

# Web API Design with Spring Boot Week 14 Coding Assignment

Points possible: 75


URL to GitHub Repository: <https://github.com/Bwcash/Web-API-Spring-Boot.git>

URL to Public Link of your Video: <https://youtu.be/fslux6a-exM>


---

## 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 14 Coding Assignment

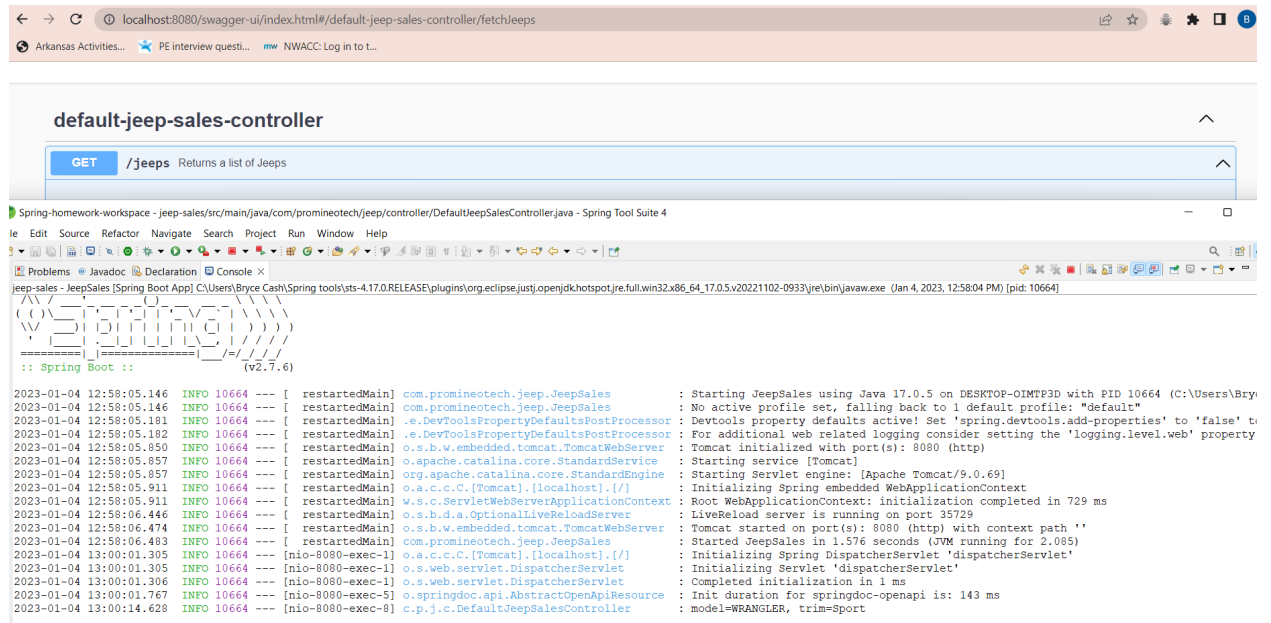
**Here's a friendly tip:** as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon:  You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

## Project Resources:

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

## Coding Steps:


- 1) In the 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). 

