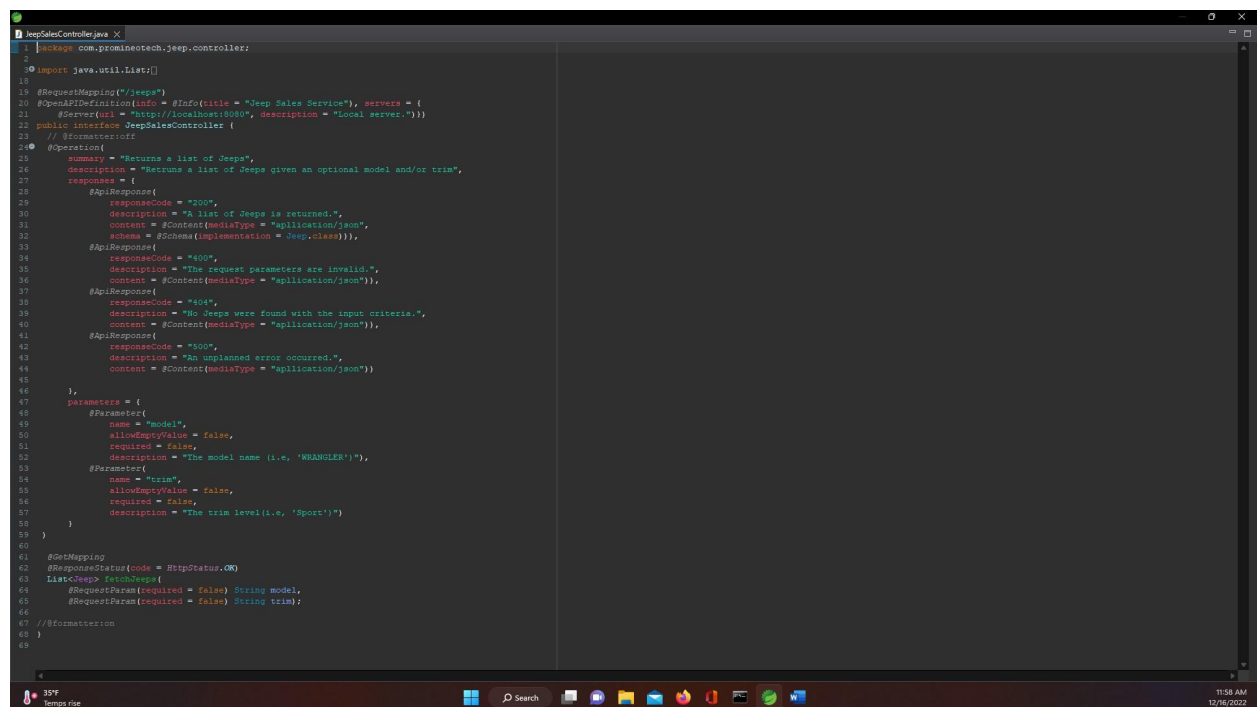
Screenshots for Assignment:



This screenshot shows an IDE with a Spring Boot test and its console output. The test, `FetchJeepTest`, is located in `com.promineotech.jeep.controller` and extends `FetchJeepTestSupport`. It includes a `@Test` method `testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()` that uses `MockMvc` to send a GET request to `/api/jeeps` with a valid model and trim. The console output shows the application starting successfully on port 51072.

```
1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.http.MediaType;
9 import org.springframework.test.web.servlet.MockMvc;
10 import org.springframework.test.web.servlet.MvcResult;
11
12 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
13 @AutoConfigureMockMvc
14 class FetchJeepTest extends FetchJeepTestSupport {
15
16     @Test
17     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
18         // Given: a valid model, trim, and URI
19         JeepModel model = JeepModel.WRANGLER;
20         String trim = "Sport";
21         String uri = String.format("/api/model=%s/trim=%s", model, trim);
22
23         // When: a connection is made to the URI
24         MockMvc mockMvc = mockMvcBuilder.build();
25         MvcResult response = mockMvc.perform(get(uri)).andReturn();
26
27         // Then: a success (200) status code is returned
28         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
29     }
30 }
```

```
2022-12-16 11:54:23.046 INFO 9984 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Starting service [Tomcat]
2022-12-16 11:54:23.046 INFO 9984 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Starting Servlet engine: [Apache Tomcat/9.0.49]
2022-12-16 11:54:23.176 INFO 9984 --- [main] o.s.b.w.e.tomcat.TomcatWebServer : Initializing Spring embedded WebApplicationContext
2022-12-16 11:54:23.176 INFO 9984 --- [main] w.s.o.s.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 793 ms
2022-12-16 11:54:24.202 INFO 9984 --- [main] o.s.b.w.e.tomcat.TomcatWebServer : Tomcat started on port(s): 51072 (http) with context path ''
2022-12-16 11:54:24.211 INFO 9984 --- [main] o.s.j.e.c.FetchJeepTest : Started FetchJeepTest in 2.073 seconds (JVM running for 2.512)
2022-12-16 11:54:24.599 INFO 9984 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-12-16 11:54:24.599 INFO 9984 --- [o-auto-1-exec-1] o.s.w.s.v.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-12-16 11:54:24.600 INFO 9984 --- [o-auto-1-exec-1] o.s.w.s.v.DispatcherServlet : Completed initialization in 1 ms
```



