UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Tutorial 2 Arsitektur & Pemrograman Aplikasi Perusahaan
Semester Ganjil Tahun Ajaran 2021/2022
Deadline: 15 September 2021, 23.55 waktu Scele

Database & Relasi Database dalam Project Spring Boot

v.1.1

PENJELASAN

Halo anak-anak Papa APAP! Setelah belajar mengenai konsep MVC di Spring Boot pada tutorial sebelumnya, di tutorial kali ini kamu akan menggunakan modul JPA untuk mengintegrasikan *database* server kamu. ***Java Persistence API*** (JPA) adalah sebuah *interface* pemrograman aplikasi Java yang menggambarkan pengelolaan data relasional dalam *database* yang memungkinkan manipulasi data tanpa menggunakan *query* (dokumentasinya bisa dilihat disini).

Pada tutorial sebelumnya, kita sudah menggunakan *model* dan *service* pada konsep MVC. Pada tutorial ini, model tersebut (*entity*) akan dijalankan menggunakan *database* server dan kita juga akan menambahkan *entity* baru serta menghubungkannya dengan *entity* sebelumnya. *Library* yang akan kamu pakai pada tutorial kali ini yaitu JPA.

Penggunaan JPA dalam pemrograman java menggunakan pendekatan ***Object Relational Mapping*** (ORM). Adapun dengan menggunakan JPA, memungkinkan kita untuk melakukan manipulasi data tanpa menggunakan *query*, namun bukan berarti tanpa menggunakan *query* sama sekali, tetap ada penggunaan *query* disana. API JPA terdapat dalam *package* javax.persistence. Di dalamnya mengandung *query* khusus yang disebut (JPQL) *Java Persistence Query Language.*

PRASYARAT

- DDP2
    - Interface
    - Overriding
- PPW
    - Request Method
    - MVC Framework
- Basis Data
    - Relationship

LINGKUP PEMBAHASAN

- Model
    - Constructor
    - Setter/getter
- Service
    - Interface
    - Implementasi Interface
- Controller
    - RequestMethod
    - RequestMapping
    - RequestParam
- Database
- JPA Repository

EKSPEKTASI

- Memahami ***Create***, ***Read***, ***Update***, dan ***Delete*** (CRUD) pada basis data dengan menggunakan konsep MVC dalam *project* Spring Boot.
- Membuat model yang terhubung dengan basis data.
- Memahami penggunaan JPARepository untuk melakukan *query* pada basis data.
- Membuat sebuah *service* dengan fungsi *create* dan *read* data menggunakan konsep MVC dalam *project* Spring Boot.

PENGUMPULAN

- Tenggat Waktu Pengumpulan (*Due Date*) :
  Rabu, 22 September, sebelum 23.55 WIB untuk semua kelas. Setiap keterlambatan selama 24 jam akan mengakibatkan -20% dari nilai sebenarnya.
- Pengumpulan tutorial dalam bentuk **Pull Request (PR) dari branch feat/tutorial-3-emsidi ke main**. Waktu keterlambatan dilihat dari **aktivitas perubahan** di Pull Request.
- Jawablah **semua** pertanyaan dan kerjakan **semua** latihan yang ada pada tutorial.
- Jawablah pertanyaan-pertanyaan yang ada pada file **README.md yang sama** seperti tutorial sebelumnya.
- Tutorial 3 di-*push* ke dalam *repository* yang sama dengan tutorial sebelumnya sehingga pada *repository* akan terdapat tiga folder yaitu *IsPalindrome, kebunsafari,* dan *emsidi*.
- Gunakan teknik *branching* dan PR yang sama seperti pada tutorial sebelumnya.
- Tutorial 3 **ada demo**, silahkan isi slot demo di SCELE.

PENILAIAN

Nilai akhir = Nilai Tutorial + Latihan + Pertanyaan + Demo

Plagiarisme akan dikenakan sanksi nilai 0! ☺

- Indikator Penilaian Tutorial
    - Kesesuaian ***Model, Controller, Repository***, dan ***View Templates***.
    - Keberhasilan proyek Emcidi.
    - Kelengkapan dan Ketepatan Jawaban pada README.md
- Demo

PRA-TUTORIAL

1. Pastikan kamu sudah melakukan **merge** dari branch **tutorial sebelumnya** ke **main**
2. Sebelum mulai mengerjakan tutorial ini, buat **issue "Pengerjaan Tutorial 3"** pada Repo GitHub kamu
3. Pastikan pada lokal kamu sudah pindah ke **branch main**, kemudian **pull main**
4. Buat **branch baru** dari **main**, dengan nama **'feat/tutorial-3-emsidi'**

PASCA-TUTORIAL

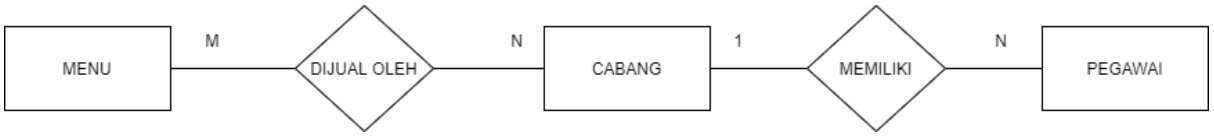
1. Pull Request dengan meminta *review* dari asisten melalui GitHub sebelum ***Deadline***
2. Pastikan pada saat Pull Request, **tambahkan issue** yang telah dibuat pada **Linked Issue** atau dapat mengikuti tutorial pada link berikut
3. Isi slot demo sesuai dengan pembagian asisten minggu ini
4. Asisten akan melakukan *approve* Pull Request setelah demo dilaksanakan

TUTORIAL

Setelah minggu lalu kamu menuruti dan menyetujui permintaan Papa APAP untuk mengembangkan sebuah *enterprise application*, Papa APAP kembali menanyakan terkait progress pengembangan proyek kebunsafari. Kamu menjawab bahwa progressnya hampir selesai. Namun, Papa APAP berkata bahwa tiba-tiba ia merasa lapar dan hal ini menginspirasi Papa APAP untuk mengembangkan *enterprise application* di bidang kuliner, sehingga Papa APAP membatalkan proyek Kebun-Safari yang teman-teman kerjakan.

Sebagai anak yang baik, tentu saja kamu menerima keputusan tersebut dengan lapang dada. Papa APAP tentu saja sangat bangga dengan anaknya yang rajin, baik hati, jujur, dan tidak sombong ini. Papa APAP berpesan, jangan lupa menggunakan ***database*** untuk menyimpan data yang ada agar ketika aplikasi kamu di-*run* ulang tidak ada data-data yang akan hilang seperti proyek yang telah kamu kerjakan pada minggu yang lalu.

Untuk mempermudah anaknya dalam belajar membuat Database, Papa APAP memberikan kepada kamu selembar kertas catatan Papa APAP yang berisi petunjuk ERD untuk hubungan antara Cabang, Pegawai, dan Menu:



Hubungan antara Cabang dan Pegawai adalah **One-to-Many** karena suatu Cabang memiliki beberapa Pegawai, tetapi, seorang Pegawai hanya dapat bekerja pada satu Cabang saja. Selain itu, Cabang juga memiliki relasi **Many-to-Many** dengan Menu. Sebuah Cabang pasti memiliki banyak Menu. Suatu Menu pasti juga terdapat di banyak Cabang. Sebagai contoh, Cabang “PAPA APAP” memiliki Menu seperti Small Mac, McBlurry, Dinner Wrap, dan DiNgin Spesial. Tetapi, Cabang “PAPA APAP” bukan satu - satunya yang memiliki menu tersebut, karena ada Cabang lain yang memiliki menu-menu tersebut juga.

Sebelum menjalankan program kamu, jalankan XAMPP atau database server kamu terlebih dahulu. Buat sebuah database pada MySQL kamu. kamu diminta untuk memahami database tersebut dengan spesifikasi seperti berikut:

CABANG

- **noCabang [Primary Key]**

  Long, Not Null, Auto Increment
- **namaCabang**

  VARCHAR (30), Not Null
- **alamatCabang**

  VARCHAR (40), Not Null
- **waktuBuka**

  TIME, Not Null
- **waktuTutup**

  TIME, Not Null

PEGAWAI

- **noPegawai [Primary Key]**
  Long, Not Null, Auto Increment
- **namaPegawai**
  VARCHAR (30), Not Null
- **jenisKelamin**
  INTEGER, Not Null
- **noCabang [Foreign Key]**

  Long, Not Null, Auto Increment

MENU

- **noMenu [Primary Key]**

  Long, Not Null, Auto Increment
- **namaMenu**

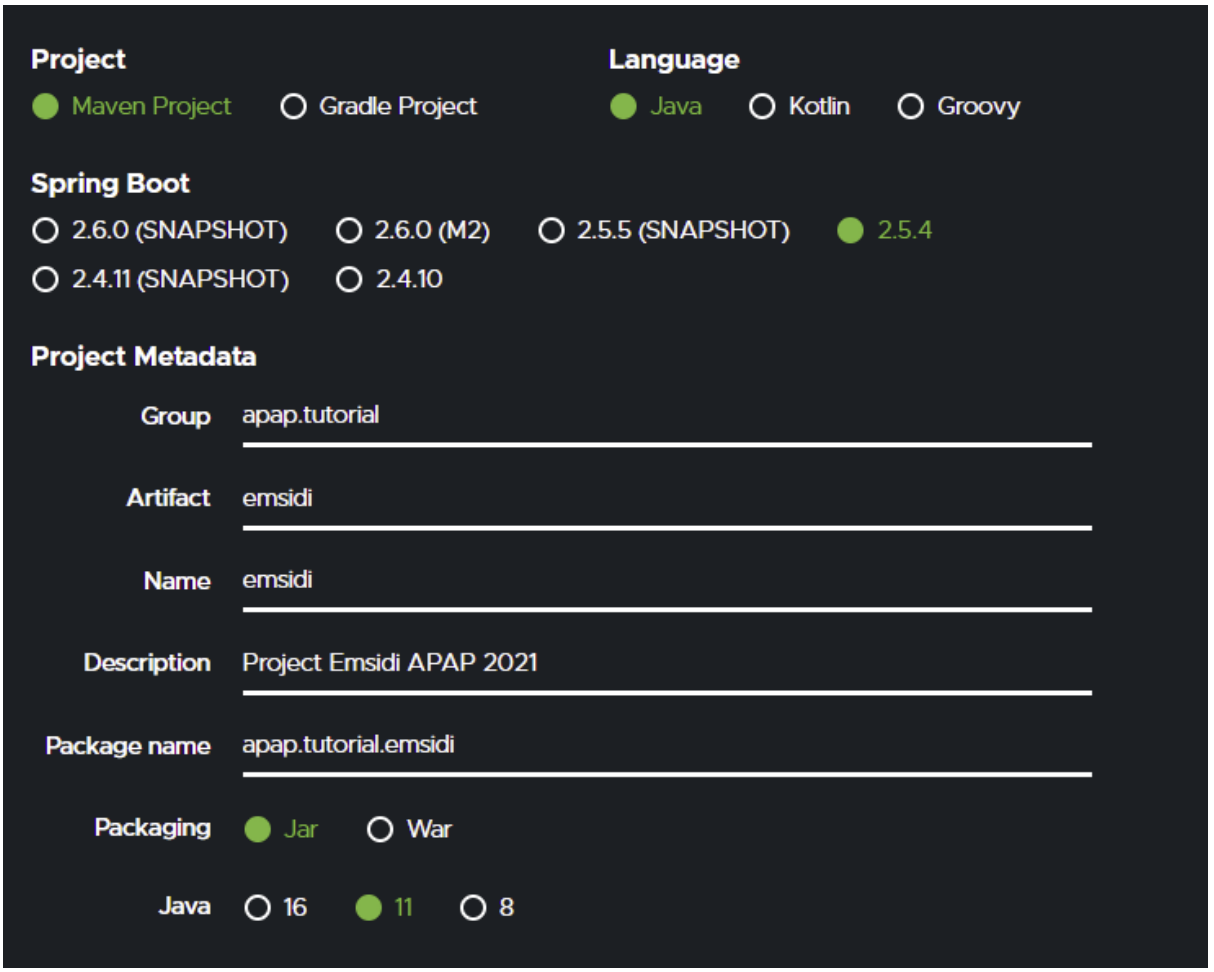
  VARCHAR (30), Not Null
- **isAvailable**

