

Web API Design with Spring Boot Week 15 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/Christopher-Needham/SpringProject>


URL to Public Link of your Video: <https://youtu.be/DJbG0SsYRFs>

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.

The screenshot shows an IDE with the `DefaultJeepSalesDao.java` file open. The code is as follows:

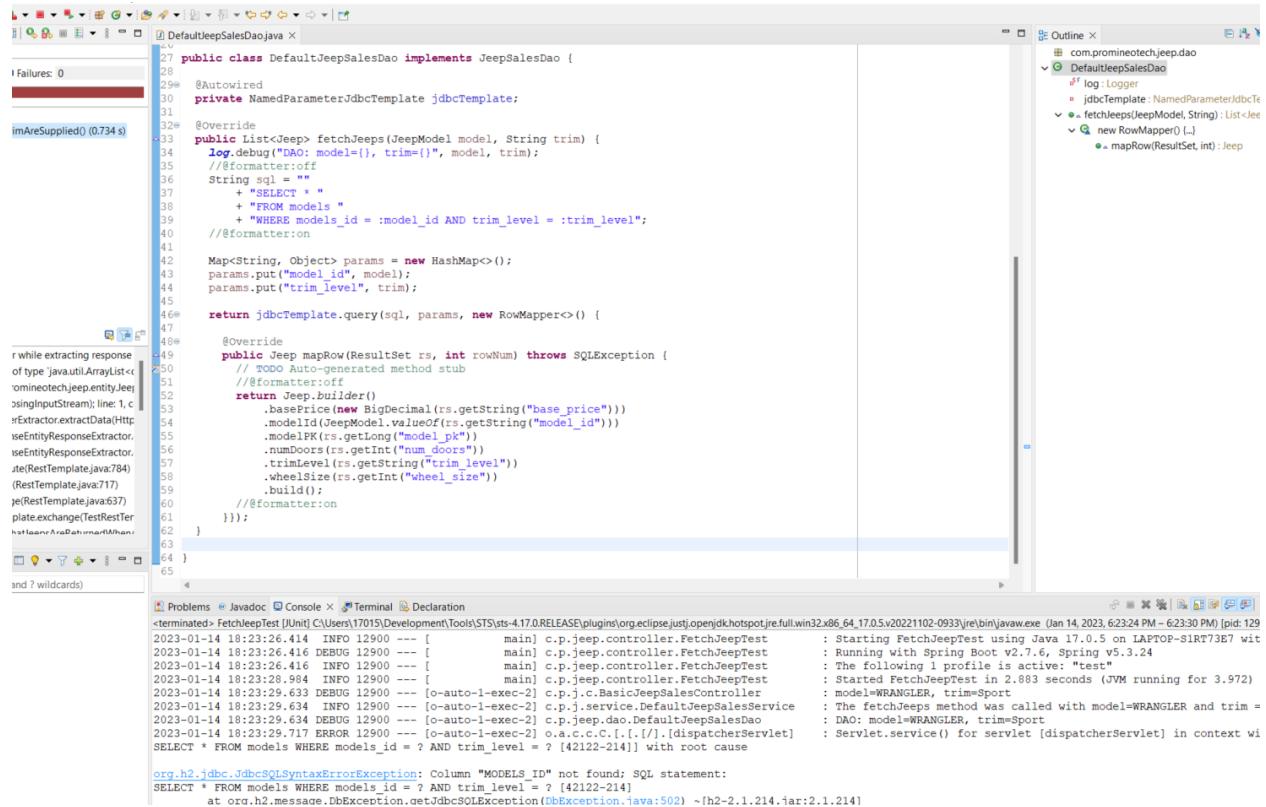
```
27 public class DefaultJeepSalesDao implements JeepSalesDao {
28     @Autowired
29     private NamedParameterJdbcTemplate jdbcTemplate;
30
31     @Override
32     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
33         log.debug("DAO: model={}, trim={}", model, trim);
34         //formatter:off
35         String sql = "
36             + "SELECT * "
37             + "FROM models "
38             + "WHERE models_id = :model_id AND trim_level = :trim_level";
39         //formatter:on
40
41         Map<String, Object> params = new HashMap<>();
42         params.put("model_id", model);
43         params.put("trim_level", trim);
44
45         return jdbcTemplate.query(sql, params, new RowMapper<>() {
46
47             @Override
48             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
49                 // TODO Auto-generated method stub
50                 //formatter:off
51                 return Jeep.builder()
52                     .basePrice(new BigDecimal(rs.getString("base_price")))
53                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
54                     .modelPK(rs.getLong("model_pk"))
55                     .numDoors(rs.getInt("num_doors"))
56                     .trimLevel(rs.getString("trim_level"))
57                     .wheelSize(rs.getInt("wheel_size"))
58                     .build();
59                 //formatter:on
60             }
61         });
62     }
63 }
64
65
```

