

1. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase :

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

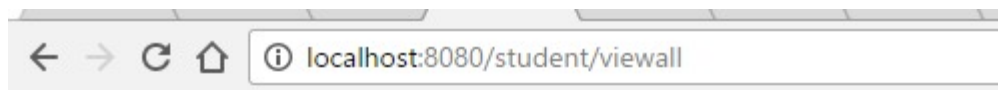
2. Melengkapi method delete di class StudentController :

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

Method Delete akan dipanggil berdasarkan NPM yang dipanggil dari mapping dan list Student yang ada, apabila NPM tidak ditemukan maka halaman akan merespond dengan not found.

Berikut SS dari latihan tersebut.

Viewall



All Students

No. 1

NPM = 1234

Name = Yogine

GPA = 4.0

[Delete User](#) [Update User](#)

No. 2

NPM = 456

Name = Dilan

GPA = 4.0

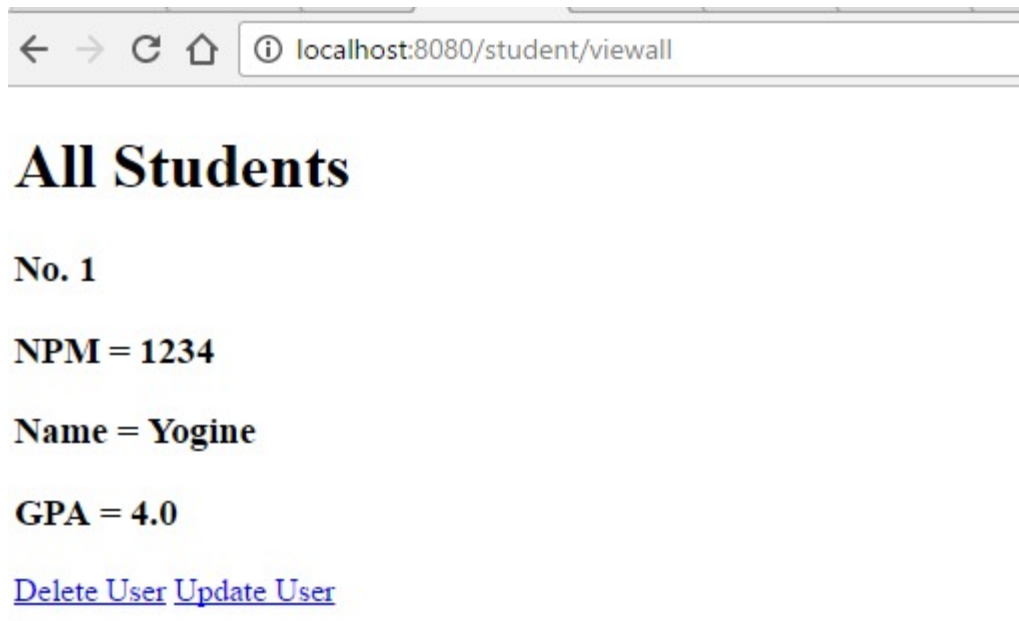
[Delete User](#) [Update User](#)

Delete berdasarkan NPM 456



Data berhasil dihapus

Jalankan View All setelah dihapus



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

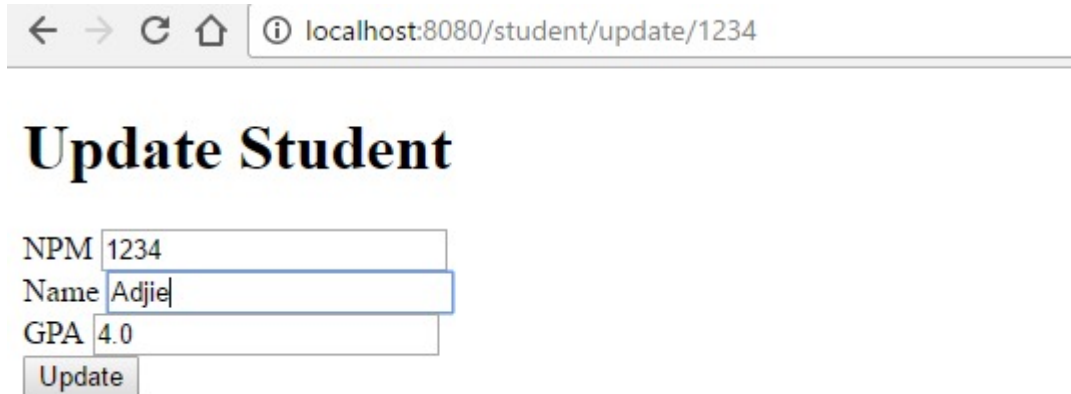
```
RequestMapping("/student/update/{npm}")  
public String update (Model model, @PathVariable(value = "npm") String npm)  
{  
    StudentModel student = studentDAO.selectStudent (npm);  
  
    model.addAttribute ("student", student);  
  
    return "form-update";  
}
```

Method update akan mengupdate data berdasarkan parameter NPM yang diberikan pada request mapping tersebut.

4. Tambahkan method **updateSubmit** pada class **StudentController**

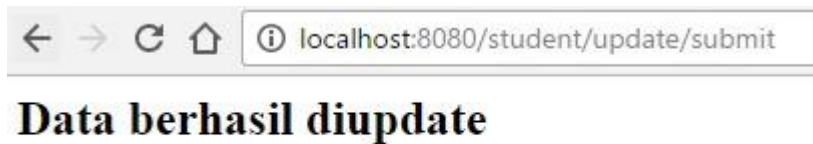
Method ini dipanggil saat tombol submit dan akan menggunakan parameter yang disediakan dan akan membuat object student baru dari parameter tersebut untuk memanggil method update.

Berikut SS dari latihan tersebut :



The screenshot shows a web browser window with the address bar displaying `localhost:8080/student/update/1234`. The main heading is **Update Student**. Below the heading, there is a form with three input fields: **NPM** with the value `1234`, **Name** with the value `Adjie`, and **GPA** with the value `4.0`. At the bottom of the form is an **Update** button.

Update Nama Yogine -> Adjie.



The screenshot shows a web browser window with the address bar displaying `localhost:8080/student/update/submit`. The main heading is **Data berhasil diupdate**.

Jalankan View all untuk melihat data berhasil terupdate

All Students

No. 1

NPM = 1234

Name = Adjie

GPA = 4.0

[Delete User](#) [Update User](#)

4. Latihan Objek sebagai parameter pada controller untuk menerima parameter saja :

```
public String updateSubmit (StudentModel model) {
```

```
    String res="success-update";
```

```
    StudentModel student = new StudentModel (model.getNpm(), model.getName(),  
model.getGpa());
```

```
    studentDAO.updateStudent (student);
```

```
    return res;
```

```
}
```

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 seperti jika menggunakan RequestParam? Apakah validasi diperlukan?

➔ Validasi dapat dilakukan di mode:

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method?

➔ Apabila kita menggunakan method GET, maka akan muncul data yang kita ambil di URL sehingga program kita akan tidak aman rentan dengan bad injection.

Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

➔ Lebih aman bila GET ditambahkan RequestParam untuk value yang akan dipakai.

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

➔ Bisa.