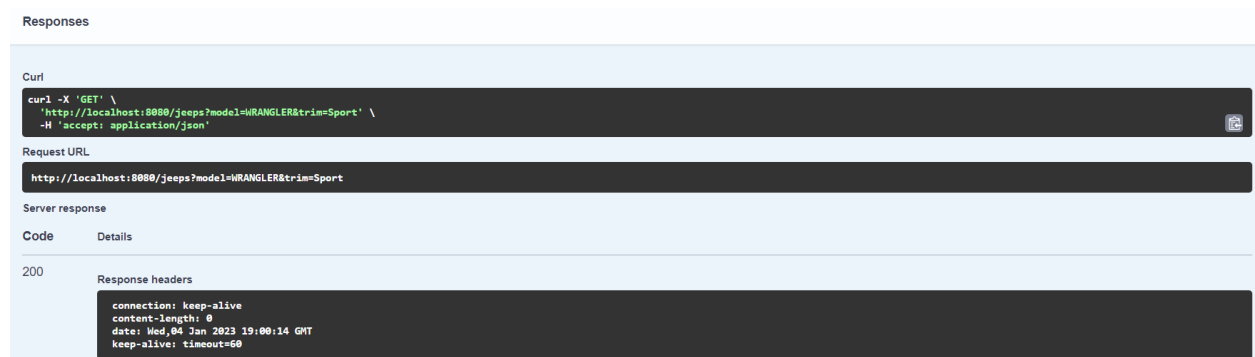



```
default-jeep-sales-controller
GET /jeeps Returns a list of Jeeps

Spring-homework-workspace - jeep-sales/src/main/java/com/promineotech/jeep/controller/DefaultJeepSalesController.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
jeep-sales - JeepSales [Spring Boot App] C:\Users\Bryce Cash\Spring tools\sts-4.17.0.RELEASE\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\java.exe (Jan 4, 2023, 12:58:04 PM) [pid: 10664]
:: Spring Boot ::
2023-01-04 12:58:05.146 INFO 10664 --- [ restartedMain] com.promineotech.jee... : Starting JeepSales using Java 17.0.5 on DESKTOP-OIMTF3D with PID 10664 (C:\Users\Bryce Cash\Spring tools\sts-4.17.0.RELEASE\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\java.exe)
2023-01-04 12:58:05.146 INFO 10664 --- [ restartedMain] com.promineotech.jee... : No active profile set, falling back to 1 default profile: "default"
2023-01-04 12:58:05.181 INFO 10664 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to
2023-01-04 12:58:05.182 INFO 10664 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property
2023-01-04 12:58:05.850 INFO 10664 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-01-04 12:58:05.857 INFO 10664 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2023-01-04 12:58:05.857 INFO 10664 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.69]
2023-01-04 12:58:05.911 INFO 10664 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-01-04 12:58:05.911 INFO 10664 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 729 ms
2023-01-04 12:58:06.446 INFO 10664 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : LiveReload server is running on port 35729
2023-01-04 12:58:06.474 INFO 10664 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-01-04 12:58:06.483 INFO 10664 --- [ restartedMain] com.promineotech.jee... : Started JeepSales in 1.576 seconds (JVM running for 2.085)
2023-01-04 13:00:01.305 INFO 10664 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-01-04 13:00:01.305 INFO 10664 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-01-04 13:00:01.306 INFO 10664 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2023-01-04 13:00:01.767 INFO 10664 --- [nio-8080-exec-5] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 143 ms
2023-01-04 13:00:14.628 INFO 10664 --- [nio-8080-exec-8] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
```

