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 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

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.

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):

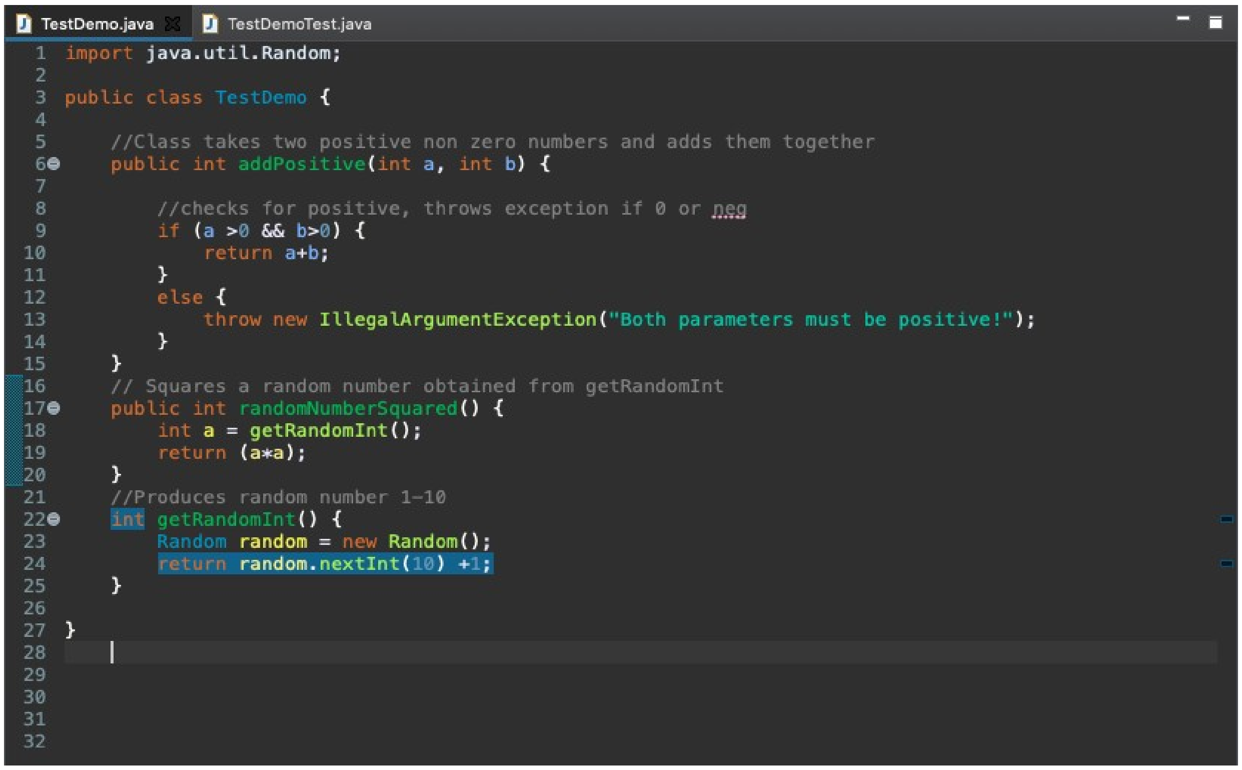
```
@RequestMapping("/jeeps")
public interface JeepSalesController {
  @GetMapping
  @ResponseStatus(code = HttpStatus.OK)
  List<Jeep> fetchJeeps(@RequestParam JeepModel model,
      @RequestParam String trim);
}
```

g) Produce a screenshot showing the interface and OpenAPI documentation.

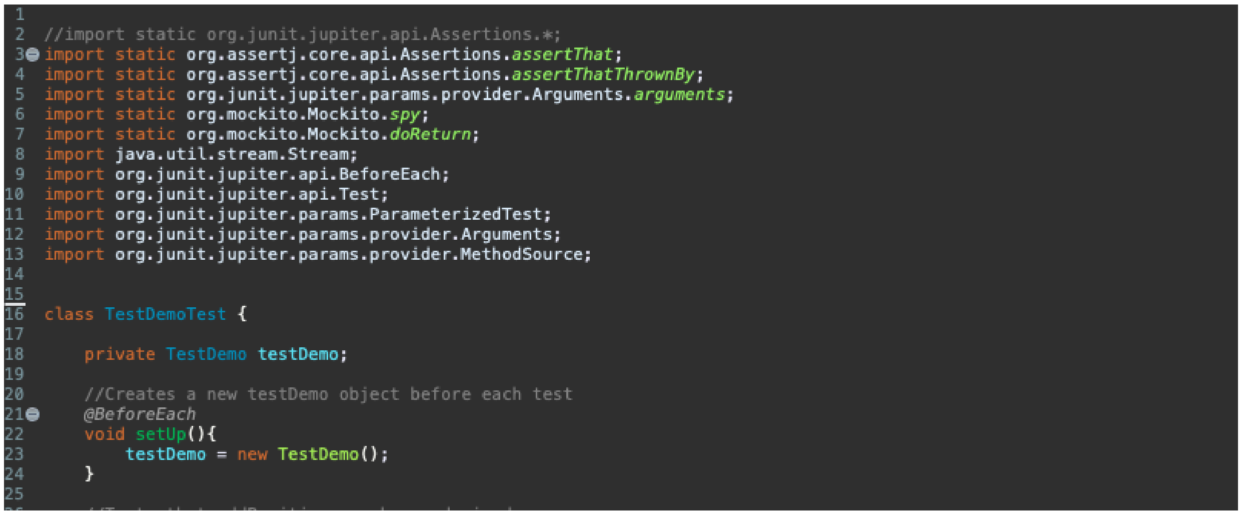
15) Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.

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.

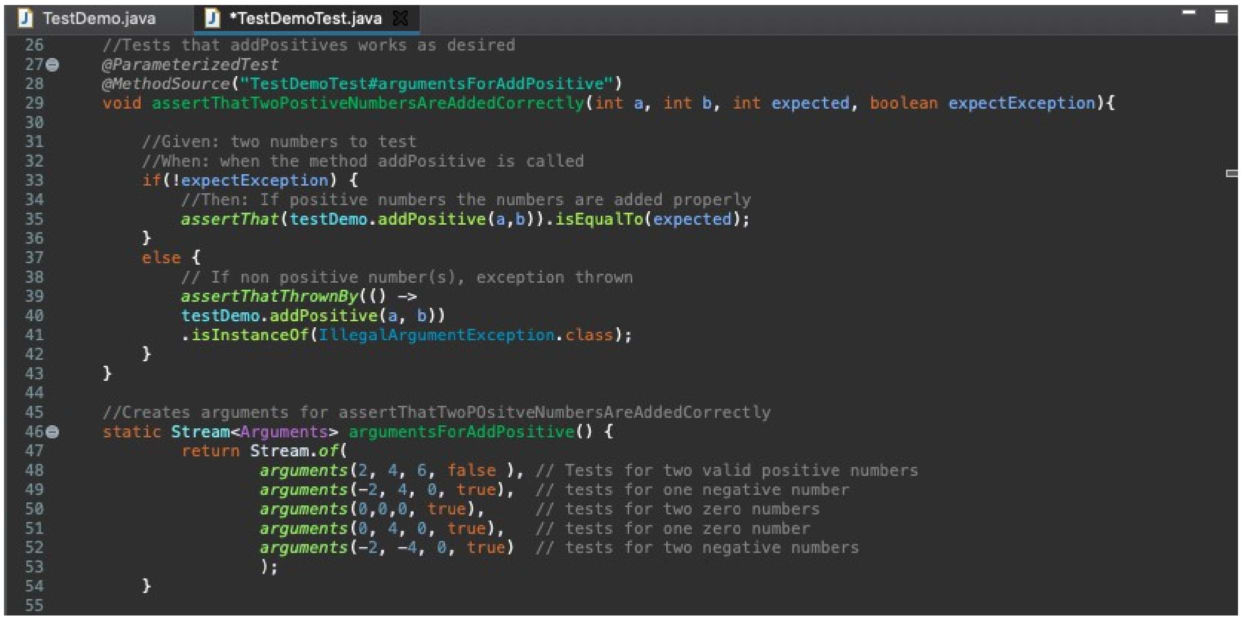
Screenshots of Code:



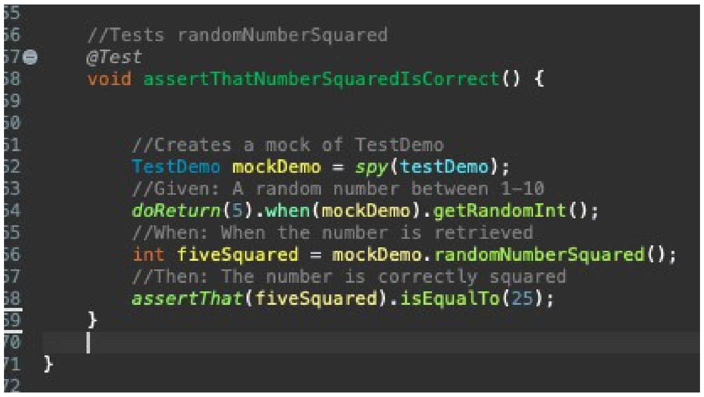
```java
import java.util.Random;

public class TestDemo {

    //Class takes two positive non zero numbers and adds them together
    public int addPositive(int a, int b) {

        //checks for positive, throws exception if 0 or neg
        if (a >0 && b>0) {
            return a+b;
        }
        else {
            throw new IllegalArgumentException("Both parameters must be positive!");
        }
    }
    // Squares a random number obtained from getRandomInt
    public int randomNumberSquared() {
        int a = getRandomInt();
        return (a*a);
    }
    //Produces random number 1-10
    int getRandomInt() {
        Random random = new Random();
        return random.nextInt(10) +1;
    }

}
```



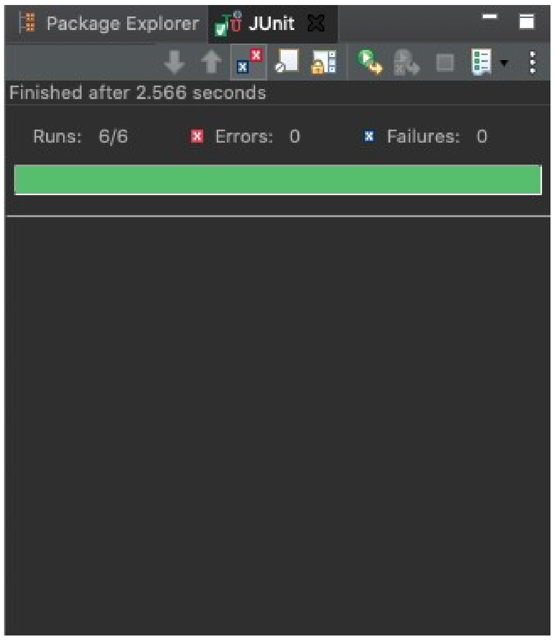
```java

//import static org.junit.jupiter.api.Assertions.*;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
import static org.junit.jupiter.params.provider.Arguments.arguments;
import static org.mockito.Mockito.spy;
import static org.mockito.Mockito.doReturn;
import java.util.stream.Stream;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;


class TestDemoTest {

    private TestDemo testDemo;

    //Creates a new testDemo object before each test
    @BeforeEach
    void setUp(){
        testDemo = new TestDemo();
    }

```



```java
    //Tests that addPositives works as desired
    @ParameterizedTest
    @MethodSource("TestDemoTest#argumentsForAddPositive")
    void assertThatTwoPostiveNumbersAreAddedCorrectly(int a, int b, int expected, boolean expectException){

        //Given: two numbers to test
        //When: when the method addPositive is called
        if(!expectException) {
            //Then: If positive numbers the numbers are added properly
            assertThat(testDemo.addPositive(a,b)).isEqualTo(expected);
        }
        else {
            // If non positive number(s), exception thrown
            assertThatThrownBy(() ->
            testDemo.addPositive(a, b))
            .isInstanceOf(IllegalArgumentException.class);
        }
    }

    //Creates arguments for assertThatTwoPOsitveNumbersAreAddedCorrectly
    static Stream<Arguments> argumentsForAddPositive() {
            return Stream.of(
                    arguments(2, 4, 6, false ), // Tests for two valid positive numbers
                    arguments(-2, 4, 0, true),  // tests for one negative number
                    arguments(0,0,0, true),     // tests for two zero numbers
                    arguments(0, 4, 0, true),   // tests for one zero number
                    arguments(-2, -4, 0, true)  // tests for two negative numbers
                    );
        }

```

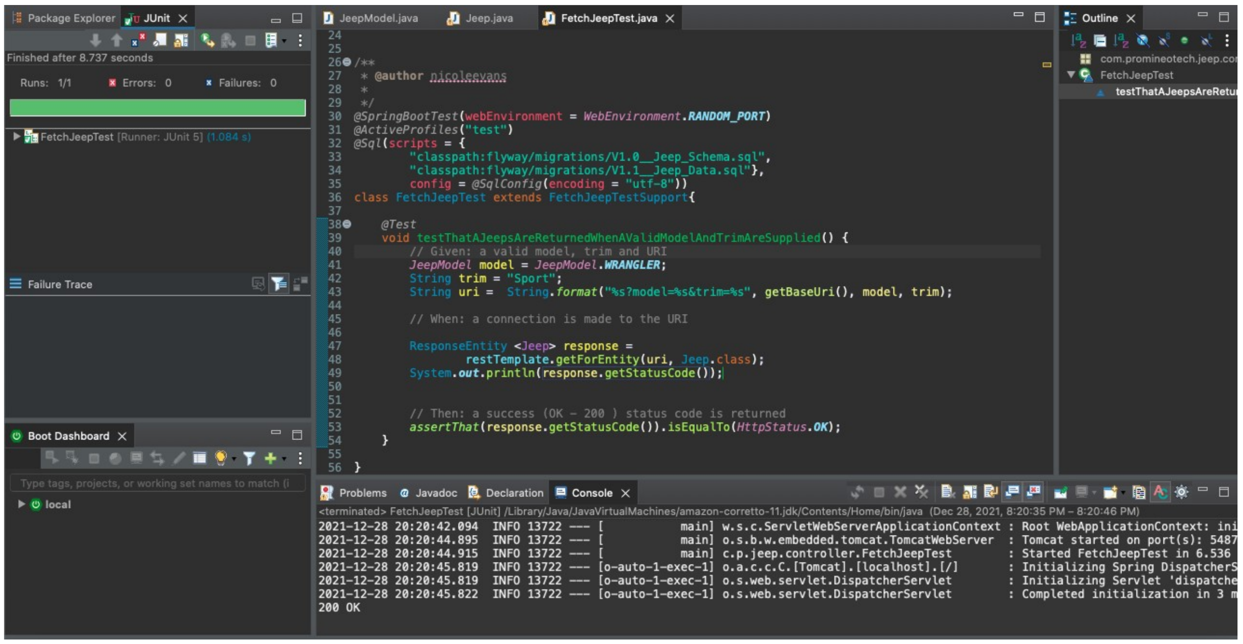


```java

    //Tests randomNumberSquared
    @Test
    void assertThatNumberSquaredIsCorrect() {


        //Creates a mock of TestDemo
        TestDemo mockDemo = spy(testDemo);
        //Given: A random number between 1-10
        doReturn(5).when(mockDemo).getRandomInt();
        //When: When the number is retrieved
        int fiveSquared = mockDemo.randomNumberSquared();
        //Then: The number is correctly squared
        assertThat(fiveSquared).isEqualTo(25);
    }

}

