LAPORAN HASIL PRAKTIKUM 7

PEMROGRAMAN BERORIENTASI OBJEK LANJUT

“Spring Boot-3”

Dosen Pengampu :

Sri Hartati Wijono, M.Kom.

Oleh

Nama : Maria Gresia Plena Br Purba

NIM : 235314094

Kelas : DP

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS SANATA DHARMA

YOGYAKARTA

2024

A. TUJUAN

1. Memahami cara membuat aplikasi Spring Boot REST API.
2. Memahami konsep CRUD (Create, Read, Update, Delete) yang lebih kompleks.
3. Memahami konsep MVC.

B. PRAKTIKUM

1. Package model

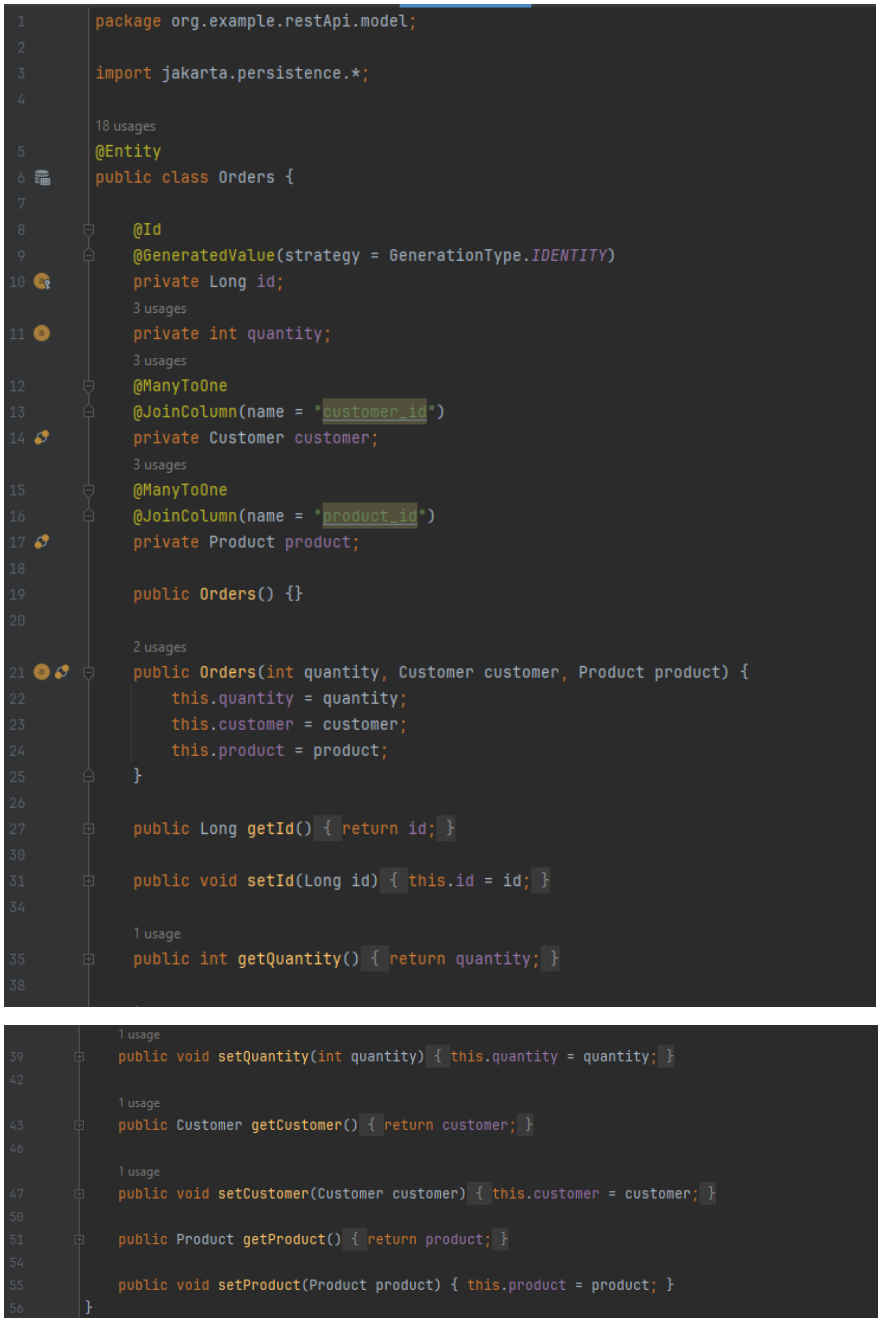
a. Customer

```java
package org.example.restApi.model;

import jakarta.persistence.*;
import java.util.List;

18 usages
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    3 usages
    private String name;
    3 usages
    private String email;
    2 usages
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<Orders> orders;

    public Customer() {}

    2 usages
    public Customer(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    1 usage
    public String getEmail() { return email; }
```

```java
    public void setEmail(String email) { this.email = email; }

    no usages
    public List<Orders> getOrders() { return orders; }

    no usages
    public void setOrders(List<Orders> orders) { this.orders = orders; }
}
```

