PROGRAM STUDI INFORMATIKA
UNIVERSITAS SANATA DHARMA

PRAKTIKUM PBO LANJUT
MODUL SPRING BOOT-1

I. TUJUAN PRAKTIKUM

1. Mahasiswa memahami cara menginisiasi proyek **Spring Boot** dan membangun REST API.
2. Mahasiswa mampu membuat REST API sederhana yang menangani operasi CRUD (Create, Read, Update, Delete)..

II. DASAR TEORI

1. Spring Boot

Spring Boot adalah framework berbasis **Java** yang dibangun di atas **Spring Framework** dan dirancang untuk menyederhanakan proses pengembangan aplikasi Java dengan konfigurasi minimal. Spring Boot memungkinkan pengembang untuk membuat aplikasi **standalone** yang siap dijalankan tanpa memerlukan konfigurasi manual yang kompleks seperti pada aplikasi Java tradisional berbasis Spring.

Fungsi Utama Spring Boot :

- **Standalone**: Spring Boot memungkinkan aplikasi berjalan sebagai aplikasi Java yang mandiri. Ini berarti Anda tidak memerlukan server aplikasi eksternal seperti Tomcat atau Jetty, karena Spring Boot sudah memiliki embedded server bawaan.
- **Auto Configuration**: Spring Boot secara otomatis mengonfigurasi aplikasi berdasarkan dependensi yang Anda tambahkan di dalam proyek. Pengembang tidak perlu menulis banyak konfigurasi manual seperti dalam Spring tradisional.
- **Production-ready**: Spring Boot menyediakan fitur bawaan seperti monitoring, health checks, dan management endpoints untuk memudahkan pengelolaan aplikasi dalam lingkungan produksi.
- **Spring Boot Initializr**: Alat untuk memulai proyek Spring Boot dengan cepat melalui web interface. Pengembang hanya perlu memilih dependensi yang dibutuhkan, dan Spring Boot Initializr akan menghasilkan proyek dasar yang siap digunakan.

.

2. Fitur Kunci Spring Boot

1. Spring Boot Starter

Spring Boot menyediakan **Starter POMs** yang memudahkan pengelolaan dependensi. Starter adalah sekumpulan dependensi yang dioptimalkan untuk kasus penggunaan tertentu. Beberapa starter yang umum adalah:

- **spring-boot-starter-web**: Untuk membuat aplikasi web dan REST API.

- **spring-boot-starter-data-jpa**: Untuk integrasi dengan database menggunakan JPA (Java Persistence API).
- **spring-boot-starter-security**: Untuk menambahkan fitur keamanan dalam aplikasi.

Starter menghemat waktu pengembang dengan memberikan konfigurasi standar yang sudah siap digunakan. URL: https://start.spring.io

2. Auto-Configuration

Salah satu fitur utama Spring Boot adalah **auto-configuration**, yang secara otomatis mengonfigurasi komponen aplikasi berdasarkan dependensi yang ada di classpath. Spring Boot mendeteksi pustaka yang Anda gunakan dan menyiapkan pengaturan default untuk memudahkan proses pengembangan. Misalnya:

III. Jika Anda menambahkan **Spring Data JPA**, Spring Boot akan mengonfigurasi koneksi ke database secara otomatis.
IV. Jika Anda menambahkan dependensi **Spring Web**, Spring Boot akan mengonfigurasi **Tomcat** sebagai embedded server.

3. Spring Boot Actuator

Spring Boot Actuator adalah salah satu modul paling penting di Spring Boot yang menambahkan berbagai endpoint untuk memantau dan mengelola aplikasi. Endpoint ini memberikan informasi tentang aplikasi, termasuk informasi konfigurasi, statistik runtime, logging, dan health status.

Contoh endpoint yang disediakan Actuator:

- `/health`: Menampilkan status kesehatan aplikasi.
- `/metrics`: Menampilkan metrik performa.
- `/info`: Menampilkan informasi tambahan yang dapat dikonfigurasi (misalnya, versi aplikasi).

3. REST API (Representational State Transfer API)

REST API adalah arsitektur yang digunakan untuk membuat layanan web yang dapat diakses oleh berbagai aplikasi melalui protokol HTTP. REST (Representational State Transfer) adalah gaya arsitektur untuk mendesain aplikasi yang terdistribusi, yang memungkinkan interaksi antara klien dan server melalui permintaan HTTP.

