


Web API Design with Spring Boot Week 2 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 project you started last week, use Lombok to add an info-level logging statement in the controller implementation method that logs the parameters that were input to the method. Remember to add the `@Slf4j` annotation to the class.
- 2) Start the application (not an integration test). Use a browser to navigate to the application passing the parameters required for your selected operation. (A browser, used in this manner, sends an HTTP GET request to the server.) Produce a screenshot showing the browser navigation bar and the log statement that is in the IDE console showing that the controller

method was reached (as in the video).

```
jeep-sales - JeepSales (Spring Boot App) [pid: 20849]
2022-09-30 13:58:34.963 INFO 20849 --- [ restartedMain] com.promineotech.jeepp.JeepSales : Starting JeepSales using Java 17.0.3.1 on Shawns
2022-09-30 13:58:34.964 INFO 20849 --- [ restartedMain] com.promineotech.jeepp.JeepSales : No active profile set, falling back to 1 default
2022-09-30 13:58:35.004 INFO 20849 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.d
2022-09-30 13:58:35.005 INFO 20849 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider sett
2022-09-30 13:58:35.850 INFO 20849 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-09-30 13:58:35.861 INFO 20849 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-30 13:58:35.861 INFO 20849 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-30 13:58:35.917 INFO 20849 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationConte
2022-09-30 13:58:35.917 INFO 20849 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization compl
2022-09-30 13:58:36.428 INFO 20849 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-09-30 13:58:36.455 INFO 20849 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with cont
2022-09-30 13:58:36.465 INFO 20849 --- [ restartedMain] com.promineotech.jeepp.JeepSales : Started JeepSales in 1.757 seconds (JVM running
2022-09-30 14:00:39.972 INFO 20849 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatc
2022-09-30 14:00:39.972 INFO 20849 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-09-30 14:00:39.973 INFO 20849 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-09-30 14:00:40.387 INFO 20849 --- [nio-8080-exec-9] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 74 ms
```

Jeep Sales Service

/v3/api-docs

Servers

http://localhost:8080 - Local server.

basic-jeep-sales-controller

GET /jeeps Returns a list of Jeeps

Schemas

Jeep >

- 3) With the application still running, use the browser to navigate to the OpenAPI documentation. Use the OpenAPI documentation to send a GET request to the server with a valid model and trim level. (You can get the model and trim from the provided data.sql file.) Produce a screenshot showing the curl command, the request URL, and the response headers.

Curl

```
curl -X 'GET' \
  'http://localhost:8080/jeeps?model=WRANGLER&trim=Sport' \
  -H 'accept: application/json'
```

Request URL

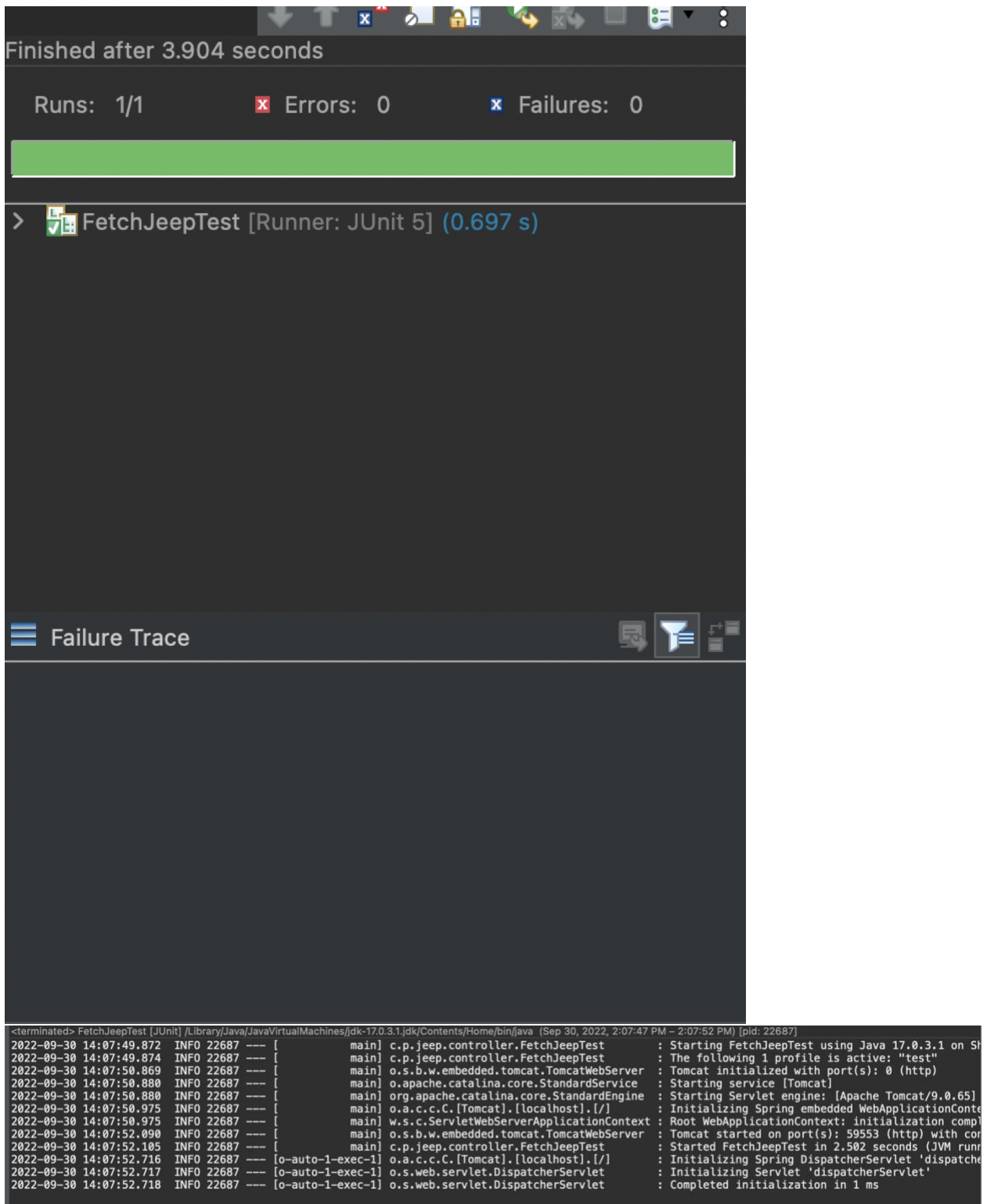
```
http://localhost:8080/jeeps?model=WRANGLER&trim=Sport
```

