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 setings:

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.jeep`. 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.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);

      }

    }
    ```
- 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 "jeep".

- 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.jeep.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.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`.

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

  - 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. **DONE**

- 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):

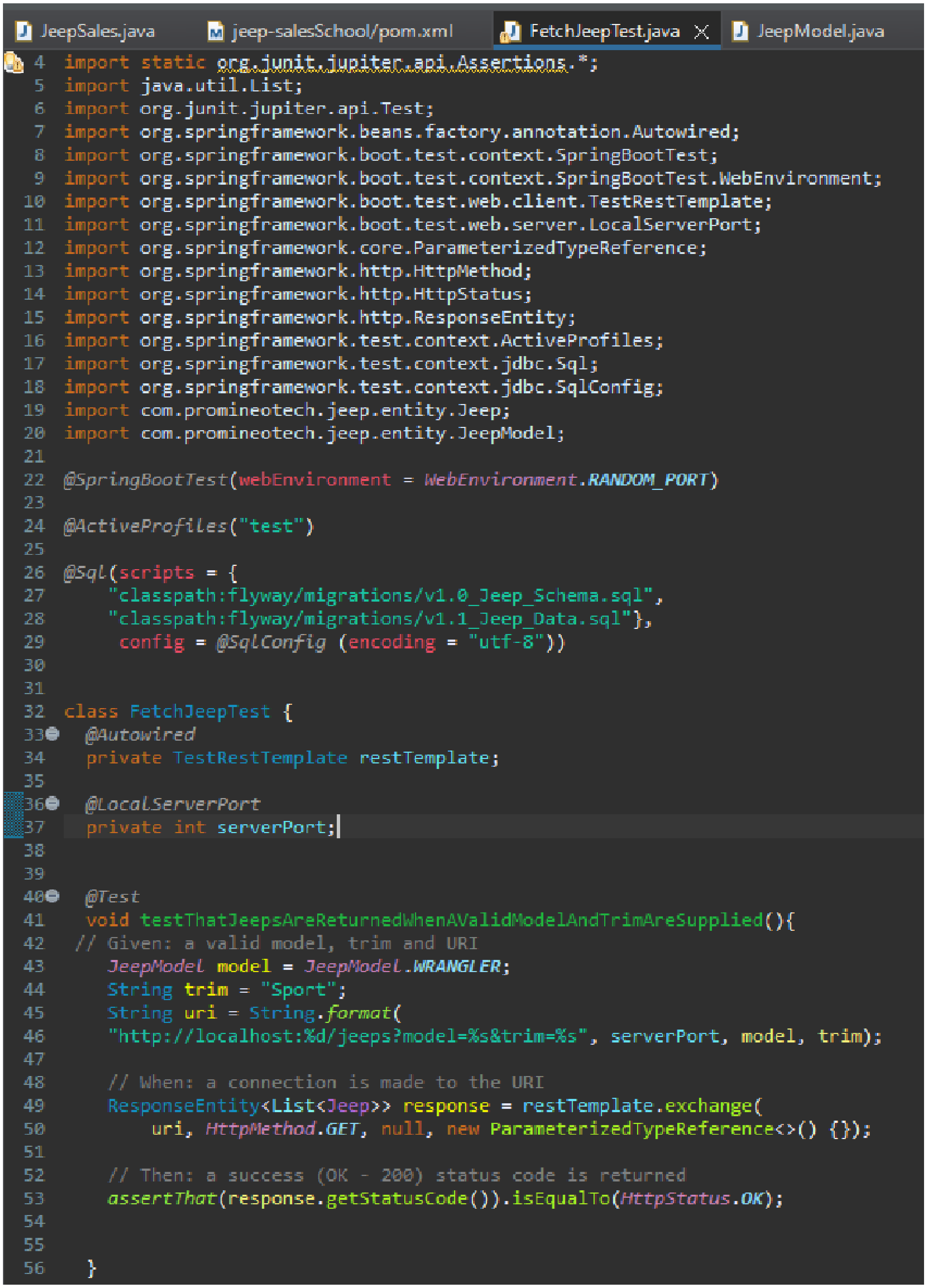
    ```
    @RequestMapping("/jeeps")

    public interface JeepSalesController {

      @GetMapping

      @ResponseStatus(code = HttpStatus.OK)

