

Web API Design with Spring Boot Week 15 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/f-guedes/jeep-sales.git>


URL to Public Link of your Video: <https://youtu.be/2YGcPIPdt6o>

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.


2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-


Web API Design with Spring Boot Week 15 Coding Assignment

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.

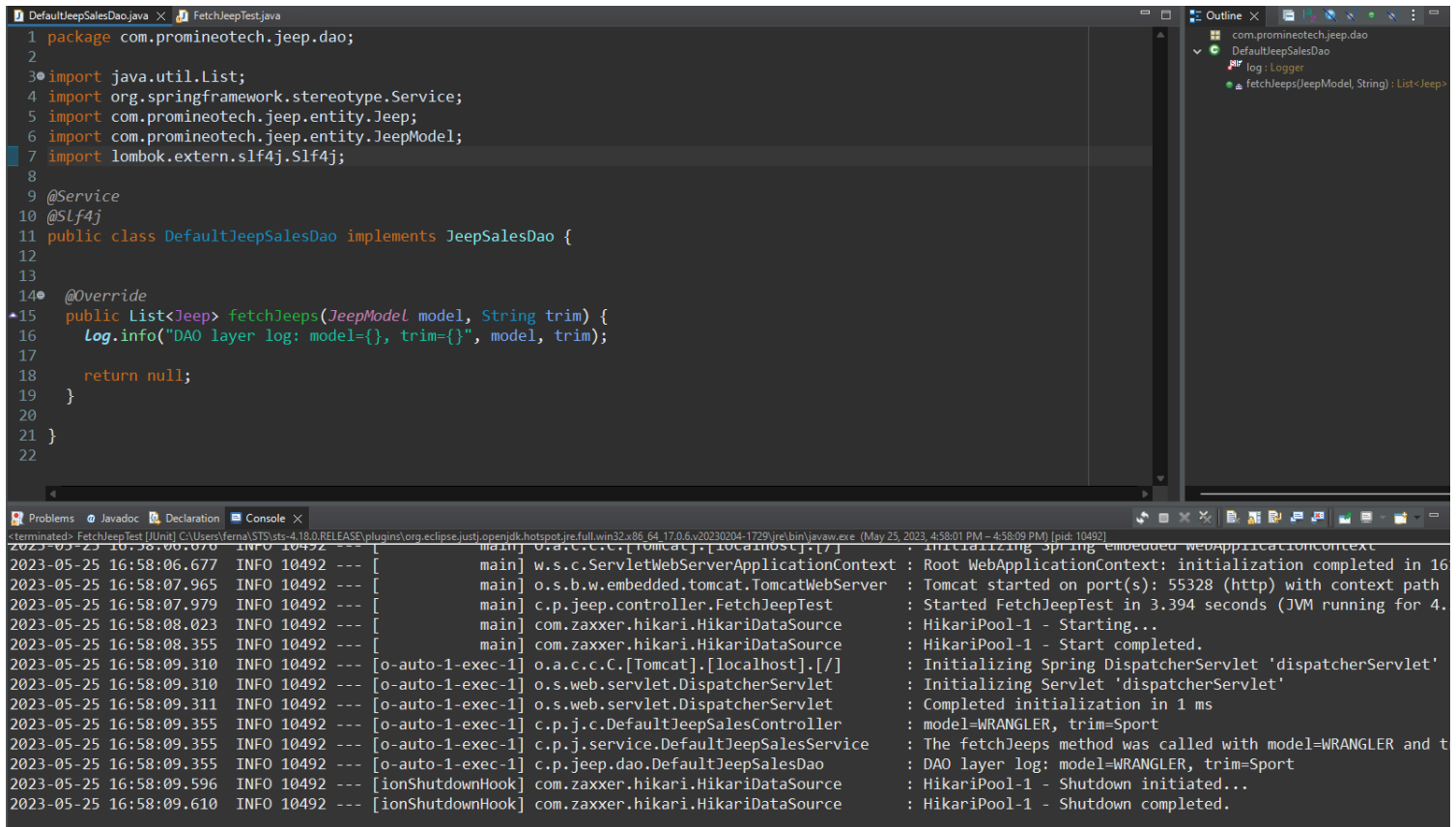
Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

- 1) In the application you've been building add a DAO layer:
 - a) Add the package, com.promineotech.jeepp.dao.
 - b) In the new package, create an interface named JeepSalesDao.
 - c) In the same package, create a class named DefaultJeepSalesDao that implements JeepSalesDao.
 - d) Add a method in the DAO interface and implementation that returns a list of Jeep models (class Jeep) and takes the model and trim parameters. Here is the method signature:

```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```
- 2) In the Jeep sales service implementation class, inject the DAO interface as an instance variable. The instance variable should be private and should be named jeepSalesDao. Call the DAO method from the service method and store the returned value in a local variable named jeeps. Return the value in the jeeps variable (we will add to this later).
- 3) In the DAO implementation class (DefaultJeepSalesDao):
 - a) Add the class-level annotation: @Service.
 - b) Add a log statement in DefaultJeepSalesDao.fetchJeeps() that logs the model and trim level. Run the integration test. Produce a screenshot showing the DAO implementation class and the log line in the IDE's console. 


Web API Design with Spring Boot Week 15 Coding Assignment



The screenshot shows an IDE with two main panels. The top panel displays the code for `DefaultJeepSalesDao.java`, which implements the `JeepSalesDao` interface. The code includes imports for `java.util.List`, `org.springframework.stereotype.Service`, `com.promineotech.jeepp.entity.Jeepp`, `com.promineotech.jeepp.entity.JeeppModel`, and `lombok.extern.slf4j.Slf4j`. The `DefaultJeepSalesDao` class is annotated with `@Service` and `@Slf4j`. It implements the `fetchJeeps(JeeppModel model, String trim)` method, which logs the input parameters and returns `null`. The bottom panel shows the console output, which includes logs from the Spring application context, Tomcat, HikariPool-1, and the `FetchJeepTest` class. The logs indicate that the application started successfully on port 55328, and the `fetchJeeps` method was called with `model=WRANGLER` and `trim=Sport`.


```
1 package com.promineotech.jeepp.dao;
2
3 import java.util.List;
4 import org.springframework.stereotype.Service;
5 import com.promineotech.jeepp.entity.Jeepp;
6 import com.promineotech.jeepp.entity.JeeppModel;
7 import lombok.extern.slf4j.Slf4j;
8
9 @Service
10 @Slf4j
11 public class DefaultJeepSalesDao implements JeepSalesDao {
12
13
14 @Override
15 public List<Jeep> fetchJeeps(JeeppModel model, String trim) {
16     Log.info("DAO layer log: model={}, trim={}", model, trim);
17
18     return null;
19 }
20
21 }
22
```

```
<terminated> FetchJeepTest [Unig] C:\Users\ferna\ST5-sts-4.18.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\javaw.exe (May 25, 2023, 4:58:01 PM - 4:58:09 PM) [pid: 10492]
2023-05-25 16:58:06.677 INFO 10492 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 16
2023-05-25 16:58:07.965 INFO 10492 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 55328 (http) with context path
2023-05-25 16:58:07.979 INFO 10492 --- [main] c.p.jeepp.controller.FetchJeepTest : Started FetchJeepTest in 3.394 seconds (JVM running for 4.
2023-05-25 16:58:08.023 INFO 10492 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-05-25 16:58:08.355 INFO 10492 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-05-25 16:58:09.310 INFO 10492 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-05-25 16:58:09.310 INFO 10492 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-05-25 16:58:09.311 INFO 10492 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2023-05-25 16:58:09.355 INFO 10492 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
2023-05-25 16:58:09.355 INFO 10492 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and t
2023-05-25 16:58:09.355 INFO 10492 --- [o-auto-1-exec-1] c.p.jeepp.dao.DefaultJeepSalesDao : DAO layer log: model=WRANGLER, trim=Sport
2023-05-25 16:58:09.596 INFO 10492 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2023-05-25 16:58:09.610 INFO 10492 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

