Nama : Windiany Lestari Sitorus

Kelas : Ekstensi

NPM : 1606955063

Latihan Menambahkan Delete

1. Menambahkan link Delete Data pada viewall.html di dalam div th:each.

```
<a th:href="'/student/delete/' + ${student.npm}" > Delete Data</a><br/>
```

2. Menambahkan method deleteStudent yang menerima parameter npm di class StudentMapper. Disini juga akan menggunakan annotation helper dari Batis sebagai penghubung ke SQL, diikuti dengan SQL query untuk delete.

```
@Delete("DELETE FROM student WHERE npm = #{npm}")
void deleteStudent (@Param("npm") String npm);
```

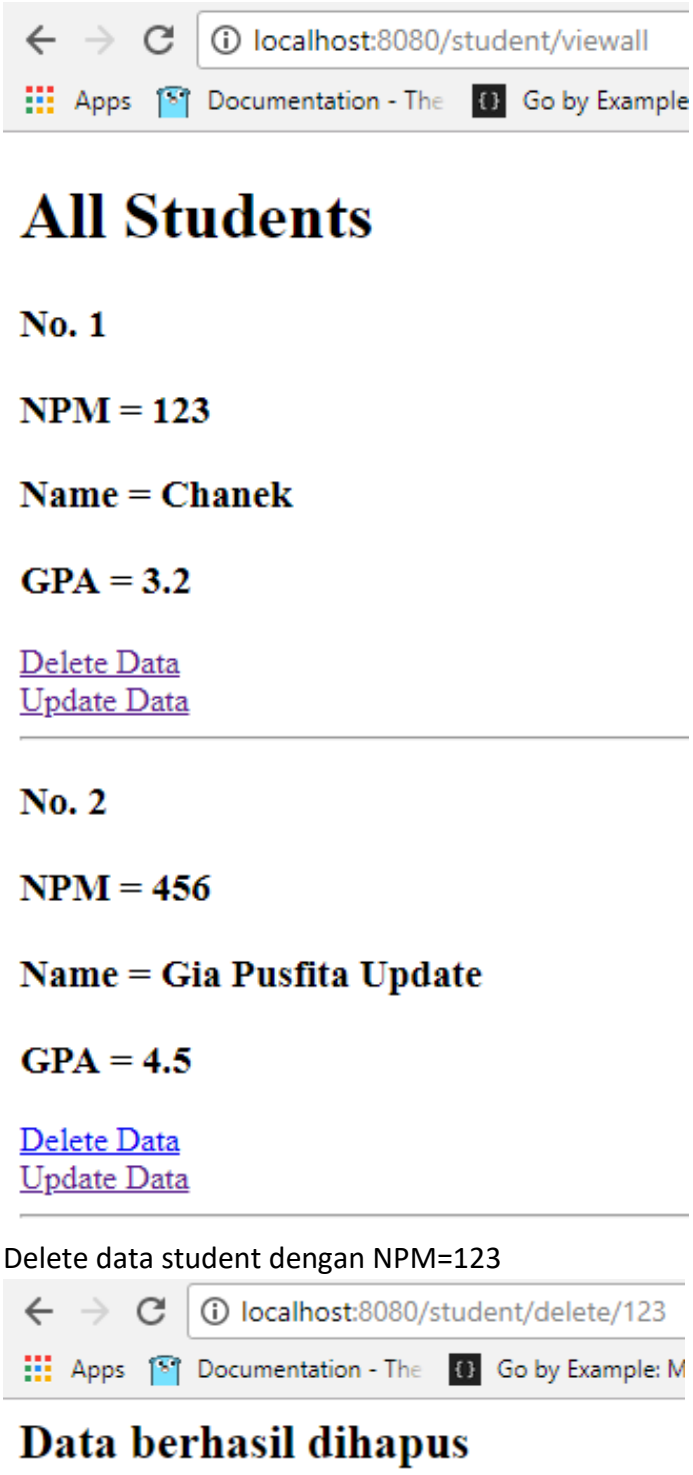
3. Melengkapi method deleteStudent di class StudentServiceDatabase, dan ditambahkan dengan log.info. Pada method ini akan dipanggil deleteStudent yang berada di StudentMapper yang berisikan sql query sebelumnya.

```
@Override
public void deleteStudent (String npm)
{
    log.info("student "+npm+" deleted");
    studentMapper.deleteStudent(npm);
}
```

4. Melengkapi method delete yang terdapat pada StudentController dengan validasi. Pada method ini akan dilakukan select student dengan npm yang sesuai dengan parameter. Kemudian apabila data student tersebut ada, maka akan dilakukan proses delete. Dan apabila tidak ada, akan dialihkan ke halaman not-found.

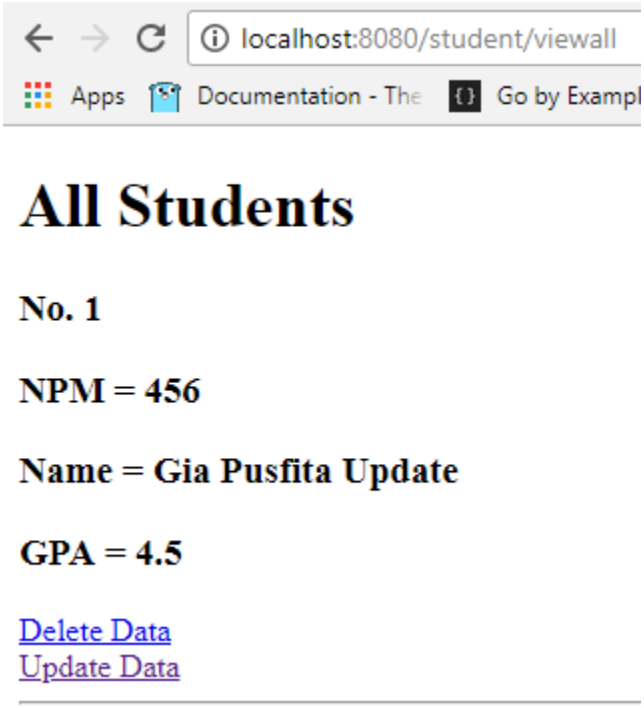
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);
    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

5. Maka setelah dijalankan
   Viewall

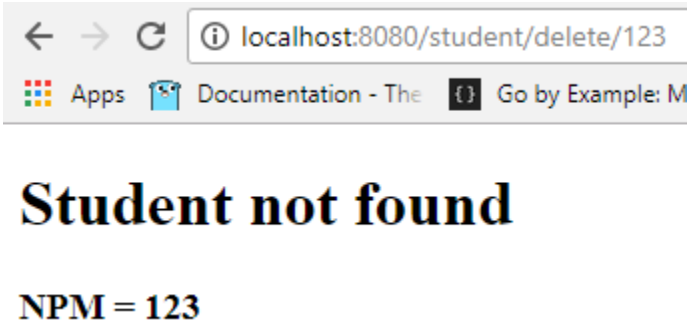


   Delete data student dengan NPM=123

Viewall untuk memastikan data yang telah dihapus tidak muncul lagi



Delete data student dengan NPM=123 untuk kedua kalinya, maka akan muncul error message



Latihan Menambahkan Update

1. Menambahkan method updateStudent pada class StudentMapper. Parameternya berupa StudentModel student. Dan dilengkapi dengan annotation helper Update diikuti dengan SQL query update.

```
@Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}")
void updateStudent (StudentModel student);
```

2. Menambahkan method updateStudent pada interface StudentService.

```
void updateStudent (StudentModel student);
```

3. Menambahkan implementasi method updateStudent di class StudentServiceDatabase. Dan penambahan log.info. Pada method ini akan dipanggil updateStudent yang berada di StudentMapper yang berisikan sql query sebelumnya.

```
@Override
public void updateStudent (StudentModel student)
{
    log.info("update student");
    studentMapper.updateStudent(student);
}
```

4. Menambahkan link Update Data di viewall.html.

```
<a th:href="'/student/update/' + ${student.npm}" > Update Data</a><br/>
```

5. Menambahkan form-update.html. Pada form ini, action form nya akan menjadi /student/update/submit dan menggunakan method POST. Kemudian input npm akan hanya dapat readonly agar tidak dapat diedit. Dan untuk npm, name dang pa akan ditampilkan valuenya.

```
<form action="/student/update/submit" method="post">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}"/>
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}"/>
    </div>
    <div>
        <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}"/>
    </div>
    <div>
        <button type="submit" name="action" value="save">Update</button>
    </div>
</form>
```

6. Menambahkan success-update.html.

```
<html>
    <head>
        <title>Add</title>
    </head>
    <body>
        <h2>Data berhasil diupdate</h2>
    </body>
</html>
```

6. Menambahkan method update di class StudentController dengan RequestMapping /student/update/{npm}. . Pada method ini akan dilakukan select student dengan npm yang sesuai dengan parameter. Kemudian apabila data student tersebut ada, maka akan dialihkan ke halaman form-update. Dan apabila tidak ada, akan dialihkan ke halaman not-found.

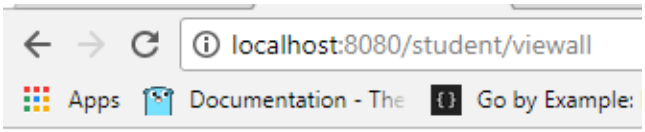
```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);
    if (student != null) {
        model.addAttribute ("student", student);
        return "form-update";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

7. Menambahkan updateSubmit di class StudentController. Method ini akan menangani bagian saat user akan submit form-update. Akan dipanggil method updateStudent yang telah di deklarasikan di servicer, kemudian akan dialihkan ke halaman success-update.

```
@RequestMapping(value="/student/update/submit", method= RequestMethod.POST)
public String updateSubmit (
        @RequestParam(value = "npm", required = false) String npm,
        @RequestParam(value = "name", required = false) String name,
        @RequestParam(value = "gpa", required = false) double gpa)
{
    StudentModel student = new StudentModel (npm, name, gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

8. Maka setelah dijalankan

Viewall



All Students

No. 1

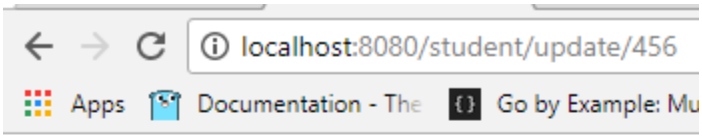
NPM = 456

Name = Gia Pusfita Update

GPA = 4.5

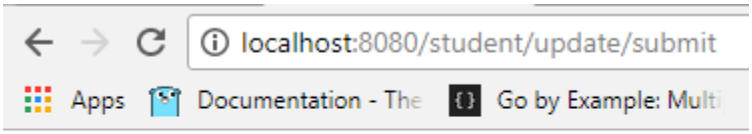
Delete Data
Update Data

Update data student



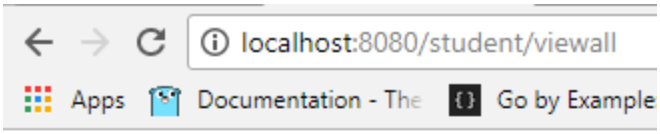
Update Student Data

NPM 456
Name Gia Pusfita Update Nama
GPA 1.5
Update



Data berhasil diupdate

Viewall untuk memastikan data yang telah diupdate ditampilkan



All Students

No. 1

NPM = 456

Name = Gia Pusfita Update Nama

GPA = 1.5

Delete Data
Update Data

Latihan Menggunakan Object Sebagai Parameter

1. Menambahkan th:object=”${student}” pada tag <form> di view.

```
<form th:object="${student}" action="/student/update/submit" method="post">
```

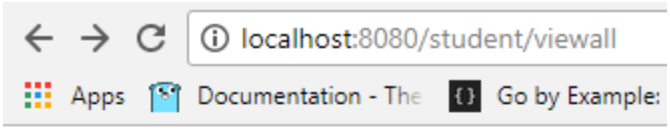
2. Menambahkan th:field=”*{[nama_field]}” pada setiap input.

```
<div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="${student.npm}"/>
</div>
<div>
    <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="${student.name}"/>
</div>
<div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="${student.gpa}"/>
</div>
```

3. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel. Pada method ini parameternya adalah berupa objek sehingga memudahkan apabila terlalu banyak parameter dalam satu method. Dan sebagai pengganti RequestParam akan digunakan ModelAttribute di parameter.

```
@RequestMapping(value="/student/update/submit", method= RequestMethod.POST)
public String updateSubmit (@ModelAttribute StudentModel student
        /* @RequestParam(value = "npm", required = false) String npm,
         @RequestParam(value = "name", required = false) String name,
         @RequestParam(value = "gpa", required = false) double gpa*/)
{
    //StudentModel student = new StudentModel (npm, name, gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

4. Tes aplikasi
   Viewall



All Students

No. 1

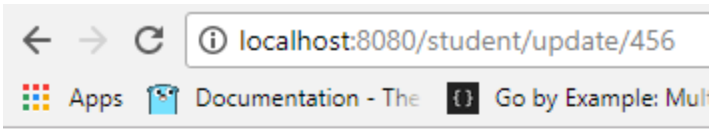
NPM = 456

Name = Gia Pusfita Update Nama

GPA = 1.5

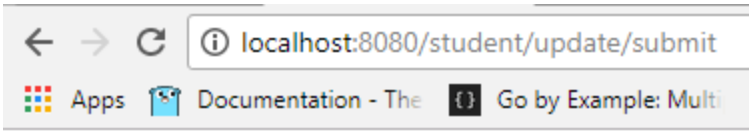
Delete Data
Update Data

Update data student



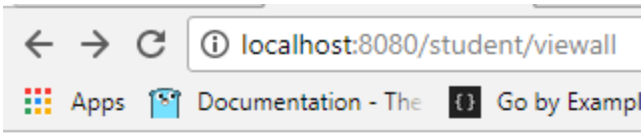
Update Student Data

NPM 456
Name Windiany Lestari Sitorus
GPA 4.0
Update



Data berhasil diupdate

Viewall untuk memastikan data yang telah diupdate ditampilkan



All Students

No. 1

NPM = 456

Name = Windiany Lestari Sitorus

GPA = 4.0

Delete Data
Update Data

Pertanyaan

1. Jika menggunakan object sebagai parameter dengan metode POST, maka validasi input akan dilakukan di bagian view. Hal ini dapat dilakukan dengan menggunakan theme thymeleaf, seperti th:required untuk validasi suatu field required atau tidak.
   Akan tetapi apabila pada kasus tidak bisa menggunakan validasi di view seperti pertanyaa, maka dapat dilakukan validasi menggunakan Hibernate validator yang menyediakan validasi untuk input sebagai objek. Dapat digunakan dengan menambahkan anotasi @valid.
2. Form submit biasanya menggunakan method POST karena akan mengirimkan data langsung ke action untuk ditampung tanpa menggunakan URL. Sementara method GET akan menampilkan data atau nilai pada URL kemudian ditampung di action. Sehingga akan lebih aman apabila menggunakan POST, karena data yang kita kirimkan tidak ditampilkan di URL. Dan apabila menggunakan method POST data yang dikirim tidak terbatas, sementara method GET tidak boleh lebih dari 2047 karakter.
3. Mungkin, contohnya adalah sebagai berikut:
   @RequestMapping(value=”/test”, method={RequestMethod.GET, RequestMethod.POST})
   Akan tetapi bukan merupakan implementasi yang baik, satu method baiknya menerima satu jenis request method saja.

Ringkasan Materi Tutorial 4 APAP

1. Pada tutorial 4 ini saya mempelajari menggunakan library dari Lombok dan MyBatis. Keduanya akan menjadi helper annotation. Lombok akan menyediakan logging yang dapat memudahkan dalam debugging dan MyBatis akan membantu melakukan koneksi ke database dan generate query dari helper annotation.

```
c.e.service.StudentServiceDatabase       : select all students
c.e.service.StudentServiceDatabase       : select student with npm 123
c.e.service.StudentServiceDatabase       : update student
c.example.controller.StudentController   : test
c.e.service.StudentServiceDatabase       : select all students
c.e.service.StudentServiceDatabase       : select student with npm 123
c.e.service.StudentServiceDatabase       : student 123 deleted
c.example.controller.StudentController   : test
c.e.service.StudentServiceDatabase       : select all students
c.example.controller.StudentController   : test
c.e.service.StudentServiceDatabase       : select all students
c.e.service.StudentServiceDatabase       : select student with npm 456
c.e.service.StudentServiceDatabase       : update student
```

2. Menggunakan database dan konfigurasi database dapat diatur di application.properties.
3. Pada tutorial ini juga saya mempelajari method GET dan POST, pada saat menggunakan GET data yang kita kirim akan ditampilkan di URL, sementara POST datanya tidak akan ditampilkan di URL.
4. Belajar menggunakan object sebagai parameter, sehingga saat menggunakan parameter yang terlalu banyak dan merepotkan, kita dapat menggunakan objek di parameter.