PROGRAM STUDI INFORMATIKA
UNIVERSITAS SANATA DHARMA

PRAKTIKUM PBO LANJUT
MODUL SPRING BOOT-1

I. TUJUAN PRAKTIKUM

1. Memahami cara membuat aplikasi **Spring Boot REST API**.
2. Memahami konsep **CRUD** (Create, Read, Update, Delete).
3. Menggunakan **H2 Database** untuk menyimpan data produk.
4. Menggunakan **Spring Data JPA** untuk operasi CRUD..

I. DASAR TEORI

I. Data Access pada Spring Boot

Data Access adalah proses mengelola dan memanipulasi data dari sumber penyimpanan, seperti **database relasional** (MySQL, PostgreSQL, H2) atau **non-relasional** (MongoDB). Pada Spring Boot, terdapat berbagai cara untuk mengakses dan mengelola data menggunakan berbagai lapisan dan teknologi, seperti **Spring Data JPA**, **JDBC**, atau **Hibernate**.

Spring Boot menyediakan dukungan bawaan untuk **ORM (Object Relational Mapping)** melalui **Spring Data JPA**, yang memungkinkan pengembang melakukan operasi **CRUD** (Create, Read, Update, Delete) dengan kode minimal tanpa perlu menulis query SQL secara manual.

Komponen Utama dalam Data Access pada Spring Boot:

1. Spring Data JPA (Java Persistence API)

Spring Data JPA adalah modul dari **Spring Framework** yang dirancang untuk mengintegrasikan **JPA** (Java Persistence API) dengan **Spring**. JPA adalah standar untuk **ORM**, yang memetakan objek Java ke tabel dalam database relasional.
Spring Data JPA menyediakan cara mudah untuk berinteraksi dengan database menggunakan interface `JpaRepository` dan mendukung berbagai operasi tanpa perlu menulis query SQL.
Keunggulan Spring Data JPA:

- **CRUD Otomatis**: Menyediakan metode CRUD dasar (create, read, update, delete) tanpa harus menulis kode SQL atau query HQL/JPQL.
- **Query Custom**: Mendukung pembuatan query khusus menggunakan nama metode atau anotasi `@Query`.
- **Pagination dan Sorting**: Mendukung paginasi dan sorting data secara otomatis.
- **Integrasi dengan Hibernate**: Biasanya menggunakan **Hibernate** sebagai implementasi default untuk ORM.

2. JpaRepository:

Spring Boot menggunakan **`JpaRepository`** sebagai antarmuka untuk melakukan operasi CRUD dan operasi lain pada entitas. Dengan **`JpaRepository`**, pengembang tidak perlu menulis kode SQL secara manual untuk operasi dasar.

Contoh `JpaRepository`:

```
@Repository
public interface ProductRepository extends JpaRepository<Product,
Long> {
  // JpaRepository menyediakan metode CRUD otomatis seperti findAll(),
save(), deleteById() dll.
}
```

Pada contoh di atas:

- **`ProductRepository`** adalah repository untuk entitas **`Product`**.
- **`JpaRepository<Product, Long>`** berarti repository ini beroperasi pada entitas **Product**, dan tipe data ID dari entitas **Product** adalah **Long**.

3. Entity Class:

Entity adalah kelas yang dipetakan ke tabel di database. Spring Boot, dengan bantuan JPA dan Hibernate, secara otomatis memetakan **kelas Java** ke **tabel** dan **properti** dari kelas Java ke **kolom** pada tabel di database.

Untuk menandai sebuah kelas sebagai entitas JPA, kita menggunakan anotasi **`@Entity`**.

Contoh Entity:

```
@Entity
public class Product {

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

    // Getters and Setters
}
```

Pada contoh di atas:

- **`@Entity`**: Menandakan bahwa kelas ini adalah entitas yang akan dipetakan ke tabel di database.
- **`@Id`**: Menandakan bahwa properti ini adalah **primary key** dari tabel.
- **`@GeneratedValue`**: Menginstruksikan bahwa nilai ID akan dihasilkan secara otomatis oleh database.

