Lab Week 04: BITP 3123 Distributed Application Development

Development of Web Service Provider Application

A Part 2 of Lab Week 04

By

Emaliana Kasmuri

FTMK, UTeM

Table of Contents

1 Learning Outcomes

At the end of this lab exercise, the student should be able to: -

1. Implement REST application that complies with MVC pattern.
2. Execute unit tests for the REST application using Postman.

2 Software Tools for the Lab Exercise

This lab exercise requires the following software tools to implement the case study.

1. Eclipse
2. Postman
3. Internet connection

3 An Overview of the Case Study

tba

4 Implementation of the Lab Exercise

There are three implementation processes in the lab exercises. The processes are:-

1. Model-Controller layer implementation
2. Model layer implementation
3. Controller layer implementation

The relationship between these processes is shown in Figure 4.1.

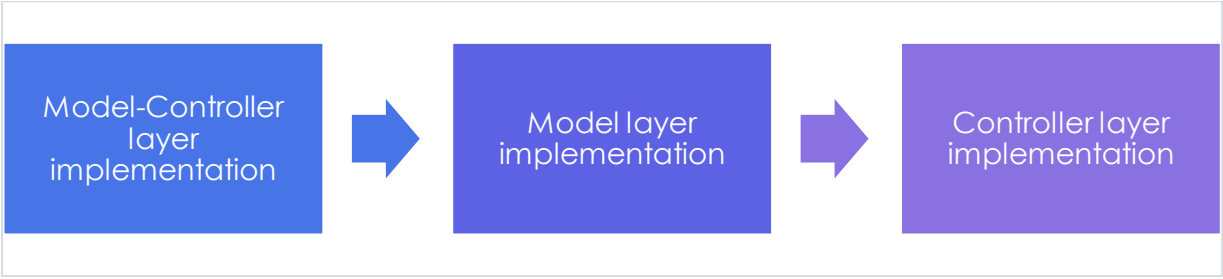

Figure 4.1: Relation of processes for the lab exercises

5 Model-Controller Layer Implementation

1. Open **Eclipse**.
2. Expand **my.edu.utem.ftmk.dad.restorderapp**.
3. Create the packages shown in Figure 5.1.

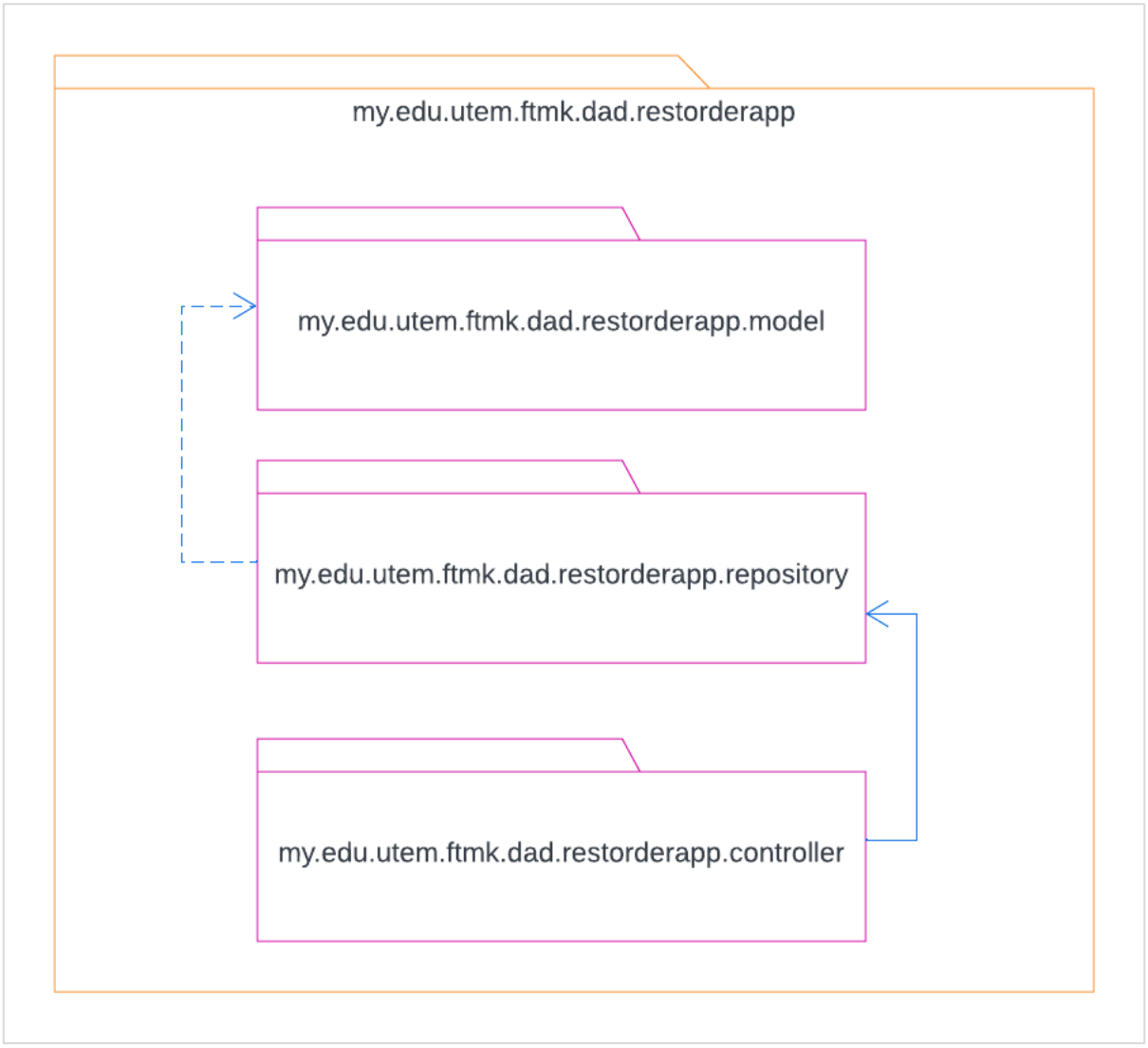

Figure 5.1: Layer design for restorderapp application

6 Model Layer Implementation

There are two activities in the model layer implementation. The activities are:-

1. Define model class.
2. Define repository interface.

The relationship between these activities is shown in Figure 6.1.

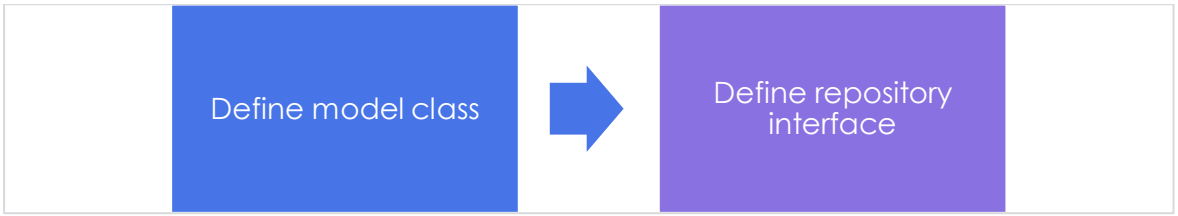

Figure 6.1: Relation of activities in the model layer implementation

6.1 Define a Class Model

1. The class in Figure 6.2 represent an entity for an order type. Define the class.

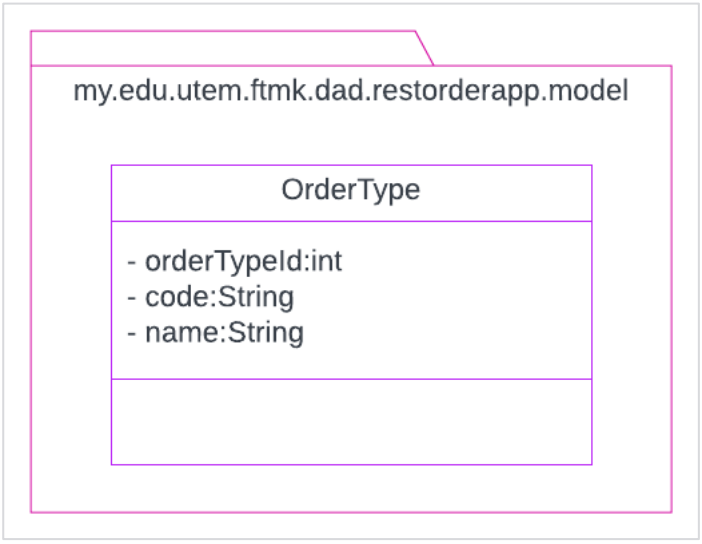

Figure 6.2: Class OrderType

2. Click **Source** from the Eclipse menu bar.
3. Then select **Generate Getters and Setters**. A window with a similar name, as shown in Figure 6.3 will be displayed.

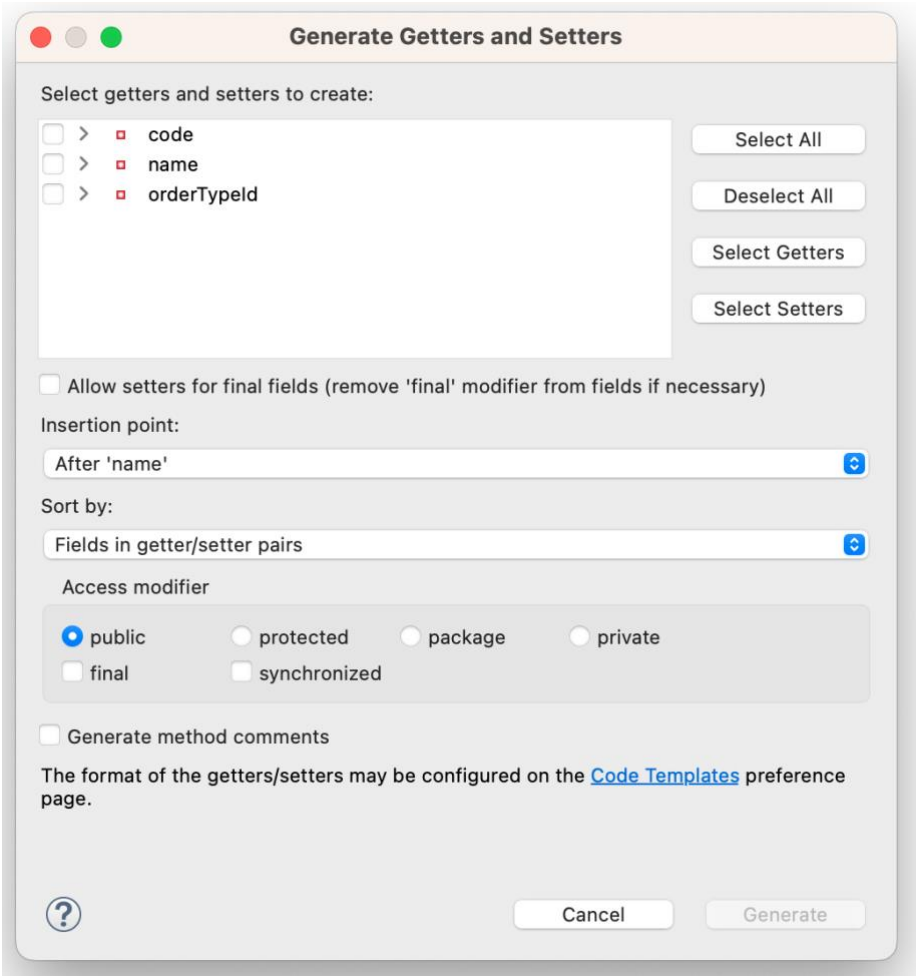

Figure 6.3: A window to generate getters and setters

4. Click the **Select All** button. All attributes will be selected, and the **Generate** button will be activated.
5. Then, click the **Generate** button. The getter and setter methods will be generated in the `OrderType` class.

6. Annotate the class with `@Entity` and `@Table(name="orderType")` before the class declaration, as shown in Figure 6.4. The name of the `@Table` must be the same as the table in the database used this application.

```
@Entity
@Table(name = "ordertype")
public class OrderType {
```

Figure 6.4: Annotating class OrderType

7. Import `@Entity` and `@Table` from `jakarta.persistence`.
8. Annotate `orderTypeId` with `@Column (name="orderTypeId")` as shown in Figure 6.5. The name of `@Column` refers to the column named orderTypeId in the table OrderType.

```
@Column (name="orderTypeId")
private int orderTypeId;
```

Figure 6.5: Annotating attribute orderTypeId as column

9. Import `@Column` from `jakarta.persistence`.
10. Repeat step 8 for other private attributes to map it with the equivalent field in table OrderType.

11. Annotate `orderTypeId` with `@Id` and `@GeneratedValue(strategy=GenerationType.IDENTITY)` as shown in Figure 6.6. The annotation represents the field as the primary key and the value of the key auto incremental.