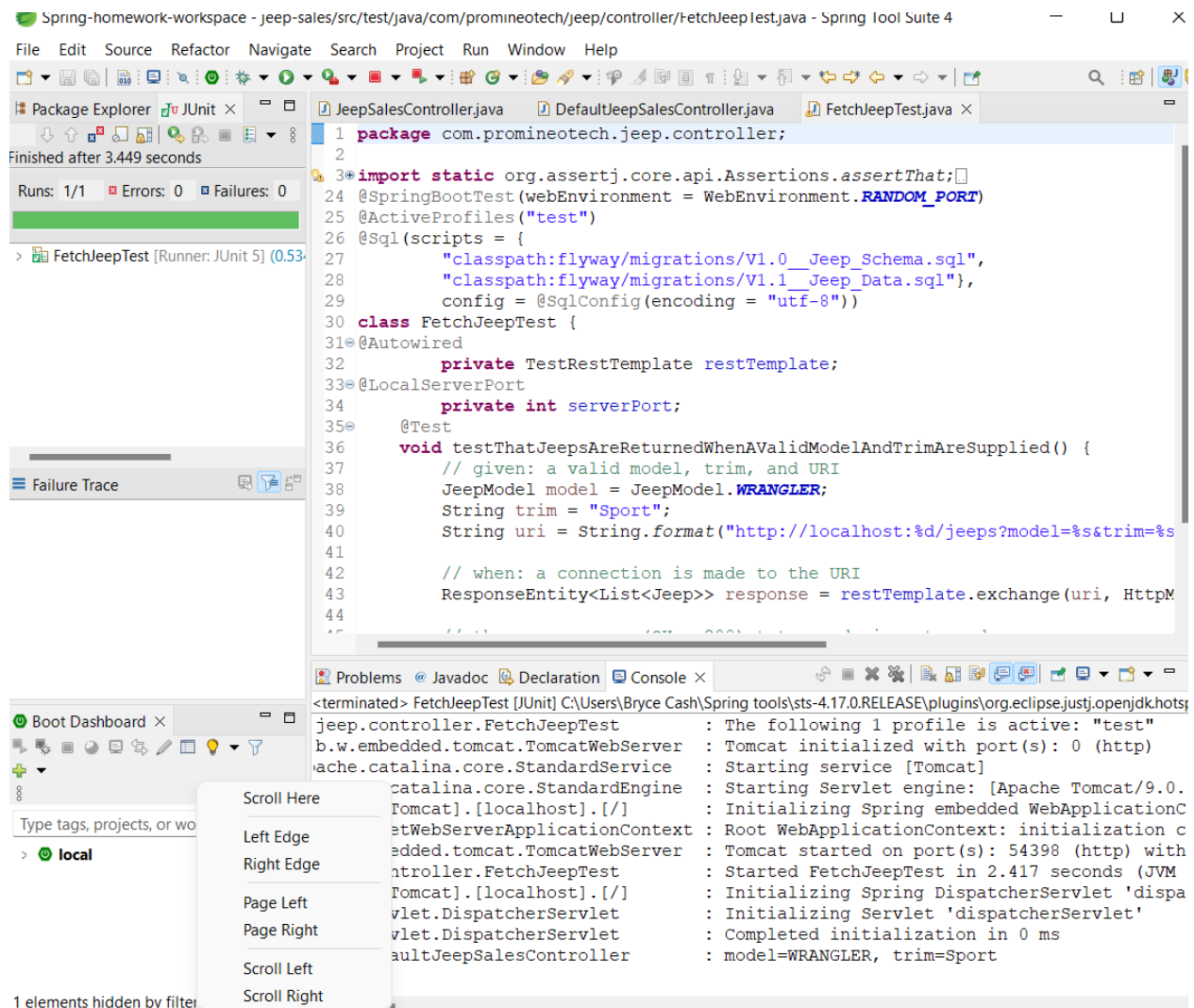
## Web API Design with Spring Boot Week 14 Coding Assignment

- 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. 

# Web API Design with Spring Boot Week 14 Coding Assignment



- 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

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

1)


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.

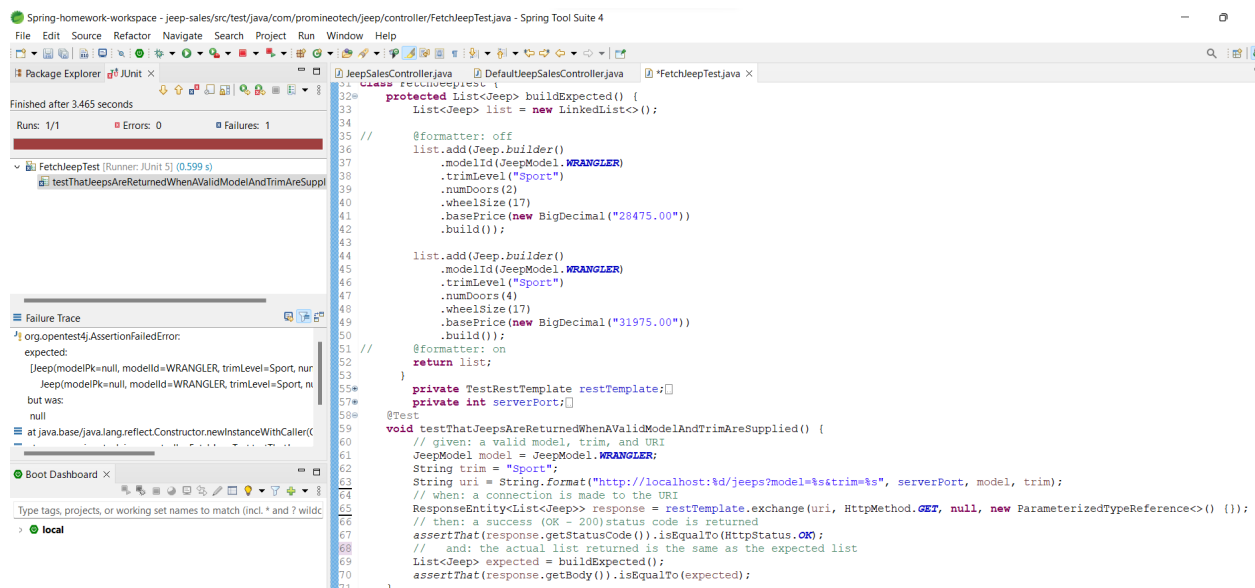
2)

