

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.

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
- ✓ 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: Web, Devtools, & 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.
- ✓ 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.
 - ✓ 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".
- ✓ 7) Using the MySQL Workbench or MySQL command line client, 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 (not public/private/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
<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&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. 🖥️

```

romineotech/jeep/controller/FetchJeepTest.java - Spring Tool Suite 4
Project Run Window Help
JeepSalesController.java BasicJeepSalesControl... JeepSales.java *FetchJeepTest.java x JeepModel.java Jeep.java
21 import io.swagger.v3.oas.models.PathItem.HttpMethod;
22 import lombok.Getter;
23
24
25
26 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
27 @ActiveProfiles("test")
28 @Sql(scripts = {
29     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
30     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
31     config = @SqlConfig(encoding = "utf-8"))
32
33 class FetchJeepTest {
34
35     @Autowired
36     @Getter
37     private TestRestTemplate restTemplate;
38
39     @LocalServerPort
40     private int serverPort;
41
42     @Test
43     void testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied() {
44
45         //Given: A valid model, trim and URI
46         JeepModel model = JeepModel.WRANGLER;
47         String trim = "Sport";
48         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
49
50         //When: a connection is made to the URI
51         ResponseEntity<List<Jeep>>
52             response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
53
54         //Then: a success (OK - 200) status code is returned
55         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
56
57     } //end TEST
58 } //end 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:

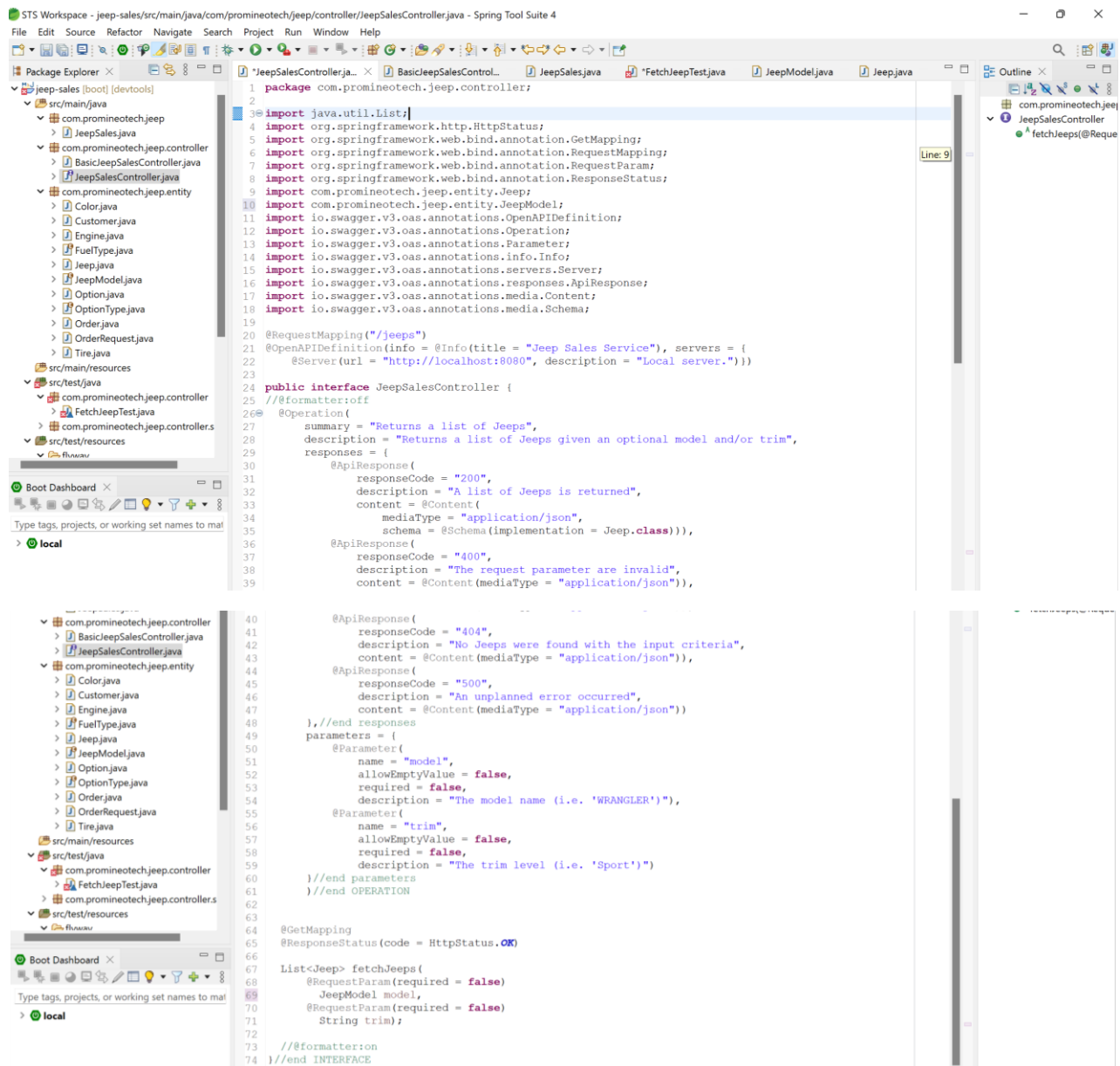
`List<Jeep> fetchJeeps(JeepModel model, String trim);`

- ✓ 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 parameter annotations in OpenAPI documentation to describe the `model` & `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. 🖥️