The console output shows the following log messages:

```
2023-01-14 18:23:26.414 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.5 on LAPTOP-S1RT73E7 wit
2023-01-14 18:23:26.416 DEBUG 12900 --- [main] c.p.jee.controller.FetchJeepTest : Running with Spring Boot v2.7.6, Spring v5.3.24
2023-01-14 18:23:26.416 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : The following 1 profile is active: "test"
2023-01-14 18:23:28.984 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : Started FetchJeepTest in 2.883 seconds (JVM running for 3.972)
2023-01-14 18:23:29.633 DEBUG 12900 --- [o-auto-1-exec-2] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport
2023-01-14 18:23:29.634 INFO 12900 --- [o-auto-1-exec-2] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim =
2023-01-14 18:23:29.717 ERROR 12900 --- [o-auto-1-exec-2] o.a.c.c.C.[.][.][dispatcherServlet] : DAO: model=WRANGLER, trim=Sport
SELECT * FROM models WHERE models_id = ? AND trim_level = ? [42122-214] with root cause
org.h2.jdbc.JdbcSQLException: Column "MODEL_ID" not found: SQL statement:
SELECT * FROM models WHERE models_id = ? AND trim_level = ? [42122-214]
at org.h2.message.DbException.getJdbcSQLException(DbException.java:502) ~[h2-2.1.214.jar:2.1.214]
```

- 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 screenshot to show the complete method in the implementation class.

Web API Design with Spring Boot Week 15 Coding Assignment



```
27 public class DefaultJeepSalesDao implements JeepSalesDao {
28
29     @Autowired
30     private NamedParameterJdbcTemplate jdbcTemplate;
31
32     @Override
33     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
34         log.debug("DAO: model={}, trim={}", model, trim);
35         //formatter:off
36         String sql = "
37             * SELECT *
38             * FROM models
39             * WHERE models_id = :model_id AND trim_level = :trim_level";
40         //formatter:on
41
42         Map<String, Object> params = new HashMap<>();
43         params.put("model_id", model);
44         params.put("trim_level", trim);
45
46         return jdbcTemplate.query(sql, params, new RowMapper<>() {
47
48             @Override
49             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
50                 // TODO Auto-generated method stub
51                 //formatter:off
52                 return Jeep.builder()
53                     .basePrice(new BigDecimal(rs.getString("base_price")))
54                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
55                     .modelPK(rs.getLong("model_pk"))
56                     .numDoors(rs.getInt("num_doors"))
57                     .trimLevel(rs.getString("trim_level"))
58                     .wheelSize(rs.getInt("wheel_size"))
59                     .build();
60                 //formatter:on
61             }
62         });
63     }
64 }
65
```