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column (name="orderTypeId")
private int orderTypeId;
```

Figure 6.6: Annotating attribute orderTypeId as primary key

12. Import `@Id` and `@GeneratedValue` from `jakarta.persistence`.
13. Add comments to describe the class.
14. Save the class.

6.2 Define a Repository Interface

1. Figure 6.7 shows a repository interface for the class `OrderType`.

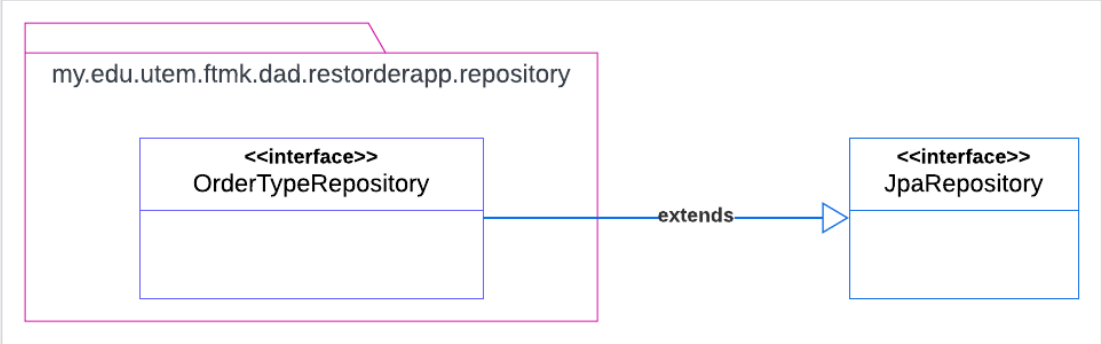

Figure 6.7: Interface OrderTypeRepository

2. Right-click on package `my.edu.utem.ftmk.dad.restorderapp.repository`.
3. Then, select `Interface,` as shown in Figure 6.8.

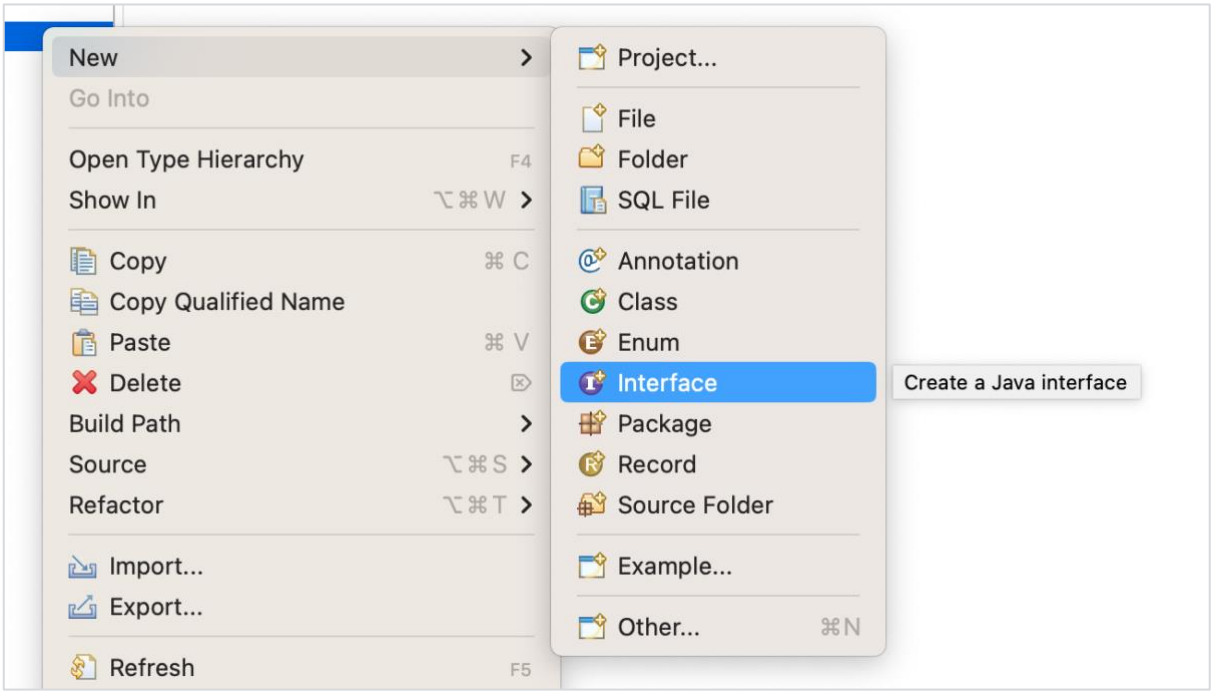

Figure 6.8: Menu to create a new interface

4. A new window named **New Java Interface** will be displayed. Name the interface as `OrderTypeRepository`, as shown in Figure 6.9.

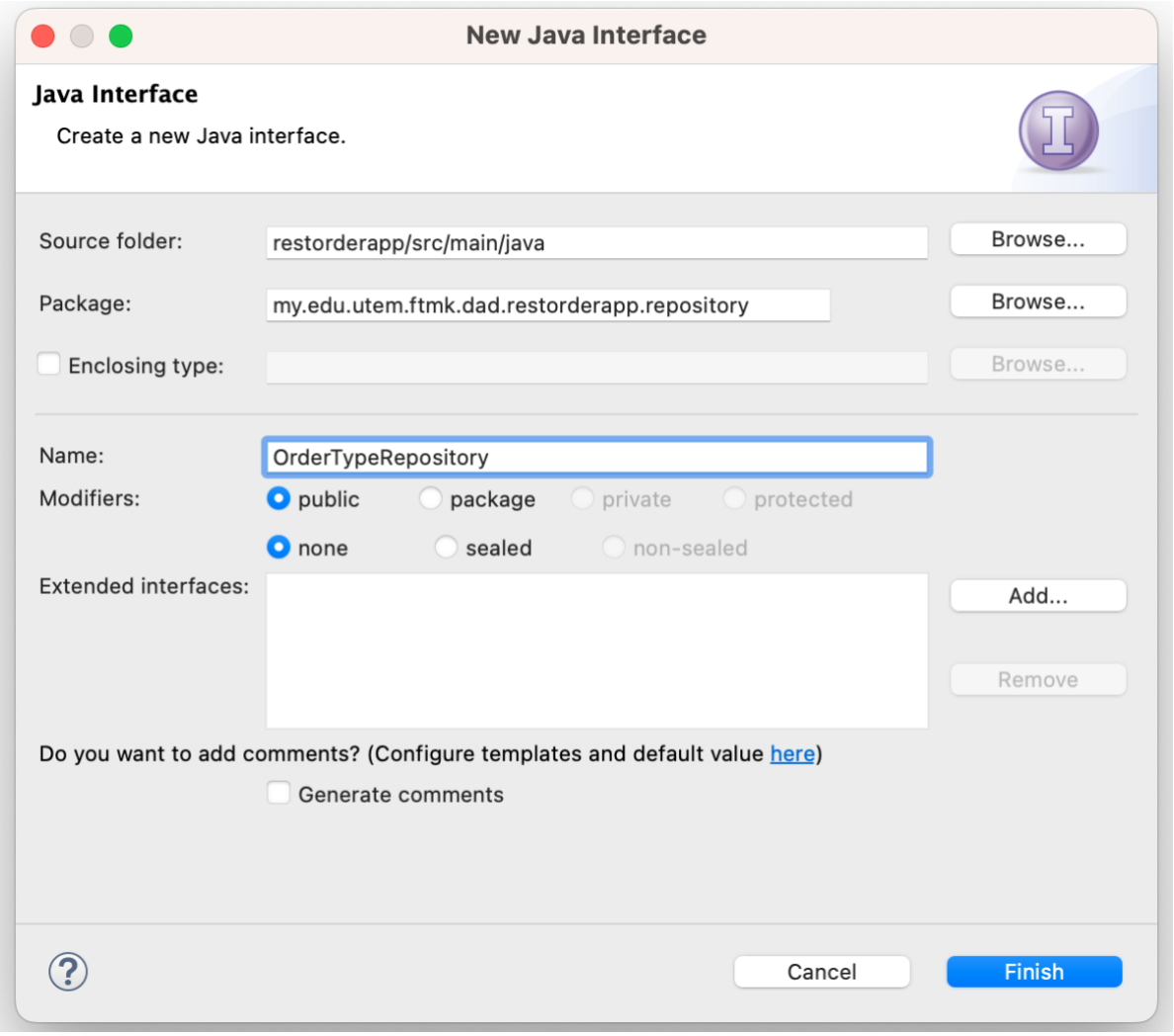

Figure 6.9: A window to create a new interface

