

Web API Design with Spring Boot Week 16 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/Bwcash/Web-API-Spring-Boot.git>


URL to Public Link of your Video: https://youtu.be/UIUrXX_X8M8

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.


- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.

2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-

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

Web API Design with Spring Boot Week 16 Coding Assignment

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 (EXT_COLORADO_RED)

ii) customer (ATTAWAY_HECKTOR)

iii) engine (2_0_TURBO)

iv) model (WRANGLER)

v) tire(s) (37_COOPER)

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!

EXT_MOPAR_TRAILER

INT_MOPAR_MATS

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.jeeep.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 1b. For example:

Web API Design with Spring Boot Week 16 Coding Assignment

```
{
  "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. 

Web API Design with Spring Boot Week 16 Coding Assignment

```
JeepSalesDao.java DefaultJeepSalesS... DefaultJeepSalesD... Jeep.java FetchJeepTest.java CreateOrderTestja... x
16 * @author Bryce Cash
17 *
18 */
19 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
20 @ActiveProfiles("test")
21 @Sql(scripts = {
22     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
23     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
24     config = @SqlConfig(encoding = "utf-8"))
25 class CreateOrderTest {
26     protected String createOrderBody() {
27         // @formatter:off
28         return "{\n"
29             + "  \"customer\": \"ATTAWAY HECKTOR\", \n"
30             + "  \"model\": \"WRANGLER\", \n"
31             + "  \"color\": \"EXT COLORADO RED\", \n"
32             + "  \"engine\": \"2.0 TURBO\", \n"
33             + "  \"tire\": \"37 COOPER\", \n"
34             + "  \"options\": [\n"
35             + "    \"EXT_MOPAR_TRAILER\", \n"
36             + "    \"INT_MOPAR_MATS\", \n"
37             + "  ] \n"
38             + "}";
39         // @formatter:on
40     }
41     @Test
42     void testCreateOrderReturnsSuccess201() {
43     }
44 }
45 }
```

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);
```


Web API Design with Spring Boot Week 16 Coding Assignment

- 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.jeeppromineotech.entity.Order` and not some other Order class.

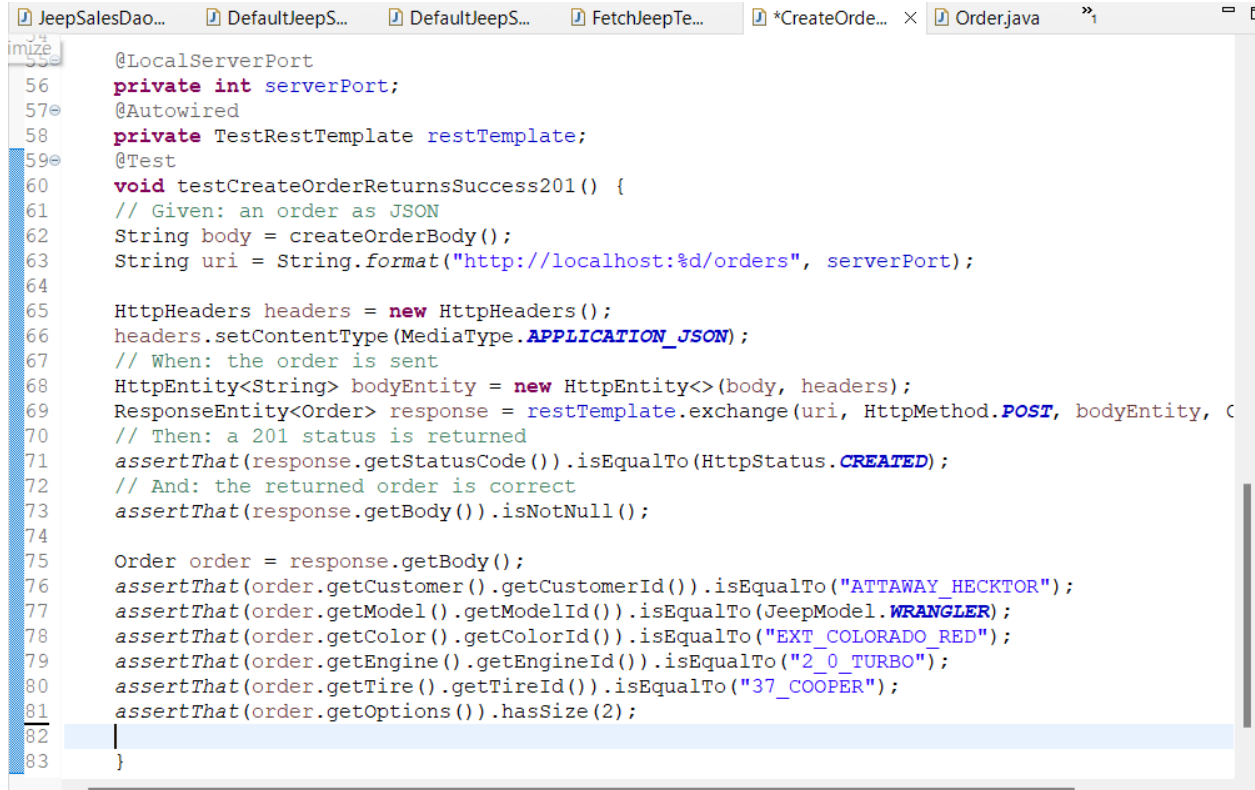
```
ResponseEntity<Order> response = restTemplate.exchange(uri,  
    HttpMethod.POST, bodyEntity, Order.class);
```