```



Finished after 2.566 seconds

Runs: 6/6   Errors: 0   Failures: 0



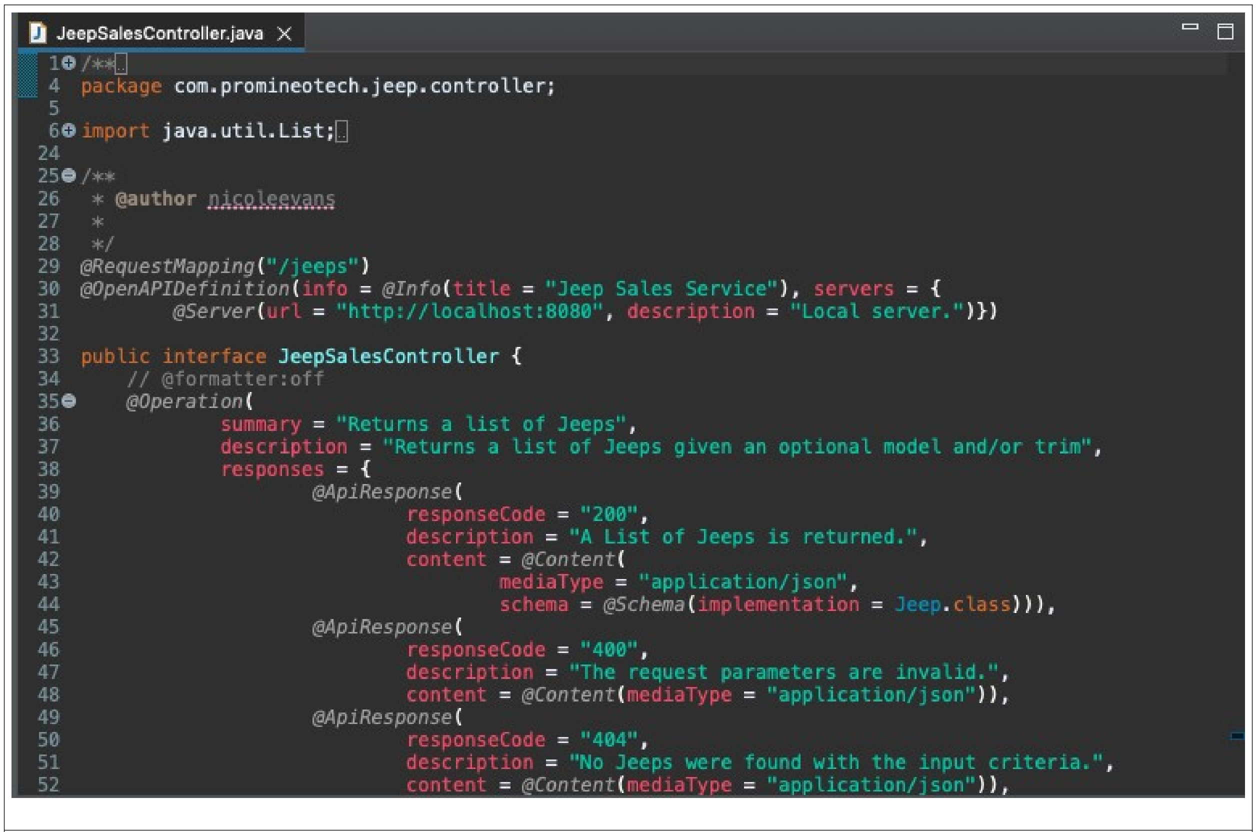
Finished after 8.737 seconds

Runs: 1/1   Errors: 0   Failures: 0

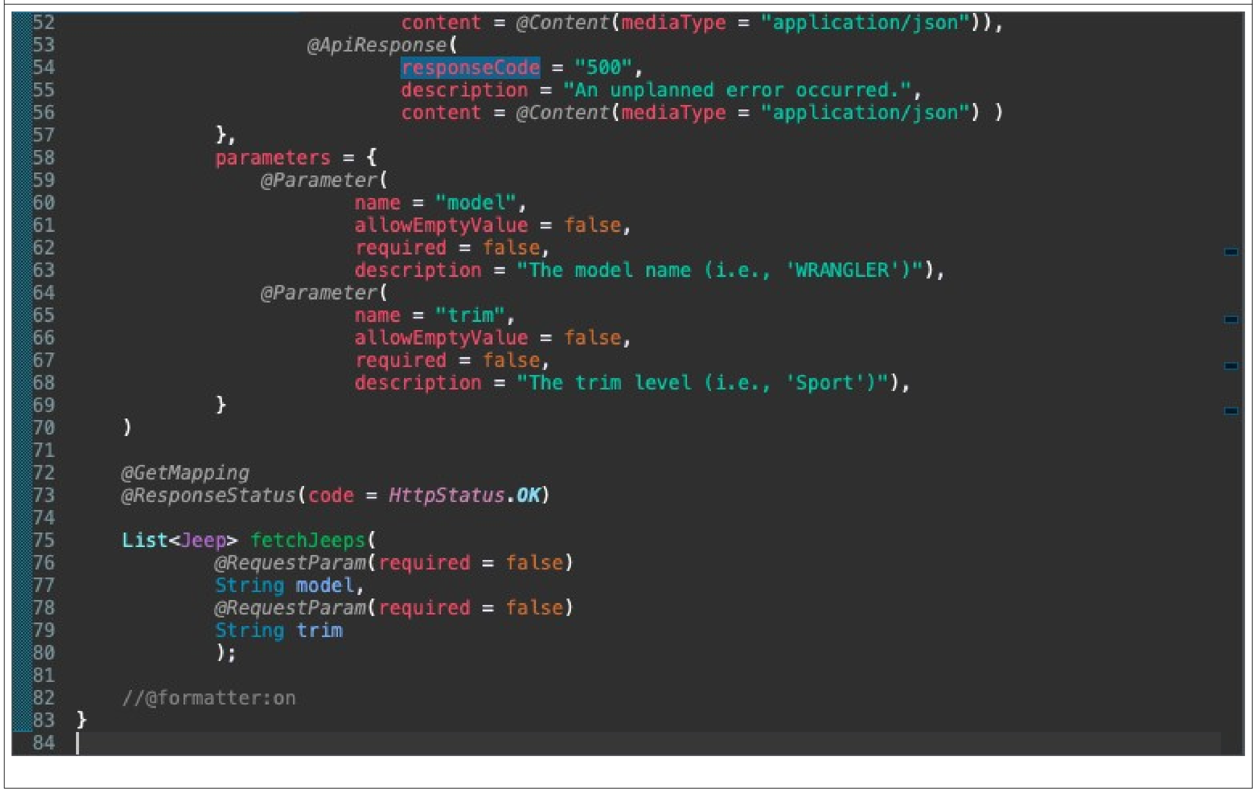
FetchJeepTest [Runner: JUnit 5] (1.084 s)

```java
/**
 * @author nicoleevans
 *
 */
@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 extends FetchJeepTestSupport{

    @Test
    void testThatAJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
        // Given: a valid model, trim and URI
        JeepModel model = JeepModel.WRANGLER;
        String trim = "Sport";
        String uri =  String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);

        // When: a connection is made to the URI

        ResponseEntity <Jeep> response =
                restTemplate.getForEntity(uri, Jeep.class);
        System.out.println(response.getStatusCode());


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

}
```

```
<terminated> FetchJeepTest [JUnit] /Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java  (Dec 28, 2021, 8:20:35 PM – 8:20:46 PM)
2021-12-28 20:20:42.094  INFO 13722 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: ini
2021-12-28 20:20:44.895  INFO 13722 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 5487
2021-12-28 20:20:44.915  INFO 13722 --- [           main] c.p.jeep.controller.FetchJeepTest        : Started FetchJeepTest in 6.536
2021-12-28 20:20:45.819  INFO 13722 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherS
2021-12-28 20:20:45.819  INFO 13722 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatche
2021-12-28 20:20:45.822  INFO 13722 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 m
200 OK
```

