Materi 8 : Model View Controller (MVC) Di Java

DAFTAR ISI

Materi 8

1.1 Konsep MVC (Model-View-Controller)

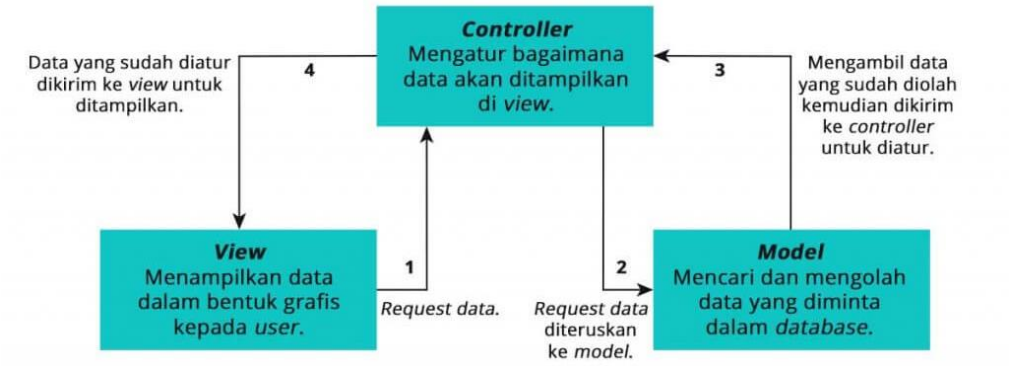
MVC adalah sebuah pola desain (design pattern) yang digunakan dalam pengembangan perangkat lunak untuk memisahkan aplikasi menjadi tiga komponen utama:

- **Model:** Bagian yang bertugas untuk menyiapkan, mengatur, memanipulasi, dan mengorganisasikan data yang ada di database.
- **View:** View bertanggung jawab untuk menampilkan data dari Model kepada pengguna dan menerima input dari pengguna. Ini adalah komponen yang mengatur tampilan grafis atau antarmuka pengguna. Dalam Java, view sering kali berupa kelas-kelas Swing atau JavaFX yang menangani GUI.
- **Controller:** Controller berfungsi sebagai penghubung antara Model dan View

1.2 Alur MVC

Setelah mengetahui penjelasan dan komponen dari MVC, sekarang kita akan membahas alur proses dari MVC. Berikut ini adalah alur prosesnya.

a) Proses pertama adalah view akan meminta data untuk ditampilkan dalam bentuk grafis kepada pengguna.
b) Permintaan tersebut diterima oleh controller dan diteruskan ke model untuk diproses.
c) Model akan mencari dan mengolah data yang diminta di dalam database
d) Setelah data ditemukan dan diolah, model akan mengirimkan data tersebut kepada controller untuk ditampilkan di view.
e) Controller akan mengambil data hasil pengolahan model dan mengaturnya di bagian view untuk ditampilkan kepada pengguna.



1.3 Keuntungan Penggunaan MVC

- **Pemisahan Keperluan**, MVC memungkinkan pemisahan tugas yang jelas antara Model, View, dan Controller. Ini membuat kode lebih mudah dipahami, dikelola, dan diperbaiki.
- **Reusabilitas**, Setiap komponen (Model, View, dan Controller) dapat digunakan kembali dalam berbagai bagian aplikasi atau dalam proyek yang berbeda.

- **Kolaborasi Tim**, Dengan pemisahan yang jelas antara komponen-komponen MVC, tim pengembangan dapat bekerja secara paralel pada berbagai bagian aplikasi.
- **Mudah Diuji**, Setiap komponen dapat diuji secara terpisah, memudahkan pengujian dan pemecahan masalah.

