

## Latihan Menambahkan Delete

1. Pada viewall.html tambahkan

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

```
viewall.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <title>View All Students</title>
5   </head>
6   <body>
7     <h1>All Students</h1>
8
9     <div th:each="student, iterationStatus: ${students}">
10       <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
11       <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
12       <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
13       <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
14       <a th:href="/student/delete/" + ${student.npm}">Delete Data</a>
15     </div>
16   </body>
17 </html>
18
19
```

2. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**

```
@Mapper
public interface StudentMapper
{
    @Select("select npm, name, gpa from student where npm = #{npm}")
    StudentModel selectStudent (@Param("npm") String npm);

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

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

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

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

4. Lengkapi method **delete** pada class **StudentController**

```
@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 {
        model.addAttribute("npm", npm);
        return "not-found";
    }
}
```

5. Jalankan Spring Boot app dan lakukan beberapa insert

Insert npm 123



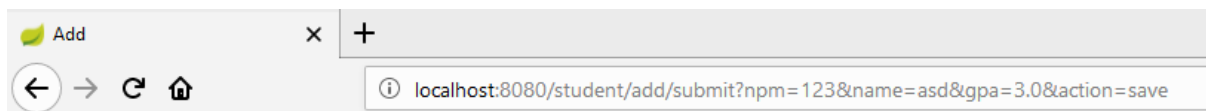
localhost:8080/student/ad

## Problem Editor

NPM

Name

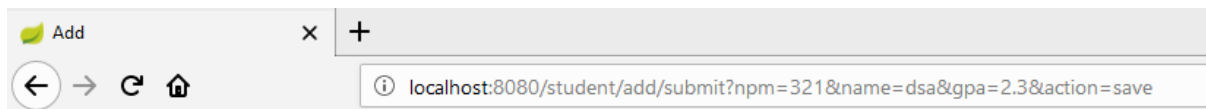
GPA



localhost:8080/student/add/submit?npm=123&name=asd&gpa=3.0&action=save

**Data berhasil ditambahkan**

Insert npm 321



localhost:8080/student/add/submit?npm=321&name=dsa&gpa=2.3&action=save

**Data berhasil ditambahkan**

Tampilan viewall.html

---

## All Students

**No. 1**

**NPM = 123**

**Name = asd**

**GPA = 3.0**

[Delete Data](#)

---

**No. 2**

**NPM = 124**

**Name = Chanek**

**GPA = 3.5**

[Delete Data](#)

---

**No. 3**

**NPM = 321**

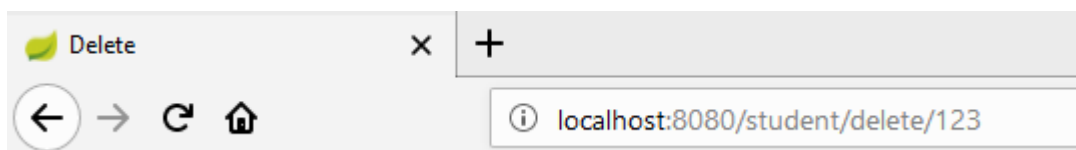
**Name = dsa**

**GPA = 2.3**

---

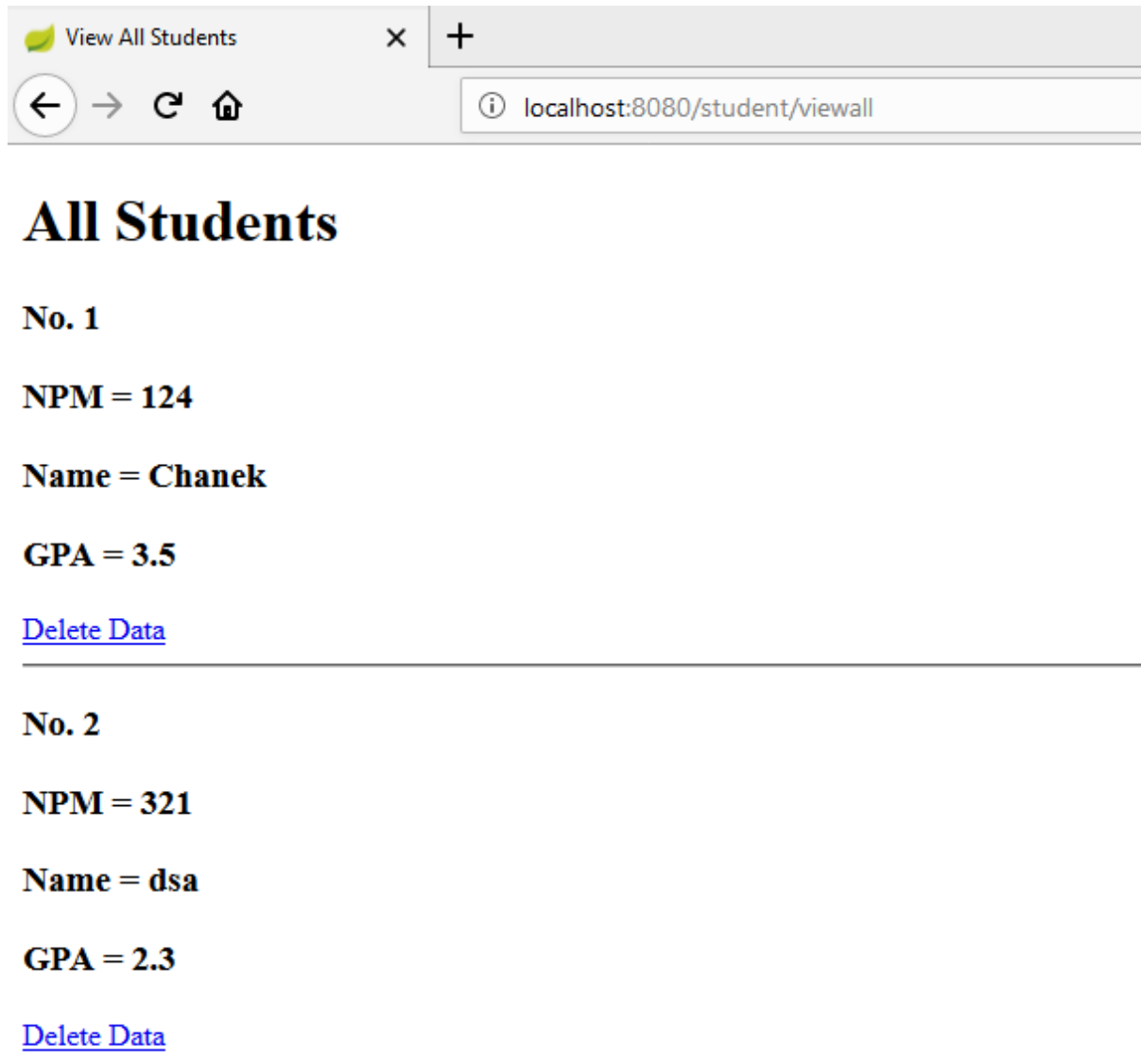
\*Terdapat chanek karena berbagi database dengan mahasiswa lain sehingga ketika viewall data tersebut masuk.

Delete npm 123

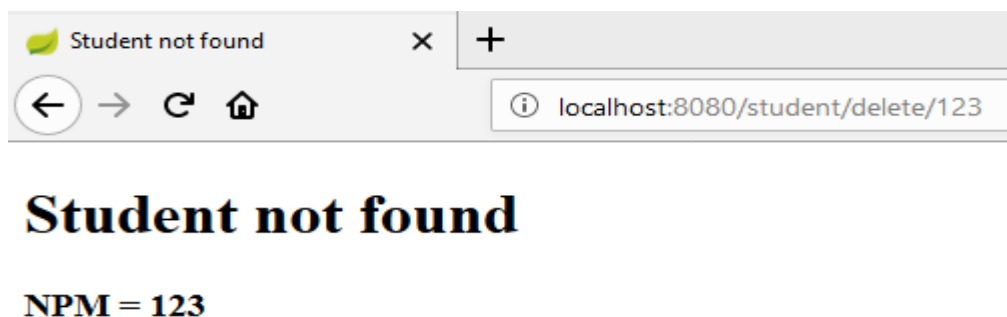


**Data berhasil dihapus**

Tampilan viewall.html setelah data dihapus.



Apabila npm yang akan dihapus tidak ditemukan.



## 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 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);  
}
```