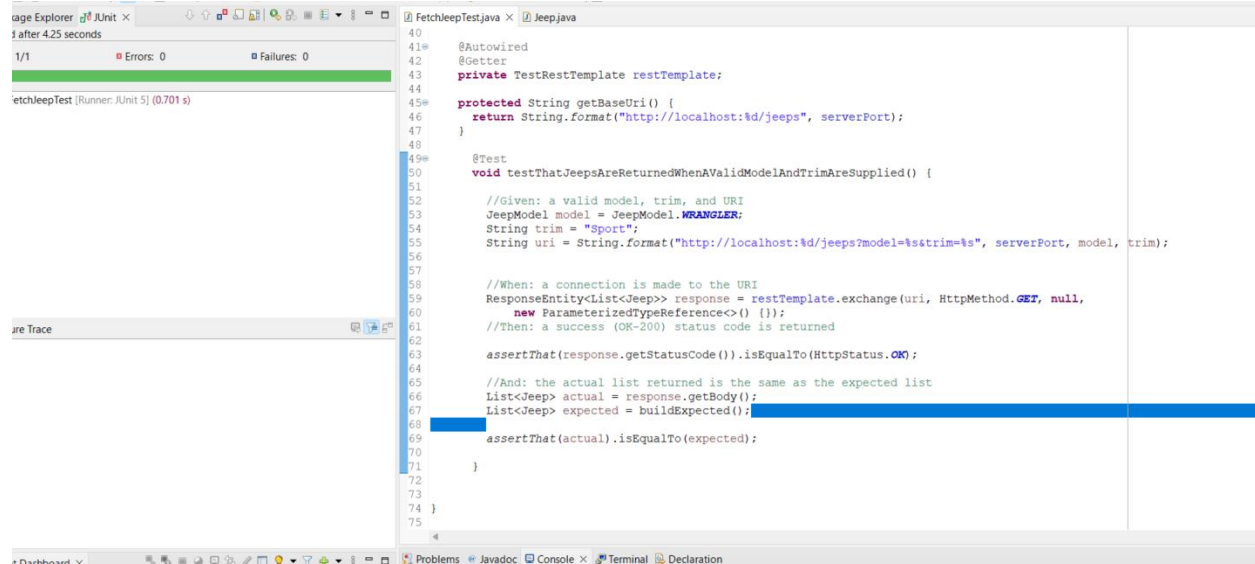
Problems | Javadoc | Console | Terminal | Declaration

```
<terminated> FetchJeepTest [JUnit] C:\Users\17015\Development\Tools\STS\sts-4.17.0.RELEASE\plugins\org.eclipse.justi.openjdk hotspot\jre\full\win32\x86_64\17.0.5\20221102-0933\jre\bin\javaw.exe (Jan 14, 2023, 6:23:24 PM - 6:23:30 PM) [pid: 129
2023-01-14 18:23:26.414 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.5 on LAPTOP-S1RT73B7 wit
2023-01-14 18:23:26.416 DEBUG 12900 --- [main] c.p.jee.controller.FetchJeepTest : Running with Spring Boot v2.7.6, Spring v5.3.24
2023-01-14 18:23:26.416 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : The following 1 profile is active: "test"
2023-01-14 18:23:28.984 INFO 12900 --- [main] c.p.jee.controller.FetchJeepTest : Started FetchJeepTest in 2.883 seconds (JVM running for 3.972)
2023-01-14 18:23:29.633 DEBUG 12900 --- [o-auto-1-exec-2] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport
2023-01-14 18:23:29.634 INFO 12900 --- [o-auto-1-exec-2] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim =
2023-01-14 18:23:29.634 DEBUG 12900 --- [o-auto-1-exec-2] c.p.jee.dao.DefaultJeepSalesDao : DAO: model=WRANGLER, trim=Sport
2023-01-14 18:23:29.717 ERROR 12900 --- [o-auto-1-exec-2] o.a.c.c.C.[.][.][dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context wi
SELECT * FROM models WHERE models_id = ? AND trim_level = ? [42122-214]] with root cause
org.h2.jdbc.JdbcSQLException: Column "MODELS_ID" not found; SQL statement:
SELECT * FROM models WHERE models_id = ? AND trim_level = ? [42122-214]
at org.h2.message.DbException.getJdbcSQLException(DbException.java:502) ~[h2-2.1.214.jar:2.1.214]
```

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:

- Top Left:** A tab labeled "Page Explorer" showing a file tree with "JUnit" and "1/1". Below it, a status bar indicates "Errors: 0" and "Failures: 0".
- Top Right:** A tab labeled "FetchJeepTest.java" showing the source code of the test class. The code includes annotations like `@Autowired`, `@Getter`, and `@Test`. It defines a `TestRestTemplate` and a test method `testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()` which uses `String.format` to build a URL, sends a GET request, and asserts the response status and body.
- Bottom Left:** A tab labeled "JUnit" showing the execution results of the test. It indicates that the test passed after 4.25 seconds.
- Bottom Right:** A tab labeled "JUnit Trace" showing the execution trace of the test.

```
40
41 @Autowired
42 @Getter
43 private TestRestTemplate restTemplate;
44
45 protected String getBaseUrl() {
46     return String.format("http://localhost:%d/jeeps", serverPort);
47 }
48
49 @Test
50 void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
51
52     //Given: a valid model, trim, and URI
53     JeepModel model = JeepModel.WRANGLER;
54     String trim = "Sport";
55     String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
56
57     //When: a connection is made to the URI
58     ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
59         new ParameterizedTypeReference<>() {});
60     //Then: a success (OK-200) status code is returned
61
62     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
63
64     //And: the actual list returned is the same as the expected list
65     List<Jeep> actual = response.getBody();
66     List<Jeep> expected = buildExpected();
67     assertThat(actual).isEqualTo(expected);
68 }
69
70
71
72
73
74
75
```