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.

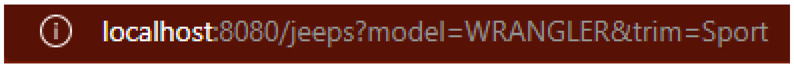
Project Resources:
https://github.com/promineotech/Spring-Boot-Course-Student-Resources

Coding Steps:

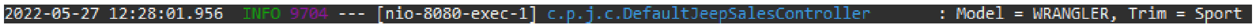
1) In the project you started last week, use Lombok to add an info-level logging statement in the controller implementation method that logs the parameters that were input to the method. Remember to add the `@Slf4j` annotation to the class.

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).

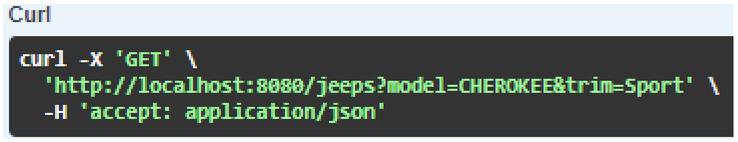


localhost:8080/jeeps?model=WRANGLER&trim=Sport



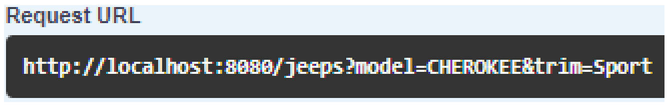
```
2022-05-27 12:28:01.956  INFO 9704 --- [nio-8080-exec-1] c.p.j.c.DefaultJeepSalesController       : Model = WRANGLER, Trim = Sport
```

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.



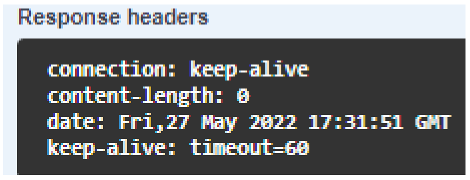
Curl

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



Request URL

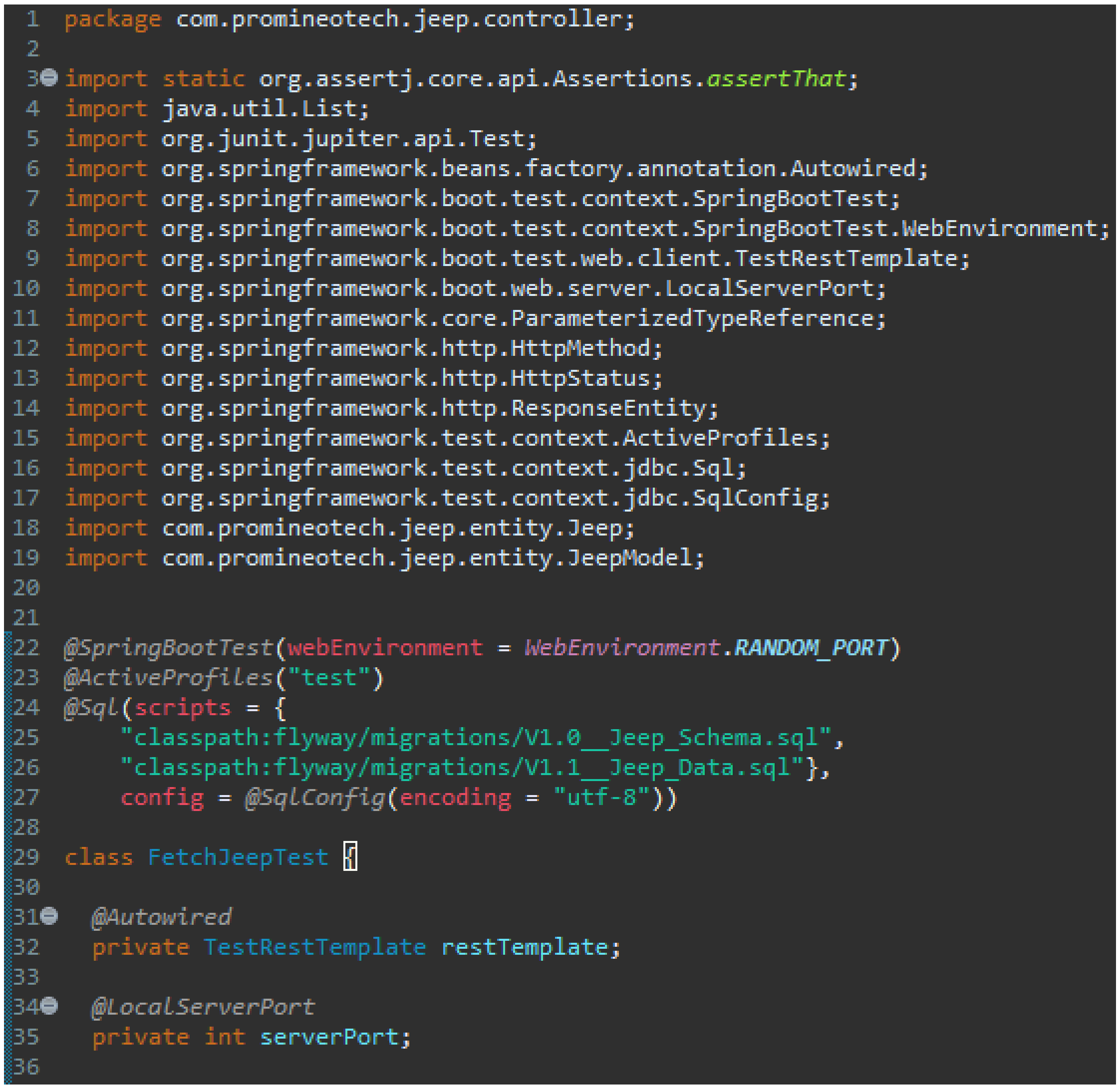
```
http://localhost:8080/jeeps?model=CHEROKEE&trim=Sport
```



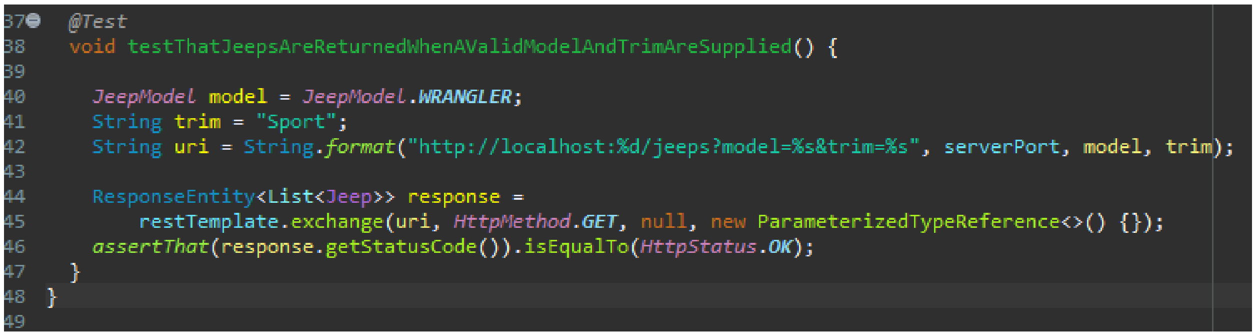
Response headers

```
 connection: keep-alive
 content-length: 0
 date: Fri,27 May 2022 17:31:51 GMT
 keep-alive: timeout=60
```

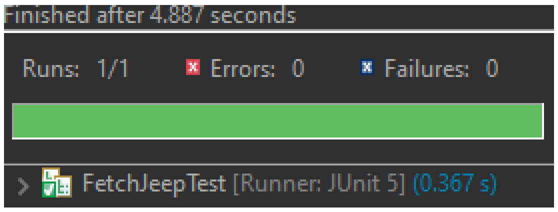
4) Run the integration test and show that the test status is green. Produce a screenshot of the test class and the status bar.



```java
package com.promineotech.jeep.controller;

import static org.assertj.core.api.Assertions.assertThat;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.web.server.LocalServerPort;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.jdbc.SqlConfig;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;


@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@Sql(scripts = {
    "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
    "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
    config = @SqlConfig(encoding = "utf-8"))

class FetchJeepTest {

  @Autowired
  private TestRestTemplate restTemplate;

  @LocalServerPort
  private int serverPort;

```



```java
  @Test
  void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {

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

    ResponseEntity<List<Jeep>> response =
        restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
  }
}

```



```
Finished after 4.887 seconds

Runs: 1/1    Errors: 0    Failures: 0

> FetchJeepTest [Runner: JUnit 5] (0.367 s)
```

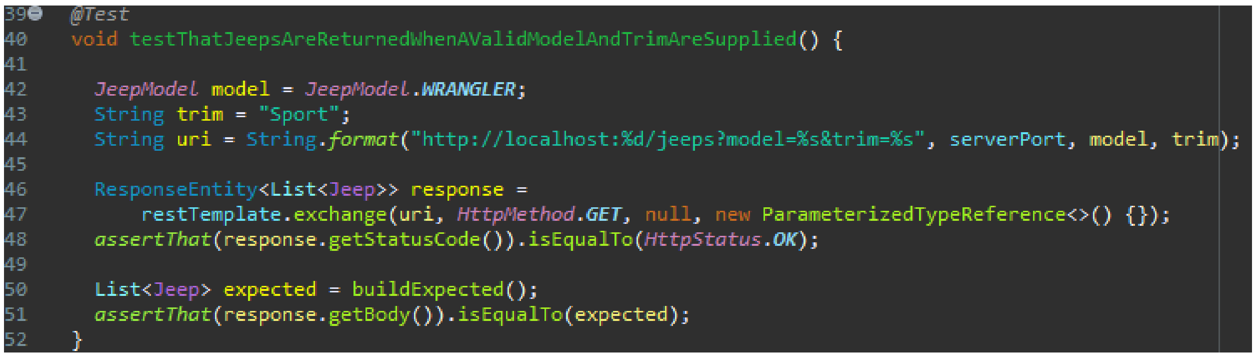
5) Add a method to the test to return a list of expected `Jeep` (`model`) objects based on the model and trim level you selected. You can get the expected list of Jeeps from the file src/test/resources/ flyway/migrations/V1.1__Jeep_Data.sql. So, for example, using the model Wrangler and trim level "Sport",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

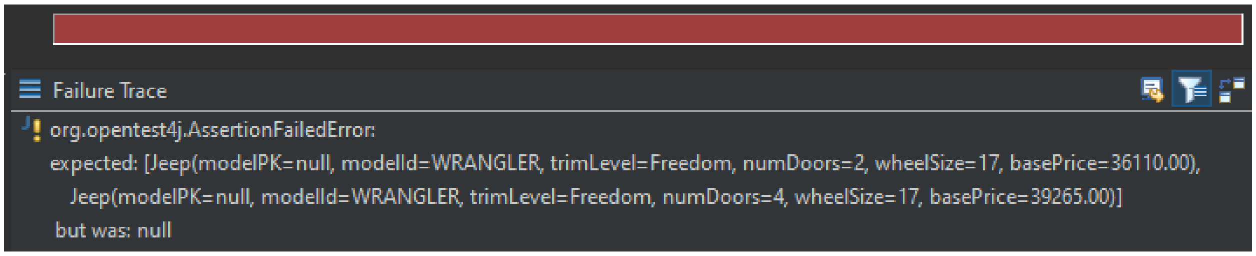
6) Write an AssertJ assertion in the test to assert that the actual list of jeeps returned by the server is the same as the expected list. Run the test. Produce a screenshot showing…

a) The test with the assertion.



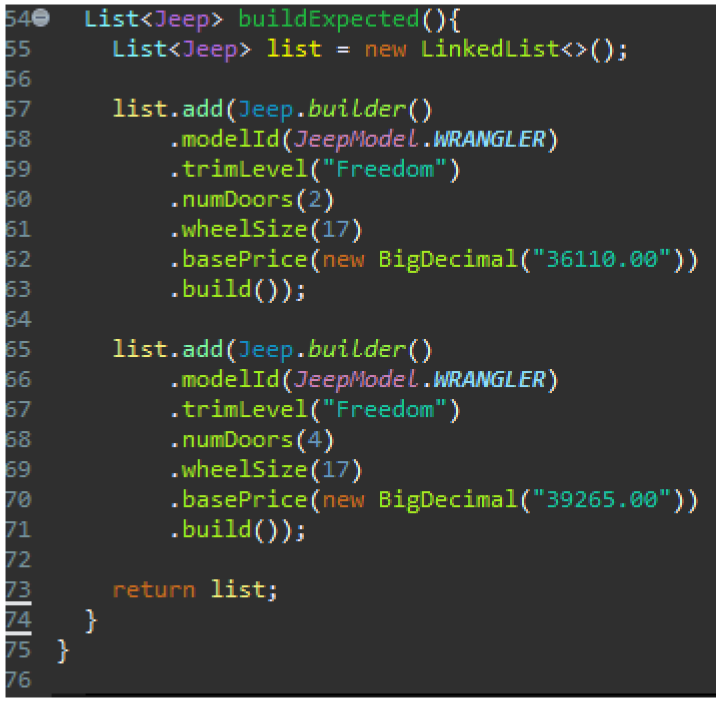
```
@Test
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {

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

  ResponseEntity<List<Jeep>> response =
      restTemplate.exchange(uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
  assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);

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

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



```
org.opentest4j.AssertionFailedError:
expected: [Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Freedom, numDoors=2, wheelSize=17, basePrice=36110.00),
  Jeep(modelPK=null, modelId=WRANGLER, trimLevel=Freedom, numDoors=4, wheelSize=17, basePrice=39265.00)]
 but was: null
```

c) The method returning the expected list of Jeeps.



```java
List<Jeep> buildExpected(){
  List<Jeep> list = new LinkedList<>();

  list.add(Jeep.builder()
      .modelId(JeepModel.WRANGLER)
      .trimLevel("Freedom")
      .numDoors(2)
      .wheelSize(17)
      .basePrice(new BigDecimal("36110.00"))
      .build());

  list.add(Jeep.builder()
      .modelId(JeepModel.WRANGLER)
      .trimLevel("Freedom")
      .numDoors(4)
      .wheelSize(17)
      .basePrice(new BigDecimal("39265.00"))
      .build());

  return list;
 }
}
```

7) 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`.

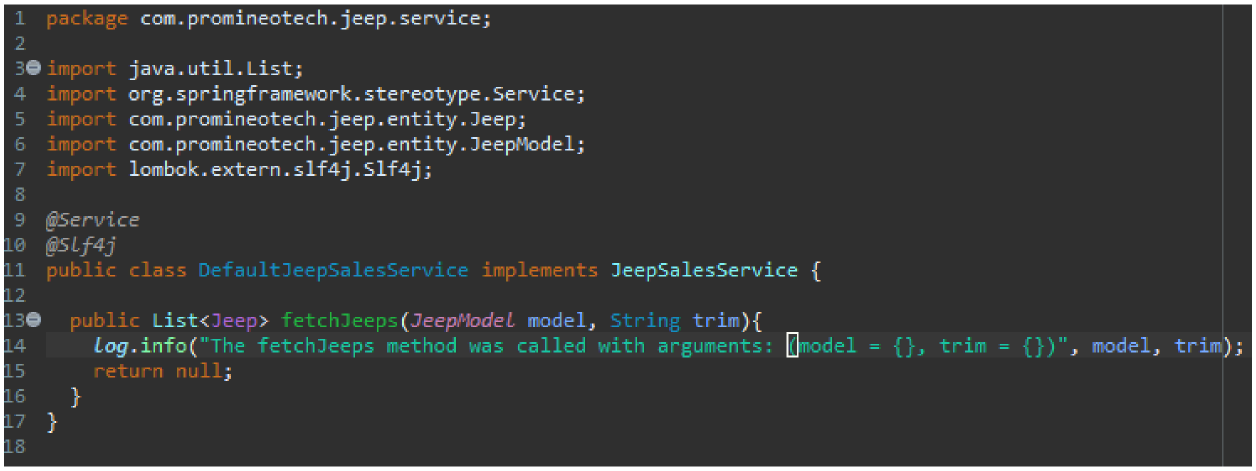
d) Inject the service interface into `DefaultJeepSalesController` using the `@Autowired` annotation. The instance variable should be `private`, and the variable should be named `jeepSalesService`.

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:

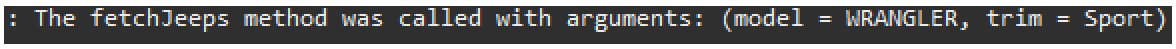
```java
List<Jeep> fetchJeeps(JeepModel model, String trim);
```

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.

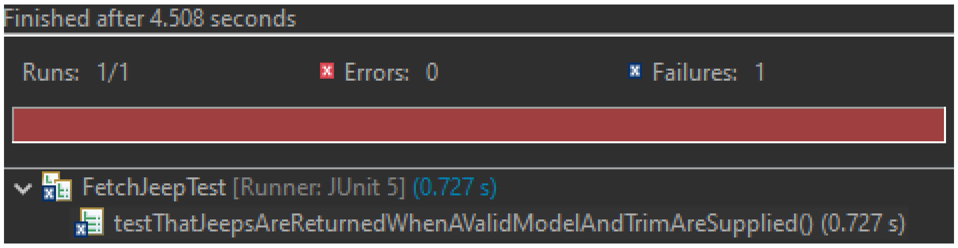
g) Run the test again. Produce a screenshot showing the service class implementation, the log line in the console, and the red status bar.



```java
package com.promineotech.jeep.service;

import java.util.List;
import org.springframework.stereotype.Service;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import lombok.extern.slf4j.Slf4j;

@Service
@Slf4j
public class DefaultJeepSalesService implements JeepSalesService {

  public List<Jeep> fetchJeeps(JeepModel model, String trim){
    log.info("The fetchJeeps method was called with arguments: (model = {}, trim = {})", model, trim);
    return null;
  }
}
```



```
: The fetchJeeps method was called with arguments: (model = WRANGLER, trim = Sport)
```



```
Finished after 4.508 seconds
Runs: 1/1    Errors: 0    Failures: 1
FetchJeepTest [Runner: JUnit 5] (0.727 s)
    testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() (0.727 s)
```

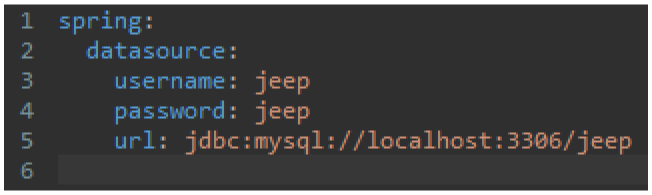
8) Add the database dependencies described in the video to the POM file (MySQL driver and Spring Boot Starter JDBC). To find them, nagivate to https://mvnrepository.com/. Search for `mysql-connector-j` and `spring-boot-starter-jdbc`. In the POM file you don't need version numbers for either dependency 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/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.

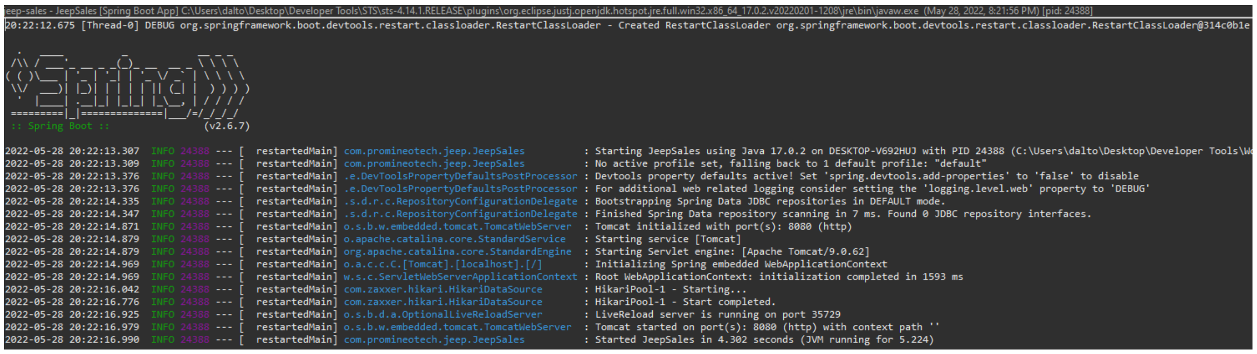
   Be careful with the indentation! YAML allows hierarchical configuration but it reads the hierarchy based on the indentation level. The keyword "spring" MUST start in the first column. It should look similar to this when done:

```yaml
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.



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

```



```
eep-sales - JeepSales [Spring Boot App] C:\Users\dalto\Desktop\Developer Tools\STS\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe  (May 28, 2022, 8:21:56 PM) [pid: 24388]
20:22:12.675 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.classloader.RestartClassLoader@314c0b1e

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.7)

2022-05-28 20:22:13.307  INFO 24388 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Starting JeepSales using Java 17.0.2 on DESKTOP-V692HUJ with PID 24388 (C:\Users\dalto\Desktop\Developer Tools\W
2022-05-28 20:22:13.309  INFO 24388 --- [  restartedMain] com.promineotech.jeep.JeepSales          : No active profile set, falling back to 1 default profile: "default"
2022-05-28 20:22:13.376  INFO 24388 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2022-05-28 20:22:13.376  INFO 24388 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-05-28 20:22:14.335  INFO 24388 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-05-28 20:22:14.347  INFO 24388 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 7 ms. Found 0 JDBC repository interfaces.
2022-05-28 20:22:14.871  INFO 24388 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-05-28 20:22:14.879  INFO 24388 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-05-28 20:22:14.879  INFO 24388 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.62]
2022-05-28 20:22:14.969  INFO 24388 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-05-28 20:22:14.969  INFO 24388 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1593 ms
2022-05-28 20:22:16.042  INFO 24388 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-05-28 20:22:16.776  INFO 24388 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-05-28 20:22:16.925  INFO 24388 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-05-28 20:22:16.979  INFO 24388 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-05-28 20:22:16.990  INFO 24388 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Started JeepSales in 4.302 seconds (JVM running for 5.224)
```

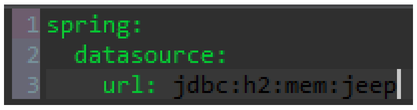
11) Add the H2 database as dependency. Search for the dependency in the Maven repository like you did above. Search for "h2" and pick the latest version. Again, you don't need the version number, but the scope should be set to "test".

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:

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

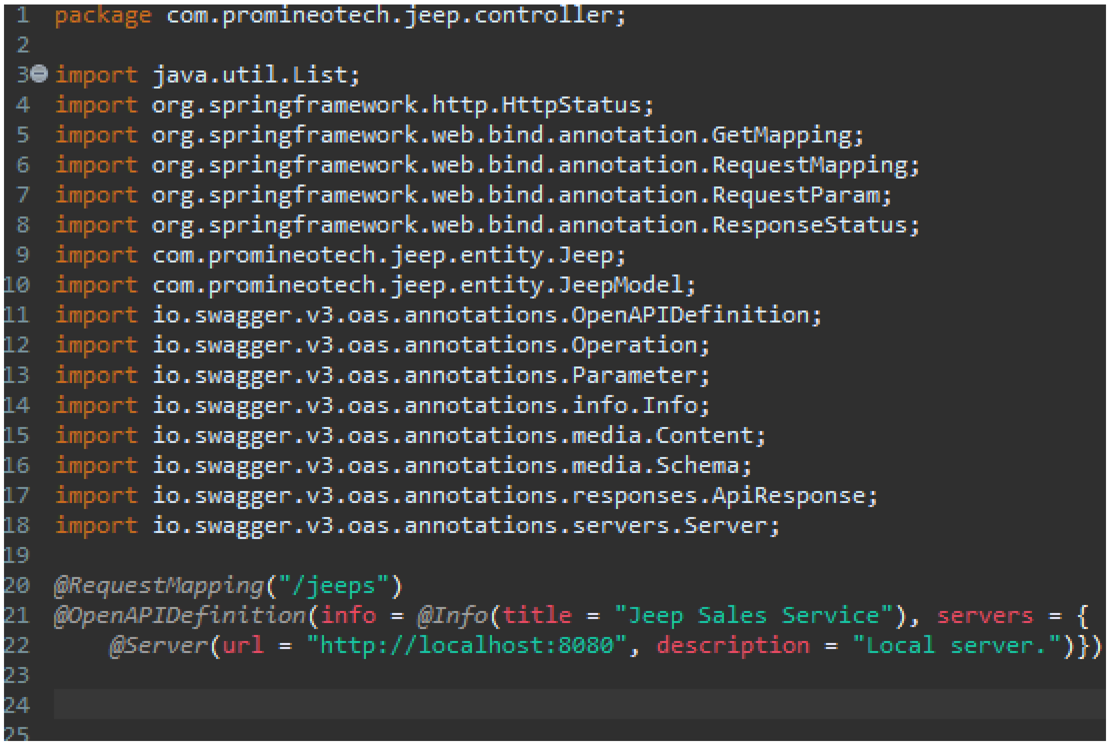
You do not need to set the username and password because the in-memory H2 database does not require them.

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

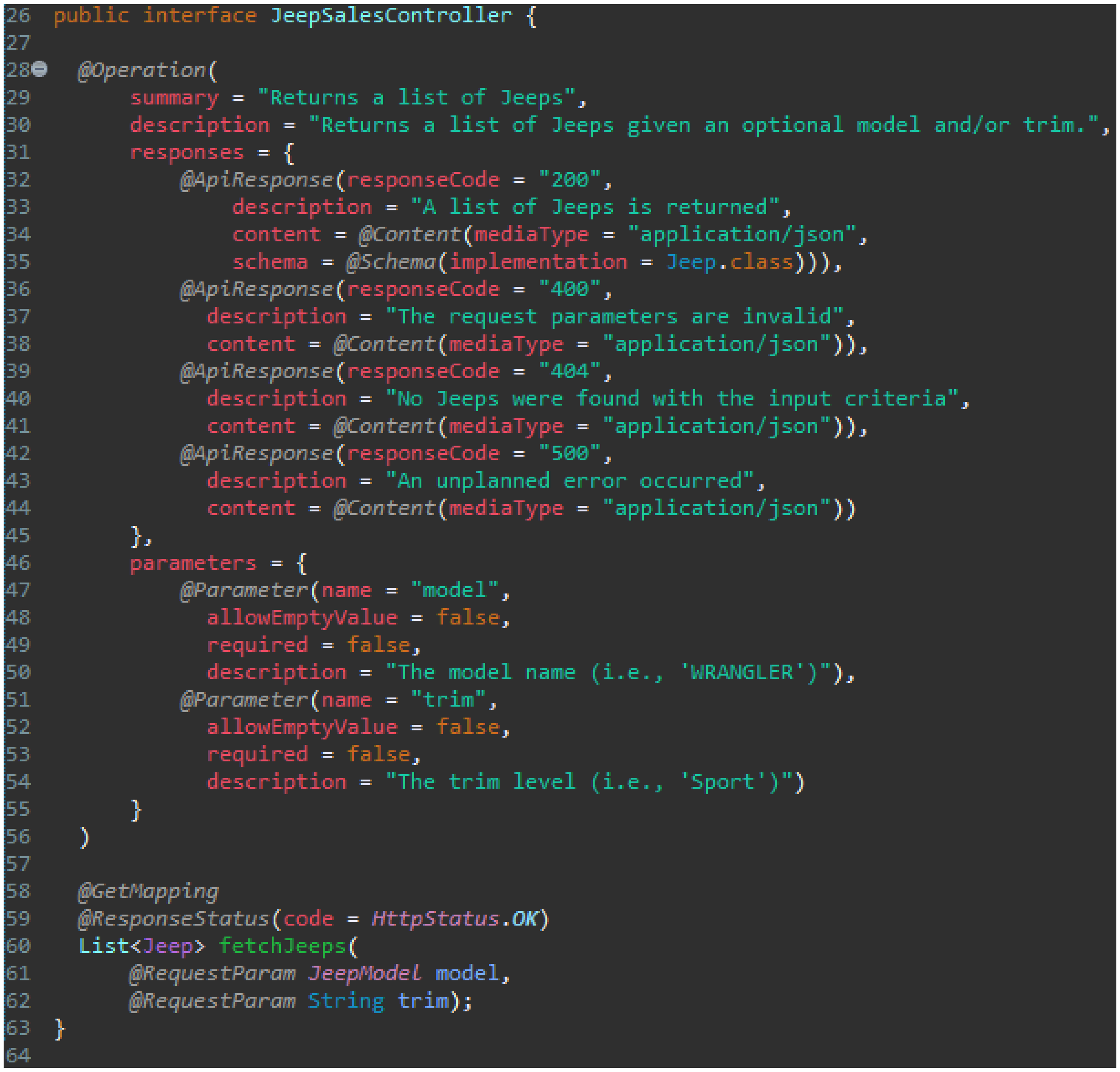


```yaml
spring:
  datasource:
    url: jdbc:h2:mem:jeep
```

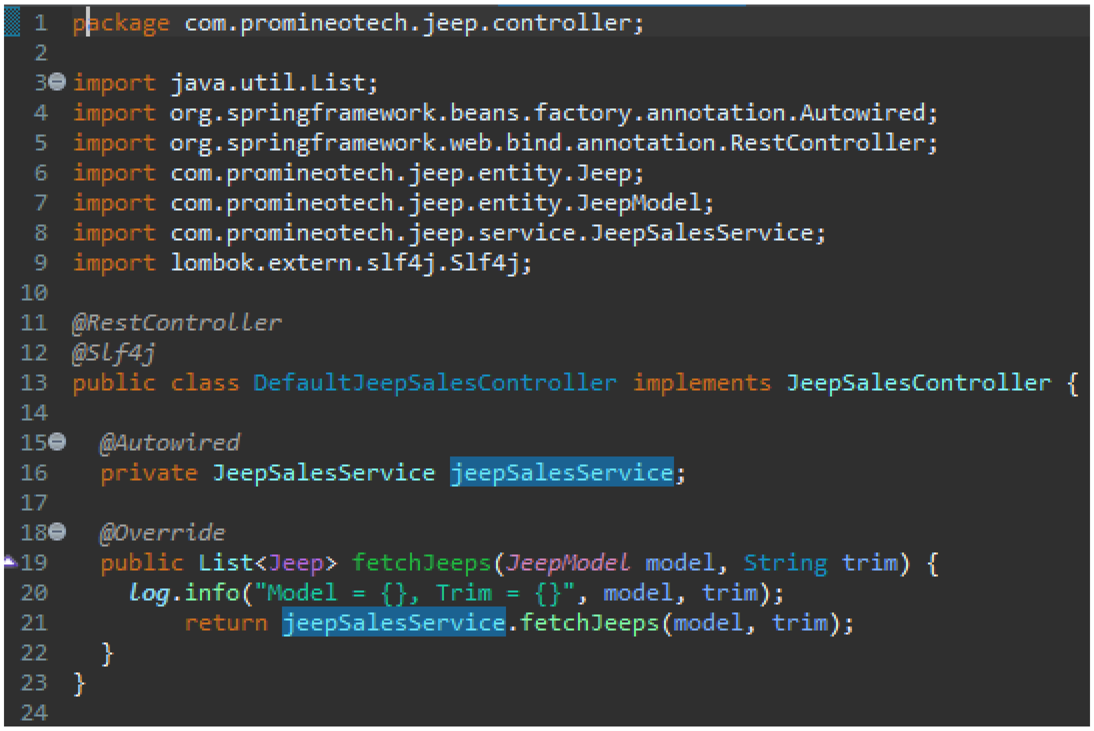
Screenshots of Code:



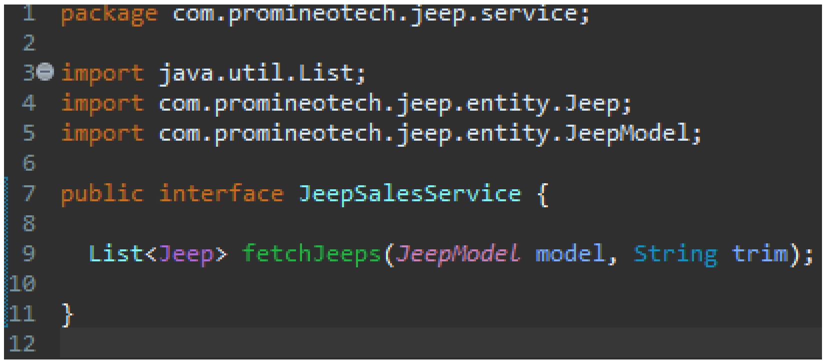
```java
package com.promineotech.jeep.controller;

import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.servers.Server;

@RequestMapping("/jeeps")
@OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})


```



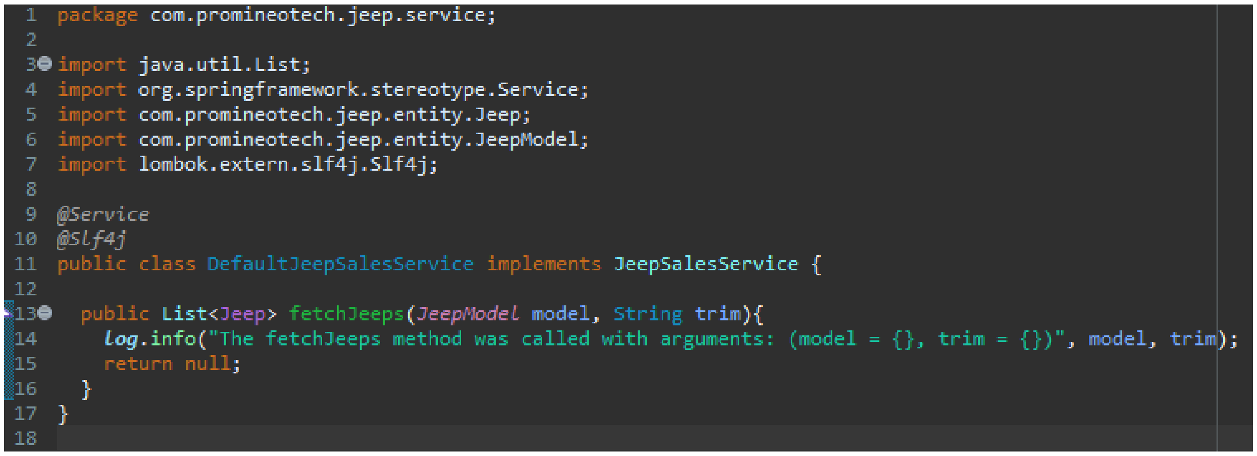
```java
public interface JeepSalesController {

  @Operation(
      summary = "Returns a list of Jeeps",
      description = "Returns a list of Jeeps given an optional model and/or trim.",
      responses = {
          @ApiResponse(responseCode = "200",
              description = "A list of Jeeps is returned",
              content = @Content(mediaType = "application/json",
              schema = @Schema(implementation = Jeep.class))),
          @ApiResponse(responseCode = "400",
            description = "The request parameters are invalid",
            content = @Content(mediaType = "application/json")),
          @ApiResponse(responseCode = "404",
            description = "No Jeeps were found with the input criteria",
            content = @Content(mediaType = "application/json")),
          @ApiResponse(responseCode = "500",
            description = "An unplanned error occurred",
            content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(name = "model",
            allowEmptyValue = false,
            required = false,
            description = "The model name (i.e., 'WRANGLER')"),
          @Parameter(name = "trim",
            allowEmptyValue = false,
            required = false,
            description = "The trim level (i.e., 'Sport')")
      }
  )

  @GetMapping
  @ResponseStatus(code = HttpStatus.OK)
  List<Jeep> fetchJeeps(
      @RequestParam JeepModel model,
      @RequestParam String trim);
}
```



```java
package com.promineotech.jeep.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import com.promineotech.jeep.service.JeepSalesService;
import lombok.extern.slf4j.Slf4j;

@RestController
@Slf4j
public class DefaultJeepSalesController implements JeepSalesController {

  @Autowired
  private JeepSalesService jeepSalesService;

  @Override
  public List<Jeep> fetchJeeps(JeepModel model, String trim) {
    log.info("Model = {}, Trim = {}", model, trim);
        return jeepSalesService.fetchJeeps(model, trim);
  }
}

```

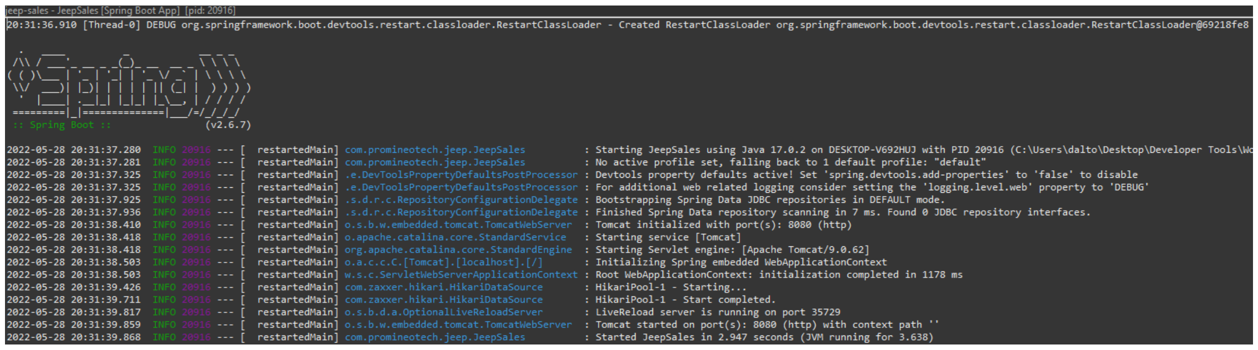


```java
package com.promineotech.jeep.service;

import java.util.List;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;

public interface JeepSalesService {

  List<Jeep> fetchJeeps(JeepModel model, String trim);

}

```



```java
package com.promineotech.jeep.service;

import java.util.List;
import org.springframework.stereotype.Service;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import lombok.extern.slf4j.Slf4j;

@Service
@Slf4j
public class DefaultJeepSalesService implements JeepSalesService {

  public List<Jeep> fetchJeeps(JeepModel model, String trim){
    log.info("The fetchJeeps method was called with arguments: (model = {}, trim = {})", model, trim);
    return null;
  }
}

```

Screenshots of Running Application:



```
jeep-sales - JeepSales [Spring Boot App]  [pid: 20916]
20:31:36.910 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.classloader.RestartClassLoader@69218fe8

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.7)

2022-05-28 20:31:37.280  INFO 20916 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Starting JeepSales using Java 17.0.2 on DESKTOP-V692HUJ with PID 20916 (C:\Users\dalto\Desktop\Developer Tools\Wo
2022-05-28 20:31:37.281  INFO 20916 --- [  restartedMain] com.promineotech.jeep.JeepSales          : No active profile set, falling back to 1 default profile: "default"
2022-05-28 20:31:37.325  INFO 20916 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2022-05-28 20:31:37.325  INFO 20916 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-05-28 20:31:37.925  INFO 20916 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-05-28 20:31:37.936  INFO 20916 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 7 ms. Found 0 JDBC repository interfaces.
2022-05-28 20:31:38.410  INFO 20916 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-05-28 20:31:38.418  INFO 20916 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-05-28 20:31:38.418  INFO 20916 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.62]
2022-05-28 20:31:38.503  INFO 20916 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-05-28 20:31:38.503  INFO 20916 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1178 ms
2022-05-28 20:31:39.426  INFO 20916 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-05-28 20:31:39.711  INFO 20916 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-05-28 20:31:39.817  INFO 20916 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-05-28 20:31:39.859  INFO 20916 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-05-28 20:31:39.868  INFO 20916 --- [  restartedMain] com.promineotech.jeep.JeepSales          : Started JeepSales in 2.947 seconds (JVM running for 3.638)
```

URL to GitHub Repository:

DaltonCash/PT-WK14 (github.com)