4. Repository Pattern:

Repository Pattern adalah pola desain yang digunakan untuk mengelola akses data dan operasi CRUD. Spring Boot mengadopsi pola ini dengan menyediakan **JpaRepository** yang memungkinkan pemisahan logika akses data dari logika bisnis.
Dengan repository, kita memiliki metode CRUD seperti **`findAll()`**, **`findById()`**, **`save()`**, dan **`deleteById()`** yang sudah disediakan oleh **JpaRepository** tanpa harus menulis kode SQL secara manual.

5. Data Source dan Konfigurasi:

DataSource adalah representasi dari koneksi database. Pada Spring Boot, konfigurasi database didefinisikan dalam file **`application.properties`** atau **`application.yml`**, dan Spring Boot akan membuat **DataSource** secara otomatis berdasarkan konfigurasi tersebut.

Contoh konfigurasi database H2 di **`application.properties`**:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

- `spring.datasource.url`: URL untuk menghubungkan ke database H2.
- `spring.datasource.driverClassName`: Driver JDBC yang digunakan (H2).
- `spring.jpa.hibernate.ddl-auto`: Menginstruksikan Hibernate untuk membuat/memperbarui tabel secara otomatis berdasarkan entitas.

6. H2 Database

H2 adalah **in-memory database** yang ringan dan digunakan terutama untuk pengembangan dan pengujian. Karena H2 adalah **in-memory**, data hanya ada selama aplikasi berjalan. Setelah aplikasi di-restart, data akan hilang kecuali H2 disetel untuk menyimpan data ke file.
Fitur H2:

- **In-memory database**: Tidak memerlukan instalasi fisik atau pengaturan kompleks, data disimpan di memori.
- **Embedded**: H2 berjalan di dalam aplikasi Spring Boot.
- **Konsol H2**: Spring Boot menyediakan akses ke konsol database H2 melalui web browser, memungkinkan pengguna untuk mengelola dan melihat data.

II. Arsitektur Data Access di Spring Boot

Lapisan Data Access:

1. **Entity Class (Model)**: Representasi objek Java yang dipetakan ke tabel di database.
    - Misalnya, `Product` dipetakan ke tabel **Product** dalam database.
2. **Repository**: Menggunakan **Spring Data JPA** untuk mengelola akses ke data.
    - Misalnya, `ProductRepository` digunakan untuk operasi CRUD pada entitas **Product**.
3. **Service Layer**: Mengelola logika bisnis dan memanggil repository untuk mengakses data.
    - Misalnya, `ProductService` memanggil `ProductRepository` untuk mengambil atau menyimpan data produk.
4. **Controller Layer**: Mengelola request dari klien (frontend) dan memanggil service untuk operasi CRUD.
    - Misalnya, `ProductController` menyediakan endpoint REST API yang diakses melalui HTTP request.

Aliran Data pada Operasi CRUD:

1. **Create**: Ketika aplikasi menerima request untuk menambahkan produk baru (misalnya dari frontend), request tersebut diproses oleh **controller** yang kemudian memanggil **service**. Service memanggil **repository** untuk menyimpan data ke dalam database H2.
2. **Read**: Ketika frontend meminta data produk, controller memanggil service untuk mengambil data dari repository, yang kemudian mengambil data dari database dan mengembalikannya ke frontend dalam format JSON.
3. **Update**: Untuk memperbarui data produk, controller menerima data baru, service mengirimkannya ke repository untuk memperbarui entitas dalam database.
4. **Delete**: Controller menerima request untuk menghapus produk, service memerintahkan repository untuk menghapus entitas dari database.

