Nama : Rinjani Samosir

NPM : 1606954994

Kelas : Ekstensi 2016

Tutorial 4

Arsitektur dan Pemrograman Aplikasi Perusahaan

Semester Genap 2017/2018

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Latihan Menambahkan Delete

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

```
<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>
```

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**
   - Tambahkan method delete student yang menerima parameter NPM

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

   - Tambahkan annotation delete di atas dan SQL untuk menghapus

```
import org.apache.ibatis.annotations.Delete;
```

4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**
   - Tambahkan log untuk method tersebut dengan cara menambahkan

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

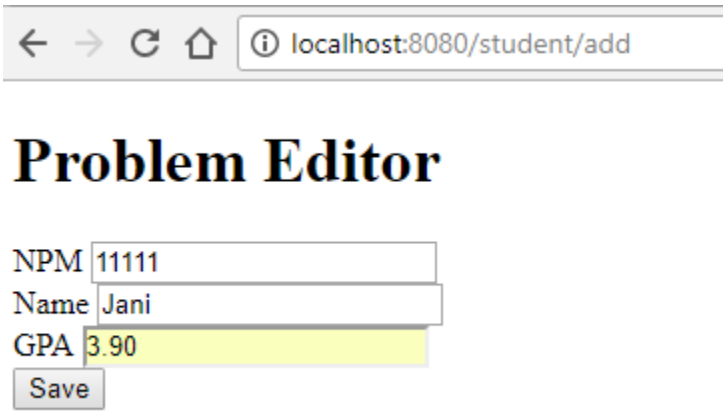
- Panggil method delete student yang ada di Student Mapper

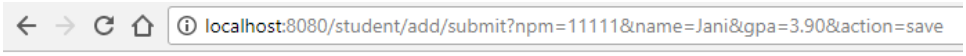
```
@Override
public void deleteStudent (String npm)
{
    log.info("student" + npm + "deleted");
    studentMapper.deleteStudent (npm);
}
```

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

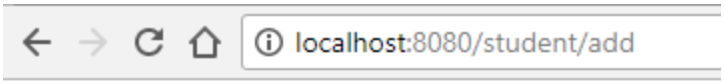
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        return "not-found";
    }