Penjelasan : class ini memiliki primary key berupa “id” yang menggunakan GenerationType.IDENTITY. kemudian, terdapat variabel name, email, dan orders yang bersifat OneToMany dengan class Orders. Relasi OneToMany pada class ini berarti setiap order terhubung ke satu customer. Lalu, terdapat penggunaan getter setter untuk mendapatkan dan mengeset nilai dari id, name, email, dan orders.

b. Orders



```
package org.example.restApi.model;

import jakarta.persistence.*;

18 usages
@Entity
public class Orders {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    3 usages
    private int quantity;
    3 usages
    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;
    3 usages
    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;

    public Orders() {}

    2 usages
    public Orders(int quantity, Customer customer, Product product) {
        this.quantity = quantity;
        this.customer = customer;
        this.product = product;
    }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    1 usage
    public int getQuantity() { return quantity; }

    1 usage
    public void setQuantity(int quantity) { this.quantity = quantity; }

    1 usage
    public Customer getCustomer() { return customer; }

    1 usage
    public void setCustomer(Customer customer) { this.customer = customer; }

    public Product getProduct() { return product; }

    public void setProduct(Product product) { this.product = product; }
}
```

Penjelasan : class ini memiliki primary key berupa “id” yang menggunakan GenerationType.IDENTITY. Kemudian, terdapat variabel quantity, customer, dan product. Variabel customer dan product bersifat ManyToOne di mana customer berelasi dengan class Customer, sedangkan product berelasi dengan class Product. @JoinColumn pada customer_id dan product_id berfungsi untuk menghubungkan order dengan pelanggan dan produk. Lalu, terdapat penggunaan getter setter untuk mendapatkan dan mengeset nilai dari id, quantity, customer, dan product.

c. Product

```java
package org.example.restApi.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
19 usages
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    7 usages
    private String name;
    7 usages
    private double price;

    5 usages
    private String description;

    public Product() {
    this.name = name;
    this.description = description;
    this.price = price;
    }

    no usages
    public Product(Long id, String name, double price) {
        this.name = name;
        this.id = id;
        this.price = price;
    }
    2 usages
    public Product(String name, double price){
        this.name = name;
        this.price = price;
    }
```

```java
    no usages
    public Product(Long id, String name, String description, double price) {
        this.name = name;
        this.id = id;
        this.description = description;
        this.price = price;
    }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    1 usage
    public double getPrice() { return price; }

    1 usage
    public void setPrice(double price) { this.price = price; }

    1 usage
    public String getDescription() { return description; }

    1 usage
    public void setDescription(String description) { this.description = description; }
}
```

Penjelasan : class ini memiliki primary key berupa “id” yang menggunakan GenerationType.IDENTITY. Kemudian, terdapat variabel name, price, dan description. Lalu, terdapat penggunaan getter setter untuk mendapatkan dan mengeset nilai dari id, name, price, dan description.

2. Package repository

a. CustomerRepository

```java
package org.example.restApi.repository;

import org.example.restApi.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

6 usages
public interface CustomerRepository extends JpaRepository<Customer, Long>{

}
```

Penjelasan : interface mewarisi JpaRepository yang akan secara otomatis menyediakan method untuk penyimpanan, pembaruan, penghapusan, dan pengambilan data.

b. OrderRepository

```java
package org.example.restApi.repository;

import org.example.restApi.model.Orders;
import org.springframework.data.jpa.repository.JpaRepository;

4 usages
public interface OrderRepository extends JpaRepository<Orders, Long>{

}
```

Penjelasan : interface mewarisi JpaRepository yang akan secara otomatis menyediakan method untuk penyimpanan, pembaruan, penghapusan, dan pengambilan data.

