LAPORAN HASIL PRAKTIKUM 5

PEMROGRAMAN BERORIENTASI OBJEK LANJUT

“Spring Boot-1”

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. 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).

B. PRAKTIKUM

1. Capture code class Product Controller

```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( body: "Product deleted successfully");
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Penjelasan :

@RestController berfungsi untuk menangani permintaan HTTP. @RequestMapping("/products") menandakan bahwa setiap request yang berhubungan dengan product akan menggunakan ("/products") . Pada class ini terdapat deklarasi variabel bernama productRepository bertipe ProductRepository yang bersifat private. @GetMapping berfungsi untuk mendapatkan product dan method getAllProducts() mengembalikan list product dengan memanggil method getAllProducts() pada productService . @GetMapping("/{id}") berfungsi untuk mendapatkan product berdasarkan id dan method getProductById() mengembalikan product yang ditemukan dan status “ok”. Jika tidak, maka akan mengembalikan status not found. @PostMapping berfungsi untuk menambahkan product baru dan method addProduct() akan mengambil objek dari @RequstBody. Kemudian, mengembalikan produk baru yang sudah ditambahkan dan status “ok”. @PutMapping("/{id}") berfungsi untuk memperbarui

product berdasarkan id dan method updateProduct() akan mengambil id dan objek dari @RequstBody. Jika product yang berhasil diupdate, maka akan mengembalikan product yang telah diperbarui dan status “ok”. Jika tidak, maka akan mengembalikan status “not found”. @DeleteMapping("/{id}") berfungsi untuk menghapus product berdasarkan id dan method deleteProduct() akan mendeklarasikan variabel bernama isDeleted yang bertipe boolean dan menginisialisasinya dengan memanggil method deletProduct(id) pada productService. Jika product terhapus, maka akan menampilkan status “ok”. Jika tidak, maka akan menampilkan status “not found”.

2. Capture code class Product

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

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

    no usages
    public Product(){

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

    2 usages
    public Long getId() { return id; }

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

    1 usage
    public String getName() { return name; }

    1 usage
    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; }
}
```

Penjelasan :

Pada class ini terdapat penggunaan method setter getter yang berfungsi untuk mengeset dan mengambil data dari id, name, dan price yang bersifat private. Terdapat constructor dengan isian parameter id bertipe Long, name bertipe String, dan price bertipe double.

3. Capture code class Product Repository

```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 {
    4 usages
    private List<Product> products = new ArrayList<>();
    1 usage
    private long idCounter = 0;

    1 usage
    public List<Product> getAllProducts() { return products; }
    2 usages
    public Optional<Product> getProductById(Long id) {
        return products.stream().filter(p -> p.getId().equals(id)).findFirst();
    }
    1 usage
    public Product addProduct(Product product) {
        product.setId(++idCounter);
        products.add(product);
        return product;
    }
    1 usage
    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;
        }
    }
    1 usage
    public boolean deleteProduct(Long id) { return products.removeIf(p -> p.getId().equals(id)); }
}
```

Penjelasan :

Mendeklarasikan variabel bernama products bertipe List<product> dan menginisialisasi idCounter dengan 0 yang bertipe long. Method getAllProducts() akan mengembalikan data “products”. Method getProductById() akan mengembalikan objek Optional<product> jika id yang dicari ditemukan. Jika tidak, akan mengembalikan Optional.empty(). Method addProduct() akan mengeset id baru dengan idCounter untuk product, memanggil method add(product) pada products, dan mengembalikannya. Method updateProduct() akan memperbarui product berdasarkan id yang dengan memanggil method getProductById(). Lalu, data nama dan harga akan di update. Jika tidak ditemukan, maka akan mengembalikan null. Method deleteProduct() akan mengembalikan data dari products jika id yang dicari sama dengan id yang ingin dihapus.