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. 



The screenshot shows an IDE window titled "Spring-homework-workspace - jeep-sales/src/test/java/com/promineotech/jeep/controller/JeepControllerTest.java - Spring Tool Suite 4". The main editor displays the `FetchJeepTest` class with the `buildExpected()` method and a test method `testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()`. The test method uses `assertThat(response.getBody()).isEqualTo(expected);`. The JUnit status bar at the bottom left shows "Finished after 3.465 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 1". The failure trace on the left indicates an `org.opentest4j.AssertionFailedError` with the message "expected: [Jeep(modelPk=null, modelId=WRANGLER, trimLevel=Sport, numDoors=2, wheelSize=17, basePrice=28475.00)] but was: [Jeep(modelPk=null, modelId=WRANGLER, trimLevel=Sport, numDoors=4, wheelSize=17, basePrice=31975.00)]".

# Web API Design with Spring Boot Week 14 Coding Assignment

- 3) Add a service layer in your application as shown in the videos:
  - a) Add a package named `com.promineotech.jeepp.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.
  - g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar.

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the package structure with `com.promineotech.jeepp.controller` and `com.promineotech.jeepp.service`.
- Source Editor:** Displays the `DefaultJeepSalesController.java` file. It shows the `@RestController` annotation, the `@Autowired` injection of `JeepSalesService`, and the `fetchJeeps` method implementation. The method calls `jeepSalesService.fetchJeeps(model, trim)` and returns the result.
- Console:** Shows the output of the application. It includes logs for starting the application, initializing the web application context, and starting the Tomcat server. The final log line shows the `fetchJeeps` method being called with `model=WRANGLER` and `trim=Sport`.
- Failure Trace:** Shows the error message: `org.opentest4j.AssertionFailedError: expected: [Jeep(modelPk=null, modelId=WRANGLER, trim=Sport)] but was: null`. This indicates that the test expects a list of Jeeps, but the service method returned null.

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

- 4) 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.
- 5) 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.

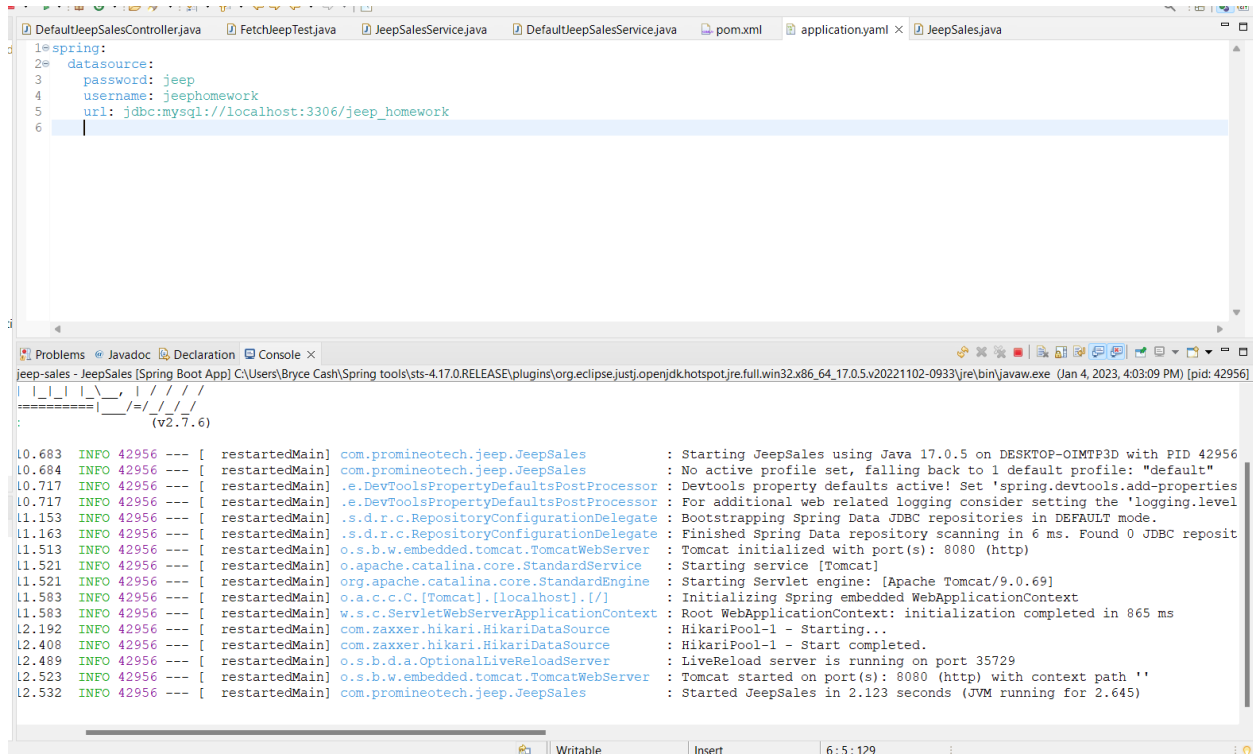
Be careful with the indentation! YAML allows hierarchical configuration but it reads the hierarchy based on the indentation level. The keyword "spring" MUST start in the first column. It should look similar to this when done:

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

- 6) 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.



# Web API Design with Spring Boot Week 14 Coding Assignment



The screenshot shows an IDE with several tabs open: DefaultJeepSalesController.java, FetchJeepTest.java, JeepSalesService.java, DefaultJeepSalesService.java, pom.xml, application.yml, and JeepSales.java. The application.yml file is selected, showing the following configuration:

```
1= spring:
2=   datasource:
3=     password: jeep
4=     username: jeephomework
5=     url: jdbc:mysql://localhost:3306/jeep_homework
6=
```

The console output shows the application starting successfully. Key messages include:

- Starting JeepSales using Java 17.0.5 on DESKTOP-OIMTF3D with PID 42956
- No active profile set, falling back to 1 default profile: "default"
- Devtools property defaults active! Set 'spring.devtools.add-properties' for additional web related logging consider setting the 'logging.level' Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
- Finished Spring Data repository scanning in 6 ms. Found 0 JDBC repositories
- Tomcat initialized with port(s): 8080 (http)
- Starting service [Tomcat]
- Starting Servlet engine: [Apache Tomcat/9.0.69]
- Initializing Spring embedded WebApplicationContext
- Root WebApplicationContext: initialization completed in 865 ms
- HikariPool-1 - Starting...
- HikariPool-1 - Start completed.
- LiveReload server is running on port 35729
- Tomcat started on port(s): 8080 (http) with context path ''
- Started JeepSales in 2.123 seconds (JVM running for 2.645)

7) 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".


8) Create application-test.yml 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.

Produce a screenshot showing application-test.yml. 



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



The screenshot shows an IDE window with three tabs: 'application-test.y...', 'DefaultJeepSalesC...', and 'FetchJeepTest.java'. The 'application-test.y...' tab is active, displaying a YAML configuration for a Spring Boot application. The configuration is as follows:

```
1 spring:
2   datasource:
3     url: jdbc:h2:mem:jeep;mode=MYSQL
4
```