This screenshot shows the `JeepSalesController.java` file in the IDE. It defines a `JeepSalesController` class with a `getJeeps` method. The class includes Swagger annotations for API documentation, such as `@ApiOperation`, `@ApiResponse`, and `@ApiResponses`. The `getJeeps` method is annotated with `@GetMapping` and `@RequestMapping`. It uses `MockMvc` to send a GET request to `/api/jeeps` and returns the response as a `Jeep` object.

```
1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.web.servlet.mvc.method.annotation.SwaggerAnnotationMethodProcessor;
7
8 @ApiOperation(value = "Returns a list of Jeeps", notes = "Returns a list of Jeeps given an optional model and/or trim")
9 @ApiResponse(responseCode = "200", description = "A list of Jeeps is returned.", content = @Content(mediaType = "application/json", schema = @Schema(implementation = Jeep.class)))
10 @ApiResponses({
11     @ApiResponse(responseCode = "400", description = "The request parameters are invalid.", content = @Content(mediaType = "application/json")),
12     @ApiResponse(responseCode = "404", description = "No Jeeps were found with the input criteria.", content = @Content(mediaType = "application/json")),
13     @ApiResponse(responseCode = "500", description = "An unplanned error occurred.", content = @Content(mediaType = "application/json"))
14 })
15 @RequestMapping("/api")
16 public class JeepSalesController {
17
18     // @formatter:off
19     @GetMapping("/jeeps")
20     @RequestMapping(method = RequestMethod.GET)
21     public ResponseEntity<List<Jeep>> getJeeps(
22         @RequestParam(required = false) String model,
23         @RequestParam(required = false) String trim
24     ) {
25         // @formatter:on
26
27         // TODO: Implement the logic to fetch Jeeps based on model and trim
28         List<Jeep> jeeps = new ArrayList<>();
29
30         return ResponseEntity.ok(jeeps);
31     }
32 }
```

Web API Design with Spring Boot Week 13 Coding Assignment

The screenshot shows the Swagger UI for the 'Jeep Sales Service' at the endpoint `/v3/api-docs`. The selected server is `http://localhost:8080 - Local server.`. The endpoint `GET /jeeps` is highlighted, with a description: 'Returns a list of Jeeps'. Below the description, the parameters section shows two query parameters: `model` (string) and `trim` (string). The responses section is currently collapsed.

Jeep Sales Service
/v3/api-docs

Servers
http://localhost:8080 - Local server.

basic-jeep-sales-controller

GET /jeeps Returns a list of Jeeps

Retruns a list of Jeeps given an optional model and/or trim

Parameters

Name	Description
model string (query)	The model name (i.e, 'WRANGLER')
trim string (query)	The trim level(i.e, 'Sport')

Responses

This screenshot shows the 'Responses' section of the Swagger UI for the `GET /jeeps` endpoint. It lists five response codes with their descriptions and media types (all set to `application/json`):

- 200**: A list of Jeeps is returned.
- 400**: The request parameters are invalid.
- 404**: No Jeeps were found with the input criteria.
- 500**: An unplanned error occurred.

Below the responses, the 'Schemas' section shows the `Jeep` schema:

```
Jeep {  
  modelId: integer(1000)  
  modelName: string  
  trim: string  
  trimLevel: string  
  wheelSize: integer(1000)  
  basePrice: number  
}
```