

WRITE UP

Tutorial 4 - Menggunakan Database dan melakukan debugging pada project Springboot

Pada tutorial kali ini, saya belajar tentang cara menggunakan database secara nyata yang dijalankan diatas MySql dari aplikasi XAMPP saya ketika perkuliahan PPW (pada tutorial 3 kita hanya belajar dari inMemory saja). Perlu beberapa persiapan sebelum mengerjakan tutorial:

1. Download file Tutorial 4 (template) dari scele
2. Download project Lombok
3. java -jar Lombok.jar
4. Karena IDE saya tidak terdeteksi di layar aplikasi lombok (saya pakai mac), maka saya perlu specify keberadaan IDE saya
5. Buka eclipse, dan open project tutorial 4 dari scele
6. Build workspace

Setelah step-step tersebut dijalankan, barulah saya dapat mengerjakan tutorial.

Selain belajar tentang database, saya juga belajar cara membuat log untuk melakukan debugging menggunakan library Slf4j.

Latihan Menambahkan Delete

Menambah hyperlink untuk melakukan delete pada 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><br/>
15     </div>
16   </body>
17 </html>
```

Menambahkan method **deleteStudent** di **StudentMapper** beserta query Delete nya. Query delete dibuat dari table student dengan kondisi npm adalah npm yang diinput user.

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

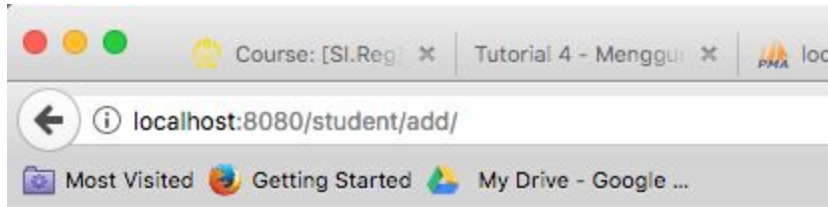
Menambah method **deleteStudent** pada **StudentServiceDatabase** serta menambahkan log juga didalamnya. Method ini hanya tinggal panggil method deleteStudent pada StudentMapper.

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

Melengkapi method **delete** pada **StudentController** beserta validasi bahwa student dengan npm tersebut ada. Method ini akan get student dulu dengan npm untuk memastikan bahwa student tersebut ada, jika terbukti ada, barulah dipanggil service (studentDAO) deleteStudent yang tadi sudah dibuat.

