

LESSON LEARNED

Dalam lab kali ini, saya mempelajari bagaimana cara melakukan operasi MVC yang melakukan akses ke database. Setiap query database yang dibutuhkan harus dibuat terlebih dahulu pada class Mapper, dimana nantinya kelas service yang berisi method yang mengimplementasikan fungsi program akan mengakses query dari Mapper. Method dari service tersebutlah yang akan digunakan pada controller untuk menjalankan implementasi penyimpanan database serta logic program. Saya juga mempelajari penggunaan method POST pada spring boot ini, dimana untuk melakukan method POST sama seperti dasar membuat web, kita membuat form yang melakukan submission dan mengarahkan halaman ke route method yang kita tuju. Pada form tersebut, kita memasukan input yang akan diproses di method, bisa dalam bentuk RequestParam atau dengan cara yang lebih efisien yaitu menggunakan parameter objek.

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 ada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend

Validasi dapat dilakukan sama seperti memproses input dari RequestsParam, namun kita tidak menggunakan variable dari parameter, melainkan kita hanya tinggal memanggil attribute dari objek yang diinput. Tentunya validasi ini tetap diperlukan untuk memproses objek yang diinput.

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?

Karena method POST berasal dari dalam page html bukan dari url, dimana jika ingin memasukan input di page html dan melakukan submit, data dari input hanya bisa diproses hanya jika tag input dimasukkan ke dalam tag form yang memiliki method POST. Tentunya diperlukan penanganan yang berbeda jika method yang digunakan berbeda.

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

Tidak bisa, Karena method akan kebingungan untuk menerima parameter yang benar

LATIHAN

Membuat DELETE

- Berikut adalah **method delete** dan **query delete** pada class mapper:

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

Method di atas memiliki notasi Delete untuk mengirimkan perintah DELETE ke database.
Method ini nantinya akan dipanggil pada class StudentServiceDatabase

- Berikut adalah method delete pada **StudentServiceDatabase**:

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

Method di atas menjalankan query yang ada di class Mapper.

- Berikut adalah method delete pada **StudentController**:

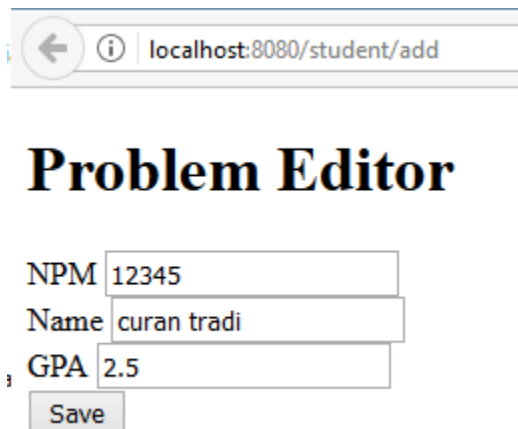
```
@RequestMapping("/student/delete/{npm}")  
public String delete (@PathVariable(value = "npm") String npm)  
{  
    StudentModel student = studentDAO.selectStudent(npm);  
    if (student != null) {  
        studentDAO.deleteStudent (npm);  
        return "delete";  
    } else {  
        return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

```
}
```

Method controller di atas bertugas untuk menjalankan implementasi delete ketika program mengarah pada routing seperti yang tertera pada RequestMapping atau menekan tombol delete pada viewall. Method tersebut akan menampilkan halaman delete jika npm yang dimasukkan terdaftar di database, jika tidak maka akan menampilkan halaman not-found.

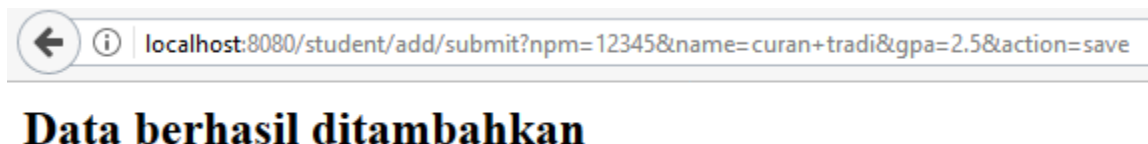
Hasil tampilan

- **Halaman Add**



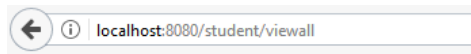
The screenshot shows a web browser window with the address bar displaying 'localhost:8080/student/add'. The main content area has the title 'Problem Editor' in a large, bold, black serif font. Below the title is a form with three input fields: 'NPM' with the value '12345', 'Name' with the value 'curan tradi', and 'GPA' with the value '2.5'. There is a 'Save' button below the GPA field.

- **Halaman add-success**



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/student/add/submit?npm=12345&name=curan+tradi&gpa=2.5&action=save'. The main content area has the text 'Data berhasil ditambahkan' in a large, bold, black serif font.

- **Halaman Viewall**



All Students

No. 1

NPM = 12345

Name = curan tradi

GPA = 2.5

[Update Data](#)

[Delete Data](#)

No. 2

NPM = 12346

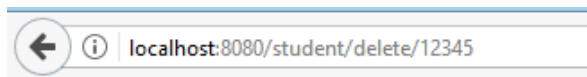
Name = evan flemming

GPA = 3.8

[Update Data](#)

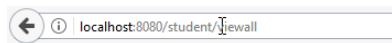
[Delete Data](#)

- **Halaman delete-success**



Data berhasil dihapus

- **Halaman setelah delete**



All Students

No. 1

NPM = 12346

Name = evan flemming

GPA = 3.8

[Update Data](#)

[Delete Data](#)

Membuat Update

- Berikut adalah **method update** dan **query delete** pada class mapper:

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

Method di atas memiliki notasi Update untuk mengirimkan perintah UPDATE ke database.
Method ini nantinya akan dipanggil pada class StudentServiceDatabase

- Berikut adalah method update pada **StudentServiceDatabase**:

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

Method di atas menjalankan query yang ada di class Mapper.

- Berikut adalah method update pada **StudentController**:

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

Method controller ini bertugas untuk memproses input update sebelum mengalihkan ke halaman form-update. Jika npm student ada, program akan terlebih dahulu memasukan objek student ke model supaya nantinya data student dapat ditampilkan pada form-update. Method ini akan berjalan ketika kita memasukan routing yang ada di RequestMapping atau menekan tombol Update pada viewall.

- Berikut adalah method updateSubmit pada StudentController:

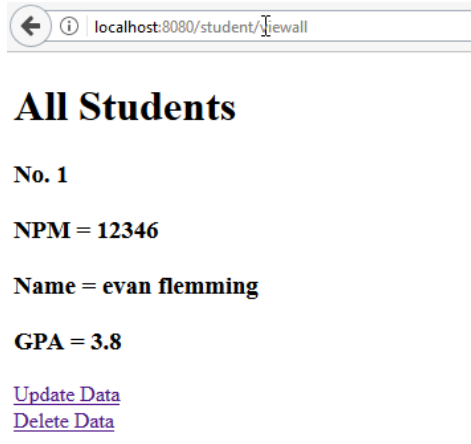
```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
```

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

Method controller ini bertugas untuk memanggil method updateStudent untuk mengupdate data student setelah memasukan routing dari RequestMapping atau menekan tombol submit.

Hasil tampilan

- **Halaman Viewall**



localhost:8080/student/viewall

All Students

No. 1

NPM = 12346

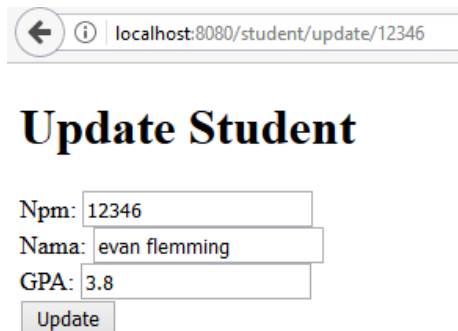
Name = evan flemming

GPA = 3.8

[Update Data](#)

[Delete Data](#)

- **Halaman form-update**



localhost:8080/student/update/12346

Update Student

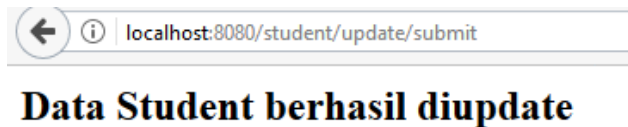
Npm: 12346

Nama: evan flemming

GPA: 3.8

Update

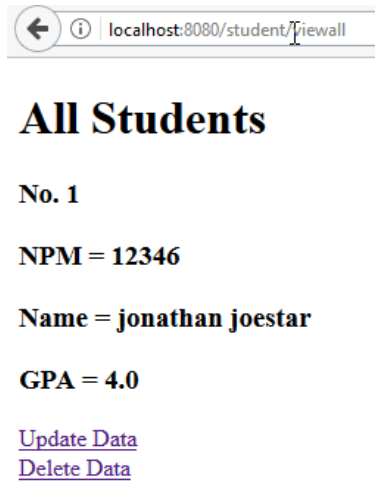
- **Halaman update-success**



localhost:8080/student/update/submit

Data Student berhasil diupdate

- Halaman setelah update



localhost:8080/student/viewall

All Students

No. 1

NPM = 12346

Name = jonathan joestar

GPA = 4.0

[Update Data](#)
[Delete Data](#)

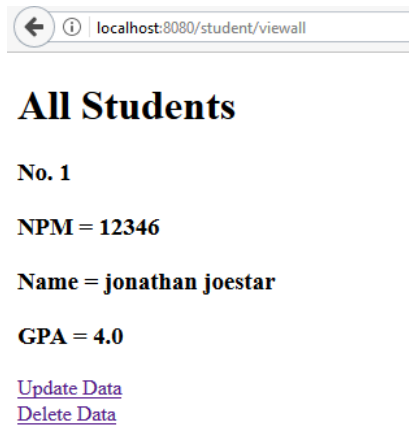
Membuat Objek Sebagai Parameter

- Tampilan html

```
1 <!DOCTYPE HTML>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:th="http://www.thymeleaf.org">
4 <head>
5
6 <title>Update student</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8 </head>
9
10 <body>
11
12 <h1 class="page-header">Update Student</h1>
13 <form th:object="${student}" action="/student/update/submit" method="post">
14     <div>
15         <label>Npm: </label>
16         <input th:field="*{npm}" type="text" name="npm" readonly="true" th:value="${student.npm}"/>
17     </div>
18     <div>
19         <label>Nama: </label>
20         <input th:field="*{name}" type="text" name="name" th:value="${student.name}"/>
21     </div>
22     <div>
23         <label>GPA: </label>
24         <input th:field="*{gpa}" type="text" name="gpa" th:value="${student.gpa}"/>
25     </div>
26     <div>
27         <button type="submit" name="action" value="save">Update</button>
28     </div>
```


Hasil tampilan

- Tampilan viewall



← ⓘ | localhost:8080/student/viewall

All Students

No. 1

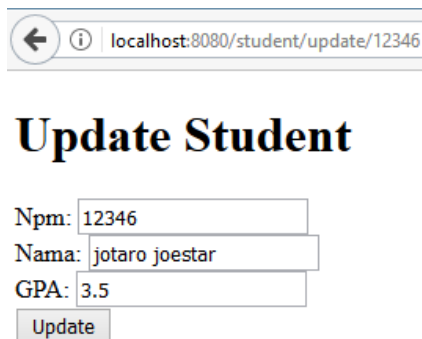
NPM = 12346

Name = jonathan joestar

GPA = 4.0

[Update Data](#)
[Delete Data](#)

- Tampilan update-form



← ⓘ | localhost:8080/student/update/12346

Update Student

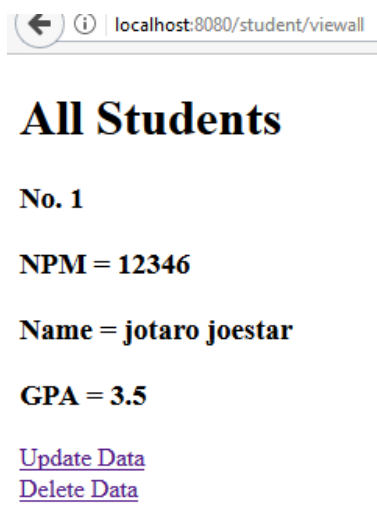
Npm: 12346

Nama: jotaro joestar

GPA: 3.5

Update

- Tampilan setelah update



← ⓘ | localhost:8080/student/viewall

All Students

No. 1

NPM = 12346

Name = jotaro joestar

GPA = 3.5

[Update Data](#)
[Delete Data](#)