FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Ganjil 2018/2019

Menggunakan Database & Relasi Database dalam Project Spring Boot

Versi: 1.2 (3 Oktober 2018) *revisi ditandai dengan warna merah

Outline:

- Melakukan mapping to relational menggunakan database server pada model di tutorial sebelumnya

Requirements:

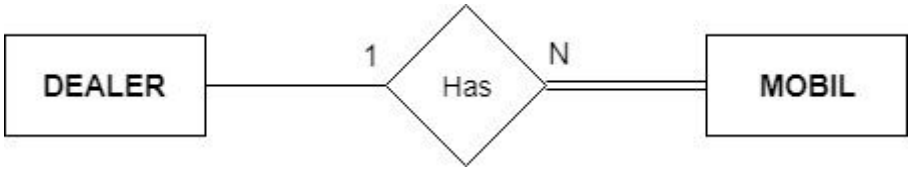
Sebelum memulai tutorial ini pastikan komputer Anda sudah terinstall JDK, Eclipse atau STS, Maven, XAMPP atau database server lainnya.

Pendahuluan

Pada tutorial kali ini Anda akan menggunakan modul JPA untuk mengintegrasikan database server Anda. Java Persistence API (JPA) adalah sebuah *interface* pemrograman aplikasi Java yang menggambarkan pengelolaan data relasional dalam database yang memungkinkan manipulasi data tanpa menggunakan query (dokumentasi 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 juga menambahkan *entity* baru serta menghubungkannya dengan entity sebelumnya.

ERD untuk hubungan antara Mobil dan Dealer adalah sebagai berikut:



Hubungan antara Dealer dan Mobile adalah One-to-Many karena Dealer dapat memiliki beberapa Mobil tetapi Mobil hanya dapat dimiliki oleh sebuah Dealer.

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

(*Note*: Implementasi database menggunakan JPA, tanpa harus membuat skema database secara manual)

DEALER

- id [PRIMARY KEY]
  long, NOT NULL, AUTO INCREMENT
- alamat
  varchar(50), NOT NULL
- no_telp
  double, NOT NULL

MOBIL

- id [PRIMARY KEY]
  long, NOT NULL, AUTO INCREMENT
- brand
  varchar(50), NOT NULL
- type
  varchar(50), NOT NULL, UNIQUE
- price
  bigint, NOT NULL
- amount
  integer, NOT NULL
- dealer_id
  long, NOT NULL

Tutorial

Anda bisa Buat New Project seperti biasa. Langkah:

1. Pada Eclipse/STS klik File > New > Other > **Spring Boot** > Spring Starter Project, tekan Next
2. Pada kolom group ganti nama menjadi com.apap
3. Beri nama *project*, *description*, *package,* dll. **Beri nama package com.apap.tutorial4**
4. Pilih **Web**, **Thymeleaf**, **DevTools, JPA dan MySQL** pada bagian *dependency*
5. Finish

Anda bisa memastikan apakah MySQL sudah terinstall atau belum dengan membuka pom.xml dan cek apakah ketika dependency tersebut telah tertulis di antara <dependencies></dependencies>

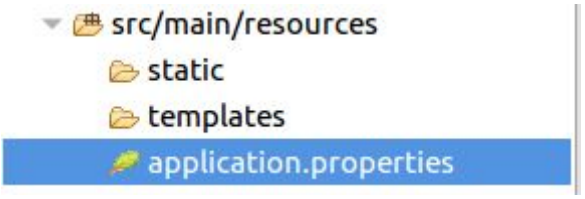
```
...
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
...
```

Jika belum Anda bisa menambahkan kode diatas ke pom.xml Anda dan Run as > Maven Install untuk mengunduh ketiga dependencies tersebut.

Tools yang Anda pakai pada tutorial kali ini yaitu JPA. Penggunaan JPA dalam pemrograman java dengan pendekatan Object Relational Maping (ORM). Dengan menggunakan JPA, memungkinkan 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.

application.properties

Jika Anda perhatikan pada folder **src/main/resource**, Anda akan menemukan sebuah file bernama application.properties. File ini berisi konfigurasi aplikasi Anda. File ini kosong awalnya namun Anda dapat menambahkan konfigurasi yang Anda inginkan.



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

application.properties

src ▸ main ▸ resources ▸ application.properties

```
1   # Konfigurasi untuk koneksi MySql
2   spring.datasource.platform=mysql
3   spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
4
5   # Sesuaikan NAMA_DATABASE dengan nama database anda
6   spring.datasource.url=jdbc:mysql://localhost:3306/[NAMA_DATABASE]?useSSL=false&serverTimezone=Asia/Jakarta
7
8   # Sesuaikan dengan NAMA dan PASSWORD mysql anda
9   spring.datasource.username=[NAMA]
10  spring.datasource.password=[PASSWORD]
11
12  # Optimize query untuk db MySql
13  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
14
15  # Pembuatan database (create || create-drop || validate || update)
16  spring.jpa.hibernate.ddl-auto=update
```

Biasanya application.properties akan di ignore pada file .gitignore jika Anda push ke repository karena setiap machine memiliki konfigurasi yang berbeda. Sehingga perlu membuat file application.properties lagi. Konsep application.properties ini tidak hanya di Spring Boot.

Framework lainnya juga mengenal file konfigurasi. Misalkan, file .env pada Laravel, nodejs, python, dll

Pada tutorial sebelumnya, Anda sudah mengenal konsep MVC. Tutorial ini Anda akan mengimplementasikan database menggunakan JPA.

Membuat *Class* Model

1. Klik kanan pada Project > New > Package. Buatlah package **com.apap.tutorial4.model**
2. Pada package model, buatlah class **DealerModel** dan **CarModel** dengan spesifikasi seperti berikut.

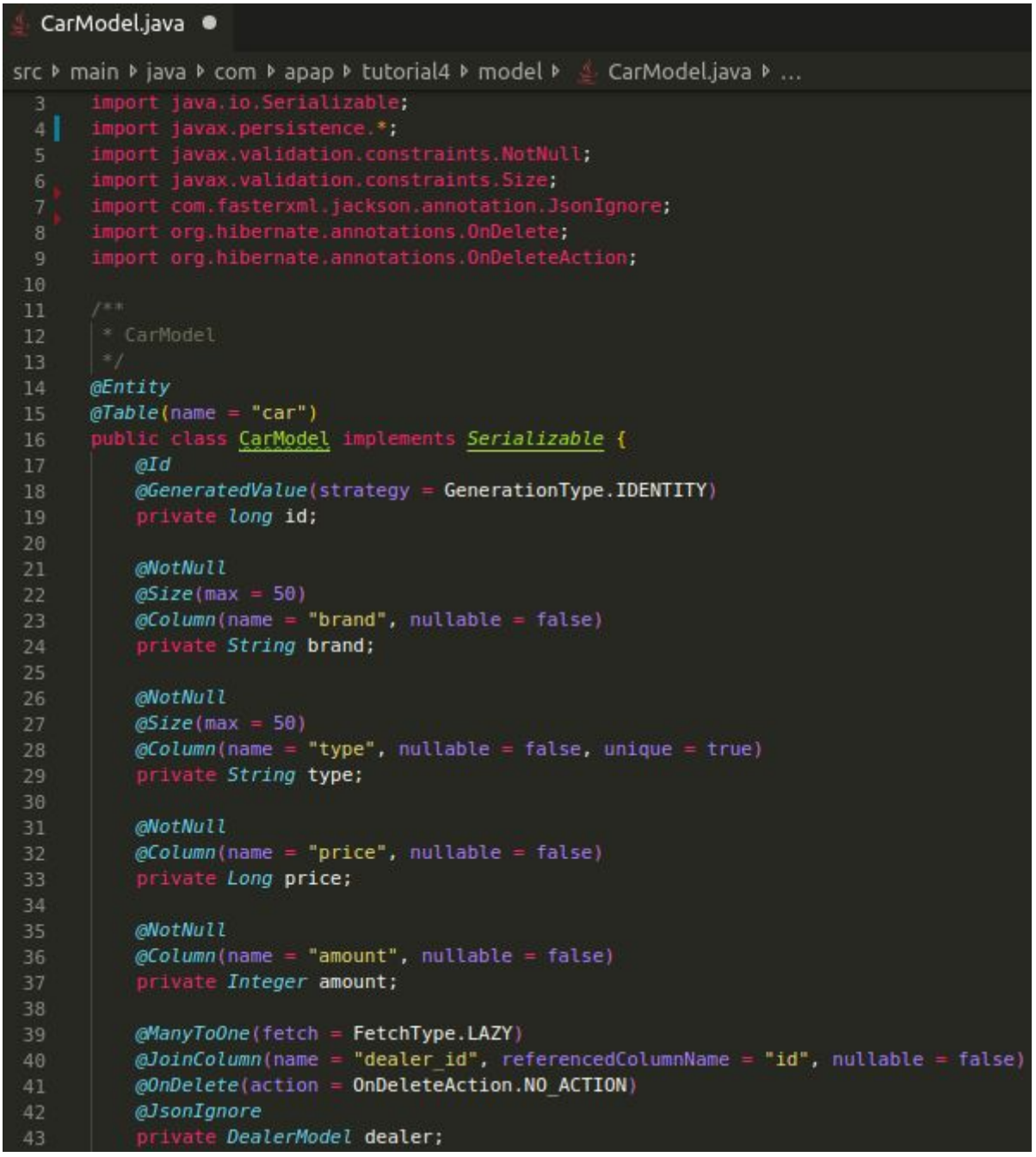
DealerModel

DealerModel.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ model ▸ DealerModel.java ▸ DealerModel

```
3   import java.io.Serializable;
4   import java.util.List;
5   import javax.persistence.*;
6   import javax.validation.constraints.NotNull;
7   import javax.validation.constraints.Size;
8
9   /**
10   * DealerModel
11   */
12  @Entity
13  @Table(name = "dealer")
14  public class DealerModel implements Serializable {
15      @Id
16      @GeneratedValue(strategy = GenerationType.IDENTITY)
17      private long id;
18
19      @NotNull
20      @Size(max = 50)
21      @Column(name = "alamat", nullable = false)
22      private String alamat;
23
24      @NotNull
25      @Size(max = 13)
26      @Column(name = "no_telp", nullable = false)
27      private String noTelp;
28
29      @OneToMany(mappedBy = "dealer", fetch = FetchType.LAZY, cascade = CascadeType.PERSIST)
30      private List<CarModel> listCar;
```

CarModel



3. Tambahkan *method setter*, dan *getter*.

Membuat *Repository*

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

1. Klik kanan pada Project > New > Package. Buatlah package **com.apap.tutorial4.repository**
2. Pada package repository, buatlah class **DealerDb** dan **CarDb** dengan spesifikasi seperti berikut.

DealerDb

DealerDb.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ repository ▸ DealerDb.java ▸ {} co

```
3   import com.apap.tutorial4.model.DealerModel;
4
5   import org.springframework.data.jpa.repository.JpaRepository;
6   import org.springframework.stereotype.Repository;
7
8   /**
9    * DealerDb
10   */
11  @Repository
12  public interface DealerDb extends JpaRepository<DealerModel, Long> {
```

CarDb

CarDb.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ repository ▸ CarDb.java ▸

```
3   import java.util.List;
4
5   import com.apap.tutorial4.model.CarModel;
6
7   import org.springframework.data.jpa.repository.JpaRepository;
8   import org.springframework.stereotype.Repository;
9
10  /**
11   * CarDb
12   */
13  @Repository
14  public interface CarDb extends JpaRepository<CarModel, Long> {
15      CarModel findByType(String type);
```

Membuat *Controller*

1. Klik kanan pada Project > New > Package. Buatlah package **com.apap.tutorial4.service**
2. Buatlah *interface* **DealerService.java** dan **CarService.java** pada *package* tersebut dengan isi sebagai berikut:

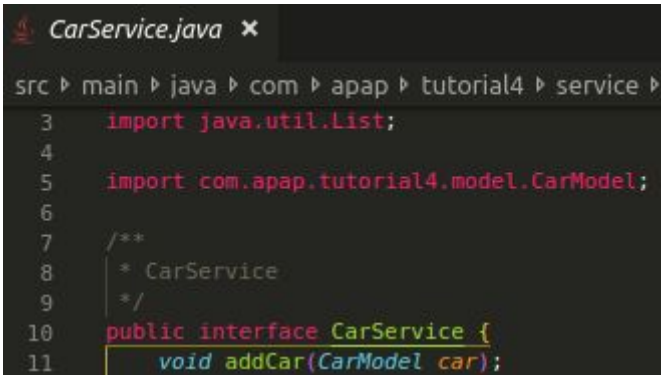
DealerService.java

DealerService.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ service ▸ DealerServic

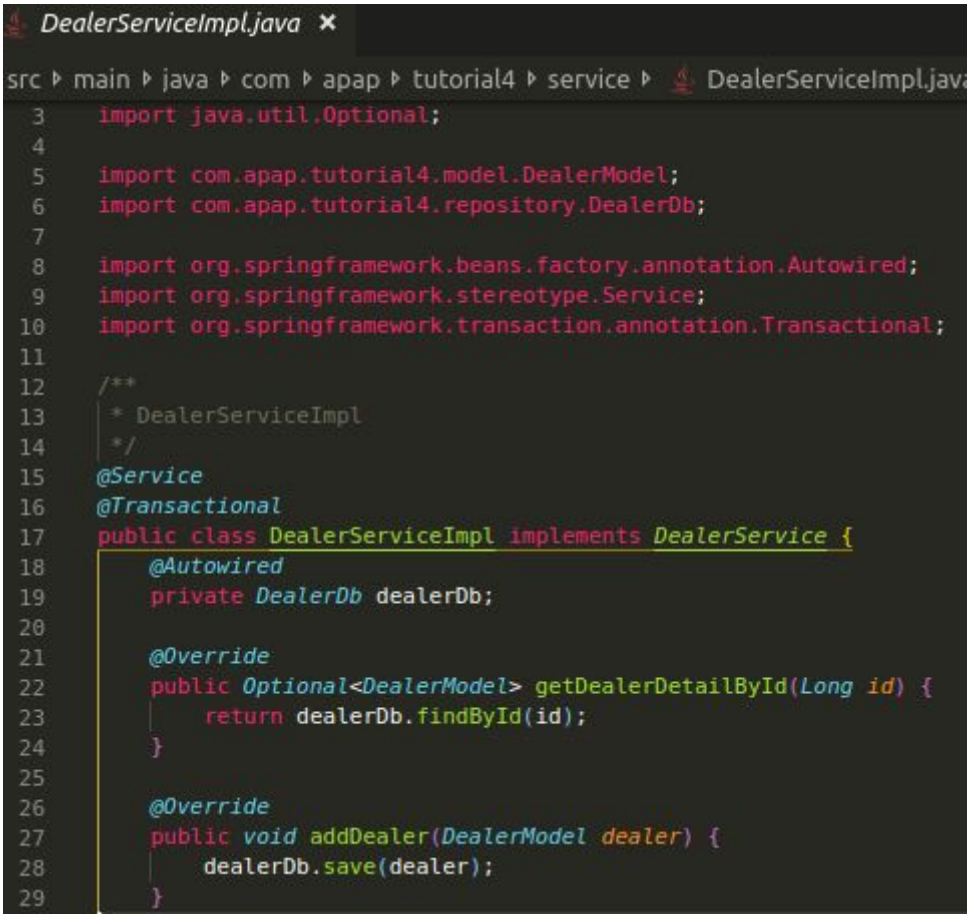
```
3   import java.util.Optional;
4
5   import com.apap.tutorial4.model.DealerModel;
6
7   /**
8    * DealerService
9    */
10  public interface DealerService {
11      Optional<DealerModel> getDealerDetailById(Long id);
12
13      void addDealer(DealerModel dealer);
```

CarService.java



3. Pada package yang sama, buat class **DealerServiceImpl** yang meng-*implements* DealerService dan **CarServiceImpl** yang meng-*implements* CarService dengan isi sebagai berikut:

DealerServiceImpl



CarServiceImpl

CarServiceImpl.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ service ▸ CarServiceImpl.java

```
3   import java.util.List;
4
5   import com.apap.tutorial4.model.CarModel;
6   import com.apap.tutorial4.repository.CarDb;
7
8   import org.springframework.beans.factory.annotation.Autowired;
9   import org.springframework.stereotype.Service;
10  import org.springframework.transaction.annotation.Transactional;
11
12  /**
13   * CarServiceImpl
14   */
15  @Service
16  @Transactional
17  public class CarServiceImpl implements CarService {
18      @Autowired
19      private CarDb carDb;
20
21      @Override
22      public void addCar(CarModel car) {
23          carDb.save(car);
24      }
```

Membuat *Controller*

1. Klik kanan pada Project > New > Package. Buatlah *package* **com.apap.tutorial4.controller**
2. @ModelAttribute merupakan metode beranotasi yang memungkinkan akses ke objek di view. Buatlah *class* **DealerController** dan **CarController** dengan isi sebagai berikut:

DealerController

DealerController.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ controller ▸ DealerController.java ▸ De

```
3   import java.util.List;
4
5   import com.apap.tutorial4.model.*;
6   import com.apap.tutorial4.service.*;
7   import org.springframework.beans.factory.annotation.Autowired;
8   import org.springframework.stereotype.Controller;
9   import org.springframework.ui.Model;
10  import org.springframework.web.bind.annotation.*;
11
12  /**
13   * DealerController
14   */
15  @Controller
16  public class DealerController {
17      @Autowired
18      private DealerService dealerService;
19
20      @Autowired
21      private CarService carService;
22
23      @RequestMapping("/")
24      private String home() {
25          return "home";
26      }
27
28      @RequestMapping(value = "/dealer/add", method = RequestMethod.GET)
29      private String add(Model model) {
30          model.addAttribute("dealer", new DealerModel());
31          return "addDealer";
32      }
33
34      @RequestMapping(value = "/dealer/add", method = RequestMethod.POST)
35      private String addDealerSubmit(@ModelAttribute DealerModel dealer) {
36          dealerService.addDealer(dealer);
37          return "add";
38      }
```

CarController

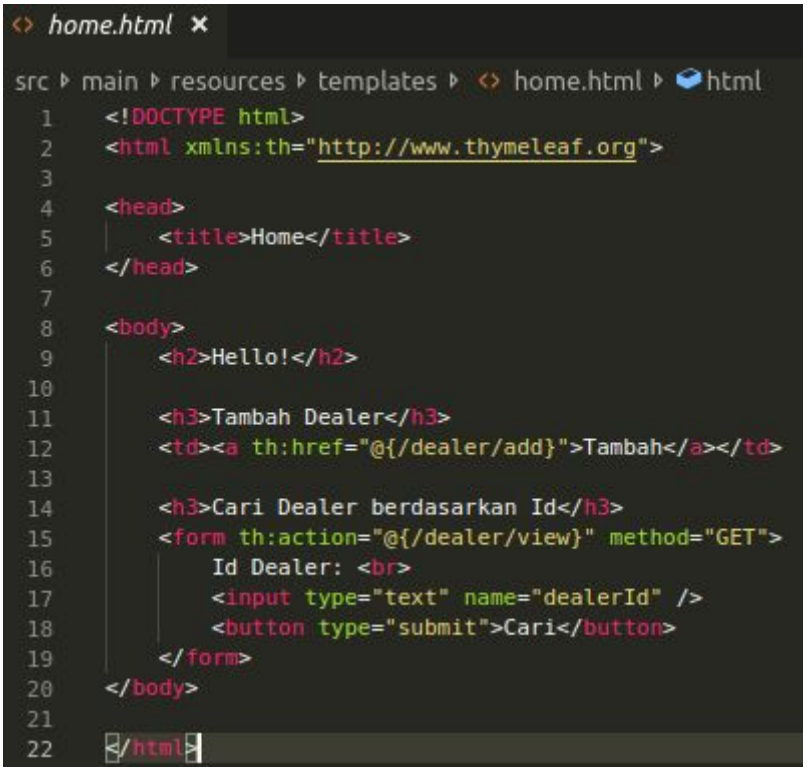
CarController.java

src ▸ main ▸ java ▸ com ▸ apap ▸ tutorial4 ▸ controller ▸ CarController.java ▸ CarController

```java
import com.apap.tutorial4.model.*;
import com.apap.tutorial4.service.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

/**
 * CarController
 */
@Controller
public class CarController {
    @Autowired
    private CarService carService;

    @Autowired
    private DealerService dealerService;

    @RequestMapping(value = "/car/add/{dealerId}", method = RequestMethod.GET)
    private String add(@PathVariable(value = "dealerId") Long dealerId, Model model) {
        CarModel car = new CarModel();
        DealerModel dealer = dealerService.getDealerDetailById(dealerId).get();
        car.setDealer(dealer);

        model.addAttribute("car", car);
        return "addCar";
    }

    @RequestMapping(value = "/car/add", method = RequestMethod.POST)
    private String addCarSubmit(@ModelAttribute CarModel car) {
        carService.addCar(car);
        return "add";
    }
```

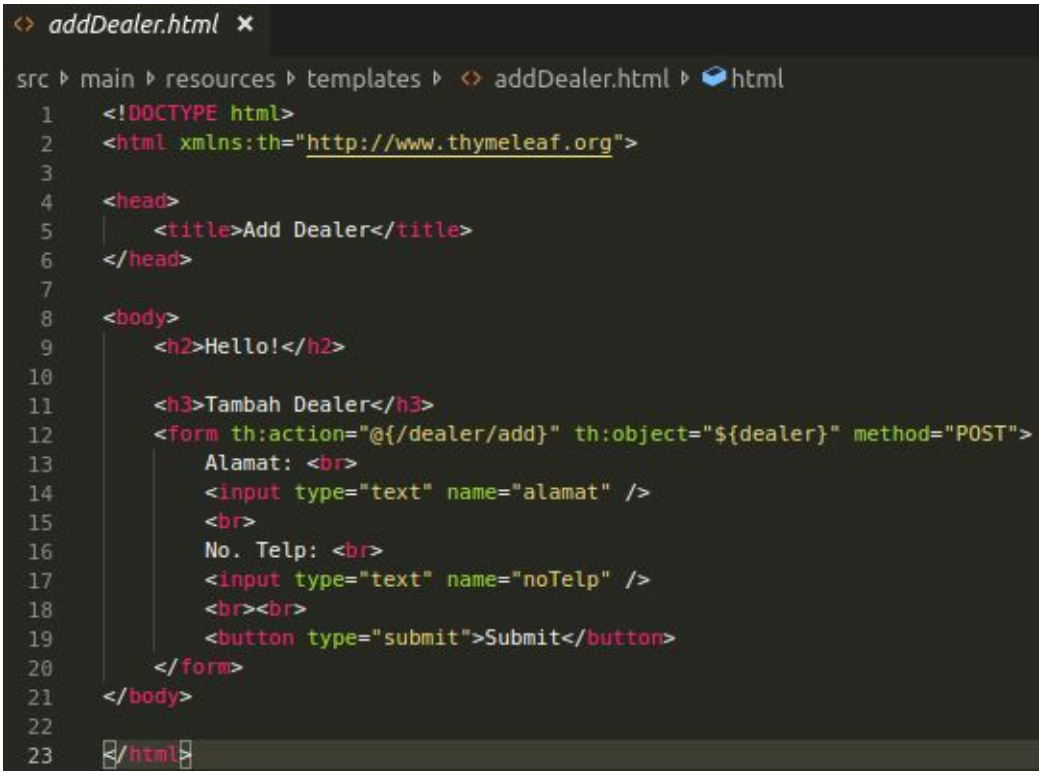
Membuat *View*

1. home.html



```
home.html
src ▸ main ▸ resources ▸ templates ▸ home.html ▸ html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <title>Home</title>
</head>

<body>
    <h2>Hello!</h2>

    <h3>Tambah Dealer</h3>
    <td><a th:href="@{/dealer/add}">Tambah</a></td>

    <h3>Cari Dealer berdasarkan Id</h3>
    <form th:action="@{/dealer/view}" method="GET">
        Id Dealer: <br>
        <input type="text" name="dealerId" />
        <button type="submit">Cari</button>
    </form>
</body>

</html>
```

2. addDealer.html dan addCar.html



```
addDealer.html
src ▸ main ▸ resources ▸ templates ▸ addDealer.html ▸ html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <title>Add Dealer</title>
</head>

<body>
    <h2>Hello!</h2>

    <h3>Tambah Dealer</h3>
    <form th:action="@{/dealer/add}" th:object="${dealer}" method="POST">
        Alamat: <br>
        <input type="text" name="alamat" />
        <br>
        No. Telp: <br>
        <input type="text" name="noTelp" />
        <br><br>
        <button type="submit">Submit</button>
    </form>
</body>

</html>
```

addCar.html

src ▸ main ▸ resources ▸ templates ▸ addCar.html ▸ html

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

<head>
    <title>Add Car</title>
</head>

<body>
    <h2>Hello!</h2>

    <h3>Tambah Mobil</h3>
    <form th:action="@{/car/add}" th:object="${car}" method="POST">
        <input type="hidden" th:field="*{dealer}"/>
        Brand: <br>
        <input type="text" name="brand" />
        <br>
        Type: <br>
        <input type="text" name="type" />
        <br>
        Price: <br>
        <input type="number" name="price" />
        <br>
        Amount: <br>
        <input type="number" name="amount" />
        <br><br>
        <button type="submit">Submit</button>
    </form>
</body>

</html>
```

3. add.html

add.html

src ▸ main ▸ resources ▸ templates ▸ add.html ▸ html

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

<head>
    <title>Add</title>
</head>

<body>
    <h2>Data Berhasil Ditambahkan</h2>
</body>

</html>
```

4. Jalankan program, **http://localhost:8080/** kemudian jalankan menambah dealer

View Dealer berdasarkan ID

1. Implementasikan method di DealerController untuk menampilkan sebuah dealer berdasarkan id dealer. Method ini menerima parameter id dealer dan mengembalikan view (view-dealer.html).
2. Buatlah tampilan view-dealer.html seperti gambar dibawah ini. (Tambahkan link untuk menambahkan mobil)

revisi

3. Jalankan program untuk menambah mobil dalam sebuah dealer

Latihan

1. Ubah method **view dealer agar menampilkan daftar mobil beserta detail terurut berdasarkan harga mobil**.
2. Buatlah fitur **delete** untuk menghapus dealer dan sebuah mobil.
3. Buatlah fitur **update** untuk seorang dealer dan sebuah mobil.
4. Buatlah fitur untuk melihat view all dealer sehingga dapat menampilkan brand mobil beserta amount dan price yang dijual.

Deliverables

Pada tutorial ini, ada beberapa *deliverables* yang Anda perlu kerjakan dan kumpulkan. *Deliverables* tersebut adalah sebagai berikut:

1. *Folder project* Anda yang berisi implementasi dari bagian 'Latihan' yang terdapat dalam tutorial ini. Tidak masalah jika dalam project tersebut juga ada hasil tutorial Anda.
1. *File write-up* berisi hasil jawaban dari setiap poin pada bagian latihan tutorial dalam format txt atau pdf. Masukkan dalam *folder project* dan pastikan *file write-up* juga di-*push* ke repositori GitHub.

Pengumpulan

Buat sebuah **project** di organisasi GitHub **/Apap-2018** dengan format nama tutorial4_[NPM]. Contoh: **tutorial4_1601234567**. *Push* seluruh isi *folder project* Anda, termasuk *file write-up*, ke repositori *project* tersebut. Anda tidak perlu membuat *branch* baru. Cukup *push* ke *branch master* saja.

Perhatikan dalam membuat **public** project tersebut. Pastikan Anda berada di dalam organisasi GitHub **/apap-2018** terlebih dahulu.

Commit* yang akan dinilai adalah *commit* terakhir yang di *push* ke repositori sebelum *deadline. Waktu yang akan dijadikan patokan adalah waktu *commit* di *server* GitHub. Pastikan Anda tidak *push commit* lagi setelah *deadline* jika tidak ingin terkena penalti.

Deadline

2 Oktober 2018, 23:59:59

Penalti

Penalti berlaku untuk kasus:

- Keterlambatan
  Nilai total akan ditambahkan -10 poin untuk setiap 1 jam keterlambatan