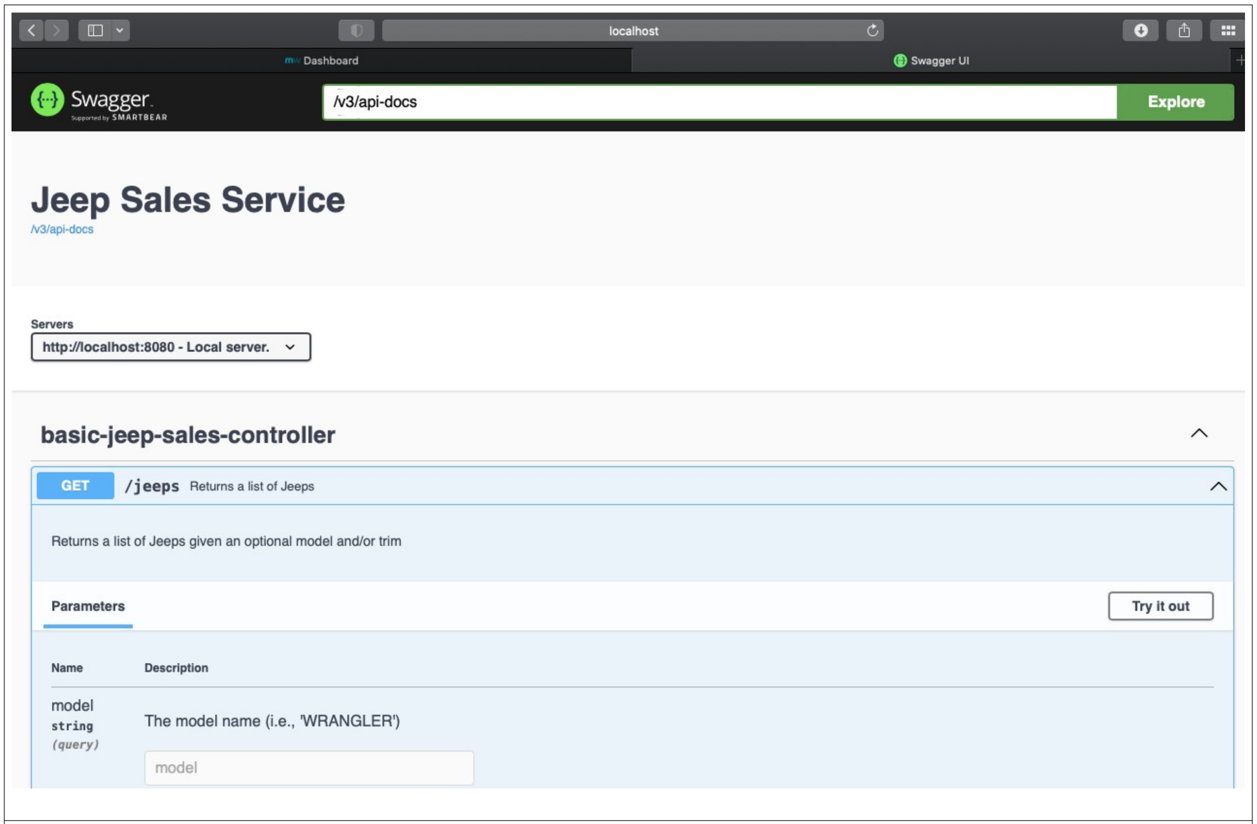


```java
/**
package com.promineotech.jeep.controller;

import java.util.List;

/**
 * @author nicoleevans
 *
 */
@RequestMapping("/jeeps")
@OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
        @Server(url = "http://localhost:8080", description = "Local server.")})

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 is 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")),
```



content = @Content(mediaType = "application/json")),

```java
                    @ApiResponse(
                            responseCode = "500",
                            description = "An unplanned error occurred.",
                            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 trim level (i.e., 'Sport')"),
            }
    )

    @GetMapping
    @ResponseStatus(code = HttpStatus.OK)

    List<Jeep> fetchJeeps(
            @RequestParam(required = false)
            String model,
            @RequestParam(required = false)
            String trim
            );

    //@formatter:on
}
```



Jeep Sales Service

/v3/api-docs

Servers

http://localhost:8080 - Local server.

basic-jeep-sales-controller

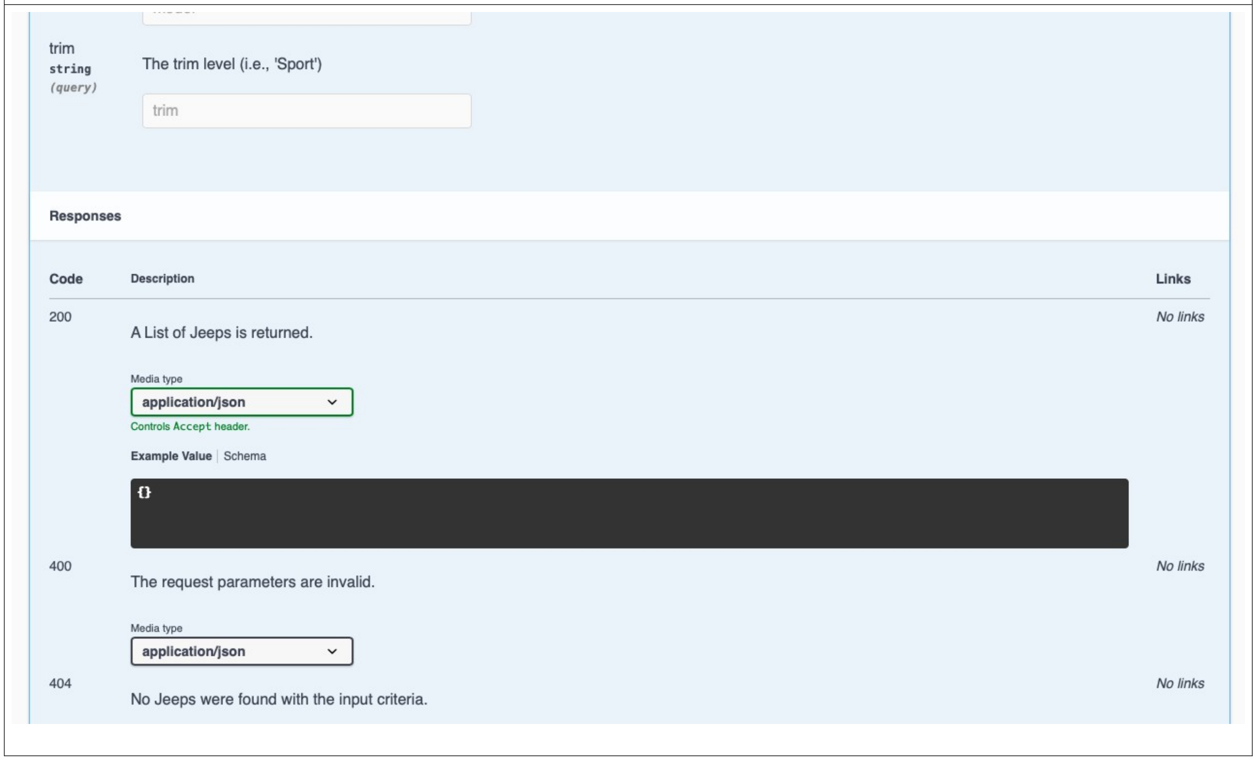
GET /jeeps Returns a list of Jeeps

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

Parameters

Try it out

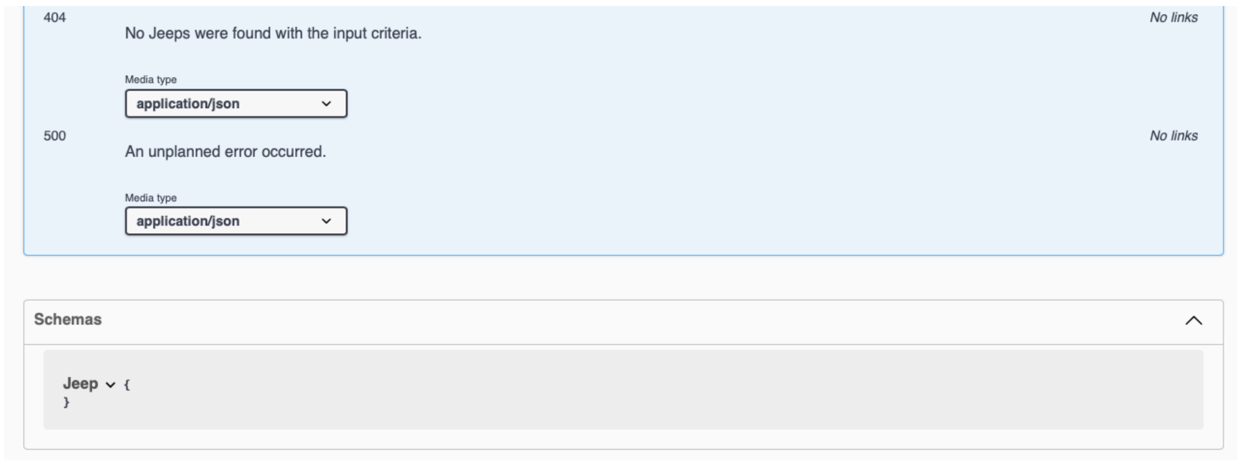
Name	Description
model string *(query)*	The model name (i.e., 'WRANGLER') model



Name	Description
trim string *(query)*	The trim level (i.e., 'Sport') trim

Responses

Code	Description	Links
200	A List of Jeeps is returned. Media type application/json Controls `Accept` header. Example Value | Schema `{}`	*No links*
400	The request parameters are invalid. Media type application/json	*No links*
404	No Jeeps were found with the input criteria.	*No links*


404

No Jeeps were found with the input criteria.

Media type

application/json

No links

500

An unplanned error occurred.

Media type

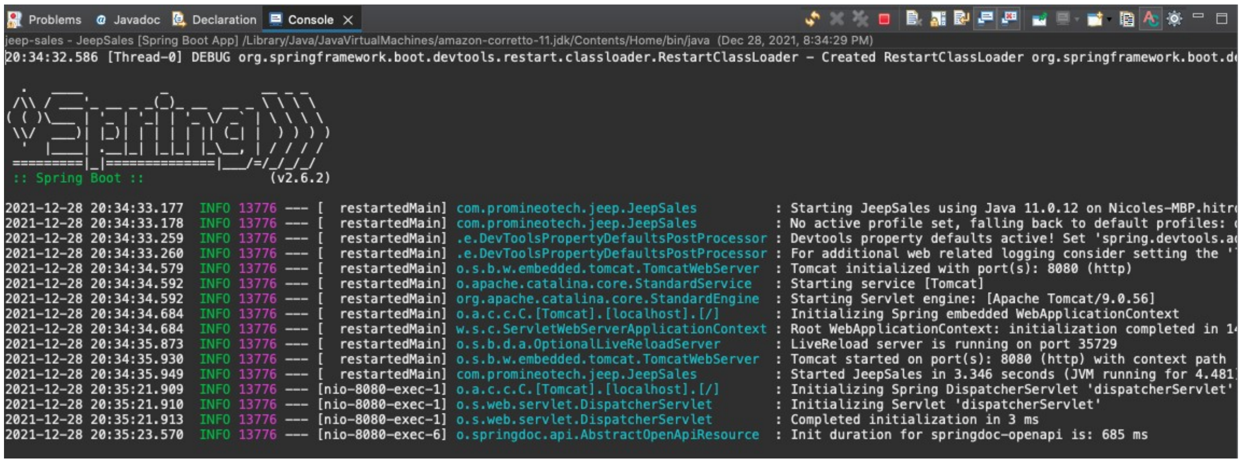
application/json

No links

Schemas

```
Jeep {
}
```

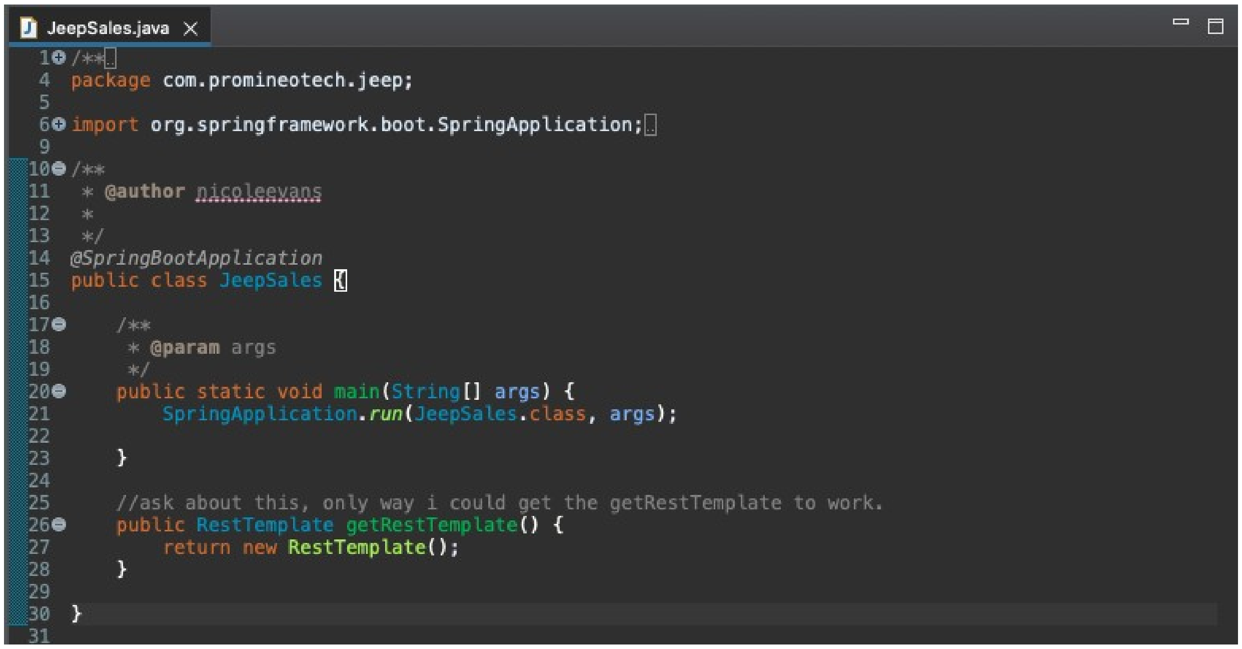
?



jeep-sales - JeepSales [Spring Boot App] /Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java (Dec 28, 2021, 8:34:29 PM)