5. Click the **Add** button. A new window named **Extended Interface Selection** will be displayed, as shown in Figure 6.10.

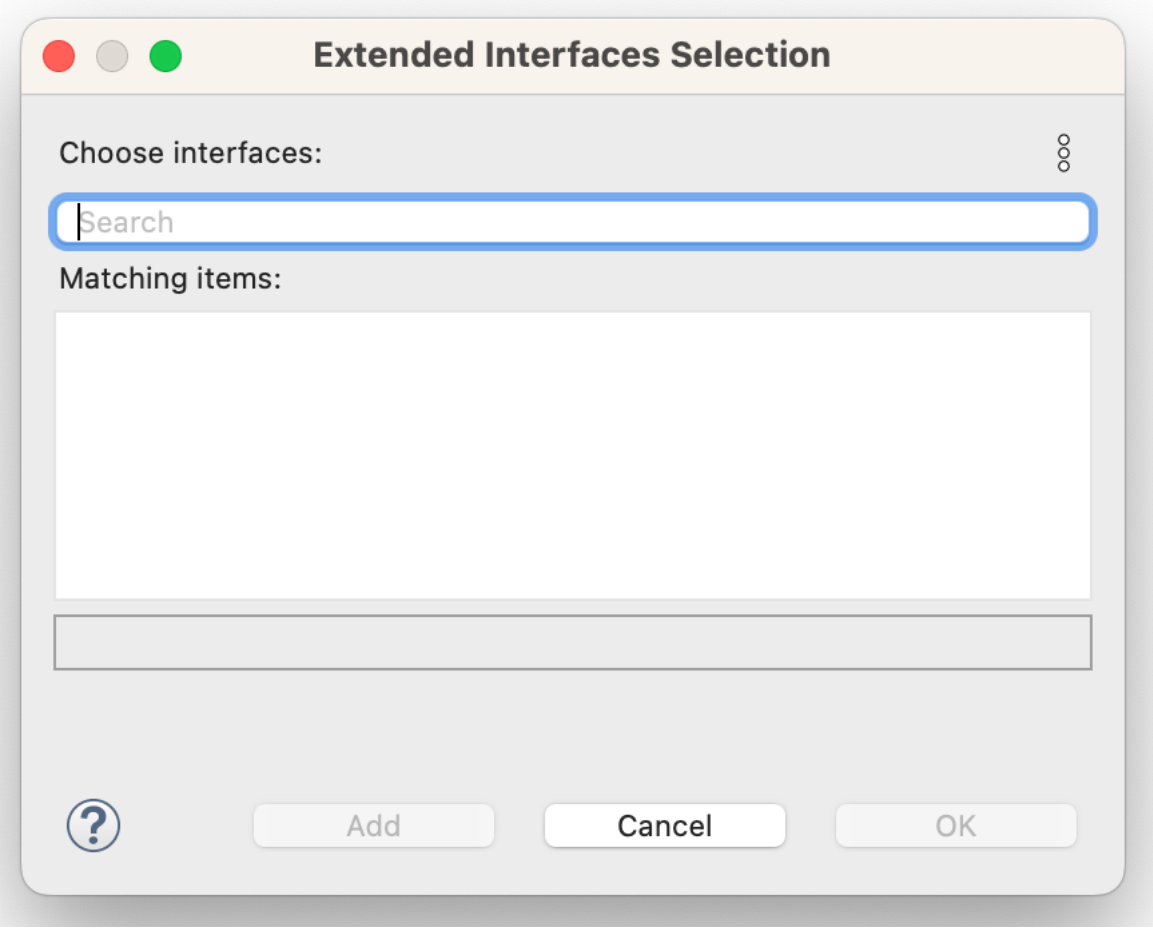

Figure 6.10: A window to select an extended interface

6. Type JpaRepo in **Choose interfaces** text box. A list of matching items will appear.
7. Select **JpaRepository** from **org.springframework.data.jpa.repository** as shown in Figure 6.11.

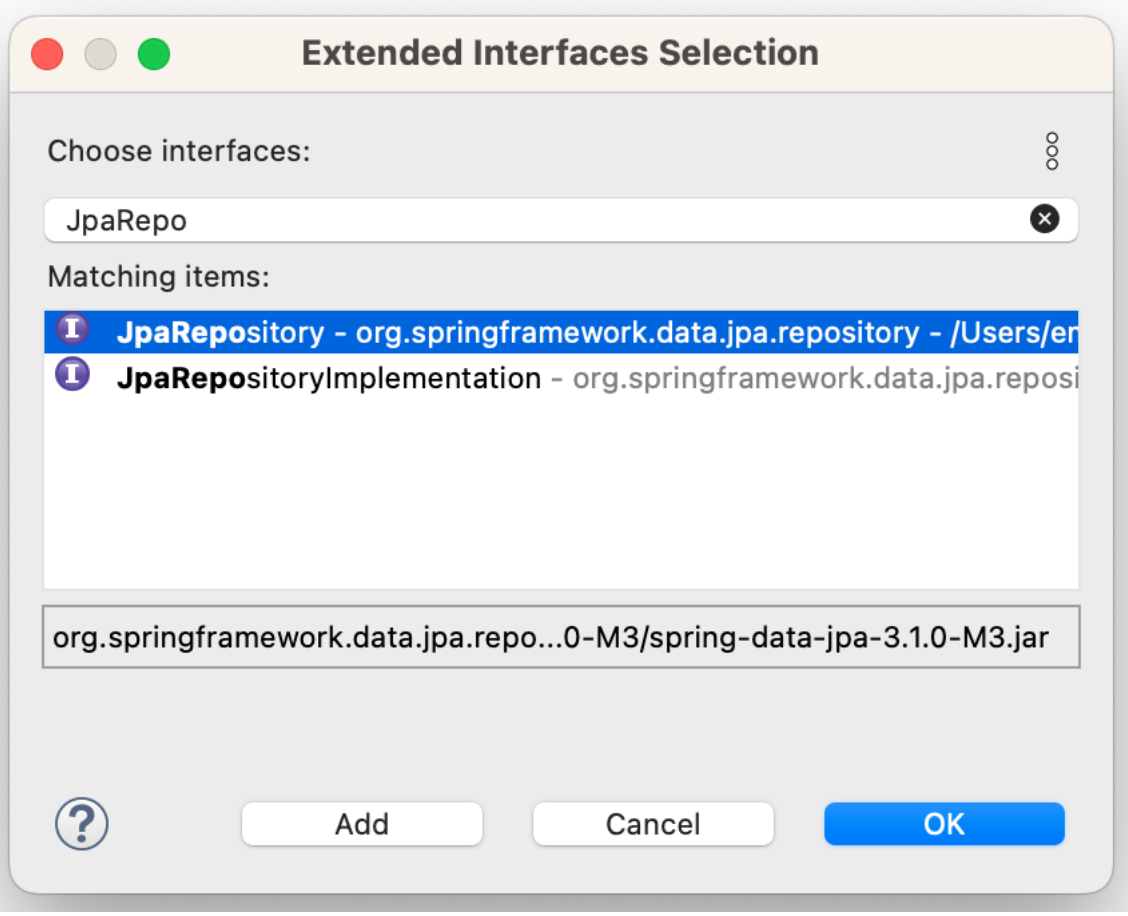

Figure 6.11: Search result from JpaRepo

8. Then click the **OK** button. The selected class will appear in the **Extended Interfaces** list, as shown in Figure 6.12.

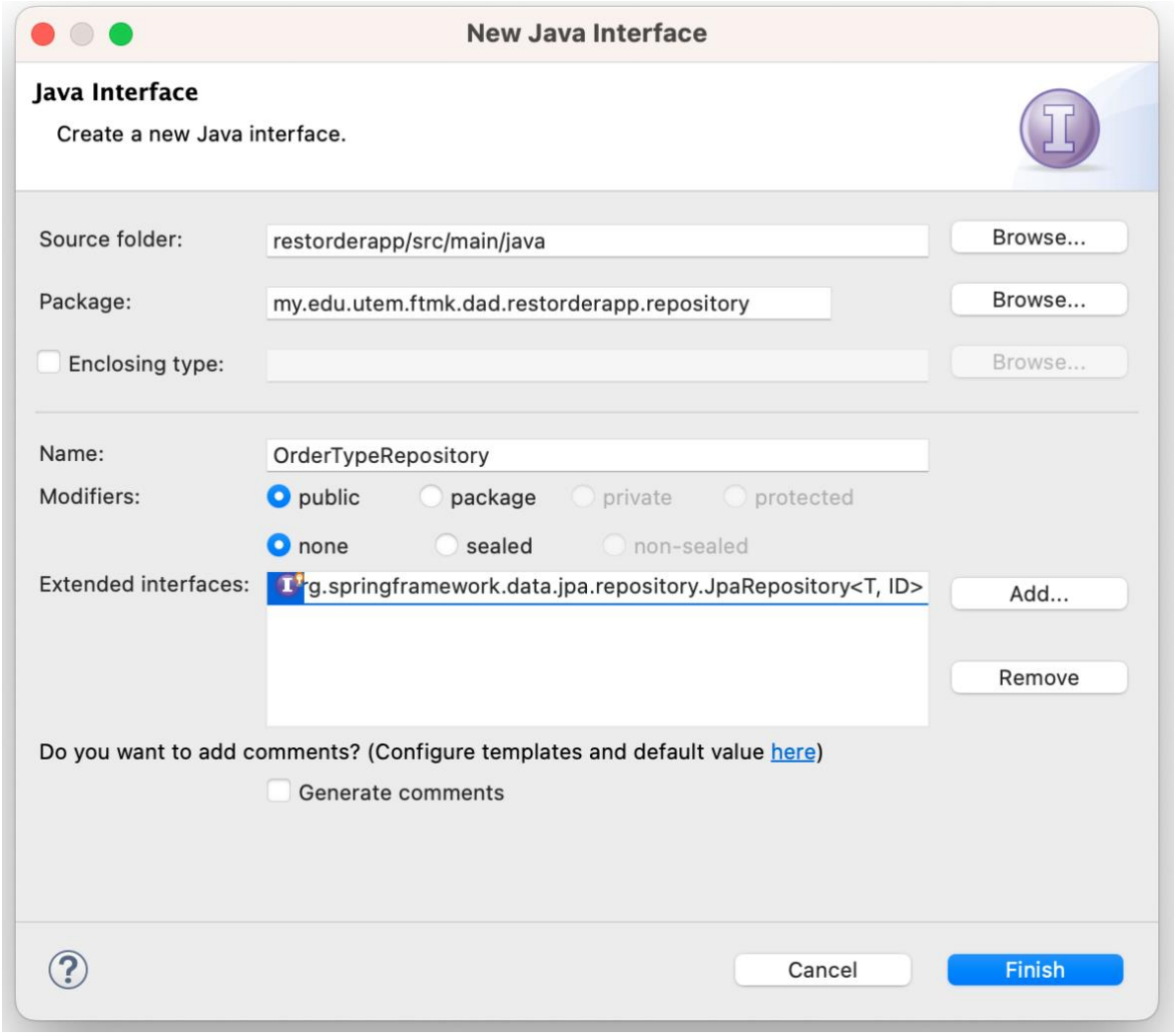

Figure 6.12: Final definition to create a repository class

9. Click the **Finish** button. The interface will be opened in the workspace. The initial look of the interface should be similar to that in Figure 6.13.