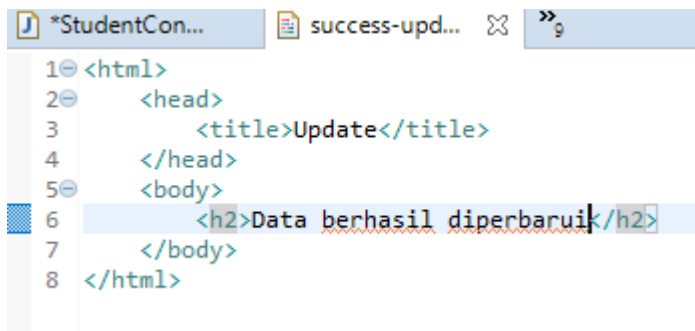
4. Tambahkan link Update Data pada **viewall.html**

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

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

```
<h1 class="page-header">Update Student</h1>  
  
<form action="/student/update/submit" method="post">  
    <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">Update</button>  
    </div>  
</form>  
  
</body>  
</html>
```

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



```
1 <html>
2   <head>
3     <title>Update</title>
4   </head>
5   <body>
6     <h2>Data berhasil diperbarui</h2>
7   </body>
8 </html>
```

7. Tambahkan method **update** pada class **StudentController**
8. Tambahkan method **updateSubmit** 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);
    if(npm.equals(null) || student==null) {
        model.addAttribute("npm", npm);
        return "not-found";
    }
    else {
        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";
}
```

9. Jalankan Spring Boot dan coba test program Anda

Tampilan viewall.html



## All Students

**No. 1**

**NPM = 125**

**Name = nom**

**GPA = 3.5**

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

---

**No. 2**

**NPM = 234**

**Name = mon**

**GPA = 2.5**

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

---

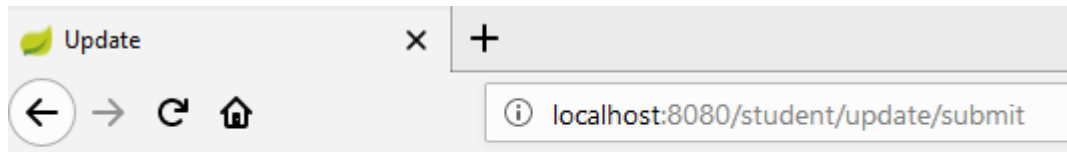
Update student dengan npm 125 dengan name menjadi nomu dan gpa menjadi 3.52



## Update Student

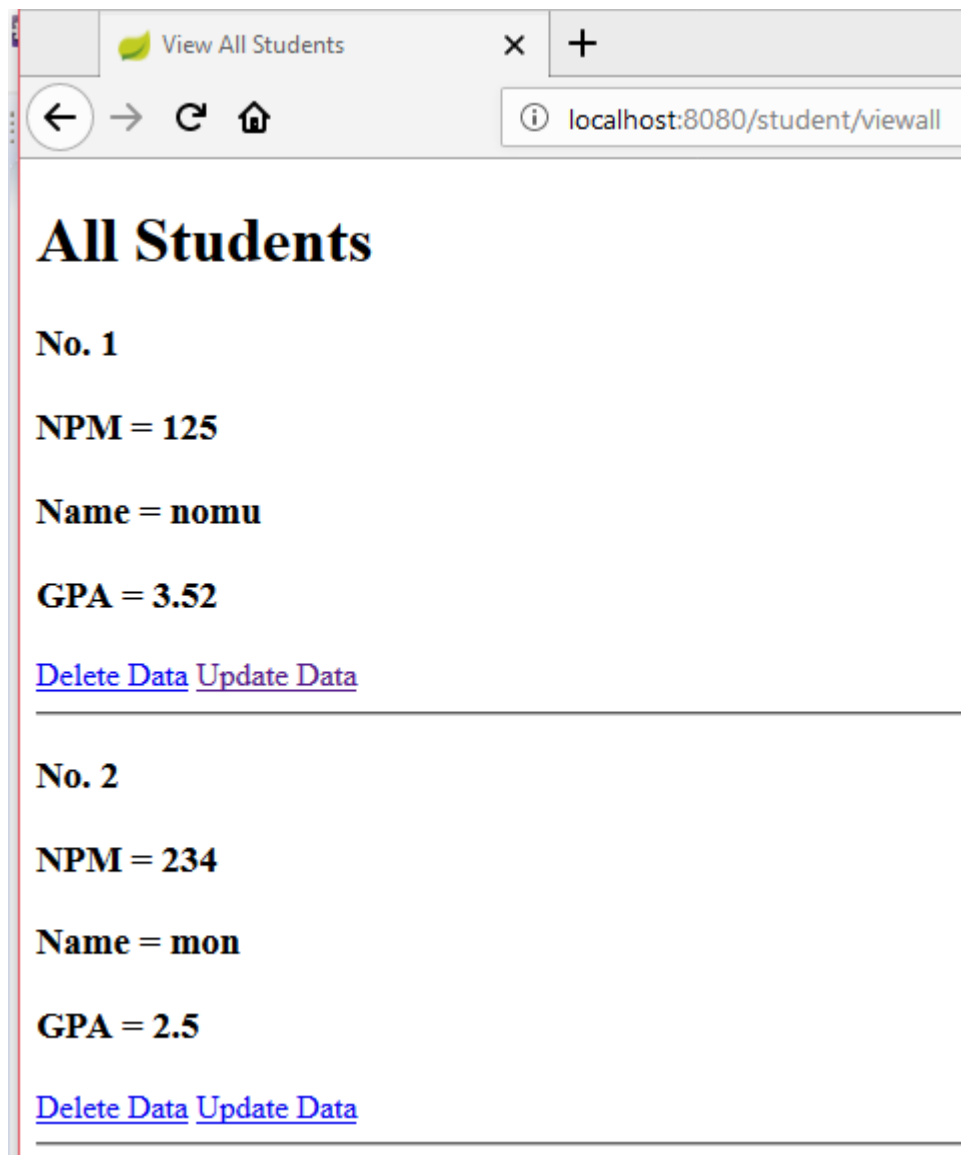
NPM	<input type="text" value="125"/>
Name	<input type="text" value="nom"/>
GPA	<input type="text" value="3.5"/>
<input type="button" value="Update"/>	

Data student berhasil diperbarui.



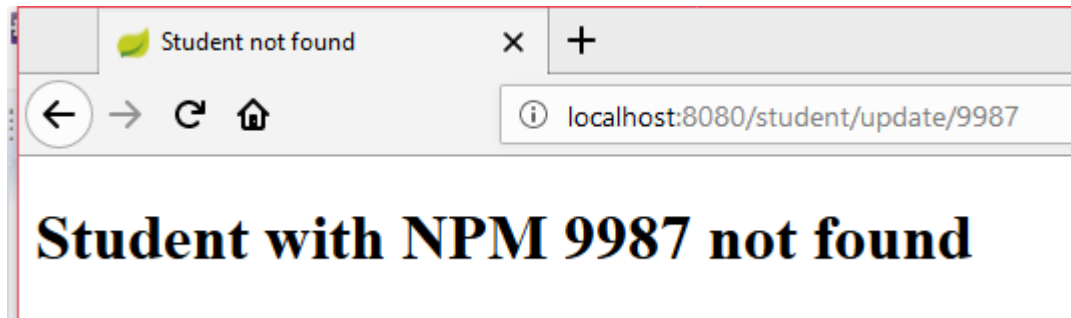
## Data berhasil diperbarui

Tampilan viewall.html setelah data diperbarui.





Apabila student dengan npm 9987 tidak ditemukan.



### Latihan Menggunakan Object sebagai Parameter

1. Menambahkan `th:object="${student}"` pada tag `<form>` di view dan menambahkan `th:field="*{[nama_field]}"` pada setiap input

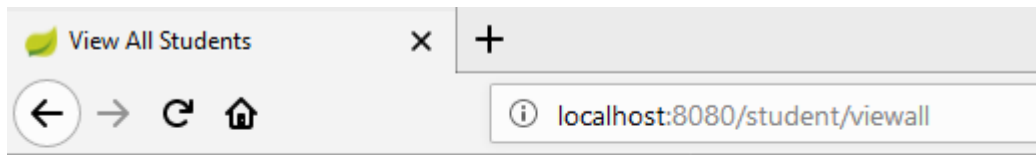
```
<h1 class="page-header">Update Student</h1>
<form th:object = "${student}" action="/student/update/submit" method="post">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:field= "${npm}" th:value = "${student.npm}" />
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" th:field= "${name}" th:value="${student.name}"/>
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" th:field= "${gpa}" th:value="${student.gpa}"/>
  </div>
  <div>
    <button type="submit" name="action" value="save">Update</button>
  </div>
</form>
```

2. Menambahkan `th:field="*{[nama_field]}"` pada setiap input

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

3. Tes lagi aplikasi Anda

Tampilan viewall.html



## All Students

**No. 1**

**NPM = 125**

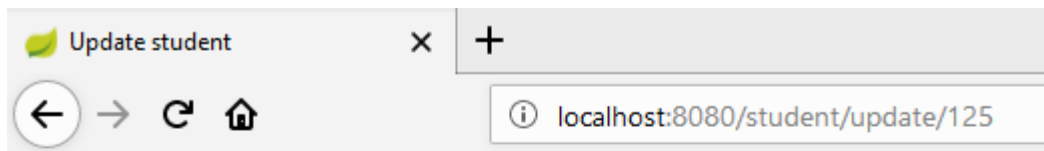
**Name = nomu**

**GPA = 3.52**

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

---

Update student dengan npm 125 dengan gpa menjadi 3.1

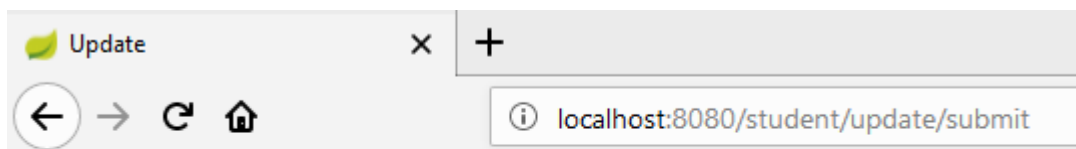


## Update Student

NPM

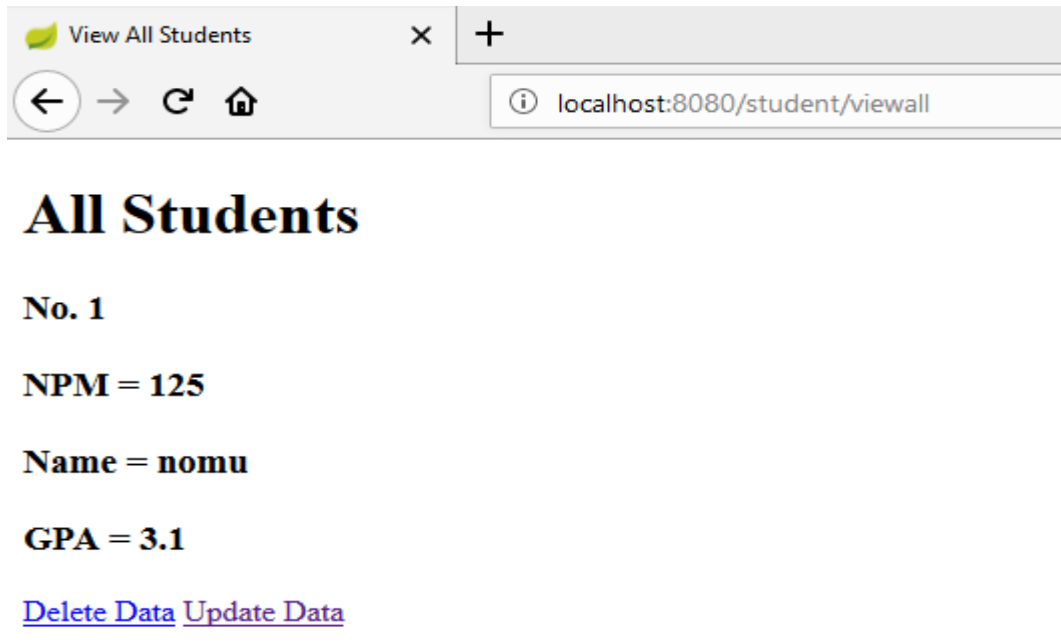
Name

GPA



**Data berhasil diperbarui**

Tampilan viewall.html setelah data diperbarui.



### 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? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

**Jawab :**

Untuk melakukan validasi sebelumnya harus dicek apakah property object null atau tidak, dapat dihandle oleh controllernya. Validasi diperlukan ketika mengecek apakah student.npm kosong sehingga input tidak bias ditinggalkan ketika kosong.

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method disbanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

**Jawab :**

GET method biasanya digunakan untuk mengambil data dalam database, sehingga form submit biasanya menggunakan POST method dibandingkan GET method. POST method bisa dikatakan lebih aman dan bisa mengubah database. Perlu penanganan apabila menggunakan method yang berbeda karena setiap method memiliki parameter berbeda.

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

**Jawab :**

Mungkin saja terjadi.

## Penjelasan Method Delete

- StudentController

```
@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 {
        model.addAttribute("npm", npm);
        return "not-found";
    }
}
```

Method delete mempunyai parameter npm. Setelah diinput dicek apakah object student memiliki npm yang diinput. Apabila studentDAO.selectStudent(npm) tidak menemukan npm maka object student menjadi null. Method tersebut mengembalikan object atau student tidak null maka method delete menjalankan method deleteStudent(npm) dan menampilkan halaman delete. Apabila student null maka method menampilkan halaman not-found.

- StudentMapper

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

Method query SQL yang menghapus student pada database dengan parameter npm dari controller.

## Penjelasan Method Update

- StudentController

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

Method update memiliki parameter npm. Parameter dimasukkan pada student. Apabila npm tersebut null yang artinya tidak ada npm yang dimasukkan atau student null yaitu npm tidak ada pada data maka akan menampilkan halaman not-found.html. Selain itu maka akan menampilkan halaman form-update yang terdapat form untuk mengubah nama dan gpa.

Untuk method updateSubmit dari parameter maka akan dibuat object student dan akan menjalankan method updateStudent(student). Kemudian method akan menampilkan halaman success-update.

- StudentMapper

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

Method query SQL yang meng-update student pada database dengan parameter StudentModel student dari controller.

### Penjelasan Method menggunakan Object sebagai Parameter

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

ModelAttribute harus digunakan apabila menggunakan object sebagai parameter. Kemudian memanggil method updateStudent(student) dan akan menampilkan halaman success-update.

### Ringkasan

Pada tutorial 4 latihan yang dilakukan yaitu menghubungkan springboot dengan database. Hal yang dilakukan yaitu menambahkan delete, menambahkan update, dan menggunakan object sebagai parameter.