

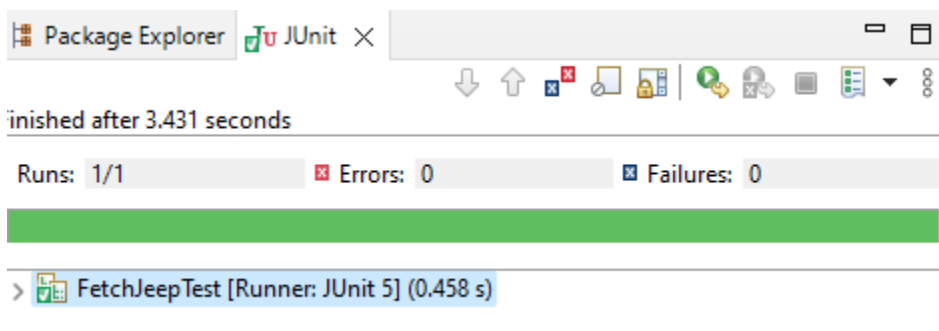
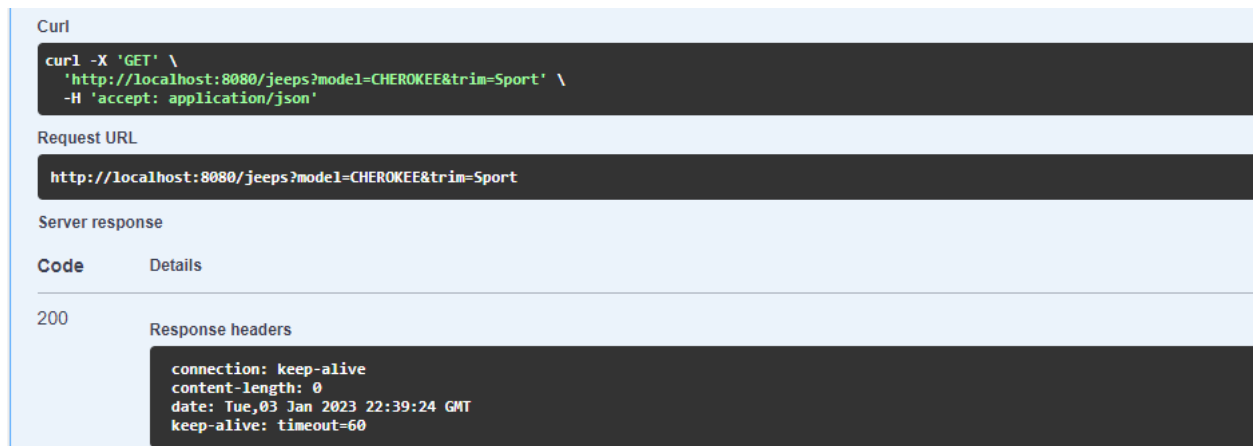
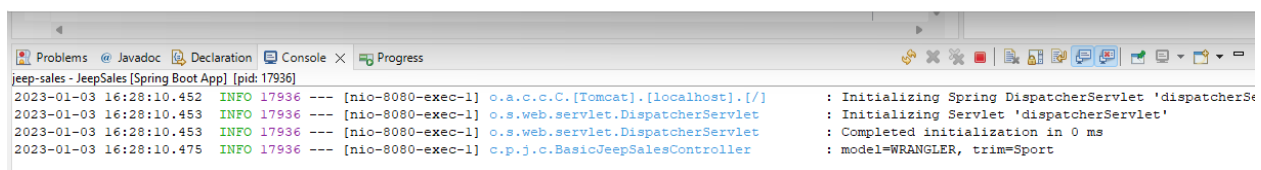
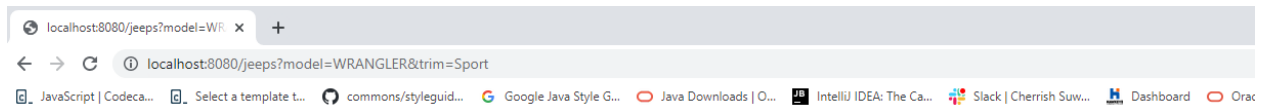
# Web API Design with Spring Boot Week 14 Coding Assignment

Points possible: 75

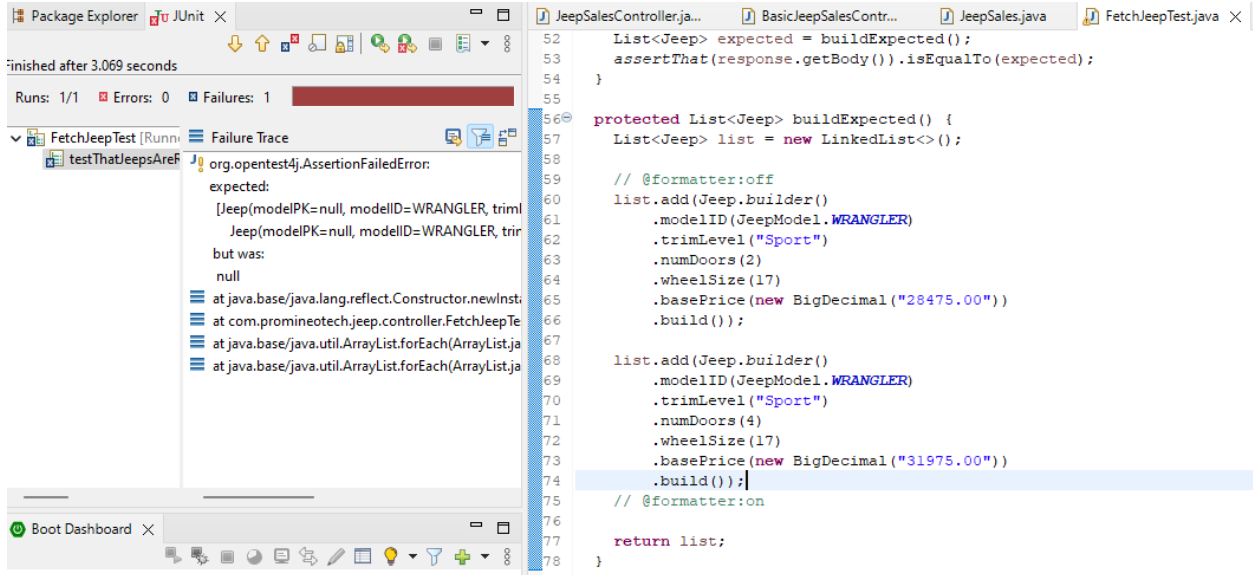
URL to GitHub Repository: <https://github.com/ldlau/week-14-coding-assignment.git>