III. Thymeleaf

1. Pengenalan Thymeleaf

Thymeleaf adalah sebuah template engine berbasis Java yang digunakan dalam pengembangan aplikasi web. Ia dirancang untuk memproses dan menghasilkan HTML, XML, JavaScript, CSS, atau teks mentah. Salah satu fitur utama dari Thymeleaf adalah kemampuannya untuk berfungsi dalam dua mode: **mode server** dan **mode standalone (offline)**. Hal ini memungkinkan template Thymeleaf untuk dirender baik saat aplikasi sedang berjalan di server maupun ketika template hanya diakses sebagai dokumen statis di browser.

Thymeleaf sering digunakan bersama **Spring Boot** sebagai template engine default untuk rendering view dalam aplikasi web yang berbasis Spring MVC. Dengan integrasi yang mendalam dengan Spring, Thymeleaf sangat populer karena kemudahan penggunaannya dan kemampuan untuk menghasilkan tampilan dinamis yang kuat dan bersih.

2. Fitur Utama Thymeleaf

Beberapa fitur utama Thymeleaf yang membuatnya populer di kalangan pengembang adalah:

- **Natural Templates**: Template Thymeleaf dapat dilihat dan diedit seperti dokumen HTML biasa, yang berarti bahwa mereka tetap valid sebagai HTML yang bisa diakses oleh browser bahkan di luar server. Ini memungkinkan kolaborasi yang mudah antara pengembang frontend dan backend.
- **Ekspresi Standar**: Thymeleaf mendukung sintaks ekspresi standar seperti `${...}` untuk menampilkan nilai dari model atau variable, serta mendukung operasi logika, iterasi, dan pemanggilan fungsi.
- **Dukungan untuk Fragment**: Thymeleaf mendukung penggunaan **template fragment** yang dapat digunakan kembali dalam berbagai bagian halaman web. Ini sangat berguna untuk menyusun komponen UI yang bersifat dinamis.
- **Mode Dual (Offline dan Online)**: Thymeleaf dapat digunakan baik dalam mode "server-side rendering" maupun mode offline di mana template masih dapat divisualisasikan oleh frontend designer di browser.

3. Arsitektur Thymeleaf

Thymeleaf bekerja dengan menggabungkan data dari model (dalam konteks Spring MVC, model biasanya diisi oleh controller) dengan template HTML yang berisi tag-tag khusus Thymeleaf. Thymeleaf memproses template ini dan menghasilkan halaman HTML dinamis yang akan dikirim ke browser pengguna.

Proses alur kerjanya adalah:

- Controller dalam Spring Boot menerima permintaan HTTP dan mengembalikan model dengan data yang akan digunakan.
- Thymeleaf menggabungkan model dengan template yang sesuai dan menggantikan placeholder dengan data dari model.
- Hasil akhirnya adalah HTML dinamis yang dikirim kembali ke browser pengguna.

4. Ekspresi Thymeleaf

Thymeleaf mendukung berbagai jenis ekspresi untuk manipulasi data. Berikut adalah beberapa yang paling sering digunakan:

- **Variable Expression (`${}`)**: Digunakan untuk mengakses data dalam model, seperti `${product.name}` untuk menampilkan nama produk.
- **Selection Variable (`*{}`)**: Dalam konteks form, ekspresi ini digunakan untuk mengakses properti objek yang sedang diproses, misalnya `*{id}` untuk mengakses **ID** objek saat rendering form.
- **Message Expression (`#{}`)**: Digunakan untuk mengambil pesan atau teks dari berkas properti lokal (untuk internasionalisasi atau i18n).
- **Link Expression (`@{}`)**: Digunakan untuk membuat tautan atau URL, seperti `@{/products}` untuk membuat tautan ke halaman daftar produk.
- **Fragment Expression (`~{}`)**: Digunakan untuk merujuk pada fragmen template, memungkinkan komposisi dan penggunaan kembali bagian halaman.