- 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. 

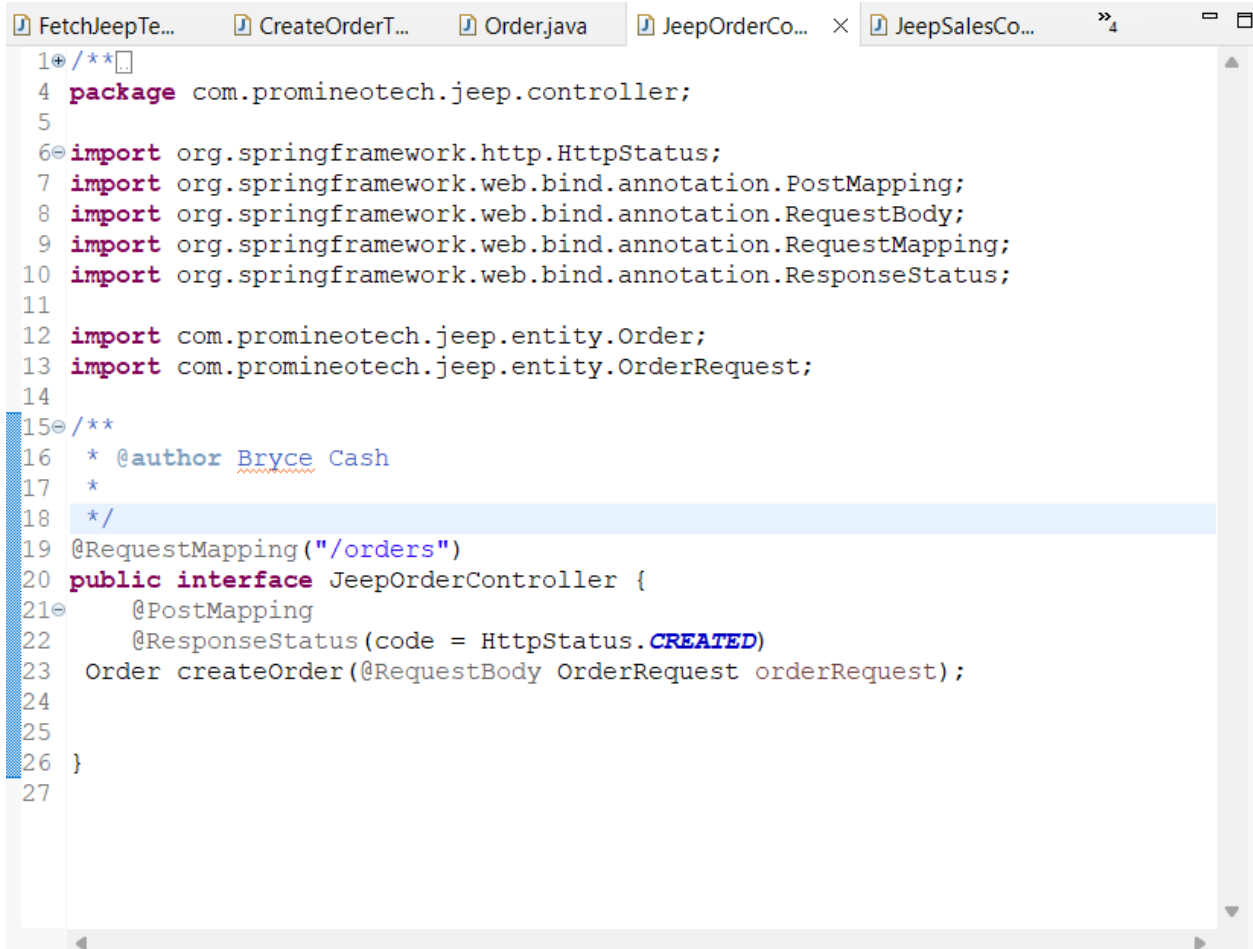
Web API Design with Spring Boot Week 16 Coding Assignment




```
56     @LocalServerPort
57     private int serverPort;
58     @Autowired
59     private TestRestTemplate restTemplate;
60     @Test
61     void testCreateOrderReturnsSuccess201() {
62         // Given: an order as JSON
63         String body = createOrderBody();
64         String uri = String.format("http://localhost:%d/orders", serverPort);
65
66         HttpHeaders headers = new HttpHeaders();
67         headers.setContentType(MediaType.APPLICATION_JSON);
68         // When: the order is sent
69         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
70         ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, C
71         // Then: a 201 status is returned
72         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
73         // And: the returned order is correct
74         assertThat(response.getBody()).isNotNull();
75
76         Order order = response.getBody();
77         assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
78         assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
79         assertThat(order.getColor().getColorId()).isEqualTo("EXT_COLORADO_RED");
80         assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
81         assertThat(order.getTire().getTireId()).isEqualTo("37_COOPER");
82         assertThat(order.getOptions()).hasSize(2);
83     }
```