URL to Public Link of your Video: <https://youtu.be/sAJBISWA9r4>

## SCREENSHOTS:

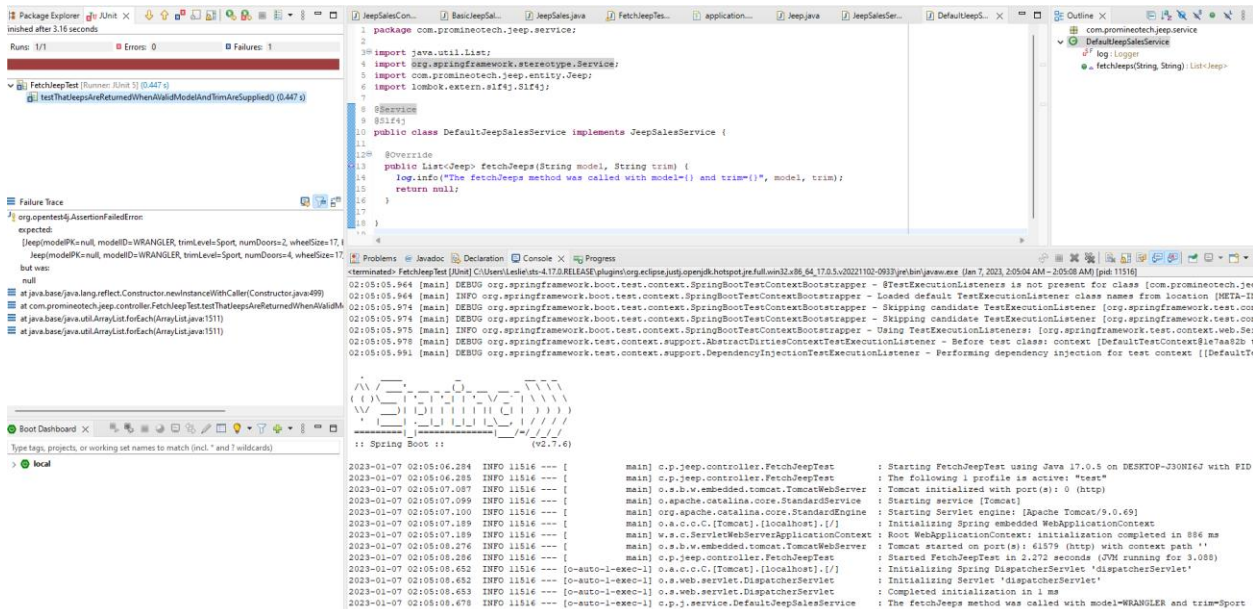


# Web API Design with Spring Boot Week 14 Coding Assignment



The screenshot shows an IDE with the following components:

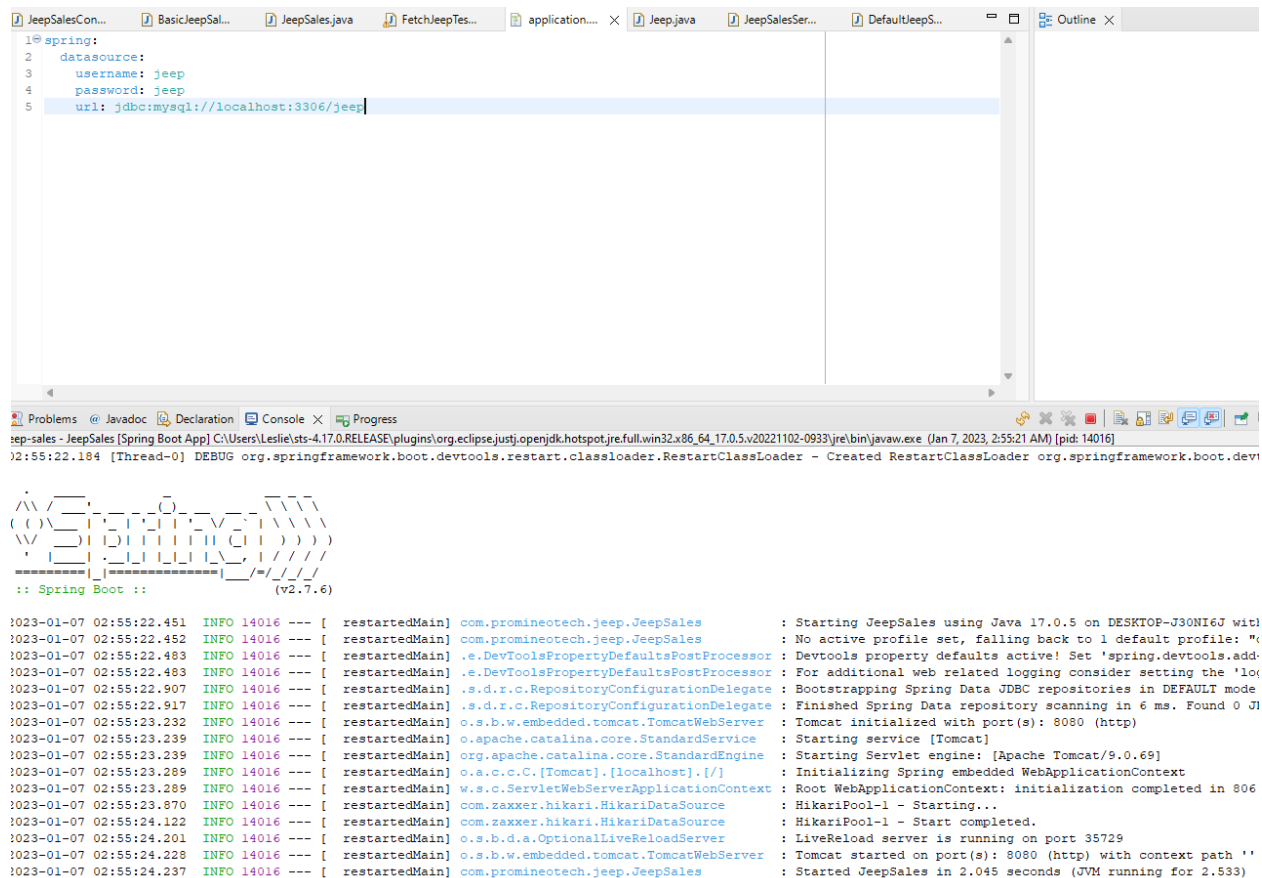
- Package Explorer:** Shows the project structure with packages like `com.promineotech.jee`.
- JUnit Console:** Displays the test results for `FetchJeepTest`. It shows a failure in the `testThatJeepsAreFetchedWhenValidModelAndTrimAreSupplied()` test. The failure message is: `org.opentest4j.AssertionFailedError: expected: [Jeep(modelPK=null, modelID=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=17, basePrice=28475.00)] but was: null`. The stack trace points to the `FetchJeepController` and the `FetchJeepTest` class.
- Failure Trace:** Provides a detailed view of the assertion failure, showing the expected and actual values.
- Code Editor:** Displays the `FetchJeepController.java` file. The `buildExpected()` method is highlighted, showing the construction of the expected `Jeep` objects. The method uses `LinkedList` to store the expected results and includes comments for formatting.