}
```

6. Jalankan Spring Boot app dan lakukan beberapa insert



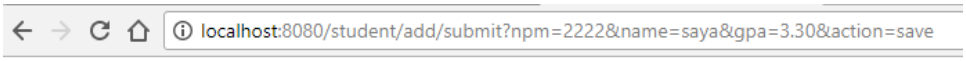


Data berhasil ditambahkan



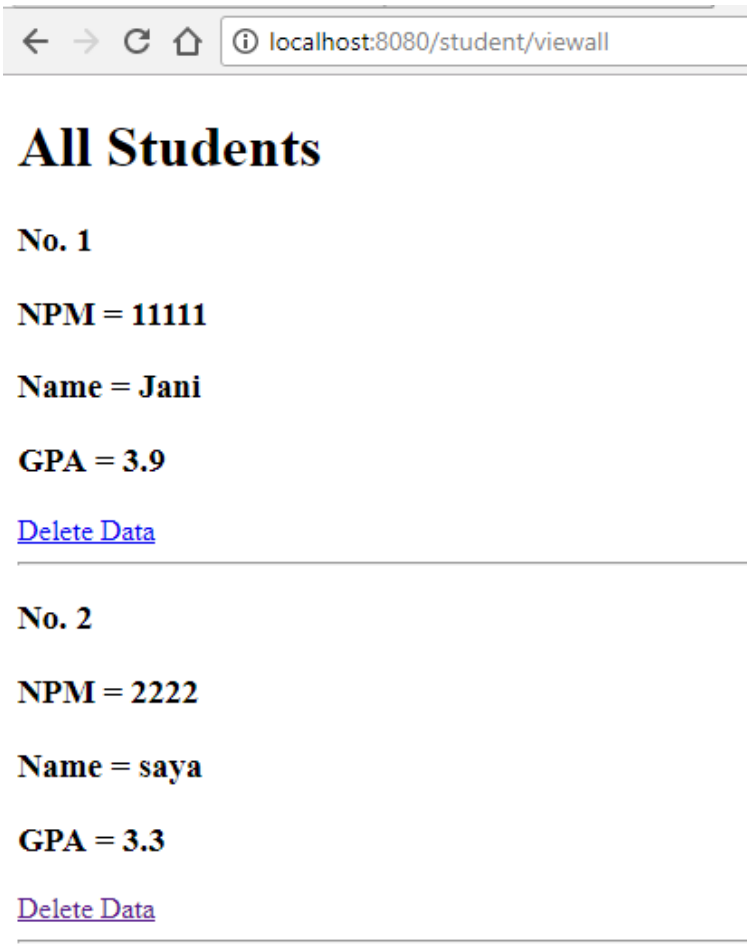
Problem Editor

NPM 2222
Name saya
GPA 3.30
Save

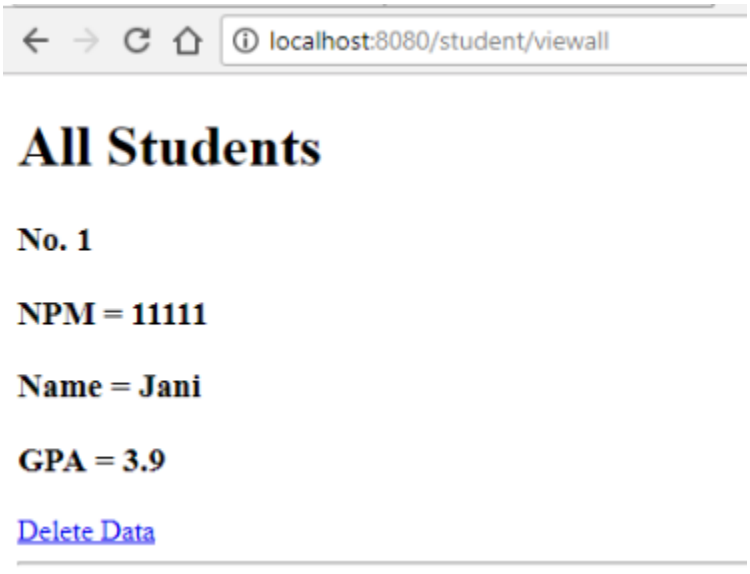


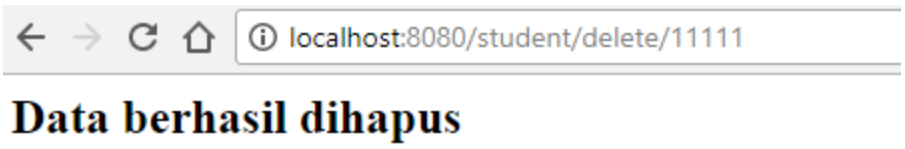
Data berhasil ditambahkan

7. Tampilan Viewall

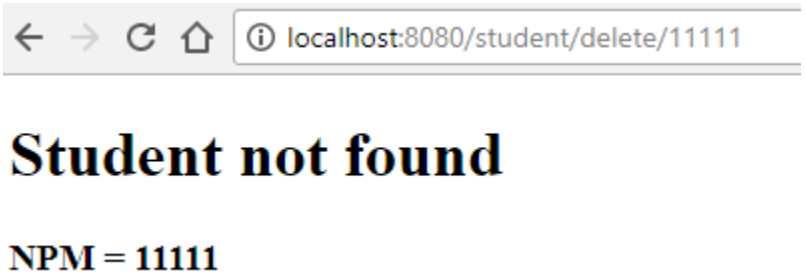


8. Tampilan Delete NPM = 11111





9. Tampilan jika dilakukan delete NPM = 11111 yang kedua kali



PENJELASAN :

Membuat delete diawali dengan pembuatan script SQL di Class StudentMapper yang menerima parameter NPM. StudentController akan memanggil method ini kemudian dikirimkan ke StudentService database untuk dicek apakah data tersebut ada di database.

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**

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

3. Tambahkan implementasi method updateStudent pada class StudentServiceDatabase.

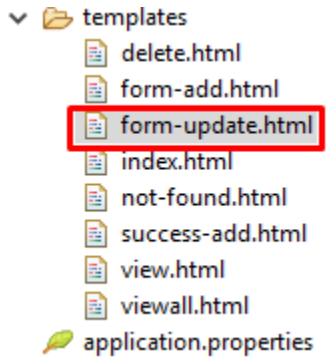
   Jangan lupa tambahkan logging pada method ini.

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

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

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

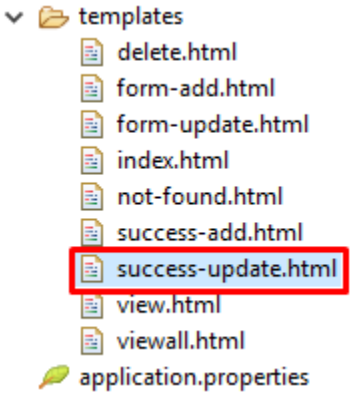
5. Copy view **form-add.html** menjadi **form-update.html.**



Isi form-update.html

```
<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>
```

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



Ubah keterangan seperlunya

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diubah</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);

    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        return "not-found";
    }
}
```

8. Tambahkan method **updateSubmit** pada class **StudentController**
   - Karena menggunakan post method maka request mappingnya adalah sebagai berikut:

```
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
```

   - Header methodnya adalah sebagai berikut:

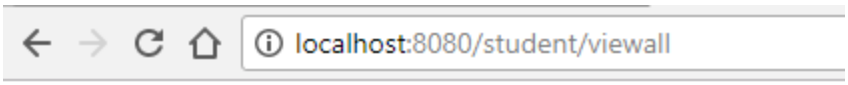
```
public String updateSubmit (
        @RequestParam(value="npm", required=false) String npm,
        @RequestParam(value="name", required=false) String name,
        @RequestParam(value="gpa", required=false) double gpa) {
```

   - Lengkapi method dengan memanggil method **update Student** dan kembalikan **view 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";

}
```

9. Jalankan Spring boot

   Data yang ingin diubah



All Students

No. 1

NPM = 6454

Name = pooerr

GPA = 3.45

Delete Data
Update Data

Form update

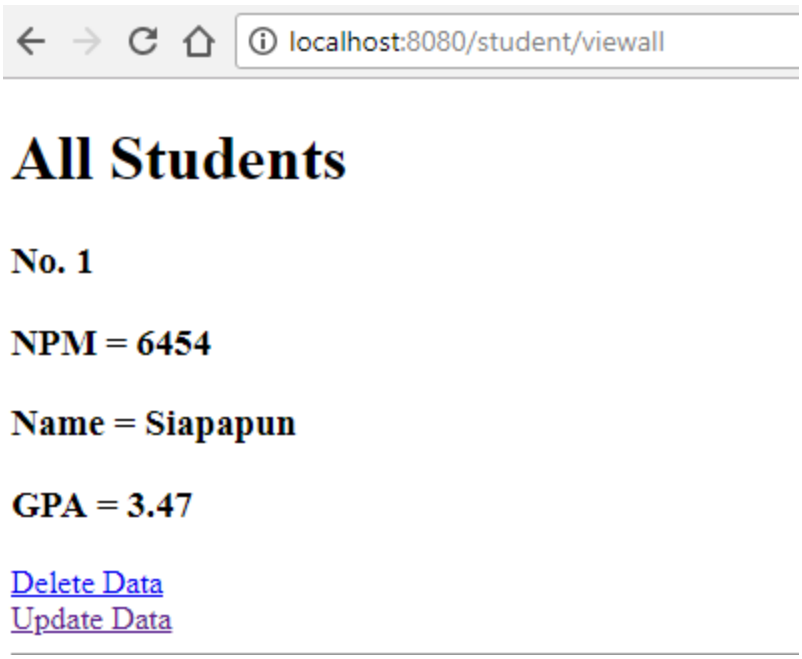
Problem Editor

NPM 6454
Name Siapapun
GPA 3.47
Save

Data berhasil diubah

Data berhasil diubah

Data setelah diubah adalah sebagai berikut.



PENJELASAN

Membuat update diawali dengan menambahkan method updateStudent pada interface StudentService. Method ini diisi dengan query SQL update yang akan dipanggil ketika proses update dilakukan. Selanjutnya dibuat method updateStudent pada class StudentServiceDatabase untuk memanggil method updateStudent yang berada pada interface StudentMapper. Selain itu ditambahkan juga logging. Kemudian dibuat view untuk menampilkan info-info yang diperlukan dengan penamaan form-update.html. Method success-update.html ditambahkan untuk memberikan informasi bahwa proses update sudah berhasil dilakukan. Pada class StudentController ditambahkan method update untuk melakukan validasi. Selain itu ditambahkan juga method updateSubmit untuk request mapping dengan method POST.

Latihan Menggunakan Object Sebagai Parameter

1. Mengubah method **updateSubmit** untuk menerima parameter berupa model **studentModel**.

```
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute StudentModel student) {

    studentDAO.updateStudent (student);

    return "success-update";

}
```

2. Melakukan handling form

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

3. Tes aplikasi kembali

All Students

No. 1

NPM = 6454

Name = Siapapun

GPA = 3.7

Delete Data
Update Data

PENJELASAN

Menggunakan Object sebagai parameter adalah untuk mengurangi jumlah parameter yang digunakan. Pada method updateSubmit diterima parameter berupa StudentModel. Tahapan yang dilakukan adalah dengan menambahkan th:object pada tag form dan th:field pada setiap input yang ada (npm, name dan gpa).

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.

   Jawab :

   Untuk melakukan validasi caranya adalah dengan mengkonfigurasi kelas Controller menggunakan @initBinder. Anotasi @initBinder digunakan untuk menginisialisasi WebDataBinder yang mengikat parameter object. Jika terjadi error maka secara otomatis error tersebut akan dikenali sebagai kesalahan di BindingResult yang berfungsi sebagai handler.

   Validasi diperlukan untuk menghandle tipe data, ukuran serta inputan data yang tidak sesuai dengan default NULL/Not NULL

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 :

POST method lebih sering digunakan karena pengirimannya lebih tertutup dan jauh lebih aman karena data yang dikirimkan tidak terlihat. Method POST digunakan untuk mengirimkan data dari Client untuk diproses di Server kemudian Server akan memberikan hasilnya ke Client. Data yang dikirimkan dengan method POST disertakan di Body of Request. Selain itu, method POST juga dapat mengirimkan parameter URL sekaligus mengirimkan data tersembunyi ke Server.

Ya, perlu penanganan yang berbeda di method Controller dengan RequestMethod.POST pada RequestMapping.

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

   Jawab : Bisa. Tetapi tidak bisa dijalankan. Untuk pemanggilannya harus satu per satu.