- 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. 

Web API Design with Spring Boot Week 16 Coding Assignment



```
1+ /**
4 package com.promineotech.jeepp.controller;
5
6+ import org.springframework.http.HttpStatus;
7 import org.springframework.web.bind.annotation.PostMapping;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.ResponseStatus;
11
12 import com.promineotech.jeepp.entity.Order;
13 import com.promineotech.jeepp.entity.OrderRequest;
14
15+ /**
16  * @author Bryce Cash
17  *
18  */
19 @RequestMapping("/orders")
20 public interface JeepOrderController {
21+     @PostMapping
22     @ResponseStatus(code = HttpStatus.CREATED)
23     Order createOrder(@RequestBody OrderRequest orderRequest);
24
25
26 }
27
```

- 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. 

Web API Design with Spring Boot Week 16 Coding Assignment

The screenshot shows an IDE with a Java test class `CreateOrderTest.java` and its execution output. The test class is part of a package `com.promineotech.jeeptest` and contains a test method `testCreateOrderReturnsSuccess2010`. The test method calls `createOrderBody()` to create a body, then `restTemplate.exchange()` to send a POST request to `http://localhost:8080/orders`. It then asserts that the response status is `201` and that the response body is not null. The test method also asserts that the response body is an instance of `Order` and that its attributes match the expected values: `customer` is `ATTAWAY_HECKTOR`, `model` is `WRANGLER`, `color` is `EXT_COLORADO_RED`, `engine` is `2.0_TURBO`, `tire` is `37_COOPER`, and `options` has a size of 2.

```
62 String body = createOrderBody();
63 String uri = String.format("http://localhost:%d/orders", serverPort);
64
65 HttpHeaders headers = new HttpHeaders();
66 headers.setContentType(MediaType.APPLICATION_JSON);
67 // When: the order is sent
68 HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
69 ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
70 // Then: a 201 status is returned