  Boolean, Not Null
- **noCabang [Foreign Key]**

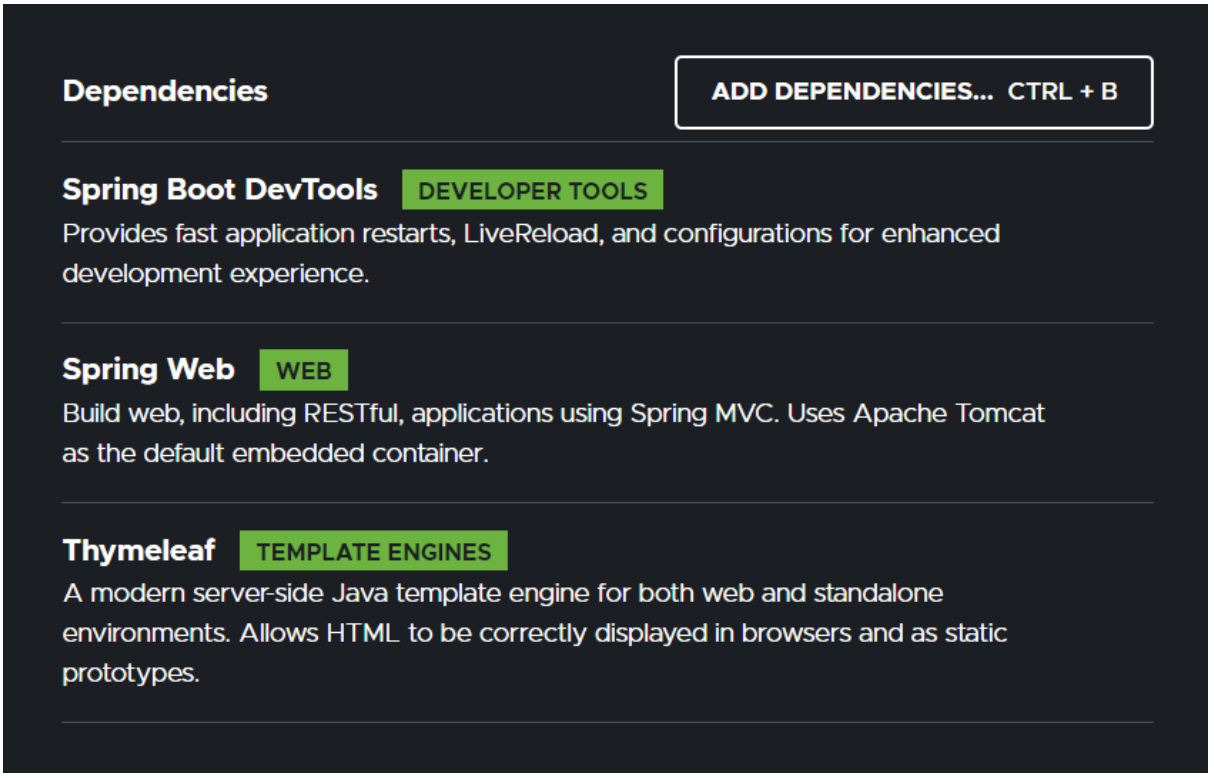
  Long, Not Null, Auto Increment

Inisiasi Proyek

1. Buatlah sebuah Spring project baru melalui https://start.spring.io/
2. Isikan form sesuai gambar berikut



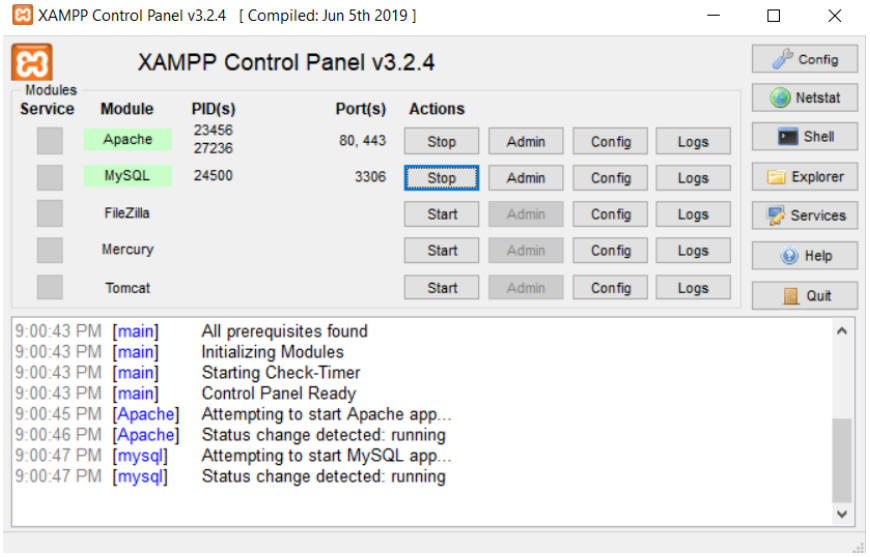
3. Tambahkan *dependencies* berikut



4. Klik pada "**Generate the project**" untuk membuat Spring project.

Tutorial Setup Database

1. Buka aplikasi XAMPP kamu, lalu start Apache dan MySQL Server. Pastikan sudah *running* ya.



2. Buka http://localhost/phpmyadmin (sesuaikan dengan port Apache masing-masing) pada *browser* kamu, lalu buat *database* baru dengan cara klik new lalu beri nama emsidi.

3. Buka file **pom.xml** yang ada di *project* kamu, lalu tambahkan *dependency* berikut:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>

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

4. Buka folder *src/main/resource*, lalu kamu akan menemukan sebuah file bernama **application.properties**. File ini berisi konfigurasi aplikasi kamu. File ini pada awalnya kosong. Namun, kamu dapat menambahkan konfigurasi yang kamu inginkan.

   Konfigurasi yang bisa kamu tambahkan sangat bervariasi. Mulai dari *port*, *database config*, sampai API Key dari *third-party service* yang kamu gunakan. Silakan buka **application.properties** kamu dan tambahkan konfigurasi berikut. Sesuaikan dengan *environment* kamu.

```
#konfigurasi untuk koneksi MySQL
spring.sql.init.platform=mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#sesuaikan NAMA_DATABASE dengan nama database anda
spring.datasource.url=jdbc:mysql://localhost:3306/[NAMA_DATABASE]?useSSL=fa
lse&serverTimezone=Asia/Jakarta

#sesuaikan dengan NAMA dan PASSWORD mysql anda
```

```
spring.datasource.username=[NAMA]
spring.datasource.password=[PASSWORD]

#optimize query untuk db MySQL
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBD
ialect

#pembuatan database (create || create drop || validate || update)
spring.jpa.hibernate.ddl-auto=create
```

Notes:

- Port belum tentu 3306, sesuaikan dengan port MySQL kamu yang berada di XAMPP
- Nama *database* sesuaikan dengan yang telah kamu buat pada step sebelumnya pada http://localhost/phpmyadmin
- Saat pertama kali *run* Spring Boot, ubah konfigurasi **spring.jpa.hibernate.ddl-auto** menjadi ***create.*** Setelah berhasil di-*run*, ubah kembali menjadi ***update***.
- Isi **Username** dan **Password** sesuai dengan yang telah kamu *configure* pada saat instalasi (default username: root, password: [kosongin aja])

Tutorial Membuat Class Model

Selanjutnya, kita akan merubah model yang telah kita buat di tutorial sebelumnya agar dapat *compatible* dengan database kita nantinya. Nanti, apabila mengalami error pada import javax untuk @NotNull dan @Size, tambahkan dependency berikut pada pom.xml:

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

Pada tutorial sebelumnya, kita membuat setter, getter, dan constructor dengan bantuan shortcut dari IDE masing - masing. Kali ini, kita akan menggunakan sebuah dependency untuk meng-handle constructor, setter, dan getter model kita. Tambahkan dependency berikut pada pom.xml:

```
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<version>1.18.20</version>
		<scope>provided</scope>
	</dependency>
```

Dengan menggunakan lombok, kita hanya perlu menambahkan @AllArgsConstructor @NoArgsConstructor @Setter @Getter. Kode kita menjadi lebih ringkas dan rapi. Ada banyak sekali konfigurasi lainnya selain tiga hal tersebut, silakan baca https://projectlombok.org/features/constructor

5. Klik kanan pada **Project > New > Package**. Buatlah *package* **apap.tutorial.emsidi.model**
6. Buatlah **CabangModel** seperti berikut:

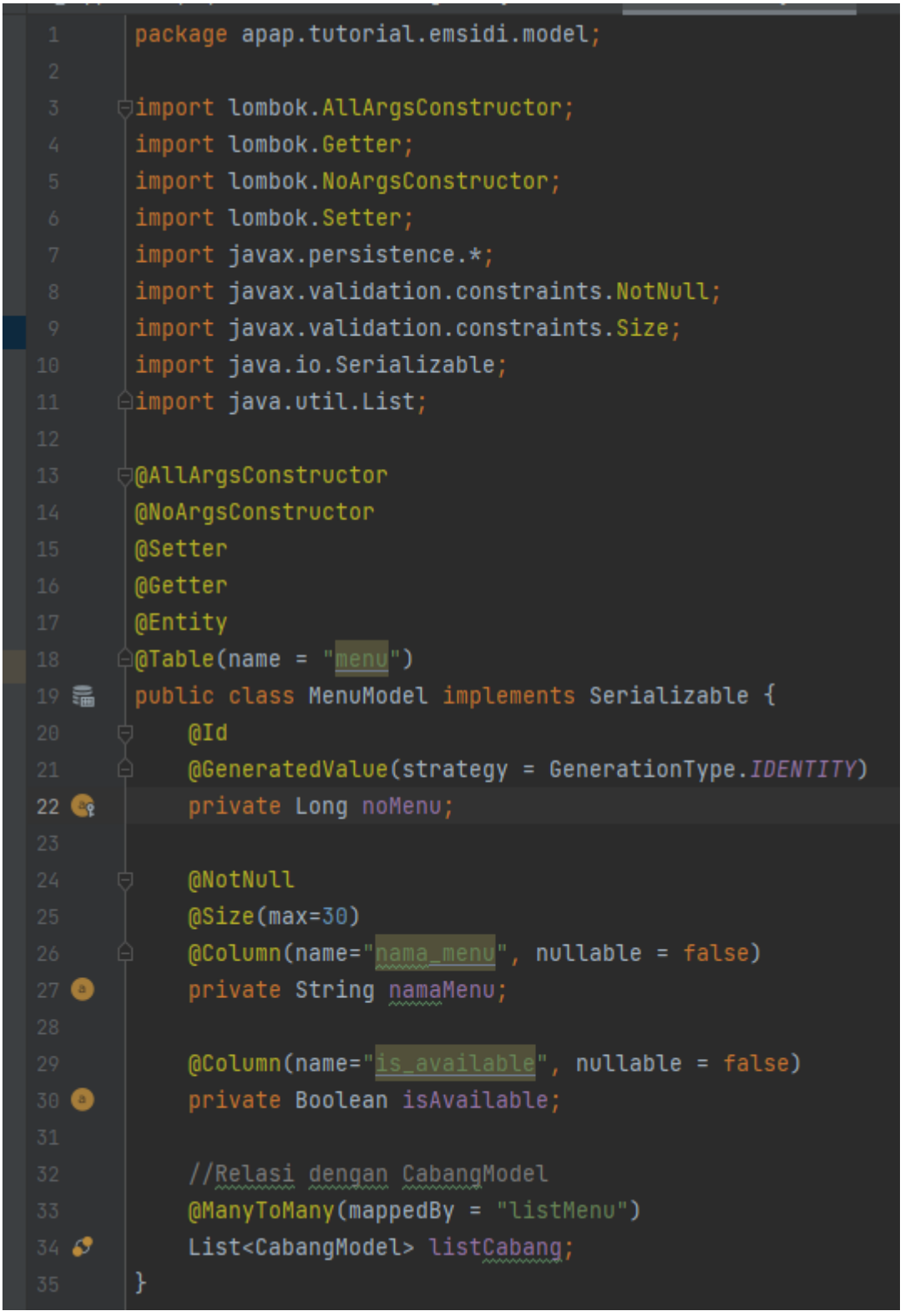
```java
package apap.tutorial.emsidi.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.format.annotation.DateTimeFormat;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.io.Serializable;
import java.time.LocalTime;
import java.util.List;

@AllArgsConstructor
@NoArgsConstructor
@Setter @Getter
@Entity
@Table(name = "cabang")
public class CabangModel implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long noCabang;

    @NotNull
    @Size(max=30)
    @Column(name="nama_cabang", nullable = false)
    private String namaCabang;

    @NotNull
    @Size(max=30)
    @Column(name="alamat_cabang", nullable = false)
    private String alamatCabang;

    @NotNull
    @Size(max=30)
    @Column(name="no_telepon_cabang", nullable = false)
    private String noTeleponCabang;
```

```
41
42      @NotNull
43      @Column(nullable = false)
44      @DateTimeFormat(pattern = "HH:mm")
45      private LocalTime waktuBuka;
46
47      @NotNull
48      @Column(nullable = false)
49      @DateTimeFormat(pattern = "HH:mm")
50      private LocalTime waktuTutup;
51
52  //    Relasi dengan PegawaiModel
53      @OneToMany(mappedBy = "cabang", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
54      private List<PegawaiModel> listPegawai;
55
56  //    Relasi dengan MenuModel
57      @ManyToMany
58      @JoinTable(
59              name = "cabang_menu",
60              joinColumns = @JoinColumn(name = "no_cabang"),
61              inverseJoinColumns = @JoinColumn(name = "no_menu"))
62      List<MenuModel> listMenu;
63  }
64
```

7. Buatlah **PegawaiModel** seperti berikut:

```java
package apap.tutorial.emsidi.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.io.Serializable;

@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@Entity
@Table(name = "pegawai")
public class PegawaiModel implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long noPegawai;

    @NotNull
    @Size(max = 30)
    @Column(name = "nama_pegawai", nullable = false)
    private String namaPegawai;

    @NotNull
    @Column(name = "jenis_kelamin", nullable = false)
    private int jenisKelamin;

    //Relasi dengan CabangModel
    @ManyToOne(fetch = FetchType.EAGER, optional = false)
    @JoinColumn(name = "noCabang", referencedColumnName = "noCabang", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE)
    private CabangModel cabang;
}
```

8. Buatlah **MenuModel** seperti berikut:



```java
package apap.tutorial.emsidi.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.io.Serializable;
import java.util.List;

@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@Entity
@Table(name = "menu")
public class MenuModel implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long noMenu;

    @NotNull
    @Size(max=30)
    @Column(name="nama_menu", nullable = false)
    private String namaMenu;

    @Column(name="is_available", nullable = false)
    private Boolean isAvailable;

    //Relasi dengan CabangModel
    @ManyToMany(mappedBy = "listMenu")
    List<CabangModel> listCabang;
}
```

Note: Periksa kembali dan pahami baik - baik relasi antar model. Kita harus paham bagaimana relasi antar model dan bagaimana relasi tersebut diimplementasikan di model

Tutorial Membuat *Repository*

Repository JPA merupakan interface yang mengandung fitur-fitur dan atribut elemen yang memungkinkan mendefinisikan repository beans.

