


Web API Design with Spring Boot Week 2 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.

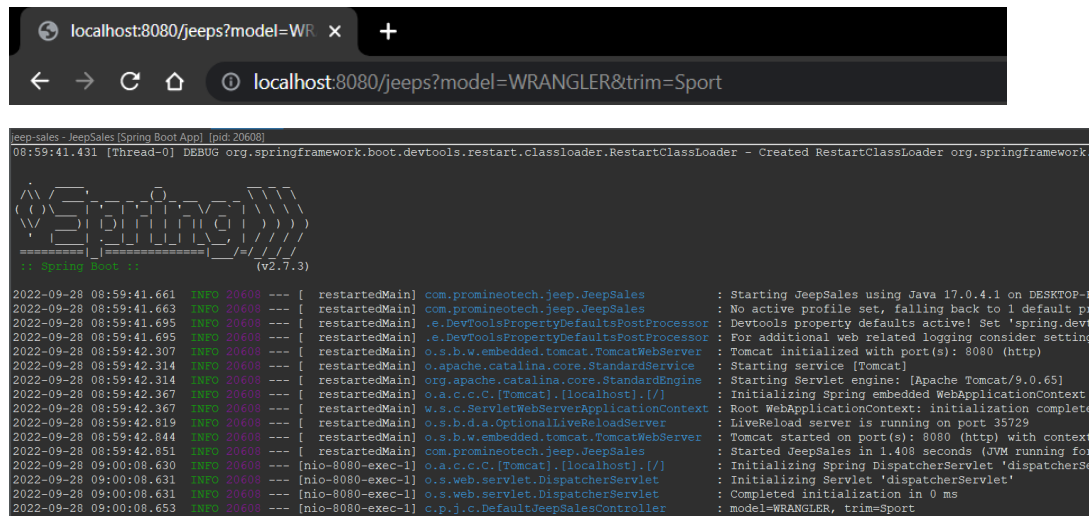
Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon:  You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

- 1) In the project you started last week, use Lombok to add an info-level logging statement in the controller implementation method that logs the parameters that were input to the method. Remember to add the `@Slf4j` annotation to the class.
- 2) Start the application (not an integration test). Use a browser to navigate to the application passing the parameters required for your selected operation. (A browser, used in this manner, sends an HTTP GET request to the server.) Produce a screenshot showing the browser


navigation bar and the log statement that is in the IDE console showing that the controller method was reached (as in the video). 