```
1 package com.promineotech.jeepp.controller;
2
3 import java.util.List;
4 import org.springframework.http.HttpStatus;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.ResponseStatus;
9 import com.promineotech.jeepp.entity.Jeep;
10 import com.promineotech.jeepp.entity.JeepModel;
11 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
12 import io.swagger.v3.oas.annotations.Operation;
13 import io.swagger.v3.oas.annotations.Parameter;
14 import io.swagger.v3.oas.annotations.info.Info;
15 import io.swagger.v3.oas.annotations.servers.Server;
16 import io.swagger.v3.oas.annotations.responses.ApiResponse;
17 import io.swagger.v3.oas.annotations.media.Content;
18 import io.swagger.v3.oas.annotations.media.Schema;
19
20 @RequestMapping("/jeeps")
21 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
22     @Server(url = "http://localhost:8080", description = "Local server.")})
23
24 public interface JeepSalesController {
25     //formatter:off
26     @Operation(
27         summary = "Returns a list of Jeeps",
28         description = "Returns a list of Jeeps given an optional model and/or trim",
29         responses = {
30             @ApiResponse(
31                 responseCode = "200",
32                 description = "A list of Jeeps is returned",
33                 content = @Content(
34                     mediaType = "application/json",
35                     schema = @Schema(implementation = Jeep.class))),
36             @ApiResponse(
37                 responseCode = "400",
38                 description = "The request parameter are invalid",
39                 content = @Content(mediaType = "application/json")),
40             @ApiResponse(
41                 responseCode = "404",
42                 description = "No Jeeps were found with the input criteria",
43                 content = @Content(mediaType = "application/json")),
44             @ApiResponse(
45                 responseCode = "500",
46                 description = "An unplanned error occurred",
47                 content = @Content(mediaType = "application/json"))
48         }, //end responses
49         parameters = {
50             @Parameter(
51                 name = "model",
52                 allowEmptyValue = false,
53                 required = false,
54                 description = "The model name (i.e. 'WRANGLER')"),
55             @Parameter(
56                 name = "trim",
57                 allowEmptyValue = false,
58                 required = false,
59                 description = "The trim level (i.e. 'Sport')")
60         }) //end parameters
61     //end OPERATION
62
63     @GetMapping
64     @ResponseStatus(code = HttpStatus.OK)
65
66     List<Jeep> fetchJeeps(
67         @RequestParam(required = false)
68         JeepModel model,
69         @RequestParam(required = false)
70         String trim);
71
72     //formatter:on
73 //end INTERFACE
```

- ✓ 15) Add the controller implementation class named `DefaultJeepSalesController`, and `@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 4 possible outcomes. 🖥️

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

basic-jeep-sales-controller

GET /jeeps Returns a list of Jeeps

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

Parameters [Try it out](#)

Name	Description
model string (query)	The model name (i.e. 'WRANGLER') Available values : WRANGLER, GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, GLADIATOR, WRANGLER_4XE <input type="text" value="WRANGLER"/>
trim string (query)	The trim level (i.e. 'Sport') <input type="text" value="Sport"/>

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/jeeps?model=WRANGLER&trim=Sport' \
  -H 'accept: application/json'
```

Request URL
http://localhost:8080/jeeps?model=WRANGLER&trim=Sport

Server response

Code	Details
200	Response headers connection: keep-alive content-length: 0 date: Sat, 24 Sep 2022 22:35:10 GMT keep-alive: timeout=60