c. ProductRepository

```java
package org.example.restApi.repository;

import org.example.restApi.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;

6 usages
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

Penjelasan : interface mewarisi JpaRepository yang akan secara otomatis menyediakan method untuk penyimpanan, pembaruan, penghapusan, dan pengambilan data.

3. Package controller

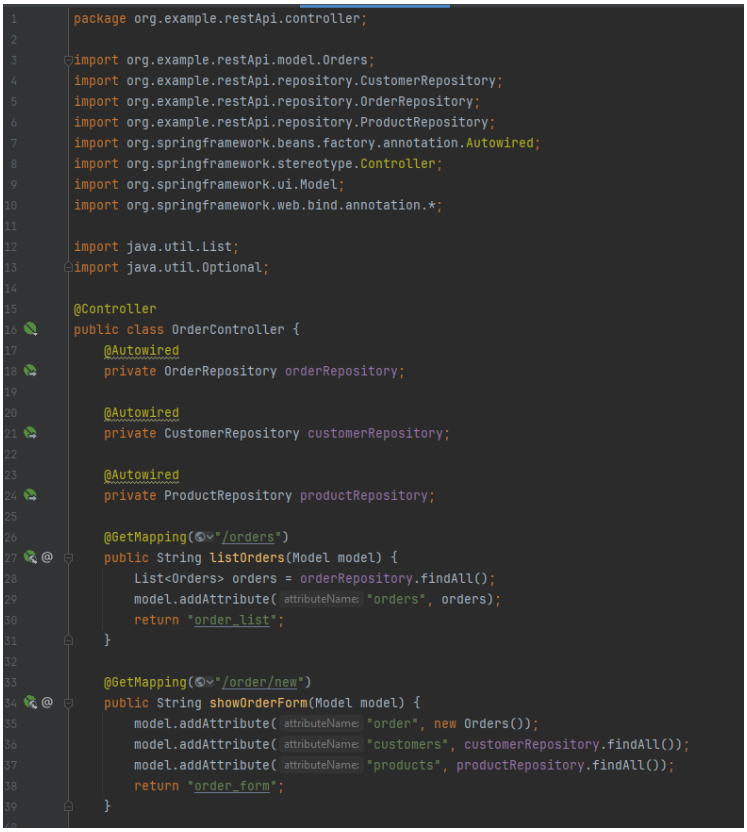
a. CustomerController

```java
package org.example.restApi.controller;

import org.example.restApi.model.Customer;
import org.example.restApi.repository.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@Controller
public class CustomerController {
    @Autowired
    private CustomerRepository customerRepository;

    @GetMapping("/customers")
    public String listCustomers(Model model) {
        List<Customer> customers = customerRepository.findAll();
        model.addAttribute( attributeName: "customers", customers);
        return "customer_list";
    }
    @GetMapping("/customer/new")
    public String showCustomerForm(Model model) {
        model.addAttribute( attributeName: "customer", new Customer());
        return "customer_form";
    }
    @PostMapping("/customer/save")
    public String saveCustomer(@ModelAttribute Customer customer) {
        customerRepository.save(customer);
        return "redirect:/customers";
    }
```

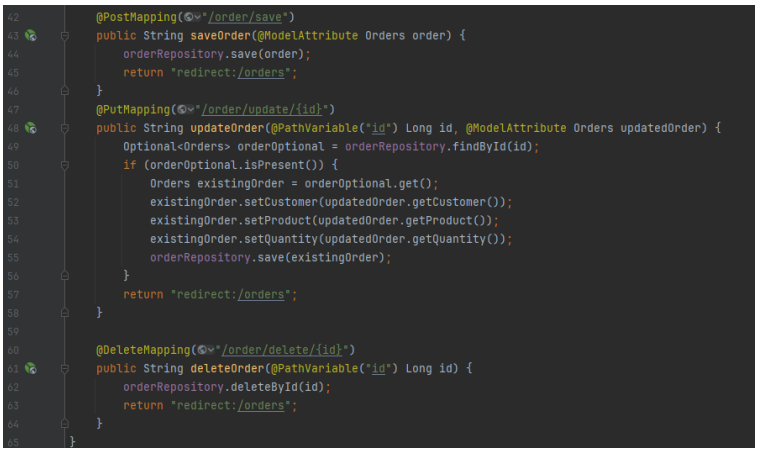
```java
    @PutMapping("/customer/update/{id}")
    public String updateCustomer(@PathVariable("id") Long id, @ModelAttribute Customer updatedCustomer) {
        Optional<Customer> customerOptional = customerRepository.findById(id);
        if (customerOptional.isPresent()) {
            Customer existingCustomer = customerOptional.get();
            existingCustomer.setName(updatedCustomer.getName());
            existingCustomer.setEmail(updatedCustomer.getEmail());
            // Remove setting ID; it's auto-generated
            customerRepository.save(existingCustomer);
        }
        return "redirect:/customers";
    }

    @DeleteMapping("/customer/delete/{id}")
    public String deleteCustomer(@PathVariable("id") Long id) {
        customerRepository.deleteById(id);
        return "redirect:/customers";
    }
}
```

Penjelasan : menggunakan @Autowired pada CustomerRepository. Method listCustomers berfungsi untuk menampilkan semua data yang ada, pada halaman “customer_list”. Method showCustomerForm berfungsi untuk menampilkan form untuk menambahkan data baru berupa objek customer. Method saveCustomer berfungsi untuk menyimpan data baru dan pengguna akan diarahkan ke halaman “/customers”. Method updateCustomer berfungsi untuk meng-update informasi customer yang sudah ada berdasarkan customer ID nya dan akan kembali ke “/customers” setelah meng-update informasi . Method deleteCustomer berfungsi untuk menghapus customer berdasarkan id-nya dan akan kembali ke “/customers” setelah penghapusan.

b. OrderController