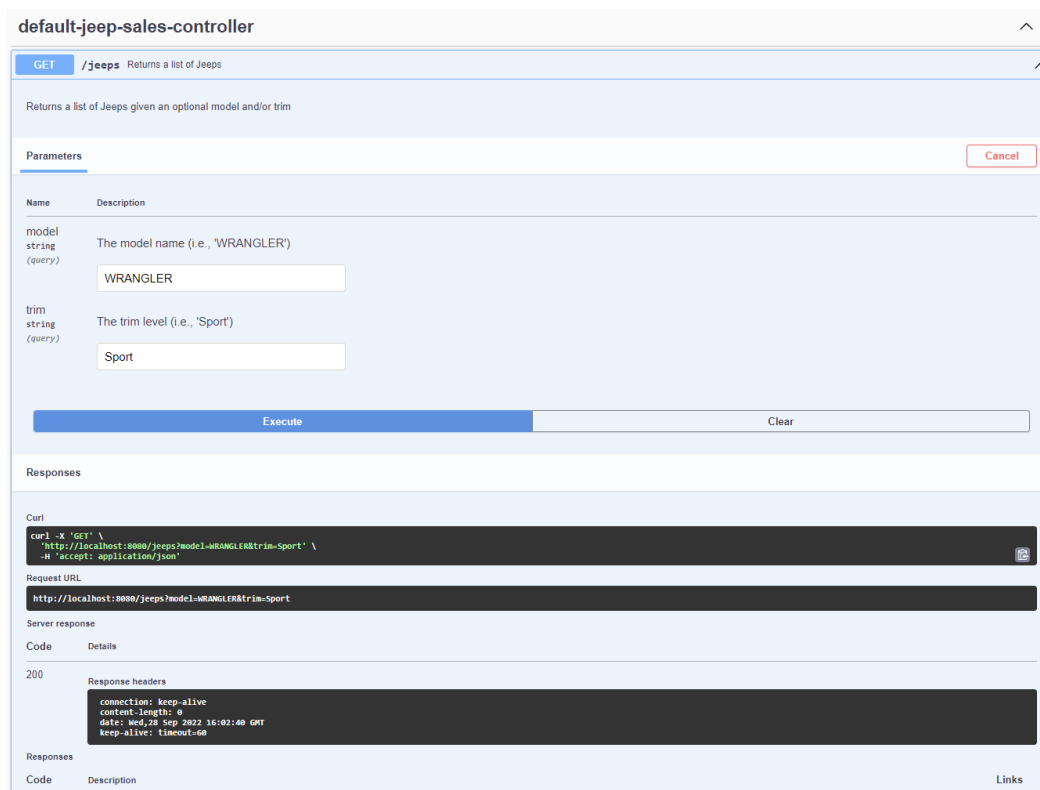


The screenshot shows a web browser at `localhost:8080/jeeps?model=WR` and an IDE console with the following logs:

```

08:59:41.431 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.
...
2022-09-28 08:59:41.661 INFO 20608 --- [ restartedMain] com.promineotech.jee JeepSales : Starting JeepSales using Java 17.0.4.1 on DESKTOP-F
2022-09-28 08:59:41.663 INFO 20608 --- [ restartedMain] com.promineotech.jee JeepSales : No active profile set, falling back to 1 default pr
2022-09-28 08:59:41.695 INFO 20608 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.dev
2022-09-28 08:59:41.695 INFO 20608 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting
2022-09-28 08:59:42.307 INFO 20608 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-09-28 08:59:42.314 INFO 20608 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-28 08:59:42.314 INFO 20608 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-28 08:59:42.367 INFO 20608 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-09-28 08:59:42.367 INFO 20608 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complete
2022-09-28 08:59:42.819 INFO 20608 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-09-28 08:59:42.844 INFO 20608 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context
2022-09-28 08:59:42.851 INFO 20608 --- [ restartedMain] com.promineotech.jee JeepSales : Started JeepSales in 1.408 seconds (JVM running for
2022-09-28 09:00:08.630 INFO 20608 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherSe
2022-09-28 09:00:08.631 INFO 20608 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-09-28 09:00:08.631 INFO 20608 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
2022-09-28 09:00:08.653 INFO 20608 --- [nio-8080-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
  
```

- 3) With the application still running, use the browser to navigate to the OpenAPI documentation. Use the OpenAPI documentation to send a GET request to the server with a valid model and trim level. (You can get the model and trim from the provided `data.sql` file.) Produce a screenshot showing the `curl` command, the request URL, and the response headers. 



The screenshot shows the OpenAPI documentation for the `default-jeep-sales-controller`. The `GET /jeeps` endpoint is selected, with the description "Returns a list of Jeeps given an optional model and/or trim". The parameters section shows `model` (string, query) with value `WRANGLER` and `trim` (string, query) with value `Sport`. The `Execute` button is highlighted. The `Responses` section shows the `curl` command:

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

The request URL is `http://localhost:8080/jeeps?model=WRANGLER&trim=Sport`. The server response is shown with a status code of 200 and the following headers:

```
connection: keep-alive
content-length: 0
date: Wed, 28 Sep 2022 16:02:40 GMT
keep-alive: timeout=60
```

- 4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar.

The screenshot shows an IDE with a Java test class named `FetchJeepTest` and its execution status. The status bar at the top indicates "Finished after 3.162 seconds" and "Runs: 1/1", "Errors: 0", "Failures: 0". The test class code is as follows:

```

package com.promineotech.jeep.controller;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

@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 {

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

        ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
            new ParameterizedTypeReference<>() {});
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    }

    @Autowired
    private TestRestTemplate restTemplate;

    @LocalServerPort
    private int serverPort;
}

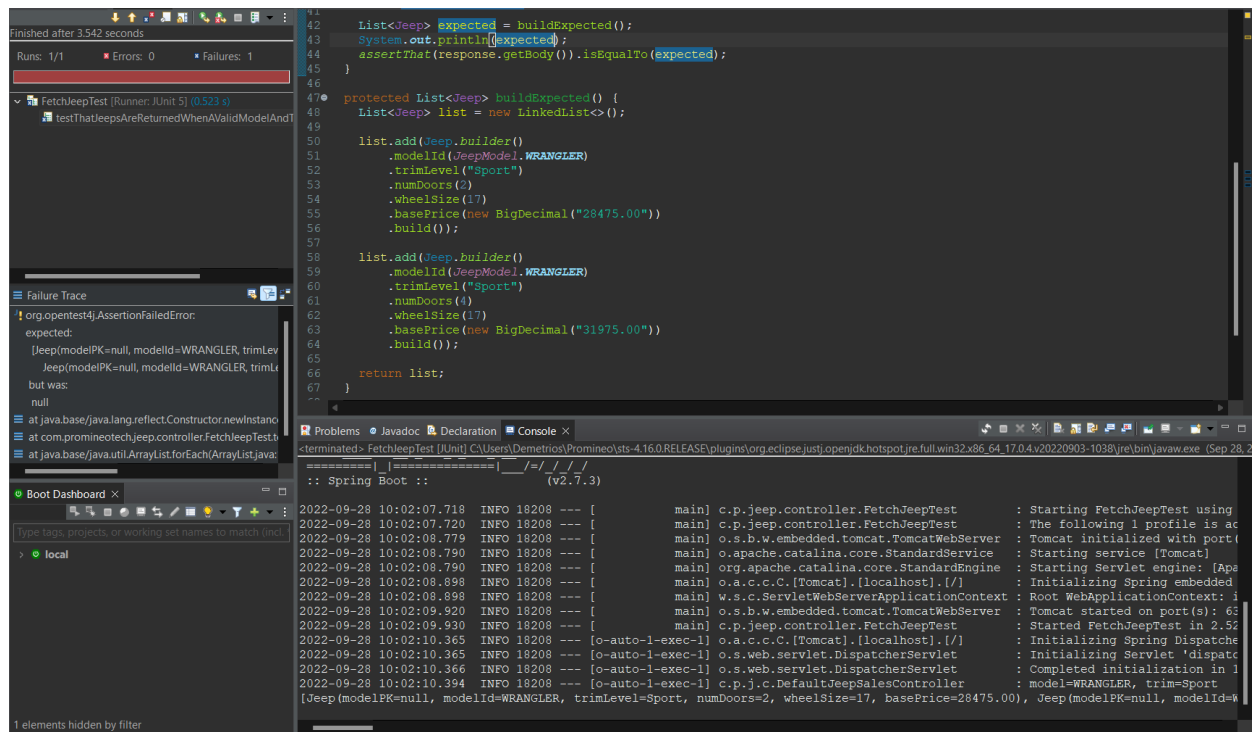
```

- 5) Add a method to the test to return a list of expected Jeep (model) objects based on the model and trim level you selected. You can get the expected list of Jeeps from the file `src/test/resources/flyway/migrations/V1.1__Jeep_Data.sql`. So, for example, using the model Wrangler and trim level "Sport", the query should return two rows:

	Row 1	Row 2
Model ID	WRANGLER	WRANGLER
Trim Level	Sport	Sport
Num Doors	2	4
Wheel Size	17	17
Base Price	\$28,475.00	\$31,975.00


The method should be named `buildExpected()`, and it should return a `List` of `Jeep`. The video put this method into a support superclass but you can include it in the main test class if you want.

- 6) Write an AssertJ assertion in the test to assert that the actual list of jeeps returned by the server is the same as the expected list. Run the test. Produce a screenshot showing...
- The test with the assertion.
 - The JUnit status bar (should be red).
 - The method returning the expected list of Jeeps.



7) Add a service layer in your application as shown in the videos:

- Add a package named `com.promineotech.jeepp.service`.
- In the new package, create an interface named `JeepSalesService`.
- In the same package (service), create a class named `DefaultJeepSalesService` that implements the `JeepSalesService` interface. Add the class-level annotation, `@Service`.
- Inject the service interface into `DefaultJeepSalesController` using the `@Autowired` annotation. The instance variable should be private, and the variable should be named `jeepSalesService`.
- Define the `fetchJeeps` method in the interface. Implement the method in the service class. Call the method from the controller (make sure the controller returns the list of Jeeps returned by the service method). The method signature looks like this:


```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```
- Add a Lombok info-level log statement in the service implementation showing that the service was called. Print the parameters passed to the method. Let the method return `null` for now.
- Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar. 

The screenshot shows an IDE with three main panels. The top panel displays the source code for `DefaultJeepSalesService` in the `com.promineotech.jeepp.service` package. It implements the `JeepSalesService` interface with a `fetchJeeps` method that logs the input parameters and returns null. The middle panel shows a 'Failure Trace' for a test named `FetchJeepTest`. The error is an `AssertionFailedError` where the expected result was a list of Jeeps, but the actual result was null. The bottom panel is the console, showing the application's startup logs, including the initialization of Spring Boot, Tomcat, and the successful execution of the `FetchJeepTest` method.

- 8) Add the database dependencies described in the video to the POM file (MySQL driver and Spring Boot Starter JDBC). To find them, navigate to <https://mvnrepository.com/>. Search for `mysql-connector-j` and `spring-boot-starter-jdbc`. In the POM file you don't need version numbers for either dependency because the version is included in the Spring Boot Starter Parent.
- 9) Create `application.yaml` in `src/main/resources`. Add the `spring.datasource.url`, `spring.datasource.username`, and `spring.datasource.password` properties to `application.yaml`. The url should be the same as shown in the video (`jdbc:mysql://localhost:3306/jeep`). The password and username should match your setup. If you created the database under your root user, the username is "root", and the password is the root user password. If you created a "jeep" user or other user, use the correct username and password.

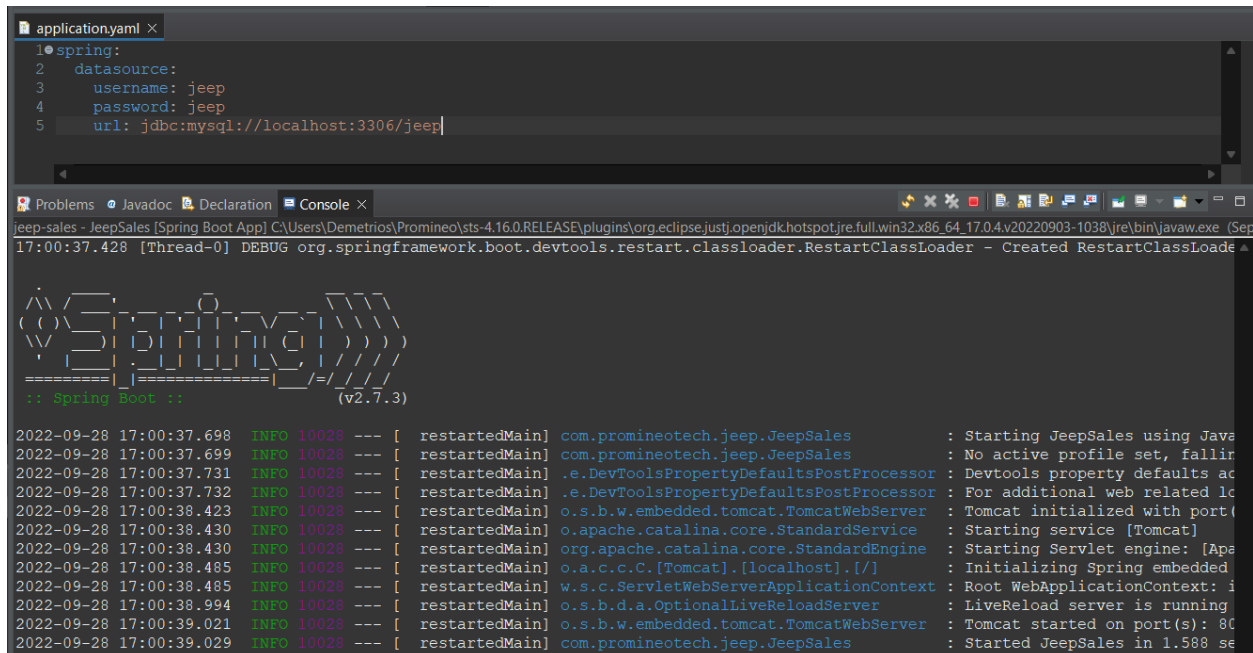
Be careful with the indentation! YAML allows hierarchical configuration but it reads the hierarchy based on the indentation level. The keyword "spring" MUST start in the first column. It should look similar to this when done:

```
spring:
  datasource:
    username: username
    password: password
    url: jdbc:mysql://localhost:3306/jeep
```

- 10) Start the application (the real application, not the test). Produce a screenshot that shows `application.yaml` and the console showing that the application has started with no errors. 🖥️

Demetrios Kopatsis

09/28/2022



The screenshot shows an IDE with two panels. The top panel displays the `application.yml` file with the following content:

```
1 spring:
2   datasource:
3     username: jeep
4     password: jeep
5     url: jdbc:mysql://localhost:3306/jeep
```


The bottom panel shows the console output, which includes a debug message from `org.springframework.boot.devtools.restart.classloader.RestartClassLoader` and a log of the application startup process. The log shows the application starting on 2022-09-28 at 17:00:37.698, with various components like `com.promineotech.jeepp.JeeppSales` and `org.springframework.boot.devtools.restart.classloader.RestartClassLoader` being initialized. The application started successfully on port 8080.

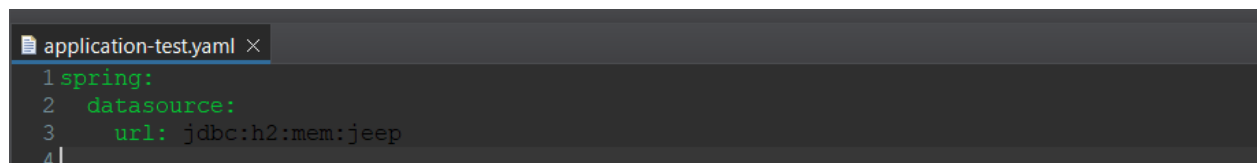
11) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".

12) Create `application-test.yml` in `src/test/resources`. Add the setting `spring.datasource.url` that points to the H2 database. It should look like this:

```
spring:
  datasource:
    url: jdbc:h2:mem:jeep
```

You do not need to set the username and password because the in-memory H2 database does not require them.

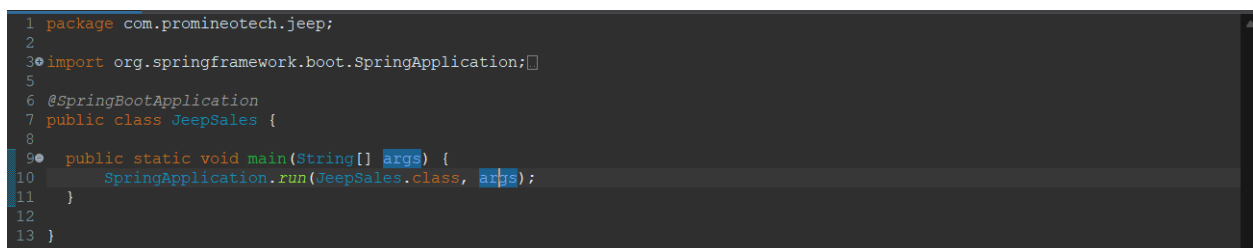
Produce a screenshot showing `application-test.yml`. 



The screenshot shows the `application-test.yml` file in the IDE. The content is:

```
1 spring:
2   datasource:
3     url: jdbc:h2:mem:jeep
4
```

Screenshots of Code:



The screenshot shows the `JeepSales.java` file in the IDE. The code is as follows:

```
1 package com.promineotech.jeepp;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class JeepSales {
8
9   public static void main(String[] args) {
10     SpringApplication.run(JeepSales.class, args);
11   }
12
13 }
```

