

# 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). 


- 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. 
- 4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar. 
- 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...
  - a) The test with the assertion.
  - b) The JUnit status bar (should be red).
  - c) The method returning the expected list of Jeeps. 
- 7) Add a service layer in your application as shown in the videos:
  - a) Add a package named `com.promineotech.jeep.service`.
  - b) In the new package, create an interface named `JeepSalesService`.
  - c) In the same package (service), create a class named `DefaultJeepSalesService` that implements the `JeepSalesService` interface. Add the class-level annotation, `@Service`.

- d) Inject the service interface into DefaultJeepSalesController using the @Autowired annotation. The instance variable should be private, and the variable should be named jeepSalesService.
  - e) 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);
```
  - f) 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.
  - g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar. 
- 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. 
- 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".

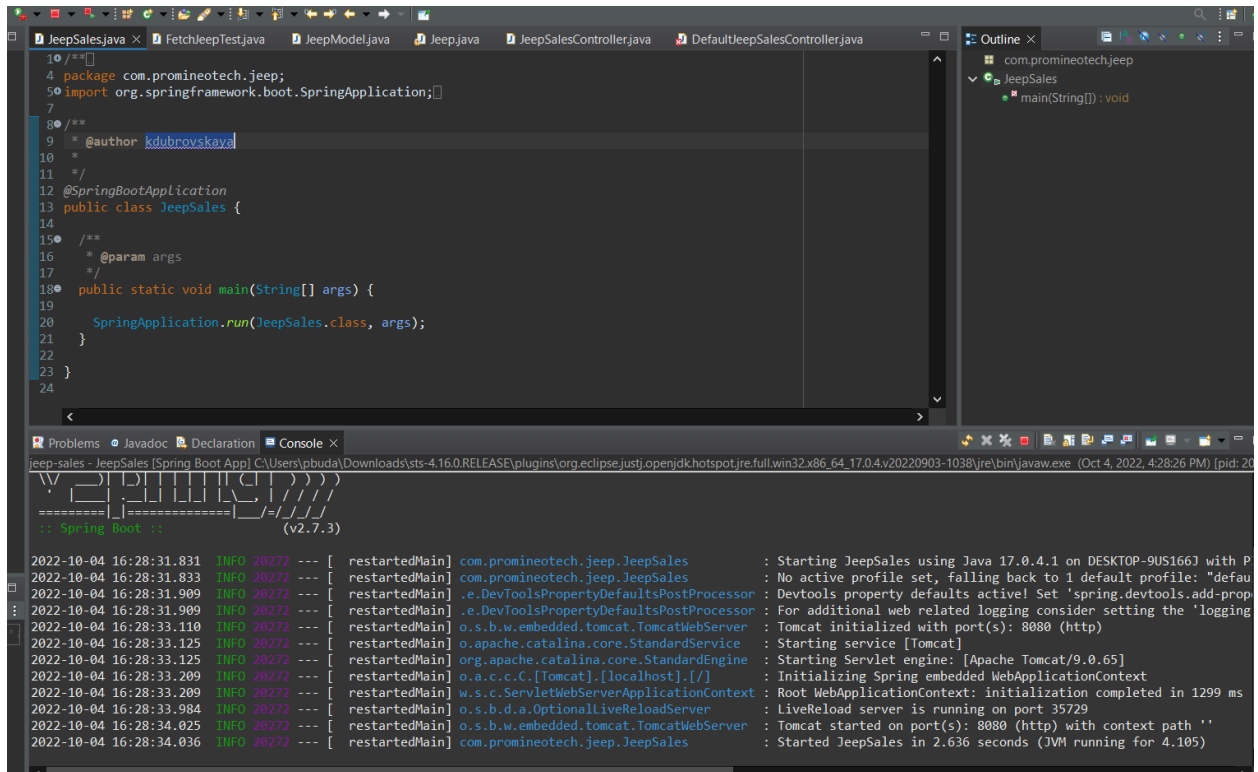
- 12) Create application-test.yaml in src/test/resources. Add the setting spring.datasource.url that points to the H2 database. It should look like this:

```
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.yaml. 

## Screenshots of Code:



The screenshot shows an IDE with the following components:

- Editor:** Displays the `JeepSales.java` file. The code is as follows:

```
10 /**
11  *
12  * @author kdubrovskaya
13  */
14 @SpringBootApplication
15 public class JeepSales {
16     /**
17      * @param args
18      */
19     public static void main(String[] args) {
20         SpringApplication.run(JeepSales.class, args);
21     }
22 }
23
24
```
- Outline:** Shows the package structure: `com.promineotech.jeep` and `JeepSales` with a `main(String[]) : void` method.
- Console:** Displays the output of the application. It starts with the Spring Boot logo and version (v2.7.3). The output shows the application starting successfully on port 8080.

```
2022-10-04 16:28:31.831 INFO 20272 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 17.0.4.1 on DESKTOP-9US166J with P
2022-10-04 16:28:31.833 INFO 20272 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default profile: "defau
2022-10-04 16:28:31.909 INFO 20272 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-prop
2022-10-04 16:28:31.110 INFO 20272 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging
2022-10-04 16:28:33.125 INFO 20272 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-10-04 16:28:33.125 INFO 20272 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-10-04 16:28:33.209 INFO 20272 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-10-04 16:28:33.209 INFO 20272 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-10-04 16:28:33.984 INFO 20272 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1299 ms
2022-10-04 16:28:34.025 INFO 20272 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-10-04 16:28:34.025 INFO 20272 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-10-04 16:28:34.036 INFO 20272 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 2.636 seconds (JVM running for 4.105)
```

Servers

http://localhost:8080 - Generated server url

## default-jEEP-sales-controller

GET /jeeps

### Parameters

Name	Description
model * required	CHEROKEE
trim * required	sport

string  
(query)

string  
(query)

Execute

### Responses

GET /jeeps

### Parameters

Name	Description
model * required	CHEROKEE
trim * required	sport

string  
(query)

string  
(query)

Execute

Clear

### Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/jeeps?model=CHEROKEE&trim=sport' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/jeeps?model=CHEROKEE&trim=sport

Server response

Code	Details
------	---------

Package Explorer | JUnit | FetchJeeptest.java | JeepModel.java | Jeep.java | JeepSalesController.java | DefaultJeepSalesController.java | jeep-sales/po...

hed after 5.098 seconds

ns: 1/1 | Errors: 0 | Failures: 0

FetchJeeptest (Runner: JUnit 5) (0.774 s)

Failure Trace

FetchJeeptest.java

```
18 import org.springframework.test.context.ActiveProfiles;
19 import org.springframework.test.context.jdbc.Sql;
20 import org.springframework.test.context.jdbc.SqlConfig;
21 import com.promineotech.jeeptest.entity.Jeeptest;
22 import com.promineotech.jeeptest.entity.JeeptestModel;
23
24 /**
25  * @author kdubravskaya
26  */
27 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
28 @ActiveProfiles("test")
29 @Sql(scripts = {
30     "classpath:flyway/migrations/V1.0_jeep_schema.sql",
31     "classpath:flyway/migrations/V1.1_jeep_data.sql"),
32     config = @SqlConfig(encoding = "utf-8"))
33
34 class FetchJeeptest {
35
36     @Autowired
37     private TestRestTemplate restTemplate;
38
39     @LocalServerPort
40     private int serverPort;
41
42     @Test
43     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
```

Outline

- com.promineotech.jeeptest.controller
- FetchJeeptest
- restTemplate: TestRestTemplate
- serverPort: int
- testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
- new ParameterizedTypeReference

Problems | Javadoc | Declaration | Console

```
<terminated> FetchJeeptest (JUnit) C:\Users\pbudal\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220903-1038\jre\bin\javaw.exe (Oct 6, 2022, 12:54:17 PM - 1
2022-10-06 12:54:20.367 INFO 3504 --- [main] c.p.jeeptest.controller.FetchJeeptest : The following 1 profile is active: "test"
2022-10-06 12:54:21.748 INFO 3504 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2022-10-06 12:54:21.761 INFO 3504 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-10-06 12:54:21.762 INFO 3504 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-10-06 12:54:21.899 INFO 3504 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-10-06 12:54:21.899 INFO 3504 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 150
2022-10-06 12:54:23.485 INFO 3504 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 20673 (http) with context path '/'
2022-10-06 12:54:23.498 INFO 3504 --- [main] c.p.jeeptest.controller.FetchJeeptest : Started FetchJeeptest in 3.544 seconds (JVM running for 4.9
2022-10-06 12:54:24.287 INFO 3504 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-10-06 12:54:24.287 INFO 3504 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-10-06 12:54:24.288 INFO 3504 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-10-06 12:54:24.327 INFO 3504 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model = WRANGLER, trim = Sport
```

Package Explorer | JUnit | FetchJeeptest.java | JeepModel.java | Jeep.java | JeepSalesController.java | DefaultJeepSalesController.java | FetchJeeptestSupport.java

inished after 5.053 seconds

Runs: 1/1 | Errors: 0 | Failures: 1

FetchJeeptest (Runner: JUnit 5) (0.794 s)

testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()

Failure Trace

org.opentest4j.AssertionFailedError: expected: [] but was: null

FetchJeeptest.java

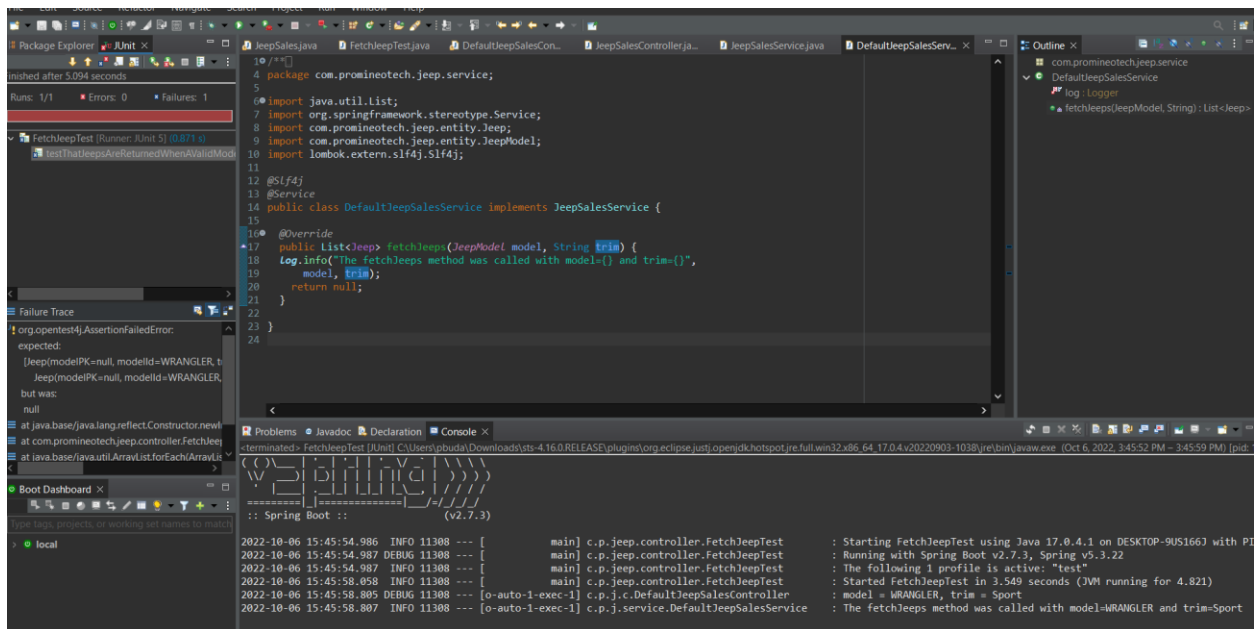
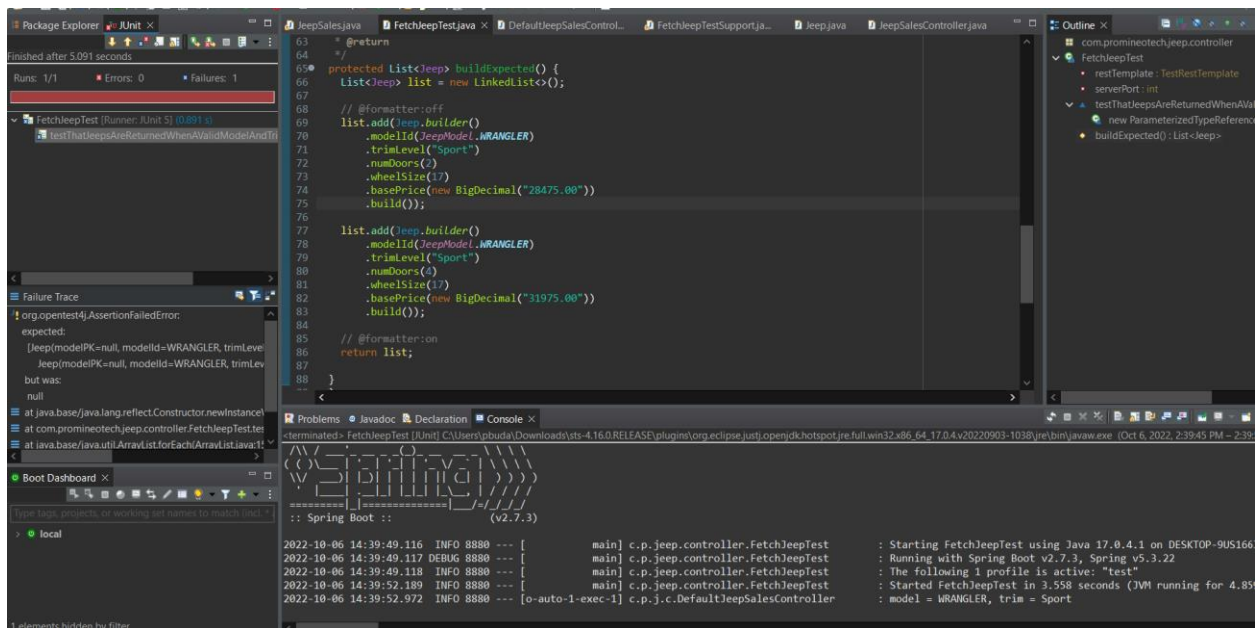
```
51 ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET,
52     null, new ParameterizedTypeReference<>() {});
53 //getRestTemplate().getForEntity(uri, Jeep.class);
54
55 // Then: a success (OK - 200) status code is returned
56 assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
57
58 // And: the actual list returned is the same as the expected list
59 List<Jeep> expected = buildExpected();
60 assertThat(response.getBody()).isEqualTo(expected);
61 }
62 /**
63  * @return
64  */
65 protected List<Jeep> buildExpected() {
66     List<Jeep> list = new LinkedList<>();
67     return list;
68 }
69
70 }
71
72
73
74
75
76
```

