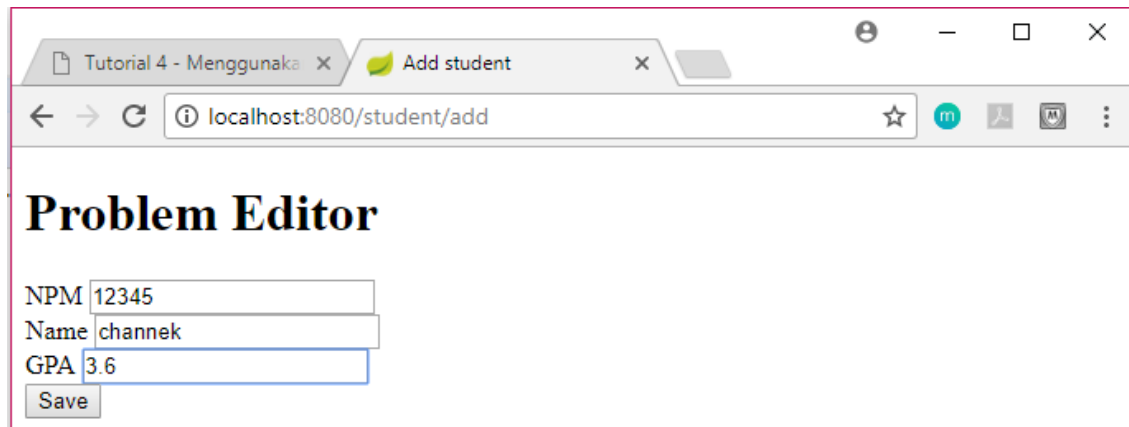


Atikah Luthfiana  
1506689250  
APAP B  
Tutorial 4  
**Tutorial**

1. Latihan menambahkan *delete*

```
viewall.html StudentServi... StudentCont... StudentServi... form-add.html
66 public String view(Model model) {
67     List<StudentModel> students = studentDAO.selectAllStudents();
68     model.addAttribute("students", students);
69
70     return "viewall";
71 }
72
73 @RequestMapping("/student/delete/{npm}")
74 public String delete(Model model, @PathVariable(value = "npm") String npm) {
75     StudentModel student = studentDAO.selectStudent(npm);
76
77     if (student != null) {
78         studentDAO.deleteStudent(npm);
79         return "delete";
80     } else {
81         model.addAttribute("npm", npm);
82         return "not-found";
83     }
84 }
85 }
```

- Menambahkan *student* baru



Tutorial 4 - Menggunakan x Add student x

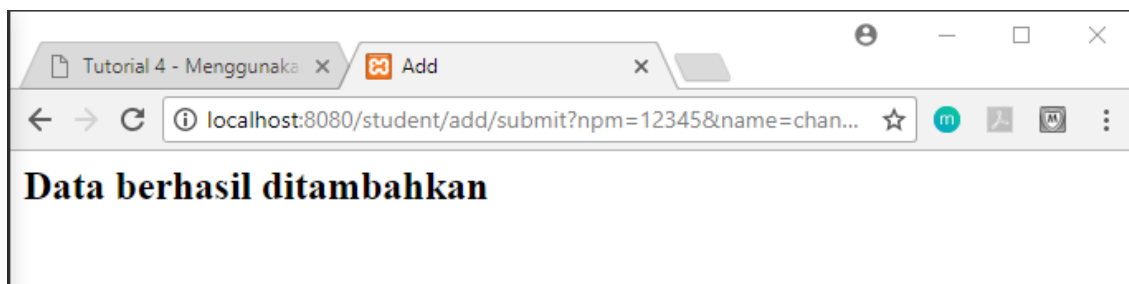
localhost:8080/student/add

## Problem Editor

NPM

Name

GPA



Tutorial 4 - Menggunakan x Add x

localhost:8080/student/add/submit?npm=12345&name=chan...

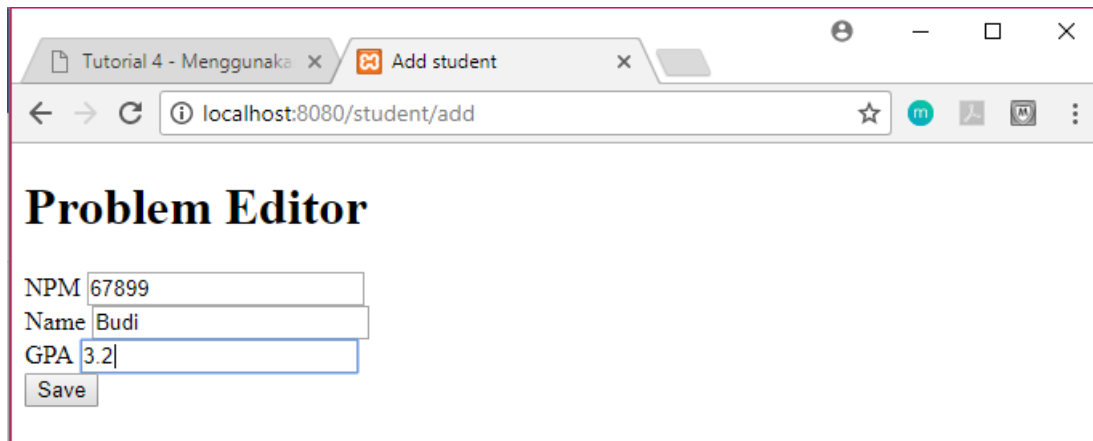
## Data berhasil ditambahkan

Atikah Luthfiana

1506689250

APAP B

Tutorial 4



Tutorial 4 - Menggunakan x Add student x

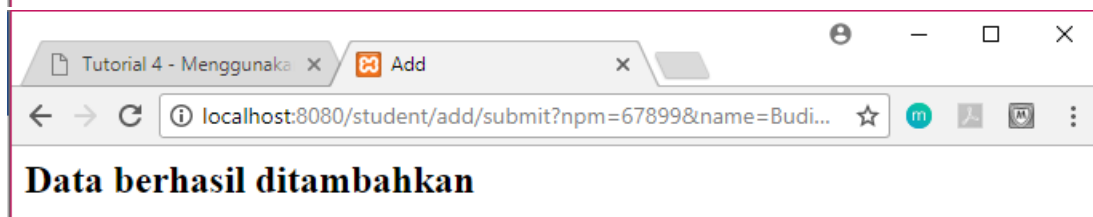
localhost:8080/student/add

## Problem Editor

NPM

Name

GPA

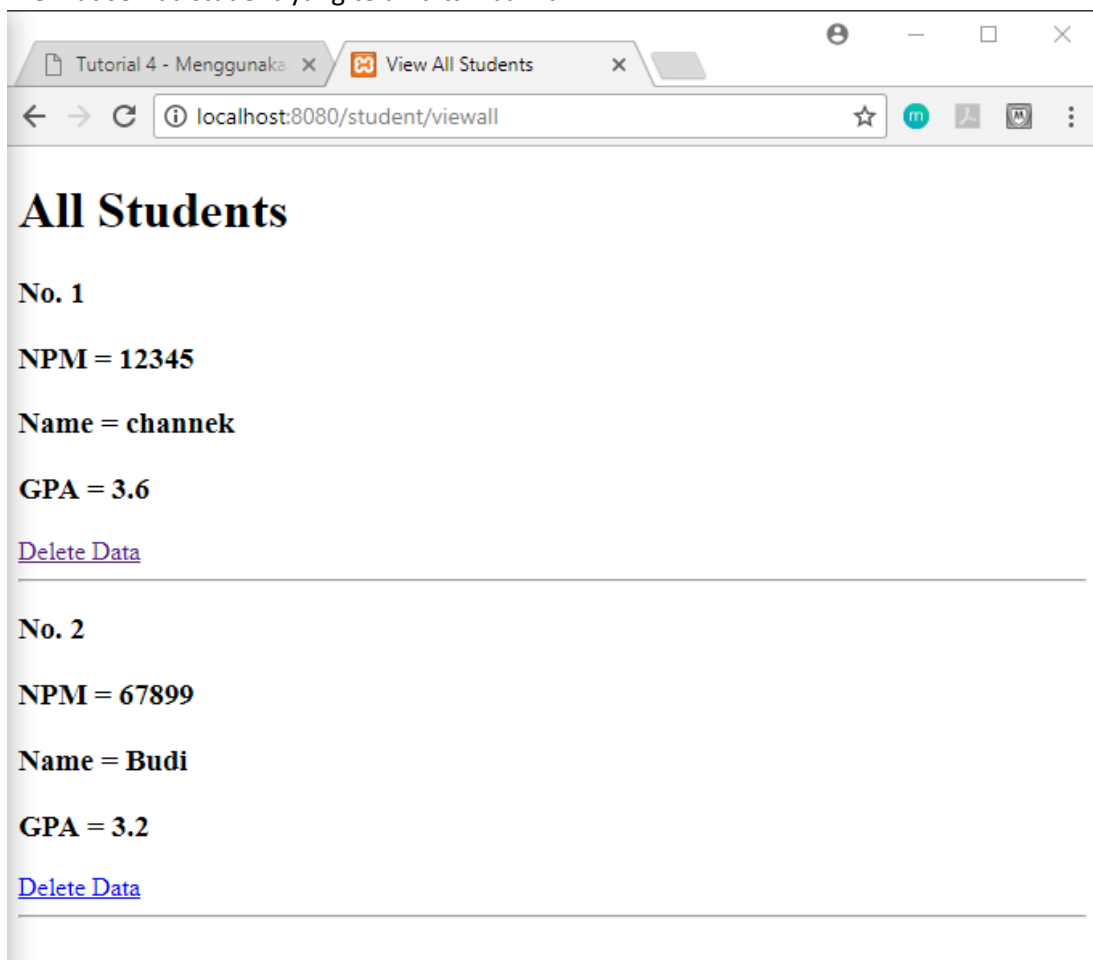


Tutorial 4 - Menggunakan x Add x

localhost:8080/student/add/submit?npm=67899&name=Budi...