      List<Jeep> fetchJeeps(@RequestParam JeepModel model,

          @RequestParam String trim);
    ```

```
}
```

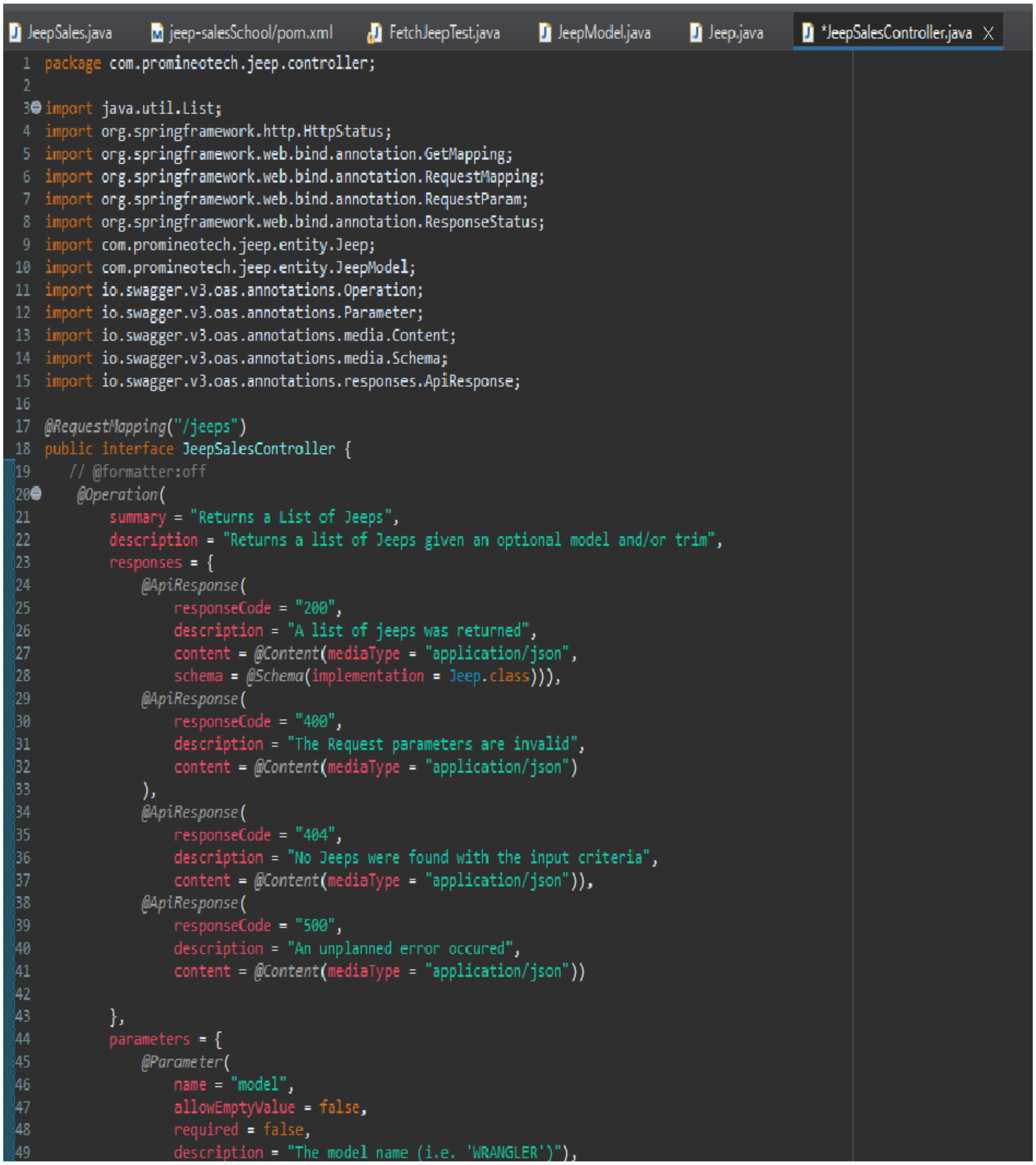
- Produce a screenshot showing the interface and OpenAPI documentation. 🖥 **DONE**

- 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. 🖥 **DONE**

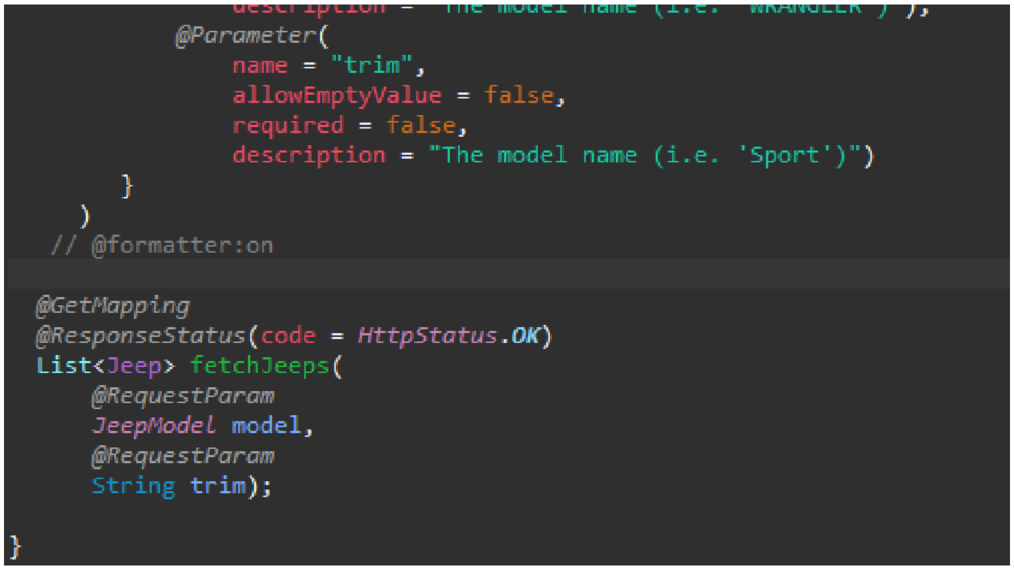
Screenshots of Code:



```java
import static org.junit.jupiter.api.Assertions.*;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.test.web.server.LocalServerPort;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.jdbc.SqlConfig;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;

@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 {
  @Autowired
  private TestRestTemplate restTemplate;

  @LocalServerPort
  private int serverPort;


  @Test
  void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied(){
 // Given: a valid model, trim and URI
    JeepModel model = JeepModel.WRANGLER;
    String trim = "Sport";
    String uri = String.format(
    "http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);

    // When: a connection is made to the URI
    ResponseEntity<List<Jeep>> response = restTemplate.exchange(
        uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});

    // Then: a success (OK - 200) status code is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);


  }
```

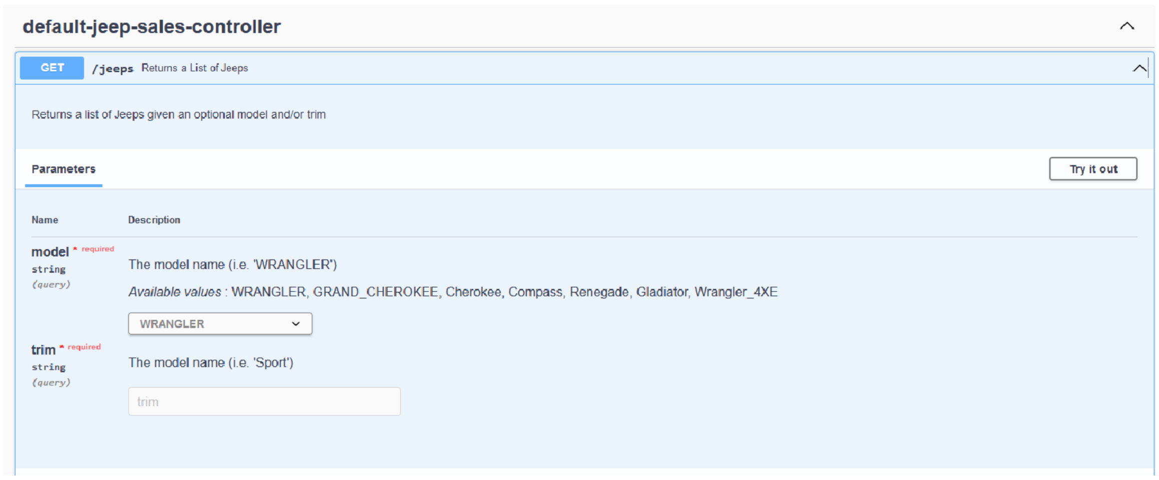


```java
package com.promineotech.jeep.controller;

import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;

@RequestMapping("/jeeps")
public interface JeepSalesController {
   // @formatter:off
    @Operation(
        summary = "Returns a List of Jeeps",
        description = "Returns a list of Jeeps given an optional model and/or trim",
        responses = {
            @ApiResponse(
                responseCode = "200",
                description = "A list of jeeps was returned",
                content = @Content(mediaType = "application/json",
                schema = @Schema(implementation = Jeep.class))),
            @ApiResponse(
                responseCode = "400",
                description = "The Request parameters are invalid",
                content = @Content(mediaType = "application/json")
            ),
            @ApiResponse(
                responseCode = "404",
                description = "No Jeeps were found with the input criteria",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "500",
                description = "An unplanned error occured",
                content = @Content(mediaType = "application/json"))

        },
        parameters = {
            @Parameter(
                name = "model",
                allowEmptyValue = false,
                required = false,
                description = "The model name (i.e. 'WRANGLER')"),
```



```
            @Parameter(
                name = "trim",
                allowEmptyValue = false,
                required = false,
                description = "The model name (i.e. 'Sport')")
        }
    )
  // @formatter:on

  @GetMapping
  @ResponseStatus(code = HttpStatus.OK)
  List<Jeep> fetchJeeps(
      @RequestParam
      JeepModel model,
      @RequestParam
      String trim);

}
```

Screenshots of Running Application:



default-jeep-sales-controller

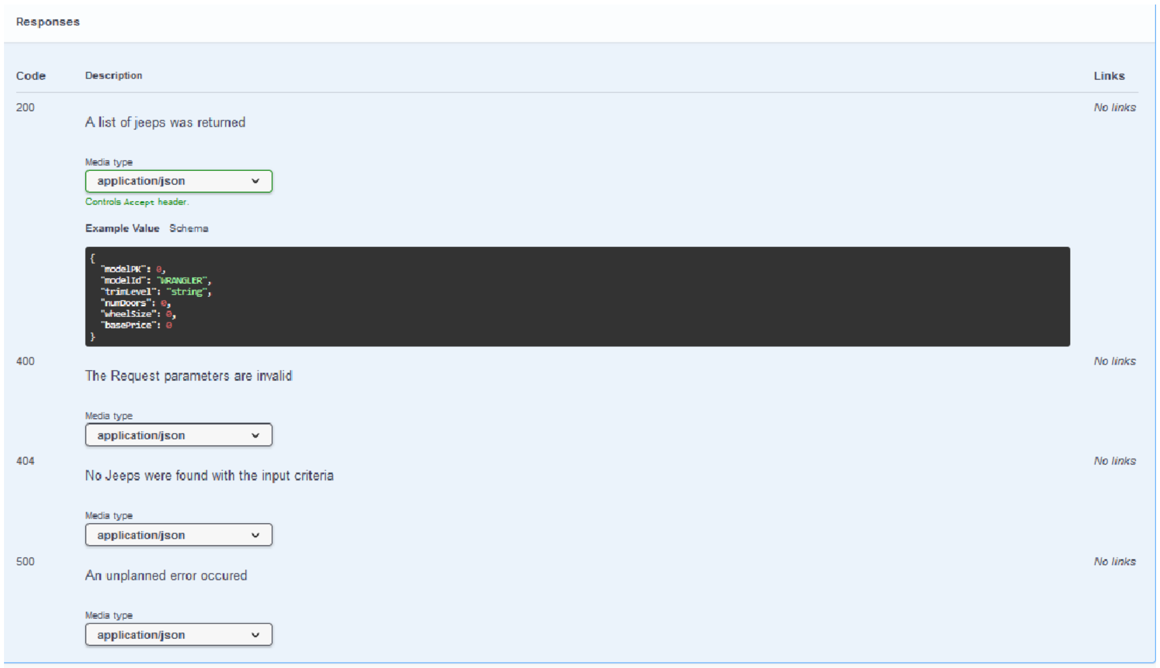
GET /jeeps Returns a List of Jeeps

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

Parameters

Try it out

Name	Description
model * required string (query)	The model name (i.e. 'WRANGLER') *Available values* : WRANGLER, GRAND_CHEROKEE, Cherokee, Compass, Renegade, Gladiator, Wrangler_4XE WRANGLER
trim * required string (query)	The model name (i.e. 'Sport') trim



Responses

Code | Description | Links

200

A list of jeeps was returned

Media type: application/json

Controls Accept header.

Example Value | Schema

```
{
  "modelPK": 0,
  "modelId": "WRANGLER",
  "trimLevel": "string",
  "numDoors": 0,
  "wheelSize": 0,
  "basePrice": 0
}
```

No links

400

The Request parameters are invalid

Media type: application/json

No links

404

No Jeeps were found with the input criteria

Media type: application/json

No links

500

An unplanned error occured

Media type: application/json

No links

URL to GitHub Repository: