


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.jeeep.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:

The screenshot shows the IDE with the `DefaultJeepSalesDao.java` file open. The code defines the `DefaultJeepSalesDao` class, which implements the `JeepSalesDao` interface. It includes a `NamedParameterJdbcTemplate` instance and a `fetchJeeps` method that logs the model and trim level, constructs a SQL query using `NamedParameterJdbcTemplate`, and returns a list of `JeepModel` objects.

The console output shows the test results for `FetchJeepTest`. The test is successful, and the status bar is green. The output includes the following log messages:

```

2022-10-13 13:10:59.666 INFO 8432 --- [main] c.promineotech.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.4.1 on Abyss-Laptop with PID 8432 (started by p
2022-10-13 13:10:59.667 INFO 8432 --- [main] c.promineotech.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-10-13 13:11:02.393 INFO 8432 --- [main] c.promineotech.controller.FetchJeepTest : Started FetchJeepTest in 3.067 seconds (JVM running for 3.966)
2022-10-13 13:11:03.276 INFO 8432 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : The Controller method was called: model = WRANGLER, trim =Sport
2022-10-13 13:11:03.284 INFO 8432 --- [o-auto-1-exec-1] c.p.j.s.DefaultJeepSalesService : The service method was called: model = WRANGLER, trim =Sport
2022-10-13 13:11:03.284 INFO 8432 --- [o-auto-1-exec-1] c.p.j.d.DefaultJeepSalesDao : The DAO method was called: model=WRANGLER, trim=sport
2022-10-13 13:11:03.532 INFO 8432 --- [o-auto-1-exec-2] c.p.j.c.DefaultJeepSalesController : The Controller method was called: model = WRANGLER, trim =Unknown Value
2022-10-13 13:11:03.532 INFO 8432 --- [o-auto-1-exec-2] c.p.j.s.DefaultJeepSalesService : The service method was called: model = WRANGLER, trim =Unknown Value
2022-10-13 13:11:03.532 INFO 8432 --- [o-auto-1-exec-2] c.p.j.d.DefaultJeepSalesDao : The DAO method was called: model=WRANGLER, trim=Unknown Value

```

```

1 package com.promineotech.jeeo.dao;
2
3 import java.math.BigDecimal;
4
5 @Component
6 @Slf4j
7 @Service
8 public class DefaultJeepSalesDao implements JeepSalesDao {
9
10     @Autowired
11     private NamedParameterJdbcTemplate jdbcTemplate;
12
13     @Override
14     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
15         log.info("The DAO method was called: model={}, trim={}", model, trim);
16
17         // @formatter:off
18         String sql = ""
19             + "SELECT "
20             + "FROM models "
21             + "WHERE model_id = :model_id AND trim_level = :trim_level";
22         // @formatter:on
23
24         Map<String, Object> params = new HashMap<>();
25         params.put("model_id", model.toString());
26         params.put("trim_level", trim);
27
28         return jdbcTemplate.query(sql, params, new RowMapper<>() {
29
30             @Override
31             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
32                 // @formatter:off
33                 return Jeep.builder()
34                     .basePrice(new BigDecimal(rs.getString("base_price")))
35                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
36                     .modelPK(rs.getLong("model_pk"))
37                     .numDoors(rs.getInt("num_doors"))
38                     .trimLevel(rs.getString("trim_level"))
39                     .wheelSize(rs.getInt("wheel_size"))
40                     .build();
41                 // @formatter:on
42             }
43         });
44     }
45 }

```

```

1 package com.promineotech.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 @EnableAutoConfiguration
6 @SpringBootTest(webEnvironment = MockEnvironment.RANDOM_PORT, classes = {JeepSales.class})
7 @Sql(scripts = { "classpath:flyway/migrations/V1.0_jeep_schema.sql",
8               "classpath:flyway/migrations/V1.1_jeep_data.sql" }, config = @SqlConfig(encoding = "utf-8"))
9
10 class FetchJeepTest {
11
12     @LocalServerPort
13     private int serverPort;
14
15     @Autowired
16     private TestRestTemplate restTemplate;
17
18     @Test
19     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
20
21         // Given i a valid model, trim, and URI
22         JeepModel model = JeepModel.WRANGLER;
23         String trim = "Sport";
24         String url = String.format("http://localhost:%d/jeeps/model=%s&trim=%s", serverPort, model, trim);
25
26         // When i a connection is made to the URI
27
28         Response response = restTemplate.exchange(url, RequestMethod.GET, null,
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Screenshots of Running Application:

Test code and console-


```

105 // Given : a valid model, trim, and URI
106
107 String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
108
109 // When : a connection is made to the URI
110
111 ResponseEntity<Map<String, Object>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
112     new ParameterizedTypeReference<>() {
113     });
114
115 // Then : a not found (404) status code is returned
116 assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
117
118 // And an error message is returned
119 Map<String, Object> error = response.getBody();
120
121 assertThatErrorMessageValid(error, HttpStatus.BAD_REQUEST);
122
123 }
124
125 static Stream<Arguments> parametersForInvalidInput () {
126     // @formatter:off
127     return Stream.of(
128         arguments("WRANGLER", "fjldfj8*89", "trim contains non-alpha-numeric characters"),
129         arguments("WRANGLER", "C".repeat(Constants.TRIM_MAX_LENGTH + 1), "Trim length too long"),
130         arguments("INVALID", "Sport", "Model is not an enum value"));
131     // @formatter:on
132 }
133
134 protected void assertThatErrorMessageValid(Map<String, Object> error, HttpStatus status) {
135     // @formatter:off
136     assertThat(error)
137         .containsKey("message")
138         .containsEntry("status code", status.value())
139         .containsEntry("uri", "/" + jeeps)
140         .containsKey("timestamp")
141         .containsEntry("reason", status.getReasonPhrase());
142     // @formatter:on
143 }
144
145 // video puts this buildExpected method in JeepControllerSupport class FetchJeepTestSupport extends BaseTest
146 protected List<Jeep> buildExpected() {
147     List<Jeep> list = new LinkedList<>();
148     // @formatter:off

```

```

44 // video puts this buildExpected method in JeepControllerSupport class FetchJeepTestSupport extends BaseTest
45 protected List<Jeep> buildExpected() {
46     List<Jeep> list = new LinkedList<>();
47     // @formatter:off
48
49     list.add(Jeep.builder()
50         .modelId(JeepModel.WRANGLER)
51         .trimLevel("Sport")
52         .numDoors(4)
53         .wheelSize(17)
54         .basePrice(new BigDecimal("31975.00"))
55         .build());
56
57     list.add(Jeep.builder()
58         .modelId(JeepModel.WRANGLER)
59         .trimLevel("Sport")
60         .numDoors(2)
61         .wheelSize(17)
62         .basePrice(new BigDecimal("28475.00"))
63         .build());
64
65     // @formatter:on
66     Collections.sort(list);
67     return list;
68 }
69
70 }
71

```

URL to GitHub Repository: <https://github.com/Hughes405/Week-15-Coding-Assignment.git>