## Data berhasil ditambahkan

- Melihat semua *student* yang telah ditambahkan



Tutorial 4 - Menggunakan x View All Students x

localhost:8080/student/viewall

## All Students

**No. 1**

**NPM = 12345**

**Name = channek**

**GPA = 3.6**

[Delete Data](#)

---

**No. 2**

**NPM = 67899**

**Name = Budi**

**GPA = 3.2**

[Delete Data](#)

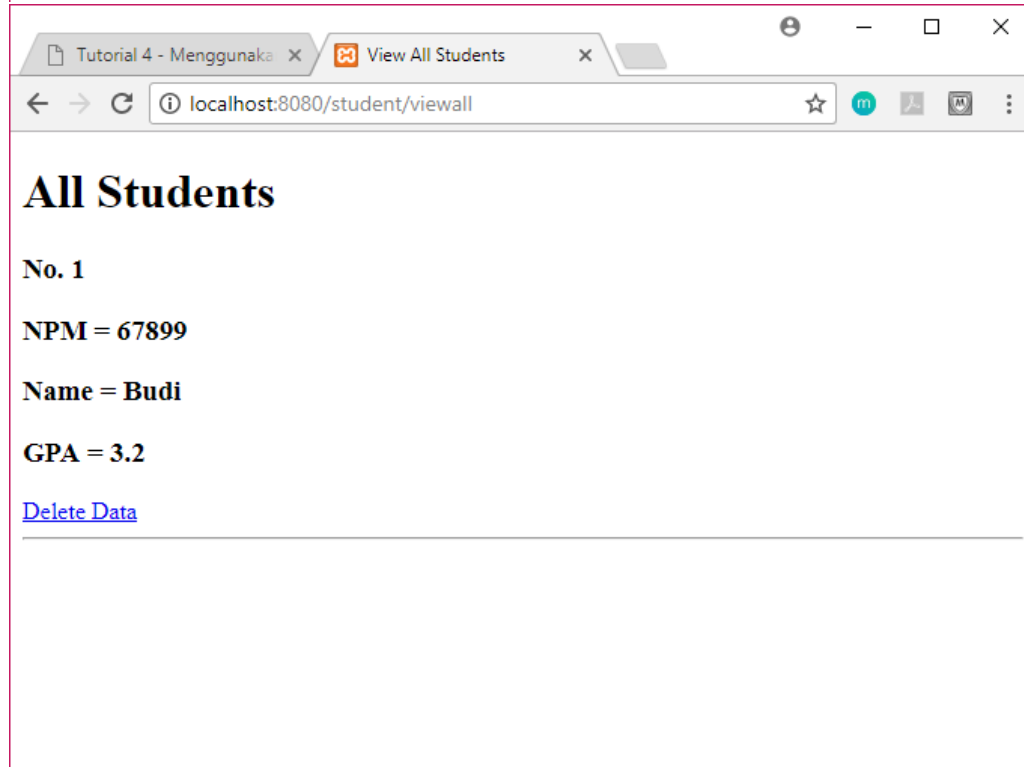
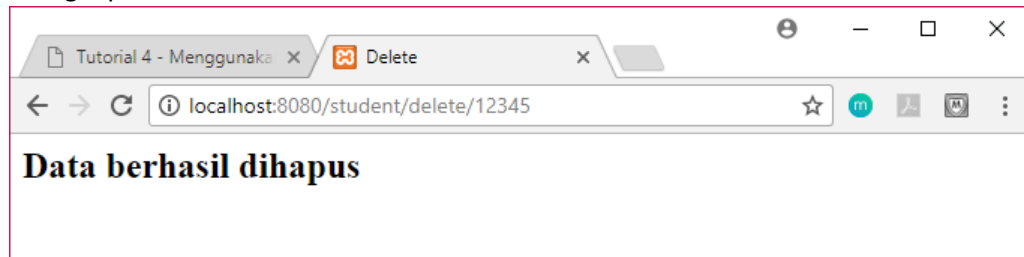
Atikah Luthfiana