Server response

Code	Details
200	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Fri, 30 Sep 2022 18:03:33 GMT keep-alive: timeout=60</pre>

Responses

- 4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar.




```
<terminated> FetchJeepTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-17.0.3.1.jdk/Contents/Home/bin/java (Sep 30, 2022, 2:07:47 PM - 2:07:52 PM) [pid: 22687]
2022-09-30 14:07:49.872 INFO 22687 --- [main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.3.1 on Si
2022-09-30 14:07:49.874 INFO 22687 --- [main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-09-30 14:07:50.869 INFO 22687 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2022-09-30 14:07:50.880 INFO 22687 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-30 14:07:50.880 INFO 22687 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-30 14:07:50.975 INFO 22687 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-09-30 14:07:50.975 INFO 22687 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2022-09-30 14:07:52.090 INFO 22687 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 59553 (http) with cor
2022-09-30 14:07:52.105 INFO 22687 --- [main] c.p.jeep.controller.FetchJeepTest : Started FetchJeepTest in 2.502 seconds (JVM runn
2022-09-30 14:07:52.716 INFO 22687 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcher
2022-09-30 14:07:52.717 INFO 22687 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-09-30 14:07:52.718 INFO 22687 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

- 5) Add a method to the test to return a list of expected Jeep (model) objects based on the model and trim level you selected. You can get the expected list of Jeeps from the file `src/test/resources/flyway/migrations/V1.1__Jeep_Data.sql`. So, for example, using the model Wrangler and trim level "Sport", the query should return two rows:

	Row 1	Row 2
Model ID	WRANGLER	WRANGLER
Trim Level	Sport	Sport
Num Doors	2	4
Wheel Size	17	17
Base Price	\$28,475.00	\$31,975.00

The method should be named `buildExpected()`, and it should return a `List` of `Jeep`. The video put this method into a support superclass but you can include it in the main test class if you want.

- 6) Write an AssertJ assertion in the test to assert that the actual list of jeeps returned by the server is the same as the expected list. Run the test. Produce a screenshot showing...
 - a) The test with the assertion.
 - b) The JUnit status bar (should be red).
 - c) The method returning the expected list of Jeeps. 

```
@Test
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {

    // Given: a valid model, trim and URI

    JeepModel model = JeepModel.WRANGLER;
    String trim = "Sport";
    String uri =
        String.format
        ("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);

    // When: a connection is made to the URI
    ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri,
        HttpMethod.GET, null, new ParameterizedTypeReference<>() {});

    // Then: a success (OK - 200) status code is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);

    // And: the actual list is the same as the expected list

    List<Jeep> expected = buildExpected();
    assertThat(response.getBody()).isEqualTo(expected);
}