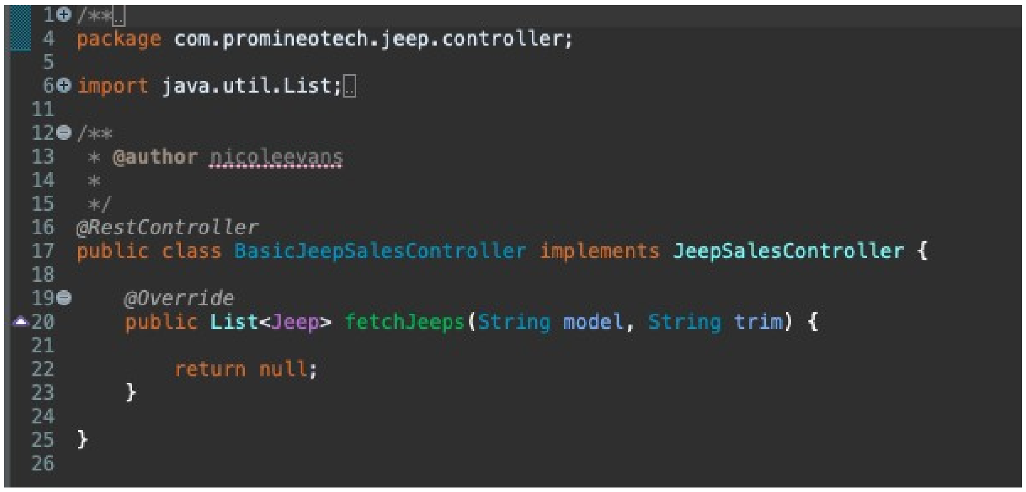
```
20:34:32.586 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.de

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.2)

2021-12-28 20:34:33.177  INFO 13776 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Starting JeepSales using Java 11.0.12 on Nicoles-MBP.hitr
2021-12-28 20:34:33.178  INFO 13776 --- [  restartedMain] com.promineotech.jeep.JeepSales          : No active profile set, falling back to default profiles: 
2021-12-28 20:34:33.259  INFO 13776 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.a
2021-12-28 20:34:33.260  INFO 13776 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the '
2021-12-28 20:34:34.579  INFO 13776 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-12-28 20:34:34.592  INFO 13776 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-12-28 20:34:34.592  INFO 13776 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.56]
2021-12-28 20:34:34.684  INFO 13776 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-12-28 20:34:34.684  INFO 13776 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1
2021-12-28 20:34:35.873  INFO 13776 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2021-12-28 20:34:35.930  INFO 13776 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path
2021-12-28 20:34:35.949  INFO 13776 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Started JeepSales in 3.346 seconds (JVM running for 4.481
2021-12-28 20:35:21.909  INFO 13776 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-12-28 20:35:21.910  INFO 13776 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2021-12-28 20:35:21.913  INFO 13776 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
2021-12-28 20:35:23.570  INFO 13776 --- [nio-8080-exec-6] o.springdoc.api.AbstractOpenApiResource  : Init duration for springdoc-openapi is: 685 ms
```

Extra Screenshots incase I missed anything?

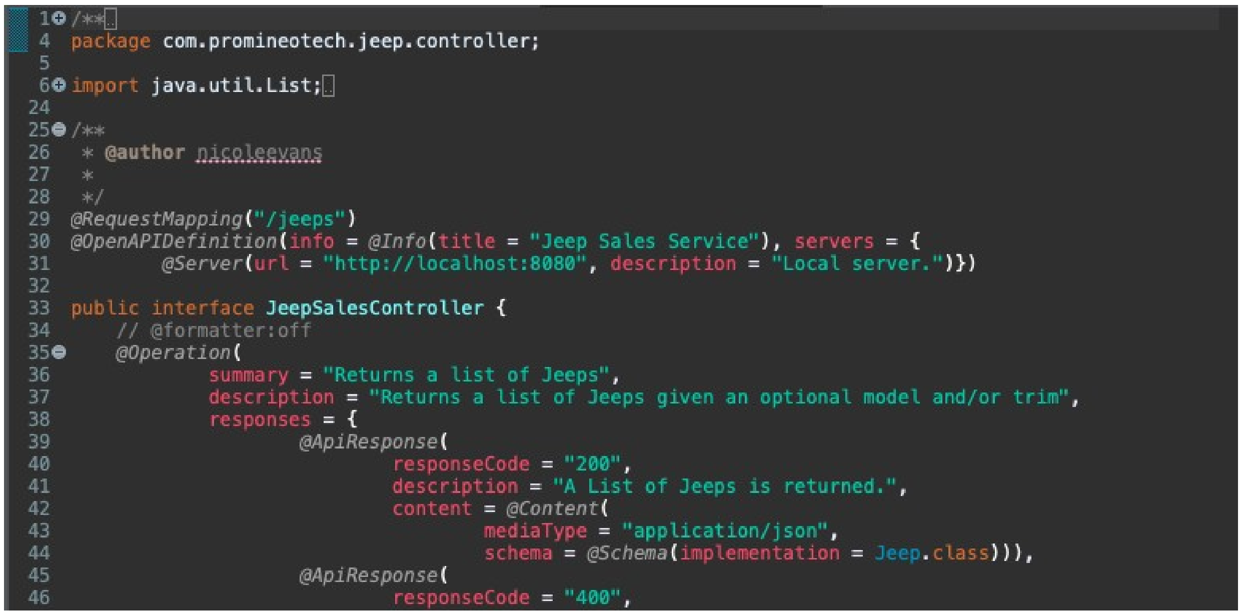


JeepSales.java

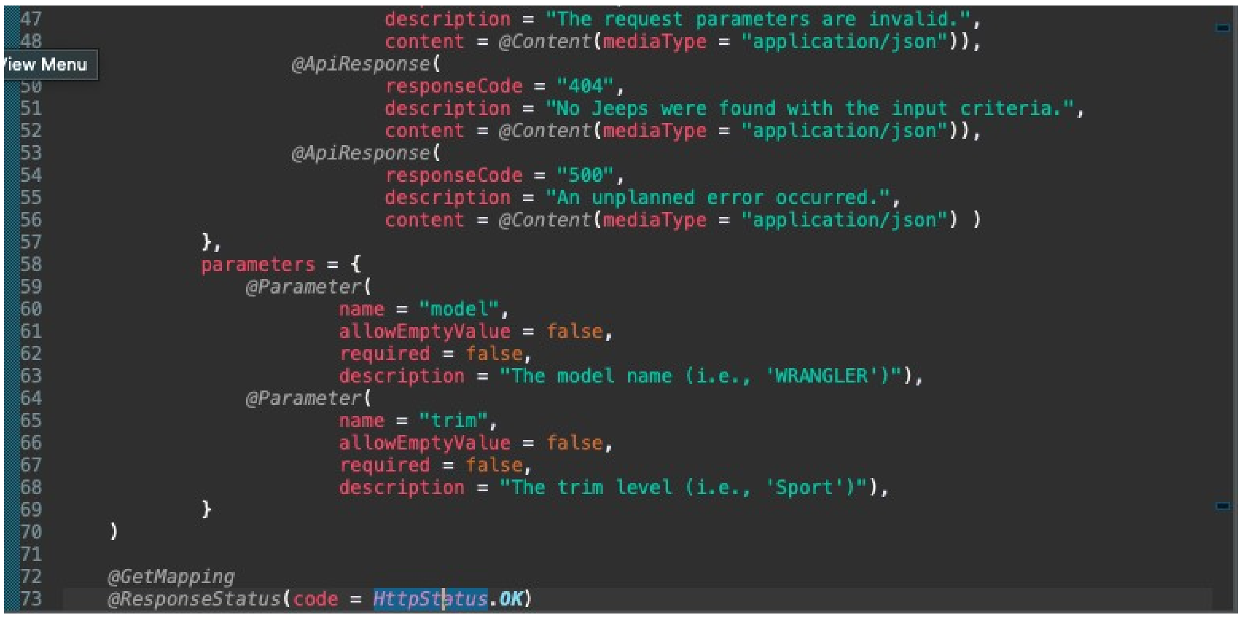
```
/**
package com.promineotech.jeep;

import org.springframework.boot.SpringApplication;

/**
 * @author nicoleevans
 *
 */
@SpringBootApplication
public class JeepSales {

    /**
     * @param args
     */
    public static void main(String[] args) {
        SpringApplication.run(JeepSales.class, args);

    }

    //ask about this, only way i could get the getRestTemplate to work.
    public RestTemplate getRestTemplate() {
        return new RestTemplate();
    }

}
```