```java
package org.example.restApi.controller;

import org.example.restApi.model.Orders;
import org.example.restApi.repository.CustomerRepository;
import org.example.restApi.repository.OrderRepository;
import org.example.restApi.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@Controller
public class OrderController {
    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private ProductRepository productRepository;

    @GetMapping("/orders")
    public String listOrders(Model model) {
        List<Orders> orders = orderRepository.findAll();
        model.addAttribute( attributeName: "orders", orders);
        return "order_list";
    }

    @GetMapping("/order/new")
    public String showOrderForm(Model model) {
        model.addAttribute( attributeName: "order", new Orders());
        model.addAttribute( attributeName: "customers", customerRepository.findAll());
        model.addAttribute( attributeName: "products", productRepository.findAll());
        return "order_form";
    }
```



```java
    @PostMapping("/order/save")
    public String saveOrder(@ModelAttribute Orders order) {
        orderRepository.save(order);
        return "redirect:/orders";
    }
    @PutMapping("/order/update/{id}")
    public String updateOrder(@PathVariable("id") Long id, @ModelAttribute Orders updatedOrder) {
        Optional<Orders> orderOptional = orderRepository.findById(id);
        if (orderOptional.isPresent()) {
            Orders existingOrder = orderOptional.get();
            existingOrder.setCustomer(updatedOrder.getCustomer());
            existingOrder.setProduct(updatedOrder.getProduct());
            existingOrder.setQuantity(updatedOrder.getQuantity());
            orderRepository.save(existingOrder);
        }
        return "redirect:/orders";
    }

    @DeleteMapping("/order/delete/{id}")
    public String deleteOrder(@PathVariable("id") Long id) {
        orderRepository.deleteById(id);
        return "redirect:/orders";
    }
}
```

Penjelasan : menggunakan @Autowired pada OrderRepository, CustomerRepository, dan ProductRepository. Method listOrders berfungsi untuk menampilkan semua data yang ada, pada halaman “order_list.html”. method showOrderForm berfungsi untuk menampilkan form untukk menambahkan data baru berupa objek order, customers, dan products. Method saveOrder berfungsi untuk menyimpan data baru dan pengguna akan diarahkan ke halaman “/orders”. Method updateOrder berfungsi untuk meng-update informasi order yang sudah ada berdasarkan order ID nya dan akan kembali ke “/orders” setelah meng-update informasi . Method deleteOrder berfungsi untuk menghapus order berdasarkan id nya dan akan kembali ke “/orders” setelah penghapusan.

c. ProductController

```java
package org.example.restApi.controller;

import org.example.restApi.model.Product;
import org.example.restApi.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@Controller
public class ProductController {
    @Autowired
    private ProductRepository productRepository;

    @GetMapping("/products")
    public String listProducts(Model model) {
        List<Product> products = productRepository.findAll();
        model.addAttribute( attributeName: "products", products);
        return "product_list";
    }
    @GetMapping("/product/new")
    public String showProductForm(Model model) {
        model.addAttribute( attributeName: "product", new Product());
        return "product_form";
    }
    @PostMapping("/product/save")
    public String saveProduct(@ModelAttribute Product product) {
        productRepository.save(product);
        return "redirect:/products";
    }
    @PutMapping("/product/update/{id}")
    public String updateProduct(@PathVariable("id") Long id, @ModelAttribute Product updatedProduct) {
        Optional<Product> orderOptional = productRepository.findById(id);
        if (orderOptional.isPresent()) {
            Product existingProduct = orderOptional.get();
            existingProduct.setName(updatedProduct.getName());
            existingProduct.setDescription(updatedProduct.getDescription());
            existingProduct.setPrice(updatedProduct.getPrice());
            productRepository.save(existingProduct);
        }
        return "redirect:/products";
    }