5. Struktur Template Thymeleaf

Struktur template Thymeleaf biasanya merupakan dokumen HTML yang berisi elemen-elemen HTML standar yang diperkaya dengan atribut-atribut Thymeleaf. Atribut-atribut tersebut memberi instruksi kepada template engine tentang cara mengganti placeholder dengan data dinamis dari model.

Pada template ini:

- **`th:field`** digunakan untuk menghubungkan input form dengan properti dari objek dalam model (misalnya **`name`**, **`description`**, dan **`price`**).
- **`th:action`** menentukan URL tempat form akan dikirim setelah form di-submit.

6. Tag dan Atribut Thymeleaf yang Umum
Thymeleaf menggunakan berbagai tag dan atribut untuk melakukan render dinamis terhadap halaman HTML. Berikut adalah beberapa atribut Thymeleaf yang umum:

- **`th:text`**: Menampilkan teks dinamis dari model
- **`th:href`**: Mengatur URL tautan dinamis.
- **`th:if`** dan **`th:unless`**: Menampilkan elemen berdasarkan kondisi boolean
- **`th:each`**: Untuk iterasi atau looping melalui koleksi

8. Thymeleaf dan Spring Boot
Dalam proyek **Spring Boot**, Thymeleaf adalah template engine yang sering digunakan untuk menghasilkan halaman dinamis. Spring Boot secara otomatis mengonfigurasi Thymeleaf jika dependensi **`spring-boot-starter-thymeleaf`** ditambahkan dalam **pom.xml**.

Teori tentang Thymeleaf

1. Pengenalan Thymeleaf

Thymeleaf adalah sebuah template engine berbasis Java yang digunakan dalam pengembangan aplikasi web. Ia dirancang untuk memproses dan menghasilkan HTML, XML, JavaScript, CSS, atau teks mentah. Salah satu fitur utama dari Thymeleaf adalah kemampuannya untuk berfungsi dalam dua mode: **mode server** dan **mode standalone (offline)**. Hal ini memungkinkan template Thymeleaf untuk dirender baik saat aplikasi sedang berjalan di server maupun ketika template hanya diakses sebagai dokumen statis di browser.

Thymeleaf sering digunakan bersama **Spring Boot** sebagai template engine default untuk rendering view dalam aplikasi web yang berbasis Spring MVC. Dengan integrasi yang mendalam dengan Spring, Thymeleaf sangat populer karena kemudahan penggunaannya dan kemampuan untuk menghasilkan tampilan dinamis yang kuat dan bersih.

2. Fitur Utama Thymeleaf

Beberapa fitur utama Thymeleaf yang membuatnya populer di kalangan pengembang adalah:

- **Natural Templates**: Template Thymeleaf dapat dilihat dan diedit seperti dokumen HTML biasa, yang berarti bahwa mereka tetap valid sebagai HTML yang bisa diakses oleh browser bahkan di luar server. Ini memungkinkan kolaborasi yang mudah antara pengembang frontend dan backend.

- **Ekspresi Standar**: Thymeleaf mendukung sintaks ekspresi standar seperti `${...}` untuk menampilkan nilai dari model atau variable, serta mendukung operasi logika, iterasi, dan pemanggilan fungsi.
- **Dukungan untuk Fragment**: Thymeleaf mendukung penggunaan **template fragment** yang dapat digunakan kembali dalam berbagai bagian halaman web. Ini sangat berguna untuk menyusun komponen UI yang bersifat dinamis.
- **Mode Dual (Offline dan Online)**: Thymeleaf dapat digunakan baik dalam mode "server-side rendering" maupun mode offline di mana template masih dapat divisualisasikan oleh frontend designer di browser.

3. Arsitektur Thymeleaf

Thymeleaf bekerja dengan menggabungkan data dari model (dalam konteks Spring MVC, model biasanya diisi oleh controller) dengan template HTML yang berisi tag-tag khusus Thymeleaf. Thymeleaf memproses template ini dan menghasilkan halaman HTML dinamis yang akan dikirim ke browser pengguna.