4. Capture code class Product Service

```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;

2 usages
@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;
    1 usage
    public List<Product> getAllProducts() {
        return productRepository.getAllProducts();
    }
    1 usage
    public Optional<Product> getProductById(Long id) {
        return productRepository.getProductById(id);
    }
    1 usage
    public Product addProduct(Product product) {
        return productRepository.addProduct(product);
    }
    1 usage
    public Product updateProduct(Long id, Product updatedProduct) {
        return productRepository.updateProduct(id, updatedProduct);
    }
    1 usage
    public boolean deleteProduct(Long id) {
        return productRepository.deleteProduct(id);
    }
}
```

Penjelasan :

Pada class ini terdapat deklarasi variabel bernama productRepository bertipe ProductRepository yang bersifat private. Terdapat method getAllProducts() yang berfungsi untuk mendapatkan semua produk yang ada dengan mengembalikan list product. Method getProductById() berfungsi untuk mencari product dengan id tertentu dan mengembalikan optional yang berisi product jika ditemukan. Method addProduct berfungsi untuk menambahkan product baru dan mengembalikan produk baru yang sudah ditambahkan. Method updateProduct() berfungsi untuk mengupdate product yang ada berdasarkan id nya dan mengembalikan product yang telah diupdate . Method deleteProduct() berfungsi untuk menghapus product berdasarkan id nya.

5. Capture Application.properties

```
spring.application.name=REST API Demo
server.port=8081

```

Penjelasan :

Menentukan server port yang akan digunakan, yaitu 8081. Jika server port telah digunakan, dapat diganti dengan server port yang belum digunakan selama tidak melewati batasan port nya.

6. Capture index.html

```
<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>
```

```
<!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>
```

```
<!-- JavaScript to handle API calls and dynamic content -->
<script>
    const apiUrl = 'http://localhost:8081/products'; // url yang akan digunakan
    // Function to fetch and display all products
    1+ usages
    function loadProducts() {
        fetch(apiUrl) // mengambil data dari apiUrl
            .then(response => response.json()) // mengubah format respons menjadi bentuk json
            .then(products => { // memproses data produk yang ada
                const productTableBody = document.getElementById('productTable').querySelector('tbody');
                productTableBody.innerHTML = ''; // Clear existing rows
                products.forEach(product => { // menambahkan setiap product ke dalam tabel
                    const row = document.createElement('tr'); // membuat baris baru
                    row.innerHTML = `
                                <td>${product.id}</td>//membuat kolom id, nama, price
                                <td>${product.name}</td>
                                <td>${product.price}</td>
                                <td>
<button onclick="deleteProduct(${product.id})">Delete</button> // membuat tombol delete
<button onclick="updateProduct(${product.id}, '${product.name}', ${product.price})">Update</button> // membuat tombol untuk mengupdate product baru
</td>
`;
                    productTableBody.appendChild(row); // menambahkan baris ke tabel
                });
            })
            // menangani kesalahan saat pengambilan data
            .catch(error => console.error('Error fetching products:', error));
    }
    // Function to add a new product
    document.getElementById('addProductForm').addEventListener('submit', function (e) {
        e.preventDefault(); // Prevent form from submitting the traditional way
        const productName = document.getElementById('productName').value; // mendapatkan nama product
        const productPrice = document.getElementById('productPrice').value; // mendapatkan harga product
        const product = {
            name: productName, // data productName disimpan ke objek name
            price: parseFloat(productPrice) // data productPrice disimpan ke objek price dalam bentuk float
        };
        fetch(apiUrl, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(product)
        })
            .then(response => response.json()) // mengubah format respons menjadi bentuk json
            .then(newProduct => {
                console.log('Product added:', newProduct); // menampilkan produk yang telah ditambahkan
                loadProducts(); // Reload the product list
            })
            // menangani kesalahan saat menambahkan data
            .catch(error => console.error('Error adding product:', error));
// Clear form fields
        document.getElementById('productName').value = '';
        document.getElementById('productPrice').value = '';
    });
    // Function to delete a product by ID
    1+ usages
    function deleteProduct(id) {
        fetch(`${apiUrl}/${id}`, {
            method: 'DELETE' // menghapus data
        })
            .then(() => {
                console.log('Product deleted:', id); //menampilkan produk yang telah dihapus
                loadProducts(); // Reload the product list
            })
            // menangani kesalahan saat penghapusan data
            .catch(error => console.error('Error deleting product:', error));
    }
    // Function to update a product
    1+ usages
    function updateProduct(id, name, price) {
        const updatedName = prompt('Enter new name:', name); // menyimpan inputan nama ke dalam updatedName
        const updatedPrice = prompt('Enter new price:', price); // menyimpan inputan price ke dalam updatedPrice
        if (updatedName !== null && updatedPrice !== null) {
            const updatedProduct = {
                name: updatedName, //menyimpan updatedName ke objek name
                price: parseFloat(updatedPrice) // mengubah updatedPrice ke bentuk float dan disimpan ke objek price
            };
            fetch(`${apiUrl}/${id}`, {
                method: 'PUT', // meng-update product
```

```
                headers: {
                    'Content-Type': 'application/json' // mengirim dalam format json
                },
                body: JSON.stringify(updatedProduct) // mengubah updatedProduct ke dalam format json
            })
                .then(response => response.json()) //mengubah format respons menjadi bentuk json
                .then(updatedProduct => {
                    console.log('Product updated:', updatedProduct); //menampilkan produk yang telah diperbarui
                    loadProducts(); // Reload the product list
                })
                // menangani kesalahan saat meng-update data
                .catch(error => console.error('Error updating product:', error));
        }
    }
    // Load products on page load
    loadProducts();
</script>
</body>
</html>
```