    @DeleteMapping("/product/delete/{id}")
    public String deleteProduct(@PathVariable("id") Long id) {
        productRepository.deleteById(id);
        return "redirect:/products";
    }
}
```

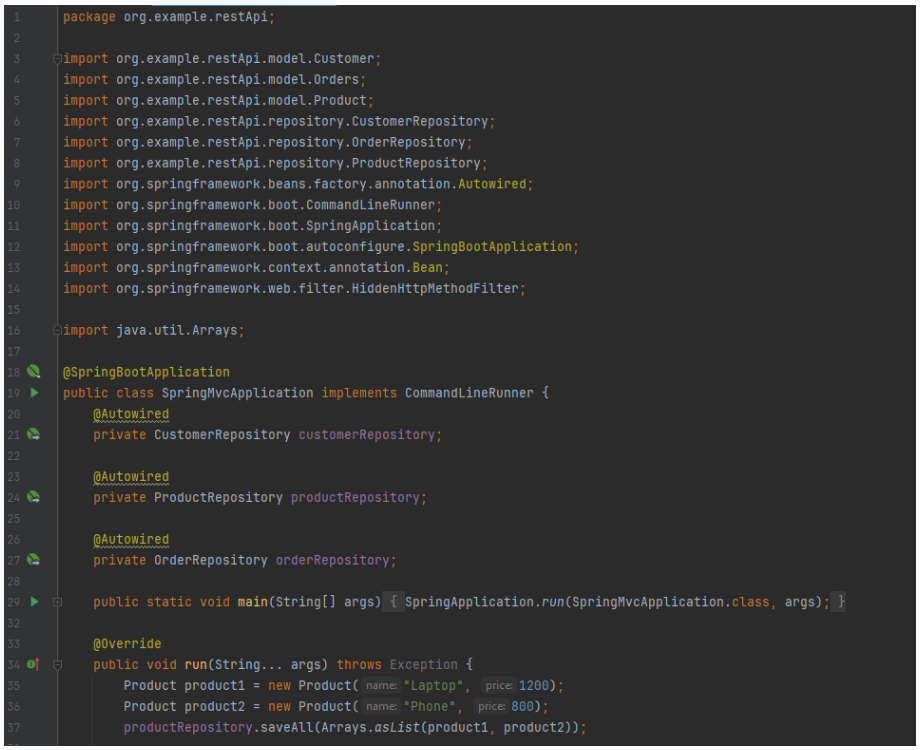
Penjelasan : menggunakan @Autowired pada ProductRepository. Method listProducts berfungsi untuk menampilkan semua data yang ada, pada halaman “product_list”. Method showProductForm berfungsi untuk menampilkan form untuk menambahkan data baru berupa objek product. Method saveProduct berfungsi untuk menyimpan data baru dan pengguna akan diarahkan ke halaman “/products”. Method updateProduct berfungsi untuk meng-update informasi product yang sudah ada berdasarkan product ID nya dan akan kembali ke “/products” setelah meng-update informasi . Method deleteProducts berfungsi untuk menghapus product berdasarkan id-nya dan akan kembali ke “/products” setelah penghapusan.

4. restApiApplication

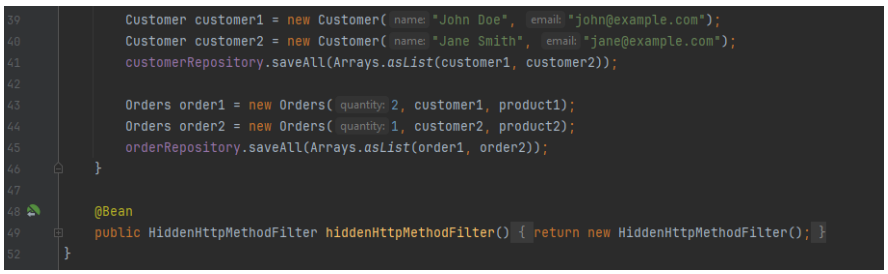
```java
package org.example.restApi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestApiApplication {

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

}
```

Penjelasan : class ini berfungsi sebagai titik awal saat ingin menjalankan aplikasi Spring Boot. Anotasi @SpringBootApplication menunjukkan bahwa ini adalah kelas utama untuk Spring Boot. Method main() memanggil SpringApplicatio.run() untuk menjalankan aplikasi-aplikasi yang ada.

5. SpringMvcApplication



```java
package org.example.restApi;

import org.example.restApi.model.Customer;
import org.example.restApi.model.Orders;
import org.example.restApi.model.Product;
import org.example.restApi.repository.CustomerRepository;
import org.example.restApi.repository.OrderRepository;
import org.example.restApi.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.filter.HiddenHttpMethodFilter;

import java.util.Arrays;

@SpringBootApplication
public class SpringMvcApplication implements CommandLineRunner {
    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private OrderRepository orderRepository;

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

    @Override
    public void run(String... args) throws Exception {
        Product product1 = new Product( name: "Laptop",  price: 1200);
        Product product2 = new Product( name: "Phone",  price: 800);
        productRepository.saveAll(Arrays.asList(product1, product2));
```



```java
        Customer customer1 = new Customer( name: "John Doe",  email: "john@example.com");
        Customer customer2 = new Customer( name: "Jane Smith",  email: "jane@example.com");
        customerRepository.saveAll(Arrays.asList(customer1, customer2));

        Orders order1 = new Orders( quantity: 2, customer1, product1);
        Orders order2 = new Orders( quantity: 1, customer2, product2);
        orderRepository.saveAll(Arrays.asList(order1, order2));
    }

