

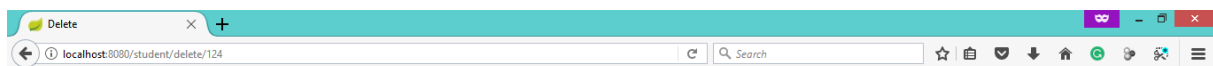
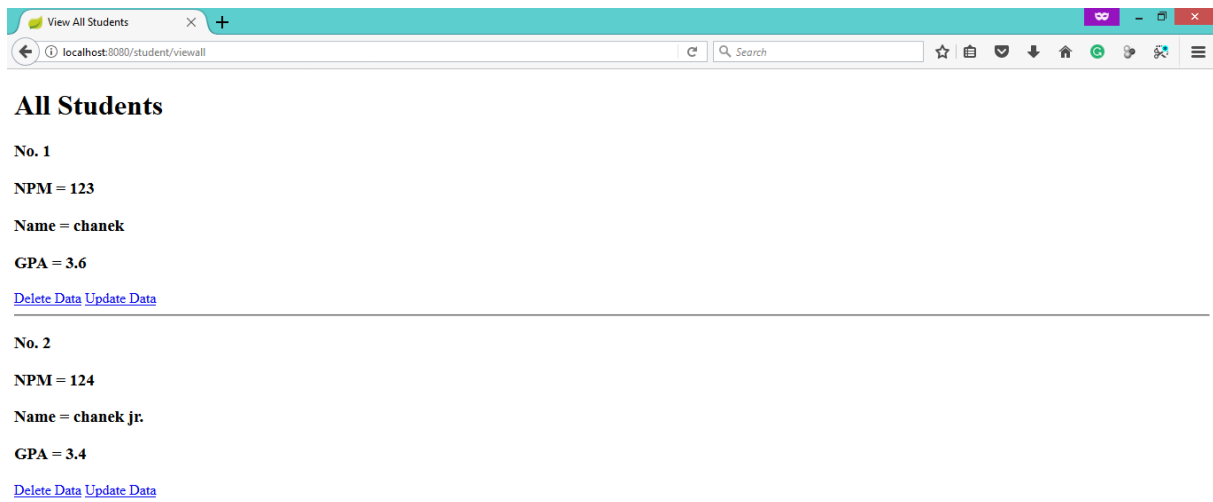
1. Kalau dalam kasus ini menurut saya tidak perlu karena sudah ada th:value yang memastikan jika field tidak ada yang kosong. Tetapi kalau memang ingin dibuat ada yang required, menurut yang saya temukan di https://www.tutorialspoint.com/spring/spring_required_annotation.htm ada syntaxnya sendiri.

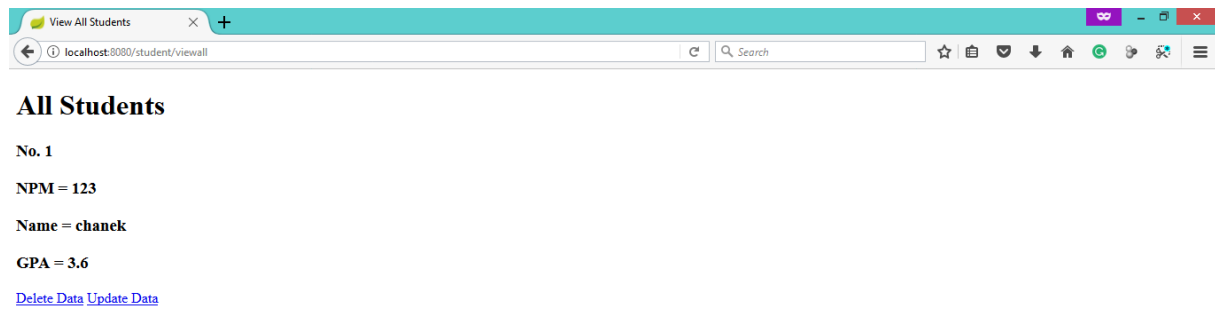
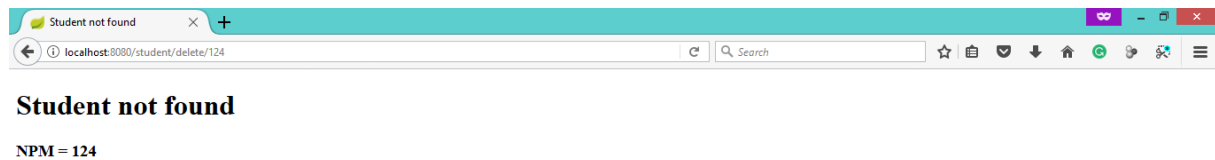
Misal :

```
@Required
public void setAge(Integer age) {
    this.age = age;
}
```

2. Form submit biasanya menggunakan method POST karena menurut saya lebih aman, apalagi jika datanya penting dan validasinya bisa lebih enak dibandingkan dengan yang method GET. Menurut saya masalah harus dibedakan atau tidak tergantung dengan kebutuhannya, misalkan tidak perlu diubah buat apa diubah.
3. Bisa, dalam satu form bisa sekaligus lebih dari satu request misal GET dan POST. Tapi kalau tidak ada kepentingannya sebaiknya tidak usah karena hanya membuat code jadi panjang

Latihan Delete





Class: StudentServiceDatabase

Code:

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

Class: StudentMapper

Code:

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

Class: StudentController

Code:

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

```

StudentModel students = studentDAO.selectStudent(npm);

if(students==null) {
    return "not-found";
}else {
    model.addAttribute ("students", students);
    studentDAO.deleteStudent (students.getNpm());

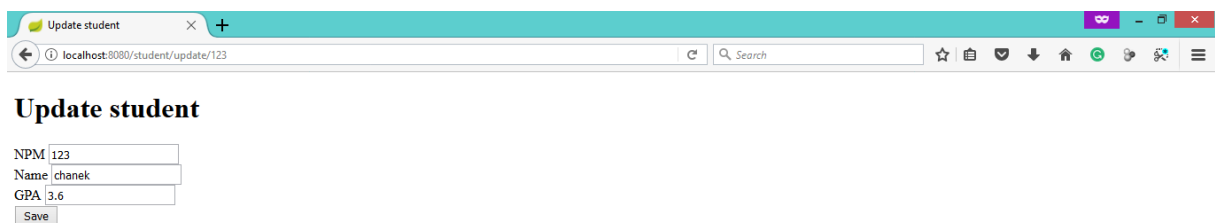
    return "delete";
}
}

```

Keterangan:

Jadi untuk method delete, akan menggunakan Model. Method akan menerima parameter npm. Setelah itu melakukan selectStudent. Akan dicek apakah student ada atau tidak. Kalau tidak ada maka akan diarahkan ke view not-found, kalau ada maka akan didelete dari database (anotasi @Delete). Setelah itu akan return view delete. Bisa dilihat di gambar pertama awalnya ada dua data. Setelah dihapus muncul bpesan data berhasil dihapus. Setelah itu jika kita mencoba delete lagi maka data tidak ditemukan dn mereturn view not-found.

Latihan Update Data



Update student

NPM 123

Name chanek

GPA 3.6

Save

Update student

localhost:8080/student/update/123

Search

Update student

NPM123

Namechanek

GPA3.8

Save

Update

localhost:8080/student/update/submit

Search

Data berhasil diperbarui

View All Students

localhost:8080/student/viewall

Search

All Students

No. 1

NPM = 123

Name = chanek

GPA = 3.8

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

Student not found

localhost:8080/student/update/150

Search

Student not found

NPM = 150

Class: StudentServiceDatabase

Code:

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

Class: StudentMapper

Code:

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

Class: StudentController

Code:

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

        if(students==null) {
            return "not-found";
        }else {
            model.addAttribute ("student", students);
            return "form-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";
    }
```

View: Form-Update

Code:

```

<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}" />
    </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>

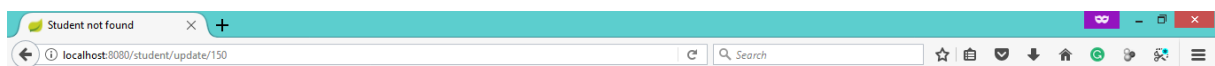
    <div>
        <button type="submit" name="action"
value="save">Save</button>
    </div>
</form>

```

Keterangan:

Mirip dengan method delete, method update menerima 3 param yaitu npm nama dan gpa. Setelah itu akan dicari dulu apakah studentnya ada. Kalau tidak ada return ke view not-found, kalau ada akan di return ke view form-update. Form ini pakai method post, jadi di form pada view form-update kodenya seperti yang tertera. Setelah di post maka akan update di database (Anotasi @Update di StudentMapper). Bisa dilihat di gambar awalnya chanek punya gpa 3.6 lalu diganti jadi 3.8. NPM tidak bisa diubah karena ada attribute readonly. Dapat dilihat juga jika npm tidak ditemukan akan return view not-found yang menyatakan bahwa npm tersebut tidak ditemukan.

Latihan Parameter Objek



Student not found

NPM = 150

View All Students

localhost:8080/student/viewall

Search

All Students

No. 1

NPM = 123

Name = chanek

GPA = 3.8

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

Update student

localhost:8080/student/update/123

Search

Update student

NPM

123

Name

chanek mar

GPA

3.8

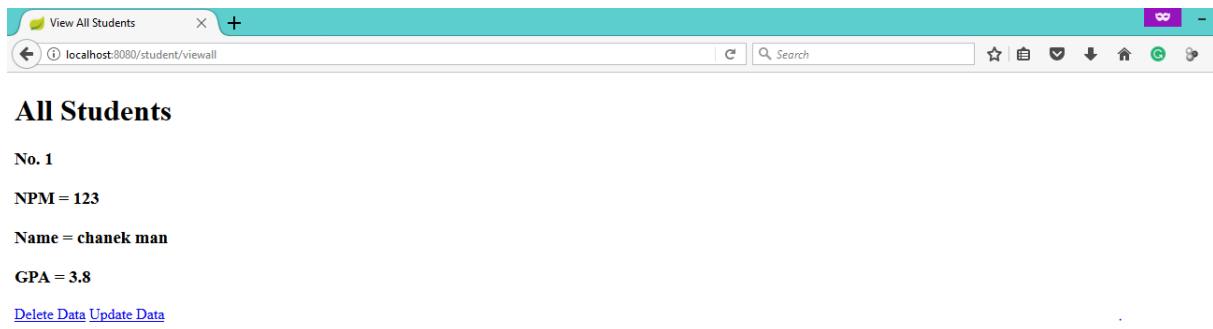
Save

Update

localhost:8080/student/update/submit

Search

Data berhasil diperbarui



Class: StudentServiceDatabase

Code:

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

Class: StudentMapper

Code:

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

Class: StudentController

Code:

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

        if(students==null) {
            return "not-found";
        }else {
            model.addAttribute("student", students);
            return "form-update";
        }
    }

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



```

{
    studentDAO.updateStudent (student);

    return "success-update";
}

```

View: Form-Update

Code:

```

<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>

```

Keterangan:

Mirip dengan method sebelumnya, tapi method update menerima 1 param yaitu StudentModel. Setelah itu akan dicari dulu apakah studentnya ada. Kalau tidak ada return ke view not-found, kalau ada akan di return ke view form-update. Form ini pakai method post, jadi di form pada view form-update kodenya seperti yang tertera. Setelah di post maka akan update di database (Anotasi @Update di StudentMapper). Disini juga beda, gaperlu bikin lagi new Object tapi langsung studentDAO dan ambil paramnya dari method. Beda juga di formnya karena jadi ada th:field Bisa dilihat di gambar awalnya chaneK tetapi jadi chaneK man. Dapat dilihat juga jika npm tidak ditemukan akan return view not-found yang menyatakan bahwa npm tersebut tidak ditemukan.