9. Klik kanan pada Project > New > Package. Buatlah package **apap.tutorial.emsidi.repository**
10. Pada *package repository*, buatlah ***interface*** **CabangDb** seperti berikut:

```java
package apap.tutorial.emsidi.repository;

import apap.tutorial.emsidi.model.CabangModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface CabangDb extends JpaRepository<CabangModel, Long> {
    Optional<CabangModel> findByNoCabang(Long noCabang);
}
```

11. Buat juga ***interface*** **PegawaiDb** seperti berikut:

```java
package apap.tutorial.emsidi.repository;

import apap.tutorial.emsidi.model.PegawaiModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface PegawaiDb extends JpaRepository<PegawaiModel, Long> {

}
```

Tutorial Membuat *Service*

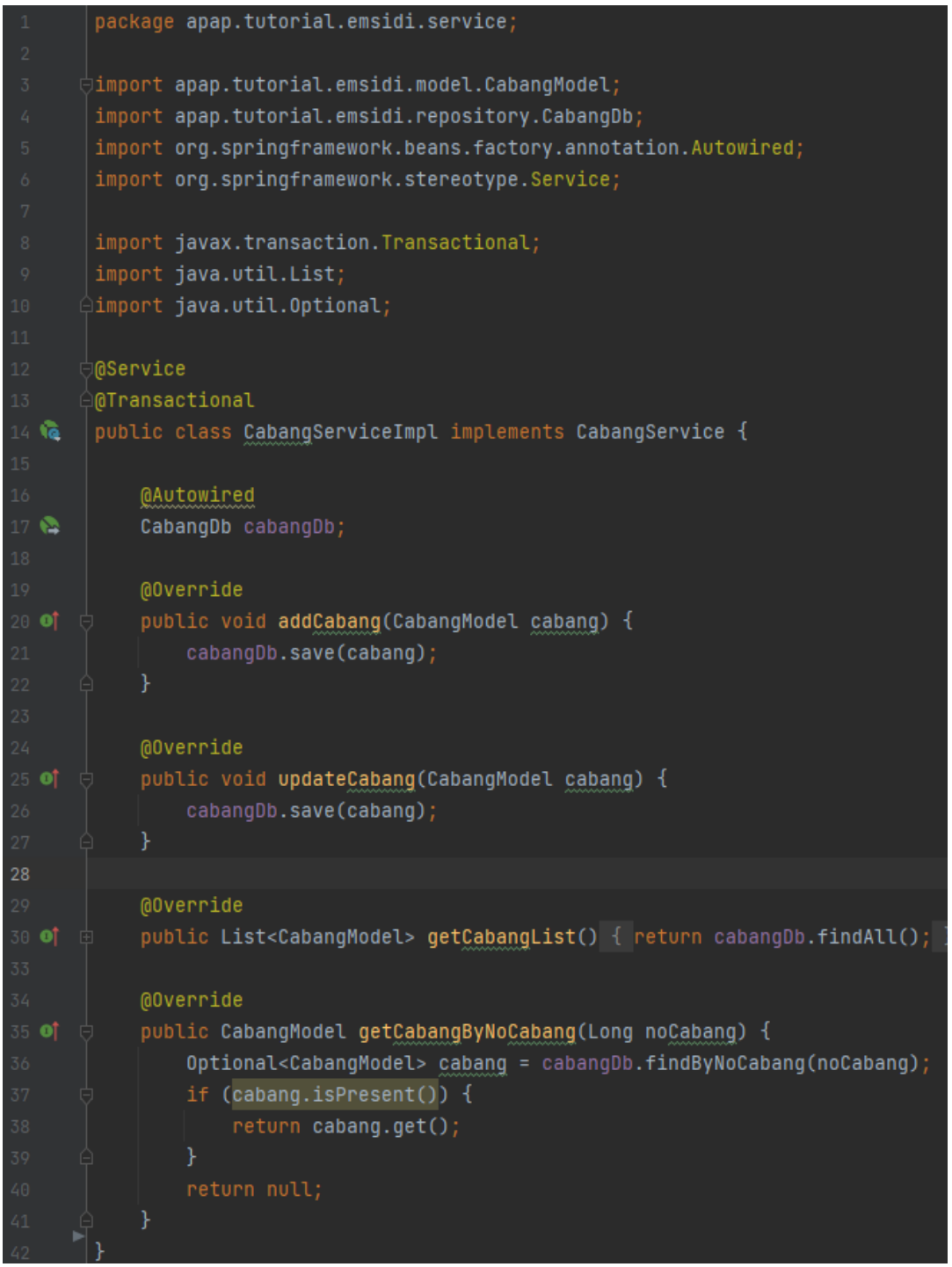
12. Klik kanan pada Project > New > Package. Buatlah *package* **apap.tutorial.emsidi.service**
13. Pada *package service* buatlah *interface* **CabangService** seperti berikut:

```java
package apap.tutorial.emsidi.service;

import apap.tutorial.emsidi.model.CabangModel;
import java.util.List;

public interface CabangService {
    void addCabang(CabangModel cabang);
    void updateCabang(CabangModel cabang);
    List<CabangModel> getCabangList();
    CabangModel getCabangByNoCabang(Long noCabang);
}
```

