

# Web API Design with Spring Boot Week 15 Coding Assignment

Points possible: 75


URL to GitHub Repository: <https://github.com/ThomasClifton/jeep-sales>

URL to Public Link of your Video: [https://youtu.be/TQO6\\_XPHvdo](https://youtu.be/TQO6_XPHvdo)


---

## 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).

## Web API Design with Spring Boot Week 15 Coding Assignment

3) In the DAO implementation class (DefaultJeepSalesDao):

- Add the class-level annotation: `@Service`.
- 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.

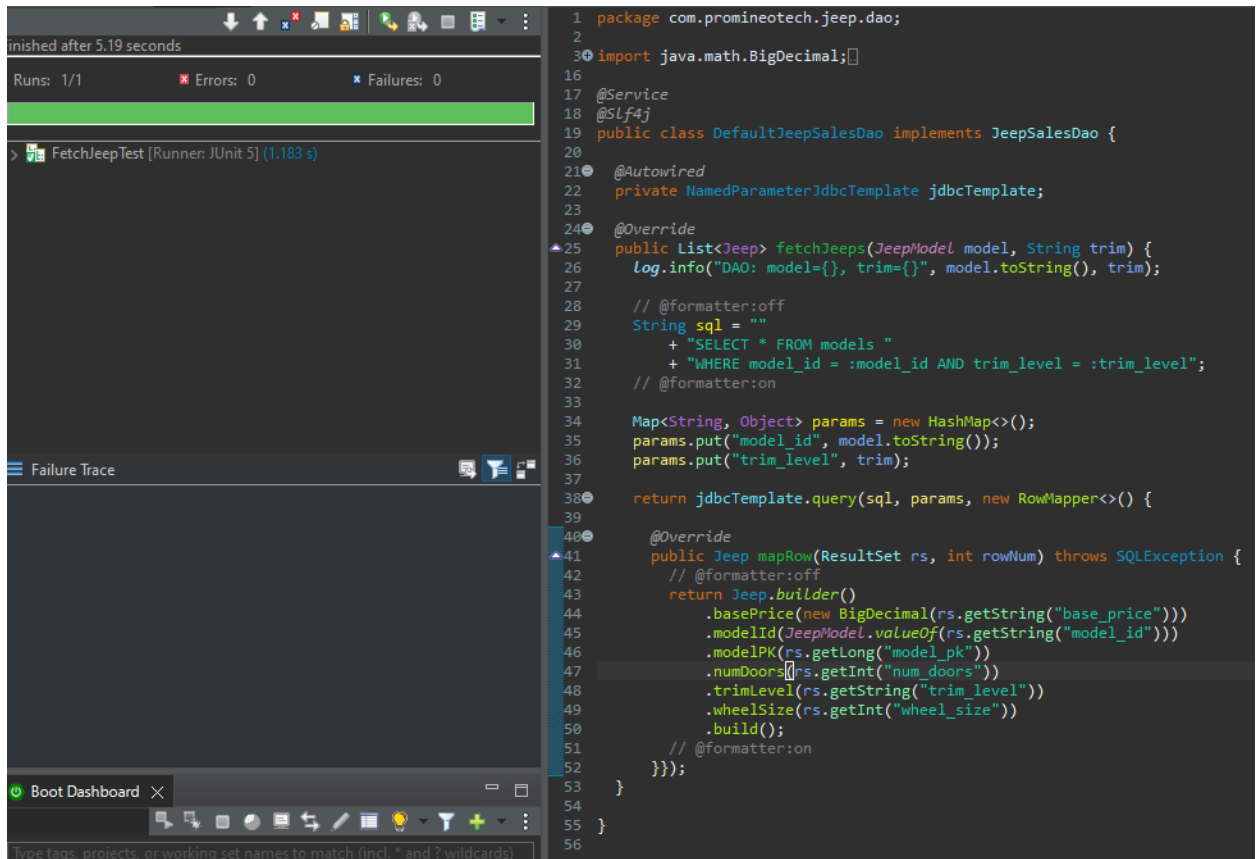
```
1 package com.promineotech.jeep.dao;
2
3 import java.util.List;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
6 import org.springframework.stereotype.Service;
7 import com.promineotech.jeep.entity.Jeep;
8 import com.promineotech.jeep.entity.JeepModel;
9 import lombok.extern.slf4j.Slf4j;
10
11 @Service
12 @Slf4j
13 public class DefaultJeepSalesDao implements JeepSalesDao {
14
15     @Autowired
16     private NamedParameterJdbcTemplate jdbcTemplate;
17
18     @Override
19     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
20         log.info("DAO: model={}, trim={}", model.toString(), trim);
21         return null;
22     }
23 }
24
25
```

```
2023-01-07 17:24:28.548 INFO 21224 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-01-07 17:24:28.548 INFO 21224 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-01-07 17:24:28.549 INFO 21224 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2023-01-07 17:24:28.582 INFO 21224 --- [o-auto-1-exec-1] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport
2023-01-07 17:24:28.582 INFO 21224 --- [o-auto-1-exec-1] c.p.jeep.service.BasicJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim=Sport
2023-01-07 17:24:28.582 INFO 21224 --- [o-auto-1-exec-1] c.p.jeep.dao.DefaultJeepSalesDao : DAO: model=WRANGLER, trim=Sport
2023-01-07 17:24:28.704 INFO 21224 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2023-01-07 17:24:28.706 INFO 21224 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

- In `DefaultJeepSalesDao`, inject an instance variable of type `NamedParameterJdbcTemplate`.
- 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.
- 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());`)
- 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

# Web API Design with Spring Boot Week 15 Coding Assignment

screenshot to show the complete method in the implementation class.



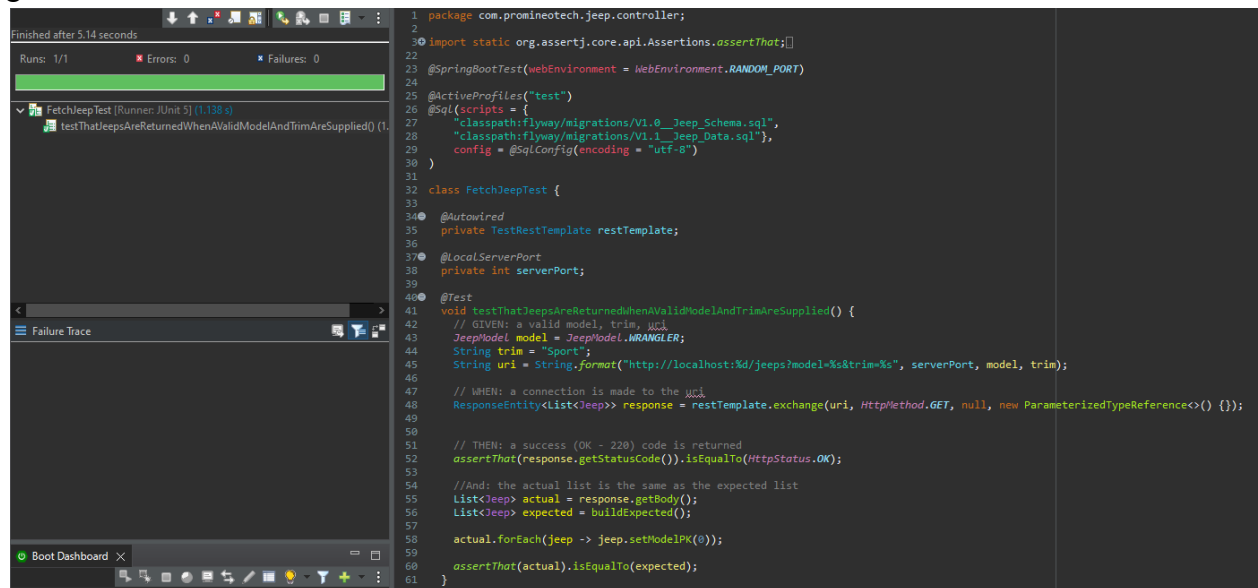
The screenshot shows an IDE with two main panels. The left panel displays the results of a test run for 'FetchJeepTest'. It indicates that the test was 'finished after 5.19 seconds', with 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, a green bar represents the successful execution. The right panel shows the source code of the 'DefaultJeepSalesDao' class, which implements the 'JeepSalesDao' interface. The code includes imports for 'BigDecimal' and 'SLF4j', and is annotated with '@Service' and '@Slf4j'. The 'fetchJeeps' method constructs an SQL query to fetch models based on 'model\_id' and 'trim\_level', and uses a 'RowMapper' to map the results to 'Jeep' objects. The 'mapRow' method is also shown, detailing how database fields are mapped to 'Jeep' object attributes like 'basePrice', 'modelId', 'modelPK', 'numDoors', 'trimLevel', and 'wheelSize'.