Penjelasan :

Menentukan style yang diinginkan dan membuat form untuk menambahkan product. Lalu, membuat tabel untuk menampilkan product yang sudah ditambahkan. Terdapat pengguanan JavaScript untuk meng-handle API dan content nya. Menyimpan url yang akan digunakan pada apiUrl. Kemudian membuat suatu function untuk menampilkan semua product yang ada. Pertama, mengambil data dari apiUrl. Kemudian, mengubah format respons menjadi bentuk json, memproses data produk yang ada, menghapus kolom yang ada dan menambahkan setiap product ke dalam tabel. Lalu, membuat baris baru, membuat tombol delete dan tombol update. menambahkan baris ke tabel dan terdapat catch untuk menangani kesalahan saat pengambilan data. Dilanjut dengan membuat function untuk menambahkan product baru. Pertama, menggunakan preventDefault() untuk mencegah submintting form secara traditional. Kemudian, menyimpan nama product dan harga product yang telah diinput ke dalam productName dan productPrice. Lalu, data dari productName akan disimpan ke name dan productPrice disimpan ke objek price dalam bentuk float. Menambahkan data dengan menggunakan method ‘POST’. Content akan dikirim dalam bentuk json, response akan diubah menjadi format json, dan akan memuat ulang daftar produk. Kemudian, terdapat catch untuk menangani kesalahan saat menambahkan data. Lalu, membuat suatu function untuk menghapus suatu product dengan menggunakan method ‘DELETE’. Kemudian, menampilkan data yang telah dihapus dan memuat ulang daftar produk. Kemudian, terdapat catch untuk menangani kesalahan saat penghapusan. Dilanjut dengan membuat suatu function untuk meng-update suatu product. Menyimpan inputan nama dan harga ke updatedName dan updatedPrice. Jika updatedName dan updatedPrice tidak null, maka data akan disimpan ke name dan price dalam bentuk float. Lalu, menggunakan method ‘PUT’ untuk meng-update product, mengirim dalam format json, dan mengubah updatedProduct ke dalam format json.

Lalu, mengubah format respons menjadi bentuk json, menampilkan product yang telah di update dan memuat ulang daftar produk. Kemudian, terdapat catch untuk menangani kesalahan saat meng-update data.

