Anggota Kelompok 3 : Studi Kasus Aplikasi Perpustakaan

- Aditya Widiyanto Nugroho (222111845)
- Azmi Zulfani Putri (222111940)
- Fathimah Az-Zahra (222112047)
- Himawan Wahid Ikhwansyah (222112094)
- R.Faras Roihan Armel (222112296)
- Zidan Al Azizi (222112433)
- Nabila Widya Putri (222112236)
- Ni Putu Sancita M. A. (222112258)

A. Design Pattern

1. Builder

Builder design pattern adalah sebuah pola desain perangkat lunak yang digunakan untuk membuat objek yang kompleks secara bertahap. Pola ini memisahkan proses pembuatan objek dari representasi internalnya, sehingga memungkinkan kita untuk membuat berbagai konfigurasi objek dengan menggunakan langkah-langkah yang terpisah.

Implementasi:

Pada kelas mapper digunakan untuk melakukan pembuatan object kompleks (entity dan dto) secara bertahap.

2. Singleton

Singleton design pattern adalah sebuah desain yang digunakan untuk memastikan bahwa hanya satu instance dari sebuah kelas yang dapat dibuat dan instance tersebut dapat diakses secara global. Tujuan utama Singleton adalah untuk membatasi pembuatan objek menjadi satu instance tunggal untuk menghindari penggunaan sumber daya yang berlebihan.

Implementasi:

Pada kelas yang menggunakan anotasi @Service, @Repository, atau @Controller yang selanjutnya dapat di inject ke kelas lain menggunakan anotasi @Autowired. Spring akan memberikan instance beans yang sama di mana pun diperlukan.

3. MVC

Spring Boot mengikuti pola arsitektur MVC untuk membangun aplikasi web. Model mewakili data dan logika bisnis aplikasi, View menangani presentasi dan antarmuka pengguna, dan Controller mengatur alur permintaan dan respons. Framework MVC Spring Boot menyederhanakan implementasi pola ini melalui anotasi dan konvensi.

- **Model**

  Model mewakili data dan logika bisnis dari aplikasi. Ini bisa berupa kelas Java yang menggambarkan entitas atau objek bisnis yang terkait dengan aplikasi Anda. Model

bertanggung jawab untuk memanipulasi dan mengelola data yang diperlukan oleh aplikasi. Dalam Spring Boot, model sering diimplementasikan sebagai kelas POJO (Plain Old Java Object) dengan anotasi JPA (Java Persistence API) jika Anda menggunakan database.

- **View**
  View adalah tampilan yang menangani presentasi dan antarmuka pengguna. Ini adalah komponen yang menghasilkan output yang akan ditampilkan kepada pengguna. Dalam Spring Boot, tampilan dapat berupa template seperti Thymeleaf atau FreeMarker, yang memungkinkan penulisan tampilan menggunakan bahasa template dan menggabungkannya dengan data yang dikirimkan oleh kontroler.

- **Controller**
  Controller bertindak sebagai penghubung antara model dan view. Itu menerima permintaan dari pengguna, memprosesnya, dan memutuskan tindakan yang diperlukan. Kontroler berisi metode yang ditandai dengan anotasi khusus seperti @RequestMapping atau @GetMapping untuk menangani URL tertentu. Metode ini berisi logika bisnis dan memanipulasi model sesuai dengan permintaan.

Implementasi:

Model dapat dilihat pada package model pada project kami, kami membuat model Book, Keuangan, Peminjaman, Pengembalian, Rating, Role, User. Pada controller juga dapat dilihat pada package controller yaitu BookController, KeuanganController, LoginController, PeminjamanController, PengembalianController, RatingController, StaffController, UserController. View dapat dilihat pada penggunaan HTML, CSS, dan Js.

4. Repository Pattern

Pola Repository sering digunakan dalam Spring Boot untuk menangani akses data dan persistensi. Pola ini menyediakan cara untuk mengkapsulasi logika akses data dan berinteraksi dengan database atau sumber data lainnya. Spring Data JPA, misalnya, menawarkan abstraksi dan implementasi yang nyaman untuk mengimplementasikan pola Repository dan bekerja dengan database relasional.

Implementasi:

Pada kelas yang menggunakan anotasi @Repository

5. Dependency Injection (DI) Pattern

Spring Boot sangat mengandalkan pola Injeksi Dependensi, terutama melalui kontainer Inversi Kontrol (IoC). Injeksi Dependensi memungkinkan keterikatan longgar antara komponen dengan menginjeksikan dependensi daripada mengandalkan pembuatan instance atau pencarian secara eksplisit. Kemampuan Injeksi Dependensi Spring Boot memudahkan pengelolaan dan *wired* dependensi di seluruh aplikasi.

Implementasi:

Pada kelas yang menggunakan anotasi @Autowired

6. **Data Access Object (DAO) Pattern**

Saat menggunakan DTO untuk memindahkan data antara lapisan data dan lapisan bisnis, pola DAO dapat digunakan untuk menyediakan akses ke sumber data dan mengubah hasilnya menjadi DTO. DAO bertanggung jawab untuk mengambil data dari sumber dan mengkonversinya ke dalam DTO sebelum dikirim ke lapisan bisnis.

Implementasi

Pada kasus project kami, kami menggunakan BookDto, KeuanganDto, PeminjamanDto, PengembalianDto, RatingDto, serta UserDto

B. Design Principle

1. **Encapsulation** -> Penggunaan User-Role(Membagi tampilan dan pembatasan akses antara admin serta pengguna perpustakaan)

2. **Single Responsibility Principle**

Prinsip Single Responsibility (SRP) adalah salah satu prinsip desain yang diterapkan dalam pengembangan perangkat lunak, termasuk dalam proyek Spring Web. Prinsip ini menyatakan bahwa setiap kelas atau komponen harus memiliki satu tanggung jawab tunggal dan hanya satu alasan untuk berubah.

- **Controller**

  Kelas controller dalam Spring Web bertanggung jawab untuk menangani permintaan HTTP dan berinteraksi dengan model dan tampilan yang sesuai. Prinsip SRP menyarankan agar kelas controller hanya menangani logika routing dan pengolahan permintaan, dan tidak melakukan tugas lain yang tidak relevan, seperti logika bisnis atau pemrosesan data.

- **Service**

  Komponen service dalam Spring Web sering digunakan untuk mengisolasi logika bisnis dari kelas controller. Prinsip SRP menyarankan agar komponen service fokus pada tugas spesifik terkait logika bisnis, seperti pemrosesan data, validasi, pemanggilan ke lapisan data, atau interaksi dengan sistem eksternal. Ini memastikan bahwa controller tetap fokus pada tugas koordinasi dan routing.

- **Repository**

  Kelas repository digunakan dalam Spring Web untuk mengelola akses dan persistensi data ke database atau sumber data lainnya. Prinsip SRP menyarankan agar repository hanya bertanggung jawab untuk operasi yang terkait dengan akses data, seperti pengambilan, penyimpanan, dan manipulasi entitas. Repository tidak seharusnya memiliki tanggung jawab lain, seperti logika bisnis atau validasi data.

- **Data Transfer Object (Dto)**
  DTO dapat membantu menerapkan prinsip SRP dengan memisahkan tanggung jawab transfer data dari logika bisnis. DTO bertugas secara khusus untuk mengangkut data antara lapisan, entitas, atau komponen tanpa adanya logika bisnis kompleks.

3. **Interface Segregation Principle**
   Interface Segregation Principle (Prinsip Segregasi Antarmuka) adalah prinsip dalam pemrograman yang mengatakan bahwa klien tidak boleh dipaksa untuk bergantung pada antarmuka yang tidak mereka gunakan

   Implementasi:
   Memecah interface menjadi beberapa interface yang lebih spesifik dan melakukan inject seperlunya. Contohnya pada interface serviceImpl yang hanya bergantung pada serviceyang dibutuhkan .
4. **Liskov Substitution Principle**
   Liskov Substitution Principle (Prinsip Substitusi Liskov) adalah prinsip dalam pemrograman berorientasi objek yang menyatakan bahwa objek dari kelas turunan harus dapat digunakan sebagai pengganti objek dari kelas induk tanpa mempengaruhi kebenaran program

   Implementasi:
   Belum ada tetapi kode akan tetap bisa berjalan walaupun objek diganti turunan dari user atau turunan dari buku
5. **Dependency Inversion Principle**
   Dependency Inversion Principle (Prinsip Inversi Ketergantungan) adalah prinsip yang menyatakan bahwa modul harus bergantung pada abstraksi daripada implementasi. Dalam implementasi yang disebutkan, prinsip ini mungkin diterapkan dengan menggunakan antarmuka atau kelas abstrak sebagai dependensi (dependency) yang diinjeksikan ke dalam kelas PeminjamanServiceImpl, sehingga kelas tersebut bergantung pada abstraksi daripada implementasi khusus

   Implementasi:
   @Autowired digunakan untuk meng-inject bean secara otomatis untuk memastikan bahwa kelas tidak perlu menciptakan instance dependency secara langsung, tetapi menerimanya dari luar