

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

# MENGGUNAKAN DATABASE DAN MELAKUKAN DEBUGGING DALAM PROJECT SPRING BOOT

---

## A. HAL YANG DIPELAJARI

Dalam pengerjaan tutorial keempat mata kuliah Arsitektur dan Pemrograman Aplikasi Perusahaan, saya belajar mengenai penggunaan *database* dan melakukan *debugging* dalam *project* Spring Boot.

Pembelajaran awal pada tutorial keempat dikenalkan dengan *library* eksternal Lombok dan MyBatis serta melakukan instalasi kedua *library* eksternal tersebut pada Eclipse. Salah satu fungsi dari *library* Lombok yang dipelajari pada tutorial ini adalah sebagai *helper annotation* untuk *project* yang dibuat. *Library* MyBatis digunakan untuk menghubungkan MySQL dengan *project* yang dibuat dalam hal melakukan koneksi serta *generate query* dengan *helper annotation*.

Dalam pengerjaan tutorial sebelumnya data-data yang di-*input* hanya disimpan dalam bentuk *List* sehingga saat *program* dihentikan data akan hilang, maka pada tutorial keempat ini data disimpan dalam *database*. Keunggulan penggunaan *database* untuk mendukung pengerjaan tutorial keempat ini adalah data yang sudah dimasukkan dapat disimpan dalam *database* dan tetap ada walaupun program sudah dihentikan (*stop*). Operasi-operasi untuk melakukan modifikasi terhadap data dilakukan dengan menggunakan perintah-perintah *script* SQL. Di tutorial ini modifikasi data yang dipelajari adalah untuk melakukan *update* dan *delete* terhadap data. Selain itu, dalam tutorial ini juga mempelajari mengenai pemanfaatan *Object* sebagai *method parameter*.

Hal lain yang dipelajari adalah melakukan *debugging* dengan menggunakan *logging*. *Library* yang digunakan untuk melakukan *logging* adalah Slf4j yang ada pada *external library* Lombok dengan menggunakan anotasi *@Slf4j* yang diletakkan diatas *class* yang akan diberikan *log*. Penggunaan *logging* untuk melakukan *debugging* adalah cara yang lebih baik dibandingkan mencetak pesan pada *console* menggunakan *System.out.println* dan biasanya digunakan di *enterprise*.

Tidak hanya hal-hal baru yang saya pelajari dalam pengerjaan tutorial keempat. Mengulas kembali mengenai *database* dan penggunaan perintah-perintah SQL untuk memodifikasi data yang ada pada *database* juga dipelajari kembali.

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

### B. PERTANYAAN DAN JAWABAN

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.

#### Jawaban

⇒ Untuk melakukan validasi terhadap *input* yang *optional* maupun *input* yang *required* dapat diimplementasikan. Sesuai asumsi yang diberikan pertanyaan yaitu jika tidak menggunakan *attribute required* pada *input form*, validasi dapat dilakukan dengan membuat *class* baru yang *implements interface* Validator untuk melakukan validasi terhadap kondisi-kondisi tertentu. Cara lain untuk melakukan validasi adalah dengan memberikan anotasi seperti NotNull, Max, Min, Pattern pada *field* kemudian pada *parameter method* ditambahkan anotasi @Valid.

Validasi diperlukan dalam menangani kondisi-kondisi input tertentu yang diharapkan. Dengan begitu data-data yang tersimpan di *database* merupakan data-data yang sesuai.

Source:

- <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#validation-beanvalidation>
- <https://lmonkiewicz.com/programming/get-noticed-2017/spring-boot-rest-request-validation/>

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?

#### Jawaban

⇒ *Form* untuk *submit* data biasanya menggunakan *method* POST karena untuk menghindari ditampilkannya data yang di-*submit* pada *url* di *browser*. *Method* POST lebih aman ketika *submit* data-data yang sifatnya sensitive maupun bersifat *private* misalnya seperti *password*, nomor rekening, atau data pribadi lain yang dimasukkan di *input form* karena tidak akan ditampilkan pada *url*.

Ya, perlu adanya penanganan berbeda pada *header method* di *controller*. Karena dalam anotasi @RequestMapping perlu *method* yang spesifik contohnya RequestMethod.POST atau RequestMethod.GET.

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

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

### Jawaban

⇒ Memungkinkan jika satu *method* menerima lebih dari satu jenis *request method*. Pada *parameter* anotasi `@RequestMapping` value *method* dibuat menjadi `{RequestMethod.POST, RequestMethod.GET}`.

Untuk kasus di tutorial 4 ini misalnya seperti ini :

```
@RequestMapping(value = "/student/update/submit", method = {  
    RequestMethod.POST, RequestMethod.GET } )
```

Source :

- <https://stackoverflow.com/questions/17987380/combine-get-and-post-request-methods-in-spring>

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

### C. PENJELASAN *METHOD* YANG DIBUAT

#### 1. *Method* pada Latihan Menambahkan DELETE

Menambahkan *method* deleteStudent yang ada di *class* StudentMapper. Berikut *screenshot* dari *method* deleteStudent pada *class* StudentMapper.

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

*Method* deleteStudent menerima parameter npm dengan tipe data String. Terdapat anotasi @Delete untuk *method* ini dalam melakukan modifikasi data pada *database*.

Kemudian melengkapi *method* deleteStudent yang ada di *class* StudentServiceDatabase. Pada *method* deleteStudent di *class* StudentServiceDatabase, meng-*Override* dari *interface* StudentService. *Method* ini menerima *parameter* npm bertipe data String yang akan digunakan menjadi parameter saat memanggil *method* deleteStudent di *class* StudentMapper yang menjalankan perintah SQL. Berikut ini *screenshot* dari *method* deleteStudent pada *class* StudentServiceDatabase :