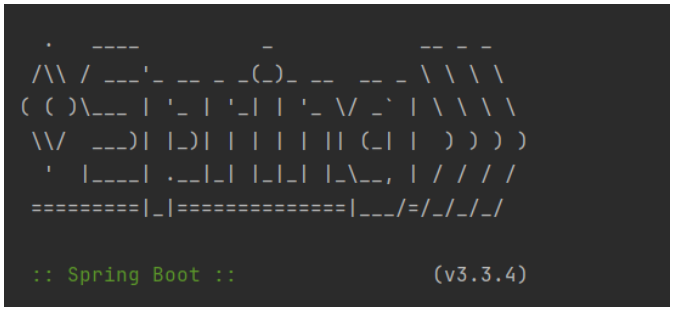
7. Capture RestApiDemoApplication

```java
package com.example.restapidemo;

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

@SpringBootApplication
public class RestApiDemoApplication {

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

}
```

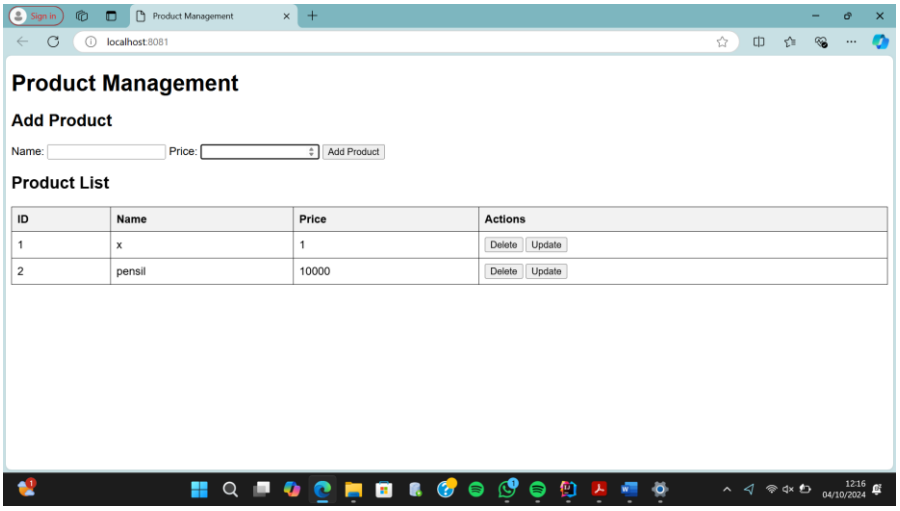
Penjelasan :

Menggunakan Spring Boot pada class RestApiDemoApplication yang ditandai dengan @SpringBootApplication (akan mengkonfigurasi secara otomatis). Lalu, SpringApplication.*run*() berfungsi untuk menginisialisai Spring dan mengelola konfigurasi yang diperlukan.

8. Capture Output




```
0-04T12:02:48.899+07:00  INFO 16924 --- [REST API Demo] [           main] c.e.restapidemo.RestApiDemoApplication   : Starting RestApiDemoApplic
0-04T12:02:48.908+07:00  INFO 16924 --- [REST API Demo] [           main] c.e.restapidemo.RestApiDemoApplication   : No active profile set, fal
0-04T12:02:49.867+07:00  INFO 16924 --- [REST API Demo] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with po
0-04T12:02:49.884+07:00  INFO 16924 --- [REST API Demo] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
0-04T12:02:49.885+07:00  INFO 16924 --- [REST API Demo] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [
0-04T12:02:49.941+07:00  INFO 16924 --- [REST API Demo] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedd
0-04T12:02:49.942+07:00  INFO 16924 --- [REST API Demo] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext
0-04T12:02:50.072+07:00  INFO 16924 --- [REST API Demo] [           main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome page: class
0-04T12:02:50.385+07:00  INFO 16924 --- [REST API Demo] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 808
0-04T12:02:50.403+07:00  INFO 16924 --- [REST API Demo] [           main] c.e.restapidemo.RestApiDemoApplication   : Started RestApiDemoApplica
0-04T12:02:52.845+07:00  INFO 16924 --- [REST API Demo] [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispat
0-04T12:02:52.845+07:00  INFO 16924 --- [REST API Demo] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'disp
0-04T12:02:52.846+07:00  INFO 16924 --- [REST API Demo] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization i
```


Sign in
Product Management
localhost:8081
Product Management
Add Product
Name:
Price:
Add Product
Product List
ID
Name
Price
Actions
1
x
1
Delete
Update
2
pensil
10000
Delete
Update
12:16
04/10/2024