71 assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
72 // And: the returned order is correct
73 assertThat(response.getBody()).isNotNull();
74
75 Order order = response.getBody();
76 assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
77 assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
78 assertThat(order.getColor().getColorId()).isEqualTo("EXT_COLORADO_RED");
79 assertThat(order.getEngine().getEngineId()).isEqualTo("2.0_TURBO");
80 assertThat(order.getTire().getTireId()).isEqualTo("37_COOPER");
81 assertThat(order.getOptions().hasSize(2));
82
83 }
```

The output shows the test results and the application logs. The test results show that the test `testCreateOrderReturnsSuccess2010` passed. The application logs show the startup of the application, including the initialization of the Spring framework, the Tomcat web server, and the HikariPool database connection pool.


- 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.

Web API Design with Spring Boot Week 16 Coding Assignment

- 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. 

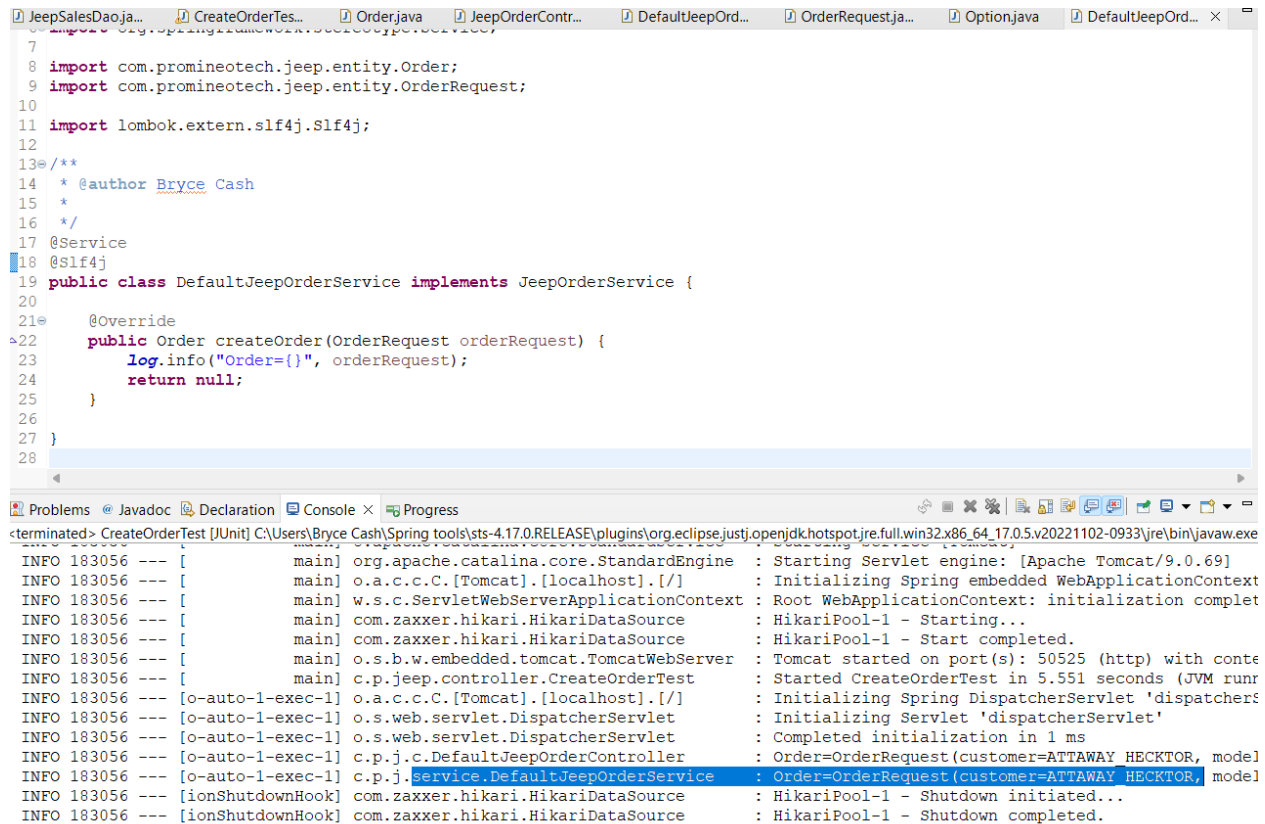


```
12
13 import com.fasterxml.jackson.annotation.JsonIgnore;
14
15 import lombok.Data;
16
17 @Data
18 public class OrderRequest {
19
20     @NotNull
21     @Length(max = 30)
22     @Pattern(regexp = "[\\w\\s]*")
23     private String customer;
24
25     @NotNull
26     private JeepModel model;
27
28     @NotNull
29     @Length(max = 30)
30     @Pattern(regexp = "[\\w\\s]*")
31     private String trim;
32
33     @Positive
34     @Min(2)
35     @Max(4)
36     private int doors;
37
38     @NotNull
39     @Length(max = 30)
40     @Pattern(regexp = "[\\w\\s]*")
41     private String color;
42
43     @NotNull
44     @Length(max = 30)
45     @Pattern(regexp = "[\\w\\s]*")
46     private String engine;
47
48     @NotNull
49     @Length(max = 30)
50     @Pattern(regexp = "[\\w\\s]*")
51     private String tire;
52
53     private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*")String> options;
54
```