REST API memungkinkan aplikasi untuk saling berkomunikasi melalui jaringan dengan menggunakan operasi HTTP standar seperti **GET**, **POST**, **PUT**, dan **DELETE**. REST API sangat populer karena kesederhanaannya dan kemampuannya untuk diakses oleh berbagai platform, termasuk web, mobile, dan desktop.

REST API didasarkan pada beberapa prinsip utama:

1. Stateless

Setiap permintaan dari klien ke server harus mandiri, artinya server tidak menyimpan informasi tentang status atau konteks klien antara permintaan. Semua data yang diperlukan untuk memproses permintaan harus disertakan dalam setiap permintaan.

2. Client-Server Architecture
REST API memisahkan tanggung jawab antara **klien** dan **server**. Klien meminta sumber daya (resource), sementara server menyediakan sumber daya yang diminta. Klien dan server berkomunikasi melalui protokol HTTP.
3. Resource-Oriented
Dalam REST API, setiap entitas atau objek yang ditangani (seperti **Book** dalam kasus "Library Management") disebut sebagai **resource**. Setiap resource diidentifikasi oleh **URI** (Uniform Resource Identifier).
4. HTTP Methods
REST API menggunakan metode HTTP standar untuk melakukan operasi CRUD pada resource:

- **GET**: Digunakan untuk mengambil data (Read).
- **POST**: Digunakan untuk membuat resource baru (Create).
- **PUT**: Digunakan untuk memperbarui resource yang sudah ada (Update).
- **DELETE**: Digunakan untuk menghapus resource (Delete).

5. Representasi Resource
Data dalam REST API sering kali diwakili dalam format yang dapat dipahami oleh manusia dan mesin, seperti **JSON** (JavaScript Object Notation) atau **XML**. JSON adalah format yang paling umum digunakan untuk REST API karena ringkas dan mudah dibaca.
6. HTTP Status Codes
REST API menggunakan **HTTP status codes** untuk memberi tahu klien tentang status permintaan mereka. Beberapa status code umum meliputi:

- **200 OK**: Permintaan berhasil dan resource dikembalikan.
- **201 Created**: Resource berhasil dibuat.
- **400 Bad Request**: Permintaan klien tidak valid.
- **404 Not Found**: Resource yang diminta tidak ditemukan.
- **500 Internal Server Error**: Terjadi kesalahan di server.

REST API memudahkan pengembangan aplikasi yang terdistribusi dengan beberapa manfaat:

1. **Keterpisahan Antara Frontend dan Backend**: REST API memungkinkan frontend (misalnya aplikasi web atau mobile) dan backend untuk berkomunikasi tanpa bergantung satu sama lain.
2. **Format Data Fleksibel**: REST API biasanya mengembalikan data dalam format **JSON** yang dapat dengan mudah diproses oleh berbagai aplikasi dan platform.
3. **Kemudahan Integrasi**: Aplikasi yang menggunakan REST API dapat dengan mudah diintegrasikan dengan aplikasi lain atau layanan pihak ketiga.
4. **Efisiensi Jaringan**: REST API memanfaatkan HTTP, sehingga tidak memerlukan protokol tambahan. Ini membuatnya ringan dan cepat untuk berkomunikasi antar sistem.

III. LATIHAN

Bagian 1: Membuat Proyek Spring Boot

Langkah 1.1: 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).
4. Klik **Generate** untuk mengunduh proyek.
5. Ekstrak proyek yang diunduh, dan buka proyek di **IDE** (**NetBeans**).

Langkah 1.2: 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.

Bagian 2: Mengembangkan REST API Sederhana

REST API ini akan menangani entitas sederhana, misalnya **Product**, yang memiliki atribut seperti **id**, **name**, dan **price**.

Langkah 2.1: Membuat Model untuk Entitas Product

Buat kelas model yang mewakili entitas **Product**.

Langkah-langkah:

1. Buat package **`model`** di dalam `com.example.restapidemo`.
2. Buat kelas **`Product.java`**:

```
package com.example.restapidemo.model;

public class Product {
    private Long id;
    private String name;
    private double price;

    public Product() {
```

```
    }

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

    public double getPrice() {
        return price;
    }

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

Langkah 2.2: Membuat Repository Sederhana

Buat repository sederhana untuk menyimpan dan mengelola data **Product** dalam memori (untuk kesederhanaan, kita tidak akan menggunakan database).

Langkah-langkah:

1. Buat package **`repository`** di dalam `com.example.restapidemo`.
2. Buat kelas **`ProductRepository.java`**:

```
package com.example.restapidemo.repository;

import com.example.restapidemo.model.Product;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Optional;

@Repository
public class ProductRepository {

    private List<Product> products = new ArrayList<>();
    private long idCounter = 0;

    public List<Product> getAllProducts() {
        return products;
    }

    public Optional<Product> getProductById(Long id) {
        return         products.stream().filter(p         ->
p.getId().equals(id)).findFirst();
    }

    public Product addProduct(Product product) {
        product.setId(++idCounter);
        products.add(product);
        return product;
    }

    public    Product    updateProduct(Long    id,    Product
updatedProduct) {
        Optional<Product>         existingProduct         =
getProductById(id);
        if (existingProduct.isPresent()) {
            Product product = existingProduct.get();
            product.setName(updatedProduct.getName());
            product.setPrice(updatedProduct.getPrice());
            return product;
        } else {
            return null;
        }
    }

    public boolean deleteProduct(Long id) {
        return products.removeIf(p -> p.getId().equals(id));
    }
}
```

Langkah 2.3. Membuat Service untuk Logika Bisnis

Buat service yang akan menangani logika bisnis untuk **Product**.

Langkah-langkah:

1. Buat package **`service`** di dalam `com.example.restapidemo`.
2. Buat kelas **`ProductService.java`**:

```
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.getAllProducts();
    }

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

    public Product addProduct(Product product) {
        return
productRepository.addProduct(product);
    }

    public  Product  updateProduct(Long  id,  Product
updatedProduct) {
        return   productRepository.updateProduct(id,
updatedProduct);
    }

    public boolean deleteProduct(Long id) {
        return productRepository.deleteProduct(id);
    }
}
```

Langkah 2.4. Membuat Controller untuk Expose REST API
Buat controller untuk menangani request HTTP dan mengembalikan response.

Langkah-langkah:

1. Buat package **`controller`** di dalam `com.example.restapidemo`.
2. Buat kelas **`ProductController.java`**:

```
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.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product>
getProductById(@PathVariable Long id) {
        return productService.getProductById(id)
                .map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Product>
addProduct(@RequestBody Product product) {
        Product newProduct =
productService.addProduct(product);
        return ResponseEntity.ok(newProduct);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product>
updateProduct(@PathVariable Long id, @RequestBody Product
product) {
```

```
        Product updatedProduct =
productService.updateProduct(id, product);
        if (updatedProduct != null) {
            return ResponseEntity.ok(updatedProduct);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String>
deleteProduct(@PathVariable Long id) {
        boolean isDeleted =
productService.deleteProduct(id);
        if (isDeleted) {
            return ResponseEntity.ok("Product deleted
successfully");
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Bagian 3: Menjalankan dan Menguji Aplikasi

3.1. Menjalankan Aplikasi Spring Boot

Jalankan aplikasi dengan cara:

- **Melalui IDE**: Klik kanan pada file utama **`RestApiDemoApplication.java`** dan pilih **Run**.
- Lalu tes program melalui http://localhost:8080

Bagian 4: Membuat File HTML dan JavaScript untuk Antarmuka Frontend

Kita akan membuat antarmuka aplikasi frontend berbasis web untuk aplikasi **Spring Boot REST API** yang mengelola produk. Kita bisa membuat aplikasi **frontend** berbasis **HTML**, **CSS**, dan **JavaScript** yang akan berinteraksi dengan REST API backend.
Berikut adalah langkah-langkah untuk membuat antarmuka frontend sederhana dengan HTML **dan** JavaScript **untuk mengelola produk melalui REST API yang sudah Anda buat.**

Langkah 1: Membuat File HTML dan JavaScript untuk Antarmuka Frontend

1. Buat folder baru di dalam proyek Anda bernama **`static`** di dalam folder **`src/main/resources`**. Folder ini digunakan oleh Spring Boot untuk melayani file statis seperti HTML, CSS, dan JavaScript.
2. Di dalam folder **`static`**, buat file bernama **`index.html`** untuk halaman web.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Product Management</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        table {
            width: 100%;
            border-collapse: collapse;
        }
        table, th, td {
            border: 1px solid black;
        }
        th, td {
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
        form {
            margin-bottom: 20px;
        }
    </style>
</head>
<body>

<h1>Product Management</h1>

<!-- Form to Add Product -->
<h2>Add Product</h2>
<form id="addProductForm">
    <label for="productName">Name:</label>
    <input type="text" id="productName" name="name"
required>
    <label for="productPrice">Price:</label>
    <input type="number" id="productPrice" name="price"
required>
    <button type="submit">Add Product</button>
</form>

<!-- Table to Display Products -->
<h2>Product List</h2>
<table id="productTable">
    <thead>
```

```
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Price</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        <!-- Rows will be added dynamically -->
    </tbody>
</table>

<!-- JavaScript to handle API calls and dynamic content -
->
<script>
    const apiUrl = 'http://localhost:8080/products';

    // Function to fetch and display all products
    function loadProducts() {
        fetch(apiUrl)
            .then(response => response.json())
            .then(products => {
                const productTableBody =
document.getElementById('productTable').querySelector('tb
ody');
                productTableBody.innerHTML = '';  //
Clear existing rows
                products.forEach(product => {
                    const row =
document.createElement('tr');
                    row.innerHTML = `
                        <td>${product.id}</td>
                        <td>${product.name}</td>
                        <td>${product.price}</td>
                        <td>
                            <button
onclick="deleteProduct(${product.id})">Delete</button>
                            <button
onclick="updateProduct(${product.id}, '${product.name}',
${product.price})">Update</button>
                        </td>
                    `;
                    productTableBody.appendChild(row);
                });
            })
            .catch(error => console.error('Error fetching
products:', error));
```

```
    }

    // Function to add a new product

document.getElementById('addProductForm').addEventListene
r('submit', function (e) {
        e.preventDefault();  // Prevent form from
submitting the traditional way
        const productName =
document.getElementById('productName').value;
        const productPrice =
document.getElementById('productPrice').value;

        const product = {
            name: productName,
            price: parseFloat(productPrice)
        };

        fetch(apiUrl, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(product)
        })
        .then(response => response.json())
        .then(newProduct => {
            console.log('Product added:', newProduct);
            loadProducts();  // Reload the product list
        })
        .catch(error => console.error('Error adding
product:', error));

        // Clear form fields
        document.getElementById('productName').value =
'';
        document.getElementById('productPrice').value =
'';
    });

    // Function to delete a product by ID
    function deleteProduct(id) {
        fetch(`${apiUrl}/${id}`, {
            method: 'DELETE'
        })
        .then(() => {
            console.log('Product deleted:', id);
```

```
            loadProducts();  // Reload the product list
        })
        .catch(error => console.error('Error deleting
product:', error));
    }

    // Function to update a product
    function updateProduct(id, name, price) {
        const updatedName = prompt('Enter new name:',
name);
        const updatedPrice = prompt('Enter new price:',
price);

        if (updatedName !== null && updatedPrice !==
null) {
            const updatedProduct = {
                name: updatedName,
                price: parseFloat(updatedPrice)
            };

            fetch(`${apiUrl}/${id}`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(updatedProduct)
            })
            .then(response => response.json())
            .then(updatedProduct => {
                console.log('Product updated:',
updatedProduct);
                loadProducts();  // Reload the product
list
            })
            .catch(error => console.error('Error updating
product:', error));
        }
    }

    // Load products on page load
    loadProducts();
</script>

</body>
</html>
```

Langkah 2: Menjalankan Aplikasi

1. Stop aplikasi sebelumnya, kemudian jalankan aplikasi Spring Boot kembali.
2. Setelah aplikasi berjalan, buka web browser dan akses **http://localhost:8080/index.html**.
3. Pada halaman tersebut, Anda bisa melakukan operasi berikut:
    - Menambahkan produk baru menggunakan form di bagian atas.
    - Melihat daftar produk yang ada di tabel.
    - Menghapus produk menggunakan tombol **Delete**.
    - Memperbarui produk menggunakan tombol **Update**.

V. LAPORAN

1. Penjelasan kode program, capture GUI untuk setiap soal (9 x (@10%)

Referensi Utama:

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