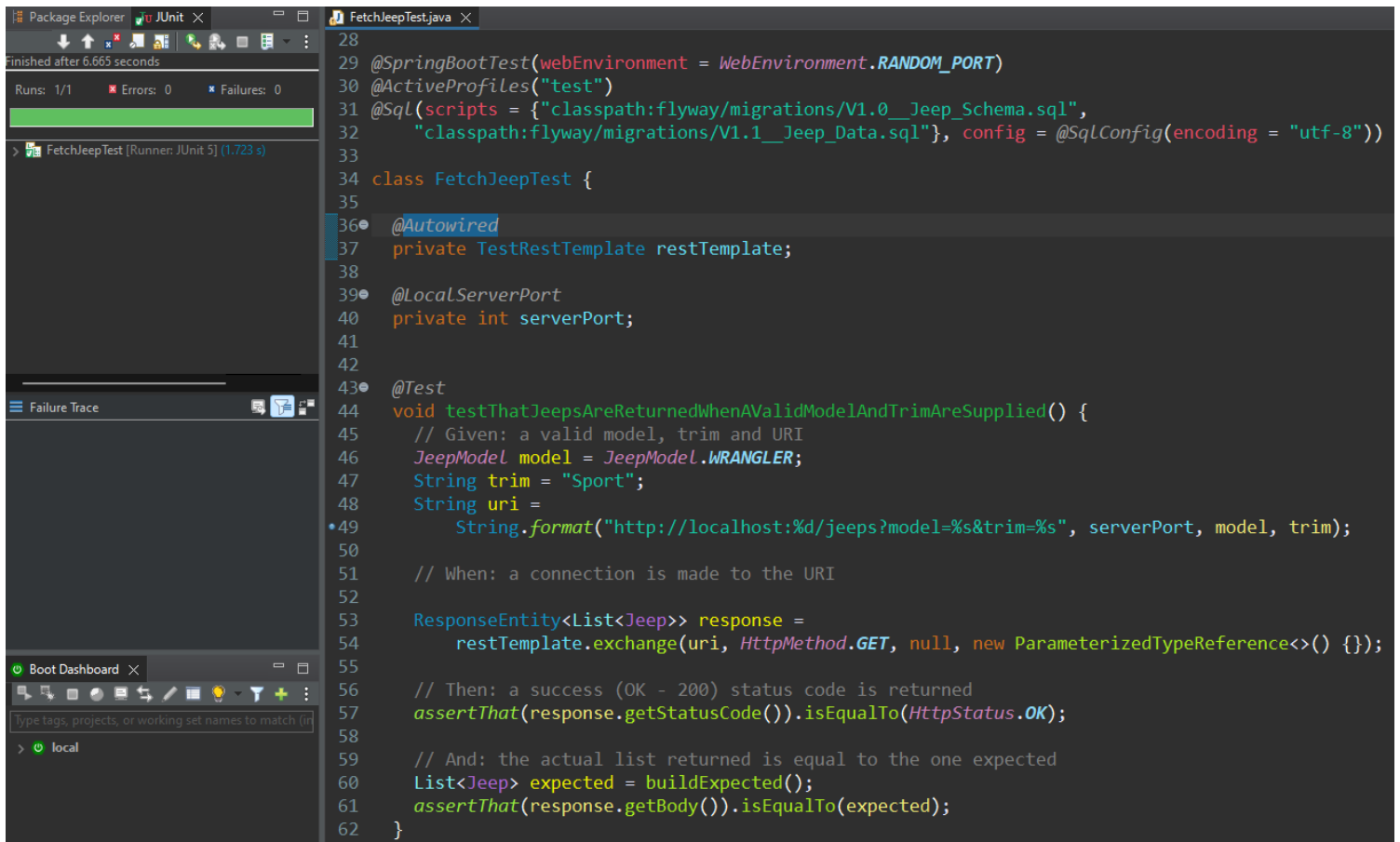
- c) In `DefaultJeepSalesDao`, inject an instance variable of type `NamedParameterJdbcTemplate`.
- d) Write SQL to return a list of Jeep models based on the parameters: `model` and `trim`. Be sure to utilize the SQL Injection prevention mechanism of the `NamedParameterJdbcTemplate` using `:model_id` and `:trim_level` in the query.
- e) Add the parameters to a parameter map as shown in the video. Don't forget to convert the `JeepModel` enum value to a `String` (i.e., `params.put("model_id", model.toString());`)
- f) Call the query method on the `NamedParameterJdbcTemplate` instance variable to return a list of Jeep model objects. Use a `RowMapper` to map each row of the result set. Remember to convert `modelId` to a `JeepModel`. See the video for details. Produce a screenshot to show the complete method in the implementation class. 