- 2) 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.

Web API Design with Spring Boot Week 16 Coding Assignment

- 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).



The screenshot shows an IDE with several tabs open: `JeepSalesDao.java`, `CreateOrderTest.java`, `Order.java`, `JeepOrderController.java`, `DefaultJeepOrderService.java`, `OrderRequest.java`, `Option.java`, and `DefaultJeepOrderDao.java`. The `DefaultJeepOrderService.java` tab is active, showing the following code:

```
7
8 import com.promineotech.jeep.entity.Order;
9 import com.promineotech.jeep.entity.OrderRequest;
10
11 import lombok.extern.slf4j.Slf4j;
12
13 /**
14  * @author Bryce Cash
15  *
16  */
17 @Service
18 @Slf4j
19 public class DefaultJeepOrderService implements JeepOrderService {
20
21     @Override
22     public Order createOrder(OrderRequest orderRequest) {
23         log.info("Order={}", orderRequest);
24         return null;
25     }
26
27 }
28
```

The console window at the bottom shows the output of running `CreateOrderTest`. The log line `Order=OrderRequest(customer=ATTAWAY HECKTOR, model=)` is highlighted, indicating that the `createOrder` method in the service layer is being called and logging the `OrderRequest` object.

- 3) 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.

Web API Design with Spring Boot Week 16 Coding Assignment

- b) Add the `@Component` annotation to the DAO implementation class.
- 4) 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.
- 5) ***** 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.**
- 6) 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`.