1506689250

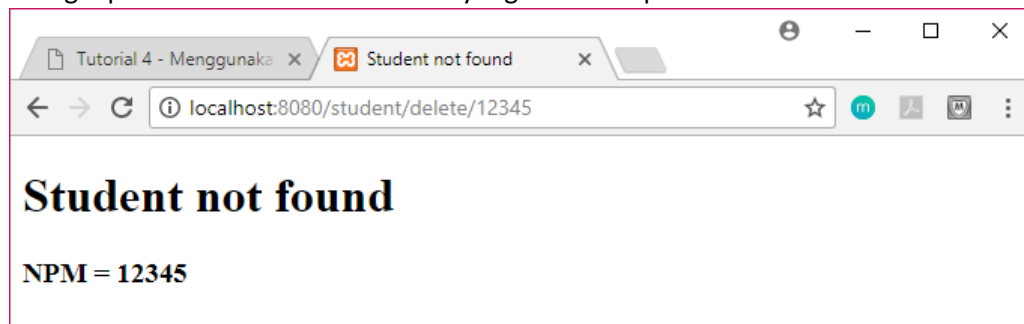
APAP B

Tutorial 4

- Menghapus *student* channel

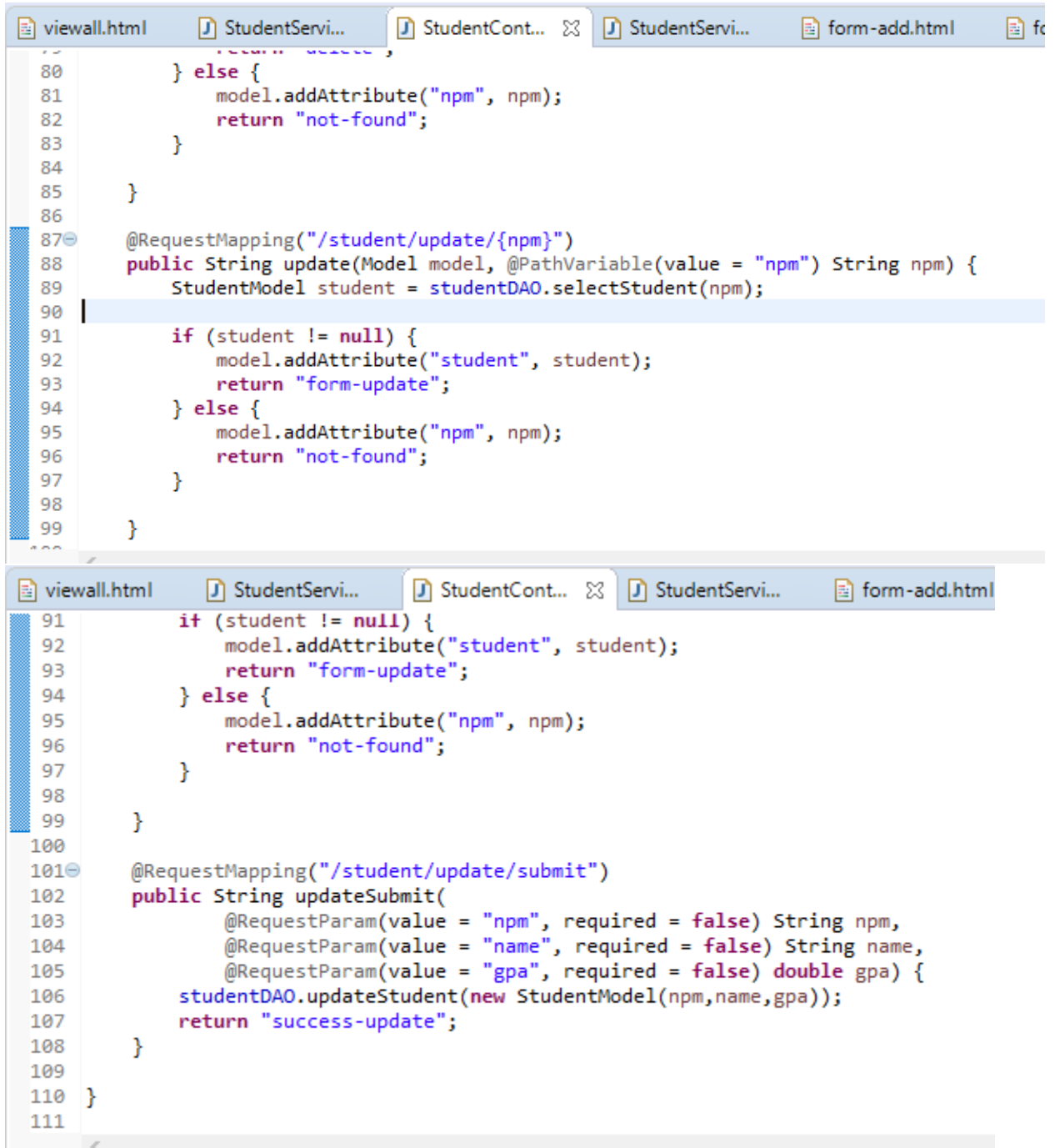


- Menghapus kembali *student* channel yang telah dihapus



Atikah Luthfiana  
1506689250  
APAP B  
Tutorial 4

## 2. Latihan menambahkan *update*



```
viewall.html StudentServi... StudentCont... StudentServi... form-add.html  
80 } else {  
81     model.addAttribute("npm", npm);  
82     return "not-found";  
83 }  
84  
85 }  
86  
87 @RequestMapping("/student/update/{npm}")  
88 public String update(Model model, @PathVariable(value = "npm") String npm) {  
89     StudentModel student = studentDAO.selectStudent(npm);  
90  
91     if (student != null) {  
92         model.addAttribute("student", student);  
93         return "form-update";  
94     } else {  
95         model.addAttribute("npm", npm);  
96         return "not-found";  
97     }  
98 }  
99 }  
100  
101 if (student != null) {  
102     model.addAttribute("student", student);  
103     return "form-update";  
104 } else {  
105     model.addAttribute("npm", npm);  
106     return "not-found";  
107 }  
108 }  
109  
110 @RequestMapping("/student/update/submit")  
111 public String updateSubmit(  
112     @RequestParam(value = "npm", required = false) String npm,  
113     @RequestParam(value = "name", required = false) String name,  
114     @RequestParam(value = "gpa", required = false) double gpa) {  
115     studentDAO.updateStudent(new StudentModel(npm, name, gpa));  
116     return "success-update";  
117 }  
118 }  
119 }  
120 }
```