}
```

Finished after 3.241 seconds

Runs: 1/1 Errors: 0 Failures: 1

FetchJeepTest [Runner: JUnit 5] (0.497 s)

testThatJeepsAreReturnedWhenAValidModelAndTrimAreS

Failure Trace


org.opentest4j.AssertionFailedError:
expected:
[Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numD
Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, num
but was:
null
at java.base/java.lang.reflect.Constructor.newInstanceWithCalle
at com.promineotech.jeep.controller.FetchJeepTest.testThatJee
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

<terminated> FetchJeepTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-17.0.3.1.jdk/Contents/Home/bin/java (Sep 30, 2022, 6:11:36 PM) [pid: 25422]

Spring Boot (v2.7.4)

```
2022-09-30 18:11:33.389 INFO 25422 --- [main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.3.1 on SH
2022-09-30 18:11:33.390 DEBUG 25422 --- [main] c.p.jeep.controller.FetchJeepTest : Running with Spring Boot v2.7.4, Spring v5.3.23
2022-09-30 18:11:33.391 INFO 25422 --- [main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-09-30 18:11:35.490 INFO 25422 --- [main] c.p.jeep.controller.FetchJeepTest : Started FetchJeepTest in 2.389 seconds (JVM runn
2022-09-30 18:11:35.968 DEBUG 25422 --- [o-auto-1-exec-1] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport,
[Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=17, basePrice=28475.00), Jeep(modelPK=null, modelId=WRANGLER, trimLevel=
```

- 7) Add a service layer in your application as shown in the videos:
- Add a package named `com.promineotech.jeep.service`.
 - In the new package, create an interface named `JeepSalesService`.

- c) In the same package (service), create a class named DefaultJeepSalesService that implements the JeepSalesService interface. Add the class-level annotation, @Service.
- d) Inject the service interface into DefaultJeepSalesController using the @Autowired annotation. The instance variable should be private, and the variable should be named jeepSalesService.
- e) Define the fetchJeeps method in the interface. Implement the method in the service class. Call the method from the controller (make sure the controller returns the list of Jeeps returned by the service method). The method signature looks like this:
`List<Jeep> fetchJeeps(JeepModel model, String trim);`
- f) Add a Lombok info-level log statement in the service implementation showing that the service was called. Print the parameters passed to the method. Let the method return null for now.
- g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar. 

```
1 package com.promineotech.jeep.service;
2
3 import java.util.List;
4
5 import org.springframework.stereotype.Service;
6
7 import com.promineotech.jeep.entity.Jeep;
8 import com.promineotech.jeep.entity.JeepModel;
9
10 import lombok.extern.slf4j.Slf4j;
11
12 @Service
13 @Slf4j
14 public class DefaultJeepSalesService implements JeepSalesService {
15
16     @Override
17     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
18         log.info("The fetchJeeps method was called with model={} and trim={}", model, trim);
19         return null;
20     }
21 }
22
23
```



```

Terminated> FetchJeepTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-17.0.3.1.jdk/Contents/Home/bin/java (Sep 30, 2022, 6:36:34 PM - 6:36:41 PM) [pid: 25739]
:: Spring Boot :: (v2.7.4)

2022-09-30 18:36:37.998 INFO 25739 --- [main] c.p.jeeptest.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.3.1 on Sh
2022-09-30 18:36:37.999 DEBUG 25739 --- [main] c.p.jeeptest.controller.FetchJeepTest : Running with Spring Boot v2.7.4, Spring v5.3.23
2022-09-30 18:36:38.000 INFO 25739 --- [main] c.p.jeeptest.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-09-30 18:36:40.421 INFO 25739 --- [main] c.p.jeeptest.controller.FetchJeepTest : Started FetchJeepTest in 2.832 seconds (JVM runn
2022-09-30 18:36:41.178 DEBUG 25739 --- [o-auto-1-exec-1] c.p.jeeptest.service.DefaultJeepSalesService : model=WRANGLER, trim=Sport,
2022-09-30 18:36:41.179 INFO 25739 --- [o-auto-1-exec-1] c.p.jeeptest.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRAN

Finished after 4.371 seconds

Runs: 1/1      x Errors: 0      x Failures: 1

FetchJeepTest [Runner: JUnit 5] (0.821 s)
  testThatJeepsAreReturnedWhenAValidModelAndTrimAreS