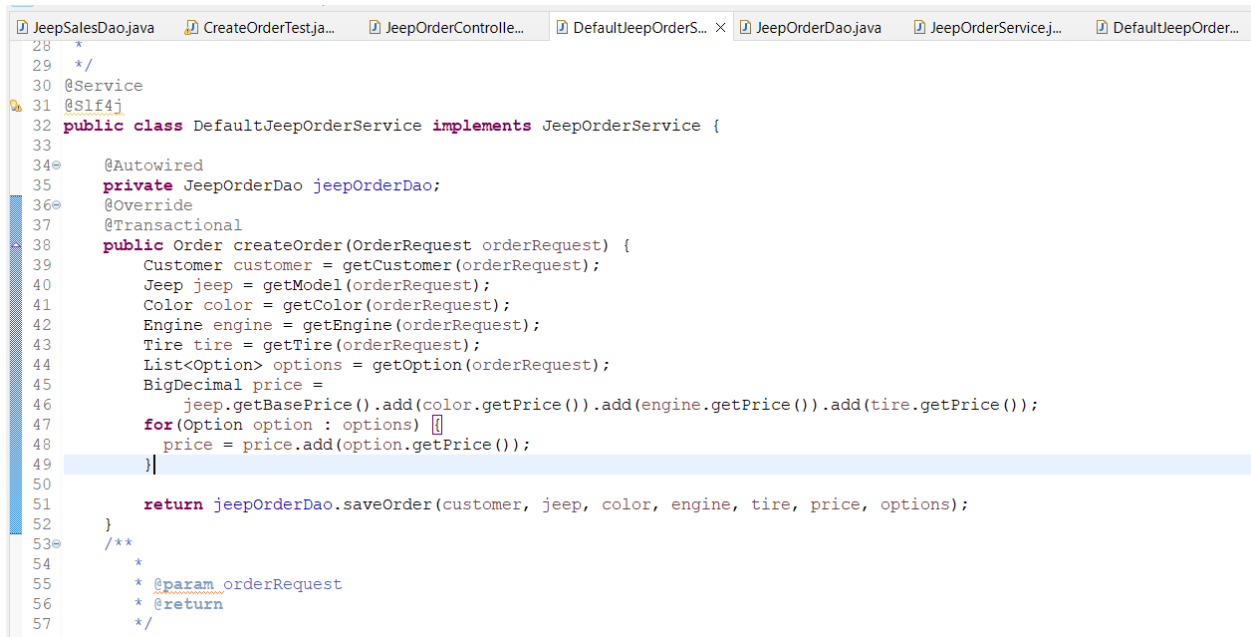
- 7) Copy the *contents* of the file `DefaultJeepOrderService.source` *into* `DefaultJeepOrderService.java`. Add the source after the `createOrder()` method, but *inside* the class body. The source file is found in the Source folder of the supplied project resources.

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

- 8) 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.
- 9) 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);
```

## Web API Design with Spring Boot Week 16 Coding Assignment

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