```
package my.edu.utem.ftmk.orderdbapp.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderTypeRepository extends JpaRepository<T, ID> {

}
```

Figure 6.13: Initial code of interface OrderTypeRepository

10. Annotate the interface with `@Repository` as shown in Figure 6.14.

```
@Repository
public interface OrderTypeRepository extends JpaRepository<T, ID> {

}
```

Figure 6.14: Annotaton of interface OrderTypeRepository

11. Import `@Repository` from `org.springframework.stereotype`.
12. Change the `<T,ID>` of `JpaRepository` to `<OrderType, Long>`.
13. Import the necessary class. The outcome of the interface definition outcome shall be similar as shown in Figure 6.15

```
package my.edu.utem.ftmk.orderdbapp.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import my.edu.utem.ftmk.orderdbapp.model.ProductType;

@Repository
public interface ProductTypeRepository
    extends JpaRepository<ProductType, Long> {

}
```

Figure 6.15: The complete definition of the interface ProductTypeRepository

14. Save the interface.

7 Controller Layer Implementation

There are three activities in the controller layer implementation. The activities are: -

1. Define a controller class.
2. Define the REST web service methods.
3. Test the REST web service methods.

The relationship between these activities is shown in Figure 7.1.

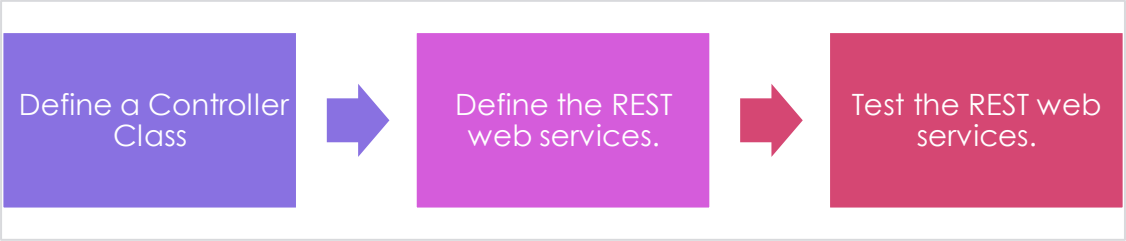

Figure 7.1: Relation of activities in the controller layer implementation

7.1 Define Controller Class

1. Figure 7.2 shows a class that represents web service methods. Create the class and define the attribute shown in Figure 7.2.

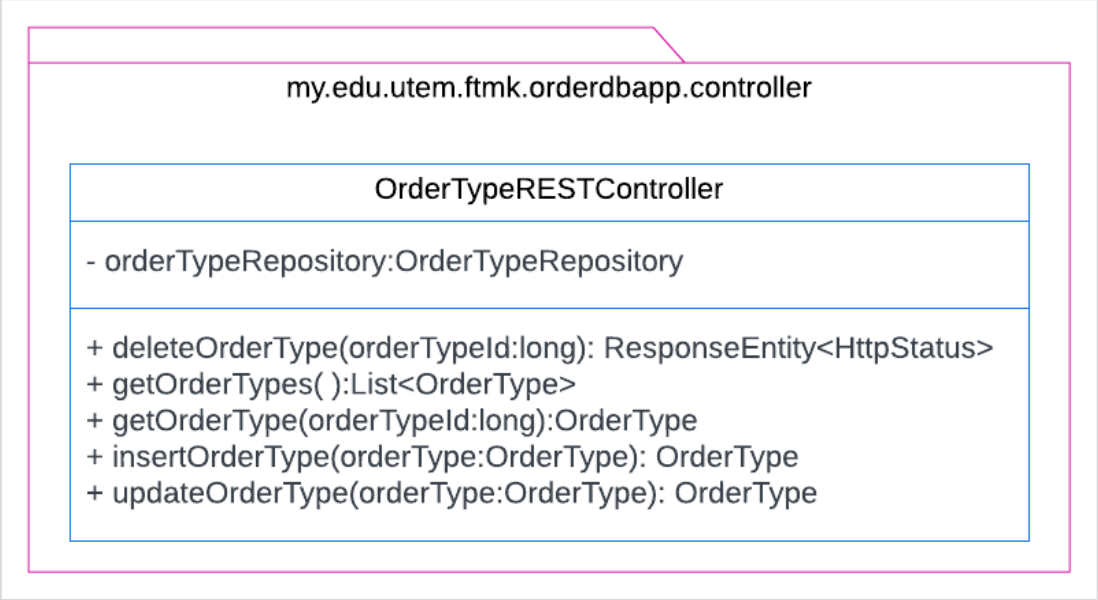

Figure 7.2: Class OrderTypeRESTController in UML notation

2. Annotate the class with `@RestController` and `@RequestMapping("/api/ordertypes")`.
3. Annotate the private attribute with `@Autowired`.
4. Import the annotations from `org.springframework.web.bind.annotation`.
5. Add a block of comments that describes the class before the annotation.

7.2 Define getOrderTypes( ): A GET Web Service Method

1. Define method `getOrderTypes()` as shown in Figure 7.2. This method retrieves all records from table OrderType.
2. Add a block of comments that describe the method.
3. Annotate the method with `@GetMapping`.
4. Provide the implementation shown in Figure 7.3 for the method.

```
@GetMapping
public List<OrderType> getOrderTypes() {

    return orderTypeRepository.findAll();
}
```

Figure 7.3: An implementation of the method getOrderTypes( )

5. Import the annotation from `org.springframework.web.bind.annotation`.
6. Save the class.

7.2.1 Test Web Request to getOrderTypes( )

1. Open **Postman**.
2. Open collection **REST OrderType Requests**.
3. Click the '**+**' shown in Figure 7.4 to add a new request to the collection.

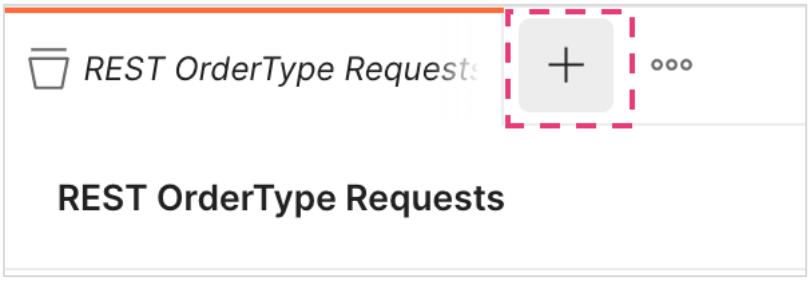

Figure 7.4: Button '+' to create a new request

4. A new tab, as shown in Figure 7.5 will be displayed.

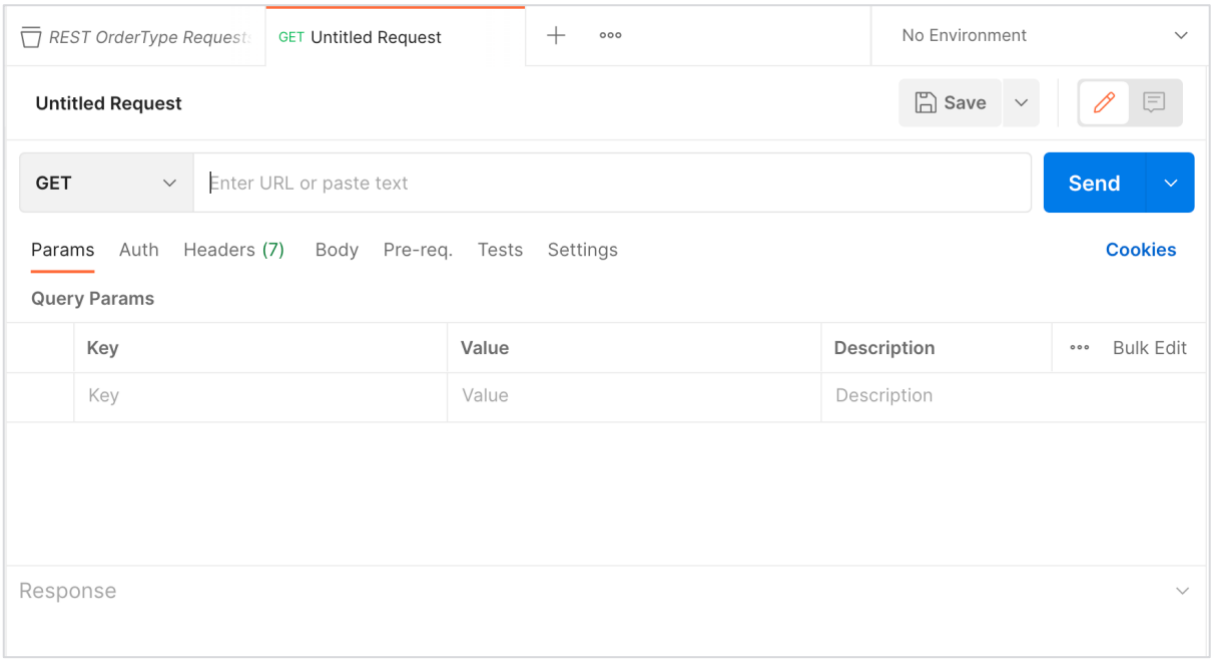

Figure 7.5: A new request tab

5. Paste `http://localhost:8080/orderapp/api/ordertypes` in the request address bar, as shown in Figure 7.6.

Figure 7.6: Request address bar

6. Click the **Send** button. The tab will display a list of records in JSON format in the **Response** area. The output should be similar to Figure 7.7.

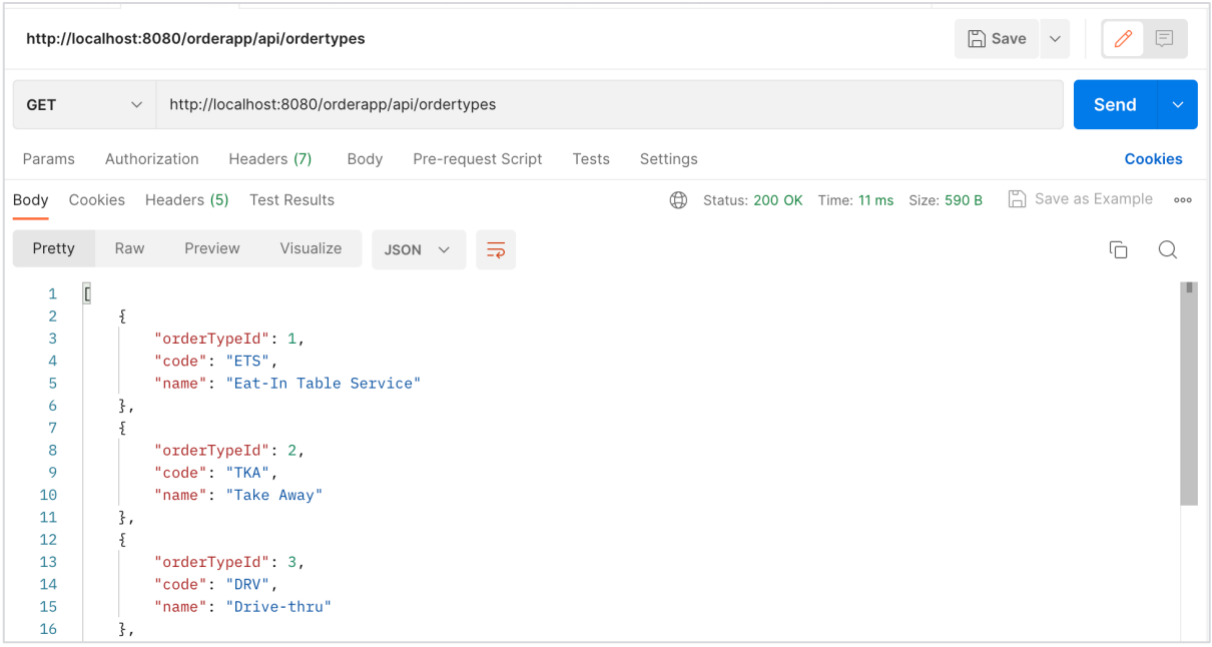

Figure 7.7: Sample of output from GET web service

7. Click **Save** button.
8. Give an appropriate name for the request.

7.3 Define getOrderType(): Another GET web service method

1. Define method `getOrderType()` as shown in Figure 7.2. This method retrieves a record from table OrderType according to a web parameter's value.
2. Add a block of comments that describe the method.
3. Annotate the method with `@GetMapping ("{orderTypeId}")`.
4. Annotate the parameter `orderTypeId` with `@PathVariable.`
5. Provide the implementation shown in Figure 7.8 for the method.