Failure Trace
! org.opentest4j.AssertionFailedError:
expected:
[Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, numD
  Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Sport, num
but was:
null
at java.base/java.lang.reflect.Constructor.newInstanceWithCalle
at com.promineotech.jeeptest.controller.FetchJeepTest.testThatJee
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)


```

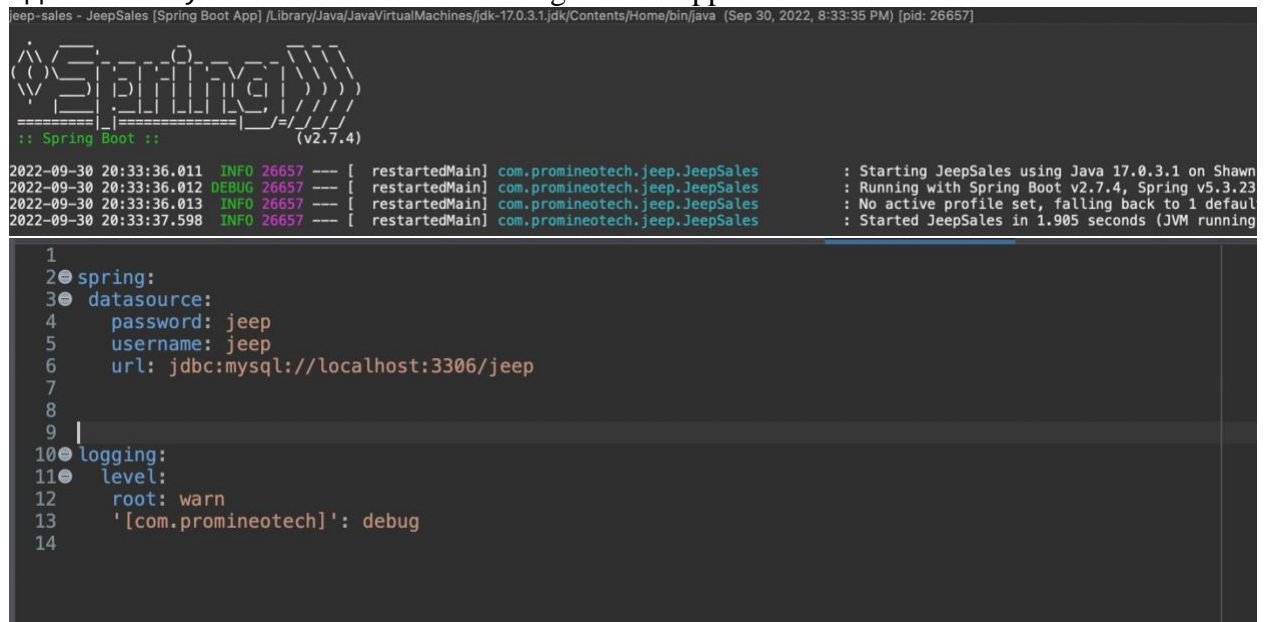
numbers for either dependency because the version is included in the Spring Boot Starter Parent.

- 9) Create `application.yaml` in `src/main/resources`. Add the `spring.datasource.url`, `spring.datasource.username`, and `spring.datasource.password` properties to `application.yaml`. The url should be the same as shown in the video (`jdbc:mysql://localhost:3306/jeep`). The password and username should match your setup. If you created the database under your root user, the username is "root", and the password is the root user password. If you created a "jeep" user or other user, use the correct username and password.

Be careful with the indentation! YAML allows hierarchical configuration but it reads the hierarchy based on the indentation level. The keyword "spring" MUST start in the first column. It should look similar to this when done:

```
spring:
  datasource:
    username: username
    password: password
    url: jdbc:mysql://localhost:3306/jeep
```

- 10) Start the application (the real application, not the test). Produce a screenshot that shows `application.yaml` and the console showing that the application has started with no errors. 



```
jeep-sales - JeepSales (Spring Boot App) /Library/Java/JavaVirtualMachines/jdk-17.0.3.1.jdk/Contents/Home/bin/java (Sep 30, 2022, 8:33:35 PM) [pid: 26657]

:: Spring Boot ::
(v2.7.4)


2022-09-30 20:33:36.011 INFO 26657 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 17.0.3.1 on Shawn
2022-09-30 20:33:36.012 DEBUG 26657 --- [ restartedMain] com.promineotech.jeep.JeepSales : Running with Spring Boot v2.7.4, Spring v5.3.23
2022-09-30 20:33:36.013 INFO 26657 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default
2022-09-30 20:33:37.598 INFO 26657 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 1.905 seconds (JVM running)

1
2● spring:
3●  datasource:
4    password: jeep
5    username: jeep
6    url: jdbc:mysql://localhost:3306/jeep
7
8
9 |
10● logging:
11●  level:
12    root: warn
13    '[com.promineotech]': debug
14
```

- 11) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".
- 12) Create `application-test.yaml` in `src/test/resources`. Add the setting `spring.datasource.url` that points to the H2 database. It should look like this:


```
spring:
  datasource:
    url: jdbc:h2:mem:jeep
```

You do not need to set the username and password because the in-memory H2 database does not require them.

Produce a screenshot showing application-test.yaml. 

```
1 spring:
2   datasource:
3     url: jdbc:h2:mem:jeep;mode=MYSQL
4
5 logging:
6   level:
7     root: warn
8     '[com.promineotech]': debug
9
10
```

Screenshots of Code:

Screenshots of Running Application:

URL to GitHub Repository: