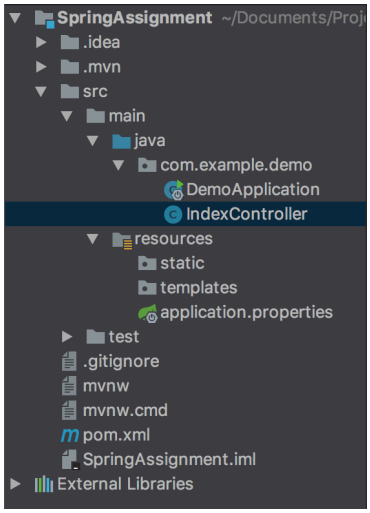
MODUL 1
KONSEP MVC

A. Menghubungkan Controller dengan View

- Jika INTELLIJ IDEA-mu versi community, buatlah project di start.spring.io, dependency yang ditambahkan **WEB** dan **THYMELEAF**. Jika INTELLIJ IDEA-mu versi ultimate, bisa langsung buat project spring di spring initializr dengan dependency **WEB** dan **THYMELEAF**.
- Setelah itu, download project dan buka di INTELLIJ IDEA.
- Buka **pom.xml** dan ubah bagian version menjadi 1.5.7.RELEASE

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

- Buatlah controller dengan nama IndexController



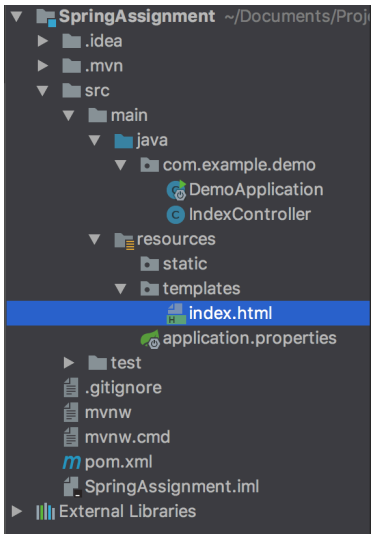
- Code IndexController

```
@RestController
public class IndexController {

    @GetMapping(value = "/")
    public ModelAndView index(){
        ModelAndView mav = new ModelAndView();
        mav.setViewName("index");
        return mav;
    }
}
```

- @RestController adalah annotation yang menandakan bahwa class tersebut adalah sebuah Controller.
- @GetMapping adalah annotation yang menandakan bahwa method tersebut terjadi saat ada method GET pada url yang ditentukan.
  (dalam hal ini adalah url "/ → localhost:8080/")

- ModelAndView adalah sebuah object yang menampung model (data dari database) untuk di-pass ke view, dan view adalah tampilan (.html)
- mav.setViewName adalah untuk men-set nama html yang ingin digunakan apabila pengguna mengakses url "/".
- Buatlah HTML bernama index.html



- Di dalam index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset=UTF-8"/>
    <title>INDEX</title>
</head>
<body>
    <h2>INDEX PAGE</h2>
</body>
</html>
```

- xmlns:th=http://www.thymeleaf.org adalah tag untuk menggunakan fungsi2 yang disediakan thymeleaf seperti th:block, th:if, th:each, dll.
- Jangan lupa, jika menggunakan thymeleaf, selalu akhiri dengan tag /> (jangan hanya >). Seperti pada <meta … />
- Sekarang **RUN**, lalu akses localhost:8080.

B. ASSIGNMENT

- Buatlah class model untuk DataDiri dengan getter dan setter variabel id, nama, dan umur.
  Buatlah juga controller dengan nama **DataDiriController** dan view bernama **form.html**, jika mengakses /form maka akan mengeluarkan html yang isinya adalah form untuk mengisi nama, umur, dan button submit. **Tampilan tidak perlu bagus, yang ditekankan disini adalah fungsionalitas backend (spring) saja** ☺

MODUL 2

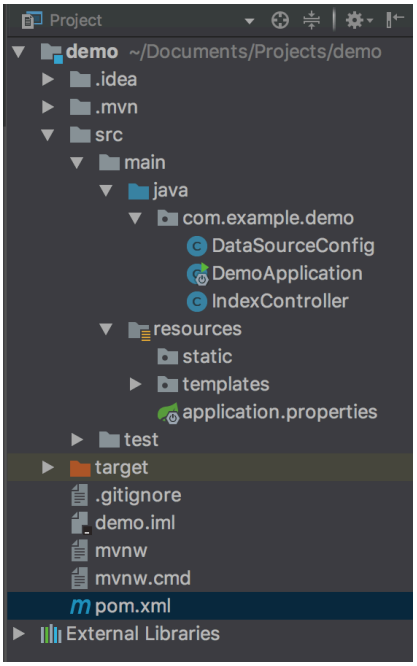
SETUP DATABASE DAN CREATE TABLE

A. DEPENDENCY DAN SETTING DATABASE

- Tambahkan dependency untuk **JDBC** dan **POSTGRESQL** di **pom.xml**

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <scope>runtime</scope>
</dependency>
```

- Buatlah class DataSourceConfig



- Di dalam DataSourceConfig

```
public class DataSourceConfig {

    public static DriverManagerDataSource dataSource() {
        DriverManagerDataSource driverManagerDataSource = new
DriverManagerDataSource();
        driverManagerDataSource.setDriverClassName("org.postgresql.Driver");

driverManagerDataSource.setUrl("jdbc:postgresql://localhost:port/Nama_Database"
);
        driverManagerDataSource.setUsername("username_database");
        driverManagerDataSource.setPassword("password_database");
        return driverManagerDataSource;
    }
}
```

- Setting pada port, nama database, username_database, dan password_database sesuai dengan postgresql mu.. Jangan lupa untuk run postgresql-mu juga di PC-mu.

- Lalu, pada file application.properties, seperti ini

```
spring.datasource.url= jdbc:postgresql://localhost:port/nama_database
spring.datasource.username=username_database
spring.datasource.password=password_database
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

- Setting pada port, nama_database, username_database, dan password_database yang sesuai.

B. Create Table

- Buat class dengan nama DataDiriDao, lalu ganti **class** menjadi **interface**.
- Isinya DataDiriDao adalah :

```
public interface DataDiriDao {
    void CreateTable();
    void InsertData(DataDiri dataDiri);
    List<DataDiri> GetAllData();
    List<DataDiri> GetOneData(int id);
    String GetNameById(int id);
    void Update(DataDiri dataDiri);
    void Delete(int id);
}
```

- Jadi, interface itu isinya adalah method-method yang nantinya akan kita *implements* di suatu class. Makannya interface DataDiriDao di atas itu isinya adalah method-method beserta tipe data method nya (void, List, dan String). Interface DataDiriDao ini nantinya akan kita *implements* di class DataDiriImpl.

- Buatlah class DataDiriImpl. Isinya DataDiriImpl adalah :

```
public class DataDiriImpl implements DataDiriDao {

    JdbcTemplate jdbcTemplate = new JdbcTemplate();

    public DataDiriImpl(){
        jdbcTemplate.setDataSource(DataSourceConfig.dataSource());
    }

    @Override
    public void CreateTable() {
        String query = "CREATE TABLE IF NOT EXISTS data_diri" +
                "(id serial primary key, nama TEXT, umur INT)";
        jdbcTemplate.execute(query);
    }

    @Override
    public void InsertData(DataDiri dataDiri) {

    }

    @Override
    public List<DataDiri> GetAllData() {
        return null;
    }

    @Override
    public List<DataDiri> GetOneData(int id) {
        return null;
    }

    @Override
    public String GetNameById(int id) {
        return null;
    }

    @Override
    public void Update(DataDiri dataDiri) {

    }

    @Override
    public void Delete(int id) {

    }
}
```

- Class DataDiriImpl *implements* DataDiriDao (lihat di atas). Lalu, tekan alt+enter pada code yang bergaris bawah merah dan klik *implements method*. Maka method2 yang ada pada interface DataDiriDao tadi akan muncul dengan sendirinya di class DataDiriImpl ini.
- JdbcTemplate jdbcTemplate = new JdbcTemplate(). JdbcTemplate adalah sebuah mekanisme untuk mengoneksikan ke database dan mengeksekusi query.

- Buatlah constructor untuk DataDiriImpl, isinya adalah jdbcTemplate.setDataSource(DataSourceConfig.dataSource()); Maksudnya adalah kita men-set dataSource untuk digunakan jdbcTemplate. DataSource adalah settingan untuk databasenya, seperti port nya, nama database, username database, dan password database (konfigurasi itu sudah dibuat di class DataSourceConfig).
- Pada kali ini, kita akan fokus hanya pada method CreateTable saja, yaitu method untuk membuat table pada database. Query nya adalah query biasa untuk membuat table di database. Lalu query tersebut dieksekusi oleh jdbcTemplate.
- Sekarang, eksekusi method CreateTable itu di main method dari spring kita.

```
@SpringBootApplication
public class DemoApplication {

   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
      DataDiriImpl dataDiriImpl = new DataDiriImpl();
      dataDiri.CreateTable();
   }
}
```

- Sekarang **RUN** aplikasi spring-nya, dan jika sudah di RUN, lihat di pgAdmin, akan terlihat bahwa table data_diri sudah dibuat.

C. ASSIGNMENT

- Buatlah sebuah class model yang berisi getter dan setter variabel id, nama, jenjang_pendidikan.
- Buatlah table di database dengan nama data_anak, columnya adalah : **id** INT, **nama** text, **jenjang_pendidikan** text, dengan column id adalah **foreign key** dari column ID yang ada pada table **data_diri**. Berikut adalah contoh query untuk table dengan column **name** merupakan foreign key dari column name table **users.**
  Ex : CREATE TABLE if not exists family (id serial primary key, name text), CONSTRAINT name_fk FOREIGN KEY (name) REFERENCES users(name));

MODUL 3

INSERT DATA KE DATABASE

A. Menggunakan model, view, dan controller yang sudah dibuat sebelumnya untuk insert data ke database

- Pertama tama, kita buat query untuk insert ke database dulu di dalam class DataDiriImpl (di method **InsertData** yang sudah ada sebelumnya).

```
@Override
public void InsertData(DataDiri dataDiri) {
    String query = "INSERT INTO data_diri (nama, umur) VALUES (?,?)";
    jdbcTemplate.update(query, new Object[] { dataDiri.getNama(),
dataDiri.getUmur() });
}
```

- Query nya adalah query insert seperti biasanya pada SQL. Yang berbeda adalah bagian ***VALUES (?,?)***. Maksudnya ***?,?*** adalah sebagai perwakilan untuk value yang akan dimasukkan ke column nama (tanda tanya pertama) dan column umur (tanda tanya kedua). Lalu bagaimana cara assign value nya? Ada pada saat **jdbcTemplate.update**. Terlihat di dalamnya ada new Object[] { dataDiri.getNama(), dataDiri.getUmur() }. Perlu diperhatikan bahwa urutannya ini tidak boleh dibolak-balik, karena pada query itu kita meletakkan column nama di urutan pertama, maka disini dataDiri.getNama() juga di urutan pertama. Begitu juga dengan column umur yang ada pada urutan kedua.
- Model yang sudah dibuat sebelumnya adalah model **DataDiri**, view nya adalah **form.html**, controller nya adalah **DataDiriController**.
- Pada MODUL 1, ada assignment untuk membuat sistem, jika mengakses /form maka akan muncul form.html yang isinya adalah form dengan input nama, umur, dan button submit. Kita akan menggunakan form ini untuk memasukkan data ke database.
- Pada **form.html**, pastikan pada elemen **<input/>** parameter **name** sesuai dengan yang ada di model **DataDiri**.

- Berikut adalah contoh model DataDiri

```
public class DataDiri {
    private int id, umur;
    private String nama;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getUmur() {
        return umur;
    }

    public void setUmur(int umur) {
        this.umur = umur;
    }

    public String getNama() {
        return nama;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }
}
```

- Pada **form.html**

```
<form>
    <label for="input_nama">Nama : </label>
    <input id="input_nama" type="text" name="nama" />
    <label for="input_umur">Umur : </label>
    <input id="input_umur" type="number" name="umur" />
    <button type="submit">SUBMIT</button>
</form>
```

- Perlu diperhatikan pada elemen **<input/>**, tag diakhiri dengan **/> bukan** > (supaya tidak error di thymeleaf).
- Lalu pada tiap elemen input, ada parameter **name**, parameter name ini disesuaikan ya dengan yang ada di model. Pada Model **DataDiri**, ada **nama** dan **umur**, maka pada elemen input juga untuk parameter **name** disesuaikan. Oleh karena itu pada input_nama, name-nya adalah nama. Untuk input_umur, name-nya adalah umur.
- Sekarang, pada form, tambahkan action dan method. Untuk action, isinya adalah URL yang akan diakses apabila kamu klik button SUBMIT. Untuk method, karena

```
<form action="/post-data" method="post">
    <label for="input_nama">Nama : </label>
    <input id="input_nama" type="text" name="nama" />
    <label for="input_umur">Umur : </label>
    <input id="input_umur" type="number" name="umur" />
    <button type="submit">SUBMIT</button>
</form>
```

ini form (ngepost data), maka kita pilih method POST. Misal, untuk action saya mau /post-data. Maka akan jadi seperti ini :

- Dengan begini, apabila pengguna klik tombol SUBMIT, maka akan di-redirect ke URL /post-data, dimana URL /post-data akan kita handle di DataDiriController setelah ini.
- Pada DataDiriController, buatlah seperti ini

```
@PostMapping(value = "/post-data")
public ModelAndView post_data(@ModelAttribute("data_diri") DataDiri dataDiri){
    ModelAndView mav = new ModelAndView();
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    dataDiriImpl.InsertData(dataDiri);
    mav.setViewName("redirect:/form");
    return mav;
}
```

- Annotation @PostMapping menandakan bahwa URL yang akan di-handle (yaitu /post-data) adalah URL yang bersifat **POST** (karena di form.html juga kita sudah set ke method=”post”).
- Disini yang berbeda dari sebelumnya adalah adanya annotation berupa **@ModelAttribute**. Annotation ini berguna untuk meminta data berupa model dari view. Jadi di view form.html tadi kan ada form yang isinya input berupa nama dan umur, nama dan umur tersebut kan termasuk dalam model bernama DataDiri. Berarti model yang di-pass dari form tersebut kan otomatis adalah model DataDiri. Makannya di @ModelAttribute yang kita minta adalah model DataDiri juga.
- dataDiriImpl.InsertData(dataDiri) adalah invoke method InsertData dengan parameternya dataDiri (isinya adalah nama dan umur yang sudah diinputkan di form tadi).
- Model dataDiri yang di-pass ke dataDiriImpl.InsertData itu nanti akan diambil nama dan umur nya untuk di-insert ke table data_diri. Mengambil value nama dengan cara dataDiri.getNama() dan mengambil value umur dengan cara dataDiri.getUmur() (lihat pada class DataDiriImpl).
- Sekarang di **RUN**, lalu akses ke /form. Isi form dengan data yang dikehendaki lalu klik SUBMIT. Setelah itu cek saja untuk data yang ada di table data_diri melalui pgAdmin, maka akan terlihat data yang baru saja diinputkan masuk ke database.

MODUL 4

MENAMPILKAN SEMUA DATA DARI DATABASE

A. Membuat class DataDiriMapper

- Buat class bernama DataDiriMapper yang *implements* **RowMapper<DataDiri>**. Jika diberikan 2 pilihan RowMapper untuk di-import, pilih RowMapper yang dari **org.springframework.jdbc.core.**
- Apa sih guna class ini? Class ini gunanya untuk memetakan (mapping) value yang diambil dari database ke web app kita, untuk nantinya ditampilkan ke view.
- Code-mu nanti akan bergaris bawah merah, klik pada code yg bergaris bawah merah, lalu klik alt+enter dan pilih override methods, maka akan muncul method bernama mapRow.
- Code DataDiriMapper

```
public class DataDiriMapper implements RowMapper<DataDiri> {
    @Override
    public DataDiri mapRow(ResultSet resultSet, int i) throws SQLException {
        DataDiri dataDiri = new DataDiri();
        dataDiri.setId(resultSet.getInt("id"));
        dataDiri.setNama(resultSet.getString("nama"));
        dataDiri.setUmur(resultSet.getInt("umur"));
        return dataDiri;
    }
}
```

- Pada code di atas, kita membuat obyek dari model DataDiri. Kita ambil setiap data dari kolom yang ada, yaitu **id**, **nama**, dan **umur**. Cara mengambil setiap data tersebut adalah dengan resultSet.getInt (untuk id dan umur karena tipe data-nya adalah int) dan resultSet.getString (untuk nama karena tipe data-nya adalah String). Parameter untuk setiap resultSet itu adalah nama kolom yang ada di table data_diri (id, nama, dan umur. Harus sesuai ya nama kolomnya, jangan salah). Lalu, resultSet tersebut kita set ke model dataDiri. Contohnya adalah dataDiri.setId, dataDiri.setNama, dataDiri.setUmur.

B. Membuat query select *

- Kembali lagi pada class DataDiriImpl, pada method GetAllData(). Kita buat query untuk mengambil semua datanya. Seperti berikut

```
@Override
public List<DataDiri> GetAllData() {
    String query = "SELECT * FROM data_diri";
    List<DataDiri> dataDiriList = new ArrayList<>();
    dataDiriList = jdbcTemplate.query(query, new DataDiriMapper());
    return dataDiriList;
}
```

- Query nya adalah query SQL biasa, yaitu SELECT * FROM [nama tabel].
- Disini kita membuat **List** dengan tipe **DataDiri** bernama dataDiriList.
- Kita eksekusi query kita dengan jdbcTemplate.query(query, new DataDiriMapper()); Maksud kode disamping adalah untuk mengeksekusi query, lalu hasil dari query (yaitu data dari kolom id, kolom nama, dan kolom umur) kita mapping (dipetakan) ke DataDiriMapper. Value nya kita assign ke dataDiriList.

C. Handling URL di DataDiriController

- Kita ingin melihat data-data yang diambil dari database itu jika mengakses URL /data, maka di DataDiriController kita tambah code seperti ini

```
@GetMapping(value = "/data")
public ModelAndView data(){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("show-data");
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    List<DataDiri> dataDiriList = new ArrayList<>();
    dataDiriList = dataDiriImpl.GetAllData();
    mav.addObject("dataDiriList", dataDiriList);
    return mav;
}
```

- Pada code di atas, kita meng-invoke method **GetAllData** yang ada di class DataDiriImpl, lalu kita assign ke dataDiriList. Setelah itu, kita ***lempar*** dataDiriList ke view melalui **mav.addObject**, kita lempar dataDiriList dengan nama dataDiriList (bebas mau dinamain apa, cuma kalo aku biasanya namanya sama dengan nama variabel yang dilempar).

D. show-data.html

- Kita akan menampilkan data yang diambil dari database tadi di show-data.html. Maka buatlah show-data.html, isi kodenya adalah sbb :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>SHOW DATA</title>
</head>
<body>
    <h2>SHOW ALL DATA</h2>
    <table>
        <thead>
            <th>ID</th>
            <th>NAMA</th>
            <th>UMUR</th>
        </thead>
        <tbody>
            <th:block th:each="data_diri : ${dataDiriList}">
                <tr>
                    <td th:text="${data_diri.id}" />
                    <td th:text="${data_diri.nama}"/>
                    <td th:text="${data_diri.umur}"/>
                </tr>
            </th:block>
        </tbody>
    </table>
</body>
</html>
```

- Pada code di atas, kita membuat table, dengan <thead> nya adalah **ID, NAMA, UMUR**. Lalu di dalam <tbody>, kita menggunakan fungsi yang disediakan oleh thymeleaf yaitu **th:block** dan **th:each**. Kedua fungsi tersebut ekuivalen dengan fungsi **foreach** pada Bahasa pemrograman pada umumnya.

- Jadi, th:each=”data_diri : ${dataDiriList}” artinya adalah, untuk setiap (each) data yang ada di dalam dataDiriList (dataDiriList adalah object yang kita lempar dari DataDiriController tadi), kita sebutkan sebagai data_diri. Maka nanti untuk mengambil value-value yang ada dalam dataDiriList, kita sudah tidak menggunakan dataDiriList lagi, melainkan menggunakan data_diri.
- Di dalam **th:block**, adalah elemen <tr>, di dalam elemen <tr> adalah elemen <td>, jika biasanya untuk memberikan value pada <td> adalah seperti <td>Axell</td>, kali ini berbeda, karena kita ingin meng-assign nilai <td> dengan nilai yang dilempar dari controller, maka kita assign nya dengan cara <td th:text=”${data_diri.id}” /> (**ingat bahwa pada thymeleaf, diakhiri dengan />**). Di dalam th:text adalah ${data_diri.id}, artinya adalah memanggil value id, ${data_diri.nama}, artinya memanggil value nama, ${data_diri.umur} artinya memanggil value umur.
- Jika sudah, **RUN** aplikasi dan akses /data, maka akan keluar tabel dengan data yang muncul adalah data yang diambil dari database

MODUL 5

Mengambil id, nama, dan umur sebuah record di database berdasarkan ID-nya

A. Buat query pada class DataDiriImpl method GetOneData

- Buatlah query seperti ini

```
@Override
public List<DataDiri> GetOneData(int id) {
    List<DataDiri> dataDiriList = new ArrayList<>();
    String query = "SELECT * FROM data_diri WHERE id=?";
    dataDiriList = jdbcTemplate.query(query, new Object[] {id}, new
DataDiriMapper());
    return dataDiriList;
}
```

- Pada code di atas, bagian query adalah untuk mengambil column nama dan umur dari tabel data_diri, dimana ID=?, kita assign “?” sesuai dengan ID yang dikehendaki. Assign value ID nya adalah pada saat jdbcTemplate.query(query, new Object[] {id}, new DataDiriMapper()). Id adalah parameter dari method ini. Kita akan bahas lebih lanjut nanti. Intinya untuk saat ini adalah querying untuk database-nya dulu.

B. Sedikit editing pada show-data.html

- Kita akan membuat tampilan show-data.html ada text **EDIT**-nya menjadi seperti ini.

SHOW ALL DATA

ID	NAMA	UMUR	EDIT
1	Axell	20	EDIT
2	Pinto	20	EDIT
3	Bless	19	EDIT

- Maka ada sedikit update di show-data.html seperti ini :

```
<body>
    <h2>SHOW ALL DATA</h2>
    <table>
        <thead>
            <th>ID</th>
            <th>NAMA</th>
            <th>UMUR</th>
            <th>EDIT</th>
        </thead>
        <tbody>
            <th:block th:each="data_diri : ${dataDiriList}">
                <tr>
                    <td th:text="${data_diri.id}" />
                    <td th:text="${data_diri.nama}"/>
                    <td th:text="${data_diri.umur}"/>
                    <td><a th:href="@{/details/{id}(id=${data_diri.id})}">EDIT</a></td>
                </tr>
            </th:block>
        </tbody>
    </table>
</body>
```

- Perubahan code-nya ditandai dengan highlight kuning.
- Yang akan aku tekankan disini adalah bagian <td> yang di highlight kuning. Disini intinya kita memasukkan elemen <a> ke dalam elemen <td>, elemen <a> itu adalah teks yang warnanya biru dan bisa di klik (berfungsi sebagai hyperlink). Tulisan dari <a> ini adalah **EDIT.** Lalu, jika kita klik EDIT, akan pergi kemana? Itu diatur di bagian **th:href**. Jangan bingung dengan syntax-nya ya, memang butuh sedikit pembiasaan untuk terbiasa dengan syntax, aku sendiri gak selalu ingat syntax-nya kok. Intinya di dalam th:href itu adalah untuk mengatur URL-nya bila tulisan EDIT di klik. URL nya adalah /details/{id}. Dimana nilai id adalah diambil dari ${data_diri.id}. Jadi, misal menurut data ku yang aku screenshot di atas, jika aku klik EDIT pada nama **Axell**, maka URL nya akan /details/1, karena ID untuk Axell adalah **1**. Tapi, jika di **klik sekarang akan error**, kenapa? **Karena** kita belum melakukan routing URL pada DataDiriController. Oleh karena itu, kita pindah ke DataDiriController sekarang.

C. DataDiriController

- Kita tambahkan code untuk handling url /details/{id}

```
@GetMapping(value = "/details/{id}")
public ModelAndView details(@PathVariable("id") int id){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("details");
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    List<DataDiri> dataDiriList = new ArrayList<>();
    dataDiriList = dataDiriImpl.GetOneData(id);
    mav.addObject("dataDiriList", dataDiriList);
    return mav;
}
```

- Pada code di atas, kita handle untuk URL /details/{id}. {id} disitu adalah nilai id dari data yang di klik di view. Ada code **@PathVariable(“id”) int id**. Maksud code

itu adalah kita mengambil nilai {id} yang ada di URL, lalu kita pindahkan value nya ke variabel int id.

- Selanjutnya, seperti sudah dijelaskan sebelumnya sih, sama-sama aja. Yang beda disini adalah kita meng-invoke method **GetOneData** dengan parameter **id** (yaitu id yang kita dapatkan dari URL /details/{id} → dari @PathVariable). Lalu hasil dari invoke method itu kita assign value nya ke dataDiriList, lalu dataDiriList kita lempar ke view dengan mav.addObject.
- Karena kita mav.setViewName adalah details, sedangkan details.html belum ada, maka kita buat dulu details.html

D. details.html

- Code nya adalah seperti berikut

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>DETAIL</title>
</head>
<body>
    <h2>DETAIL DATA</h2>
    <table>
        <thead>
        <th>ID</th>
        <th>NAMA</th>
        <th>UMUR</th>
        </thead>
        <tbody>
        <th:block th:each="data_diri : ${dataDiriList}">
            <tr>
                <td th:text="${data_diri.id}" />
                <td th:text="${data_diri.nama}"/>
                <td th:text="${data_diri.umur}"/>
            </tr>
        </th:block>
        </tbody>
    </table>
</body>
</html>
```

- Code di atas tidak ada yang baru, sama seperti code untuk di show-data.html (sudah dijelaskan sebelumnya).
- Sekarang coba **RUN** aplikasi, lalu buka /data, lalu klik **EDIT** pada suatu data, maka akan muncul detail nya di page baru.

MODUL 6

Mengambil satu column saja (column nama) berdasarkan ID-nya.

Jadi, pada modul sebelumnya kan kita mengambil nilai dari database dalam bentuk sebuah LIST karena data yang diambil ada banyak yaitu id, nama, dan umur. Bagaimana jika kita cuma mau ambil satu column saja, misal column nama saja berdasarkan ID yang ditentukan. Akan kita bahas di modul ini.

A. Query di DataDiriImpl

- Kita buat query pada method GetNameById.

```
@Override
public String GetNameById(int id) {
    String query = "SELECT nama FROM data_diri WHERE id=?";
    String nama;
    nama = jdbcTemplate.queryForObject(query, new Object[] {id}, String.class);
    return nama;
}
```

- Intinya adalah query untuk mengambil column nama dari table data_diri dimana nilai id=?. Yang berbeda dari sebelum2nya adalah sebelumnya kita assign value ke dataDiriList (bentuk LIST), sekarang kita assign ke nama (String). Lalu biasanya jdbcTemplate.query, sekarang jdbcTemplate.queryForObject. Parameternya tetap sama, yaitu query, new Object[] {id} untuk assign ke “?” nilai ID nya, lalu yang ketiga biasanya adalah new DataDiriMapper(), sekarang kita pakai String.class, karena kita sekarang assign hasil dari query ke String (bukan LIST lagi). Setelah itu return variable nama.

B. Perubahan sedikit pada show-data.html

- Kita akan mengubah sedikit show-data.html menjadi seperti berikut

```
<body>
    <h2>SHOW ALL DATA</h2>
    <table>
        <thead>
            <th>ID</th>
            <th>NAMA</th>
            <th>UMUR</th>
            <th>EDIT</th>
            <th>LIHAT NAMA</th>
        </thead>
        <tbody>
            <th:block th:each="data_diri : ${dataDiriList}">
                <tr>
                    <td th:text="${data_diri.id}" />
                    <td th:text="${data_diri.nama}"/>
                    <td th:text="${data_diri.umur}"/>
                    <td><a th:href="@{/details/{id}(id=${data_diri.id})}">EDIT</a></td>
                    <td><a th:href="@{/nama/{id}(id=${data_diri.id})}">LIHAT NAMA</a></td>
                </tr>
            </th:block>
        </tbody>
    </table>
</body>
```

- Code yang berbeda adalah yang di highlight kuning.
- Gunanya memunculkan teks elemen <a> (hyperlink) berupa **LIHAT NAMA** yang apabila nanti di klik akan redirect ke URL /nama/{id} untuk melihat value **NAMA** di page yang baru.
- Sekarang kita handling URL /nama/{id} di DataDiriController

C. DataDiriController

- Tambahkan code ini :

```
@GetMapping(value = "/nama/{id}")
public ModelAndView nama(@PathVariable("id") int id){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("nama");
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    String nama = dataDiriImpl.GetNameById(id);
    mav.addObject("nama", nama);
    return mav;
}
```

- Code di atas kurang lebih sama dengan code pada DataDiriController pada modul sebelumnya. Yang berbeda adalah disini kita meng-invoke method **GetNameById** dengan parameter id. Lalu hasil dari invoke method tersebut kita simpan nilainya ke String nama. Lalu variabel nama itu kita lempar ke view melalui mav.addObject. View yang kita gunakan disini adalah nama.html, namun karena nama.html belum ada, maka kita buat dulu nama.html

D. nama.html

- Buat nama.html dan isi code nya adalah sebagai berikut

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8" />
    <title>NAMA</title>
</head>
<body>
    <h2>DATA NAMA BERDASARKAN ID USER</h2>
    <h5 th:text="${nama}"/>
</body>
</html>
```

- Kita assign value dari variabel nama yang dilempar dari DataDiriController tadi di bagian elemen <h5>, terlihat bahwa ada **th:text=”${nama}”**. Yaitu untuk menampilkan value dari variabel nama ke elemen <h5>.
- Sekarang coba **RUN,** lalu akses /data dan klik di **LIHAT NAMA** dari salah satu data yang ada.

MODUL 7

UPDATE data yang ada di database

A. Query di DataDiriImpl pada method Update

- Buat query pada method Update di DataDiriImpl seperti berikut

```
@Override
public void Update(DataDiri dataDiri) {
    String query = "UPDATE data_diri SET nama=?, umur=? WHERE id=?";
    jdbcTemplate.update(query, new Object[] {dataDiri.getNama(), dataDiri.getUmur(),
dataDiri.getId()});
}
```

- Query di atas adalah query untuk UPDATE pada SQL pada umumnya. Kita ingin mengupdate column nama dan umur dari data yang memiliki id=?. Pada dasarnya hampir sama dengan insert, hanya saja disini berbeda query nya dan disini menggunakan condition berupa **WHERE id=?**

B. Sedikit perubahan pada details.html

- Jika awalnya pada page details.html hanyalah menampilkan data saja, sekarang akan kita ganti menjadi form, sehingga kita bisa update melalui form ini nantinya.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>DETAIL</title>
</head>
<body>
    <h2>DETAIL DATA</h2>
    <form action="/update" method="post">
        <th:block th:each="data_diri : ${dataDiriList}">
            <input type="hidden" th:value="${data_diri.id}" name=”id” />
            <label for="input_nama">Nama : </label>
            <input id="input_nama" type="text" th:value="${data_diri.nama}" name=”nama” />
            <label for="input_umur">Umur : </label>
            <input id="input_umur" type="number" th:value="${data_diri.umur}" name=”umur” />
        </th:block>
        <button type="submit">UPDATE</button>
    </form>
</body>
</html>
```

- Code <table> dihapus semua aja, diganti dengan yang di highlight kuning.
- Pada <form> di atas, action nya ketika button UPDATE di klik adalah redirect ke URL /update dengan method post.
- Pada code di atas, ada yang type nya hidden, yaitu untuk nilai ID, kenapa nilai ID kita hidden? Karena nilai ID tidak perlu diganti, jadi yang diganti hanya nama dan umur saja, makannya nilai ID di-hidden supaya tidak bisa diganti.
- Ada fungsi thymeleaf berupa **th:value** yaitu untuk memberikan value pada elemen <input>. Sisanya adalah code yang *self explanatory*, bisa dipahami sendiri mungkin ya.
- Sekarang kita handle URL /action di DataDiriController

C. DataDiriController

- Buat code seperti ini

```
@PostMapping(value = "/update")
public ModelAndView update(@ModelAttribute("data_diri") DataDiri dataDiri){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("redirect:/data");
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    dataDiriImpl.Update(dataDiri);
    return mav;
}
```

- Aku rasa code di atas *self explanatory* juga ya, bisa dipahami sendiri karena syntax-syntax dan logic-nya kurang lebih sama dengan method untuk insert data.
- Sekarang **RUN** aplikasi, dan pada page /data, klik **EDIT** pada salah satu data, lalu ganti value yang ada di input **nama** atau **umur** dan klik **UPDATE**, maka data akan berubah value-nya.

MODUL 8 (FINAL)

Delete record di database

A. Query di DataDiriImpl

- Kita buat query di method Delete pada class DataDiriImpl

```
@Override
public void Delete(int id) {
    String query = "DELETE FROM data_diri WHERE id=?";
    jdbcTemplate.update(query, new Object[] {id});
}
```

B. Perubahan sedikit pada show-data.html

- Kita akan menambahkan kolom baru pada table yaitu kolom berisikan elemen <a> bertuliskan DELETE untuk menghapus data.

```
<body>
    <h2>SHOW ALL DATA</h2>
    <table>
        <thead>
            <th>ID</th>
            <th>NAMA</th>
            <th>UMUR</th>
            <th>EDIT</th>
            <th>LIHAT NAMA</th>
            <th>DELETE</th>
        </thead>
        <tbody>
            <th:block th:each="data_diri : ${dataDiriList}">
                <tr>
                    <td th:text="${data_diri.id}" />
                    <td th:text="${data_diri.nama}"/>
                    <td th:text="${data_diri.umur}"/>
                    <td><a th:href="@{/details/{id}(id=${data_diri.id})}">EDIT</a></td>
                    <td><a th:href="@{/nama/{id}(id=${data_diri.id})}">LIHAT NAMA</a></td>
                    <td><a th:href="@{/delete/{id}(id=${data_diri.id})}">DELETE</a></td>
                </tr>
            </th:block>
        </tbody>
    </table>
</body>
```

- Code di atas sudah pernah dijelaskan pada modul2 sebelumnya, jadi langsung saja ke DataDiriController, untuk menghandle URL /delete/{id}

C. DataDiriController

```
@GetMapping(value = "/delete/{id}")
public ModelAndView delete(@PathVariable("id") int id){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("redirect:/data");
    DataDiriImpl dataDiriImpl = new DataDiriImpl();
    dataDiriImpl.Delete(id);
    return mav;
}
```

- Code di atas juga persis dengan code GetNameById, jadi aku kira bisa dipahami sendiri juga ya.
- Sekarang **RUN** aplikasinya dan akses /data, lalu klik DELETE pada salah satu data maka data akan terhapus.

ASSIGNMENT

Tugas terkait materi-materi yang udah dijelaskan pada 8 modul sebelumnya

A. Deskripsi Tugas

Buatlah sebuah web yang bisa menyimpan data ke database POSTGRESQL. Data yang disimpan adalah :

ID, Nama mata kuliah, nama dosen

Operasi yang bisa dilakukan adalah insert, menampilkan semua data, menampilkan nama dosen saja, mengupdate data, menghapus data