Web API Design with Spring Boot Week 4 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:

For this week's homework you need to copy source code from the supplied resources.

For this week's homework you need to copy source code from the Source folder in the supplied resources. Wait until the instructions tell you to copy the resources or you will get errors.

1) Select some options for a Jeep order:

a) Use the `data.sql` file or the jeep database tables to select options for a Jeep order. Select any one of each of the following for the order:
   i) color
   ii) customer
   iii) engine
   iv) model
   v) tire(s)

b) Select one or more options from the options table as well. Keep in mind that some options may work better than others – but if you want to put 37-inch tires on your Jeep Renegade, so be it!

2) Create a new integration test class to test a Jeep order named `CreateOrderTest.java`. Create this class in `src/test/java` in the `com.promineotech.jeep.controller` package.

a) Add the Spring Boot Test annotations: `@SpringBootTest`, `@ActiveProfiles`, and `@Sql`. They should have the same parameters as the test created in weeks 1 and 2.

b) Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.

c) In the test class, create a method named `createOrderBody`. This method returns a type of String. In this method, return a JSON object with the IDs that you picked in Step 1a and b. For example:

```
{
  "customer":"MORISON_LINA",
  "model":"WRANGLER",
  "trim":"Sport Altitude",
  "doors":4,
  "color":"EXT_NACHO",
  "engine":"2_0_TURBO",
  "tire":"35_TOYO",
  "options":[
    "DOOR_QUAD_4",
    "EXT_AEV_LIFT",
    "EXT_WARN_WINCH",
    "EXT_WARN_BUMPER_FRONT",
    "EXT_WARN_BUMPER_REAR",
    "EXT_ARB_COMPRESSOR"
  ]
}
```

Make sure that the JSON is correct! If necessary, use a JSON formatter/validator like the one here: https://jsonformatter.curiousconcept.com/.

Produce a screenshot of the `createOrderBody()` method.

```
String createOrderBody() {
  return "{\n"
      + "  \"customer\":\"MAYNARD_TORBJORG\",\n"
      + "  \"model\":\"GLADIATOR\",\n"
      + "  \"trim\":\"Mojave\",\n"
      + "  \"doors\":4,\n"
      + "  \"color\":\"EXT_SARGE_GREEN\",\n"
      + "  \"engine\":\"3_6_HYBRID\",\n"
      + "  \"tire\":\"255_GOODYEAR\",\n"
      + "  \"options\":[\n"
      + "    \"DOOR_QUAD_4\",\n"
      + "    \"EXT_QUAD_ALUM_FRONT\",\n"
      + "    \"EXT_WARN_WINCH\",\n"
      + "    \"EXT_WARN_BUMPER_FRONT\",\n"
      + "    \"EXT_WARN_BUMPER_REAR\",\n"
      + "    \"EXT_ARB_COMPRESSOR\"\n"
      + "  ]\n"
      + "}\n"
      + "";
}
```

In the test method, assign the return value of the `createOrderBody()` method to a variable named `body`.

d) In the test class, add an instance variable named `serverPort` to hold the port that Tomcat is listening on in the test. Annotate the variable with `@LocalServerPort`.

e) Add another instance variable for an injected `TestRestTemplate` named `restTemplate`.

f) In the test method, assign a value to a local variable named uri as follows:

```
String uri = String.format("http://localhost:%d/orders", serverPort);
```

g) In the test method, create an HttpHeaders object and set the content type to "application/json" like this:

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
```

Make sure to import the package `org.springframework.http.HttpHeaders`.

h) Create an HttpEntity object and set the request body and headers:

```
HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
```

i) Send the request body and headers to the server. The Order class should have been copied earlier from the supplied resources. Ensure that you import `com.promineotech.jeep.entity.Order` and not some other Order class.

```
ResponseEntity<Order> response = restTemplate.exchange(uri,
    HttpMethod.POST, bodyEntity, Order.class);
```

j) Add the AssertJ assertions to ensure that the response is correct. Replace the expected values to match the JSON in step 2c.

```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
assertThat(response.getBody()).isNotNull();

Order order = response.getBody();
assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
assertThat(order.getModel().getNumDoors()).isEqualTo(4);
assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
assertThat(order.getOptions()).hasSize(6);
```

k) Produce a screenshot of the test method.

```
@Test
void testCreateOrderReturnsSuccess201() {

  String body = createOrderBody();
  String uri = String.format("http://localhost:%d/orders", serverPort);
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);
  HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
  ResponseEntity<Order> response = restTemplate.exchange(uri,HttpMethod.POST, bodyEntity, Order.class);

  assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
  assertThat(response.getBody()).isNotNull();

  Order order = response.getBody();

  assertThat(order.getCustomer().getCustomerId()).isEqualTo("MAYNARD_TORBJORG");
  assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GLADIATOR);
  assertThat(order.getModel().getTrimLevel()).isEqualTo("Mojave");
  assertThat(order.getModel().getNumDoors()).isEqualTo(4);
  assertThat(order.getColor().getColorId()).isEqualTo("EXT_SARGE_GREEN");
  assertThat(order.getEngine().getEngineId()).isEqualTo("3_6_HYBRID");
  assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
  assertThat(order.getOptions()).hasSize(6);
}
```

3) In the `controller` sub-package in `src/main/java`, create an interface named `JeepOrderController`. Add `@RequestMapping("/orders")` as a class-level annotation.

a) Create a method in the interface to create an order (`createOrder`). It should return an object of type `Order` (see below). It should accept a single parameter of type `OrderRequest` as described in the video. Make sure it accepts an `HTTP POST` request and returns a status code of 201 (created).

b) Add the `@RequestBody` annotation to the orderRequest parameter. Make sure to add the `RequestBody` annotation from the `org.springframework.web.bind.annotation` package.

c) Produce a screenshot of the finished `JeepOrderController` interface showing no compile errors.

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

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.OrderRequest;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;

@RequestMapping("/orders")
public interface JeepOrderController {

  @Operation(
      summary = "Create an order for a Jeep",
      description = "Retruns the created Jeep",
      responses = {
          @ApiResponse(responseCode = "201",
              description = "The created Jeep 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 = "A Jeep component was not 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 = "orderRequest",
            required = true,
            description = "The order as JSON"),

      }
  )
  @PostMapping
  @ResponseStatus(code = HttpStatus.CREATED)
  Order createOrder(@RequestBody OrderRequest orderRequest);
}
```

4) Create a class that implements `JeepOrderController` named `DefaultJeepOrderController`.

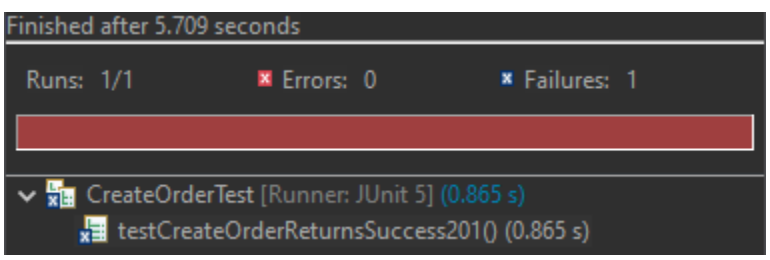
   a) Add `@RestController` as a class-level annotation.

b) Add a log line to the implementing controller method showing the input request body (orderRequest)

c) Run the test to show a red status bar. Produce a screenshot that shows the test method, the log line, and the red JUnit status bar.

```
@Test
void testCreateOrderReturnsSuccess201() {

  String body = createOrderBody();
  String uri = String.format("http://localhost:%d/orders", serverPort);
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);
  HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
  ResponseEntity<Order> response = restTemplate.exchange(uri,HttpMethod.POST, bodyEntity, Order.class);

  assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
  assertThat(response.getBody()).isNotNull();

  Order order = response.getBody();

  assertThat(order.getCustomer().getCustomerId()).isEqualTo("MAYNARD_TORBJORG");
  assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GLADIATOR);
  assertThat(order.getModel().getTrimLevel()).isEqualTo("Mojave");
  assertThat(order.getModel().getNumDoors()).isEqualTo(4);
  assertThat(order.getColor().getColorId()).isEqualTo("EXT_SARGE_GREEN");
  assertThat(order.getEngine().getEngineId()).isEqualTo("3_6_HYBRID");
  assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
  assertThat(order.getOptions()).hasSize(6);
}
```

```
: Order = OrderRequest(customer=MAYNARD_TORBJORG, model=GLADIATOR, trim=Mojave, doors=4,
color=EXT_SARGE_GREEN, engine=3_6_HYBRID, tire=255_GOODYEAR, options=[DOOR_QUAD_4,
EXT_QUAD_ALUM_FRONT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ARB_COMPRESSOR])
```



5) Find the Maven dependency `spring-boot-starter-validation` by looking it up at https://mvnrepository.com/. Add this repository to the project POM file (pom.xml).

6) Add the class-level annotation `@Validated` to the `JeepOrderController` interface.

7) Add Bean Validation annotations to the `OrderRequest` class as shown in the video.

   a) Use these annotations for String types:

      i) `@NotNull`
      ii) `@Length(max = 30)`
      iii) `@Pattern(regexp = "[\\w\\s]*")`

   b) Use these annotations for integer types:

i) `@Positive`
ii) `@Min(2)`
iii) `@Max(4)`

c) Add `@NotNull` to the enum type.

d) Add validation to the list element (type String) by adding the validation annotations *inside* the generic definition. So, to add the String validation to the options, you would do this:

```
private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String>
options;
```

Do not apply a `@NotNull` annotation to the `List` because if you have no options the `List` may be null.

e) Produce a screenshot of this class with the annotations.

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

import java.util.List;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Positive;
import org.hibernate.validator.constraints.Length;
import lombok.Data;

@Data
public class OrderRequest {

  @NotNull
  @Length(max =30)
  @Pattern(regexp = "[\\w\\s]*")
  private String customer;

  @NotNull
  private JeepModel model;

  @NotNull
  @Length(max =30)
  @Pattern(regexp = "[\\w\\s]*")
  private String trim;

  @Positive
  @Min(2)
  @Max(4)
  private int doors;
  @NotNull
  @Length(max =30)
  @Pattern(regexp = "[\\w\\s]*")
  private String color;

  @NotNull
  @Length(max =30)
  @Pattern(regexp = "[\\w\\s]*")
  private String engine;

  @NotNull
  @Length(max =30)
  @Pattern(regexp = "[\\w\\s]*")
  private String tire;

  private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String> options;
}

```

8) In the `jeep.service` sub-package, create the empty (no methods yet) order service interface (named `JeepOrderService`) and implementation (named `DefaultJeepOrderService`).

   a) Inject the interface into the order controller implementation class.
   b) Add the `@Service` annotation to the service implementation class.

c) Create the `createOrder` method in the interface and implementing service. The method signature should look like this:

```
Order createOrder(OrderRequest orderRequest);
```

d) Call the `createOrder` method from the controller and return the value returned by the service.

e) Add a log line in the createOrder method and log the `orderRequest` parameter.

f) Run the test CreateOrderTest again. Produce a screenshot showing that the service layer createOrder method correctly prints the log line in the console. (e.g. prints out the OrderRequest in the console from within the Service Layer).

```
c.p.j.service.DefaultJeepOrderService    : Order = OrderRequest(customer=MAYNARD_TORBJORG,
model=GLADIATOR, trim=Mojave, doors=4, color=EXT_SARGE_GREEN, engine=3_6_HYBRID,
tire=255_GOODYEAR, options=[DOOR_QUAD_4, EXT_QUAD_ALUM_FRONT, EXT_WARN_WINCH,
EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ARB_COMPRESSOR])
```

9) In the `jeep.dao` sub-package, create the empty (no methods yet) DAO interface (named `JeepOrderDao`) and implementation (named `DefaultJeepOrderDao`).

a) Inject the DAO interface into the order service implementation class.

b) Add the `@Component` annotation to the DAO implementation class.

10) Replace the entire content of `JeepOrderDao.java` with the source found in `JeepOrderDao.source`. The source file is found in the Source folder of the supplied project resources.

11) * The next steps require you to copy source code from the Source directory in the supplied resources. Please follow the instructions EXACTLY. Some steps require you to replace ALL the source in a file. Some steps require you to ADD source to a file.**

12) Copy the *contents* of the file `DefaultJeepOrderDao.source` *into* `DefaultJeepOrderDao.java`. The source file is found in the Source folder of the supplied project resources.

In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable. Make sure you pick the import `java.util.Optional`, `java.util.List`, and `org.springframework.jdbc.core.RowMapper`.

13) Copy the *contents* of the file `DefaultJeepOrderService.source` *into* `DefaultJeepOrderService.java`. Add the source after the `createOrder()` method, but *inside* the class body. The source file is found in the `Source` folder of the supplied project resources.

In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable.

14) In `DefaultJeepOrderService.java`, work with the method `createOrder`.

a) Add the `@Transactional` annotation to the createOrder method.

b) In the `createOrder` method call the copied methods: `getCustomer`, `getModel`, `getColor`, `getEngine`, `getTire` and `getOption`, assigning the return values of these methods to variables of the appropriate types.

c) Calculate the price, including all options.

15) In `JeepOrderDao.java` and `DefaultJeepOrderDao.java`, add the method:

```
Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire
      tire, BigDecimal price, List<Option> options);
```

a) Call the jeepOrder.Dao.saveOrder method from the jeepOrderSalesService.createOrder service. Produce a screenshot of the jeepOrderSalesService.createOrder method.

```
@Transactional
public Order createOrder(OrderRequest orderRequest) {
  log.info("Order = {}",orderRequest);

  Customer customer = getCustomer(orderRequest);
  Jeep jeep = getModel(orderRequest);
  Color color = getColor(orderRequest);
  Engine engine = getEngine(orderRequest);
  Tire tire = getTire (orderRequest);
  List<Option> options = getOption(orderRequest);

  BigDecimal price =
      jeep.getBasePrice()
      .add(color.getPrice())
      .add(engine.getPrice())
      .add(tire.getPrice());

  for(Option option : options) {
    price.add(option.getPrice());
  }

  return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
}
```

b) Write the implementation of the `saveOrder` method in the DAO.

i) Call the supplied `generateInsertSql` method, passing in the customer, jeep, color, engine, tire and price. Assign the return value of the method to a `SqlParams` object.

ii) Call the `update` method on the `NamedParameterJdbcTemplate` object, passing in a `KeyHolder` object as shown in the video. Create the `KeyHolder` like this:

```
KeyHolder keyHolder = new GeneratedKeyHolder();
```

Be sure to extract the order primary key from the `KeyHolder` object into a variable of type `Long` named `orderPK`.

iii) Write a method named `saveOptions` as shown in the video. This method should have the following method signature:

```
private void saveOptions(List<Option> options, Long orderPK)
```

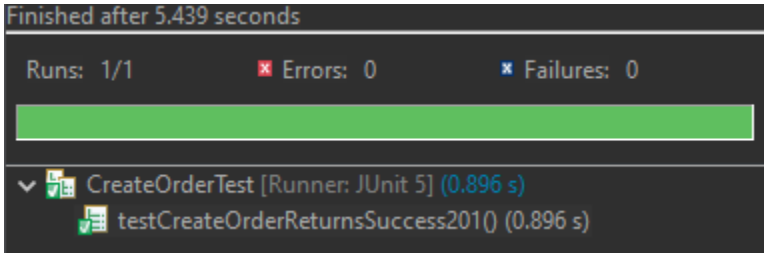
For each option in the `Options` list, call the supplied `generateInsertSql` method passing the parameters `option` and order primary key (`orderPK`). Call the `update` method on the `NamedParameterJdbcTemplate` object.

iv) In the `saveOrder` method in the DAO implementation, return an `Order` object using the `Order.builder`. The `Order` should include orderPK, customer, jeep (model), color, engine, tire, options and price.

v) Produce a screenshot of the saveOrder method.

```java
@Override
public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
  BigDecimal price, List<Option> options) {

  SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);

  KeyHolder keyHolder = new GeneratedKeyHolder();
  jdbcTemplate.update(params.sql, params.source, keyHolder);

  Long orderPK = keyHolder.getKey().longValue();
  saveOptions(options, orderPK);

  return Order.builder()
      .orderPK(orderPK)
      .customer(customer)
      .model(jeep)
      .color(color)
      .engine(engine)
      .tire(tire)
      .options(options)
      .price(price)
      .build();
}
```

c) Run the integration test in `CreateOrderTest`. Produce a screenshot of the test method that shows the green JUnit status bar, the console output, and the test class.




```
2022-06-06 23:32:04.799  INFO 10928 --- [           main] c.p.jeep.controller.CreateOrderTest      : Starting CreateOrderTest using Java 17.0.2 on DESKTOP-V692HUJ with PID 10928 (started by dalto in C:\Users\dalto\Desktop\Developer
2022-06-06 23:32:04.800  INFO 10928 --- [           main] c.p.jeep.controller.CreateOrderTest      : The following 1 profile is active: "test"
2022-06-06 23:32:05.454  INFO 10928 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-06-06 23:32:05.474  INFO 10928 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 15 ms. Found 0 JDBC repository interfaces.
2022-06-06 23:32:06.139  INFO 10928 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 0 (http)
2022-06-06 23:32:06.155  INFO 10928 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-06-06 23:32:06.155  INFO 10928 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.62]
2022-06-06 23:32:06.304  INFO 10928 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-06-06 23:32:06.304  INFO 10928 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1481 ms
2022-06-06 23:32:07.992  INFO 10928 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-06-06 23:32:08.174  INFO 10928 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-06-06 23:32:08.376  INFO 10928 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 61315 (http) with context path ''
2022-06-06 23:32:08.386  INFO 10928 --- [           main] c.p.jeep.controller.CreateOrderTest      : Started CreateOrderTest in 3.952 seconds (JVM running for 5.051)
2022-06-06 23:32:09.084  INFO 10928 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-06-06 23:32:09.084  INFO 10928 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-06-06 23:32:09.085  INFO 10928 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2022-06-06 23:32:09.129  INFO 10928 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController       : Order = OrderRequest(customer=MAYNARD_TORBJORG, model=GLADIATOR, trim=Mojave, doors=4, color=EXT_SARGE_GREEN, engine=3_6_HYBRID, 
2022-06-06 23:32:09.144  INFO 10928 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepOrderService    : Order = OrderRequest(customer=MAYNARD_TORBJORG, model=GLADIATOR, trim=Mojave, doors=4, color=EXT_SARGE_GREEN, engine=3_6_HYBRID, 
2022-06-06 23:32:09.331  INFO 10928 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2022-06-06 23:32:09.334  INFO 10928 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

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

import static org.assertj.core.api.Assertions.assertThat;
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.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
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 org.springframework.validation.annotation.Validated;
import com.promineotech.jeep.entity.JeepModel;
import com.promineotech.jeep.entity.Order;

@Validated
@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"))

public class CreateOrderTest {

  @LocalServerPort
  private int serverPort;

```

```java
@Autowired
private TestRestTemplate restTemplate;

@Test
void testCreateOrderReturnsSuccess201() {

  String body = createOrderBody();
  String uri = String.format("http://localhost:%d/orders", serverPort);
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);
  HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
  ResponseEntity<Order> response = restTemplate.exchange(uri,HttpMethod.POST, bodyEntity, Order.class);

  assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

  assertThat(response.getBody()).isNotNull();

  Order order = response.getBody();
  assertThat(order.getCustomer().getCustomerId()).isEqualTo("MAYNARD_TORBJORG");
  assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GLADIATOR);
  assertThat(order.getModel().getTrimLevel()).isEqualTo("Mojave");
  assertThat(order.getModel().getNumDoors()).isEqualTo(4);
  assertThat(order.getColor().getColorId()).isEqualTo("EXT_SARGE_GREEN");
  assertThat(order.getEngine().getEngineId()).isEqualTo("3_6_HYBRID");
  assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
  assertThat(order.getOptions()).hasSize(6);
}

String createOrderBody() {
  return "{\n"
      + "  \"customer\":\"MAYNARD_TORBJORG\",\n"
      + "  \"model\":\"GLADIATOR\",\n"
      + "  \"trim\":\"Mojave\",\n"
      + "  \"doors\":4,\n"
      + "  \"color\":\"EXT_SARGE_GREEN\",\n"
      + "  \"engine\":\"3_6_HYBRID\",\n"
      + "  \"tire\":\"255_GOODYEAR\",\n"
      + "  \"options\":[\n"
      + "    \"DOOR_QUAD_4\",\n"
      + "    \"EXT_QUAD_ALUM_FRONT\",\n"
      + "    \"EXT_WARN_WINCH\",\n"
      + "    \"EXT_WARN_BUMPER_FRONT\",\n"
      + "    \"EXT_WARN_BUMPER_REAR\",\n"
      + "    \"EXT_ARB_COMPRESSOR\"\n"
      + "  ]\n"
      + "}\n"
      + "";
  }
}
```

Screenshots of Code:

```java
package com.promineotech.jeep;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class JeepSales {

  public static void main(String[] args) {
    SpringApplication.run(JeepSales.class, args);
  }
}

```

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

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@Slf4j
public class DefaultJeepOrderController implements JeepOrderController {

  @Autowired
  private JeepOrderService jeepOrderService;

  @Override
  public Order createOrder(OrderRequest orderRequest) {
    log.info("Order = {}",orderRequest);
    return jeepOrderService.createOrder(orderRequest);
  }
}

```

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

import java.util.List;

@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.controller;

import org.springframework.http.HttpStatus;

@RequestMapping("/orders")
public interface JeepOrderController {

  @Operation(
      summary = "Create an order for a Jeep",
      description = "Retruns the created Jeep",
      responses = {
          @ApiResponse(responseCode = "201",
              description = "The created Jeep 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 = "A Jeep component was not 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 = "orderRequest",
            required = true,
            description = "The order as JSON"),

      }
  )
  @PostMapping
  @ResponseStatus(code = HttpStatus.CREATED)
  Order createOrder(@RequestBody OrderRequest orderRequest);
}
```

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

import java.util.List;

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



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.dao;

import java.math.BigDecimal;

@Component
public class DefaultJeepOrderDao implements JeepOrderDao {
  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  @Override
  public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
    BigDecimal price, List<Option> options) {

    SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);

    KeyHolder keyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(params.sql, params.source, keyHolder);

    Long orderPK = keyHolder.getKey().longValue();
    saveOptions(options, orderPK);

    return Order.builder()
        .orderPK(orderPK)
        .customer(customer)
        .model(jeep)
        .color(color)
        .engine(engine)
        .tire(tire)
        .options(options)
        .price(price)
        .build();
  }

  private void saveOptions(List<Option> options, Long orderPK) {
    for(Option option : options) {
      SqlParams params = generateInsertSql(option, orderPK);
      jdbcTemplate.update(params.sql, params.source);
    }

  }

```

```
66
67  /**
68   *
69   * @param option
70   * @param orderPK
71   * @return
72   */
73  private SqlParams generateInsertSql(Option option, Long orderPK) {
74    SqlParams params = new SqlParams();
75
76    // @formatter:off
77    params.sql = ""
78        + "INSERT INTO order_options ("
79        + "option_fk, order_fk"
80        + ") VALUES ("
81        + ":option_fk, :order_fk"
82        + ")";
83    // @formatter:on
84
85    params.source.addValue("option_fk", option.getOptionPK());
86    params.source.addValue("order_fk", orderPK);
87
88    return params;
89  }
90
91  /**
92   *
93   * @param customer
94   * @param jeep
95   * @param color
96   * @param engine
97   * @param tire
98   * @param price
99   * @return
100  */
101 private SqlParams generateInsertSql(Customer customer, Jeep jeep, Color color,
102     Engine engine, Tire tire, BigDecimal price) {
103   // @formatter:off
104   String sql = ""
105       + "INSERT INTO orders ("
106       + "customer_fk, color_fk, engine_fk, tire_fk, model_fk, price"
107       + ") VALUES ("
108       + ":customer_fk, :color_fk, :engine_fk, :tire_fk, :model_fk, :price"
109       + ")";
110   // @formatter:on
111
```

```java
    SqlParams params = new SqlParams();

    params.sql = sql;
    params.source.addValue("customer_fk", customer.getCustomerPK());
    params.source.addValue("color_fk", color.getColorPK());
    params.source.addValue("engine_fk", engine.getEnginePK());
    params.source.addValue("tire_fk", tire.getTirePK());
    params.source.addValue("model_fk", jeep.getModelPK());
    params.source.addValue("price", price);

    return params;
  }

  /**
   *
   */
  @Override
  public List<Option> fetchOptions(List<String> optionIds) {
    if (optionIds.isEmpty()) {
      return new LinkedList<>();
    }

    Map<String, Object> params = new HashMap<>();

    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM options "
        + "WHERE option_id IN(";
    // @formatter:on

    for (int index = 0; index < optionIds.size(); index++) {
      String key = "option_" + index;
      sql += ":" + key + ", ";
      params.put(key, optionIds.get(index));
    }

    sql = sql.substring(0, sql.length() - 2);
    sql += ")";

    return jdbcTemplate.query(sql, params, new RowMapper<Option>() {
```

```java
153        @Override
154        public Option mapRow(ResultSet rs, int rowNum) throws SQLException {
155          // @formatter:off
156          return Option.builder()
157              .category(OptionType.valueOf(rs.getString("category")))
158              .manufacturer(rs.getString("manufacturer"))
159              .name(rs.getString("name"))
160              .optionId(rs.getString("option_id"))
161              .optionPK(rs.getLong("option_pk"))
162              .price(rs.getBigDecimal("price"))
163              .build();
164          // @formatter:on
165        }
166      });
167    }
168
169    /**
170     *
171     */
172    @Override
173    public Optional<Customer> fetchCustomer(String customerId) {
174      // @formatter:off
175      String sql = ""
176          + "SELECT * "
177          + "FROM customers "
178          + "WHERE customer_id = :customer_id";
179      // @formatter:on
180
181      Map<String, Object> params = new HashMap<>();
182      params.put("customer_id", customerId);
183
184      return Optional.ofNullable(
185          jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
186    }
187
188    /**
189     *
190     */
191    @Override
192    public Optional<Jeep> fetchModel(JeepModel model, String trim, int doors) {
193      // @formatter:off
194      String sql = ""
195          + "SELECT * "
196          + "FROM models "
197          + "WHERE model_id = :model_id "
198          + "AND trim_level = :trim_level "
199          + "AND num_doors = :num_doors";
200      // @formatter:on
```

```java
    Map<String, Object> params = new HashMap<>();
    params.put("model_id", model.toString());
    params.put("trim_level", trim);
    params.put("num_doors", doors);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new ModelResultSetExtractor()));
  }

  /**
   *
   */
  @Override
  public Optional<Color> fetchColor(String colorId) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM colors "
        + "WHERE color_id = :color_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("color_id", colorId);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new ColorResultSetExtractor()));
  }

  /**
   *
   */
  @Override
  public Optional<Engine> fetchEngine(String engineId) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM engines "
        + "WHERE engine_id = :engine_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("engine_id", engineId);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new EngineResultSetExtractor()));
  }
```

```java
249  /**
250   *
251   */
252  @Override
253  public Optional<Tire> fetchTire(String tireId) {
254    // @formatter:off
255    String sql = ""
256        + "SELECT * "
257        + "FROM tires "
258        + "WHERE tire_id = :tire_id";
259    // @formatter:on
260
261    Map<String, Object> params = new HashMap<>();
262    params.put("tire_id", tireId);
263
264    return Optional.ofNullable(
265        jdbcTemplate.query(sql, params, new TireResultSetExtractor()));
266  }
267
268  /**
269   *
270   * @author Promineo
271   *
272   */
273  class TireResultSetExtractor implements ResultSetExtractor<Tire> {
274    @Override
275    public Tire extractData(ResultSet rs) throws SQLException {
276      rs.next();
277
278      // @formatter:off
279      return Tire.builder()
280          .manufacturer(rs.getString("manufacturer"))
281          .price(rs.getBigDecimal("price"))
282          .tireId(rs.getString("tire_id"))
283          .tirePK(rs.getLong("tire_pk"))
284          .tireSize(rs.getString("tire_size"))
285          .warrantyMiles(rs.getInt("warranty_miles"))
286          .build();
287      // @formatter:on
288    }
289  }
290
```

```java
291 /**
292  *
293  * @author Promineo
294  *
295  */
296 class EngineResultSetExtractor implements ResultSetExtractor<Engine> {
297   @Override
298   public Engine extractData(ResultSet rs) throws SQLException {
299     rs.next();
300
301     // @formatter:off
302     return Engine.builder()
303         .description(rs.getString("description"))
304         .engineId(rs.getString("engine_id"))
305         .enginePK(rs.getLong("engine_pk"))
306         .fuelType(FuelType.valueOf(rs.getString("fuel_type")))
307         .hasStartStop(rs.getBoolean("has_start_stop"))
308         .mpgCity(rs.getFloat("mpg_city"))
309         .mpgHwy(rs.getFloat("mpg_hwy"))
310         .name(rs.getString("name"))
311         .price(rs.getBigDecimal("price"))
312         .sizeInLiters(rs.getFloat("size_in_liters"))
313         .build();
314     // @formatter:on
315   }
316 }
317
318 /**
319  *
320  * @author Promineo
321  *
322  */
323 class ColorResultSetExtractor implements ResultSetExtractor<Color> {
324   @Override
325   public Color extractData(ResultSet rs) throws SQLException {
326     rs.next();
327
328     // @formatter:off
329     return Color.builder()
330         .color(rs.getString("color"))
331         .colorId(rs.getString("color_id"))
332         .colorPK(rs.getLong("color_pk"))
333         .isExterior(rs.getBoolean("is_exterior"))
334         .price(rs.getBigDecimal("price"))
335         .build();
336     // @formatter:on
337   }
338 }
339
```

```java
340  /**
341   *
342   * @author Promineo
343   *
344   */
345  class ModelResultSetExtractor implements ResultSetExtractor<Jeep> {
346    @Override
347    public Jeep extractData(ResultSet rs) throws SQLException {
348      rs.next();
349
350      // @formatter:off
351      return Jeep.builder()
352          .basePrice(rs.getBigDecimal("base_price"))
353          .modelId(JeepModel.valueOf(rs.getString("model_id")))
354          .modelPK(rs.getLong("model_pk"))
355          .numDoors(rs.getInt("num_doors"))
356          .trimLevel(rs.getString("trim_level"))
357          .wheelSize(rs.getInt("wheel_size"))
358          .build();
359      // @formatter:on
360    }
361  }
362
363  /**
364   *
365   * @author Promineo
366   *
367   */
368  class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
369    @Override
370    public Customer extractData(ResultSet rs) throws SQLException {
371      rs.next();
372
373      // @formatter:off
374      return Customer.builder()
375          .customerId(rs.getString("customer_id"))
376          .customerPK(rs.getLong("customer_pk"))
377          .firstName(rs.getString("first_name"))
378          .lastName(rs.getString("last_name"))
379          .phone(rs.getString("phone"))
380          .build();
381      // @formatter:on
382
383    }
384  }
385
386  class SqlParams {
387    String sql;
388    MapSqlParameterSource source = new MapSqlParameterSource();
389  }
390 }
```

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

import java.math.BigDecimal;

@Service
@Slf4j
public class DefaultJeepSalesDao implements JeepSalesDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  @Override
  public List<Jeep> fetchJeeps(JeepModel model, String trim) {
    log.info("DAO: model = {}, trim = {}", model, trim);

    String sql = ""
        + "SELECT * "
        + "FROM models "
        + "WHERE model_id = :model_id AND trim_level = :trim_level";

    Map<String, Object> params = new HashMap<>();
    params.put("model_id", model.toString());
    params.put("trim_level", trim);

    return jdbcTemplate.query(sql, params,new RowMapper<>(){

      @Override
      public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {

        return Jeep.builder()
            .basePrice(new BigDecimal(rs.getString("base_price")))
            .modelId(JeepModel.valueOf(rs.getString("model_id")))
            .modelPK(rs.getLong("model_PK"))
            .numDoors(rs.getInt("num_doors"))
            .trimLevel(rs.getString("trim_level"))
            .wheelSize(rs.getInt("wheel_size"))
            .build();
      }
    });
  }
}
```

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

import java.math.BigDecimal;

public interface JeepOrderDao {
  List<Option> fetchOptions(List<String> optionIds);
  Optional<Customer> fetchCustomer(String customerId);
  Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
  Optional<Color> fetchColor(String colorId);
  Optional<Engine> fetchEngine(String engineId);
  Optional<Tire> fetchTire(String tireId);

  Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price, List<Option> options);

}
```

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

import java.util.List;

public interface JeepSalesDao {
  List<Jeep> fetchJeeps(JeepModel model, String trim);
}

```

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

import java.math.BigDecimal;

@Service
@Slf4j
public class DefaultJeepOrderService implements JeepOrderService {

  @Autowired
  private JeepOrderDao jeepOrderDao;

  @Transactional
  public Order createOrder(OrderRequest orderRequest) {
    log.info("Order = {}",orderRequest);

    Customer customer = getCustomer(orderRequest);
    Jeep jeep = getModel(orderRequest);
    Color color = getColor(orderRequest);
    Engine engine = getEngine(orderRequest);
    Tire tire = getTire (orderRequest);
    List<Option> options = getOption(orderRequest);

    BigDecimal price =
        jeep.getBasePrice()
        .add(color.getPrice())
        .add(engine.getPrice())
        .add(tire.getPrice());

    for(Option option : options) {
      price = price.add(option.getPrice());
    }

    return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
  }

  /**
   *
   * @param orderRequest
   * @return
   */
  private List<Option> getOption(OrderRequest orderRequest) {
    return jeepOrderDao.fetchOptions(orderRequest.getOptions());
  }

```

```java

/**
 *
 * @param orderRequest
 * @return
 */
private Tire getTire(OrderRequest orderRequest) {
  return jeepOrderDao.fetchTire(orderRequest.getTire())
      .orElseThrow(() -> new NoSuchElementException(
          "Tire with ID=" + orderRequest.getTire() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
private Engine getEngine(OrderRequest orderRequest) {
  return jeepOrderDao.fetchEngine(orderRequest.getEngine())
      .orElseThrow(() -> new NoSuchElementException(
          "Engine with ID=" + orderRequest.getEngine() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
private Color getColor(OrderRequest orderRequest) {
  return jeepOrderDao.fetchColor(orderRequest.getColor())
      .orElseThrow(() -> new NoSuchElementException(
          "Color with ID=" + orderRequest.getColor() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
private Jeep getModel(OrderRequest orderRequest) {
  return jeepOrderDao
      .fetchModel(orderRequest.getModel(), orderRequest.getTrim(),
          orderRequest.getDoors())
      .orElseThrow(() -> new NoSuchElementException("Model with ID="
          + orderRequest.getModel() + ", trim=" + orderRequest.getTrim()
          + orderRequest.getDoors() + " was not found"));
}

```

```java
  /**
   *
   * @param orderRequest
   * @return
   */
  private Customer getCustomer(OrderRequest orderRequest) {
    return jeepOrderDao.fetchCustomer(orderRequest.getCustomer())
        .orElseThrow(() -> new NoSuchElementException("Customer with ID="
            + orderRequest.getCustomer() + " was not found"));
  }
}
```

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

import java.util.List;

@Service
@Slf4j
public class DefaultJeepSalesService implements JeepSalesService {

  @Autowired
  private JeepSalesDao jeepSalesDao;


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

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

import com.promineotech.jeep.entity.Order;

public interface JeepOrderService {

  Order createOrder(OrderRequest orderRequest);

}
```

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

import java.util.List;

public interface JeepSalesService {

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

}
```

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

import static org.assertj.core.api.Assertions.assertThat;

@Validated
@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"))

public class CreateOrderTest {

  @LocalServerPort
  private int serverPort;

  @Autowired
  private TestRestTemplate restTemplate;

  @Test
  void testCreateOrderReturnsSuccess201() {

    String body = createOrderBody();
    String uri = String.format("http://localhost:%d/orders", serverPort);
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
    ResponseEntity<Order> response = restTemplate.exchange(uri,HttpMethod.POST, bodyEntity, Order.class);

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

    assertThat(response.getBody()).isNotNull();

    Order order = response.getBody();
    assertThat(order.getCustomer().getCustomerId()).isEqualTo("MAYNARD_TORBJORG");
    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GLADIATOR);
    assertThat(order.getModel().getTrimLevel()).isEqualTo("Mojave");
    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
    assertThat(order.getColor().getColorId()).isEqualTo("EXT_SARGE_GREEN");
    assertThat(order.getEngine().getEngineId()).isEqualTo("3_6_HYBRID");
    assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
    assertThat(order.getOptions()).hasSize(6);
  }

```

```
  String createOrderBody() {
    return "{\n"
        + "  \"customer\":\"MAYNARD_TORBJORG\",\n"
        + "  \"model\":\"GLADIATOR\",\n"
        + "  \"trim\":\"Mojave\",\n"
        + "  \"doors\":4,\n"
        + "  \"color\":\"EXT_SARGE_GREEN\",\n"
        + "  \"engine\":\"3_6_HYBRID\",\n"
        + "  \"tire\":\"255_GOODYEAR\",\n"
        + "  \"options\":[\n"
        + "    \"DOOR_QUAD_4\",\n"
        + "    \"EXT_QUAD_ALUM_FRONT\",\n"
        + "    \"EXT_WARN_WINCH\",\n"
        + "    \"EXT_WARN_BUMPER_FRONT\",\n"
        + "    \"EXT_WARN_BUMPER_REAR\",\n"
        + "    \"EXT_ARB_COMPRESSOR\"\n"
        + "  ]\n"
        + "}\n"
        + "";
  }
}
```

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

import static org.assertj.core.api.Assertions.assertThat;


@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;

  @Test
  void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {

    JeepModel model = JeepModel.WRANGLER;
    String trim = "Freedom";
    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);
  }

  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());

```

```java

    list.add(Jeep.builder()
        .modelId(JeepModel.WRANGLER)
        .trimLevel("Freedom")
        .numDoors(4)
        .wheelSize(17)
        .basePrice(new BigDecimal("39365.00"))
        .build());

    return list;
  }
}

```

Screenshots of Running Application:

```
2022-06-06 23:52:45.121  INFO 5820 --- [           main] c.p.jeep.controller.CreateOrderTest      : Starting CreateOrderTest using Java 17.0.2 on DESKTOP-V692HUJ
2022-06-06 23:52:45.122  INFO 5820 --- [           main] c.p.jeep.controller.CreateOrderTest      : The following 1 profile is active: "test"
2022-06-06 23:52:45.812  INFO 5820 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-06-06 23:52:45.832  INFO 5820 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 15 ms. Found 0 JDB
2022-06-06 23:52:46.512  INFO 5820 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 0 (http)
2022-06-06 23:52:46.522  INFO 5820 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-06-06 23:52:46.522  INFO 5820 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.62]
2022-06-06 23:52:46.657  INFO 5820 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-06-06 23:52:46.657  INFO 5820 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1505 m
2022-06-06 23:52:48.176  INFO 5820 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-06-06 23:52:48.362  INFO 5820 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-06-06 23:52:48.579  INFO 5820 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 64758 (http) with context path ''
2022-06-06 23:52:48.588  INFO 5820 --- [           main] c.p.jeep.controller.CreateOrderTest      : Started CreateOrderTest in 3.866 seconds (JVM running for 4.97
2022-06-06 23:52:49.338  INFO 5820 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-06-06 23:52:49.338  INFO 5820 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-06-06 23:52:49.340  INFO 5820 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2022-06-06 23:52:49.392  INFO 5820 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController       : Order = OrderRequest(customer=MAYNARD_TORBJORG, model=GLADIATO
2022-06-06 23:52:49.407  INFO 5820 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepOrderService    : Order = OrderRequest(customer=MAYNARD_TORBJORG, model=GLADIATO
2022-06-06 23:52:49.607  INFO 5820 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2022-06-06 23:52:49.609  INFO 5820 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

```
ESKTOP-V692HUJ with PID 5820 (started by dalto in C:\Users\dalto\Desktop\Developer Tools\Workspaces\Spring Boot\jeep-sales)

DEFAULT mode.
ms. Found 0 JDBC repository interfaces.



ext
leted in 1505 ms


ntext path ''
unning for 4.974)
erServlet'


 model=GLADIATOR, trim=Mojave, doors=4, color=EXT_SARGE_GREEN, engine=3_6_HYBRID, tire=255_GOODYEAR, options=[DOOR_QUAD_4,
 model=GLADIATOR, trim=Mojave, doors=4, color=EXT_SARGE_GREEN, engine=3_6_HYBRID, tire=255_GOODYEAR, options=[DOOR_QUAD_4,
```

```
EXT_QUAD_ALUM_FRONT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ARB_COMPRESSOR])
EXT_QUAD_ALUM_FRONT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ARB_COMPRESSOR])
```

Servers

http://localhost:8080 - Local server.

default-jeep-order-controller

POST /orders Create an order for a Jeep

Retruns the created Jeep

Parameters

Try it out

Name	Description
orderRequest * required ()	The order as JSON orderRequest

Request body required

application/json

Example Value | Schema

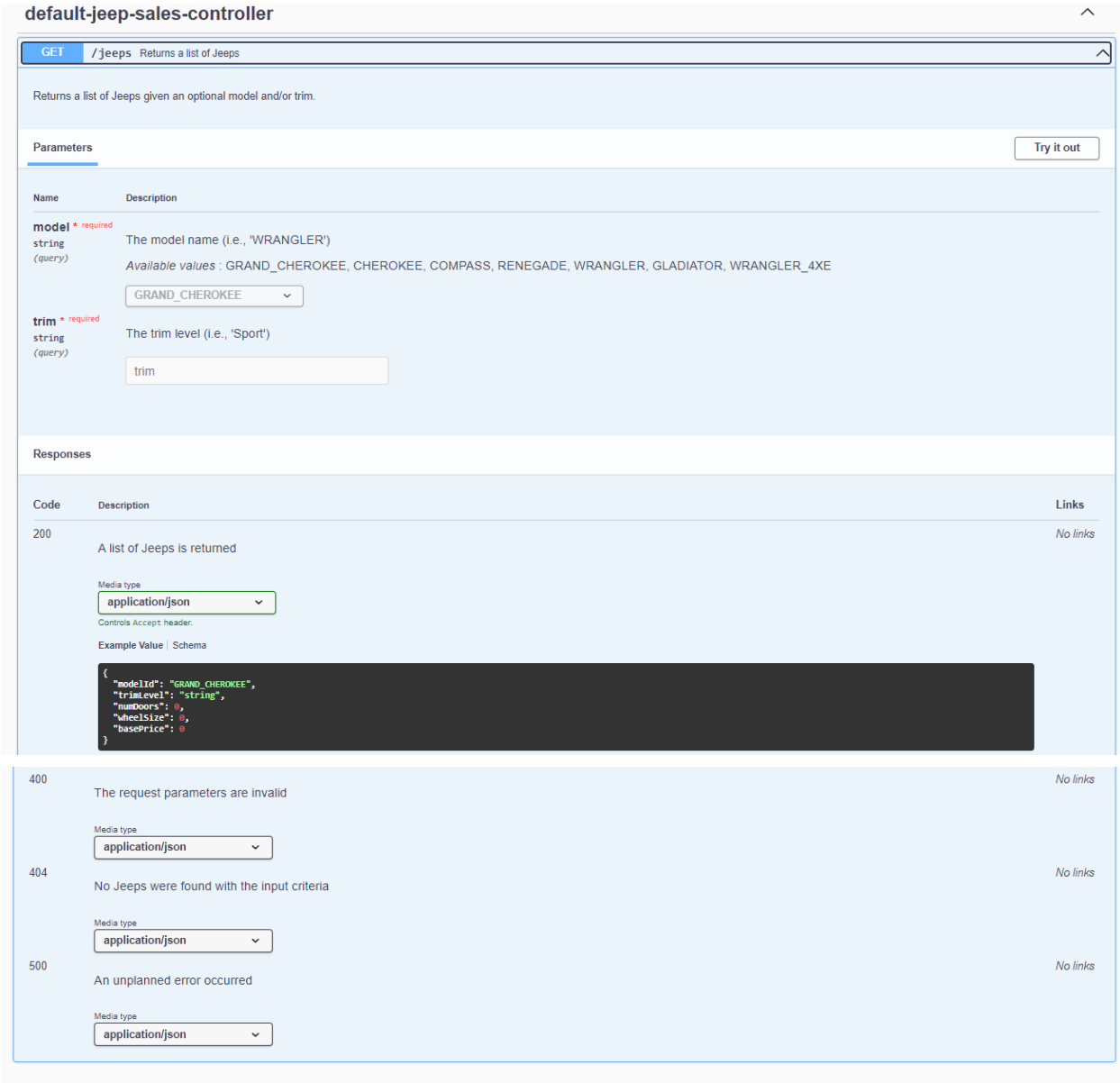

```
{
  "customer": "I8t ltv7jeEb0sG28CPcIQZCM_9XI9dJntsUKiEbQ_k1uCdRcdeuS5prrlU",
  "model": "GRAND_CHEROKEE",
  "trim": "f MCaJ Os tHfWYk1Nd4K1BDJQ",
  "doors": 4,
  "color": "5137El34nh_v",
  "engine": "0yCUaoBdYFasakK7Mugy1rFDAaag 4kSBJQC5izeNZpskDMmZuXJ3Z7DA7MlOV0DznNNymDm0e1i",
  "tire": "ev9gjJBZkEtN",
  "options": [
    "string"
  ]
}
```

Responses

Code	Description	Links

201	The created Jeep is returned
	
	Media type
	application/json
	Controls Accept header.
	
	Example Value | Schema
	
	{
	"modelId": "GRAND_CHEROKEE",
	"trimLevel": "string",
	"numDoors": 0,
	"wheelSize": 0,
	"basePrice": 0
	}
		No links

400	The request parameters are invalid Media type application/json	*No links*
404	A Jeep component was not found with the input criteria Media type application/json	*No links*
500	An unplanned error occurred Media type application/json	*No links*



URL to GitHub Repository:

[DaltonCash/PT-WK16 (github.com)]