    @Bean
    public HiddenHttpMethodFilter hiddenHttpMethodFilter() { return new HiddenHttpMethodFilter(); }
}
```

Penjelasan : class ini menggunakan implemnetasi CommandLineRunner untuk mengeksekusi kode. Pada method run(), terdapat pembuatan objek dari Product, Customer, dan Orders. Kemudian, mengisi data pada parameternya. Lalu, menggunakan saveAll() untuk menyimpan objek Product ke ProductRepository, Customer ke CustomerRepository, dan objek Order ke OrderRepository. Method HiddenHttpMethodFilter() merupakan bean yang memungkinkan dukungan untuk HTTP method.

6. Html

a. customer_form.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Customer Form</title>
</head>
<body>
<h1>Customer Form</h1>

<!-- Form untuk menambahkan/mengedit customer -->
<form th:action="@{/customer/save}" th:object="${customer}" method="post">

    <!-- Input untuk nama customer -->
    <label>Customer Name:</label>
    <input type="text" th:field="*{name}" placeholder="Customer Name" required>
    <br>

    <!-- Input untuk email customer -->
    <label>Customer Email:</label>
    <input type="email" th:field="*{email}" placeholder="Customer Email" required>
    <br>

    <!-- Submit button -->
    <button type="submit">Save Customer</button>
</form>
</body>
</html>
```

Penjelasan : form ini berfungsi agar customer dapat menambahkan atau mengedit data customer dengan meng-input data nama dan email. Lalu, membuat tombol dengan nama submit agar pengguna dapat menyimpan data-data tersebut.

b. customer_list.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Customer List</title>
</head>
<body>
<h1>Customer List</h1>
<a href="/customer/new">Add New Customer</a>
<table border="1">
    <tr>
        <th>Customer ID</th>
        <th>Customer Name</th>
        <th>Customer Email</th>
        <th>Actions</th>
    </tr>
    <tr th:each="customer : ${customers}">
        <td th:text="${customer.id}"></td>
        <td th:text="${customer.name}"></td>
        <td th:text="${customer.email}"></td>
        <td>
            <form th:action="@{/customer/delete/{id}(id=${customer.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="delete" />
                <button type="submit">Delete</button>
            </form>
            <!-- Form untuk Update -->
            <form th:action="@{/customer/update/{id}(id=${customer.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="put" />

                <!-- Input fields untuk data baru -->
                <input type="text" name="name" placeholder="New Name" value="" required />
                <input type="email" name="email" placeholder="New Email" value="" required />

                <button type="submit">Update</button>
            </form>
        </td>
    </tr>
</table>
</body>
</html>
```

Penjelasan : halaman ini bernama “Customer List” dan memiliki tautan “Add New Customer” yang mengarah ke halaman untuk menambah customer baru. Membuat kolom pada tabel dengan nama “Customer ID”, “Customer Name”, “Customer Email”, dan “Actions”. Pada kolom Actions, terdapat tombol Delete dengan method DELETE dan terdapat form untuk meng-update nama dan email customer dengan method PUT.

c. order_from.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <title>Order Form</title>
</head>
<body>
<h1>Order Form</h1>

<!-- Form untuk menambahkan/mengedit produk -->
<form th:action="@{/order/save}" th:object="${order}" method="post">

    <!-- Input untuk ID produk (hidden) -->
    <input type="hidden" th:field="*{id}">

    <!--     Input untuk ID customer -->
    <label>Customer ID :</label>
    <input type="text" th:field="*{customer}" placeholder="Customer ID" required>
    <br>

    <!--     Input untuk ID produk -->
    <label>ID Product:</label>
    <input type="text" th:field="*{product}" placeholder="Product ID" required>
    <br>

    <!-- Input untuk jumlah produk -->
    <label>Quantity:</label>
    <input type="text" th:field="*{quantity}" placeholder="Quantity" required>
    <br>

    <!-- Submit button -->
    <button type="submit">Save Order</button>
</form>
</body>
</html>
```

Penjelasan : form ini berfungsi untuk menambahkan atau mengedit data order dengan meng-input data dari id customer, product, dan quantity-nya. Lalu, membuat tombol dengan nama submit agar pengguna dapat menyimpan data-data tersebut.

d. order_list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Order List</title>
</head>
<body>
 <h1>Order List</h1>
 <a href="/order/new">Add New Order</a>
<table border="1">
    <tr>
        <th>Customer Name</th>
        <th>Product Name</th>
        <th>Product Quantity</th>
        <th>Actions</th>
    </tr>
    <tr th:each="order : ${orders}">
        <td th:text="${order.customer.name}"></td>
        <td th:text="${order.product.name}"></td>
        <td th:text="${order.quantity}"></td>
        <td>
            <form th:action="@{/order/delete/{id}(id=${order.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="delete" />
                <button type="submit">Delete</button>
            </form>
            <form th:action="@{/order/update/{id}(id=${order.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="put" />

                <!-- Input fields untuk data baru -->
                <input type="number" name="quantity" placeholder="New Product Quantity" value="" required />

