PROGRAM STUDI INFORMATIKA
UNIVERSITAS SANATA DHARMA

PRAKTIKUM PBO LANJUT
MODUL SPRING BOOT-3

I. TUJUAN PRAKTIKUM

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

II. DASAR TEORI

1. Konsep Dasar Spring MVC

Spring MVC adalah salah satu komponen inti dari **Spring Framework** yang digunakan untuk membangun aplikasi web yang mengikuti arsitektur **Model-View-Controller (MVC)**. Pendekatan MVC memisahkan aplikasi menjadi tiga komponen utama:

- **Model**: Bagian yang mengelola data dan logika bisnis dari aplikasi.
- **View**: Bagian yang bertanggung jawab untuk menampilkan data kepada pengguna (biasanya berupa HTML).
- **Controller**: Bagian yang menangani permintaan pengguna, berinteraksi dengan model, dan mengembalikan respons ke view.

2. Alur Kerja Spring MVC

Alur kerja **Spring MVC** adalah sebagai berikut:

1. **Client Request**: Pengguna (client) mengirim permintaan HTTP ke aplikasi.
2. **DispatcherServlet**: Spring memiliki **DispatcherServlet** yang bertugas menerima semua permintaan web, kemudian mengarahkan permintaan ke controller yang tepat berdasarkan konfigurasi.
3. **Controller**: Controller memproses permintaan, berinteraksi dengan model untuk mengambil atau memanipulasi data.
4. **Model**: Controller mengisi model dengan data yang diperlukan untuk ditampilkan.
5. **View**: Setelah controller selesai, data yang ada dalam model akan dikirimkan ke view untuk dirender dan menghasilkan halaman HTML yang dikirimkan kembali kepada client.

3. Model-View-Controller dalam Spring MVC

3.1 Model

- **Model** adalah objek yang membawa data dari controller ke view. Dalam konteks Spring MVC, data yang dikirimkan ke view disimpan dalam objek model, seperti `Model` atau `ModelMap`.
- **Model** berisi logika bisnis dan data yang dibutuhkan oleh aplikasi. Dalam konteks Spring, model biasanya berupa objek yang merepresentasikan data dari basis data atau logika aplikasi lainnya.

3.2 View

- **View** adalah representasi data yang akan ditampilkan kepada pengguna. Dalam Spring MVC, view biasanya berupa file **HTML** atau **Thymeleaf template** yang akan direnderdan dikirimkan ke browser.
- View hanya bertanggung jawab untuk menampilkan data dan tidak melakukan logika bisnis.

3.3 Controller

- **Controller** adalah komponen yang menangani permintaan HTTP. Controller menerima permintaan dari pengguna, berinteraksi dengan **model**, dan mengembalikan **view** yang sesuai dengan data dari model.
- Controller dalam Spring biasanya didefinisikan sebagai **class** yang diberi anotasi `@Controller` dan metode-metode di dalamnya diberi anotasi `@RequestMapping` untuk memetakan permintaan HTTP ke metode yang tepat.

III. LATIHAN

Dalam praktikum ini kita akan menangani :

 Model: Membuat tiga entitas (Product, Customer, Order) dan menghubungkan mereka menggunakan relasi.

 Controller: Membuat controller untuk menangani operasi CRUD untuk semua entitas.

 View: Menggunakan **Thymeleaf** untuk menampilkan daftar produk, customer, dan order serta menyediakan form untuk menambah dan mengedit entitas.

 Database**:** Menggunakan **H2** sebagai database memori untuk menyimpan data.

Bagian 1: Membuat Proyek Spring Boot dengan Database

Langkah 1: Membuat Proyek Menggunakan Spring Boot Initializr

1. Buka **Spring Boot Initializr** di browser: https://start.spring.io.
2. Isikan pengaturan sebagai berikut:
   - **Project**: Maven
   - **Language**: Java
   - **Spring Boot Version**: 3.x (versi terbaru)
   - **Group**: `com.example`
   - **Artifact**: `rest-api`
   - **Name**: `REST API`
   - **Description**: Simple REST API with Spring Boot.
   - **Package Name**: `com.example.restapi`
   - **Packaging**: JAR
   - **Java Version**: 17 (atau sesuai versi Java di lingkungan pengembangan).
3. Tambahkan dependensi berikut:
   - **Spring Web** (untuk membuat REST API).
   - **Spring Data JPA** (untuk mengelola data)
   - **H2 Database** (untuk database in-memory)
   - Thymeleaf (untuk frontend).

4. Klik **Generate** untuk mengunduh proyek.
5. Ekstrak proyek yang diunduh, dan buka proyek di **IDE** (**NetBeans**).

Langkah 2: Struktur Proyek Spring Boot

1. Berikutnya lakukan perintah **clean and build** pada project.
2. Setelah proyek dibuka, Anda akan melihat struktur proyek dasar sebagai berikut:
   - **src/main/java**: Tempat kode sumber aplikasi.
   - **src/main/resources**: Tempat file konfigurasi seperti **application.properties**.
   - **pom.xml**: File Maven untuk manajemen dependensi.
3. Cek isi dari pom.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>com.example</groupId>
    <artifactId>rest-api-jpa-demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>rest-api-jpa-demo</name>
    <description>Demo project for Spring
Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
```

```
        <java.version>17</java.version>
        <start-
class>com.example.rest_api_jpa_demo.RestApiJpaDemoApplica
tion</start-class> <!-- Ganti dengan main class Anda -->

    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-
jpa</artifactId>
        </dependency>
        <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>
```

Langkah 3: Konfigurasi `application.properties`

Spring Boot memerlukan konfigurasi untuk mengaktifkan **H2 Database** dan **JPA**. Buka file **`src/main/resources/application.properties`** dan tambahkan konfigurasi berikut:

```
# Konfigurasi Database H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
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=8080
spring.thymeleaf.cache=false
```

Langkah 4: Membuat Model

Buat kelas **Product** yang akan menjadi entitas di database. Buat file **`Product.java`** di package **`com.example.restapidemo.model`**.

`Product.java`:

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;

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

```
    public Product(Long id, String name, double price) {
        this.name = name;
        this.description = description;
        this.price = price;    }

    // Getters and Setters
    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;
    }

    public double getPrice() {
        return price;
    }

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

Buat entitas **Customer** di package model. Ini adalah pelanggan yang memesan produk.

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

@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    @OneToMany(mappedBy = "customer", cascade =
CascadeType.ALL)
    private List<Orders> orders;
```

```
    // Constructor, Getters, Setters
    public Customer() {}

    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;
    }

    public String getEmail() {
        return email;
    }

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

    public List<Orders> getOrders() {
        return orders;
    }

    public void setOrders(List<Orders> orders) {
        this.orders = orders;
    }
}
```

Buat entitas **Orders** yang menghubungkan **Product** dan **Customer**.

```
import jakarta.persistence.*;

@Entity
public class Orders {
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private int quantity;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;

    // Constructor, Getters, Setters
    public Orders() {}

    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;
    }

    public int getQuantity() {
        return quantity;
    }

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

    public Customer getCustomer() {
        return customer;
    }

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

    public Product getProduct() {
```

```
        return product;
    }

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

Langkah 5: Membuat Repository

Buat repository untuk setiap entitas di package **`repository/`**. Spring Data JPA akan otomatis menghasilkan implementasi dasar CRUD.

ProductRepository.java:
```
import com.example.springmvc.model.Product;
import
org.springframework.data.jpa.repository.JpaRepository;

public       interface       ProductRepository       extends
JpaRepository<Product, Long> {
}
```

CustomerRepository.java:
```
import com.example.springmvc.model.Customer;
import
org.springframework.data.jpa.repository.JpaRepository;

public       interface       CustomerRepository       extends
JpaRepository<Customer, Long> {
}
```

OrderRepository.java:
```
import com.example.springmvc.model.Orders;
import
org.springframework.data.jpa.repository.JpaRepository;

public       interface       OrderRepository       extends
JpaRepository<Orders, Long> {
}
```

Langkah 6. Membuat Controller untuk Expose REST API

Buat controller untuk masing-masing entitas.