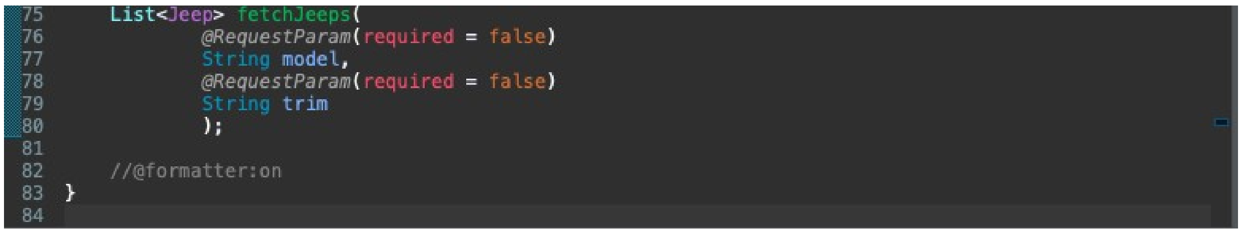
```
/**
package com.promineotech.jeep.controller;

import java.util.List;

/**
 * @author nicoleevans
 *
 */
@RestController
public class BasicJeepSalesController implements JeepSalesController {

    @Override
    public List<Jeep> fetchJeeps(String model, String trim) {

        return null;
    }

}
```



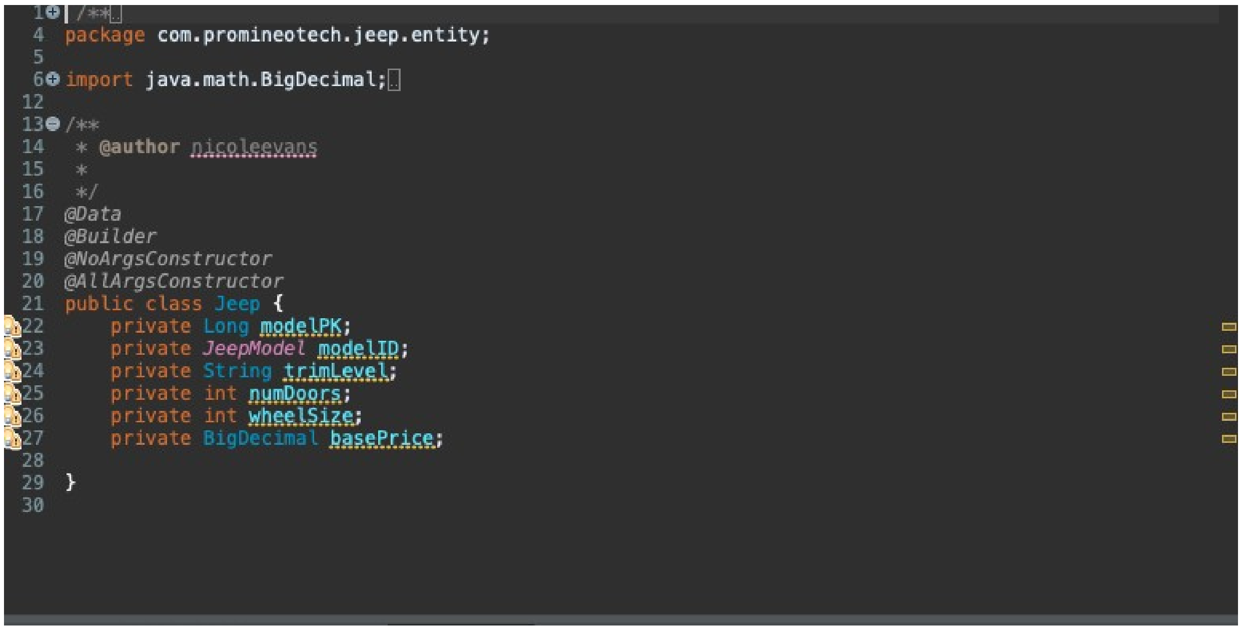
```java
/**
package com.promineotech.jeep.controller;

import java.util.List;

/**
 * @author nicoleevans
 *
 */
@RequestMapping("/jeeps")
@OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
        @Server(url = "http://localhost:8080", description = "Local server.")})

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 is returned.",
                            content = @Content(
                                    mediaType = "application/json",
                                    schema = @Schema(implementation = Jeep.class))),
                    @ApiResponse(
                            responseCode = "400",
```



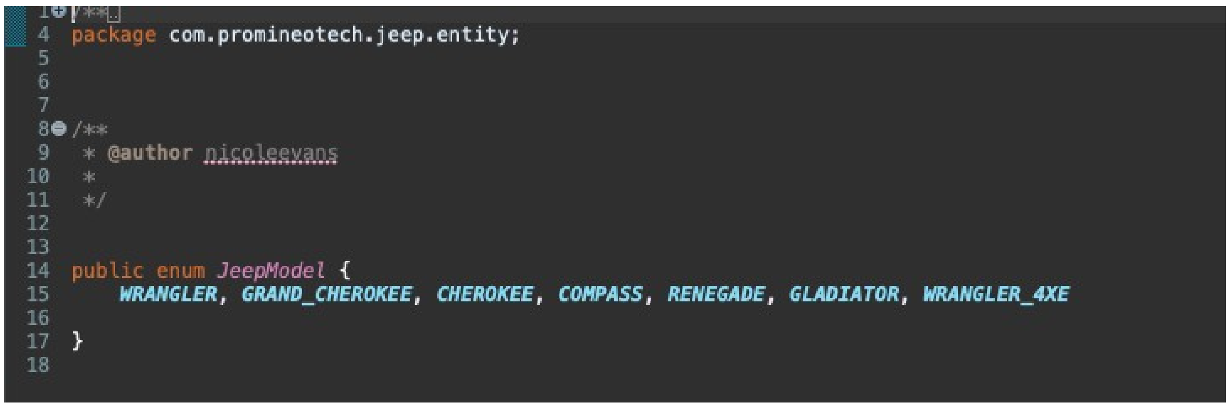
```java
                            description = "The request parameters are invalid.",
                            content = @Content(mediaType = "application/json")),
                    @ApiResponse(
                            responseCode = "404",
                            description = "No Jeeps were found with the input criteria.",
                            content = @Content(mediaType = "application/json")),
                    @ApiResponse(
                            responseCode = "500",
                            description = "An unplanned error occurred.",
                            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 trim level (i.e., 'Sport')"),
            }
    )

    @GetMapping
    @ResponseStatus(code = HttpStatus.OK)
```