Outline

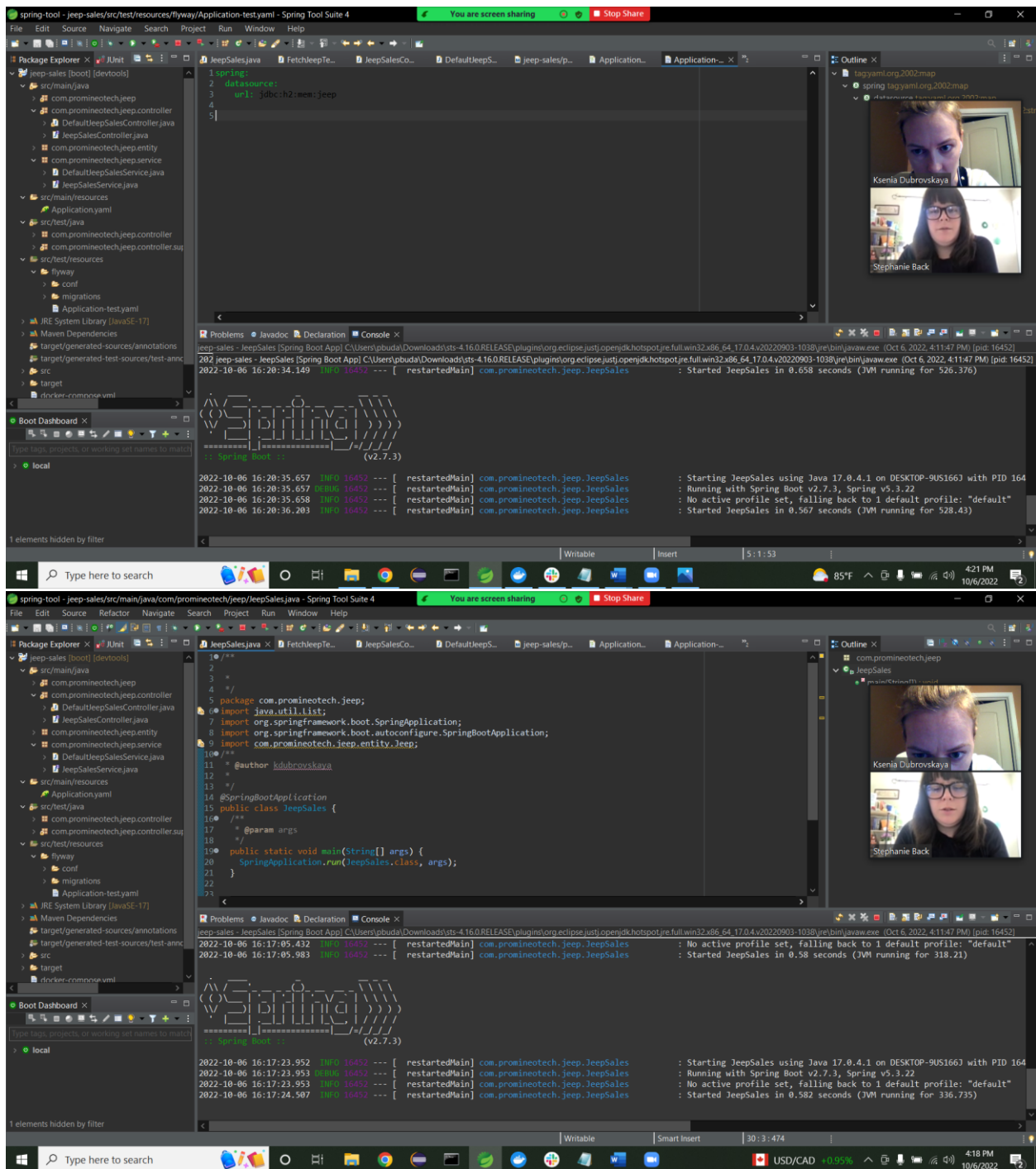
- com.promineotech.jeeptest
- FetchJeeptest
- restTemplate: TestRestTemplate
- serverPort: int
- testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
- new ParameterizedTypeReference

Problems | Javadoc | Declaration | Console

```
<terminated> FetchJeeptest (JUnit) C:\Users\pbudal\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220903-1038\jre\bin\javaw.exe
:: Spring Boot :: (v2.7.3)
2022-10-06 14:10:11.974 INFO 4724 --- [main] c.p.jeeptest.controller.FetchJeeptest : Starting FetchJeeptest using Java 17.0
2022-10-06 14:10:11.975 DEBUG 4724 --- [main] c.p.jeeptest.controller.FetchJeeptest : Running with Spring Boot v2.7.3, Spring
2022-10-06 14:10:11.976 INFO 4724 --- [main] c.p.jeeptest.controller.FetchJeeptest : The following 1 profile is active: "test"
2022-10-06 14:10:15.062 INFO 4724 --- [main] c.p.jeeptest.controller.FetchJeeptest : Started FetchJeeptest in 3.542 seconds
2022-10-06 14:10:15.762 INFO 4724 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model = WRANGLER, trim = Sport
```







## Screenshots of Running Application:

Hello! Please do not lower the points for the work submitted later. I live in the area affected by hurricane Ian. My house is without power and these circumstances affected the deadline for the completion of the work.

Best, Ksenia



**URL to GitHub Repository:**

**<https://github.com/KsuLip2022/Spring-Boot-App-14.git>**