                <button type="submit">Update</button>
            </form>
        </td>
    </tr>
</table>
</body>
</html>

```

Penjelasan : halaman ini bernama “Order List” dan memiliki tautan Add New Order yang mengarah ke halaman untuk menambah order baru. Membuat kolom pada tabel dengan nama “Customer ID”, “Product Name”, “Product Quantity”, dan “Actions”. Pada kolom Actions, terdapat tombol Delete dengan method DELETE dan terdapat form untuk meng-update jumlah product dengan method PUT.

e. product_form.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Product Form</title>
</head>
<body>
<h1>Product Form</h1>

<!-- Form untuk menambahkan/mengedit produk -->
<form th:action="@{/product/save}" th:object="${product}" method="post">

    <!-- Input untuk ID produk (hidden) -->
    <input type="hidden" th:field="*{id}">

    <!-- Input untuk ID produk -->
    <label>Product ID:</label>
    <input type="text" th:field="*{id}" placeholder="Product ID" required>
    <br>

    <!-- Input untuk nama produk -->
    <label>Product Name:</label>
    <input type="text" th:field="*{name}" placeholder="Product Name" required>
    <br>

    <!-- Input untuk harga produk -->
    <label>Product Price:</label>
    <input type="number" th:field="*{price}" placeholder="Product Price" step="0.01" required>
    <br>

    <!-- Submit button -->
    <button type="submit">Save Product</button>
</form>
</body>
</html>
```

Penjelasan : form ini berfungsi untuk menambahkan atau mengedit data product dengan meng-input data id produk, nama produk, dan harga produk. Lalu, membuat tombol dengan nama submit untuk menyimpan data-data tersebut.

f. product _list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Product List</title>
</head>
<body>
<h1>Product List</h1>
<a href="/product/new">Add New Product</a>
<table border="1">
    <tr>
        <th>Product ID</th>
        <th>Product Name</th>
        <th>Product Price</th>
        <th>Actions</th>
    </tr>
    <tr th:each="product : ${products}">
        <td th:text="${product.id}"></td>
        <td th:text="${product.name}"></td>
        <td th:text="${product.price}"></td>
        <td>
            <form th:action="@{/product/delete/{id}(id=${product.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="delete" />
                <button type="submit">Delete</button>
            </form>
            <form th:action="@{/product/update/{id}(id=${product.id})}" method="post" style="display:inline;">
                <input type="hidden" name="_method" value="put" />

                <!-- Input fields untuk data baru -->
                <input type="text" name="name" placeholder="New Name" value="" required />
                <input type="text" name="price" placeholder="New Price" value="" required />

                <button type="submit">Update</button>
            </form>
        </td>
    </tr>
</table>
</body>
</html>

```

Penjelasan : halaman ini bernama “Product List” dan memiliki tautan “Add New Product” yang mengarah ke halaman untuk menambah product baru. Membuat kolom pada tabel dengan nama “Product ID”, “Product Name”, “Product Price”, dan “Actions”. Pada kolom Actions, terdapat tombol Delete dengan method DELETE dan terdapat form untuk meng-update nama dan price product dengan method PUT.

7. application.properties

```
spring.application.name=rest api

# Konfigurasi Database H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=la
spring.datasource.password=
#spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# Mengaktifkan Konsol H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