```
@GetMapping("{orderTypeId}")
public OrderType getOrderType(@PathVariable long orderTypeId) {

    OrderType orderType = orderTypeRepository.findById(orderTypeId).get();

    return orderType;
}
```

Figure 7.8: An implementation of method getOrderType

6. Import the annotations from `org.springframework.web.bind.annotation`.
7. Save the class.

7.3.1 Test Web Request to getOrderType( )

1. Open **Postman**.
2. Create a new request in the workspace. Refer to step 2 in Exercise 7.2.1.
3. Paste `http://localhost:8080/orderapp/api/ordertypes/3` in the address bar.
4. Click the **Send** button. A record of OrderType will be displayed in the **Response** area. The output should be similar to Figure 7.9.

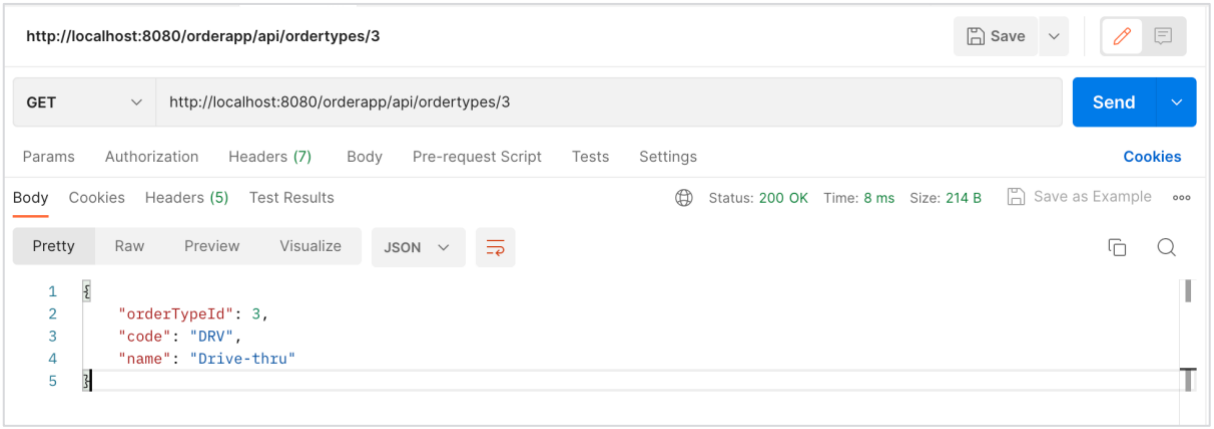

Figure 7.9: A sample of output from GET web service with parameter

5. Click the **Save** button.
6. Give an appropriate name for the request.

7.4 Define insertOrderType: A POST Web Service Method

1. Define the method `insertOrderType` as shown in Figure 7.2. This method creates a new record in table OrderType.
2. Add a block of comments that describe the method.
3. Annotate the method with `@PostMapping( )`.
4. Annotate the parameter `orderType` with `@RequestBody`.
5. Provide the implementation shown in Figure 7.10 for the method.

```
@PostMapping()
public OrderType insertOrderType(@RequestBody OrderType orderType) {

