


# THOMAS LE

## 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.jeepp.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. 

```

protected String createOrderBody(){
    return "{\n"
        + "  \"customer\": \"MORISON_LINA\", \n"
        + "  \"model\": \"WRANGLER\", \n"
        + "  \"trim\": \"Sport Altitude\", \n"
        + "  \"doors\": 4, \n"
        + "  \"color\": \"EXT_NACHO\", \n"
        + "  \"engine\": \"2_0_TURBO\", \n"
        + "  \"tire\": \"35_TOYO\", \n"
        + "  \"options\": [\n"
        + "    \"DOOR_QUAD_4\", \n"
        + "    \"EXT_AEV_LIFT\", \n"
        + "    \"EXT_WARN_WINCH\", \n"
        + "    \"EXT_WARN BUMPER_FRONT\", \n"
        + "    \"EXT_WARN BUMPER_REAR\", \n"
        + "    \"EXT_ARB_COMPRESSOR\" \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.jeepp.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);
```

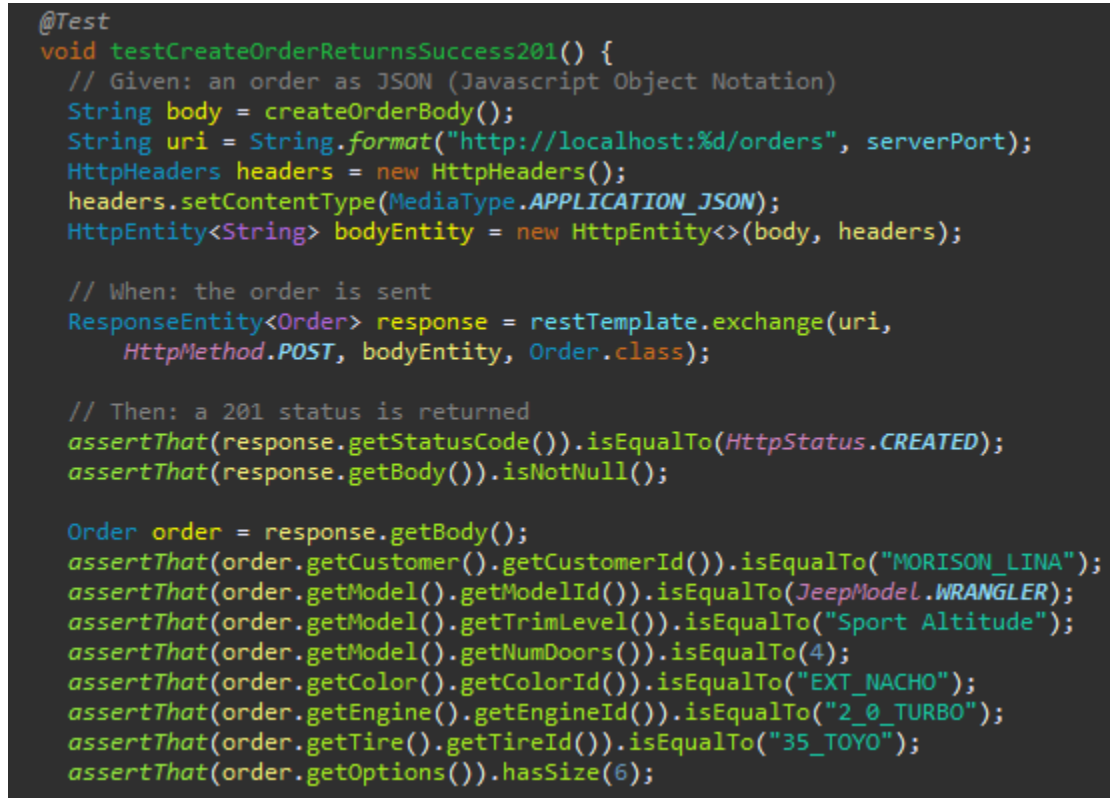
```

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() {
    // Given: an order as JSON (Javascript Object Notation)
    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);


    // When: the order is sent
    ResponseEntity<Order> response = restTemplate.exchange(uri,
        HttpMethod.POST, bodyEntity, Order.class);

    // Then: a 201 status is returned
    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);
}


```

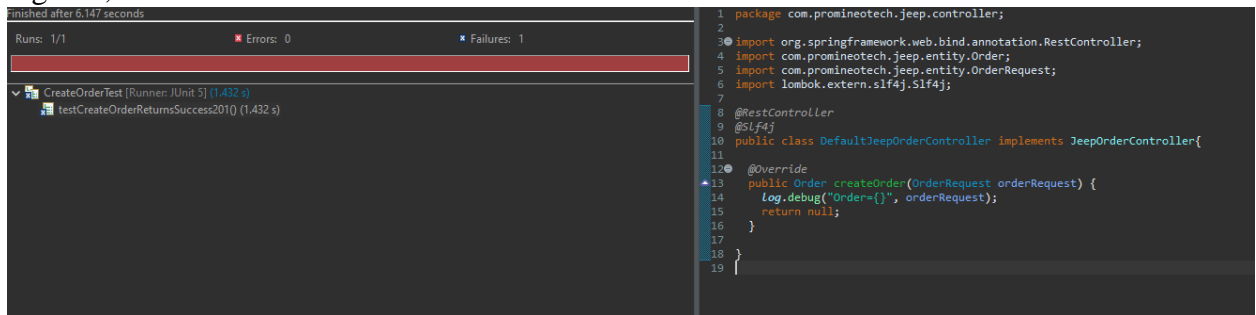
- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
  - a) Create a method in the interface to create an order (createOrder). It should return an object of type Order (see below). It should accept 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. 

```
1 package com.promineotech.jeepp.controller;
2
3 import java.util.List;
4
5 @RequestMapping("/orders")
6 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
7     @Server(url = "http://localhost:8080", description = "Local server.")})
8 public interface JeepOrderController {
9     // @formatter:off
10    @Operation(
11        summary = "Create an order for a Jeep",
12        description = "Returns the created Jeep",
13        responses = {
14            @ApiResponse(
15                responseCode = "201",
16                description = "The created Jeep is returned",
17                content = @Content(
18                    mediaType = "application/json",
19                    schema = @Schema(implementation = Order.class))),
20            @ApiResponse(
21                responseCode = "400",
22                description = "The request parameters are invalid",
23                content = @Content(mediaType = "application/json")),
24            @ApiResponse(
25                responseCode = "404",
26                description = "A Jeep component was not found with the input criteria",
27                content = @Content(mediaType = "application/json")),
28            @ApiResponse(
29                responseCode = "500",
30                description = "An unplanned error occurred.",
31                content = @Content(mediaType = "application/json"))
32        },
33        parameters = {
34            @Parameter(
35                name = "orderRequest",
36                required = true,
37                description = "The order as JSON")
38        }
39    )
40    @GetMapping
41    @ResponseStatus(code = HttpStatus.CREATED)
42    Order createOrder(@RequestBody OrderRequest orderRequest);
43 }
```


- 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. 



```
1 package com.promineotech.jeepp.controller;
2
3 import org.springframework.web.bind.annotation.RestController;
4 import com.promineotech.jeepp.entity.Order;
5 import com.promineotech.jeepp.entity.OrderRequest;
6 import lombok.extern.slf4j.Slf4j;
7
8 @RestController
9 @Slf4j
10 public class DefaultJeepOrderController implements JeepOrderController{
11
12     @Override
13     public Order createOrder(OrderRequest orderRequest) {
14         log.debug("Order={}", orderRequest);
15         return null;
16     }
17
18 }
19 }
```

- 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.

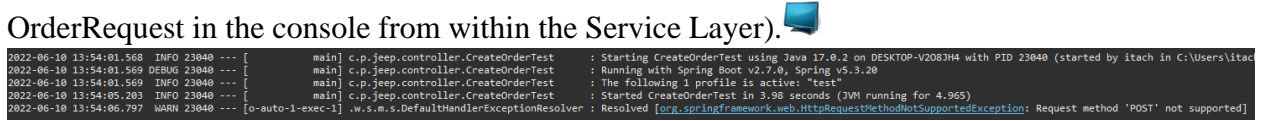
e) Produce a screenshot of this class with the annotations. 

```
1 package com.promineotech.jee.entity;
2
3 import java.util.List;
4 import javax.validation.constraints.Max;
5 import javax.validation.constraints.Min;
6 import javax.validation.constraints.NotNull;
7 import javax.validation.constraints.Pattern;
8 import javax.validation.constraints.Positive;
9 import org.hibernate.validator.constraints.Length;
10 import lombok.Data;
11
12 @Data
13 public class OrderRequest {
14     @NotNull
15     @Length(max = 30)
16     @Pattern(regexp = "[\\w\\s]*")
17     private String customer;
18
19     @NotNull
20     private JeepModel model;
21
22     @NotNull
23     @Length(max = 30)
24     @Pattern(regexp = "[\\w\\s]*")
25     private String trim;
26
27     @Positive
28     @Min(2)
29     @Max(4)
30     private int doors;
31
32     @NotNull
33     @Length(max = 30)
34     @Pattern(regexp = "[\\w\\s]*")
35     private String color;
36
37     @NotNull
38     @Length(max = 30)
39     @Pattern(regexp = "[\\w\\s]*")
40     private String engine;
41
42     @NotNull
43     @Length(max = 30)
44     @Pattern(regexp = "[\\w\\s]*")
45     private String tire;
46
47     private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String> options;
48
49 }
```

- 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).



```
2022-06-10 13:54:01.568 INFO 23040 --- [main] c.p.jee.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.2 on DESKTOP-V208JH4 with PID 23040 (started by itach in C:\Users\itach)
2022-06-10 13:54:01.569 DEBUG 23040 --- [main] c.p.jee.controller.CreateOrderTest : Running with Spring Boot v2.7.0, Spring v5.3.20
2022-06-10 13:54:01.569 INFO 23040 --- [main] c.p.jee.controller.CreateOrderTest : The following 1 profile is active: 'Test'
2022-06-10 13:54:05.203 INFO 23040 --- [main] c.p.jee.controller.CreateOrderTest : Started CreateOrderTest in 3.98 seconds (JVM running for 4.965)
2022-06-10 13:54:06.797 WARN 23040 --- [o-auto-1-exec-1] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.HttpRequestMethodNotSupportedException: Request method 'POST' not supported]
```

- 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. 



```
@Service
public class DefaultJeepOrderService implements JeepOrderService, JeepOrderDao{

    @Autowired
    private JeepOrderDao jeepOrderDao;

    @Transactional
    public Order createOrder(OrderRequest orderRequest) {

        Customer customer = getCustomer(orderRequest);

        Jeep model = 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, model, 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 SqlParameter 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. 

```
@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. 



```

1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
28
29 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
30 @ActiveProfiles("test")
31 @Sql(scripts = {
32     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
33     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
34     config = @SqlConfig(encoding = "utf-8"))
35
36 class CreateOrderTest {
37     @LocalServerPort
38     private int serverPort;
39
40     @Autowired
41     @Getter
42     private TestRestTemplate restTemplate;
43
44     @Test
45     void testCreateOrderReturnsSuccess201() {
46         // Given: an order as JSON (JavaScript Object Notation)
47         String body = createOrderBody();
48         String uri = String.format("http://localhost:%d/orders", serverPort);
49
50         HttpHeaders headers = new HttpHeaders();
51         headers.setContentType(MediaType.APPLICATION_JSON);
52
53         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
54
55         // When: the order is sent
56         ResponseEntity<Order> response = restTemplate.exchange(uri,
57             HttpMethod.POST, bodyEntity, Order.class);
58
59         // Then: a 201 status is returned
60         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
61
62         // And: the returned order is correct
63         assertThat(response.getBody()).isNotNull();
64
65         Order order = response.getBody();
66         assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
67         assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
68         assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
69         assertThat(order.getModel().getNumDoors()).isEqualTo(4);
70         assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
71         assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
72         assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
73         assertThat(order.getOptions()).hasSize(6);
74     }
75
76     protected String createOrderBody(){
77         return "{\n"
78             + "    \"customer\": \"MORISON_LINA\", \n"
79             + "    \"model\": \"WRANGLER\", \n"
80             + "    \"trim\": \"Sport Altitude\", \n"
81             + "    \"doors\": 4, \n"
82             + "    \"color\": \"EXT_NACHO\", \n"
83             + "    \"engine\": \"2_0_TURBO\", \n"
84             + "    \"tire\": \"35_TOYO\", \n"
85             + "    \"options\": [\n"
86             + "        \"DOOR_QUAD_4\", \n"
87             + "        \"EXT_AEV_LIFT\", \n"
88             + "        \"EXT_WARN_WINCH\", \n"
89             + "        \"EXT_WARN BUMPER_FRONT\", \n"
90             + "        \"EXT_WARN BUMPER_REAR\", \n"
91             + "        \"EXT_ARB_COMPRESSOR\" \n"
92             + "    ] \n"
93             + "}";
94     }
95
96 }

```

**URL to GitHub Repository:**

**<https://github.com/AkemiTCGyt/PromineoTechWeek16CodingAssignment>**