14. Buat juga *interface* **PegawaiService** seperti berikut:

```java
package apap.tutorial.emsidi.service;

import apap.tutorial.emsidi.model.PegawaiModel;

public interface PegawaiService {
    void addPegawai(PegawaiModel pegawai);
}
```

15. Kemudian buatlah class **CabangServiceImpl** dan **PegawaiServiceImpl** seperti berikut:



```java
package apap.tutorial.emsidi.service;

import apap.tutorial.emsidi.model.CabangModel;
import apap.tutorial.emsidi.repository.CabangDb;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.List;
import java.util.Optional;

@Service
@Transactional
public class CabangServiceImpl implements CabangService {

    @Autowired
    CabangDb cabangDb;

    @Override
    public void addCabang(CabangModel cabang) {
        cabangDb.save(cabang);
    }

    @Override
    public void updateCabang(CabangModel cabang) {
        cabangDb.save(cabang);
    }

    @Override
    public List<CabangModel> getCabangList() { return cabangDb.findAll(); }

    @Override
    public CabangModel getCabangByNoCabang(Long noCabang) {
        Optional<CabangModel> cabang = cabangDb.findByNoCabang(noCabang);
        if (cabang.isPresent()) {
            return cabang.get();
        }
        return null;
    }
}
```

```java
package apap.tutorial.emsidi.service;

import apap.tutorial.emsidi.model.PegawaiModel;
import apap.tutorial.emsidi.repository.PegawaiDb;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

@Service
@Transactional
public class PegawaiServiceImpl implements PegawaiService {

    @Autowired
    PegawaiDb pegawaiDb;

    @Override
    public void addPegawai(PegawaiModel pegawai) { pegawaiDb.save(pegawai); }
}
```

Tutorial Membuat *Controller*

16. Buatlah **BaseController** seperti berikut:

```java
package apap.tutorial.emsidi.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class BaseController {

    @GetMapping("/")
    private String home() { return "home"; }
}
```

17. Buatlah **CabangController** seperti berikut:

```java
package apap.tutorial.emsidi.controller;

import apap.tutorial.emsidi.model.CabangModel;
import apap.tutorial.emsidi.model.PegawaiModel;
import apap.tutorial.emsidi.service.CabangService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.ui.Model;

import java.util.List;

@Controller
public class CabangController {

    @Qualifier("cabangServiceImpl")
    @Autowired
    private CabangService cabangService;

    @GetMapping("/cabang/add")
    public String addCabangForm(Model model) {
        model.addAttribute( s: "cabang", new CabangModel());
        return "form-add-cabang";
    }

    @PostMapping("/cabang/add")
    public String addCabangSubmit(
            @ModelAttribute CabangModel cabang,
            Model model
    ) {
        cabangService.addCabang(cabang);
        model.addAttribute( s: "noCabang", cabang.getNoCabang());
        return "add-cabang";
    }

}
```

```java
@GetMapping("/cabang/viewall")
public String listCabang(Model model) {
    List<CabangModel> listCabang = cabangService.getCabangList();
    model.addAttribute( s: "listCabang", listCabang);
    return "viewall-cabang";
}

@GetMapping("/cabang/view")
public String viewDetailCabang(
        @RequestParam(value = "noCabang") Long noCabang,
        Model model
) {
    CabangModel cabang = cabangService.getCabangByNoCabang(noCabang);
    List<PegawaiModel> listPegawai = cabang.getListPegawai();

    model.addAttribute( s: "cabang", cabang);
    model.addAttribute( s: "listPegawai", listPegawai);

    return "view-cabang";
}

@GetMapping("/cabang/update/{noCabang}")
public String updateCabangForm(
        @PathVariable Long noCabang,
        Model model
) {
    CabangModel cabang = cabangService.getCabangByNoCabang(noCabang);
    model.addAttribute( s: "cabang", cabang);
    return "form-update-cabang";
}

@PostMapping("/cabang/update")
public String updateCabangSubmit(
        @ModelAttribute CabangModel cabang,
        Model model
) {
    cabangService.updateCabang(cabang);
    model.addAttribute( s: "noCabang", cabang.getNoCabang());
    return "update-cabang";
}
```

Tips: *Qualifier* digunakan untuk memastikan bahwa **cabangService** yang digunakan dari **cabangServiceImpl**