    return orderTypeRepository.save(orderType);
}
```

Figure 7.10: An implementation of method insertOrderType

6. Import the annotations from `org.springframework.web.bind.annotation`.
7. Save the class.

7.4.1 Test insertOrderType

1. Open **Postman**.
2. Create a new request in the workspace. Refer to step 2 in Exercise 7.2.1.
3. Paste `http://localhost:8080/orderapp/api/ordertypes` in the address bar.
4. Change the web method from GET to **POST**.
5. Click **Body**, **raw** and then **JSON**, as shown in Figure 7.11 to specify the parameters to be provided by the service requestor.

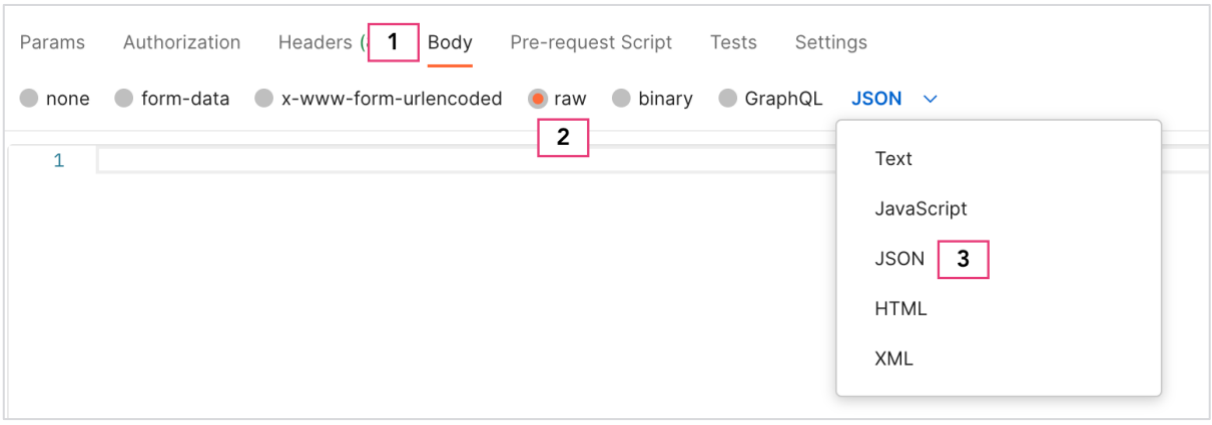

Figure 7.11: Parameters specification from the service requestor

6. Then, add the JSON data shown in Figure 7.12 in the text area.

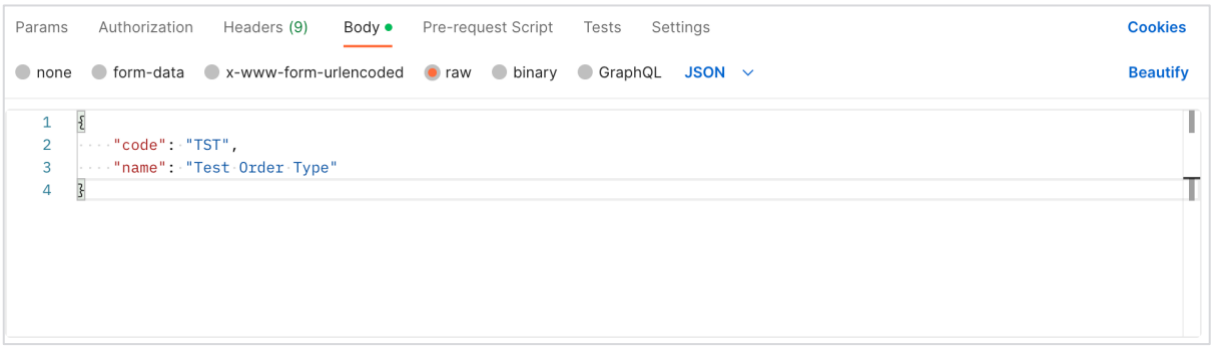

Figure 7.12: Adding JSON data from a service requestor

7. After that, click the **Send** button. The tab will display a new record in JSON format in the **Response** area. The output should be similar to Figure 7.13.

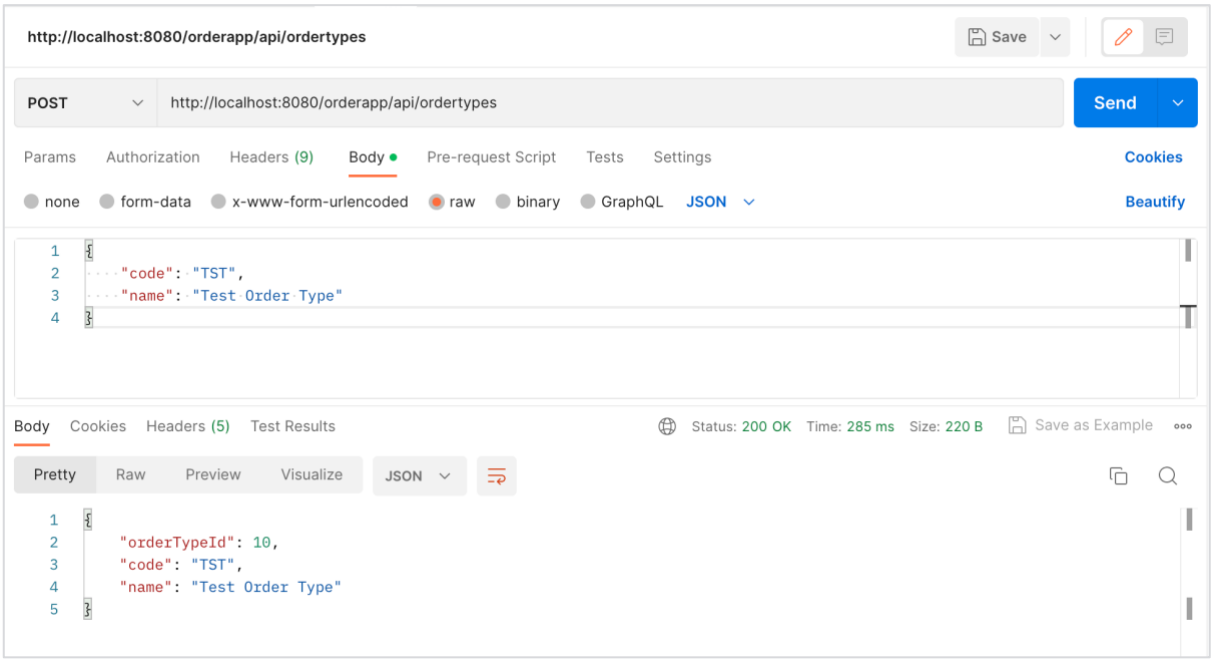

Figure 7.13: A sample of output from POST web service using JSON data as parameters

8. Finally, click the **Save** button.
9. Give an appropriate name for the request.

7.5 Define updateOrderType: A PUT Web Service Method

1. Define method `updateOrderType` as shown in Figure 7.2. This method updates a record in table OrderType.
2. Add a block of comments that describe the method.
3. Annotate the method with `@PuttMapping ( )`.
4. Annotate the parameter `orderType` with `@RequestBody`.
5. Provide the implementation shown in Figure 7.14 for the method.

```
@PutMapping()
public OrderType updateOrderType(@RequestBody OrderType orderType) {

