

## Tutorial 4 - 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** tambahkan

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

<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/>
</div>
/body>
1\
```

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**

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

4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**

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

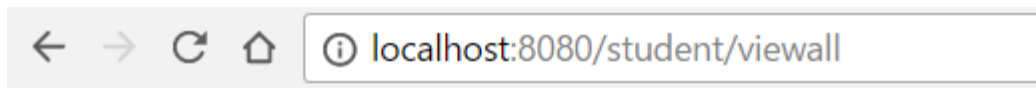
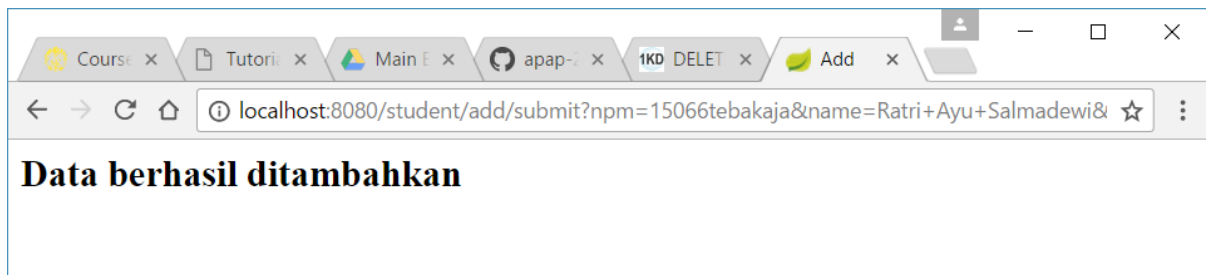
5. Lengkapi method delete pada class **StudentController**

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    for(int i = 0; i< studentDAO.selectAllStudents().size(); i++){
        if(studentDAO.selectAllStudents().get(i).getNpm().equalsIgnoreCase(npm)){
            studentDAO.deleteStudent (npm);
            return "delete";
        }
    }

    return "not-found";
}
```

6. Jalankan **Spring Boot app** dan lakukan beberapa **insert**

7. Tampilan **viewAll**



## All Students

**No. 1**

**NPM = 15066tebakaja**

**Name = Ratri Ayu Salmadewi**

**GPA = 3.5**

[Delete Data](#)

---

**No. 2**

**NPM = 1506757831**

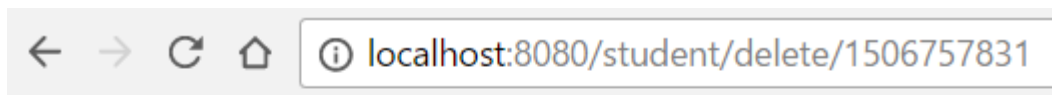
**Name = Abdalla Chair**

**GPA = 3.5**

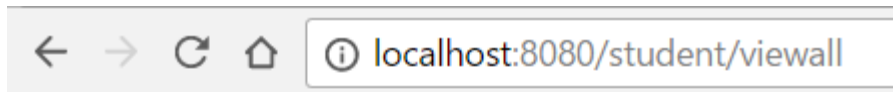
[Delete Data](#)

---

#### 8. Contoh tampilan delete NPM



**Data berhasil dihapus**



## All Students

**No. 1**

**NPM = 15066tebakaja**

**Name = Ratri Ayu Salmadewi**

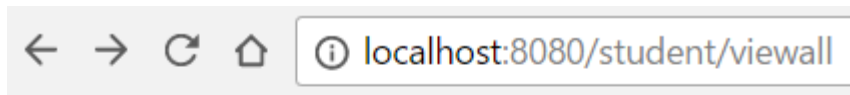
**GPA = 3.5**

[Delete Data](#)

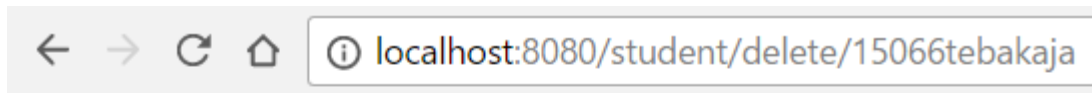


**Data berhasil dihapus**

9. Tampilan jika dilakukan **delete NPM** yang kedua kalinya



## All Students



## Student not found

**NPM = 15066tebakaja**

### Latihan Menambahkan Update

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

```
@Update("UPDATE student SET npm = #{npm}, 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.getNpm() + " updated");
    studentMapper.updateStudent(student);
}
```

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

```
<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/>
</div>
```

#### 5. Copy view **form-add.html** menjadi **form-update.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
14 <form action="/student/update/submit" method="post">
15   <div>
16     <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" />
17   </div>
18   <div>
19     <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" />
20   </div>
21   <div>
22     <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" />
23   </div>
24   <div>
25     <button type="submit" name="action" value="save">Update</button>
26   </div>
27 </form>
28
29 </body>
30
31 </html>
```

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

