

Ringkasan:

Pada tutorial kali ini, saya belajar menggunakan logging menggunakan framework Lombok. Saya juga belajar menggunakan database untuk operasi-operasi update dan delete.

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.

Jawab: Validasi input dapat dilakukan dengan menggunakan `@Valid` dan `BindingRequest` yang akan melakukan pengecekan apabila terjadi error atau tidak. Untuk validasi dapat dilakukan dengan menggunakan `if` untuk menanggulangi setiap validasi yang diinginkan.

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: Karena jika menggunakan POST hasil dari form dapat muncul di link ketika di submit; Diperlukan penanganan menurut best practice.

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

Jawab: Tidak, karena method GET dan POST adalah 2 hal yang bertolak belakang.

Latihan Menambahkan Delete

Pada StudentMapper, menambahkan method untuk menghapus yaitu dengan menggunakan query untuk menghapus di database.

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

Pada StudentServiceDatabase ditambahkan method deleteStudent yang menambahkan log dan memanggil method deleteStudent yang ada di StudentMapper.

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

Pada Controller juga ditambahkan method delete yang akan mengambil value npm yang nantinya akan dihapus.

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

    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        return "not-found";
    }
}
```

Testing:

Add student

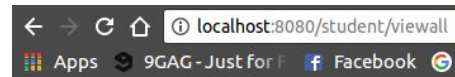


Problem Editor

NPM

Name

GPA



All Students

No. 1

NPM = 21345

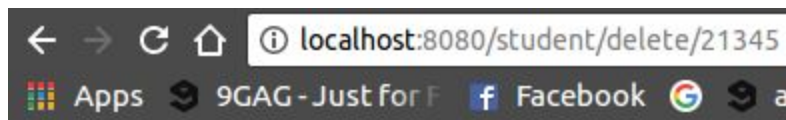
Name = Adi

GPA = 3.41

[Delete Data](#)

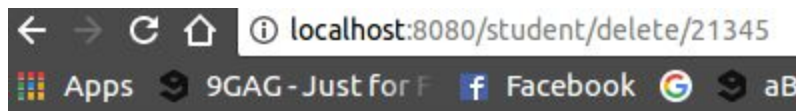
[Update Data](#)

Delete data



Data berhasil dihapus

Not found



Student not found

NPM = 21345

Latihan Menambahkan Update

Pada StudentMapper, menambahkan method untuk menrubah data dengan npm yang diberikan yaitu dengan menggunakan query untuk merubah di database.

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

Pada StudentServiceDatabase ditambahkan method updateStudent yang menambahkan log dan memanggil method updateStudent yang ada di StudentMapper.

```
@Override
public void updateStudent(StudentModel student) {

    log.info("student updated");
    studentMapper.updateStudent(student);
}
```

Pada Controller juga ditambahkan method update yang akan memanggil form update dan mengambil value npm yang nantinya akan dirubah tersebut. Ditambahkan juga method updateSubmit yang dipanggil ketika *user* mengklik tombol submit yang akan mengambil npm,name, dan juga gpa untuk nantinya diubah.

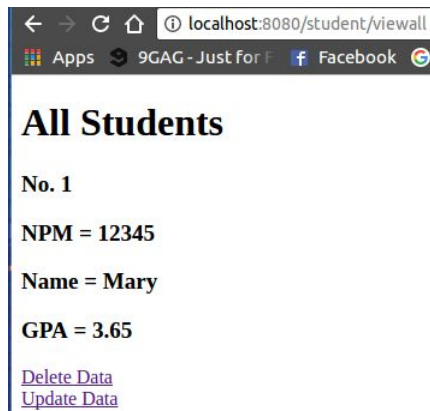
```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);
    if(student != null) {
        model.addAttribute ("student", student);
        return "form-update";
    }
    return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

@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";
}
```

Testing:

Viewall student



← → ↻ 🏠 ⓘ localhost:8080/student/viewall

Apps 9GAG - Just for F Facebook

All Students

No. 1


NPM = 12345

Name = Mary

GPA = 3.65

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

Isi form update



← → ↻ 🏠 ⓘ localhost:8080/student/update/12345

Apps 9GAG - Just for F Facebook

Problem Editor

NPM

Name

GPA

Data diubah



← → ↻ 🏠 ⓘ localhost:8080/student/update/submit

Apps 9GAG - Just for F Facebook aBY

Data berhasil diubah



← → ↻ 🏠 ⓘ localhost:8080/student/viewall

Apps 9GAG - Just for F Facebook

All Students

No. 1

NPM = 12345

Name = Mary

GPA = 3.32

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

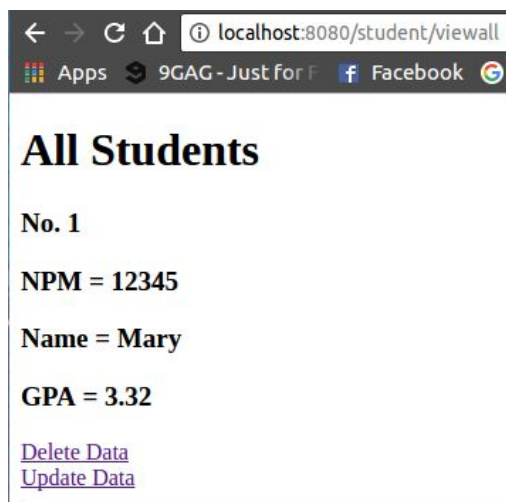
Latihan Menggunakan Object Sebagai Parameter

Pada controller method updateSubmit diganti menjadi menerima object student bukan menerima npm, name, gpa. Kemudian akan diupdate.

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

Testing:

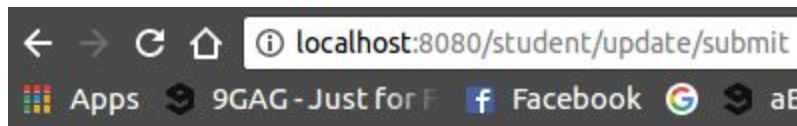
View all students



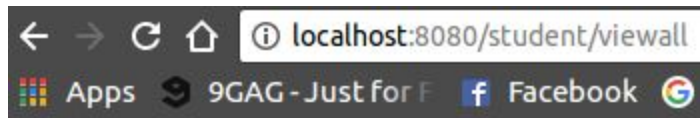
Merubah pada form



Data diubah



Data berhasil diubah



All Students

No. 1

NPM = 12345

Name = John

GPA = 3.32

[Delete Data](#)

[Update Data](#)
