Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER — UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018

Latihan Menambahkan Delete

1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.
2. Pada **viewall.html** tambahkan

viewall.html | StudentServi... | StudentServi... | StudentMapp... | application....

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

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**
   a. Tambahkan method delete student yang menerima parameter **NPM**.
   b. Tambahkan annotation delete di atas dan SQL untuk menghapus

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

4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**
   a. Tambahkan log untuk method tersebut dengan cara menambahkan
   b. Panggil method delete student yang ada di Student Mapper

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

5. Lengkapi method delete pada class **StudentController**
   a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found.

Poppy Rosaline 1606954956

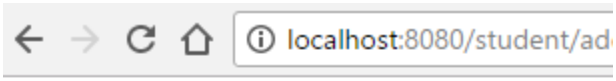
FAKULTAS ILMU KOMPUTER

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018

StudentCont... | viewall.html | StudentServi... | form-update.... | Form.html

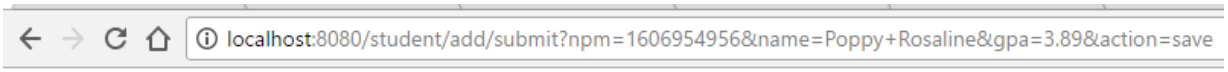
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);
    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {

        return "not-found";
    }

}
```

- **Add Student**



Problem Editor

NPM 1606954956
Name Poppy Rosaline
GPA 3.89
Save



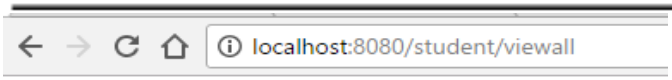
Data berhasil ditambahkan

- **View All Students**

Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER — UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018



All Students

No. 1

NPM = 1606954956

Name = Poppy Rosaline

GPA = 3.89

Delete Data

No. 2

NPM = 1606955994

Name = Rinjani

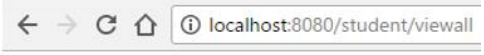
GPA = 3.9

Delete Data

- **Delete Data**

Data berhasil dihapus

- **Hasil View Ketika Data Telah di Delete**



All Students

No. 1

NPM = 1606955994

Name = Rinjani

GPA = 3.9

Delete Data

FAKULTAS ILMU KOMPUTER — UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018

- **View Student NPM 1606954956**

Student not found

NPM = 1606954956

Penjelasan Delete

Delete delete berfungsi untuk menghapus data student berdasarkan hanya dari NPM student. Untuk membuat method ini menggunakan log.info di pada StudentServiceDatabase, untuk memberikan argument dengan format string. Untuk menghapus data student menggunakan query yaitu delete from student where npm= “npm yang ingin dihapus” selanjutnya untuk menghandle pada controller maka jika npm ditemukan maka akan mengarahkan pada halaman view delete namun jika data npm tidak ditemukan mengarahkan ke halaman tidakDitemukan.html. Untuk menjalankannya, lakukan penambahan student kemudian ke halam viewall dan selanjutnya lakukan hapus data kemudian jika berhasil, cek kembali kehalam viewall untuk mengetahui apakah data tersebut telah hilang.

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);
```

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

StudentCont... | viewall.html | StudentServi... | St

```
package com.example.service;

import java.util.List;

public interface StudentService
{
    StudentModel selectStudent (String npm);

    List<StudentModel> selectAllStudents ();

    void addStudent (StudentModel student);

    void deleteStudent (String npm);

    void updateStudent (StudentModel student);
}
```

3. Tambahkan implementasi method **updateStudent** pada class **StudentServiceDatabase**. Jangan lupa tambahkan **logging** pada method ini

FAKULTAS ILMU KOMPUTER

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018

```
    @Override
    public void updateStudent (StudentModel student)
    {
        log.info("Student {} name update to", student.getNpm(), student.getName());
        studentMapper.updateStudent(student);
    }
```

4. Tambahkan link Update Data pada **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 Data</a><br/>
            <a th:href="'/student/update/' + ${student.npm}">Update Data</a><br/>
            <hr/>
        </div>
    </body>
</html>
```

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

```
<!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">Update</button>
        </div>
    </form>

</body>

</html>
```

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

Poppy Rosaline 1606954956

viewall.html StudentServi... StudentServi...

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil di Update</h2>
    </body>
</html>
```

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

StudentCont... viewall.html StudentServi... StudentServi... form-update....

```
            return "delete";
        } else {

            return "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 {

            return "not-found";
        }
```

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

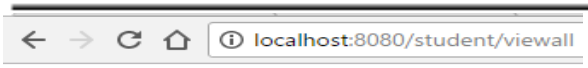
```
@RequestMapping(value = "/student/update/{npm}", method = RequestMethod.POST)
public String updateSubmit (@RequestParam(value = "npm", required =false)String npm,
                            @RequestParam(value = "name", required =false)String name,
                            @RequestParam(value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

    return "success-update";
}
```

9. Jalankan Spring Boot dan coba test program Anda.
    - **View All Student**

Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER — UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018



All Students

No. 1

NPM = 1606954956

Name = Poppy Rosaline

GPA = 3.87

Delete Data
Update Data

No. 2

NPM = 1606955994

Name = Rinjani

GPA = 3.9

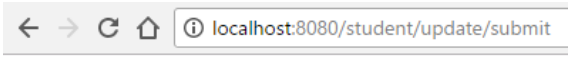
Delete Data
Update Data

- **Update Student No.1**

Problem Editor

NPM 1606954956
Name Poppy Rosaline Stefany
GPA 3.87
Update

- **Konfirmasi setelah data di update**



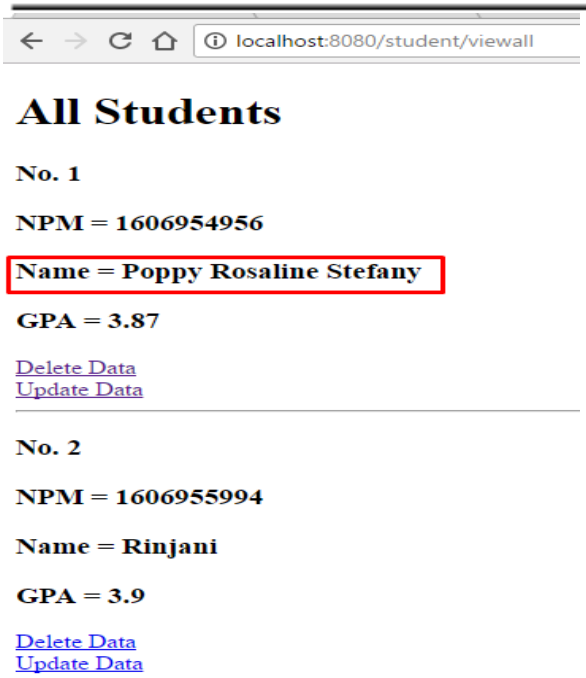
Data berhasil di Update

- **View All Setelah di update**

Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018



Penjelasan Update Data

Update ini berfungsi untuk melakukan pembaharuan data student, dimana attribute student yang dapat di update adalah name dan GPA untuk npm tidak dapat di update karena pada database NPM adalah **PrimaryKey**. Untuk membuat method ini, maka pada **StudentMapper** tambahkan method untuk update berdasarkan npm yang dipilih kemudian akan disimpan pada **StudentModel**. Selanjutnya, menambahkan method update pada **StudentController** untuk melakukan pengecekan apakah data student yang di update ada dan jika berhasil update akan menampilkan halaman **success-update.html.** Kemudian menambahkan method UpdateSubmit dimana method ini berfungsi untuk menerima data NPM, nama dan GPA yang akan dilakukan pada halaman form-update akan dipanggil method updateStudent yang mempunyai object student, dan action yang dilakukan akan sama halnya dengan action method updateStudent.

Latihan Menggunakan Object Sebagai Parameter

1. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakan RequestParam.
2. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakan RequestParam.

Poppy Rosaline 1606954956


```
@RequestMapping(value = "/student/update/{npm}", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute StudentModel student)
{
    studentDAO.updateStudent (student);

    return "success-update";
}
```

3. Tahapannya kurang lebih sebagai berikut:
   a. Menambahkan th:object="${student}" pada tag di view
   b. Menambahkan th:field="*{[nama_field]}" pada setiap input
   c. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

InMemoryStu... | form-update.... | StudentCont... | viewall.html | StudentServi... | StudentServi... | success-upd...

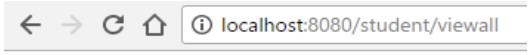
```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

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

<body>

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

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

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

</body>

</html>
```

c. Tes lagi aplikasi Anda.
   - **View All Student**

Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018



All Students

No. 1

NPM = 1606954956

Name = Poppy Rosaline Stefany

GPA = 3.87

Delete Data
Update Data

No. 2

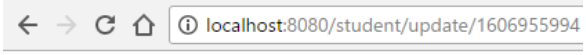
NPM = 1606955994

Name = Rinjani Samosir

GPA = 3.9

Delete Data
Update Data

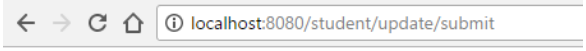
- **Edit data Student**



Edit Student

NPM 1606955994
Name Rinjani
GPA 3.9
Update

- **Konfirmasi Student**



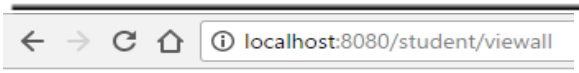
Data berhasil di Update

- **View All Data**

Poppy Rosaline 1606954956

FAKULTAS ILMU KOMPUTER — UNIVERSITAS INDONESIA

Tutorial 4
Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Genap 2017/2018



All Students

No. 1

NPM = 1606954956

Name = Poppy Rosaline Stefany

GPA = 3.87

Delete Data
Update Data

No. 2

NPM = 1606955994

Name = Rinjani

GPA = 3.9

Delete Data
Update Data

Penjelasan Object Sebagai Parameter

Pada Update student akan diubah, dengan menggunakan object sebagai parameter. Ini berfungsi untuk meminimalisir dalam melakukan update, yang sebelumnya di lakukan untuk per satu attribut. Pada bagian ini attribute tersebut akan disimpan pada object dan akan dilakukan deklarasi pada object yang akan mengumpulkan data mahasiswa yaitu dengan th:obejct dan dengan mendeklarasikan th:field utnuk field yang digunakan sesuai pada objek student.

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?

Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Jawab :

Cara melakukan validasi adalah dengan pemanfaatan model.

Pada controller yang menerima submit ditambahkan @Valid pada parameter Object dan tambahkan parameter Object bindingResult. Selnajutnya pada body tambahkan kondisi untuk melakukan validasi dengan menggunakan code "bindingResult.hasErrors()" jika hasil validasi error maka akan kembali ke halaman form gagal dan jika validasi berhasil maka data berhasil di update. Pada method updateSubmit juga dibuat kondisi jika kondisi null terjadi maka akan memanggil halaman gagal], jika kondisi null tidak terjadi maka akan data *student* akan diupdate. Selain itu, juga diperlukan notasi @NotNull pada variable nama dan gpa pada *class* StudentMapper

Validasi diperlukan karena form berhubungan langsung dengan database. Jika pada struktur table database terdapat field not null maka harus dipastikan bahwa pada form telah diinputkan nilai untuk field tersebut agar tidak terjadi error atau ketidaksesuaian data. Selain validasi optional atau required juga perlu untuk dilakukan validasi tipe data dan lenght yang diinputkan pada form harus sama dengan tipe data dan lenght pada struktur table database.

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 di post dikirim menggunakan method berbeda?

Jawab :

Form submit berisi nilai atau data-data yang bersifat sensitive dan nilai yang cukup panjang, sehingga nilai-nilai atau isi pada form ini tidak perlu di tampilkan pada alamat browser. Secara mendasar fungsi POST digunakan untuk mengirimkan data ke server, sedangkan GET digunakan untuk mengambil data dari server. tidak efisien jika menggunakan method GET dimana aka tertampil nilai yang diinputkan pada url.

Kelebihan jika menggunakan method POST:

1. Parameter/nilai tidak tersimpan di browser
2. Dengan method POST lebih aman karena data langsung dikirim ke server
3. Dapat mengirim berbagai jenis data seperti gambar, file, dll, tidak harus teks

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

Jawab :

Bisa, jika dijalankan secara bergantian namun jika tidak bergantian tidak bisa karena nantinya controller akan sulit untuk menentukan apakah request method tersebut akan menerima GET atau POST dan sebaiknya menggunakan request method yang berbeda.