```

1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
24 @ActiveProfiles("test")
25 @Sql(scripts = {
26     "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
27     "classpath:flyway/migrations/V1.1_Jeep_Data.sql"},
28     config = @SqlConfig(encoding = "utf-8"))
29 class FetchJeepTest {
30
31     @Test
32     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
33         JeepModel model = JeepModel.WRANGLER;
34         String trim = "Sport";
35         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
36
37         ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
38             new ParameterizedTypeReference<>() {});
39         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
40
41         List<Jeep> expected = buildExpected();
42         System.out.println(expected);
43         assertThat(response.getBody()).isEqualTo(expected);
44     }
45
46     protected List<Jeep> buildExpected() {
47         List<Jeep> list = new LinkedList<>();
48
49         list.add(Jeep.builder()
50             .modelId(JeepModel.WRANGLER)
51             .trimLevel("Sport")
52             .numDoors(2)
53             .wheelSize(17)
54             .basePrice(new BigDecimal("28475.00"))
55             .build());
56
57         list.add(Jeep.builder()
58             .modelId(JeepModel.WRANGLER)
59             .trimLevel("Sport")
60             .numDoors(4)
61             .wheelSize(17)
62             .basePrice(new BigDecimal("31975.00"))
63             .build());
64
65         return list;
66     }
67
68     @Autowired
69     private TestRestTemplate restTemplate;
70     @LocalServerPort
71     private int serverPort;
72
73 }
74

```

```

1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10 @RestController
11 @Slf4j
12 public class DefaultJeepSalesController implements JeepSalesController {
13
14     @Autowired
15     private JeepSalesService jeepSalesService;
16
17
18     @Override
19     public List<Jeep> fetchJeeps(String model, String trim) {
20         log.debug("model={}, trim={}", model, trim);
21         return jeepSalesService.fetchJeeps(model, trim);
22     }
23
24 }
25

```

```

1 package com.promineotech.jeep.service;
2
3 import java.util.List;
4
5
6 public interface JeepSalesService {
7
8     List<Jeep> fetchJeeps(String model, String trim);
9 }
10

```

09/28/2022

```
1 package com.promineotech.jeeptest.service;
2
3 import java.util.List;
4
5
6 @Service
7 @Slf4j
8 public class DefaultJeepSalesService implements JeepSalesService {
9
10     @Override
11     public List<Jeep> fetchJeeps(String model, String trim) {
12         log.info("The fetchJeeps method was called with model={} and trim={}", model, trim);
13         System.out.println("Model: " + model + " Trim: " + trim);
14         return null;
15     }
16 }
17
18
19
```

Screenshots of Running Application:

```

  ____  __
 / ___/  /
/  _  /___
/  / /___/
/___/___/

Spring Boot
(v2.7.3)

2022-09-28 17:08:30.616 INFO 24088 --- [ restartedMain] com.promineotech.jeepp.JeepSales : Starting JeepSales using Java
2022-09-28 17:08:30.617 INFO 24088 --- [ restartedMain] com.promineotech.jeepp.JeepSales : No active profile set, fallin
2022-09-28 17:08:30.648 INFO 24088 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults ac
2022-09-28 17:08:30.648 INFO 24088 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For additional web related lc
2022-09-28 17:08:31.330 INFO 24088 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(
2022-09-28 17:08:31.337 INFO 24088 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-28 17:08:31.337 INFO 24088 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Aps
2022-09-28 17:08:31.400 INFO 24088 --- [ restartedMain] a.a.c.c.c.localhost.localhost.1 : Initializing Spring embedded
2022-09-28 17:08:31.400 INFO 24088 --- [ restartedMain] w.s.c.RootWebApplicationContext : Root WebApplicationContext: i
2022-09-28 17:08:31.932 INFO 24088 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running.
2022-09-28 17:08:31.903 INFO 24088 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 80
2022-09-28 17:08:31.941 INFO 24088 --- [ restartedMain] com.promineotech.jeepp.JeepSales : Started JeepSales in 1.598 se

```

URL to GitHub Repository:

<https://github.com/kopatsis/JeepSalesSpring>