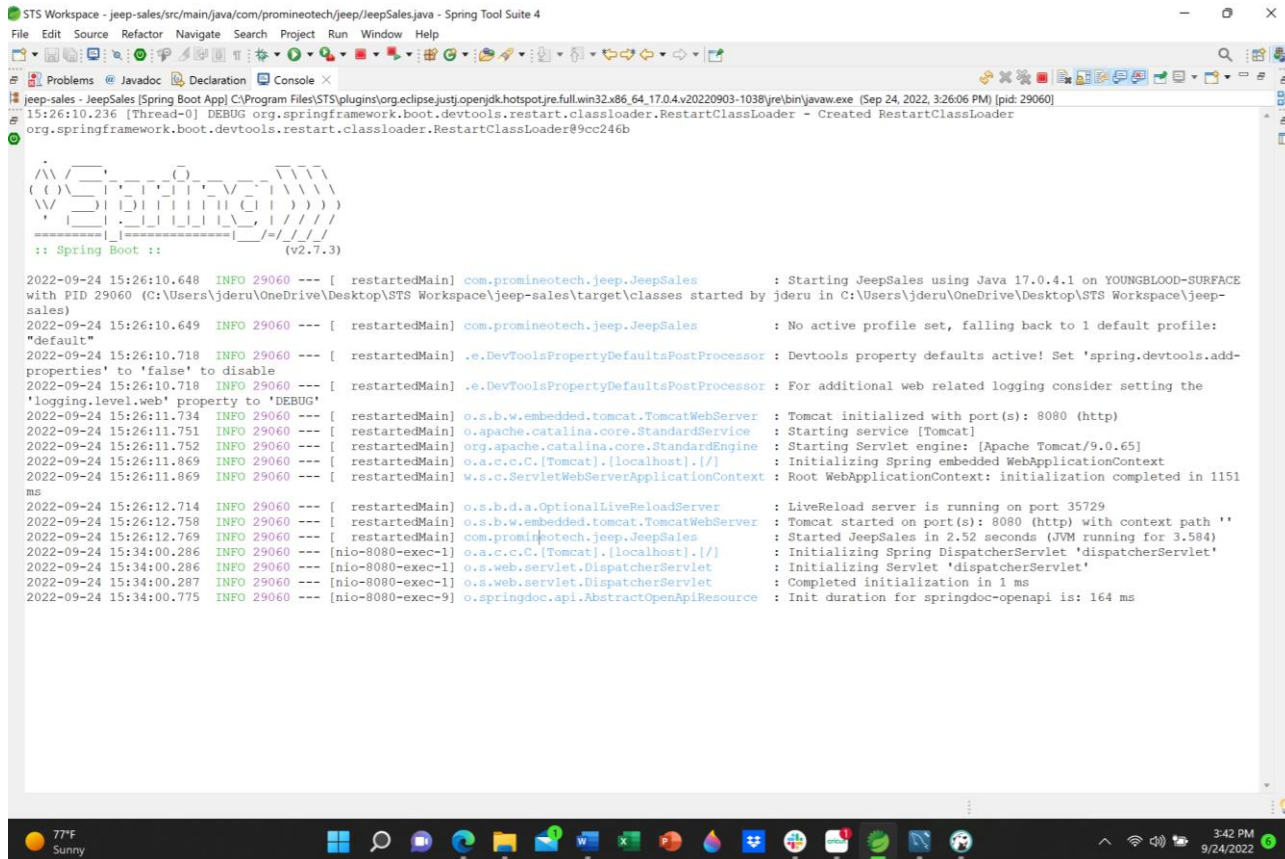
Responses

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

Schemas

Jeep

```
{
  modelPK: integer($int64)
  modelId: string
  trimLevel: string
  numDoors: integer($int32)
  wheelSize: integer($int32)
  basePrice: number
}
```

```
STS Workspace - jeep-sales/src/main/java/com/promineotech/jeep/JeepSales.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
jeep-sales - JeepSales [Spring Boot App] C:\Program Files\STS\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220903-1038\jre\bin\javaw.exe (Sep 24, 2022, 3:26:06 PM) [pid: 29060]
15:26:10.236 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader
org.springframework.boot.devtools.restart.classloader.RestartClassLoader@9cc246b

:: Spring Boot :: (v2.7.3)

2022-09-24 15:26:10.648 INFO 29060 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Starting JeepSales using Java 17.0.4.1 on YOUNGBLOOD-SURFACE
with PID 29060 (C:\Users\jderu\OneDrive\Desktop\STS Workspace\jeep-sales\target\classes started by jderu in C:\Users\jderu\OneDrive\Desktop\STS Workspace\jeep-
sales)
2022-09-24 15:26:10.649 INFO 29060 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : No active profile set, falling back to 1 default profile:
"default"
2022-09-24 15:26:10.718 INFO 29060 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-
properties' to 'false' to disable
2022-09-24 15:26:10.718 INFO 29060 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the
'logging.level.web' property to 'DEBUG'
2022-09-24 15:26:11.734 INFO 29060 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-09-24 15:26:11.751 INFO 29060 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-24 15:26:11.752 INFO 29060 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-24 15:26:11.869 INFO 29060 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-09-24 15:26:11.869 INFO 29060 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1151
ms
2022-09-24 15:26:12.714 INFO 29060 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-09-24 15:26:12.758 INFO 29060 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-09-24 15:26:12.769 INFO 29060 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Started JeepSales in 2.52 seconds (JVM running for 3.584)
2022-09-24 15:34:00.286 INFO 29060 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-09-24 15:34:00.286 INFO 29060 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-09-24 15:34:00.287 INFO 29060 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-09-24 15:34:00.775 INFO 29060 --- [nio-8080-exec-9] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 164 ms
```

URL to GitHub Repository:

<https://github.com/JaxYoungblood/Week13-SpringBootCodingAssignment.git>