- **ProductController.java** untuk mengelola produk:
  ```
  org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.stereotype.Controller;
  ```

```
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;

import java.util.List;

@Controller
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @GetMapping("/products")
    public String listProducts(Model model) {
        List<Product> products =
productRepository.findAll();
        model.addAttribute("products", products);
        return "product_list";
    }

    @GetMapping("/product/new")
    public String showProductForm(Model model) {
        model.addAttribute("product", new Product());
        return "product_form";
    }

    @PostMapping("/product/save")
    public String saveProduct(@ModelAttribute Product
product) {
        productRepository.save(product);
        return "redirect:/products";
    }
}
```

- **CustomerController.java** untuk mengelola customer:

```
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
```

```
import
org.springframework.web.bind.annotation.ModelAttribute;

import java.util.List;

@Controller
public class CustomerController {

    @Autowired
    private CustomerRepository customerRepository;

    @GetMapping("/customers")
    public String listCustomers(Model model) {
        List<Customer> customers =
customerRepository.findAll();
        model.addAttribute("customers", customers);
        return "customer_list";
    }

    @GetMapping("/customer/new")
    public String showCustomerForm(Model model) {
        model.addAttribute("customer", new Customer());
        return "customer_form";
    }

    @PostMapping("/customer/save")
    public String saveCustomer(@ModelAttribute Customer
customer) {
        customerRepository.save(customer);
        return "redirect:/customers";
    }
}
```

- **OrderController.java** untuk mengelola order:

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;

import java.util.List;
```

```
@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("orders", orders);
        return "order_list";
    }

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

    @PostMapping("/order/save")
    public String saveOrder(@ModelAttribute Orders order)
{
        orderRepository.save(order);
        return "redirect:/orders";
    }
}
```

Bagian 2: View dengan Thymeleaf

Langkah 1: Membuat File HTML untuk Antarmuka Frontend

1. Pada folder bernama **`templates`** di dalam folder **`src/main/resources`** buat file html.
2. Buat file bernama **`product_list.html`** untuk menampilkan produk.

```
<!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>ID</th>
            <th>Name</th>
            <th>Price</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>
        </tr>
    </table>
</body>
</html>
```

2. Buat file bernama **`product_form.html`** untuk menambah/update produk.

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

        <!-- Input untuk harga produk -->
        <label>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>
```

3. Buat 2 file html untuk menampilkan dan menambah/update customer.

Langkah 2: Tambahkan data awal

1. Tambahkan kode program di SpringApplication.java (menyesuaikan)

```
package com.example.springmvc;

import com.example.springmvc.model.Customer;
import com.example.springmvc.model.Orders;
import com.example.springmvc.model.Product;
import com.example.springmvc.repository.CustomerRepository;
import com.example.springmvc.repository.OrderRepository;
import com.example.springmvc.repository.ProductRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

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("Laptop", 1200);
        Product product2 = new Product("Phone", 800);
        productRepository.saveAll(Arrays.asList(product1,
product2));

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

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

Langkah 3: Jalankan aplikasi

1. Lakukan clean and build
2. **Melalui IDE**: Klik kanan pada project dan pilih **Run**.
3. Setelah aplikasi berjalan, buka web browser dan akses
   - **http://localhost:8080/products** untuk daftar produk.
   - **http://localhost:8080/customers** untuk daftar customer.
   - **http://localhost:8080/orders** untuk daftar order.

Bagian 3: Menggunakan H2 Database Console

Karena Anda menggunakan H2 sebagai database in-memory, Anda juga bisa memeriksa data yang disimpan melalui **H2 Console**:

1. Pastikan H2 Console diaktifkan di file application.properties
   - Akses H2 Console di browser: http://localhost:8080/h2-console.
   - Gunakan koneksi berikut:
3. JDBC URL: jdbc:h2:mem:testdb
4. Username: sa
5. Password: password (sesuai konfigurasi di application.properties).

III. LAPORAN

1. Penjelasan kode program, capture GUI untuk setiap soal .

Referensi Utama:

1. Craig Walls, "Spring in Action," 6th Edition, Manning Publications, 2022.
2. Juergen Hoeller, "Spring Framework Reference Documentation," Spring.io.