

Web API Design with Spring Boot Week 15 Coding Assignment


weePoints possible: 75

URL to GitHub Repository: <https://github.com/DavidSteffen1/weekFifteenRepository>


URL to Public Link of your Video: <https://youtu.be/3kAlsaSOjqM>

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.

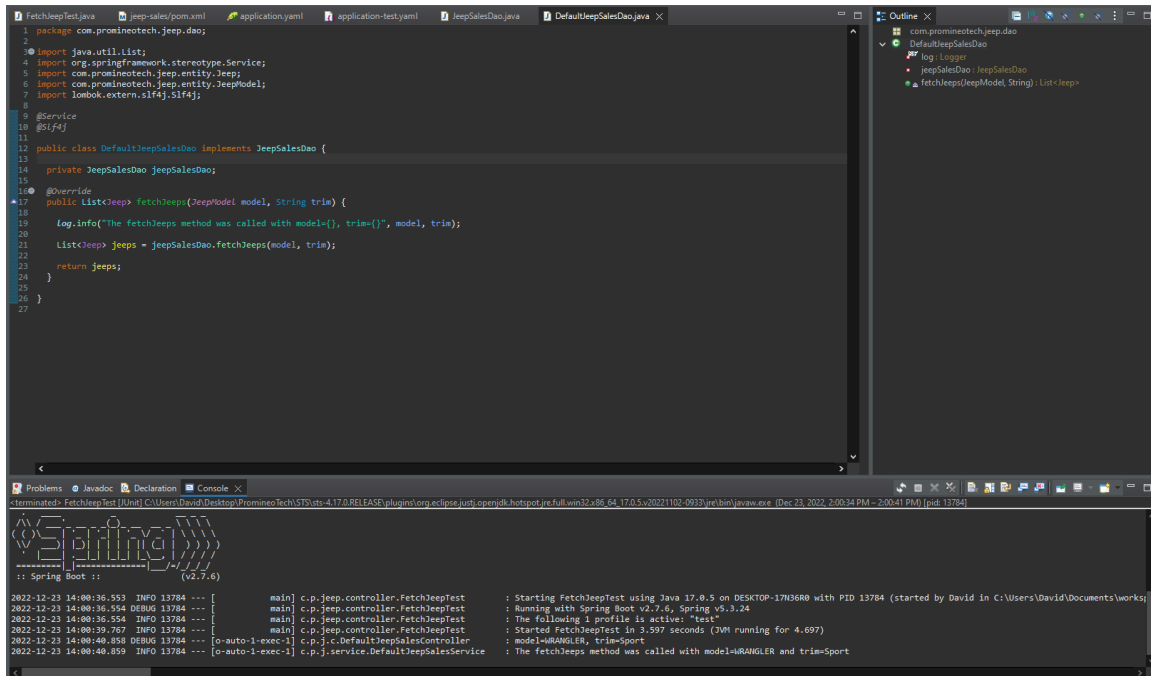
Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

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 windows. The top window displays the `DefaultJeepSalesDao` class, which implements the `JeepSalesDao` interface. It includes a `fetchJeeps` method that logs the input parameters and calls the `fetchJeeps` method on the `JeepSalesDao` interface. The bottom window shows the Spring Boot console output, which includes the Spring logo, version information, and a log of the application startup. The log shows that the application is running with Spring Boot v2.7.6 and Spring v5.3.24, and that the `fetchJeeps` method was called with `model=URANGLER` and `trim=Sport`.

```
1 package com.promineotech.jee.dao;
2
3 import java.util.List;
4 import org.springframework.stereotype.Service;
5 import com.promineotech.jee.entity.Jee;
6 import com.promineotech.jee.entity.JeeModel;
7 import lombok.extern.slf4j.Slf4j;
8
9 @Service
10 @Slf4j
11
12 public class DefaultJeepSalesDao implements JeepSalesDao {
13
14     private JeepSalesDao jeepSalesDao;
15
16     @Override
17     public List<Jee> fetchJeeps(JeeModel model, String trim) {
18
19         log.info("The fetchJeeps method was called with model={}, trim={}", model, trim);
20
21         List<Jee> jeeps = jeepSalesDao.fetchJeeps(model, trim);
22
23         return jeeps;
24     }
25
26 }
27
```

