


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). 
- 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
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. 
- 7) Add a service layer in your application as shown in the videos:
- a) Add a package named `com.promineotech.jeepproject.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. 
- 8) 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.
- 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/jeepproject`). 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.



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:

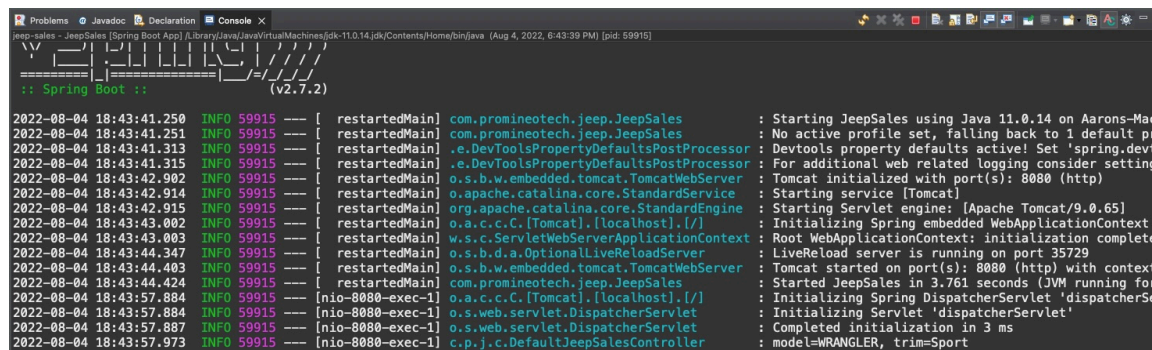
    url: jdbc:h2:mem:jeep
```

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. 

Screenshots of Code:

Step 2:



```

2022-08-04 18:43:41.250 INFO 59915 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSales using Java 11.0.14 on Aarons-Mac
2022-08-04 18:43:41.251 INFO 59915 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profile set, falling back to 1 default p
2022-08-04 18:43:41.313 INFO 59915 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.dev
2022-08-04 18:43:41.315 INFO 59915 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider settin
2022-08-04 18:43:42.902 INFO 59915 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-08-04 18:43:42.914 INFO 59915 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-04 18:43:42.915 INFO 59915 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-04 18:43:43.002 INFO 59915 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-04 18:43:43.003 INFO 59915 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complet
2022-08-04 18:43:44.347 INFO 59915 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-08-04 18:43:44.403 INFO 59915 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with contex
2022-08-04 18:43:44.424 INFO 59915 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSales in 3.761 seconds (JVM running fo
2022-08-04 18:43:57.804 INFO 59915 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherS
2022-08-04 18:43:57.884 INFO 59915 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-04 18:43:57.887 INFO 59915 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
2022-08-04 18:43:57.973 INFO 59915 --- [nio-8080-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport

```

Step 6c:

```
// And: the actual list returned is the same as the expected list
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());

    // @formatter:on
    return list;
}

//('WRANGLER', 'Sport', 2, 17, 28475.00);
//('WRANGLER', 'Sport', 4, 17, 31975.00);
```

Step 7g:

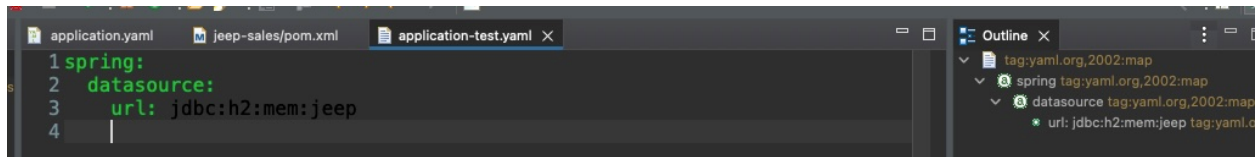
The screenshot shows an IDE with a Java test failure in the left pane and a Spring Boot console log in the bottom pane.

Test Failure: The test `FetchJeepTest` failed at line 13. The failure message is: `org.opentest4j.AssertionFailedError: expected: [Jeep(modelId=Kcmul, model=WRANGLER, trimLevel=Sport, numDoors=2, basePrice=28475.00), Jeep(modelId=Kcmul, model=WRANGLER, trimLevel=Sport, numDoors=4, basePrice=31975.00)] but was: null`.

Spring Boot Console Log: The log shows the application starting successfully. The output is as follows:

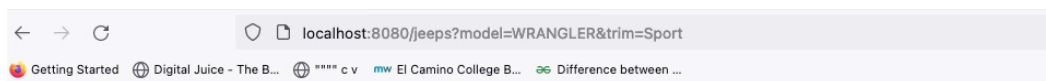
```
2022-08-05 19:45:32.823 INFO 17530 [main] c.p.jee.controller.FetchJeepTest : Starting FetchJeepTest using Java 11.0.14 on Aaron's-MacBook-Air.local w
2022-08-05 19:45:32.825 INFO 17530 [main] c.p.jee.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-08-05 19:45:35.627 INFO 17530 [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2022-08-05 19:45:35.648 INFO 17530 [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-05 19:45:35.648 INFO 17530 [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-05 19:45:35.847 INFO 17530 [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-05 19:45:35.848 INFO 17530 [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2957 ms
2022-08-05 19:45:39.095 INFO 17530 [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 51814 (http) with context path '/'
2022-08-05 19:45:39.124 INFO 17530 [main] c.p.jee.controller.FetchJeepTest : Started FetchJeepTest in 7.398 seconds (JVM running for 9.56)
2022-08-05 19:45:39.902 INFO 17530 [o-auto-1-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-08-05 19:45:39.903 INFO 17530 [o-auto-1-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-05 19:45:39.916 INFO 17530 [o-auto-1-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 13 ms
2022-08-05 19:45:39.987 INFO 17530 [o-auto-1-exec-2] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
2022-08-05 19:45:39.993 INFO 17530 [o-auto-1-exec-2] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model=WRANGLER and trim=Sport
```

Step 12:

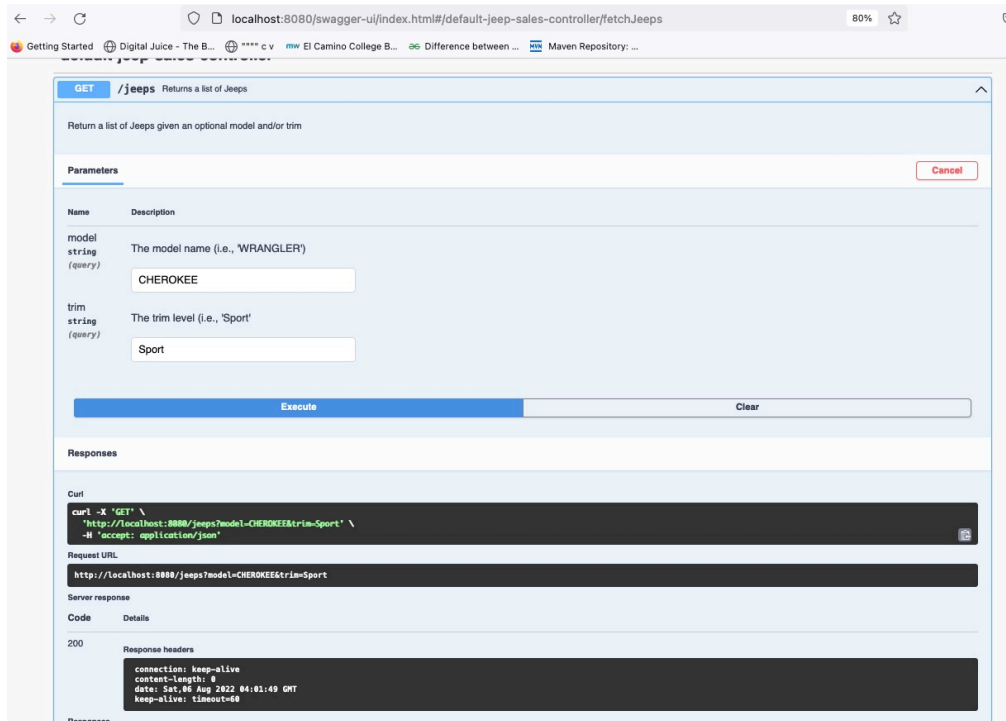


Screenshots of Running Application:

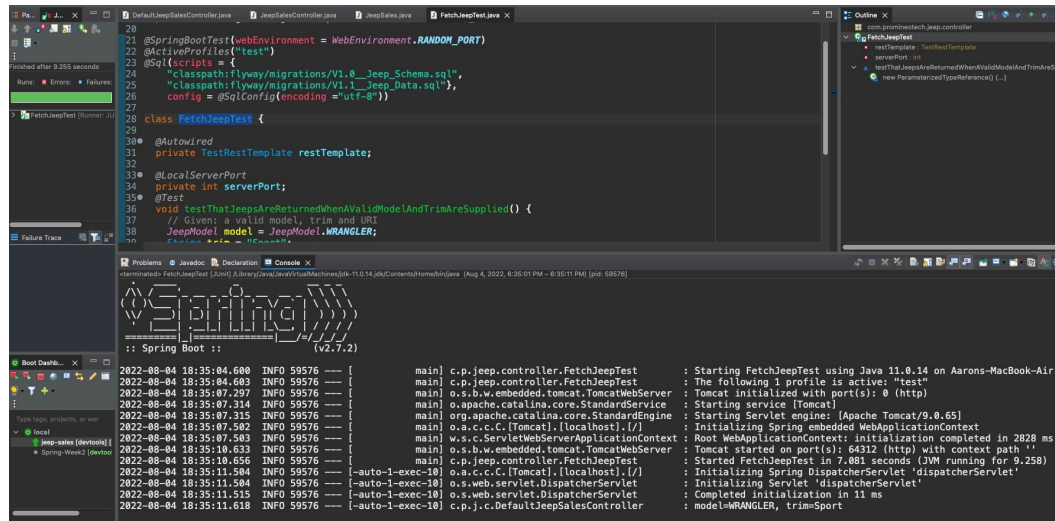
Step 2:



Step 3:



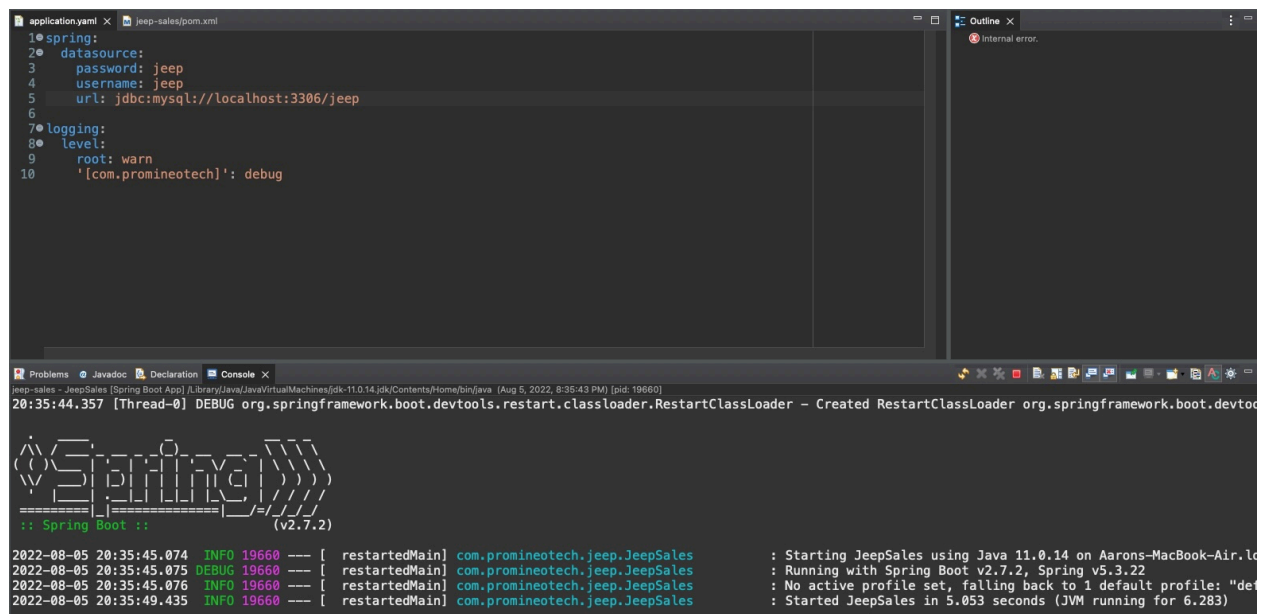
Step 4:



The screenshot shows an IDE with a Java test class `FetchJeepTest` and its console output. The test class is annotated with `@SpringBootTest` and `@ActiveProfiles("test")`. It uses `@Sql` to load a schema and data. The test method `testThatJeepsAreReturnedWhenValidModelAndTrimAreSupplied()` is annotated with `@Test` and `@LocalServerPort`. The console output shows the Spring Boot application starting successfully, with the following log messages:

```
2022-08-04 18:35:04.600 INFO 59576 --- [main] c.p.jeepp.controller.FetchJeepTest : Starting FetchJeepTest using Java 11.0.14 on Aarons-MacBook-Air.
2022-08-04 18:35:04.603 INFO 59576 --- [main] c.p.jeepp.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-08-04 18:35:07.297 INFO 59576 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2022-08-04 18:35:07.314 INFO 59576 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-04 18:35:07.315 INFO 59576 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-04 18:35:07.502 INFO 59576 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-04 18:35:07.503 INFO 59576 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2020 ms
2022-08-04 18:35:10.633 INFO 59576 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 64312 (http) with context path '/'
2022-08-04 18:35:10.656 INFO 59576 --- [main] c.p.jeepp.controller.FetchJeepTest : Started FetchJeepTest in 7.081 seconds (JVM running for 9.258)
2022-08-04 18:35:11.584 INFO 59576 --- [-auto-1-exec-10] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-08-04 18:35:11.584 INFO 59576 --- [-auto-1-exec-10] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-04 18:35:11.515 INFO 59576 --- [-auto-1-exec-10] o.s.web.servlet.DispatcherServlet : Completed initialization in 11 ms
2022-08-04 18:35:11.618 INFO 59576 --- [-auto-1-exec-10] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
```

Step 10:



The screenshot shows an IDE with a `application.yml` file and its console output. The `application.yml` file contains the following configuration:

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

The console output shows the Spring Boot application starting successfully, with the following log messages:

```
2022-08-05 20:35:45.074 INFO 19660 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Starting JeepSales using Java 11.0.14 on Aarons-MacBook-Air.
2022-08-05 20:35:45.075 DEBUG 19660 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Running with Spring Boot v2.7.2, Spring v5.3.22
2022-08-05 20:35:45.076 INFO 19660 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : No active profile set, falling back to 1 default profile: "default"
2022-08-05 20:35:49.435 INFO 19660 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Started JeepSales in 5.053 seconds (JVM running for 6.283)
```

URL to GitHub Repository:

<https://github.com/AaronL1981/Java-Week-14-Web-API-Design.git>