Proses alur kerjanya adalah:

- Controller dalam Spring Boot menerima permintaan HTTP dan mengembalikan model dengan data yang akan digunakan.
- Thymeleaf menggabungkan model dengan template yang sesuai dan menggantikan placeholder dengan data dari model.
- Hasil akhirnya adalah HTML dinamis yang dikirim kembali ke browser pengguna.

4. Ekspresi Thymeleaf

Thymeleaf mendukung berbagai jenis ekspresi untuk manipulasi data. Berikut adalah beberapa yang paling sering digunakan:

- **Variable Expression (`${}`)**: Digunakan untuk mengakses data dalam model, seperti `${product.name}` untuk menampilkan nama produk.
- **Selection Variable (`*{}`)**: Dalam konteks form, ekspresi ini digunakan untuk mengakses properti objek yang sedang diproses, misalnya `*{id}` untuk mengakses **ID** objek saat rendering form.
- **Message Expression (`#{}`)**: Digunakan untuk mengambil pesan atau teks dari berkas properti lokal (untuk internasionalisasi atau i18n).
- **Link Expression (`@{}`)**: Digunakan untuk membuat tautan atau URL, seperti `@{/products}` untuk membuat tautan ke halaman daftar produk.
- **Fragment Expression (`~{}`)**: Digunakan untuk merujuk pada fragmen template, memungkinkan komposisi dan penggunaan kembali bagian halaman.

5. Struktur Template Thymeleaf

Struktur template Thymeleaf biasanya merupakan dokumen HTML yang berisi elemen-elemen HTML standar yang diperkaya dengan atribut-atribut Thymeleaf. Atribut-atribut tersebut memberi instruksi kepada template engine tentang cara mengganti placeholder dengan data dinamis dari model.

Contoh struktur template Thymeleaf sederhana:

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

    <form th:action="@{/product/save}" method="post">
        <input type="hidden" th:field="*{id}">
        <div>
            <label>Name:</label>
            <input type="text" th:field="*{name}" placeholder="Product
Name">
        </div>
        <div>
            <label>Description:</label>
            <input type="text" th:field="*{description}"
placeholder="Product Description">
        </div>
        <div>
            <label>Price:</label>
            <input type="number" th:field="*{price}"
placeholder="Product Price">
        </div>
        <button type="submit">Save</button>
    </form>
</body>
</html>
```

Pada template ini:

- **`th:field`** digunakan untuk menghubungkan input form dengan properti dari objek dalam model (misalnya **`name`**, **`description`**, dan **`price`**).
- **`th:action`** menentukan URL tempat form akan dikirim setelah form di-submit.

6. Tag dan Atribut Thymeleaf yang Umum

Thymeleaf menggunakan berbagai tag dan atribut untuk melakukan render dinamis terhadap halaman HTML. Berikut adalah beberapa atribut Thymeleaf yang umum:

- **`th:text`**: Menampilkan teks dinamis dari model.
  ```
  html
  <p th:text="${product.name}">Nama Produk</p>
  ```

- **`th:href`**: Mengatur URL tautan dinamis.
  ```
  html
  <a th:href="@{/product/list}">Daftar Produk</a>
  ```
- **`th:if`** dan **`th:unless`**: Menampilkan elemen berdasarkan kondisi boolean.
  ```
  html
  <div th:if="${product.price > 1000}">
      Produk premium
  </div>
  ```
- **`th:each`**: Untuk iterasi atau looping melalui koleksi.
  ```
  html
  <ul>
    <li th:each="product : ${products}"
  th:text="${product.name}"></li>
  </ul>
  ```

7. Fragment dalam Thymeleaf

Thymeleaf fragment digunakan untuk mendefinisikan bagian template yang dapat digunakan kembali di berbagai halaman. Hal ini sangat berguna untuk membangun bagian yang sering diulang seperti header, footer, atau menu navigasi.

Contoh fragment:

Template **`header.html`**:

```
html
Copy code
<div th:fragment="header">
    <h1>My Website</h1>
    <p>Welcome to my website</p>
</div>
```

Template utama **`index.html`** yang memanggil fragment tersebut:

```
html
Copy code
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
    <div th:insert="header :: header"></div>
    <div>Main content goes here</div>
</body>
</html>
```

8. Thymeleaf dan Spring Boot

Dalam proyek **Spring Boot**, Thymeleaf adalah template engine yang sering digunakan untuk menghasilkan halaman dinamis. Spring Boot secara otomatis mengonfigurasi Thymeleaf jika dependensi **`spring-boot-starter-thymeleaf`** ditambahkan dalam **pom.xml**.

Contoh dependensi **Maven** untuk Thymeleaf dalam **pom.xml**:

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

Setelah dependensi ini ditambahkan, template Thymeleaf akan secara otomatis diproses ketika aplikasi Spring Boot dijalankan, dan template HTML yang disimpan di `src/main/resources/templates` akan diambil oleh **Spring MVC**.

9. Keuntungan Menggunakan Thymeleaf

Beberapa keuntungan utama menggunakan Thymeleaf adalah:

- **Integrasi yang baik dengan Spring**: Thymeleaf terintegrasi dengan sangat baik dalam arsitektur Spring, sehingga memudahkan pengelolaan model dan view.
- **Readable HTML templates**: Template Thymeleaf tetap valid sebagai HTML biasa, membuatnya mudah dibaca dan digunakan oleh desainer web dan developer backend.
- **Dukungan untuk fragment dan template reuse**: Ini memungkinkan pengembangan modular dan pemeliharaan yang lebih baik.

III. LATIHAN

Bagian 1: Membuat Proyek Spring Boot dengan Database

Pada langkah ini, kita akan menggantikan repository in-memory sebelumnya dengan **Spring Data JPA** yang terhubung ke **H2 Database**. Kita akan memanfaatkan **JpaRepository** yang disediakan oleh Spring untuk melakukan operasi CRUD langsung dari 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-demo`
   - **Name**: `REST API Demo`
   - **Description**: Simple REST API with Spring Boot.
   - **Package Name**: `com.example.restapidemo`
   - **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 Produk (Product)

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;
    }
}
```

- **`@Entity`**: Anotasi ini memberi tahu JPA bahwa kelas ini adalah entitas yang akan dipetakan ke tabel dalam database.
- **`@Id`**: Menandakan bahwa **id** adalah primary key.
- **`@GeneratedValue`**: Menetapkan strategi untuk menghasilkan nilai ID secara otomatis.

Langkah 5: Membuat Repository Sederhana

Repository bertanggung jawab untuk mengelola akses data, seperti mengambil, menambah, memperbarui, dan menghapus produk dari database. Buat file **`ProductRepository.java`** di package **`com.example.restapidemo.repository`**.

```
import com.example.restapidemo.model.Product;
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
```

```
public interface ProductRepository extends
JpaRepository<Product, Long> {
    // JpaRepository sudah menyediakan operasi CRUD
}
```

`JpaRepository`: Menyediakan operasi CRUD otomatis untuk entitas **Product**.

Langkah 6. Membuat Service untuk Logika Bisnis

Layanan ini akan berfungsi sebagai lapisan bisnis untuk mengelola produk. Buat file `ProductService.java` di package **`com.example.restapidemo.service`.**

```
import com.example.restapidemo.model.Product;
import
com.example.restapidemo.repository.ProductRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;

    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    public Optional<Product> getProductById(Long id) {
        return productRepository.findById(id);
    }

    public Product saveProduct(Product product) {
        return productRepository.save(product);
    }

    public void deleteProduct(Long id) {
        productRepository.deleteById(id);
    }}
```

