

# Web API Design with Spring Boot Week 14 Coding Assignment

**Points possible: 75**


**URL to GitHub Repository: <https://github.com/sraznoff/JeepSalesAPI.git>**

**URL to Public Link of your Video: <https://youtu.be/4po2c4LxMJM>**


---

## 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.
-

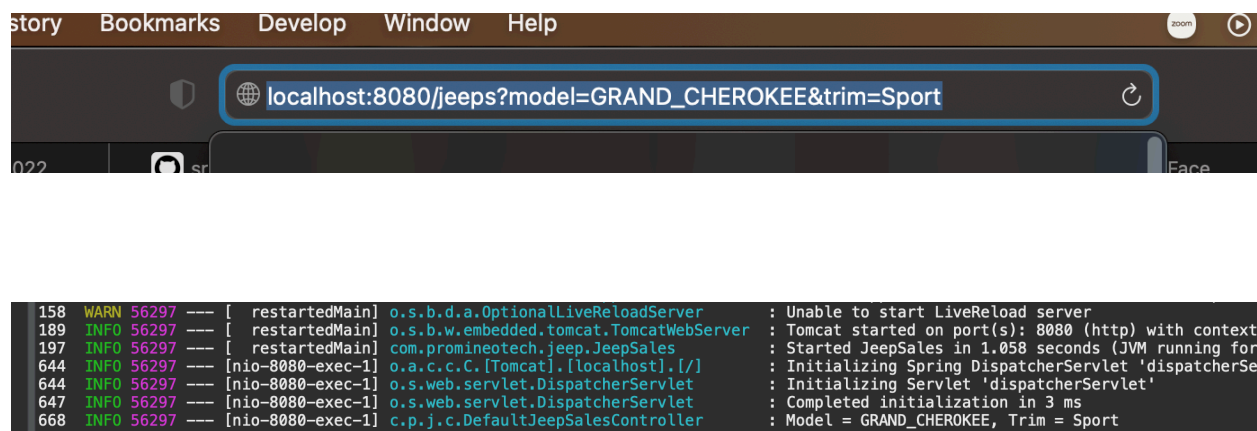
# Web API Design with Spring Boot Week 14 Coding Assignment

**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>

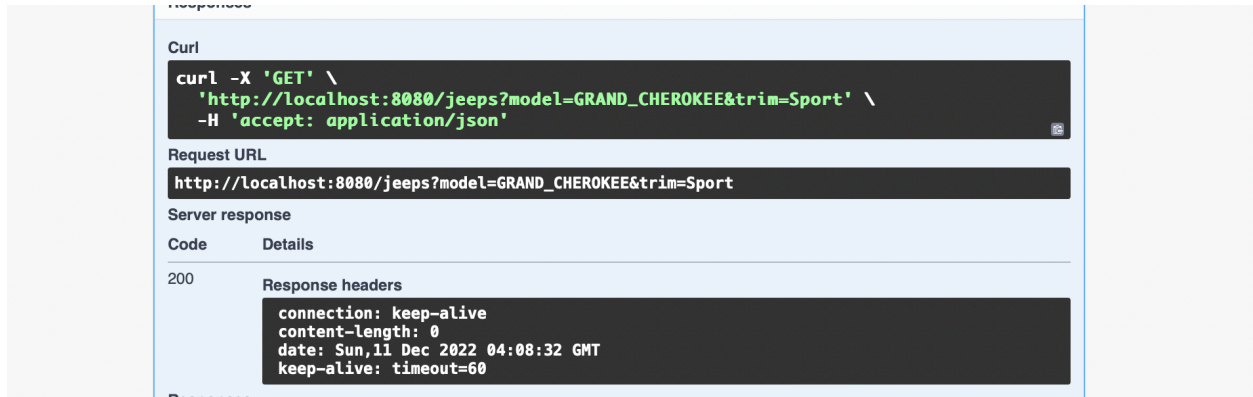
## Coding Steps:

- 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). 

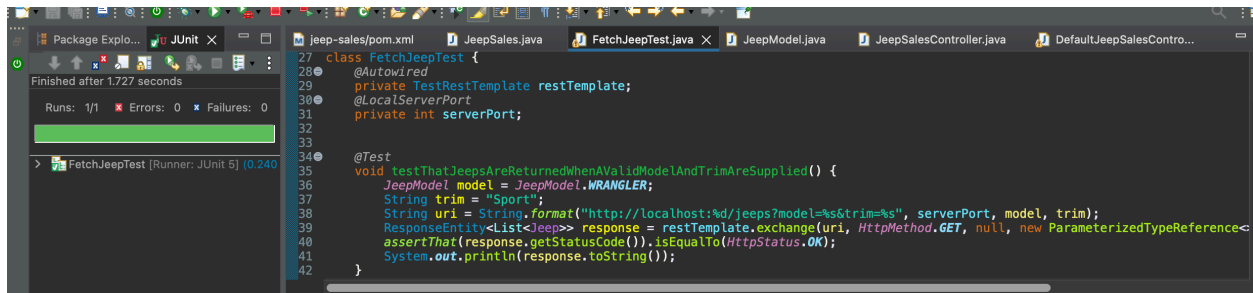


## Web API Design with Spring Boot Week 14 Coding Assignment

- 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       |
|-------------------|-------------|-------------|
| <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.

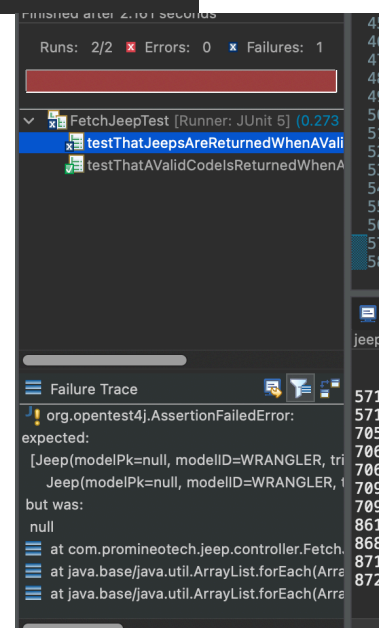
```
@Test
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied(){
    //Given
    JeepModel model = JeepModel.WRANGLER;
    String trim = "Sport";
    String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
    //When
    ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<List<Jeep>>(){});
    //Then
    List<Jeep> expected = buildExpected();
    assertThat(response.getBody()).isEqualTo(expected);
}
```

```
private List<Jeep> buildExpected() {
    List<Jeep> list = new LinkedList<>();
    //formatter:off
    list.add(Jeep.builder()
        .modelID(JeepModel.WRANGLER)
        .trimLevel("Sport")
        .numDoors(2)
        .wheelSize(17)
        .basePrice(new BigDecimal(28475.00))
        .build());

    list.add(Jeep.builder()
        .modelID(JeepModel.WRANGLER)
        .trimLevel("Sport")
        .numDoors(4)
        .wheelSize(17)
        .basePrice(new BigDecimal(31975.00))
        .build());
}
```

b)The JUnit status bar (should be red).

c)The method returning the expected list of Jeeps. 🖥️



# Web API Design with Spring Boot Week 14 Coding Assignment

- 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.
  - 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:

- Project Explorer:** Shows the package structure with `com.promineotech.jeep.service`.
- JUnit Test Results:** Shows a failure for `FetchJeepTest` with the message `org.opentest4j.AssertionFailedError: expected: null but was: null`.
- Source Editor:** Displays the `DefaultJeepSalesService` class implementation:

```
@Service
@Slf4j
public class DefaultJeepSalesService implements JeepSalesService {

    @Override
    public List<Jeep> fetchJeeps(JeepModel model, String trim) {
        log.info("The fetchJeeps method was called with model={} and trim={}", model, trim);
        return null;
    }
}
```
- Console:** Shows the output of the test run, including the log statement from the service class:

```
main] c.p.jeep.controller.FetchJeepTest : Starting FetchJeepTest using Java 11.0.16 on Scotts-Mac-mini.local with I
main] c.p.jeep.controller.FetchJeepTest : The following 1 profile is active: "test"
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.69]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 568 ms
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 63648 (http) with context path ''
main] c.p.jeep.controller.FetchJeepTest : Started FetchJeepTest in 1.466 seconds (JVM running for 1.936)
o-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
o-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
o-1-exec-1] c.p.j.c.DefaultJeepSalesController : Completed initialization in 1 ms
o-1-exec-1] c.p.j.c.DefaultJeepSalesController : Model = WRANGLER, Trim = Sport
o-1-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim=Sport
```

- h)
- 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

# Web API Design with Spring Boot Week 14 Coding Assignment

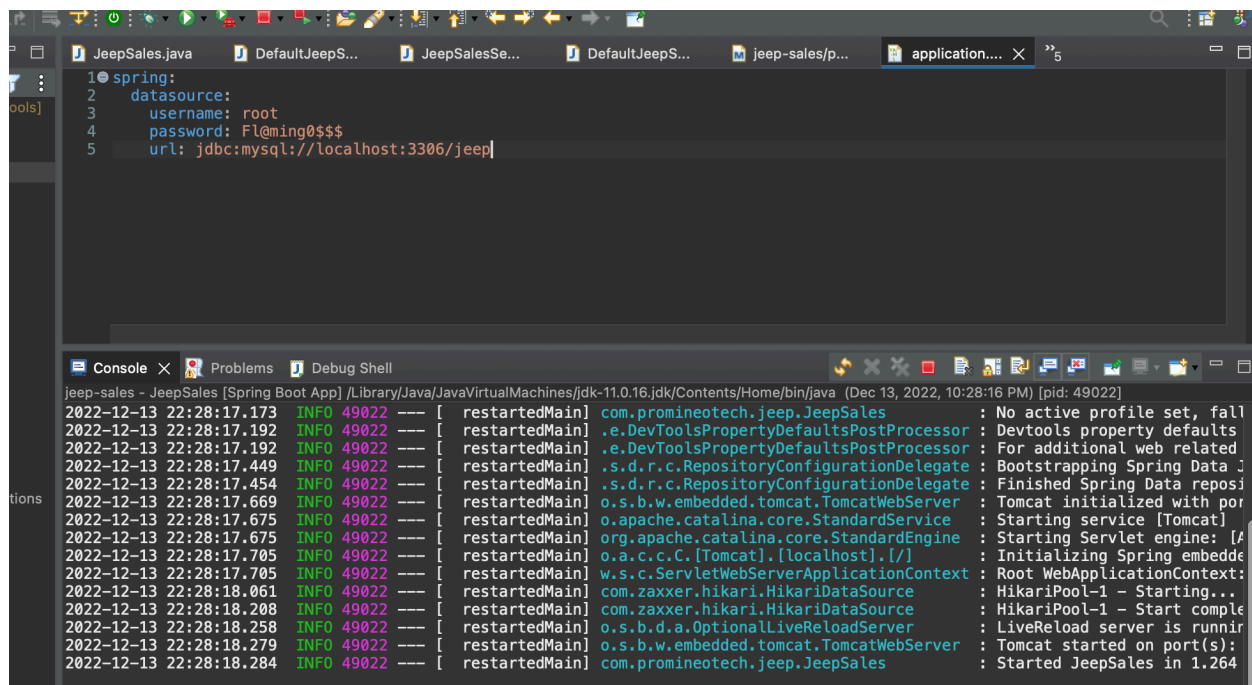
version numbers for either dependency because the version is included in the Spring Boot Starter Parent.

- 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.

A screenshot of an IDE (IntelliJ IDEA) showing the `application.yaml` file and the console output. The `application.yaml` file contains the following configuration:

```
1 spring:
2   datasource:
3     username: root
4     password: Fl@ming0$$$
5     url: jdbc:mysql://localhost:3306/jeep
```

The console output shows the application starting successfully with no errors. The output is as follows:

```
jeep-sales - JeepSales [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk-11.0.16.jdk/Contents/Home/bin/java (Dec 13, 2022, 10:28:16 PM) [pid: 49022]
2022-12-13 22:28:17.173 INFO 49022 --- [ restartedMain] com.promineotech.jee... : No active profile set, fall
2022-12-13 22:28:17.192 INFO 49022 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults
2022-12-13 22:28:17.192 INFO 49022 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related
2022-12-13 22:28:17.449 INFO 49022 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data J
2022-12-13 22:28:17.454 INFO 49022 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data reposi
2022-12-13 22:28:17.669 INFO 49022 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with por
2022-12-13 22:28:17.675 INFO 49022 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-12-13 22:28:17.675 INFO 49022 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [A
2022-12-13 22:28:17.705 INFO 49022 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedde
2022-12-13 22:28:17.705 INFO 49022 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
2022-12-13 22:28:18.061 INFO 49022 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-12-13 22:28:18.208 INFO 49022 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start comple
2022-12-13 22:28:18.258 INFO 49022 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is runnin
2022-12-13 22:28:18.279 INFO 49022 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s):
2022-12-13 22:28:18.284 INFO 49022 --- [ restartedMain] com.promineotech.jee... : Started JeepSales in 1.264
```

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

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.

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