    return orderTypeRepository.save(orderType);
}
```

Figure 7.14: An implementation of method updateOrderType

6. Import the annotations from `org.springframework.web.bind.annotation`.
7. Save the class.

7.5.1 Test updateOrderType

1. Open **Postman**.
2. Create a new request in the workspace. Refer to step 2 in Exercise 7.2.1.
3. Paste `http://localhost:8080/orderapp/api/ordertypes` in the address bar.
4. Change the web method from GET to **PUT**.
5. Click **Body**, **raw** and then **JSON** as shown in Figure 7.15.

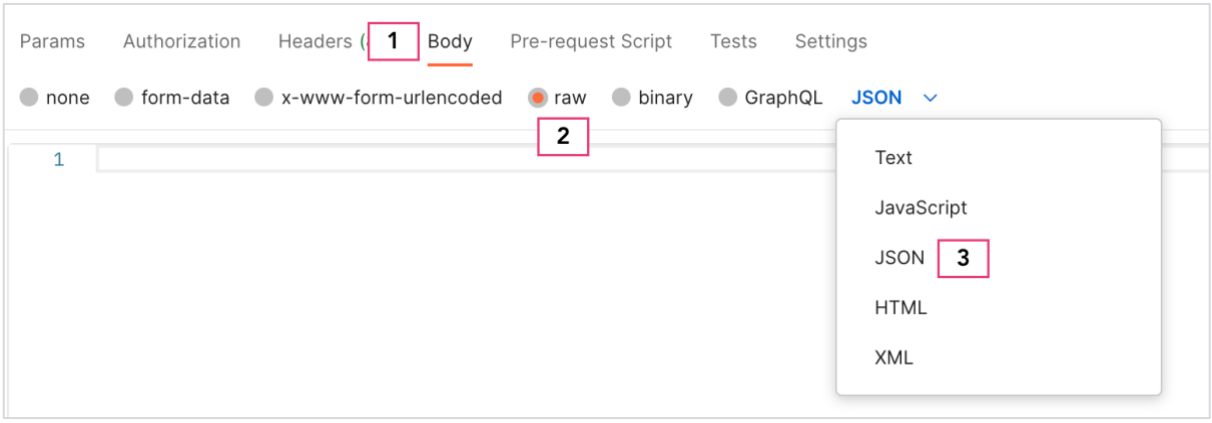

Figure 7.15: Parameters specification from the service requestor

6. Then add the JSON data as shown in Figure 7.16.

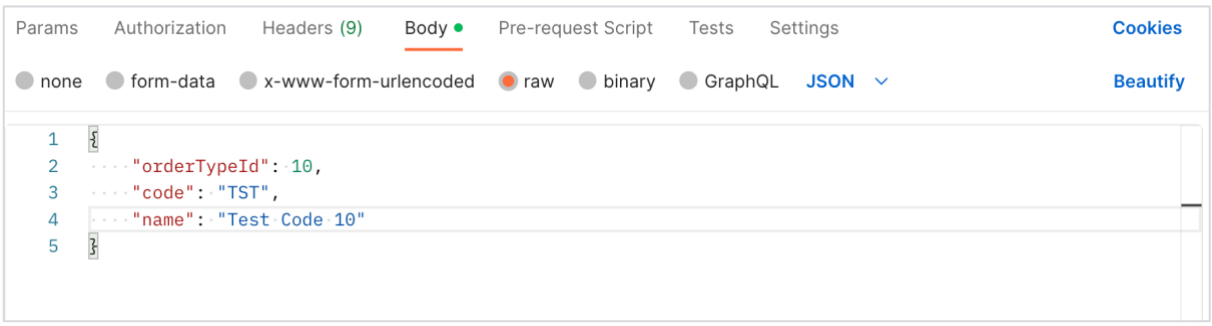

Figure 7.16: Adding JSON data from a service requestor

7. After that, click the **Send** button. The output should be similar to Figure 7.17.

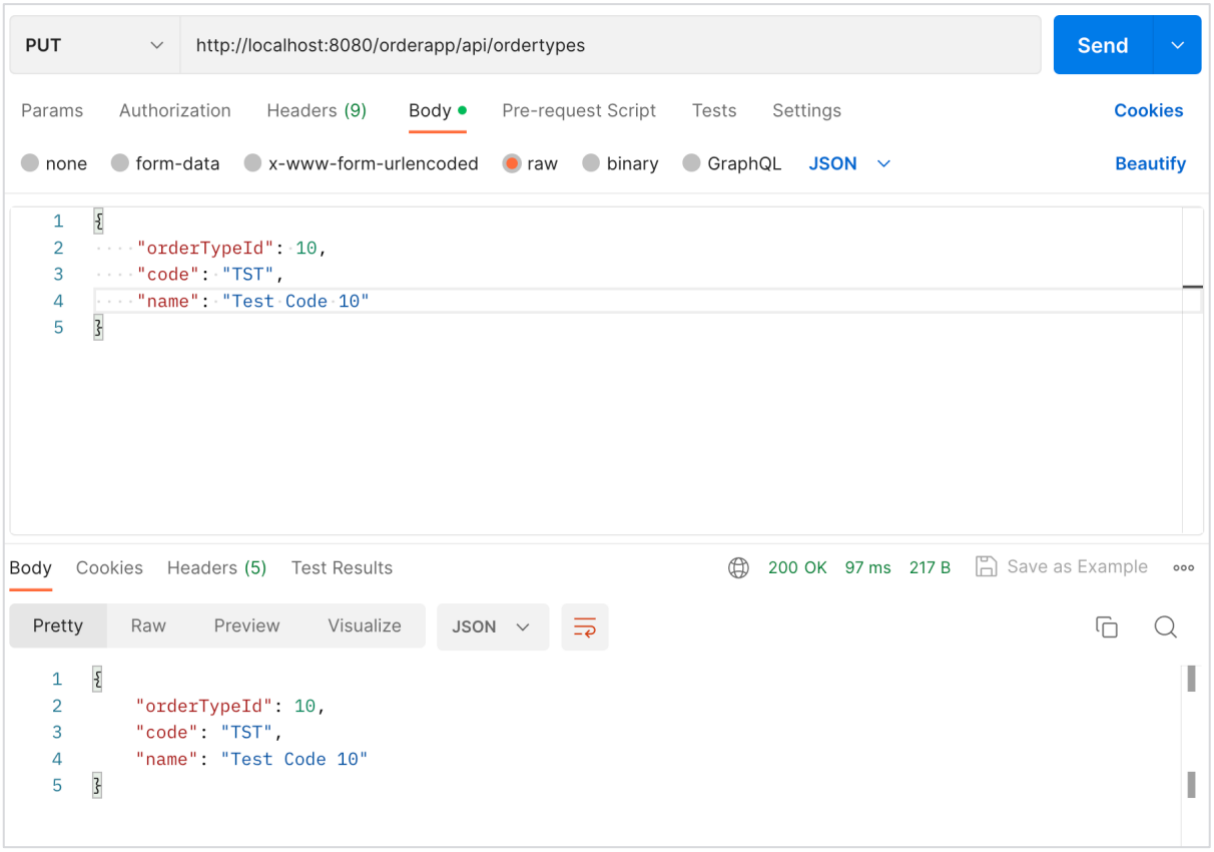

Figure 7.17: A sample of output from PUT web service using JSON data as parameters

8. Click the **Save** button.
9. Give an appropriate name to the request.

7.6 Define deleteOrderType: A DELETE Web Service Method

1. Define method `deleteOrderType()` as shown in Figure 7.2. This method deletes a record from table OrderType.
2. Import `ResponseEntity` and `HttpStatus` from `org.springframework.http`.
3. Add a block of comments that describe the method.
4. Annotate the method with `@DeleteMapping ("{orderTypeId}")`.
5. Annotate the parameter `orderTypeId` with `@PathVariable`.
6. Provide the implementation shown in Figure 7.18 for the method.

```
@DeleteMapping("{orderTypeId}")
public ResponseEntity<HttpStatus> deleteOrderType(@PathVariable long orderTypeId) {

    orderTypeRepository.deleteById(orderTypeId);

    return new ResponseEntity<>(HttpStatus.OK);
}
```

Figure 7.18: Implementation of method deleteOrderType

7. Save the class.

7.6.1 Test deleteOrderType

1. Open **Postman**.
2. Create a new request in the workspace. Refer to step 2 in Exercise 7.2.1.
3. Paste `http://localhost:8080/orderapp/api/ordertypes/10` in the request address bar.
4. Change the web method from GET to **DELETE**.
5. After that, click the button **Send**. The output should be similar to .

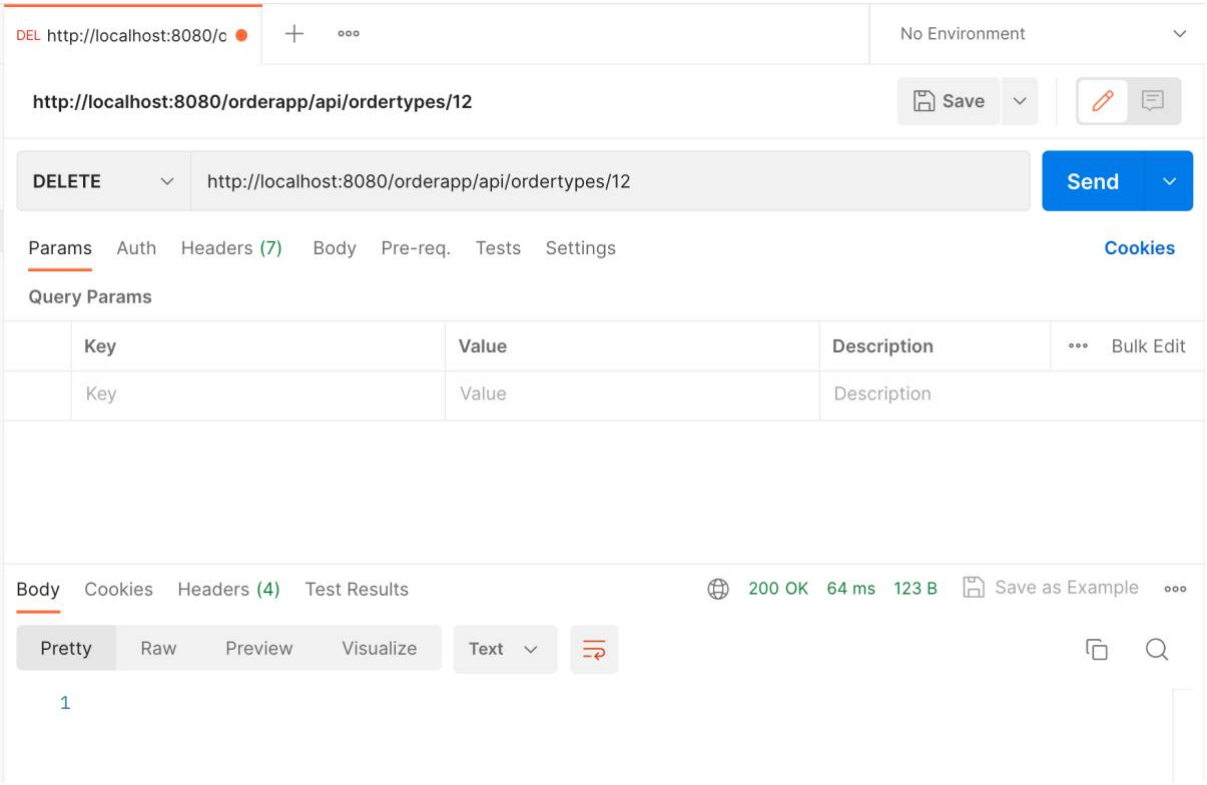

Figure 7.19: A sample of output from DELETE web service

6. Finally, click the **Save** button.
7. Give an appropriate name for the request.

End of Document