Langkah 7. Membuat Controller untuk Expose REST API

Controller bertanggung jawab untuk menangani permintaan HTTP dari klien dan mengarahkan operasi CRUD yang sesuai. Buat file **`ProductController.java`** di package **`com.example.restapidemo.controller`**.:

```
import com.example.restapidemo.model.Product;
import com.example.restapidemo.service.ProductService;
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.PathVariable;
import
org.springframework.web.bind.annotation.PostMapping;
import java.util.List;

@Controller
public class ProductController {
    @Autowired
    private ProductService productService;

    // Mendapatkan semua produk
    @GetMapping("/products")
    public String viewProducts(Model model) {
         List<Product> products
        =productService.getAllProducts();
        model.addAttribute("products", products);
        return "products";
        // Mengarahkan ke file products.html
    }

     // Menampilkan form untuk menambahkan produk baru
    @GetMapping("/product/new")
    public String showCreateProductForm(Model model) {
        Product product = new Product();
        model.addAttribute("product", product);
        System.out.println("Product object sent to the
view: " + product);
        return "product_form";
    }

    // Menyimpan produk baru
    @PostMapping("/product/save")
    public String saveProduct(Product product) {
        productService.saveProduct(product);
```

```
        return "redirect:/products";
    }

    // Menampilkan form untuk mengedit produk
    @GetMapping("/product/edit/{id}")
    public String showEditProductForm(@PathVariable("id")
         Long id, Model model) {
        Product product =
productService.getProductById(id).orElseThrow(() -> new
IllegalArgumentException("Invalid product Id:" + id));
        model.addAttribute("product", product);
        return "product_form";
        // Mengarahkan ke file product_form.html
    }

    // Menghapus produk
    @GetMapping("/product/delete/{id}")
    public String deleteProduct(@PathVariable("id") Long
id) {
        productService.deleteProduct(id);
        return "redirect:/products";
    }}
```

- **`@RestController`**: Menandakan bahwa kelas ini adalah REST Controller yang menangani permintaan HTTP dan mengembalikan data dalam format JSON.
- **CRUD Methods**: Terdapat metode untuk menambah, mendapatkan, memperbarui, dan menghapus produk.

Bagian 2: Frontend dengan Thymeleaf

Untuk antarmuka frontend, kita akan menggunakan Thymeleaf. Berikut adalah contoh file HTML untuk menampilkan dan mengelola produk..

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 **`products.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">
        <thead>
            <tr>
```

```
                <th>ID</th>
                <th>Name</th>
                <th>Description</th>
                <th>Price</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="product : ${products}">
                <td th:text="${product.id}">ID</td>
                <td th:text="${product.name}">Product
Name</td>
                <td
th:text="${product.description}">Product Description</td>
                <td th:text="${product.price}">Product
Price</td>
                <td>
                    <a
th:href="@{/product/edit/{id}(id=${product.id})}">Edit</a
> |
                    <a
th:href="@{/product/delete/{id}(id=${product.id})}"
                       onclick="return confirm('Are you
sure you want to delete this product?');">Delete</a>
                </td>
            </tr>
        </tbody>
    </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>


        <form th:action="@{/product/save}" method="post">
            <input type="hidden"
th:field="*{product.id}">

            <div>
                <label>Name:</label>
```

```
                <input type="text"
th:field="*{product.name}" placeholder="Product Name"
required>
            </div>

            <div>
                <label>Description:</label>
                <input type="text"
th:field="*{product.description}" placeholder="Product
Description" required>
            </div>

            <div>
                <label>Price:</label>
                <input type="number"
th:field="*{product.price}" placeholder="Product Price"
step="0.01" required>
            </div>

            <div>
                <button type="submit">Save</button>
            </div>
        </form>

    </body>
</html>
```

Langkah 2: Menjalankan Aplikasi

1. Stop aplikasi sebelumnya,
2. Lakukan clean and build
3. **Melalui IDE**: Klik kanan pada project dan pilih **Run**.
4. Setelah aplikasi berjalan, buka web browser dan akses **http://localhost:8080/products**

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).

II. 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.