Spring Boot (v2.7.6)

```
2022-12-23 14:00:36.553 INFO 13784 --- [main] c.p.jee.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.5 on DESKTOP-17H36RD with PID 13784 (started by David in C:\Users\David\Documents\work\jeep)
2022-12-23 14:00:36.554 DEBUG 13784 --- [main] c.p.jee.controller.FetchJeepTest : Running with Spring Boot v2.7.6, Spring v5.3.24
2022-12-23 14:00:36.554 INFO 13784 --- [main] c.p.jee.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-12-23 14:00:39.767 INFO 13784 --- [main] c.p.jee.controller.FetchJeepTest : Started FetchJeepTest in 3.597 seconds (JVM running for 4.697)
2022-12-23 14:00:40.858 DEBUG 13784 --- [o-auto-1-exec-1] c.p.jee.DefaultJeepSalesController : model=URANGLER, trim=Sport
2022-12-23 14:00:40.859 INFO 13784 --- [o-auto-1-exec-1] c.p.jee.service.DefaultJeepSalesService : The fetchJeeps method was called with model=URANGLER and trim=Sport
```

- 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

```
DefaultJeepSalesService.java | DefaultJeepSalesDao.java | JeepSalesDao.java | FetchJeepTest.java
1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
4
5 @Component
6 @Service
7 @Slf4j
8
9 public class DefaultJeepSalesDao implements JeepSalesDao {
10
11     @Autowired
12     private NamedParameterJdbcTemplate jdbcTemplate;
13
14     @Override
15     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
16
17         Log.info("DAO: model={}, trim={}", model, trim);
18
19         //List<Jeep> jeeps = jeepSalesDao.fetchJeeps(model, trim);
20
21         // @formatter:off
22         String sql = "SELECT * FROM models WHERE model_id = :model_id AND trim_level = :trim_level";
23         // @formatter:on
24
25         Map<String, Object> params = new HashMap<>();
26         params.put("model_id", model.toString());
27         params.put("trim_level", trim);
28
29         return jdbcTemplate.query(sql, params, new RowMapper<>() {
30
31             @Override
32             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
33                 // @formatter:off
34                 return jeep.builder()
35                     .basePrice(new BigDecimal(rs.getString("base_price")))
36                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
37                     .modelPK(rs.getLong("model_pk"))
38                     .numDoors(rs.getInt("num_doors"))
39                     .trimLevel(rs.getString("trim_level"))
40                     .wheelSize(rs.getInt("wheel_size"))
41                     .build();
32                 // @formatter:on
43             }
44         });
45     }
46 }
```

- 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.

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with a green status bar for the `FetchJeepTest` class.
- Run:** Shows the test results with a green status bar and a message: "Finished after 3.714 seconds".
- Failure Trace:** Shows the test results with a green status bar and a message: "1 elements hidden by filter".
- Code Editor:** Shows the `FetchJeepTest` class with the following code:

```
1 @Test
2 void testThatJeepIsReturnedWhenValidModelAndTrimAreSupplied() {
3     //Given: a valid model, trim and trim
4     JeepModel model = JeepModel.WRANGLER;
5     String trim = "Sport";
6     String url = String.format("%smodels-%s", getBaseUrl(), model, trim);
7
8     //When: a connection is made to the URL
9     ResponseEntity<List<Jeep>> response = getRestTemplate().exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<>());
10
11     //Then: a list of Jeeps is returned
12     assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));
13
14     //And: the actual list returned is the same as the expected list
15     List<Jeep> actual = response.getBody();
16     List<Jeep> expected = buildExpected();
17
18     assertThat(actual, equalTo(expected));
19 }
20
21 protected List<Jeep> buildExpected() {
22     List<Jeep> list = new LinkedList<>();
23
24     // @formatter:off
25     list.add(jeep.builder()
26         .modelId(JeepModel.WRANGLER)
27         .trimLevel("Sport")
28         .numDoors(4)
29         .wheelSize(17)
30         .basePrice(new BigDecimal("28475.00"))
31         .build());
32
33     list.add(jeep.builder()
34         .modelId(JeepModel.WRANGLER)
35         .trimLevel("Sport")
36         .numDoors(4)
37         .wheelSize(17)
38         .basePrice(new BigDecimal("11975.00"))
39         .build());
40
41     // @formatter:on
42
43     return list;
44 }
```
- Console:** Shows the test results with a green status bar and a message: "Running with Spring Boot v2.7.6, Spring v5.3.24".