Latihan Menambahkan Delete

1. Tambahkan method **deleteStudent** yang ada di class **Student** Mapper

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

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

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

}
```

3. Lengkapi method **Delete** pada class **StudentController**

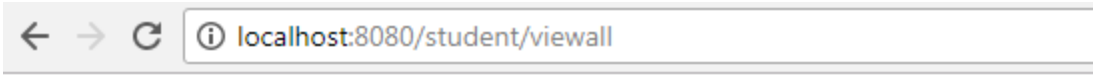
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    studentDAO.deleteStudent (npm);
    StudentModel student = studentDAO.selectStudent(npm);

    if (student != null) {
        model.addAttribute("student",student);
        return "not-found";
    }else {
        model.addAttribute("npm", npm);
        return "delete";
    }

}
```

Problem Editor

NPM 1506
Name Santo
GPA 3.01
Save



All Students

No. 1

NPM = 1234

Name = ester

GPA = 3.45

Delete Data

No. 2

NPM = 1506

Name = Santo

GPA = 3.01

Delete Data

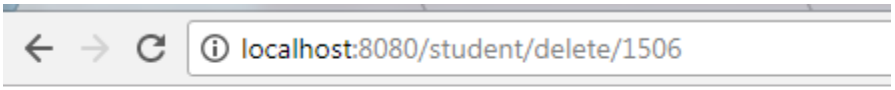
No. 3

NPM = 321

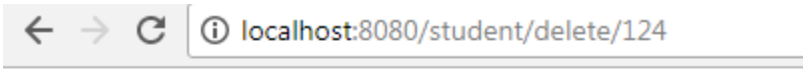
Name = dsa

GPA = 2.3

Delete Data



Data berhasil dihapus



Student not found

NPM = 124

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 implemtasi method **updateStudent** pada class **StudentServiceDatabase**

```
@Override
public void updateStudent(StudentModel student) {
    // TODO Auto-generated method stub
    log.info("student" +student.getNpm()+ "updated");
    studentMapper.updateStudent(student);
}
```

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

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

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="get">
        <div>
            <label for="npm">NPM</label> <input type="text" readonly="true" name="npm" th:value="${student.npm}"/>
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:value={student.nama} />
        </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>

```

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

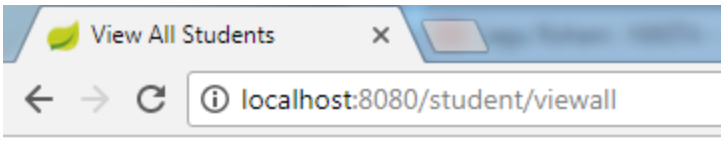
success-update.html

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

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

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

@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";
}
```



All Students

No. 1

NPM = 125

Name = nomu

GPA = 3.1

Delete Data
Update Data

No. 2

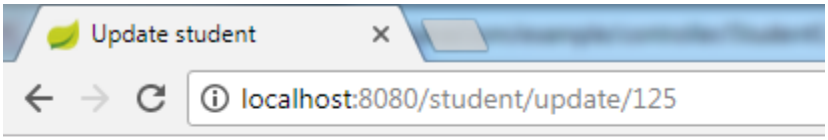
NPM = 234

Name = mon

GPA = 2.5

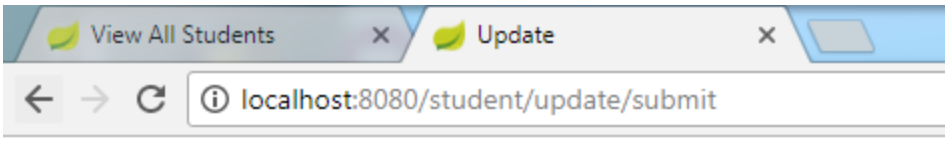
Delete Data
Update Data

Update Student

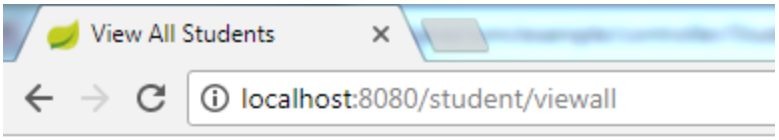


Problem Editor

NPM 125
Name nomu
GPA 3.3
Update



Data berhasil diperbaharui



All Students

No. 1

NPM = 125

Name = nomu

GPA = 3.3

Delete Data
Update Data

No. 2

NPM = 234

Name = mon

GPA = 2.5

Delete Data
Update Data

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?

   Jawab:

   Validasi input yang *optional* dan *required* dapat dilakukan dengan menambah method untuk validasi *request* jika menggunakan *object* sebagai parameter pada *form* POST. Validasi *form* perlu dilakukan pada view dan controller.

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 biasanya menggunakan POST method dibanding GET method karena method GET akan menampilkan semua *field* pada request URL. Keamanan data menggunakan POST method lebih aman dibanding GET method.

   Tidak perlu ada penanganan berbeda di header jika form post dikirim menggunakan method berbeda.

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

   Satu method mungkin saja menerima lebih dari satu jenis request dari GET sekaligus POST. Namun, lebih baik satu method menangani satu *request*.