RINGKASAN TUTORIAL 4

I. Latihan DELETE

- Pada file template viewall.html tambahkan click-link untuk menghapus data student

```
<div th:each="student,iterationStatus: ${students}">
        <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
        <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
        <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
        <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
        <a th:href="'/student/delete/' + ${student.npm}" > Delete Data </a><br/>
```

- Pada StudentMapper tambahkan anotasi delete yang berisi query SQL yang memerlukan parameter npm untuk menghapus data berdasarkan npm.

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

- Menambah interface deleteStudent pada StudentService.

```
void deleteStudent (String npm);
```

- Pada StudentServiceDatabase tambahkan method deleteStudent, lalu panggil method deleteStudent yang ada pada studentMapper.
  Terdapat log.info yang akan memberikan informasi terkait student yang di delete pada console.

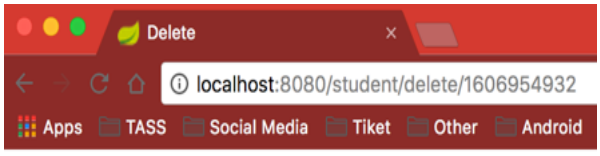
```
@Override
public void deleteStudent (String npm)
{
    log.info("student " + npm + "deleted");
    studentMapper.deleteStudent(npm);
}
```

- Pada studentController tambahkan method delete dengan menambahkan validasi untuk parameter npm,

```
@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 {
            return "not-found";
        }


}
```

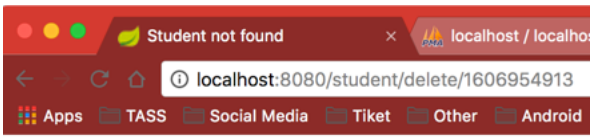
  Di dalam kode tersebut, terdapat validasi dimana jika npm terdapat pada model student dan student tidak null maka data akan dihapus berdasarkan npm yang kemudian dilempar ke halaman delete.html dan jika student bernilai null (npm tidak ditemukan) maka akan dilempar ke halaman not-found.html.

  - Jika npm tersedia.



Data berhasil dihapus

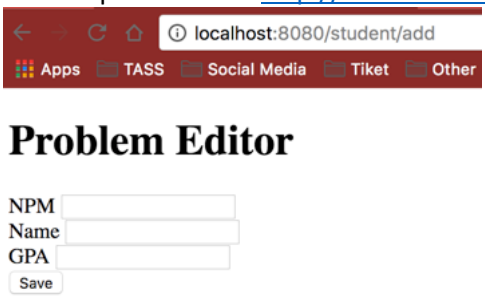
- Jika npm tidak tersedia



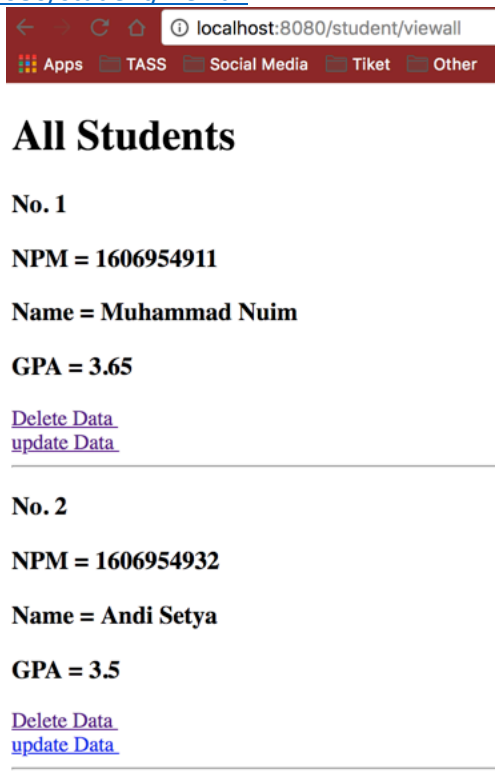
Student not found

NPM = 1606954913

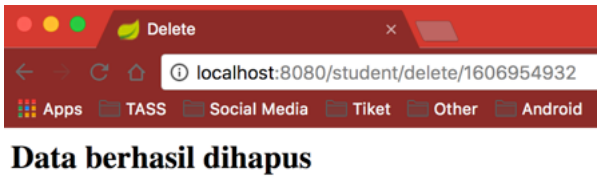
- Tutorial untuk delete student
  - Lakukan add data student pada alamat: http://localhost:8080/student/add



  - Lakukan view ke semua data yang ada di database: http://localhost:8080/student/viewall

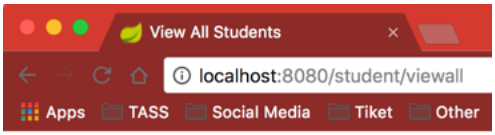


- Kemudian cobalah untuk menghapus salah satu data dengan menekan link Delete Data



Data berhasil dihapus

- Kemudian cek apakah data telah terhapus atau tidak dengan melihat pada viewall



All Students

No. 1

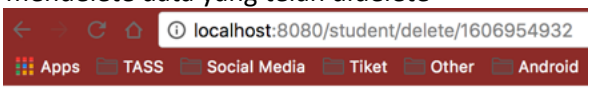
NPM = 1606954911

Name = Muhammad Nuim

GPA = 3.4

Delete Data
update Data

- Jika mencoba mendelete data yang telah didelete



Student not found

NPM = 1606954932

II. Latihan Update

- Pada file template viewall.html tambahkan click-link untuk meng-update data student

```
<div th:each="student,iterationStatus: ${students}">
        <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
        <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
        <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
        <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
        <a th:href="'/student/delete/' + ${student.npm}" > Delete Data </a><br/>
        <a th:href="'/student/update/' + ${student.npm}" > update Data </a><br/>
        <hr/>
    </div>
```

- Pada StudentMapper tambahkan anotasi update yang berisi query SQL update berdasarkan npm dan disimpan StudentModel

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

- Tambahkan method updateStudent pada interface StudentService.

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

- Pada StudentServiceDatabase tambahkan method updateStudent, lalu panggil method updateStudent yang ada pada studentMapper.
  Terdapat log.info yang akan memberikan informasi terkait student yang di update pada console.

```
@Override
public void updateStudent (StudentModel student)
{
    log.info("select student with npm: {}, name: {}, gpa: {} have been update", student.getNpm(), student.getName(), student.getGpa());
    studentMapper.updateStudent(student);
}
```

- Membuat file html untuk form update data.

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem Editor</h1>

    <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">Save</button>
        </div>
    </form>

</body>

</html>
```

- Membuat file html untuk tampilan sukses update

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

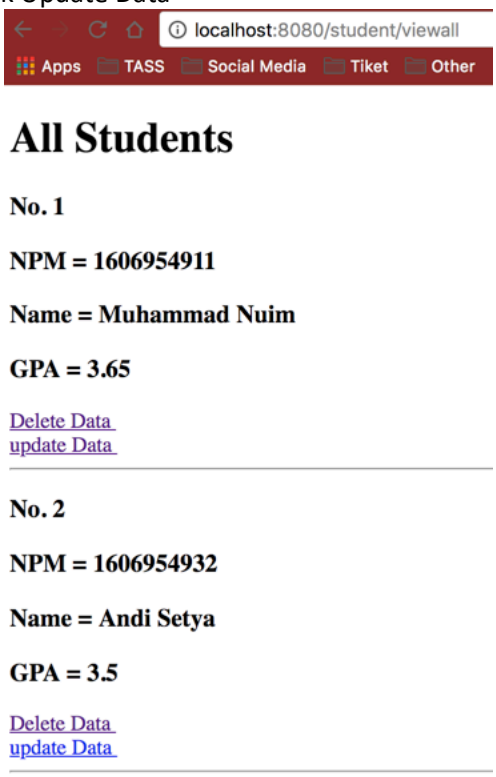
- Menambahkan method update pada StudentController dimana terdapat validasi untuk mengecek apakan npm ada pada data jika ada maka akan dilempar halaman form-update dan jika tidak maka akan dilempar pada file not-found.

```
@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";
        }

}
```

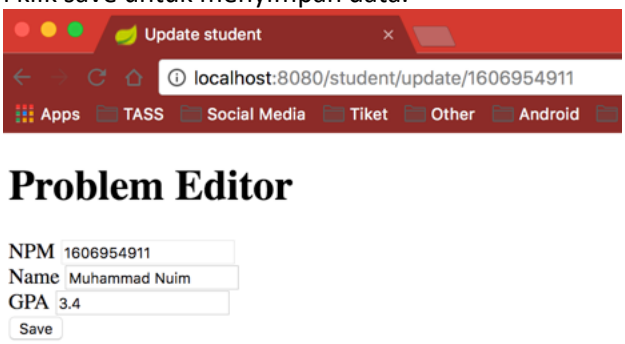
- Menambahkan method update pada StudentController, dimana terdapat parameter yang berfungsi untuk menerima data dari form-update, jika data berhasil diupdate pada database maka akan dilempar ke file success-update.

```
@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";
 }
```

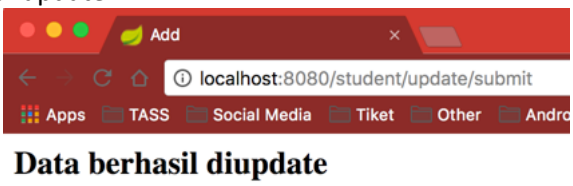
- Tutorial update data, pertama masuk ke halaman viewall untuk melihat semua data yang ada, kemudia klik Update Data



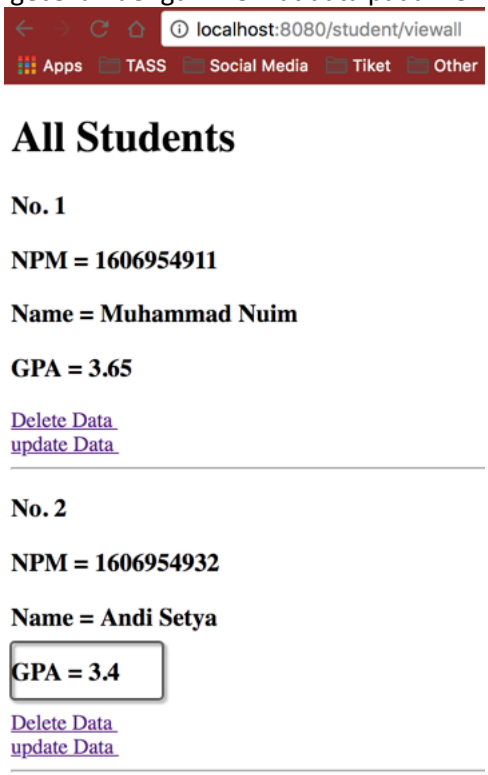
- Kemudian akan muncul data mahasiswa pada form update, dan dapat dilakukan update data, dan klik save untuk menyimpan data.



- Jika data berhasil di update



- Kemudian lakukan pengecekan dengan melihat data pada view all.



III. Latihan Mengganti Object sebagai Parameter

✓ Tambahkan method updateSubmit pada studentController.

```
@PostMapping(value="/student/update/submit")
public String updateSubmit (@ModelAttribute StudentModel student)
{

    studentDAO.updateStudent (student);

    return "success-update";
}
```

✓ Lakukan perubahan pada *code* form-update seperti gambar dibawah ini:
Terdapat *th:object* digunakan untuk mendeklarasikan sebuah object yang akan digunakan pada tutorial ini yaitu object student. Dan juga terdapat *th:field* yang digunakan untuk mendeklarasikan field yaitu npm, nama, gpa yang juga terdapat pada object student.

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

    <div>
        <button type="submit" name="action" value="save">Save</button>
    </div>
</form>
```

IV. Ringkasan Materi

Pada tutorial 4 ada beberapa hal yaitu membuat project yang terhubung dengan database, pengguanaan *library* Lombok yang berfungsi sebagai *helper annotation* pada project untuk penggunaan *library* ini dengan menambahkan anotasi @Slf4j diatas kelas yang ingin digunakan dan juga penggunaan MyBatis berfungsi sebagai penghubung dan untuk generate query dengan helper annotation. Dalam tutorial ini juga mempelajari bagaimana membuat object menjadi parameter.

V. Pertanyaan

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required seperti jika menggunakan RequestParam ? Apakah validasi diperlukan?

   Jawab :

   Pada model tersebut dapat ditambahkan standar *validation* seperti @valid pada parameter object kemudian ditambahkan Object bindingResult. Kemudian pada *body* akan ditambahkan validasi yang menggunakan code "bindingResult.hasErrors()".

```
@PostMapping(value="/student/update/submit")
//public String updateSubmit (@ModelAttribute StudentModel student)
public String updateSubmit (@Valid StudentModel student, BindingResult br)
{
    if (br.hasErrors()) {
        return "error-update";
    }else {
         studentDAO.updateStudent (student);

            return "success-update";
    }

}
```

2. Menurut anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form post dikirim menggunakan method berbeda?

   Jawab:

   POST digunakan karena lebih aman dibandingkan dengan GET, karena data dan variable parameter tidak terlihat. Dan juga POST tidak memiliki batas panjang karakter data.

   Cara Penanganannya dengan menambahkan method = RequestMethod.POST pada @RequestMapping

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?

   Jawab:

   Bisa untuk dijalankan, tidak bisa dijalankan secara bersamaan tetapi harus dijalankan secara bergantian.