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

Selanjutnya, melengkapi *method* delete pada *controller* yaitu di *class* StudentController. Pada *method* delete ini akan dilakukan pengecekan apakah variabel student untuk *Object* StudentModel berdasarkan npm yang di-*input* apakah terdapat *student* dengan npm tersebut. Pertama-tama menggunakan *method* selectStudent(npm) untuk mencari *student* berdasarkan npm. Jika tidak ada maka akan addAttribute sesuai dengan npm yang dimasukkan lalu mengembalikan *view* not-found.html. Jika *student* dengan npm yang dimasukkan ada, maka akan dilakukan penghapusan (*delete*) *student* tersebut dari *database* melalui *method* deleteStudent (npm). Berikut ini *screenshot* dari *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";
    }
}
```

## WRITE-UP TUTORIAL 4 ADPAP

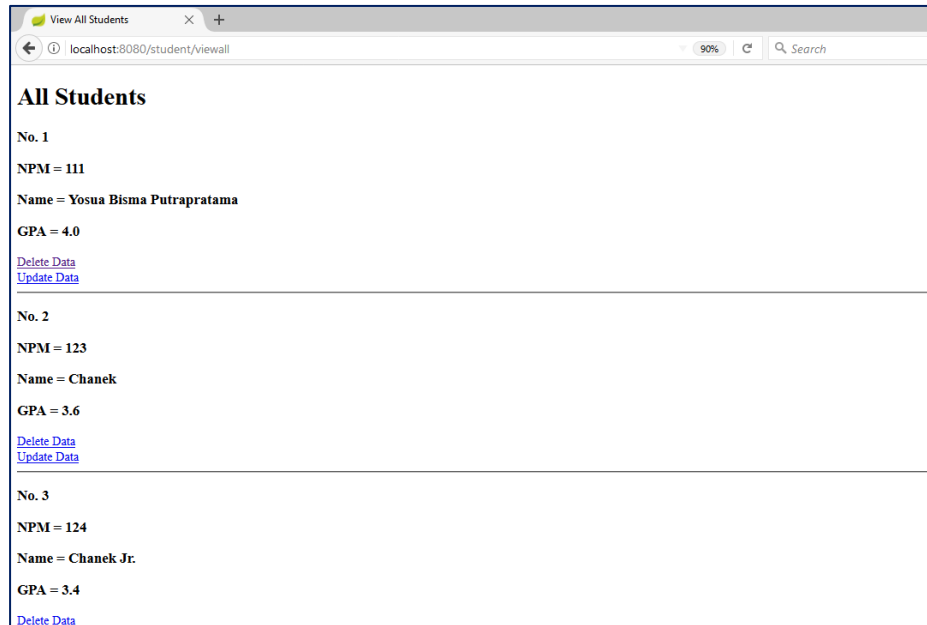
Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

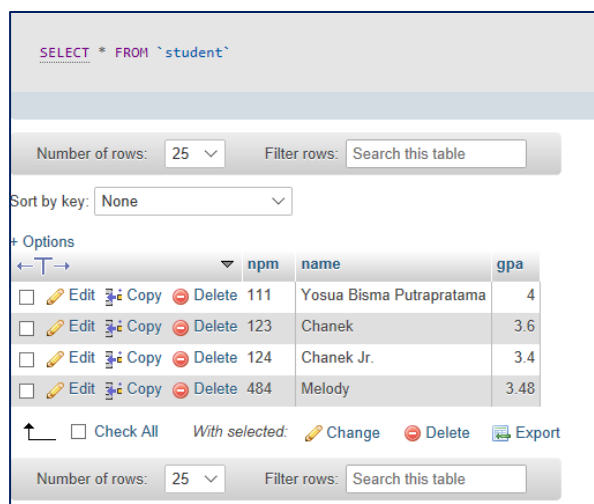
Contoh penerapan *method* delete saat program dijalankan (*run*).

Data Student yang sudah ditambahkan



All Students			
No. 1	NPM = 111	Name = Yosua Bisma Putrapratama	GPA = 4.0
<a href="#">Delete Data</a> <a href="#">Update Data</a>			
No. 2	NPM = 123	Name = Chanek	GPA = 3.6
<a href="#">Delete Data</a> <a href="#">Update Data</a>			
No. 3	NPM = 124	Name = Chanek Jr.	GPA = 3.4
<a href="#">Delete Data</a>			

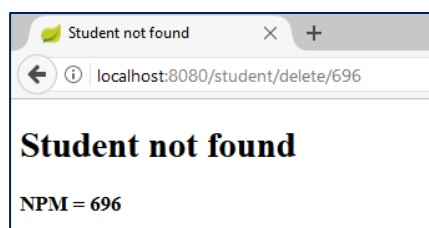
Dilihat pada phpmyadmin adalah sebagai berikut



```
SELECT * FROM `student`
```

	npm	name	gpa
<input type="checkbox"/> Edit Copy Delete	111	Yosua Bisma Putrapratama	4
<input type="checkbox"/> Edit Copy Delete	123	Chanek	3.6
<input type="checkbox"/> Edit Copy Delete	124	Chanek Jr.	3.4
<input type="checkbox"/> Edit Copy Delete	484	Melody	3.48

Kemudian coba dilakukan delete terhadap npm 696 yang tidak ada di *database*.



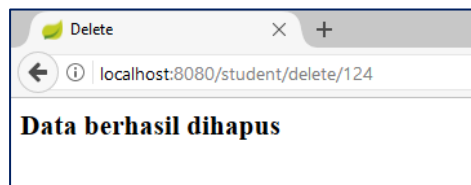
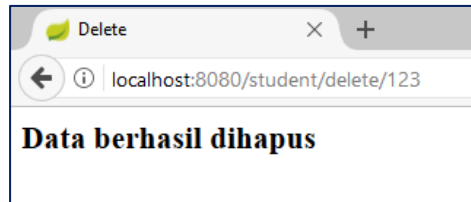
#### WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

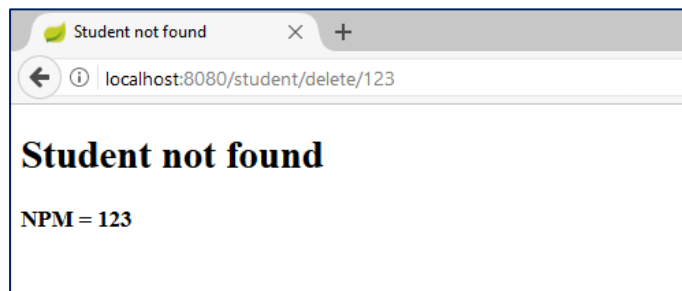
NPM : 1506689622

Kelas : ADPAP – B

Selanjutnya mencoba melakukan *delete* terhadap npm 123 dan 124

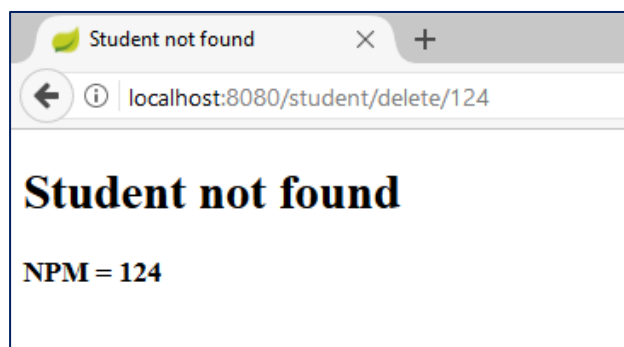


Kemudian mencoba melakukan *delete* kembali terhadap npm 123.



*Student* dengan npm 123 sudah tidak ada lagi sehingga tidak ditemukan dalam *database*.

Mencoba melakukan hal yang sama yaitu *delete* kembali terhadap npm 124



Sama halnya dengan npm 123, npm 124 yang sudah tidak ada lagi di *database* tidak dapat ditemukan.

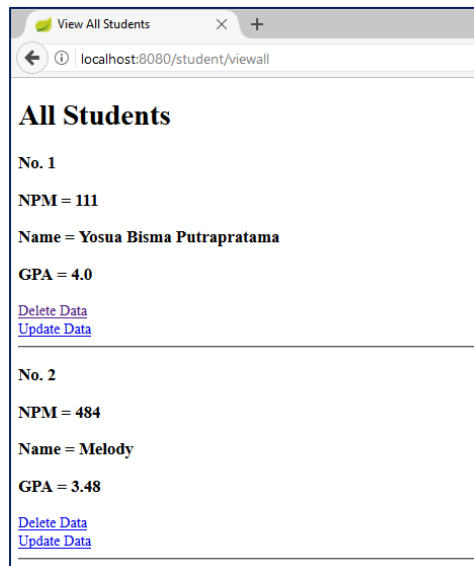
## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

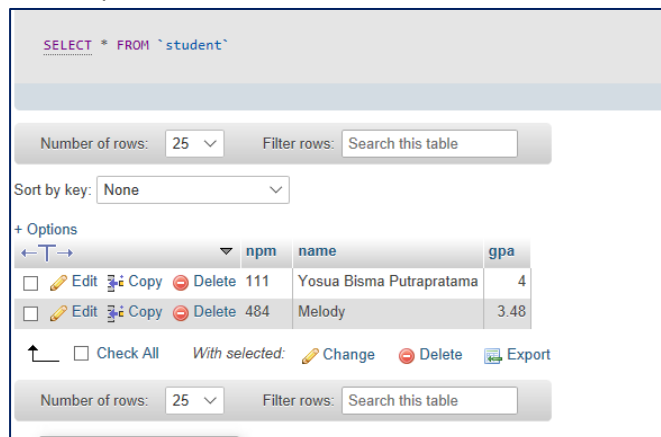
Kelas : ADPAP – B

View kembali semua data yang ada pada *database*



All Students			
No. 1	NPM = 111	Name = Yosua Bisma Putrapratama	GPA = 4.0
<a href="#">Delete Data</a> <a href="#">Update Data</a>			
No. 2	NPM = 484	Name = Melody	GPA = 3.48
<a href="#">Delete Data</a> <a href="#">Update Data</a>			

Lihat kembali pada phpmyadmin data *student* yang tersisa setelah dilakukan *delete* terhadap dua data *student*



```
SELECT * FROM `student`
```

	npm	name	gpa
<input type="checkbox"/> Edit Copy Delete	111	Yosua Bisma Putrapratama	4
<input type="checkbox"/> Edit Copy Delete	484	Melody	3.48

☐ Check All With selected: [Change](#) [Delete](#) [Export](#)

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

### 2. *Method* pada Latihan Menambahkan UPDATE

Menambahkan *method* `updateStudent` yang ada di *class* `StudentMapper`. Berikut *screenshot* dari *method* `updateStudent` pada *class* `StudentMapper`.

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

*Method* `updateStudent` menerima parameter *object* `StudentModel`. Terdapat anotasi `@Update` untuk *method* ini dalam melakukan modifikasi data pada *database* berdasarkan NPM.

Menambahkan *method* `updateStudent` pada *class* `StudentService`

```
void updateStudent (StudentModel student);
```

Kemudian menambahkan implementasi *method* `updateStudent` di *class* `StudentServiceDatabase`. Pada *method* `updateStudent` di *class* `StudentServiceDatabase`, meng-*Override* dari *interface* `StudentService`. *Method* ini menerima parameter *object* `StudentModel` yang akan digunakan menjadi parameter saat memanggil *method* `updateStudent` dari *class* `StudentMapper` yang menjalankan perintah SQL. Berikut ini *screenshot* dari *method* `updateStudent` pada *class* `StudentServiceDatabase` :

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

*Method* berikutnya yang ditambahkan adalah `update` pada *class* `StudentController` untuk mengakses *form input* untuk *update*. *Path variable* menerima input npm pada url yang nantinya digunakan untuk mencari dengan *method* `selectStudent` apakah ada *student* dengan npm tersebut. Jika ada maka akan *addAttribute* *object* `StudentModel` yang berisi name, npm, gpa dan mengembalikan `form-update.html`. Jika tidak ada maka akan *addAttribute* npmnya dan mengembalikan `not-found.html`. Berikut ini *screenshot* dari *method* `update` pada *class* `StudentController` :

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



## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

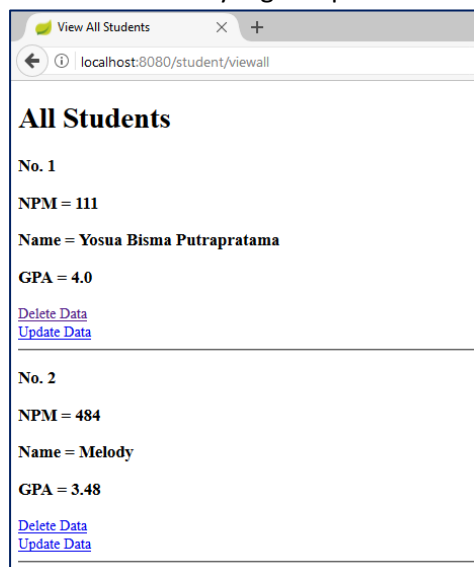
Kelas : ADPAP – B

Setelah menambahkan *method* update, pada *class* StudentController juga ditambahkan *method* updateSubmit untuk men-submit pembaharuan data ke *database*. Tugas dari *method* ini adalah menjalankan *method* updateStudent dengan parameter *object* StudentModel yang dibuat didalamnya. Kemudian mengembalikan *view* success-update. Data-data diambil dari *input form* yang ditampilkan pada *view* form-update dengan RequestMethod.POST.

```
@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";
}
```

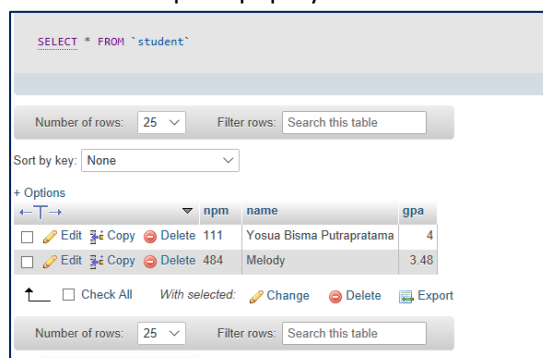
Contoh penerapan *method* update saat program dijalankan (*run*).

View semua data yang ada pada *database* saat ini.



No.	NPM	Name	GPA	Actions
No. 1	NPM = 111	Name = Yosua Bisma Putrapratama	GPA = 4.0	<a href="#">Delete Data</a> <a href="#">Update Data</a>
No. 2	NPM = 484	Name = Melody	GPA = 3.48	<a href="#">Delete Data</a> <a href="#">Update Data</a>

Lihat kembali pada phpmyadmin data *student*.



	npm	name	gpa
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	111	Yosua Bisma Putrapratama	4
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	484	Melody	3.48

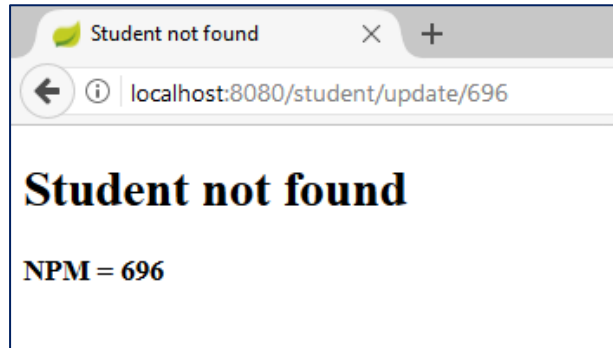
## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

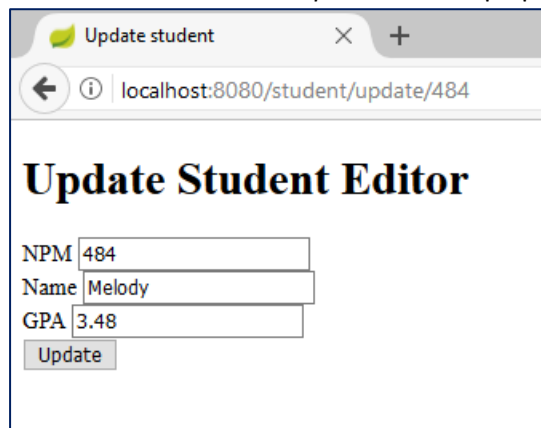
Kelas : ADPAP – B

Mencoba melakukan *update* terhadap npm 696 yang tidak ada di data *student*.

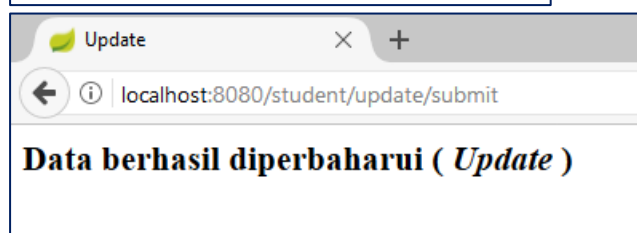
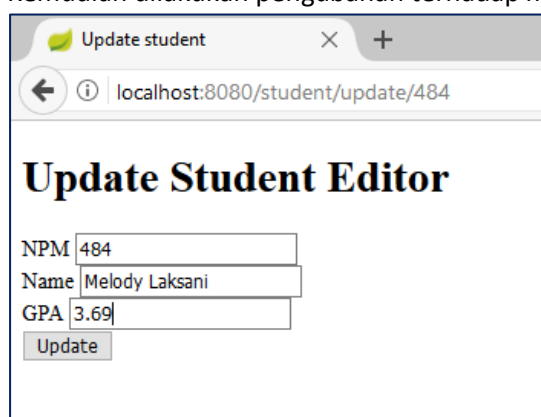


Melakukan *update* terhadap npm 484

Data sebelum dilakukan *update* terhadap npm 484.



Kemudian dilakukan perubahan terhadap *name* dan GPA, lalu klik *button* Update



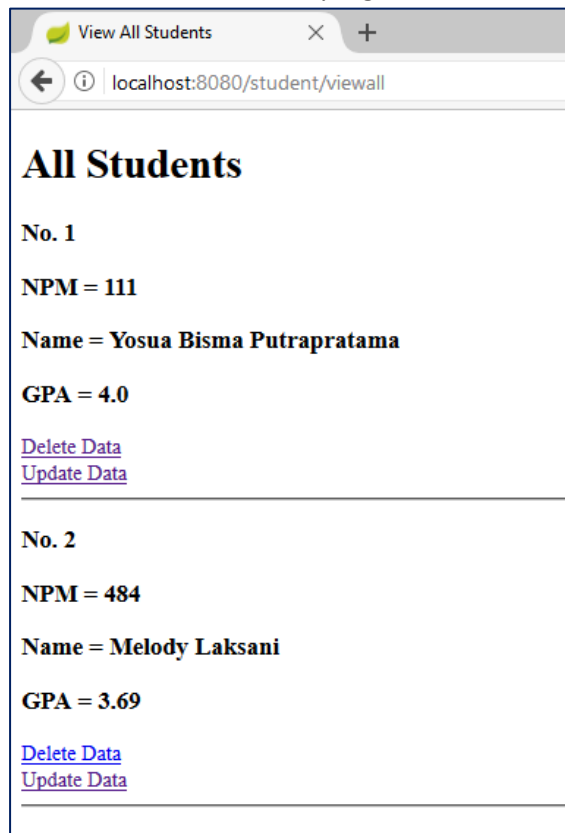
## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

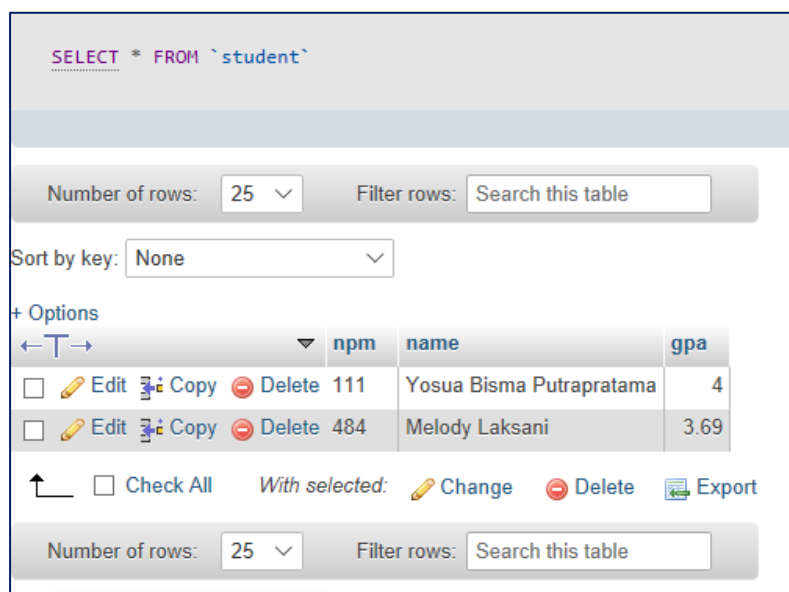
Kelas : ADPAP – B

View kembali semua data yang ada



All Students				
No. 1	NPM = 111	Name = Yosua Bisma Putrapratama	GPA = 4.0	<a href="#">Delete Data</a> <a href="#">Update Data</a>
No. 2	NPM = 484	Name = Melody Laksani	GPA = 3.69	<a href="#">Delete Data</a> <a href="#">Update Data</a>

Lihat kembali pada phpmyadmin data *student* setelah dilakukan *update*.



SELECT \* FROM `student`

Number of rows: 25 Filter rows: Search this table

Sort by key: None

+ Options

		npm	name	gpa
<input type="checkbox"/>	Edit Copy Delete	111	Yosua Bisma Putrapratama	4
<input type="checkbox"/>	Edit Copy Delete	484	Melody Laksani	3.69

Check All With selected: Change Delete Export

Number of rows: 25 Filter rows: Search this table

Name dan GPA untuk *student* dengan npm 484 sudah diperbaharui.

## WRITE-UP TUTORIAL 4 ADPAP

Nama : Yosua Bisma Putrapratama

NPM : 1506689622

Kelas : ADPAP – B

### 3. *Method* pada Latihan Menggunakan OBJECT Sebagai Parameter

Langkah pertama yang dilakukan adalah menambahkan `th:object="${student}"` pada `tag <form>` serta tambahkan `th:field="**{npm}"`, `th:field="**{name}"`, `th:field="**{gpa}"` yang terdapat di `view form-update`. Berikut ini *screenshot* dari `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">Update Student Editor</h1>

<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">Update</button>
</div>
</form>
</body>
</html>
```

Attribute `th:object` yang nantinya akan melakukan *passing* value dari *input form* menjadi variabel-variabel yang ada pada *object student*.

Kemudian ubah *method* `updateSubmit` pada `StudentController` menjadi hanya menerima parameter *object* dengan menggunakan anotasi `@ModelAttribute`. *Request mapping* tidak perlu diubah.

Nantinya melalui *method* `updateStudent` akan menerima parameter *object* tersebut dan melakukan *update* terhadap *student* berdasarkan input yang diberikan pada *form* serta mengembalikan *view success-update*.

Dengan parameter *object* ini kita tidak perlu mengambil value dari *request parameter* dan membentuk *object* di dalam *method*. Kita bisa langsung membentuk *object* di *parameter method*.

```
public String updateSubmit(@ModelAttribute StudentModel student)
{
    studentDAO.updateStudent (student);
    return "success-update";
}
```