```
1 <html>
2   <head>
3     <title>Add</title>
4   </head>
5   <body>
6     <h2>Data berhasil diupdate</h2>
7   </body>
8 </html>
```

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

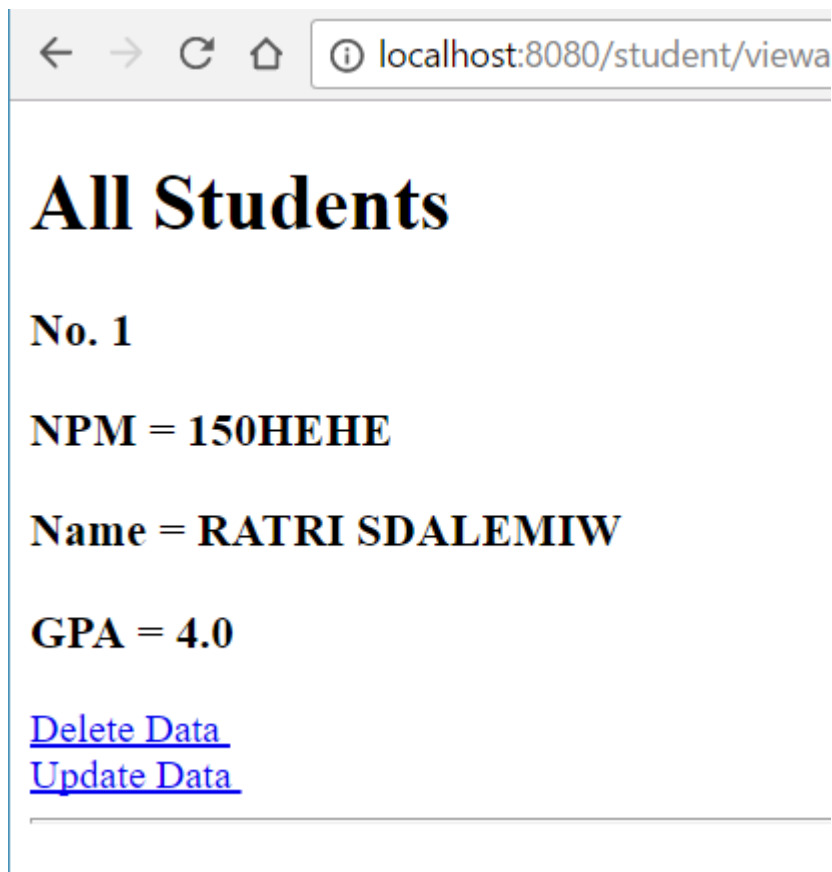
```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    List<StudentModel> list = studentDAO.selectAllStudents();
    for(int i = 0; i < list.size(); i++){
        if(list.get(i).getNpm().equalsIgnoreCase(npm)){
            model.addAttribute("student", list.get(i));
            return "form-update";
        }
    }

    return "not-found";
}
```

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)
{
    studentDAO.updateStudent(new StudentModel(npm, name, gpa));
    return "success-update";
}
```

9. Jalankan **Spring Boot** dan coba test program Anda.



← → ↺ 🏠 ⓘ localhost:8080/student/viewa

# All Students

No. 1

NPM = 150HEHE

Name = RATRI SDALEMIW

GPA = 4.0

[Delete Data](#)

[Update Data](#)

[←](#) [→](#) [↻](#) [🏠](#)

localhost:8080/student/update/150HEHE

# Update Student

NPM

150HEHE

Name

RATRI SDALEMIWIWIW

GPA

4.9999

Update

[←](#) [→](#) [↻](#) [🏠](#)

localhost:8080/student/update/submit

**Data berhasil diupdate**

[←](#) [→](#) [↻](#) [🏠](#)

localhost:8080/student/viewall

## All Students

**No. 1**

**NPM = 150HEHE**

**Name = RATRI SDALEMIWIWIW**

**GPA = 4.9999**

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

## Latihan Menggunakan Object Sebagai Parameter

a. Pada tutorial di atas Anda masih menggunakan RequestParam untuk handle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.

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

c. Cara lengkapnya silakan ikuti pada link Handling Form berikut:  
(<https://spring.io/guides/gs/handling-form-submission/>)

d. Tahapannya kurang lebih sebagai berikut:

- o Menambahkan **th:object="\${student}"** pada tag di view

- o Menambahkan **th:field="\*{[nama\_field]}"** pada setiap input

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

- o Ubah method **updateSubmit** pada **StudentController** yang hanya menerima parameter berupa **StudentModel**

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, StudentModel student)
{
    List<StudentModel> list = studentDAO.selectAllStudents();
    for(int i = 0; i < list.size(); i++){
        if(list.get(i).getNpm().equalsIgnoreCase(student.getNpm())){
            model.addAttribute("student", student);
            return "form-update";
        }
    }

    return "not-found";
}
```

- o Tes lagi aplikasi Anda



←

→

↻

🏠

localhost:8080/student/viewall

# All Students

No. 1

NPM = 150HEHE

Name = RATRI SDALEMIWIWIW

GPA = 4.9999

[Delete Data](#)

[Update Data](#)

←

→

↻

🏠

localhost:8080/student/update/150HEHE

# Update Student

NPM

Name

GPA

←

→

↻

🏠

localhost:8080/student/update/submit

# Data berhasil diupdate

[←](#) [→](#) [↻](#) [🏠](#) [🔍](#) localhost:8080/student/viewall

# All Students

**No. 1**

**NPM = 150HEHE**

**Name = RATRI ONMAR**

**GPA = 7.0**

[Delete Data](#)

[Update Data](#)

---

## 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? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

- Mengecek apakah property dari objek null atau tidak, dapat di handle pada controllernya, dimana gpa tidak boleh kosong, hal tersebut disebabkan karena apabila gpa kosong maka textbox akan mengembalikan tipe string, sedangkan gpa merupakan double sehingga akan menimbulkan error. mengecek apakah student.npm tidak ada isi / null sehingga input tidak bisa ditinggalkan.

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 POST dapat merubah database dan GET dapat diakses dari URL jadi lebih tidak secure, sedangkan GET lebih digunakan kearah mengambil data pada database. Tidak karena sebelumnya sudah di validasi oleh post terlebih dahulu

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

- Hal tersebut mungkin terjadi.