The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `com.promineotech.jee`.
- JUnit Console:** Displays the test results for `FetchJeepTest`. It shows a failure in the `testThatJeepsAreFetchedWhenValidModelAndTrimAreSupplied()` test. The failure message is: `org.opentest4j.AssertionFailedError: expected: [Jeep(modelPK=null, modelID=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=17, basePrice=28475.00)] but was: null`. The stack trace points to the `FetchJeepController` and the `FetchJeepTest` class.
- Failure Trace:** Provides a detailed view of the assertion failure, showing the expected and actual values.
- Code Editor:** Displays the `DefaultJeepSalesService.java` file. The `fetchJeeps` method is highlighted, showing the construction of the expected `Jeep` objects. The method uses `LinkedList` to store the expected results and includes comments for formatting.
- Console:** Displays the application logs, showing the startup of the application and the execution of the test. The logs include the following information:
  - Starting FetchJeepTest using Java 17.0.5 on DESKTOP-230N16J with PID 11516
  - The following profile is active: "test"
  - Tomcat initialized with port(s): 8080 (http)
  - Starting service [Tomcat]
  - Starting Servlet engine: [Apache Tomcat/9.0.69]
  - Tomcat started on port(s): 8080 (http) with context path ''
  - Started FetchJeepTest in 3.272 seconds (JVM running for 3.058)
  - Initializing Spring DispatcherServlet 'dispatcherServlet'
  - Initializing Servlet 'dispatcherServlet'
  - Completed initialization in 1 ms
  - The fetchJeeps method was called with model=WRANGLER and trim=Sport

