Latihan Menambahkan Delete

1. Memodifikasi viewall.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

        <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 User</a>
```

Menambah text delete user yang dapat di klik

2. Menambahkan method deleteStudent di calss StudentMapper

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

Menambahkan anotasi delete beserta querynya lalu menambahkan method delete yang berisi parameter npm

3. Memodifikasi method deleteStudent di class StudentServiceDatabase

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

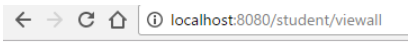
Methode deleteStudent menerina npm lalu memanggil interface method deleteStudent yang berada pada interface

4. Melengkapi method delete di class StudentController

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

Method delete menerima parameter berupa model dan npm . Berdasarkan npm dicari object sudent terlebih dahulu dengan method selectStudent . Setelah object ditemukan dipanggil method deleteStudent yang menerima parameter npm dan memanggil halaman delete ,bila object tidak ditemukan maka halaman not-found akan dipanggil

Hasil :



All Students

No. 1

NPM = 1

Name = naomiolga

GPA = 12.0

Delete User Update User

No. 2

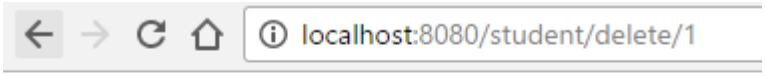
NPM = 123

Name = naomi

GPA = 4.0

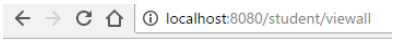
Delete User Update User

Tekan delete pada No 1



Data berhasil dihapus

Lihat Kembali All Student



All Students

No. 1

NPM = 123

Name = naomi

GPA = 4.0

Delete User Update User

No. 2

NPM = 124

Name = Tina

GPA = 4.0

Delete User Update User

2. Latihan Menambahkan Update

1. Tambahkan method **updateStudent** pada class **StudentMapper**

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

Menambahkan anatation update beserta querynya

2. Tambahkan method **updateStudent** pada interface **StudentService**

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

Menambahkan method updateStudent yang menerima parameter object student

3. Tambahkan implementasi method **updateStudent** pada class **StudentServiceDatabase.**

```
public void updateStudent (StudentModel student)
{
    studentMapper.updateStudent (student);
}
```

Methode updateStudent menerina object student lalu memanggil interface method updateStudent yang berada pada interface

4. Tambahkan link Update Data pada **viewall.html**

```
<a th:href="'/student/delete/'+${student.npm}">Delete User</a>
    <a th:href="'/student/update/'+${student.npm}">Update User</a>
<hr/>
```

Menambahkan text Update User yang bisa di klik

5. Copy view form-add.html menjadi **form-update.html**

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

<body>

    <h1 class="page-header">Update Student</h1>

    <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}"/>
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name"   th:field="*{name}" />
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa"   th:field="*{gpa}"/>
        </div>

        <div>
            <button type="submit" name="action" value="update">Update</button>
        </div>
    </form>
```

6. Copy view success-add.html menjadi **success-update.html**.

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

7. Tambahkan method **update** pada class **StudentController**

```
@RequestMapping("/student/update/{npm}")
public String update (Model model,@PathVariable(value = "npm") String npm)
{
     StudentModel student = studentDAO.selectStudent (npm);

            model.addAttribute ("student", student);

    return "form-update";
}
```

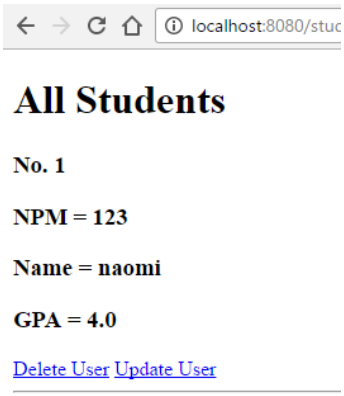
Method ini menerima parameter berupa model dan npm , seperti halnya method delete lebih dahulu dicari studentnya dengan method selectStudent sehingga objectnya bisa di parsing di halaman upate

8. Tambahkan method **updateSubmit** pada class **StudentController**

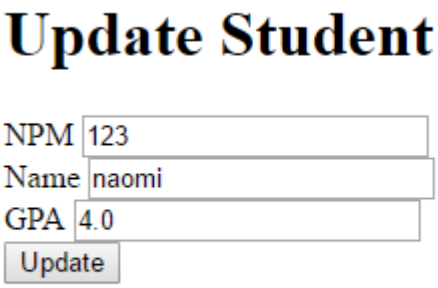
```
@RequestMapping(value="/student/update/submit", method =RequestMethod.POST)
public String updateSubmit (
        @RequestParam(value = "npm", required = false) String npm,
        @RequestParam(value = "name", required = false) String name,
        @RequestParam(value = "gpa", required = false) double gpa) {

    String res="success-update";
      StudentModel student = new StudentModel (npm, name, gpa);
      studentDAO.updateStudent (student);

    return res;
}
```

Method ini dipanggil saat tombol submit untuk update di tekan , akan menerima 3 parameter , setelah itu membuat object student baru dengan ketiga parameter tersebut dan memanggi method updateStudent

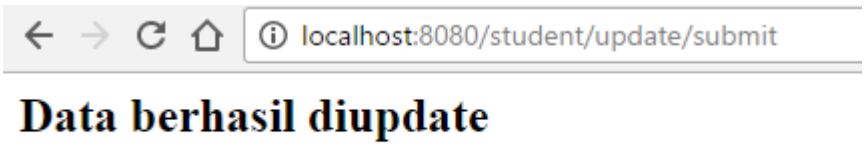
Hasil:



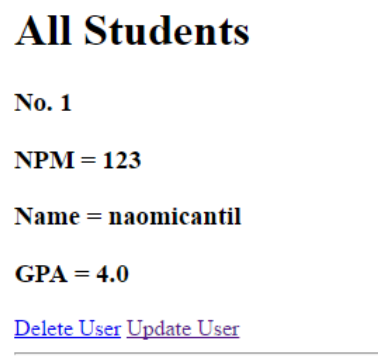
Klik Update User



Masuk ke form update student , lalu ganti Namanya dan tekan update



Update berhasil dilakukan dan view all untuk memeriksa



Data berhasil diubah

3. Latihan Menggunakan Object Sebagai Parameter

1. Menambahkan th:object=”${student}” pada tag **<form>** di view dan Menambahkan th:field=”*{[nama_field]}” pada setiap input

```
<body>

    <h1 class="page-header">Update Student</h1>

    <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}"/
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name"   th:field="*{name}" />
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa"   th:field="*{gpa}"/>
        </div>

        <div>
            <button type="submit" name="action" value="update">Update</button>
```

2. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

```
public String updateSubmit (StudentModel model) {

    String res="success-update";
      StudentModel student = new StudentModel (model.getNpm(), model.getName(), model.getGpa());
      studentDAO.updateStudent (student);

    return res;
```

Pertanyaan

Beberapa pertanyaan yang perlu Anda jawab:
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?
Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Validasi dilakukan pada model

Seperti contoh di bawah ini

```java
public class User {

    private Integer id;
    private String firstName;
    private String sex; // I'm ready when you are :P

    /**
    * ... All Your Getters and Setters Here
    */

    public boolean validate() {
        if (this.firstName == null) {
            return false;
        }
        if (this.sex == null) {
            return false;
        }
        return true;
    }
```

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method?

POST lebih aman karna tidak muncul di url sedangkan GET ada di url dan ada limitasi

Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

Iya ada

Bila di GET perlu menambahkan parameter RequestParam untuk setiap value yang di submit sedangkan untuk POST hanya menerima parameter object saja

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