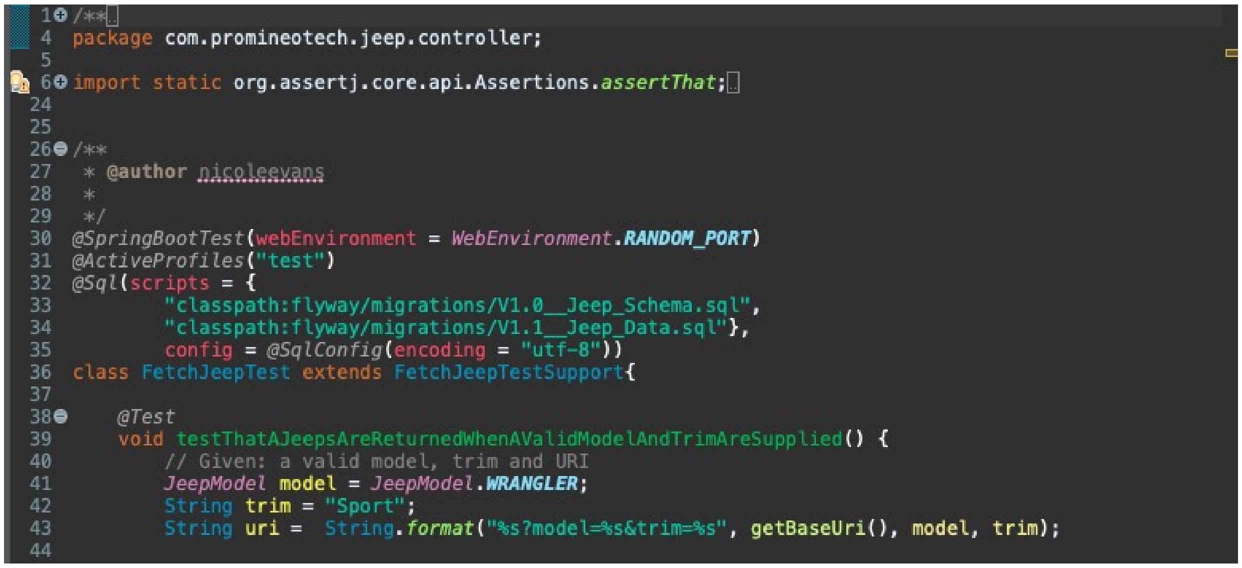
```java
    List<Jeep> fetchJeeps(
            @RequestParam(required = false)
            String model,
            @RequestParam(required = false)
            String trim
            );

    //@formatter:on
}
```



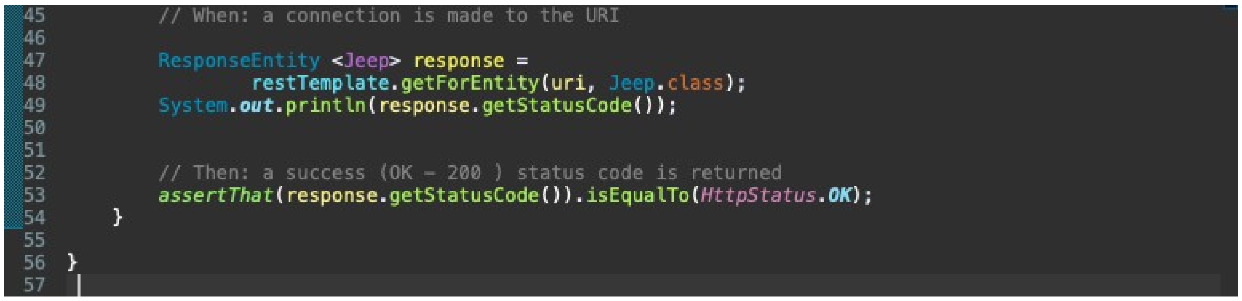
```java
/**
package com.promineotech.jeep.entity;

import java.math.BigDecimal;

/**
 * @author nicoleevans
 *
 */
@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;

}
```



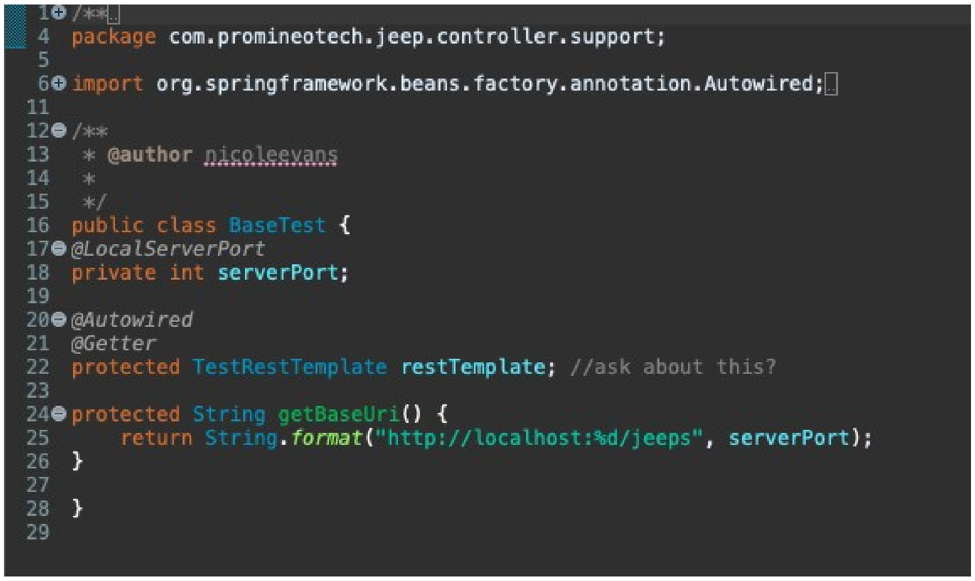
```java
/**
package com.promineotech.jeep.entity;



/**
 * @author nicoleevans
 *
 */


public enum JeepModel {
    WRANGLER, GRAND_CHEROKEE, CHEROKEE, COMPASS, RENEGADE, GLADIATOR, WRANGLER_4XE

}
```



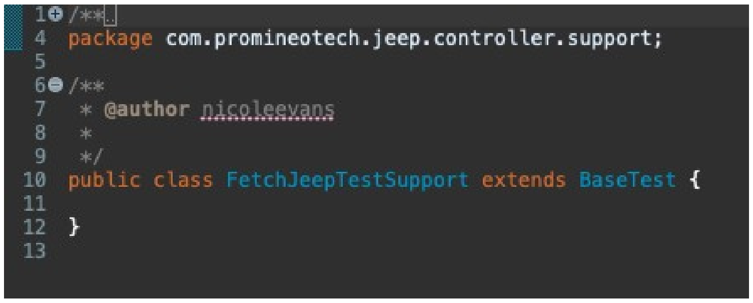
```java
/**
package com.promineotech.jeep.controller;

import static org.assertj.core.api.Assertions.assertThat;


/**
 * @author nicoleevans
 *
 */
@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 extends FetchJeepTestSupport{

    @Test
    void testThatAJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
        // Given: a valid model, trim and URI
        JeepModel model = JeepModel.WRANGLER;
        String trim = "Sport";
        String uri =  String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);

```



```java
        // When: a connection is made to the URI

        ResponseEntity <Jeep> response =
                restTemplate.getForEntity(uri, Jeep.class);
        System.out.println(response.getStatusCode());


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

}
```



```java
/**
package com.promineotech.jeep.controller.support;

import org.springframework.beans.factory.annotation.Autowired;
/**
 * @author nicoleevans
 *
 */
public class BaseTest {
@LocalServerPort
private int serverPort;

@Autowired
@Getter
protected TestRestTemplate restTemplate; //ask about this?

protected String getBaseUri() {
    return String.format("http://localhost:%d/jeeps", serverPort);
}

}
```



```java
/**
package com.promineotech.jeep.controller.support;

/**
 * @author nicoleevans
 *
 */
public class FetchJeepTestSupport extends BaseTest {

}
```

Screenshots of Running Application:

URL to GitHub Repository:

https://github.com/nicolekevans/SpringBootWeek1