# Web API Design with Spring Boot Week 14 Coding Assignment

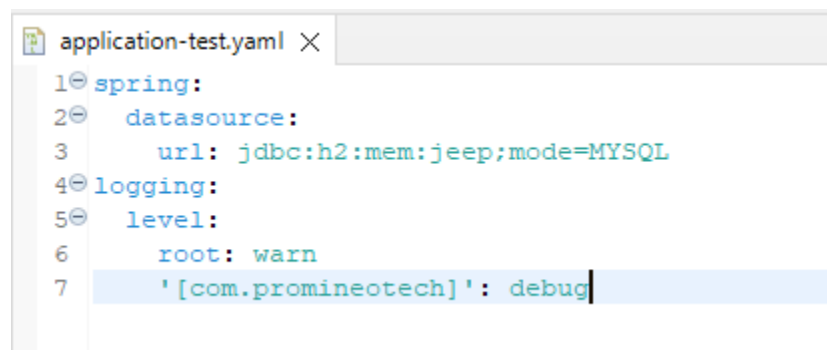


The screenshot shows an IDE with a project named 'JeepSales'. The main editor displays a configuration file with the following content:

```
1 @spring:
2   datasource:
3     username: jeep
4     password: jeep
5     url: jdbc:mysql://localhost:3306/jeep
```

The console output shows the application starting successfully. The output is as follows:

```
1023-01-07 02:55:22.451 INFO 14016 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 17.0.5 on DESKTOP-J30NI6J with
1023-01-07 02:55:22.452 INFO 14016 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default profile: "(
1023-01-07 02:55:22.483 INFO 14016 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : DevTools property defaults active! Set 'spring.devtools.add
1023-01-07 02:55:22.483 INFO 14016 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'lo
1023-01-07 02:55:22.907 INFO 14016 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode
1023-01-07 02:55:22.917 INFO 14016 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 6 ms. Found 0 J
1023-01-07 02:55:23.232 INFO 14016 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
1023-01-07 02:55:23.239 INFO 14016 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
1023-01-07 02:55:23.239 INFO 14016 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.69]
1023-01-07 02:55:23.289 INFO 14016 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
1023-01-07 02:55:23.289 INFO 14016 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 806
1023-01-07 02:55:23.870 INFO 14016 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
1023-01-07 02:55:24.122 INFO 14016 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
1023-01-07 02:55:24.201 INFO 14016 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
1023-01-07 02:55:24.228 INFO 14016 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
1023-01-07 02:55:24.237 INFO 14016 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 2.045 seconds (JVM running for 2.533)
```



The screenshot shows an IDE with a project named 'JeepSales'. The main editor displays a configuration file with the following content:

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


## 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.,

# Web API Design with Spring Boot Week 14 Coding Assignment

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.




---

**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>

# Web API Design with Spring Boot Week 14 Coding Assignment

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


## Web API Design with Spring Boot Week 14 Coding Assignment

<b>Model ID</b>	WRANGLER	WRANGLER
<b>Trim Level</b>	Sport	Sport
<b>Num Doors</b>	2	4
<b>Wheel Size</b>	17	17
<b>Base Price</b>	\$28,475.00	\$31,975.00

6)

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.

7) 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. 

8) Add a service layer in your application as shown in the videos:


- a) Add a package named `com.promineotech.jeep.service`.
- b) 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.

## Web API Design with Spring Boot Week 14 Coding Assignment

- g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar. 
- 9) Add the database dependencies described in the video to the POM file (MySQL driver and Spring Boot Starter JDBC). To find them, navigate to <https://mvnrepository.com/>. Search for `mysql-connector-j` and `spring-boot-starter-jdbc`. In the POM file you don't need version numbers for either dependency because the version is included in the Spring Boot Starter Parent.
- 10) 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
```

- 11) 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. 
- 12) 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".
- 13) 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;mode=MYSQL
```

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

## Web API Design with Spring Boot Week 14 Coding Assignment

Produce a screenshot showing `application-test.yaml`. 