


# Web API Design with Spring Boot Week 3 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 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. 
  - 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. 
- 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. 

### Screenshots of Code:

```

DefaultJeepSalesDao.java ×
30 import java.math.BigDecimal;
17
18 @Service
19 @Slf4j
20 @Component
21 public class DefaultJeepSalesDao implements JeepSalesDao {
22
23     @Autowired
24     private NamedParameterJdbcTemplate jdbcTemplate;
25
26     @Override
27     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
28         log.info("DAO: model={}, trim={}", model, trim);
29
30         //formatter:off
31         String sql = "
32             + "SELECT * "
33             + "FROM models "
34             + "WHERE model_id = :model_id AND trim_level = :trim_level";
35         //formatter:on
36
37         Map<String, Object> params = new HashMap<>();
38         params.put("model_id", model.toString());
39         params.put("trim_level", trim);
40
41         return jdbcTemplate.query(sql, params, new RowMapper<Jeep>() {
42
43             @Override
44             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
45                 // formatter: off
46                 return Jeep.builder()
47                     .basePrice(new BigDecimal(rs.getString("base_price")))
48                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
49                     .modelPk(rs.getLong("model_pk"))
50                     .numDoors(rs.getInt("num_doors"))
51                     .trimLevel(rs.getString("trim_level"))
