Ricca Fitriani
1506689616
APAP – B

**Tutorial 4**

**Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot**

## Write-up

Yang saya pelajari pada tutorial kali ini adalah menghubungkan *database* dengan *project spring boot* serta melakukan *debugging* dengan *logging*. Untuk melakukan operasi dengan data pada *database*, kita dapat menggunakan anotasi-anotasi *query* untuk mengelola *database* seperti SELECT, DELETE, UPDATE, INSERT pada kelas yang bertugas untuk melakukan pengelolaan data ke *database*. Kelas tersebut kemudian dapat dihidupkan dan dipanggil method-method yang diperlukan pada *service layer*, yang mana menghubungkan *controller* dan *database*. Sehingga, kita dapat melakukan pengelolaan data terlebih dahulu pada *controller* dan kemudian dapat disimpan ke *database*, atau kita dapat melakukan pengelolaan data yang sudah tersimpan di database untuk dilakukan operasi pada kelas *controller*.

Sedangkan untuk melakukan *debugging*, kita dapat menggunakan *logging* dengan library eksternal yaitu Lombok. Pada kelas yang ingin dilakukan *debugging*, kita dapat menggunakan anotasi @Slf4j pada bagian atas kelas, dan kemudian secara otomatis akan ada variable bernama log. Variable tersebut memiliki *method-method* untuk melakukan debugging seperti log.info, log.error, dan log.debug yang dapat ditulis pada *method-method* yang ingin dilakukan *debugging.*

## Latihan Menambahkan Delete

- Link delete data pada viewall.html

StudentModel... | InMemoryStu... | viewall.html | form-update.... | »21

```
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><br/>
15             <hr/>
16         </div>
17
18
19
20     </body>
21 </html>
```

Ricca Fitriani
1506689616
APAP – B

- Method deleteStudent pada StudentMapper

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

- Method deleteStudent pada StudentServiceDatabase

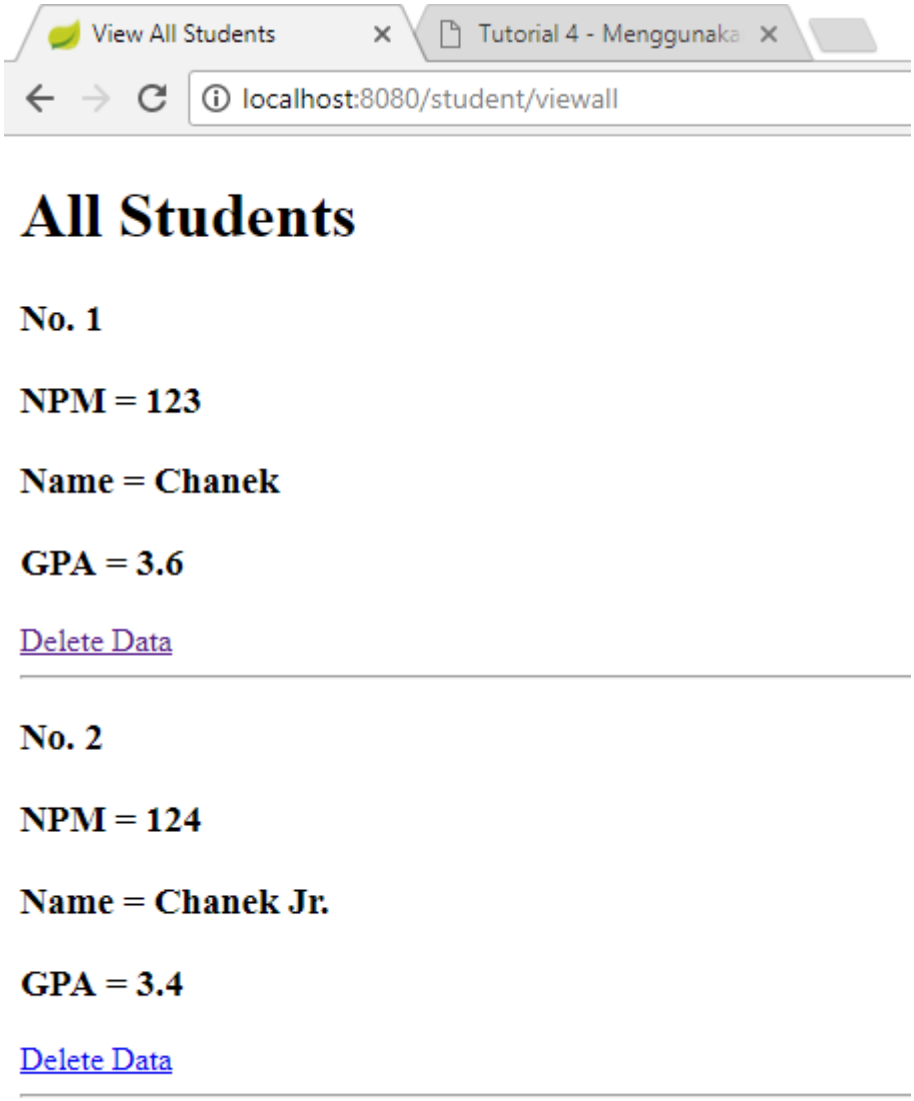
```
43
44    @Override
45    public void deleteStudent (String npm)
46    {
47        log.info ("student " + npm + " deleted");
48        studentMapper.deleteStudent (npm);
49    }
50
```

- Method delete pada StudentController

```
91
92    @RequestMapping("/student/delete/{npm}")
93    public String delete (Model model, @PathVariable(value = "npm") String npm)
94    {
95        StudentModel student = studentDAO.selectStudent (npm);
96
97        if (student != null) {
98            studentDAO.deleteStudent (npm);
99            return "delete";
100       } else {
101           model.addAttribute ("npm", npm);
102           return "not-found";
103       }
104   }
105
```

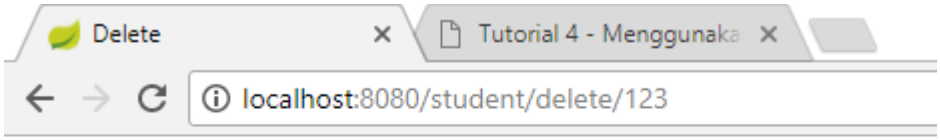
Ketika program di-*run*, karena *database* masih kosong, kita perlu memasukan data-data *student* untuk dicoba untuk dihapus nantinya terlebih dahulu. Data yang telah dimasukkan adalah sebagai berikut:

Ricca Fitriani
1506689616
APAP – B



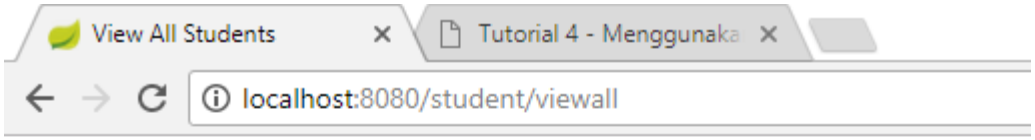

Kemudian, ketika *user* menekan tombol delete data pada npm 123 atau data nomor 1, maka sistem akan memanggil *method* dengan RequestMapping student/delete/npm pada *controller*. *Method* tersebut kemudian akan mengecek apakah data *student* yang ingin dihapus tersebut ada dengan memanggil *method* selectStudent dari kelas StudentServiceDatabase. Jika ada, maka sistem akan menghapus *student* tersebut dengan memanggil *method* deleteStudent pada kelas StudentServiceDatabase. Pada method deleteStudent yang ada di StudentServiceDatabase, akan dipanggil *method* deleteStudent yang ada pada kelas StudentMapper sehingga data pada *database* dapat terhapus. Pada method deleteStudent kelas StudentMapper terdapat *annotation* Delete yang dapat mengeksekusi *query* delete pada database. Kemudian, akan ditampikan halaman *template* delete yang memberikan informasi bahwa kelas berhasil dihapus sebagai berikut:

Ricca Fitriani
1506689616
APAP – B




**Data berhasil dihapus**

Tampilan viewall menjadi hanya satu *student* karena *student* dengan npm 123 sudah dihapus:




# All Students

**No. 1**

**NPM = 124**

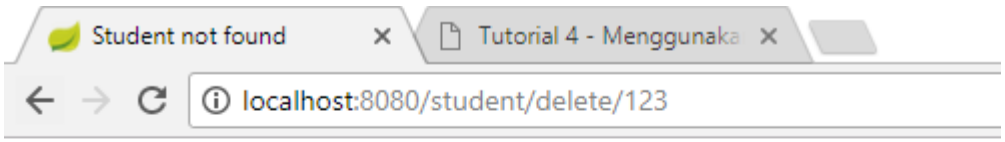
**Name = Chanek Jr.**

**GPA = 3.4**

Delete Data

---

Ketika *user* mencoba menghapus kembali data *student* melalui URL, maka akan ditampilkan *student not found* yang di-*handle* pada controller StudentController *method* delete yang menangani RequestMapping /student/delete/npm.




# Student not found

**NPM = 123**

Ricca Fitriani
1506689616
APAP – B

**Latihan Menambahkan Update**

- Method updateStudent pada kelas StudentMapper

```
28
29    @Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}")
30    void updateStudent(StudentModel student);
31 }
32
```

- Method updateStudent pada interface StudentService

```
19
20    void updateStudent(StudentModel student);
21 }
22
```

- Method updateStudent pada kelas StudentServiceDatabase

```
50
51    @Override
52    public void updateStudent (StudentModel student)
53    {
54        log.info ("student " + student.getNpm() + " update");
55        studentMapper.updateStudent(student);
56    }
57
```

- Link update data pada viewall.html

form-update.... | viewall.html | StudentCont... | success-upd... | »21

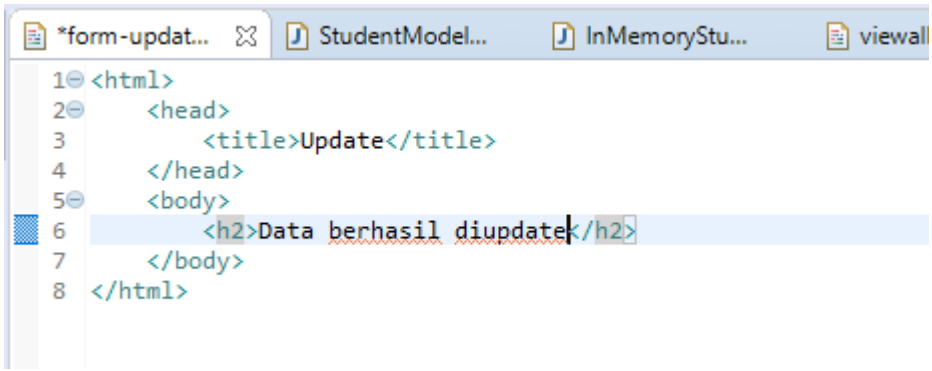
```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3      <head>
4          <title>View All Students</title>
5      </head>
6      <body>
7          <h1>All Students</h1>
8
9          <div th:each="student,iterationStatus: ${students}">
10             <h3 th:text="'No. ' + ${iterationStatus.count}">No. 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><br/>
15             <a th:href="'/student/update/' + ${student.npm}" > Update Data</a><br/>
16             <hr/>
17         </div>
18     </body>
19 </html>
20
```

Ricca Fitriani
1506689616
APAP – B

- Halaman template form-update.html

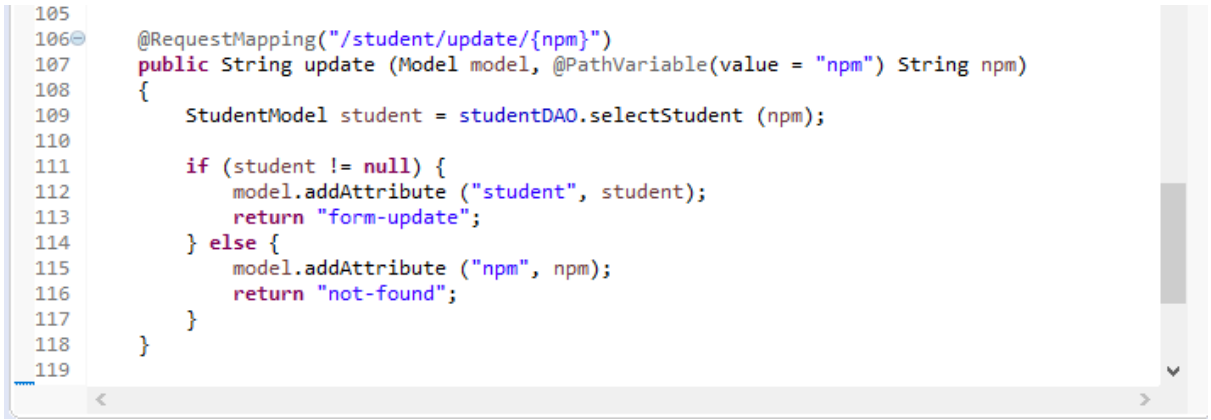
```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem Editor</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>
```

- Halaman template success-update.html



```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diupdate</h2>
    </body>
</html>
```

- Method update pada kelas StudentController



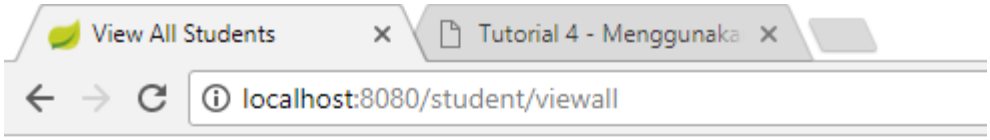
```
    @RequestMapping("/student/update/{npm}")
    public String update (Model model, @PathVariable(value = "npm") String npm)
    {
        StudentModel student = studentDAO.selectStudent (npm);

        if (student != null) {
            model.addAttribute ("student", student);
            return "form-update";
        } else {
            model.addAttribute ("npm", npm);
            return "not-found";
        }
    }
```

Ricca Fitriani
1506689616
APAP – B

- Method updateSubmit pada kelas StudentController

```
119
120    @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
121    public String updateSubmit (
122            @RequestParam(value = "npm", required = false) String npm,
123            @RequestParam(value = "name", required = false) String name,
124            @RequestParam(value = "gpa", required = false) double gpa)
125    {
126        StudentModel student = new StudentModel (npm, name, gpa);
127        studentDAO.updateStudent(student);
128
129        return "success-update";
130
131    }
```

Ketika program di-*run,* tampilkan *students* yang ada di *database* saat ini menggunakan *path variable* viewall pada url. Maka akan terlihat tampilan sebagai berikut:




# All Students

**No. 1**

**NPM = 124**

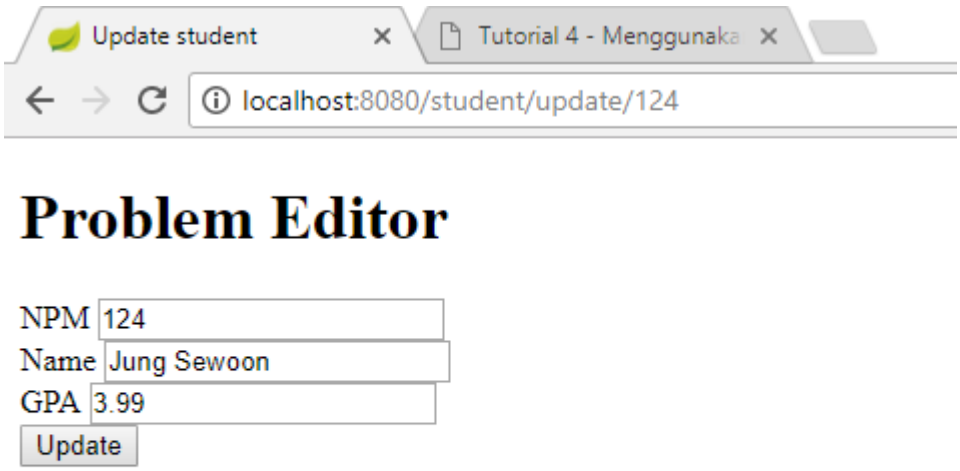
**Name = Chanek Jr.**

**GPA = 3.4**

Delete Data
Update Data

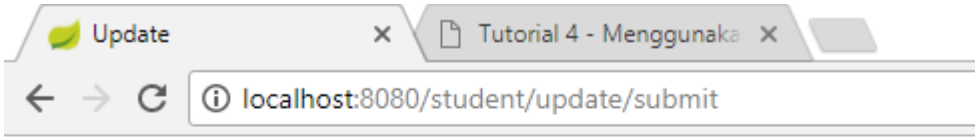
Ricca Fitriani
1506689616
APAP – B

Kemudian, ketika *user* menekan tombol update data pada npm 124, maka sistem akan memanggil *method* dengan RequestMapping student/update/npm pada *controller*. *Method* tersebut kemudian akan mengecek apakah data *student* yang ingin diupdate tersebut ada dengan memanggil *method* selectStudent dari kelas StudentServiceDatabase. Jika ada, maka sistem akan menampilkan form-update dan *passing object student* tersebut ke view form-update untuk ditampilkan data-data yang dimiliki *student* tersebut. Setelah form ditampilkan, maka kita dapat melakukan perubahan data sebagai berikut:



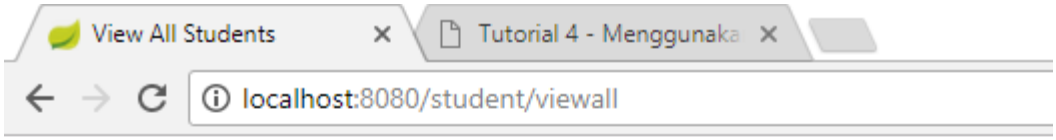

Namun, untuk data NPM tidak dapat dilakukan perubahan karena pada tag input npm terdapat atribut readonly true yang membuat kolom NPM hanya dapat dibaca atau tidak dapat diedit. Ketika *user* menekan tombol *update,* maka akan dipanggil *method* yang menangani RequestMapping /student/update/submit. Pada *method* tersebut, akan dipanggil *method* updateStudent pada kelas StudentServiceDatabase. Pada method updateStudent yang ada di StudentServiceDatabase, akan dipanggil *method* updateStudent yang ada pada kelas StudentMapper sehingga data pada *database* dapat terupdate dengan nilai yang baru yang dipassing melalui parameter pada setiap *method* yang berhubungan. Pada method updateStudent kelas StudentMapper terdapat *annotation* Update yang dapat mengeksekusi *query* update pada database. Kemudian, akan ditampikan halaman *template* success-update yang memberikan informasi bahwa kelas berhasil diupdate sebagai berikut:

Ricca Fitriani
1506689616
APAP – B




**Data berhasil diupdate**

Data pada *student* dengan npm 124 pun berhasil diupdate:




# All Students

**No. 1**

**NPM = 124**

**Name = Jung Sewoon**

**GPA = 3.99**

Delete Data
Update Data

Ricca Fitriani
1506689616
APAP – B

**Latihan Menggunakan Object Sebagai Parameter**

- View form-update yang sudah disesuaikan untuk passing *object student*.

form-update.... | StudentModel... | InMemoryStu... | viewall.html | StudentMapp... | StudentServi... | StudentCont... | success-upd...

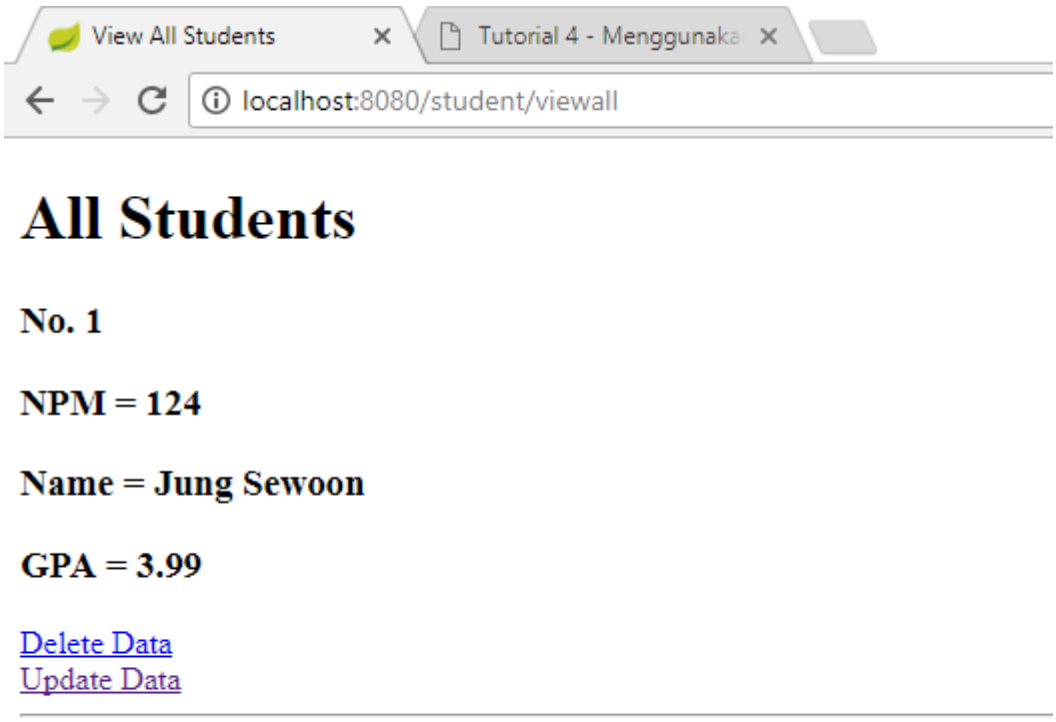
```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem 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>
```

- Method updateSubmit hanya menerima parameter *object student*

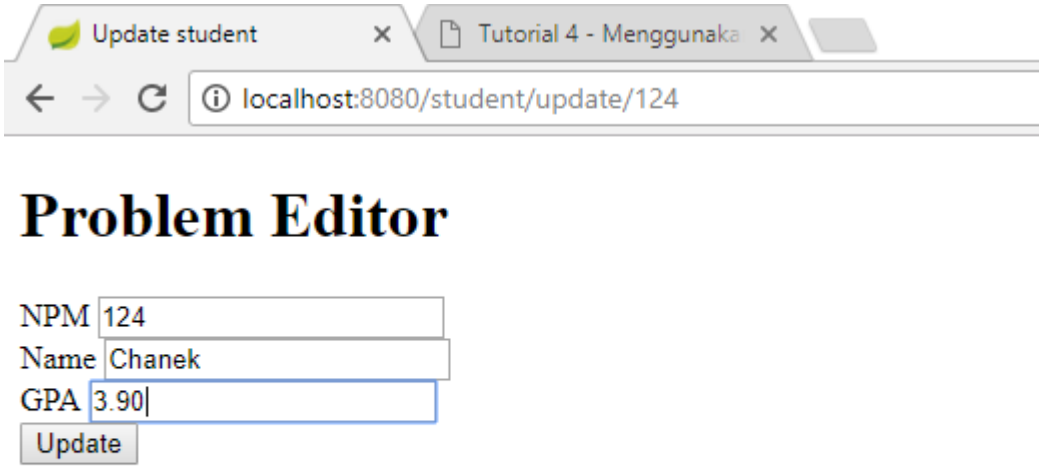
```
    @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
    public String updateSubmit (StudentModel student)
    {
        studentDAO.updateStudent(student);

        return "success-update";

    }

}
```

Ricca Fitriani
1506689616
APAP – B

Ketika program di-*run,* keadaan *database* adalah sebagai berikut:

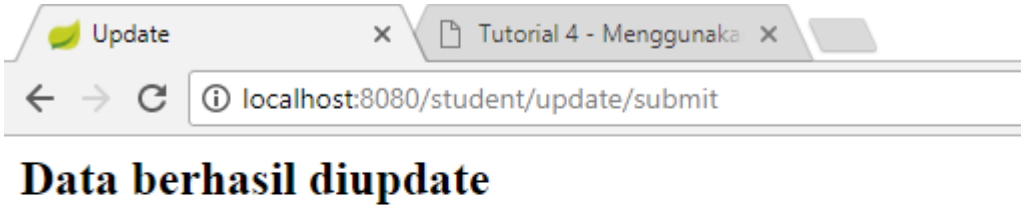



Kemudian, ketika *user* menekan tombol update data, maka *method* yang menangani RequestMapping /student/update/npm akan dipanggil. Sama seperti update pada sebelumnya, jika npm *student* yang ingin diupdate ditemukan, maka akan ditampilkan form-update sebagai berikut:

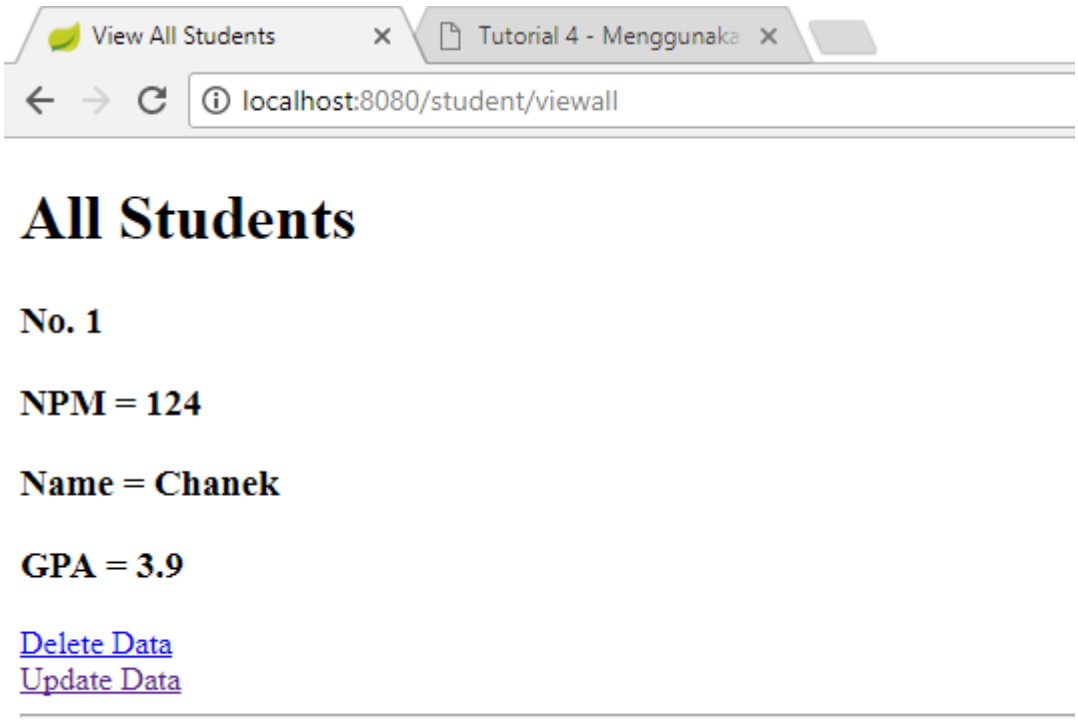


Ricca Fitriani
1506689616
APAP – B

Ketika data yang ingin diupdate sudah diganti, *user* akan menekan tombol update. Sama seperti proses update sebelumnya, *method* yang menangani RequestMapping /student/update/submit akan dipanggil. Hanya saja, saat ini *method* tersebut hanya menerima parameter *student* yang sudah dipassing dari form menggunakan *field student*. Kemudian proses berikutnya sama dengan proses update pada sebelumnya, *object student* tersebut akan diupdate dengan *method* updateStudent pada kelas StudentServiceDatabase dan StudentMapper, dan ditampilkan halaman *template* yang menginformasikan data berhasil diupdate sebagai berikut:




Data *student* tersebut pun berhasil diupdate:



Ricca Fitriani
1506689616
APAP – B

**Pertanyaan**

1. **Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan?**

   Validasi diperlukan karena jika menggunakan *object* sebagai parameter pada form POST, sistem tidak dapat otomatis meng-*handle* parameter yang required/optional sehingga jika terdapat parameter yang harus diisi seperti primary key pada database, tetap bisa tidak terisi dan membuat data menjadi tidak sesuai atau error.

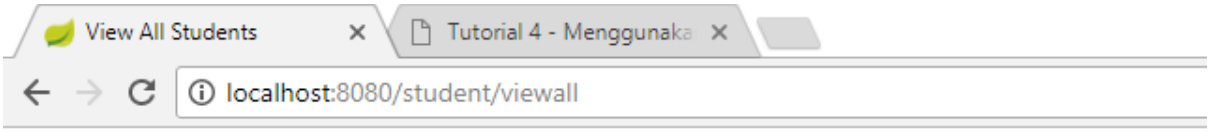
   Validasi optional atau required dapat dilakukan menggunakan objek yang menjadi parameter di kelas controller. Misalnya jika nama harus diisi, maka di controller dapat dibuat kondisi jika nama tidak diisi maka akan *redirect* ke halaman form kembali atau ke halaman error yang memberikan informasi bahwa nama harus diisi. Lebih lanjut dapat dijelaskan melalui screenshot berikut:

   Pada controller method updateSubmit dapat dibuat kondisi jika nama student kosong maka redirect ke halaman form.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (Model model, StudentModel student)
{
    if(student.getName().equals("")) {
        model.addAttribute ("student", student);
        return "form-update";
    } else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
```

   Ketika program di-*run*, data sebelumnya memiliki nama Chanek, kemudian diupdate tetapi tidak menyertakan nama, maka sistem akan *redirect* ke halaman form dan data pun tidak terupdate.

Ricca Fitriani
1506689616
APAP – B




# All Students

**No. 1**

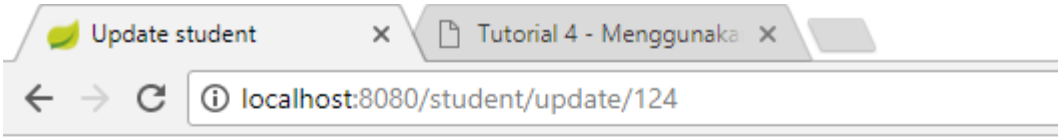
**NPM = 124**

**Name = Chanek**

**GPA = 3.9**

Delete Data
Update Data

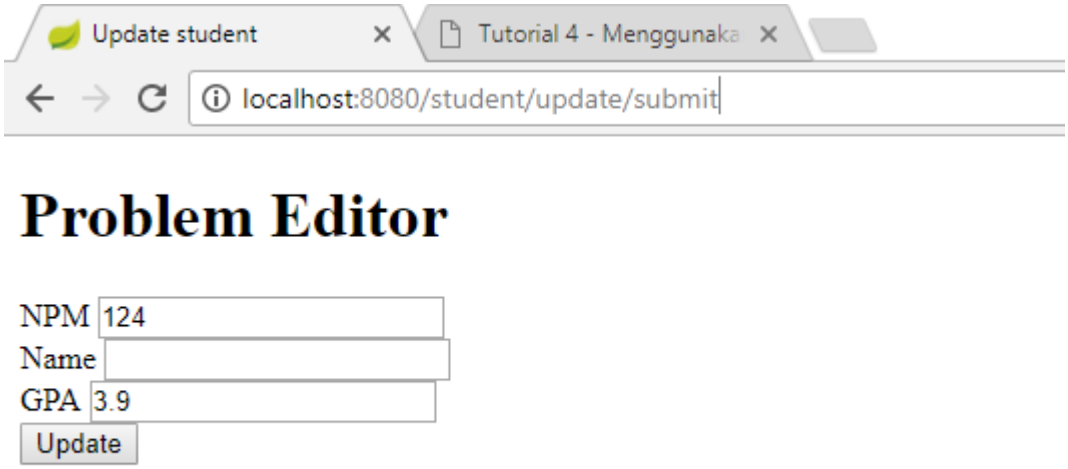
---




# Problem Editor

NPM 124
Name
GPA 3.9
Update

Ricca Fitriani
1506689616
APAP – B




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?**

   Form submit biasanya menggunakan POST method dibanding GET method karena pada GET method semua variable yang ada di form ditampilkan pada URL. Sehingga, tidak aman untuk digunakan karena data-data yang bersifat privat seperti password dapat terlihat pada URL.
   Pada header method controller juga terdapat perbedaan. Jika menggunakan GET method maka RequestMapping perlu ditambahkan "RequestMethod.GET" sedangkan jika menggunakan POST maka perlu ditambahkan "RequestMethod.POST"
   Reference: http://www.diffen.com/difference/GET-vs-POST-HTTP-Requests

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

   Mungkin. Karena dapat dituliskan pada RequestMapping method = { RequestMethod.GET, RequestMethod.POST }

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