```
1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
4
5 @Service
6 @Slf4j
7 public class DefaultJeepSalesDao implements JeepSalesDao {
8
9     @Autowired
10     private NamedParameterJdbcTemplate jdbcTemplate;
11
12     @Override
13     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
14         log.info("DAO: model={}, trim={}", model.toString(), trim);
15
16         // @formatter:off
17         String sql = "
18             + \"SELECT * FROM models \"
19             + \"WHERE model_id = :model_id AND trim_level = :trim_level\";
20         // @formatter:on
21
22         Map<String, Object> params = new HashMap<>();
23         params.put(\"model_id\", model.toString());
24         params.put(\"trim_level\", trim);
25
26         return jdbcTemplate.query(sql, params, new RowMapper<>() {
27
28             @Override
29             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
30                 // @formatter:off
31                 return Jeep.builder()
32                     .basePrice(new BigDecimal(rs.getString(\"base_price\")))
33                     .modelId(JeepModel.valueOf(rs.getString(\"model_id\")))
34                     .modelPK(rs.getLong(\"model_pk\"))
35                     .numDoors(rs.getInt(\"num_doors\"))
36                     .trimLevel(rs.getString(\"trim_level\"))
37                     .wheelSize(rs.getInt(\"wheel_size\"))
38                     .build();
39                 // @formatter:on
40             }
41         });
42     }
43 }
```

- 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.

## Web API Design with Spring Boot Week 15 Coding Assignment

- 5) Run the test to produce a green status bar. Produce a screenshot showing the test and the green status bar.



The screenshot displays an IDE interface with a test runner on the left and a code editor on the right. The test runner shows a green progress bar and a message 'Finished after 5.14 seconds'. Below this, it lists 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A tree view shows a test class 'FetchJeepTest' with a single test method 'testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() (1.138 s)'. The code editor shows the implementation of this test, including imports, annotations, and the test logic. The test logic involves setting up a test environment, creating a test template, and making an HTTP GET request to a specific URL. The response is then checked for a 200 status code and compared to an expected list of Jeeps.

```
1 package com.promineotech.jeepp.controller;
2
30 import static org.assertj.core.api.Assertions.assertThat;
31
32 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
33
34 @ActiveProfiles("test")
35 @Sql(scripts = {
36     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
37     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
38     config = @SqlConfig(encoding = "utf-8")
39 )
40
41 class FetchJeepTest {
42
43     @Autowired
44     private TestRestTemplate restTemplate;
45
46     @LocalServerPort
47     private int serverPort;
48
49     @Test
50     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
51         // GIVEN: a valid model, trim, url
52         JeepModel model = JeepModel.WRANGLER;
53         String trim = "Sport";
54         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
55
56         // WHEN: a connection is made to the url
57         ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
58
59         // THEN: a success (OK - 200) code is returned
60         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
61
62         //And: the actual list is the same as the expected list
63         List<Jeep> actual = response.getBody();
64         List<Jeep> expected = buildExpected();
65
66         actual.forEach(jeep -> jeep.setModelPK(0));
67
68         assertThat(actual).isEqualTo(expected);
69     }
70 }
```