52                     .wheelSize(rs.getInt("wheel_size"))
53                     .build();
54                 //formatter:on
55             }
56         });
57     }
58 }
59

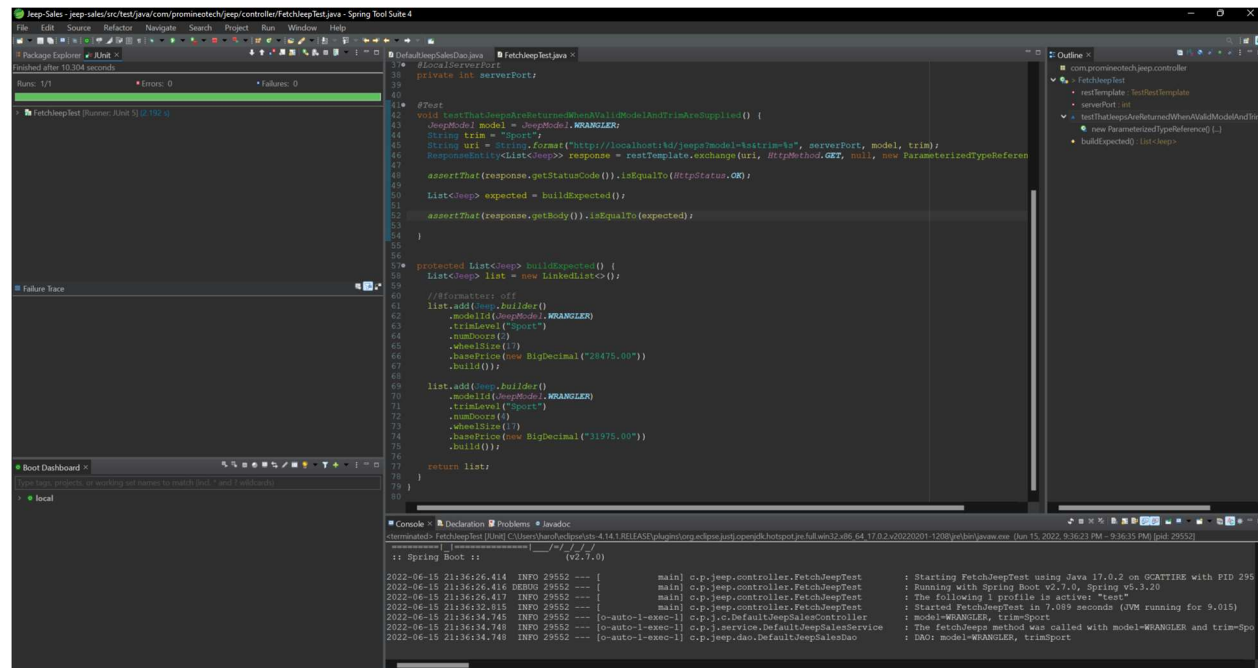
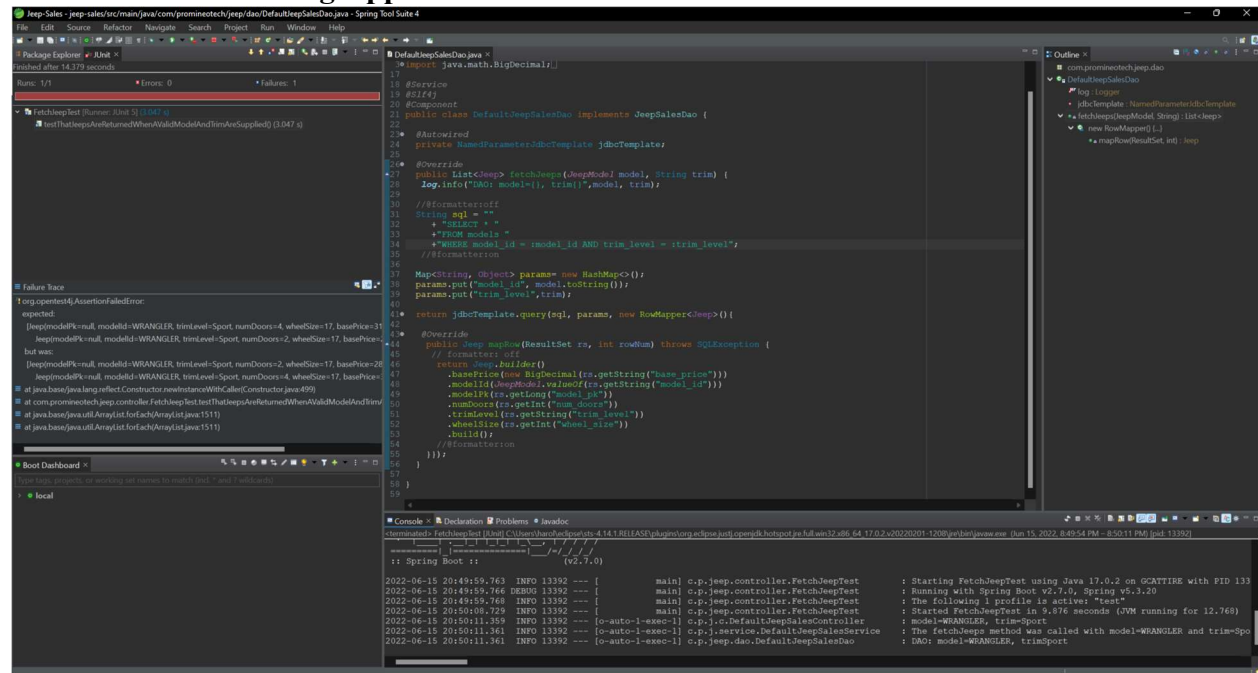
```

```

DefaultJeepSalesDao.java    FetchJeepTest.java ×
37 @LocalServerPort
38 private int serverPort;
39
40
41 @Test
42 void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
43     JeepModel model = JeepModel.WRANGLER;
44     String trim = "Sport";
45     String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
46     ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
47
48     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
49
50     List<Jeep> expected = buildExpected();
51
52     assertThat(response.getBody()).isEqualTo(expected);
53 }
54
55
56
57 protected List<Jeep> buildExpected() {
58     List<Jeep> list = new LinkedList<>();
59
60     //formatter: off
61     list.add(Jeep.builder()
62         .modelId(JeepModel.WRANGLER)
63         .trimLevel("Sport")
64         .numDoors(2)
65         .wheelSize(17)
66         .basePrice(new BigDecimal("28475.00"))
67         .build());
68
69     list.add(Jeep.builder()
70         .modelId(JeepModel.WRANGLER)
71         .trimLevel("Sport")
72         .numDoors(4)
73         .wheelSize(17)
74         .basePrice(new BigDecimal("31975.00"))
75         .build());
76
77     return list;
78 }
79 }
80

```

### Screenshots of Running Application:



### URL to GitHub Repository:

**<https://github.com/HaroldLujan/SpringBootWeek15CodingAssignment.git>**