```
28 *
29 */
30 @Service
31 @Slf4j
32 public class DefaultJeepOrderService implements JeepOrderService {
33
34 @Autowired
35 private JeepOrderDao jeepOrderDao;
36 @Override
37 @Transactional
38 public Order createOrder(OrderRequest orderRequest) {
39 Customer customer = getCustomer(orderRequest);
40 Jeep jeep = getModel(orderRequest);
41 Color color = getColor(orderRequest);
42 Engine engine = getEngine(orderRequest);
43 Tire tire = getTire(orderRequest);
44 List<Option> options = getOption(orderRequest);
45 BigDecimal price =
46 jeep.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPrice());
47 for(Option option : options) {
48 price = price.add(option.getPrice());
49 }
50
51 return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
52 }
53 /**
54 *
55 * @param orderRequest
56 * @return
57 */
58 }
```

- 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:

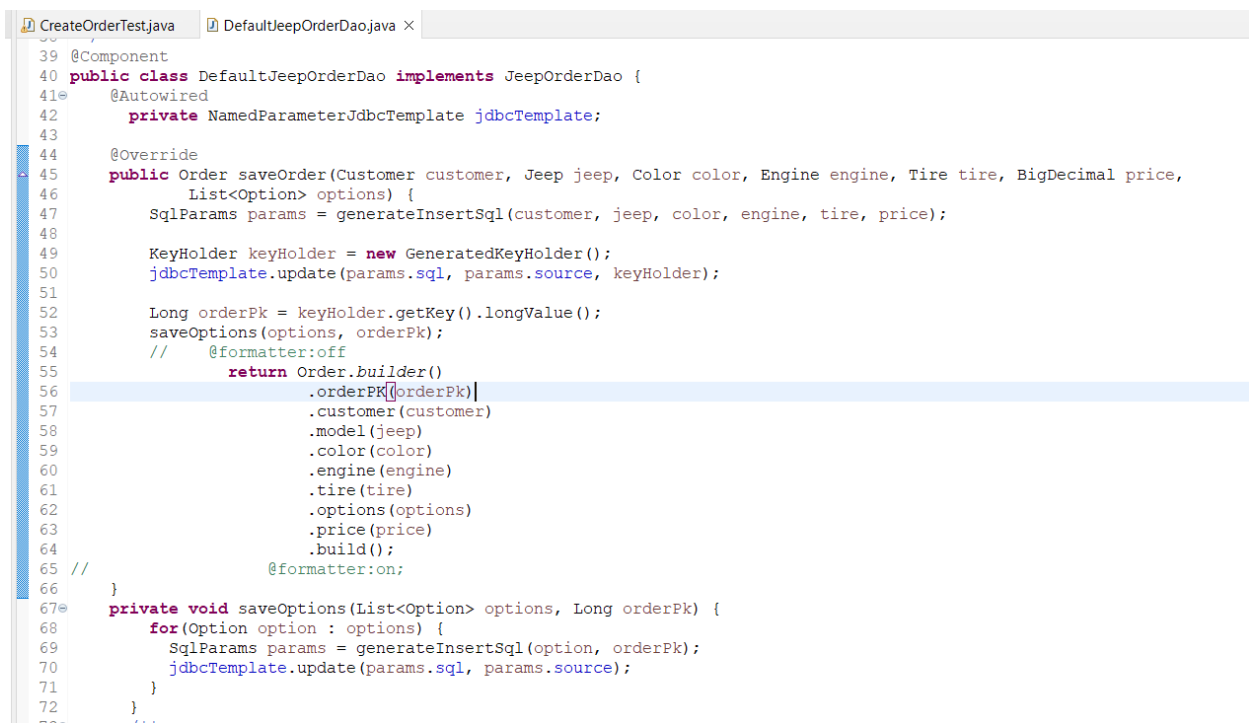
```
private void saveOptions(List<Option> options, Long orderPK)
```

## Web API Design with Spring Boot Week 16 Coding Assignment


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`.

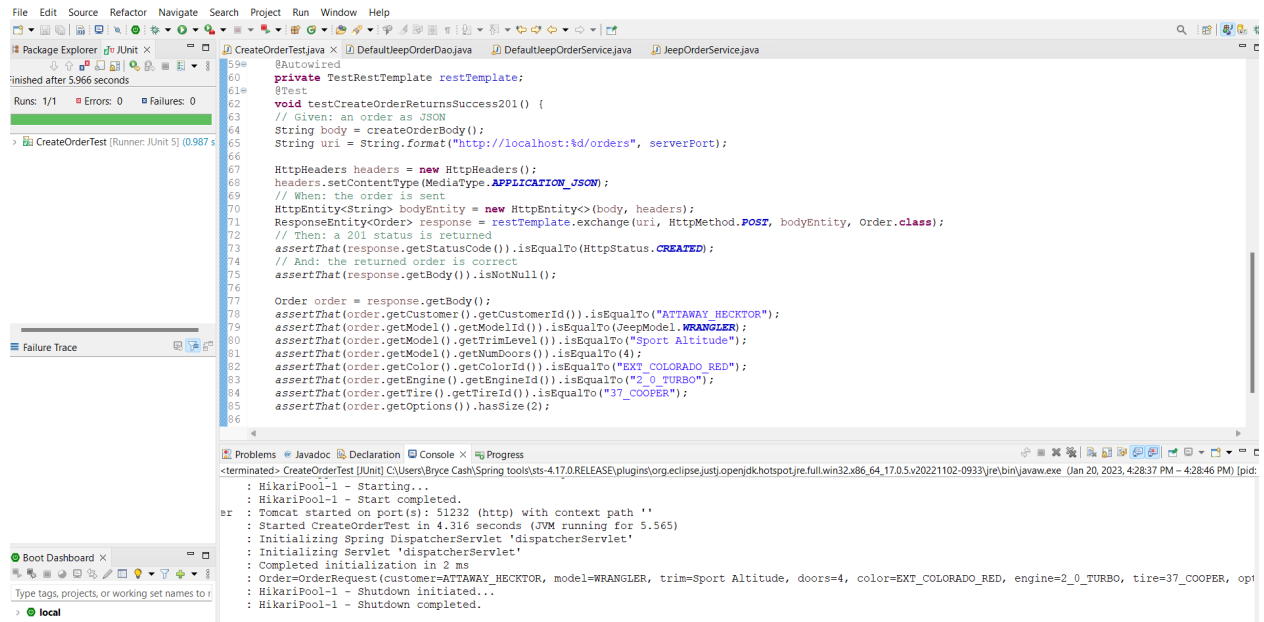
v) Produce a screenshot of the `saveOrder` method. 