Web API Design with Spring Boot Week 15 Coding Assignment

```
DefaultJeepSalesDao.java X
1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
16
17 @Service
18 @Slf4j
19 public class DefaultJeepSalesDao implements JeepSalesDao {
20
21     @Autowired
22     NamedParameterJdbcTemplate jdbcTemplate;
23
24     @Override
25     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
26         log.info("DAO layer log: model={}, trim={}", model, trim);
27
28         String sql = "SELECT * FROM models WHERE model_id = :model_id AND trim_level = :trim_level";
29
30         Map<String, Object> params = new HashMap<>();
31         params.put("model_id", model.toString());
32         params.put("trim_level", trim);
33
34         return jdbcTemplate.query(sql, params, new RowMapper<>() {
35
36             @Override
37             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
38                 // @formatter:off
39                 return Jeep.builder()
40                     .modelPK(rs.getLong("model_pk"))
41                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
42                     .trimLevel(rs.getString("trim_level"))
43                     .numDoors(rs.getInt("num_doors"))
44                     .wheelSize(rs.getInt("wheel_size"))
45                     .basePrice(new BigDecimal(rs.getString("base_price")))
46                     .build();
47                 // @formatter:off
48             }});
49     }
50 }
```

- 4) Add a getter in the Jeep class for modelPK. Add the @JsonIgnore annotation to the getter to exclude the modelPK value from the returned object.
- 5) Run the test to produce a green status bar. Produce a screenshot showing the test and the green status bar. 

Web API Design with Spring Boot Week 15 Coding Assignment



The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with a folder named 'FetchJeepTest'.
- JUnit Runner:** Shows the test results for 'FetchJeepTest'. It indicates 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test was 'Finished after 6.665 seconds'.
- FetchJeepTest.java:** The source code of the test class. It is annotated with `@SpringBootTest`, `@ActiveProfiles("test")`, and `@Sql` to load specific SQL scripts. The class uses `@Autowired` for a `TestRestTemplate` and `@LocalServerPort` for the `serverPort`. A `@Test` method `testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()` is present, which performs an HTTP GET request and asserts the response status and body.

```
28
29 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
30 @ActiveProfiles("test")
31 @Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
32               "classpath:flyway/migrations/V1.1__Jeep_Data.sql"}, 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
43     @Test
44     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
45         // Given: a valid model, trim and URI
46         JeepModel model = JeepModel.WRANGLER;
47         String trim = "Sport";
48         String uri =
49             String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
50
51         // When: a connection is made to the URI
52
53         ResponseEntity<List<Jeep>> response =
54             restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
55
56         // Then: a success (OK - 200) status code is returned
57         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
58
59         // And: the actual list returned is equal to the one expected
60         List<Jeep> expected = buildExpected();
61         assertThat(response.getBody()).isEqualTo(expected);
62     }
}
```