#mengatur frontend
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
server.port=8081
spring.thymeleaf.cache=false
```

Penjelasan : Database h2 disetting menggunakan “jdbc:h2:mem:testdb” dengan driverClassName “org.h2.Driver”. Kemudian, memberikan data username, yaitu “sa” tanpa password. Mengaktifkan konsol h2 dengan menggunakan “spring.h2.console.enabled=true” dan memiliki path “path=/h2-console”. Lalu, mengatur frontend dengan menggunakan “spring.thymeleaf” yang di mana “.prefix=classpath:/templates/” untuk menentukan lokasi template dan “.suffix=.html” untuk format file template. Menggunakan port 8081 dan “spring.thymeleaf.cache=false” untuk memudahkan pengembangan.

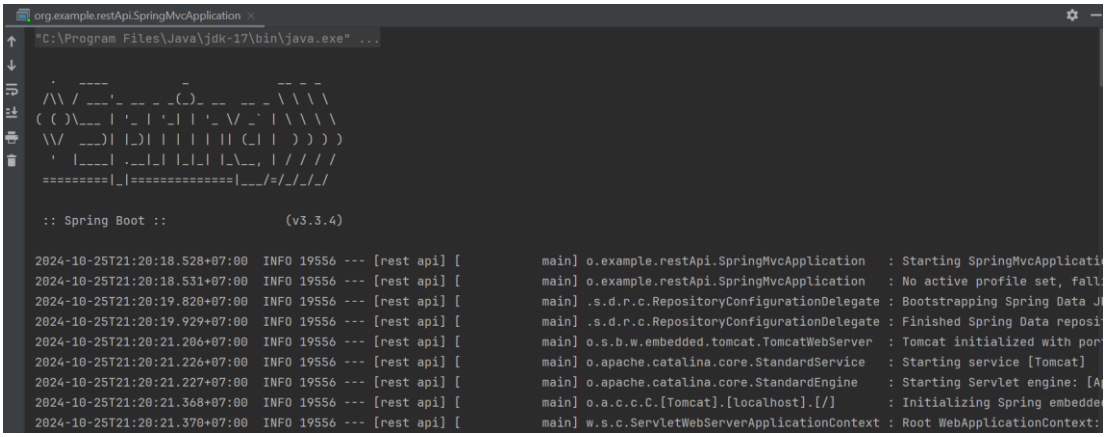
8. pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>org.example</groupId>
    <artifactId>restApi</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>rest api</name>
    <description>rest api</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies> Edit Starters...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
```

```xml
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Penjelasan : Memiliki groupId org.example dan artifactId restApi, Menggunakan java dengan versi 17. Pada dependencies terdapat spring-boot-starter-data-jpa yang berfungsi untuk mengimplementasikan JPA repositories, spring-boot-starter-thymeleaf yang berfungsi untuk mengonfigurasi template Thymeleaf secara otomatis dalam aplikasi web, spring-boot-starter-web yang berfungsi untuk menyediakan aplikasi web Spring Boot yang siap produksi, com.h2database:h2 yang berfugsi untuk pengujian, dan spring-boot-starter-test yang berfungsi menyediakan alat untuk menguji. Pada build plugin terdapat spring-boot-maven-plugin yang berfungsi untuk menyediakan dukungan Spring Boot di Apache Maven.

9. Output



a. Customer

Menampilkan list dari customer

Customer List

Add New Customer

Customer ID	Customer Name	Customer Email	Actions
1	John Doe	john@example.com	Delete / New Name / New Email / Update
2	Jane Smith	jane@example.com	Delete / New Name / New Email / Update

Menambahkan data baru

Customer List

Add New Customer

Customer ID	Customer Name	Customer Email	Actions
1	John Doe	john@example.com	Delete / New Name / New Email / Update
2	Jane Smith	jane@example.com	Delete / New Name / New Email / Update
3	Maria Gresia Plena Br Purba	mariagresia347@gmail.com	Delete / New Name / New Email / Update
4	mark	markGanteng@gmail.com	Delete / New Name / New Email / Update

Mengganti data nama dan email pada Customer ID 3

Customer List

Add New Customer

Customer ID	Customer Name	Customer Email	Actions
1	John Doe	john@example.com	Delete / New Name / New Email / Update
2	Jane Smith	jane@example.com	Delete / New Name / New Email / Update
3	maria	mariagresia347@gmail.com	Delete / New Name / New Email / Update
4	mark	markGanteng@gmail.com	Delete / New Name / New Email / Update

Menghapus data dengan customer ID 2

Customer List

Add New Customer

Customer ID	Customer Name	Customer Email	Actions
1	John Doe	john@example.com	Delete / New Name / New Email / Update
3	maria	mariagresia347@gmail.com	Delete / New Name / New Email / Update
4	mark	markGanteng@gmail.com	Delete / New Name / New Email / Update

b. Order

Menampilkan list dari order

Order List

Add New Order

Customer Name	Product Name	Product Quantity	Actions
John Doe	Laptop	2	Delete New Product Quantity Update
Jane Smith	Phone	1	Delete New Product Quantity Update

Menambahkan data baru

Order List

Add New Order

Customer Name	Product Name	Product Quantity	Actions
John Doe	Laptop	2	Delete New Product Quantity Update
Jane Smith	Phone	1	Delete New Product Quantity Update
Jane Smith	Laptop	1	Delete New Product Quantity Update
mark	Phone	3	Delete New Product Quantity Update

Mengganti data product name Laptop dengan pc dan menghapus data dengan customer name “Jane Smith”

Order List

Add New Order

Customer Name	Product Name	Product Quantity	Actions
John Doe	pc	2	Delete New Product Quantity Update
mark	Phone	3	Delete New Product Quantity Update

c. Product

Menampilkan list dari product

Product List

Add New Product

Product ID	Product Name	Product Price	Actions
1	Laptop	1200.0	Delete New Name New Price Update
2	Phone	800.0	Delete New Name New Price Update

Menambahkan data baru

Product List

Add New Product

Product ID	Product Name	Product Price	Actions
1	Laptop	1200.0	Delete New Name New Price Update
2	Phone	800.0	Delete New Name New Price Update
3	pulpen	3.0	Delete New Name New Price Update
4	buku tulis	1.5	Delete New Name New Price Update

Mengganti data product name laptop dengan pc

Product List

Add New Product

Product ID	Product Name	Product Price	Actions
1	pc	1200.0	Delete New Name New Price Update
2	Phone	800.0	Delete New Name New Price Update
3	pulpen	3.0	Delete New Name New Price Update
4	buku tulis	1.5	Delete New Name New Price Update

Menghapus data dengan product id 3

Product List

Add New Product

Product ID	Product Name	Product Price	Actions			
1	pc	1200.0	Delete	New Name	New Price	Update
2	Phone	800.0	Delete	New Name	New Price	Update
4	buku tulis	1.5	Delete	New Name	New Price	Update