1.4 Implementasi MVC dalam Java Spring Boot

Dalam Java Spring Boot, Kita dapat menerapkan konsep MVC menggunakan Spring MVC. Spring MVC adalah bagian dari framework Spring yang dirancang untuk memfasilitasi pengembangan aplikasi web berbasis Model-View-Controller. Berikut adalah langkah-langkah dasar untuk mengimplementasikan MVC dalam Spring Boot:

a) Model

Model dalam Spring Boot biasanya berupa kelas Java yang merepresentasikan data dan logika bisnis aplikasi. Kita dapat menggunakan anotasi seperti `@Entity` (untuk Hibernate) atau hanya kelas POJO (Plain Old Java Object) biasa untuk mendefinisikan Model Kita.

b) View

View dalam Spring Boot dapat berupa berbagai hal, mulai dari halaman HTML, Thymeleaf templates, hingga file JSON atau XML. View digunakan untuk menampilkan data dari Model kepada pengguna.

c) Controller

Controller adalah kelas yang menangani permintaan/request HTTP dari pengguna. Kita dapat menggunakan anotasi `@Controller` atau `@RestController` untuk mendefinisikan Controller. Controller berisi metode-metode yang menangani permintaan/request dan memberikan respon/response.

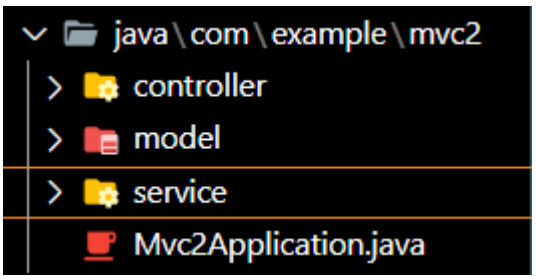
d) Routing

Kita harus menentukan rute (route) atau URL yang akan dikendalikan oleh masing-masing Controller. Kita dapat menggunakan anotasi `@RequestMapping` atau `@GetMapping`, `@PostMapping`, dll.

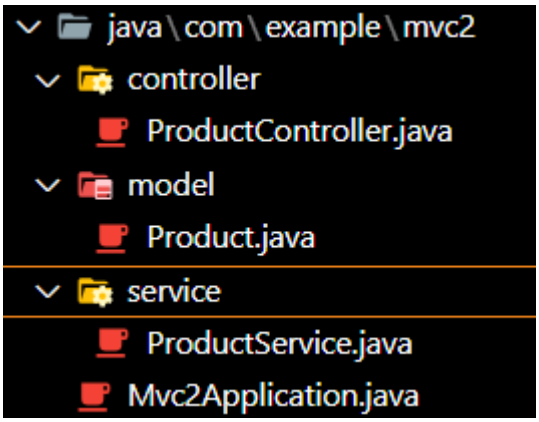
1.5 Contoh Implementasi

a) Buat Project Maven Spring Boot seperti yang sudah dilakukan di materi ke 7

b) Tambahkan folder/package controller, model, dan service**,** Berikut adalah contohnya :



c) Buat file **`ProductController.java`** di dalam package **`controller`**, **`Product.java`** di package **`model`**, dan **`ProductService.java`** di package **`service`**. Berikut adalah contohnya :



d) Selanjutnya, kita akan fokus ke bagian model terlebih dahulu. Buka file **`Product.java`** lalu isi dengan kode berikut :

```
package com.example.mvc2.model;

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

    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;
    }
}
```

Class **Product** ini adalah representasi dasar dari produk dalam aplikasi Kita. Kita dapat membuat objek **Product**, mengatur atau mendapatkan nilai atributnya, dan menggunakannya dalam aplikasi Kita, seperti menampilkan daftar produk atau melakukan operasi CRUD pada produk.

e) Selanjutnya adalah bagian service, di dalam ProductService.java tambahkan kode berikut :

```
package com.example.mvc2.service;

import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Service;
import com.example.mvc2.model.Product;

@Service
public class ProductService {
    public List<Product> getAllProducts() {
        List<Product> products = new ArrayList<>();
        products.add(new Product(1L, "Produk A", 10.0));
        products.add(new Product(2L, "Produk B", 20.0));
        products.add(new Product(3L, "Produk C", 30.0));
        return products;
    }
}
```