18. Buatlah **PegawaiController** seperti berikut:

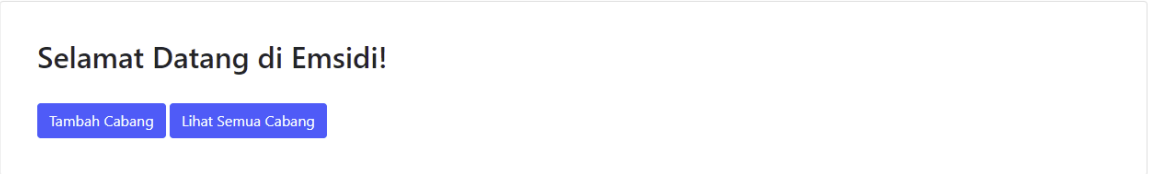
```java
package apap.tutorial.emsidi.controller;

import apap.tutorial.emsidi.model.CabangModel;
import apap.tutorial.emsidi.model.PegawaiModel;
import apap.tutorial.emsidi.service.CabangService;
import apap.tutorial.emsidi.service.PegawaiService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class PegawaiController {
    @Qualifier("pegawaiServiceImpl")
    @Autowired
    PegawaiService pegawaiService;

    @Qualifier("cabangServiceImpl")
    @Autowired
    CabangService cabangService;

    @GetMapping("/pegawai/add/{noCabang}")
    public String addPegawaiForm(@PathVariable Long noCabang, Model model){
        PegawaiModel pegawai = new PegawaiModel();
        CabangModel cabang = cabangService.getCabangByNoCabang(noCabang);
        pegawai.setCabang(cabang);
        model.addAttribute( s: "noCabang", noCabang);
        model.addAttribute( s: "pegawai", pegawai);
        return "form-add-pegawai";
    }
```

```java

    @PostMapping("/pegawai/add")
    public String addPegawaiSubmit(
            @ModelAttribute PegawaiModel pegawai,
            Model model
    ){
        pegawaiService.addPegawai(pegawai);
        model.addAttribute( s: "noCabang", pegawai.getCabang().getNoCabang());
        model.addAttribute( s: "namaPegawai", pegawai.getNamaPegawai());
        return "add-pegawai";
    }
}
```

Tutorial Membuat *View*

19. Buatlah **home.html** seperti berikut:
    https://pastebin.com/gQhyyjp5
20. Buatlah **form-add-cabang.html** seperti berikut:
    https://pastebin.com/cbktAz7j
21. Buatlah **form-add-pegawai.html** seperti berikut:
    https://pastebin.com/jM3L2nLL
22. Buatlah **form-update-cabang.html** seperti berikut:
    https://pastebin.com/3nmeqyNM
23. Buatlah **add-cabang.html** seperti berikut:
    https://pastebin.com/Kg4TQCwW
24. Buatlah **add-pegawai.html** seperti berikut:
    https://pastebin.com/xpkgsdBx
25. Buatlah **update-cabang.html** seperti berikut:
    https://pastebin.com/DvEUppQP
26. Buatlah **view-cabang.html** seperti berikut:
    https://pastebin.com/SDreDFKi
27. Buatlah **viewall-cabang.html** seperti berikut:
    https://pastebin.com/CkdVjPwe

    Tips: Silahkan mencoba-coba untuk mengakses semua *mapping* yang telah kamu buat. Pastikan semuanya sudah berjalan dengan lancar dan menampilkan sesuai yang diminta. Berikut ini sedikit spoiler tampilan Home yang seharusnya terlihat:



PERTANYAAN

Jawablah pertanyaan dibawah ini pada file **README**:

1. Tolong jelaskan secara singkat apa kegunaan dari anotasi-anotasi yang ada pada model (@AllArgsConstructor, @NoArgsConstructor, @Setter, @Getter, @Entity, @Table)

2. Pada class CabangDB, terdapat method `findByNoCabang`, apakah kegunaan dari method tersebut?

3. Jelaskan perbedaan kegunaan dari anotasi @JoinTable dan @JoinColumn

4. Pada class PegawaiModel, digunakan anotasi @JoinColumn pada atribut cabang, apa kegunaan dari name, referencedColumnName, dan nullable dalam anotasi tersebut? dan apa perbedaan nullable dan penggunaan anotasi @NotNull

5. Jelaskan kegunaan FetchType.LAZY, CascadeType.ALL, dan FetchType.EAGER

LATIHAN

1. Tambahkan fitur **View All Cabang** yang menampilkan seluruh cabang beserta atributnya **terurut** berdasarkan **nama cabang**. (***Hint***: Gunakan fitur yang dimiliki oleh JPA Repository!)
    a. **Spesifikasi**: Terdapat tombol **View All** pada **Home.**

2. Tambahkan fitur **Update Pegawai**. Fitur ini dapat mengubah seluruh informasi pegawai kecuali **id pegawai**. Spesifikasi dari fitur ini:
    a. Pegawai cabang hanya dapat di-*update* ketika **cabang sedang tutup**.
    b. Terdapat tombol **Update Pegawai** di setiap pegawai pada suatu halaman cabang.

3. Tambahkan fitur ***delete* pegawai cabang** yang dapat digunakan untuk menghapus seorang pegawai dari sebuah cabang. Spesifikasi dari fitur ini:
    a. Pegawai cabang hanya dapat di-*delete* ketika **cabang sedang tutup**.
    b. Terdapat tombol **Delete Pegawai** di setiap pegawai pada suatu halaman cabang.

4. Tambahkan fitur ***delete* Cabang** yang dapat digunakan untuk menghapus sebuah cabang. Spesifikasi dari fitur ini:
    a. Tombol ***delete*** terdapat di halaman suatu cabang.
    b. Cabang yang dapat di *delete* **hanyalah** cabang yang **tidak memiliki pegawai.**
    c. Cabang yang sedang pada jam buka **tidak dapat di-*delete*.**

5. Tambahkan **halaman *error*** berisi informasi sebuah cabang tidak ditemukan atau sebuah pekerjaan tidak berhasil dikerjakan (cth: jika cabang masih buka dan di-*delete*, maka kembalikan halaman *error*).

Notes: Untuk mencoba fitur ***update*** dan ***delete*** yang berkaitan dengan jam buka/tutup sebuah cabang, disarankan untuk membuat cabang yang waktu tutupnya sudah lewat dari waktu mencoba fiturnya. (Cth: mencoba fitur ***update*** pada jam 15.30, sehingga membuat cabang yang memiliki waktu buka jam 00.00 dan waktu tutup pada jam 01.00).