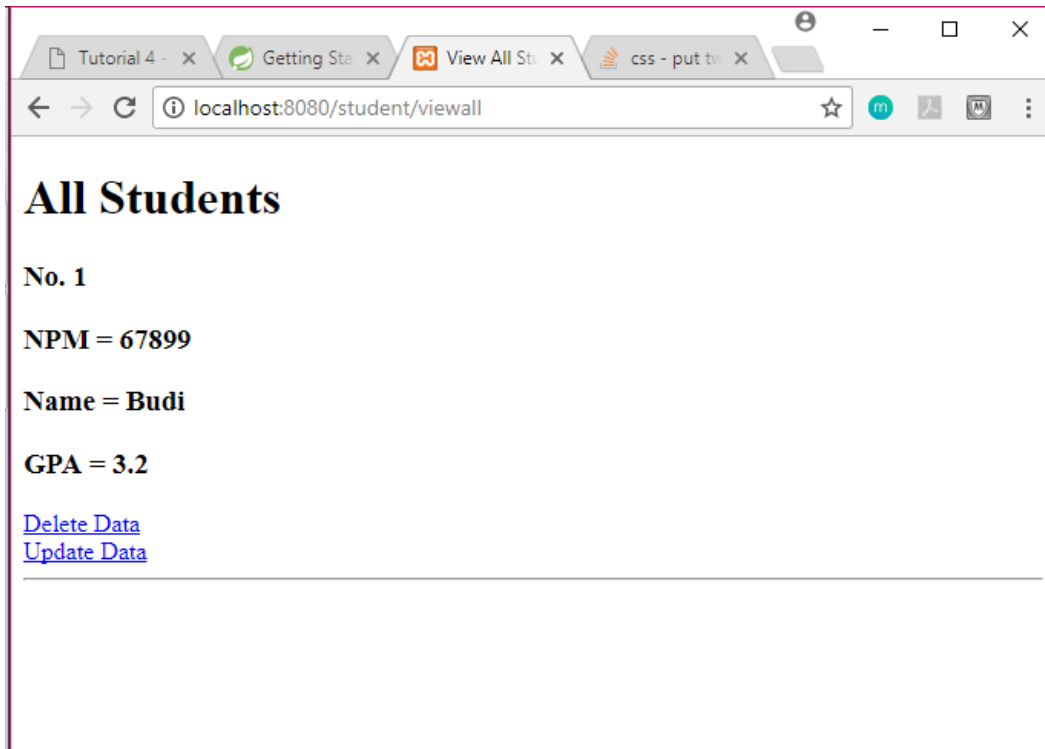
Atikah Luthfiana

1506689250

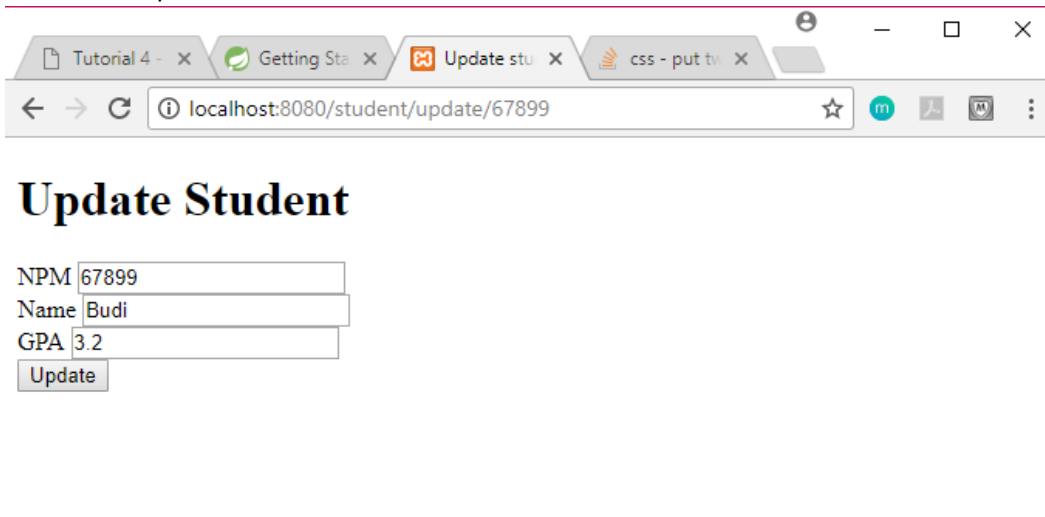
APAP B

Tutorial 4

- Melihat semua *student*



- Melakukan *update student*

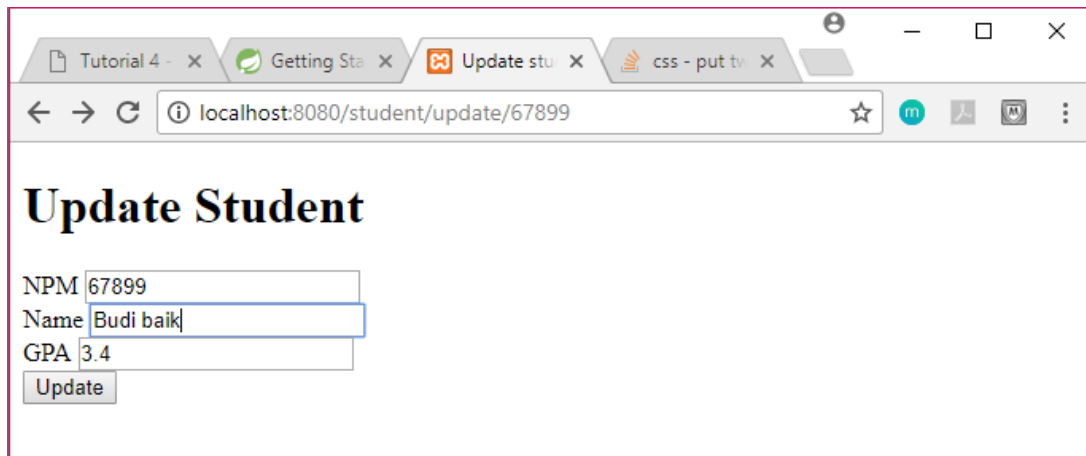


Atikah Luthfiana

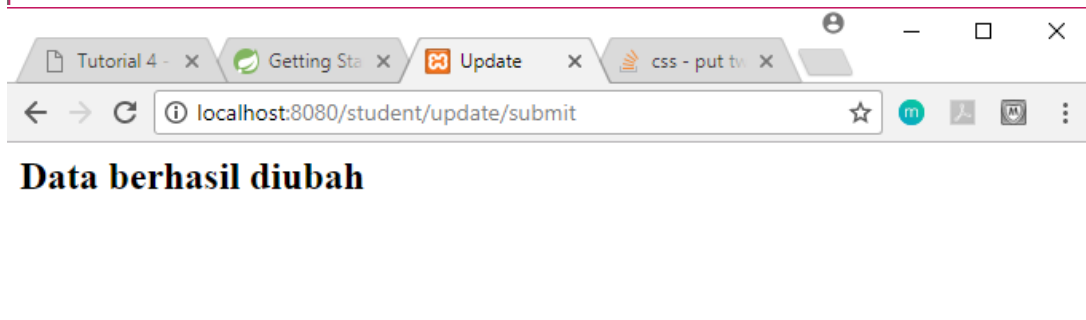
1506689250

APAP B

Tutorial 4

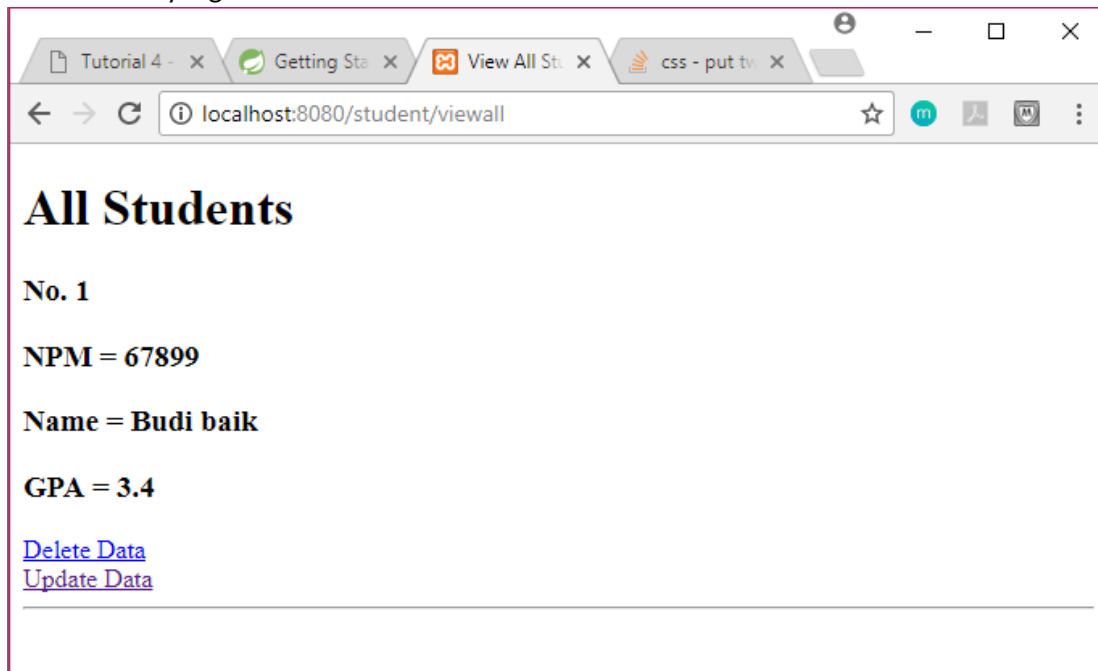


The screenshot shows a web browser window with the address bar displaying `localhost:8080/student/update/67899`. The page title is "Update Student". Below the title, there is a form with three input fields: "NPM" containing "67899", "Name" containing "Budi baik", and "GPA" containing "3.4". Below these fields is an "Update" button.



The screenshot shows a web browser window with the address bar displaying `localhost:8080/student/update/submit`. The page displays the message "Data berhasil diubah" (Data successfully changed).

- Melihat data yang telah diubah



The screenshot shows a web browser window with the address bar displaying `localhost:8080/student/viewall`. The page title is "All Students". Below the title, the following information is displayed:

- No. 1
- NPM = 67899
- Name = Budi baik
- GPA = 3.4

Below the information, there are two links: [Delete Data](#) and [Update Data](#).

Atikah Luthfiana

1506689250

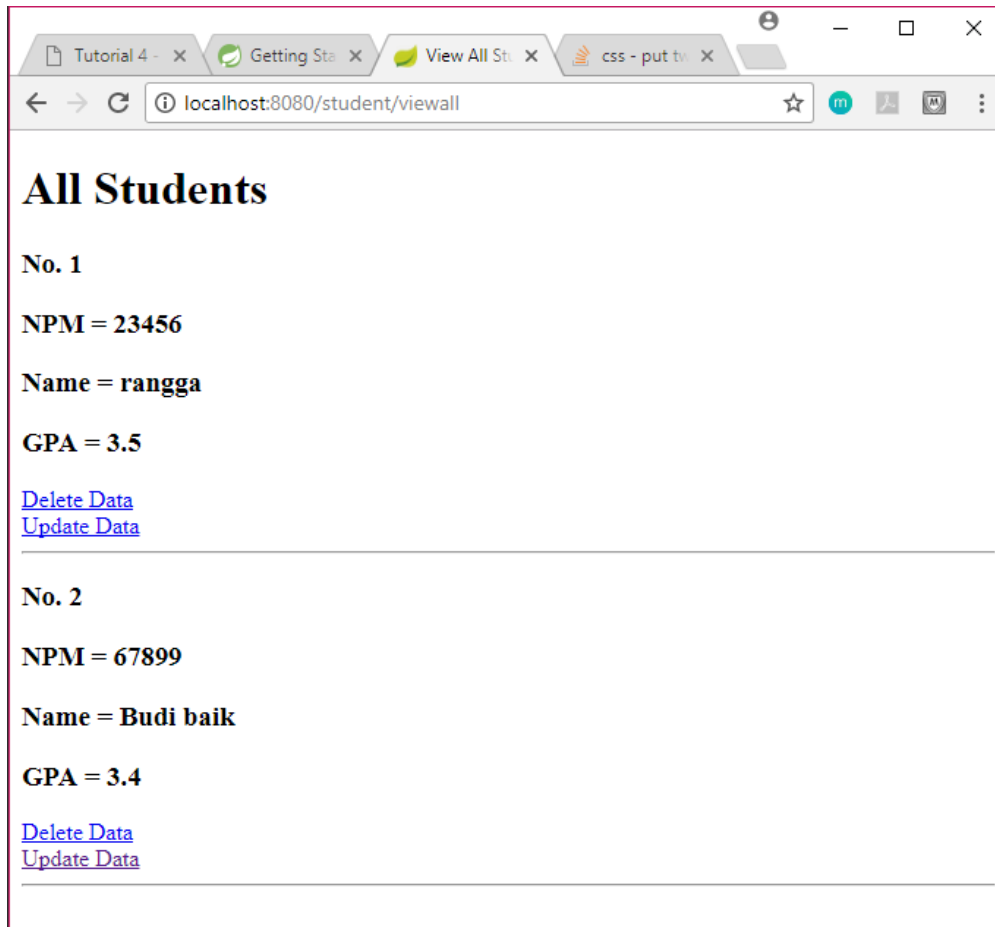
APAP B

Tutorial 4

3. Latihan menggunakan *object* sebagai *parameter*

```
viewall.html StudentServi... StudentCont... not-found.html form-add.html
89
90 @GetMapping("/student/update/{npm}")
91 public String update(Model model, @PathVariable(value = "npm") String npm) {
92     StudentModel student = studentDAO.selectStudent(npm);
93
94     if (student != null) {
95         model.addAttribute("student", student);
96         return "form-update";
97     } else {
98         model.addAttribute("npm", npm);
99         return "not-found";
100     }
101
102 }
103
104 @PostMapping("/student/update/submit")
105 public String updateSubmit(
106     @ModelAttribute StudentModel student) {
107     studentDAO.updateStudent(student);
108     return "success-update";
109 }
```

- Melihat daftar semua *student*



Atikah Luthfiana

1506689250

APAP B

Tutorial 4

- Mengubah data *student*

The screenshot shows a web browser window with the URL `localhost:8080/student/update/23456`. The page title is "Update Student". Below the title, there is a form with three input fields: "NPM" with the value "23456", "Name" with the value "rangga jahad", and "GPA" with the value "3.45". There is an "Update" button below the form. The browser's address bar shows the URL `localhost:8080/student/update/submit` after the form is submitted. The page title is "Data berhasil diubah".

- Melihat perubahan data yang telah diubah

The screenshot shows a web browser window with the URL `localhost:8080/student/viewall`. The page title is "All Students". Below the title, there is a table with two rows of student data. Each row has a "Delete Data" link and an "Update Data" link.

No.	NPM	Name	GPA	Delete Data	Update Data
No. 1	NPM = 23456	Name = rangga jahad	GPA = 3.45	<a href="#">Delete Data</a>	<a href="#">Update Data</a>
No. 2	NPM = 67899	Name = Budi baik	GPA = 3.4	<a href="#">Delete Data</a>	<a href="#">Update Data</a>



Atikah Luthfiana

1506689250

APAP B

Tutorial 4

## Lesson learned dari tutorial 4

Pada tutorial kali ini saya belajar mengenai *mapping parameter* yang kemudian digunakan untuk pengaksesan database menggunakan input parameter tersebut pada spring mvc. Selain itu kita juga dapat melakukan debugging pada program yang kita buat dengan bantuan anotasi Slf4j. dengan bantuan Lombok, kita tidak perlu mendeklarasikan setter getter dan juga constructor karena telah di generate secara otomatis.

### Penjelasan method

#### 1. Method delete

```
@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";
    }
}

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

Penjelasan :

*Method delete* menerima *parameter* berupa string npm. Kemudian *input* npm digunakan untuk mencari *student* yang ingin dihapus dengan bantuan *method selectStudent()*. Jika *student* yang dicari ditemukan, dilakukan penghapusan *student* dan ditampilkan pesan penghapusan berhasil dilakukan. Proses penghapusan *student* dilakukan dengan bantuan *studentMapper* yang melakukan pengaksesan ke database lalu menghapus data yang diinginkan sesuai npm yang *input*. Jika *student* tidak ditemukan, ditampilkan pesan bahwa *student* tidak ditemukan.

#### 2. Method update

```
@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";
    }
}
```

```
    }

    @PostMapping("/student/update/submit")
    public String updateSubmit(
        @RequestParam(value = "npm", required = true) String npm,
        @RequestParam(value = "name", required = true) String name,
        @RequestParam(value = "gpa", required = true) double gpa) {
        studentDAO.updateStudent(new StudentModel( npm, name, gpa));
        return "success-update";
    }

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

Penjelasan:

*Method update* menerima *parameter* berupa string npm. Kemudian *input* npm digunakan untuk mencari *student* yang ingin diubah datanya dengan bantuan *method selectStudent()*. Jika *student* yang dicari ditemukan, dilakukan *mapping* atribut *student* ke *form update*. *Form update* akan menerima current data student. Selanjutnya user dapat melakukan pengubahan data *student* pada *form update* lalu menekan tombol simpan. *Method updateSubmit* akan menerima *input parameter* berupa atribut-atribut yang dimasukkan oleh user melalui *form update*. Selanjutnya method ini kemudian melakukan pengaksesan database dengan bantuan *studentMapper* lalu dilakukan pengubahan data pada *database* sesuai *input* yang dimasukkan dan ditampilkan pesan pengubahan berhasil dilakukan. Jika *student* tidak ditemukan, ditampilkan pesan bahwa *student* tidak ditemukan.

### 3. Method update dengan *object* sebagai *parameter*

```
@GetMapping("/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";
    }
}

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

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

Atikah Luthfiana

1506689250

APAP B

Tutorial 4

Penjelasan:

Prosesnya sama seperti saat *update* sebelumnya. Bedanya, *form update* melakukan *post* ke *method updateSubmit* dalam bentuk *object student* secara langsung sehingga *parameter* pada *studentDAO.updateStudent(student)*; hanya cukup memasukkan *student* langsung tanpa harus memasukkan atribut *student* satu per satu.

#### Jawaban Pertanyaan

1. Pada beberapa kasus, validasi mungkin diperlukan untuk dapat menjaga keamanan dan/atau keutuhan data. Salah satu cara yang dapat dilakukan untuk melakukan validasi *backend* adalah dengan menggunakan bantuan *validation* yang telah disediakan Java. Kita dapat menggunakan *@* (semacam anotasi) untuk memberikan validasi tertentu yang dideklarasikan pada *object class* sebelum penulisan atribut. Misalnya menambahkan *@NotNull* sebelum penulisan atribut *npm* pada class *StudentModel*.

Sumber :

"Validating Form Input." Getting Started ·, [spring.io/guides/gs/validating-form-input/](https://spring.io/guides/gs/validating-form-input/). Accessed 30 Sept. 2017.

"Spring MVC Form Validation Example with Bean Validation API." CodeJava, [www.codejava.net/frameworks/spring/spring-mvc-form-validation-example-with-bean-validation-api](http://www.codejava.net/frameworks/spring/spring-mvc-form-validation-example-with-bean-validation-api). Accessed 30 Sept. 2017.

2. Form submit akan lebih aman ketika menggunakan method POST dibandingkan method GET. Ketika menggunakan method GET, hasil submission juga akan muncul pada URL. Sehingga untuk melakukan submit data, terutama data sensitive, akan lebih baik menggunakan method POST dibandingkan GET. Hal ini menyebabkan method POST lebih sering digunakan untuk form submit dibandingkan method GET. Jika dikirim menggunakan method yang berbeda, dapat digunakan anotasi yang sesuai dengan method yang diberikan/digunakan.

Sumber:

"PHP GET and POST Method – Tutorial." PHP GET and POST Method – Tutorial | FormGet, [www.formget.com/php-post-get/](http://www.formget.com/php-post-get/). Accessed 30 Sept. 2017.

3. Ya, sebuah method mungkin dapat menerima lebih dari satu jenis request method.