Ratri Ayu
1506689276
C

# 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 Delete Data

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

Di dalam div dengan th:each

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

3. Tambahkan method deleteStudent yang ada di class StudentMapper

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

a. Tambahkan method delete student yang menerima parameter NPM.

b. Tambahkan annotation delete di atas dan SQL untuk menghapus @Delete("[LENGKAPI]") Lengkapi script SQL untuk menghapus mahasiswa dengan NPM tertentu.

Hint: Delete dalam SQL “DELETE FROM table_name [WHERE Clause]”

4. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

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

a. Tambahkan log untuk method tersebut dengan cara menambahkan log.info ("student " + npm + " deleted");

b. Panggil method delete student yang ada di Student Mapper

5. Lengkapi method delete pada class StudentController

Ratri Ayu
1506689276
C

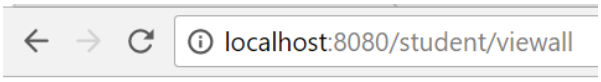
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    for(int i=0; i< studentDAO.selectAllStudents().size(); i++) {
        if(studentDAO.selectAllStudents().get(i).getNpm().equalsIgnoreCase(npm)) {

            studentDAO.deleteStudent (npm);
            return "delete";
        }
    }

    return "not-found";
}
```

a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found

b. Jika berhasil delete student dan tampilkan view delete

● Hint: lakukan select student terlebih dahulu dengan NPM, kurang lebih mirip dengan method view 6. Jalankan Spring Boot app dan lakukan beberapa insert

7. Contoh tampilan View All




# All Students

**No. 1**

**NPM = 1506681234**

**Name = Abdalla**

**GPA = 2.99**

Delete Data

---

**No. 2**

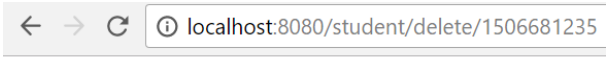
**NPM = 1506681235**

**Name = Rolly**

**GPA = 3.0**

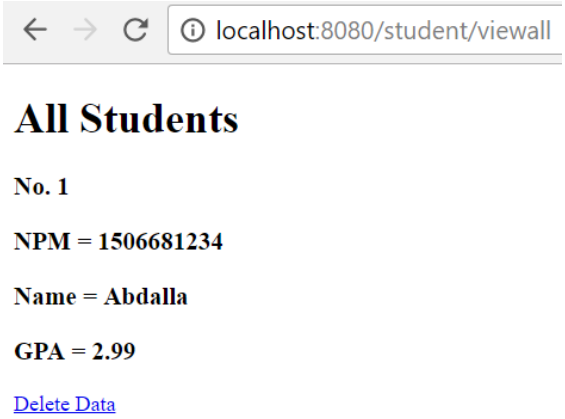
Delete Data

8. Contoh tampilan delete NPM 123

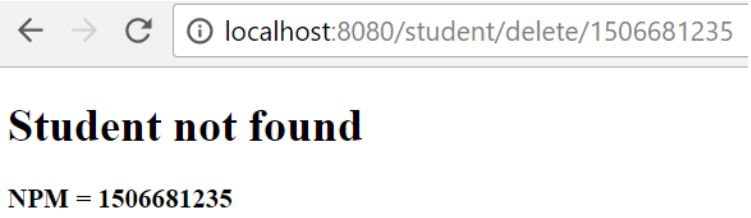



**Data berhasil dihapus**

Ratri Ayu
1506689276
C




9. Contoh tampilan jika dilakukan delete NPM 1506681235 yang kedua kalinya




Method yang dibuat untuk menambahkan delete, pertama dimulai dengan mencari student yang ingin didelete dengan memasukan NPM pada parameter. Hal ini diawali dengan membuat query di class StudentMapper dengan annotation @Delete, kemudian dilengkapi dengan method di class StudentServiceDatabase beserta log dan StudentController. Dalam StudentController, akan dilakukan pencarian NPM dari student yang ingin didelete. Jika tidak ditemukan, akan direturn "not-found" ke view not-found.html, sedangkan jika ditemukan, direturn "delete" ke view delete.html.

**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);
```

a. Parameternya adalah StudentModel student

b. Annotationnya adalah @Update

c. Lengkapi SQL update-nya Hint: Query SQL untuk update

2. Tambahkan method updateStudent pada interface StudentService

Ratri Ayu
1506689276
C

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

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

a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.

b. Ubah action form menjadi /student/update/submit

c. Ubah method menjadi post

d. Untuk input npm ubah menjadi

```
<input type="text" name="npm" readonly="true" th:value="${student.npm}" />
```

readonly agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada.
Input name ubah menjadi

```
<input type="text" name="name" th:value="${student.name}"/>
```

Lakukan hal yang sama untuk GPA.

Ratri Ayu
1506689276
C

```
<body>

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

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

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

a. Ubah keterangan seperlunya

```
<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);
    if(student != null) {
        model.addAttribute("student", student);
        return "form-update";
    }

    return "not-found";
}
```

a. Request mapping ke /student/update/{npm}

b. Sama halnya seperti delete, lakukan validasi.

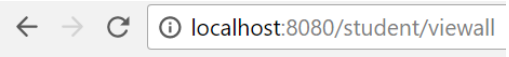
c. Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update

8. Tambahkan method updateSubmit pada class StudentController

Ratri Ayu
1506689276
C

```
@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.




## All Students

**No. 1**

**NPM = 1506681234**

**Name = Hamdallah**

**GPA = 1.0**

Delete Data
Update Data

---

**No. 2**

**NPM = xxx**

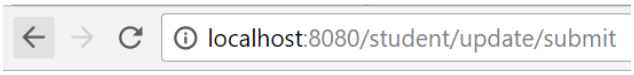
**Name = x**

**GPA = 3.0**

Delete Data
Update Data

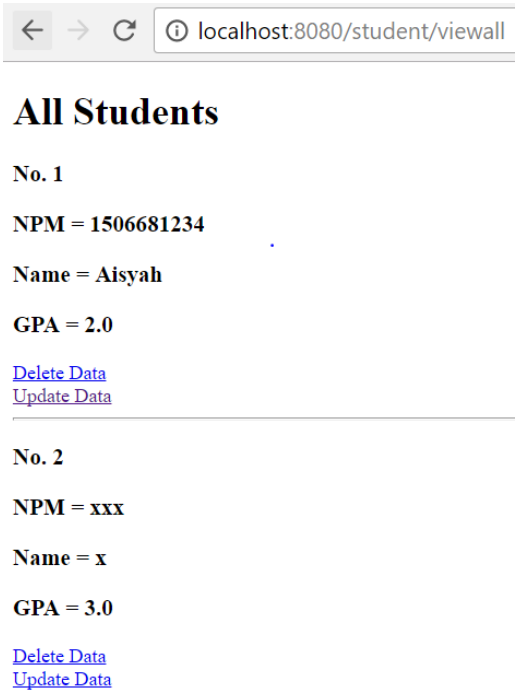

## Update Student

NPM 1506681234
Name Aisyah
GPA 2.0
Save




**Data berhasil diupdate**

Ratri Ayu
1506689276
C




Untuk melakukan update, pertama dimulai dengan membuat query dengan annotation @Update. Method tersebut akan menerima parameter StudentModel pada class StudentMapper. Kemudian dilanjutkan dengan membuat method updateStudent di interface StudentService, dilengkapi juga dengan implementasi pada StudentServiceDatabase beserta log. Setelah itu, pada view all.html ditambahkan link update data dengan href ke student/update/{npm}. Untuk pengisian form variable yang akan diupdate, buat form-update.html. Kemudian membuat method dengan request mapping /student/update/{npm} pada StudentController untuk mencari NPM dari student yang ingin di-update dengan mengiterasi list student. Jika tidak ditemukan, akan return “not-found” ke view not-found.html, sedangkan jika ditemukan akan mereturn “form-update” ke view form-update untuk melakukan update. Kemudian jika form sudah diisi dan mengklik tombol submit, akan masuk ke method updateSubmit, kemudian akan mereturn “success-update” ke view success-update.html

**Latihan Menggunakan Object Sebagai Parameter**

a. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle 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/)

Ratri Ayu
1506689276
C

d. Tahapannya kurang lebih sebagai berikut:

○ Menambahkan th:object=”${student}” pada tag
di view

```
<form th:object="${student}" action="/student/update/submit" method="post">
```

○ Menambahkan th:field=”*{[nama_field]}” pada setiap input

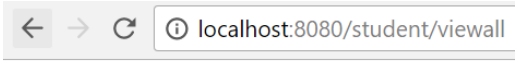
```
<h1 class="page-header">Update Student</h1>

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

○ Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

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

○ Tes lagi aplikasi Anda.

Ratri Ayu
1506689276
C




# All Students

**No. 1**

**NPM = 1506681234**

**Name = Aisyah**

**GPA = 2.0**

Delete Data
Update Data

---

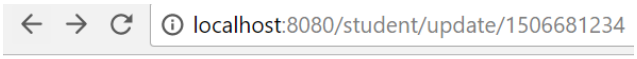
**No. 2**

**NPM = xxx**

**Name = x**

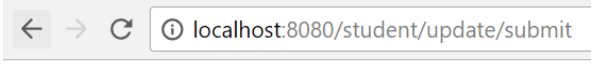
**GPA = 3.0**

Delete Data
Update Data



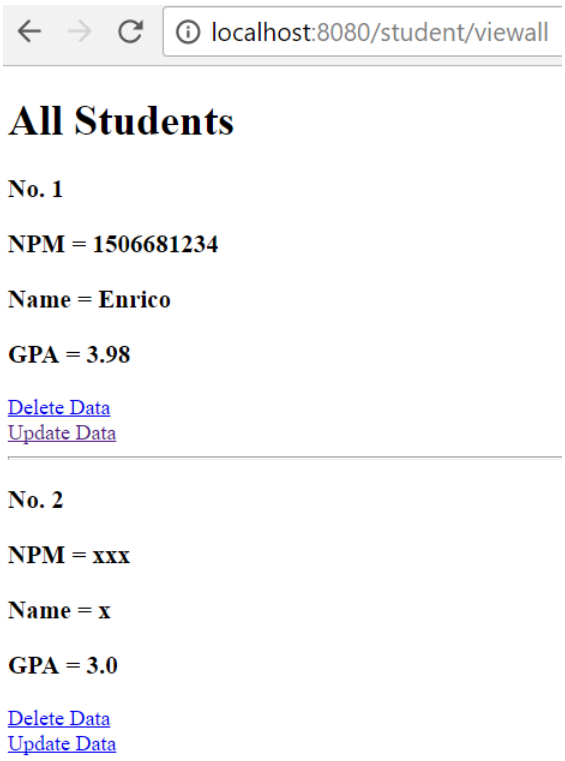

# Update Student

NPM 1506681234
Name Enrico
GPA 3.98
Save




**Data berhasil diupdate**

Ratri Ayu
1506689276
C




Di method ini, dilakukan perubahan penggunaan requestParam sehingga menjadi object. Pada StudentModel, dituliskan th:object=”${student}” pada tag <form> dan th:field=”*{[nama_field]} pada setiap input. Kemudian juga dilakukan perubahan param updateSubmit dengan StudentController untuk menerima parameter StudentModel.

**Pertanyaan**

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.

   Cara melakukan validasi input yang optional adalah dengan mengecek apakah properti objek null atau tidak. Hal tersebut dapat dilakukan pada bagian controller, validasi yang diperlukan ada pada GPA yang tidak boleh kosong, karena jika kosong maka akan terjadi *error parsing*.

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?

Ratri Ayu
1506689276
C

Karena method GET dapat diakses kembali pada URL, sehingga GET lebih tidak secure dibandingkan POST. Oleh karena itu, POST lebih sering digunakan untuk *database*, sedangkan GET digunakan untuk mengambil data dari *database*. Butuh penangan berbeda di header atau body method di *controller* karena pada *request mapping* method update, parameter method harus diubah menjadi request method GET atau method yang sesuai.

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

   Mungkin. Terdapat beberapa cara untuk melakukan hal tersebut, yaitu dengan menambah method = {RequestMethod.GET, RequestMethod.POST} pada parameter @RequestMapping. Selain itu, dapat juga dengan menggunakan Class HttpServletRequest dan melakukan pengecekan *request type* pada method *body controller*.