```
39 @Component
40 public class DefaultJeepOrderDao implements JeepOrderDao {
41 @Autowired
42 private NamedParameterJdbcTemplate jdbcTemplate;
43
44 @Override
45 public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price,
46 List<Option> options) {
47 SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
48
49 KeyHolder keyHolder = new GeneratedKeyHolder();
50 jdbcTemplate.update(params.sql, params.source, keyHolder);
51
52 Long orderPk = keyHolder.getKey().longValue();
53 saveOptions(options, orderPk);
54 // @formatter:off
55 return Order.builder()
56 .orderPK(orderPk)
57 .customer(customer)
58 .model(jeep)
59 .color(color)
60 .engine(engine)
61 .tire(tire)
62 .options(options)
63 .price(price)
64 .build();
65 // @formatter:on;
66 }
67 private void saveOptions(List<Option> options, Long orderPk) {
68 for(Option option : options) {
69 SqlParams params = generateInsertSql(option, orderPk);
70 jdbcTemplate.update(params.sql, params.source);
71 }
72 }
73 }
```

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. 

# Web API Design with Spring Boot Week 16 Coding Assignment



The screenshot displays the Eclipse IDE interface. The main editor shows the file `CreateOrderTest.java` with the following code:

```
59 @Autowired
60 private TestRestTemplate restTemplate;
61 @Test
62 void testCreateOrderReturnsSuccess201() {
63 // Given: an order as JSON
64 String body = createOrderBody();
65 String url = String.format("http://localhost:%d/orders", serverPort);
66
67 HttpHeaders headers = new HttpHeaders();
68 headers.setContentType(MediaType.APPLICATION_JSON);
69 // When: the order is sent
70 HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
71 ResponseEntity<Order> response = restTemplate.exchange(url, HttpMethod.POST, bodyEntity, Order.class);
72 // Then: a 201 status is returned
73 assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
74 // And: the returned order is correct
75 assertThat(response.getBody()).isNotNull();
76
77 Order order = response.getBody();
78 assertThat(order.getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
79 assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
80 assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
81 assertThat(order.getModel().getNumDoors()).isEqualTo(4);
82 assertThat(order.getColor().getColorId()).isEqualTo("EXT_COLORADO_RED");
83 assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
84 assertThat(order.getTire().getTireId()).isEqualTo("37_COOPER");
85 assertThat(order.getOptions()).hasSize(2);
86 }
```

The left sidebar shows the Package Explorer with the project structure. The bottom console shows the following output:

```
<terminated> CreateOrderTest [JUnit] C:\Users\Byrce\Cash\Spring\tools\sts-4.17.0.RELEASE\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (Jan 20, 2023, 4:28:37 PM - 4:28:46 PM) [pid:
: HikariPool-1 - Starting...
: HikariPool-1 - Start completed.
: Tomcat started on port(s): 51232 (http) with context path ''
: Started CreateOrderTest in 4.316 seconds (JVM running for 5.565)
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 2 ms
: Order=OrderRequest(customer=ATTAWAY_HECKTOR, model=WRANGLER, trim=Sport Altitude, doors=4, color=EXT_COLORADO_RED, engine=2_0_TURBO, tire=37_COOPER, op
: HikariPool-1 - Shutdown initiated...
: HikariPool-1 - Shutdown completed.
```