Kode ProductService.java yang Kita berikan adalah kelas service yang menyediakan layanan terkait dengan entitas produk dalam aplikasi Kita. Mari kita bahas dengan lebih detail:

- `@Service` Dengan anotasi ini, Spring akan mendeteksi dan mengelola kelas `ProductService` sebagai komponen `service`, dan Kita dapat menginjeksinya ke dalam komponen lain seperti kelas controller.
- `return products;` Metode `getAllProducts()` mengembalikan List yang berisi semua produk yang telah ditambahkan. Ini adalah daftar produk yang akan digunakan oleh controller untuk menampilkan daftar produk dalam tampilan.

`ProductService` berfungsi untuk menyediakan daftar produk secara statis. Dalam pengembangan yang lebih lanjut, Kita bisa menggantikan implementasi ini dengan memuat produk dari database atau sumber data lain sesuai kebutuhan Kita.

f) Selanjutnya adalah bagian controller, di dalam ProductController.java tambahkan kode berikut :

```
package com.example.mvc2.controller;

import java.util.List;
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 com.example.mvc2.model.Product;
import com.example.mvc2.service.ProductService;

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

    @GetMapping("/products")
    public String listProducts(Model model) {
        List<Product> products = productService.getAllProducts();
        model.addAttribute("products", products);
        return "productList";
    }
}
```

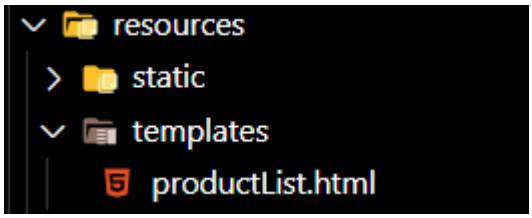
Kode `ProductController.java` adalah kelas controller yang menangani permintaan/request HTTP terkait produk dan mengkoordinasikan antara `model` (entitas produk) dan tampilan (`view`). Mari kita bahas dengan lebih detail:

- `@Controller` adalah anotasi Spring yang digunakan untuk menandai kelas `ProductController` sebagai komponen controller.

- `@Autowired` adalah anotasi Spring yang digunakan untuk melakukan injeksi dependensi. Dalam kasus ini, kita menggunakannya untuk menginjeksikan kelas `ProductService` ke dalam kelas `ProductController`, sehingga Anda dapat mengakses layanan yang disediakan oleh `ProductService`.
- `@GetMapping("/products")` adalah anotasi `@GetMapping` yang menentukan bahwa method `listProducts` akan menangani permintaan/request HTTP `GET` pada URL `/products`. Ketika URL `/products` diakses melalui peramban web, metode `listProducts` akan dipanggil.
- `return "productList";` Metode `listProducts` mengembalikan string "`productList`". Ini adalah nama tampilan (view) yang akan digunakan untuk menampilkan produk. Nama ini harus sesuai dengan nama file template Thymeleaf atau JSP yang akan digunakan sebagai tampilan produk.

Kode ini menggambarkan bagaimana kelas `controller` bekerja dalam aplikasi Spring MVC, mengelola permintaan HTTP, berinteraksi dengan layanan (`ProductService`), dan mengirimkan data ke tampilan untuk ditampilkan kepada pengguna.

g) Sekarang tambahkan tampilan sederhana html dengan menggunakan bantuan dependensi thymeleaf. Buat file `productList.html` di dalam resources/templates



Lalu tambahkan kode html berikut :

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

<body>
   <h1>Daftar Produk</h1>
   <table>
       <tr>
           <th>ID</th>
           <th>Nama</th>
           <th>Harga</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>
```

Jangan lupa menambahkan dependensi thymeleaf di **`pom.xml`** :

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

Setelah itu coba build ulang semua dependensi sampai selesai, kemudian project tersebut di run. Terakhir, buka hasil project melalui browser dengan URL http://localhost:8080/products maka akan menampilkan output berikut :

Note : Untuk Thymeleaf akan dipelajari di pertemuan mendatang setelah materi JPA.