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

Lakukan run, dan buka /student/add untuk menambahkan data student baru

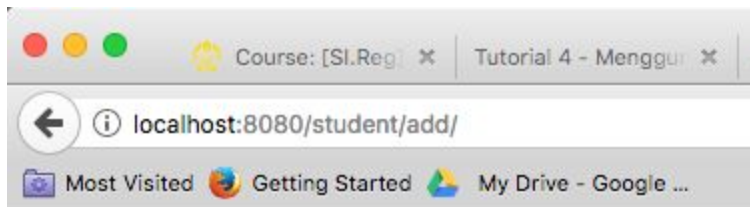


Problem Editor

NPM

Name

GPA



Problem Editor

NPM

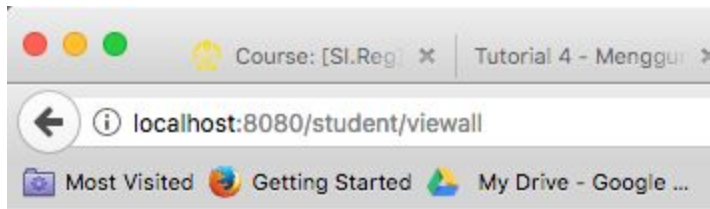
Name

GPA



Data berhasil ditambahkan

Sekarang studentnya sudah ter input, dapat dilihat di /student/viewall bahwa datanya sudah masuk.



All Students

No. 1

NPM = 12345678

Name = Martha Bucks

GPA = 3.65

[Delete Data](#)

No. 2

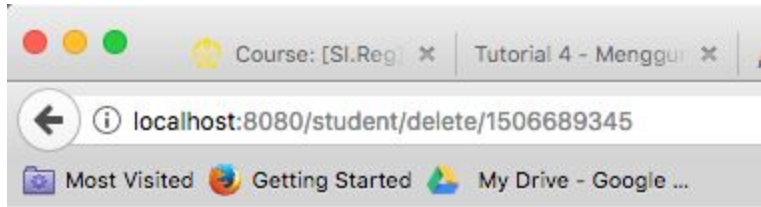
NPM = 1506689345

Name = Bella

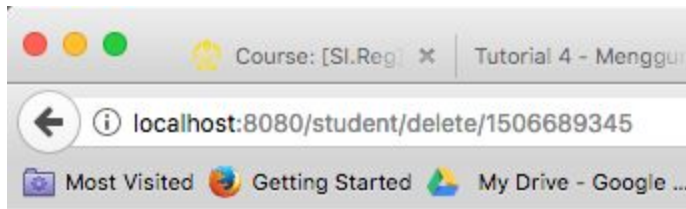
GPA = 4.0

[Delete Data](#)

Mencoba menghapus data dengan npm 1506689345 dengan klik delete pada data tersebut., data pun berhasil dihapus. Namun saat kita coba kembali hapus data tersebut (klik refresh), penghapusan gagal karena datanya sudah tidak ada (berhasil tampilkan error kalo datanya tidak ada).



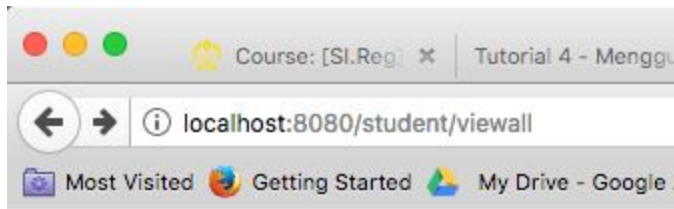
Data berhasil dihapus



Student not found

NPM = 1506689345

Dapat dilihat pada /student/viewall maupun pada phpmyadmin, data yang sebelumnya sudah terhapus.



All Students

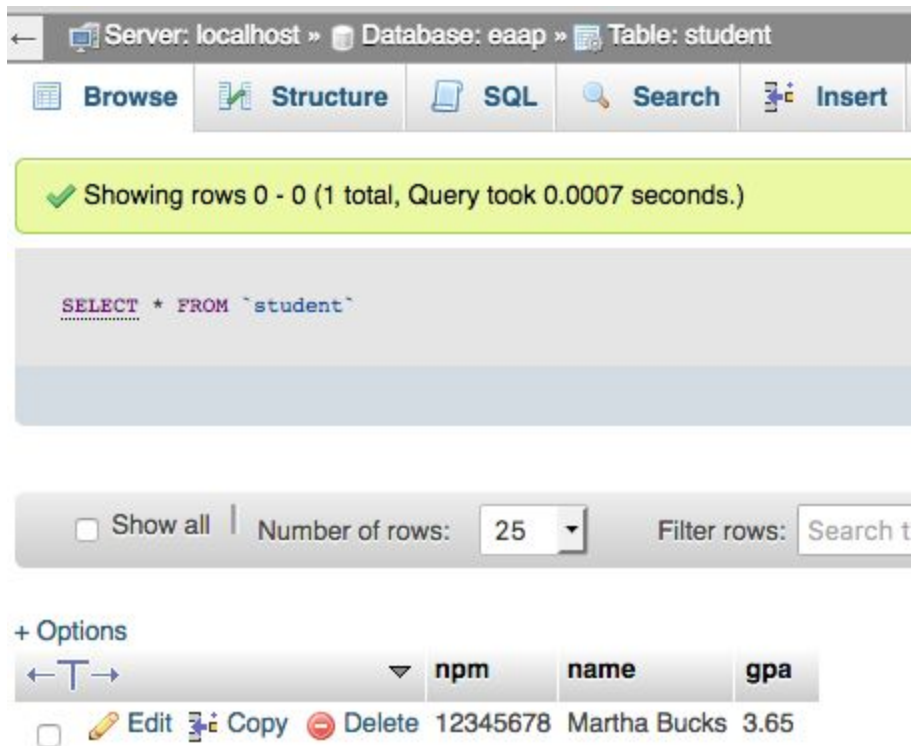
No. 1

NPM = 12345678

Name = Martha Bucks

GPA = 3.65

[Delete Data](#)



Seluruh proses yang kita lakukan tadi, tercatat di **log** sebagaimana kita tadi sudah berikan log di method2 nya.

```
2017-09-29 20:07:01.180 INFO 9007 --- [nio-8080-exec-4] c.e.service.StudentServiceDatabase : select student with npm 150
2017-09-29 20:07:01.182 INFO 9007 --- [nio-8080-exec-4] c.e.service.StudentServiceDatabase : student 1506689345 deleted
```

Latihan Menambahkan Update

Menambahkan hyperlink untuk melakukan update data student pada viewall.html

```
4 <a th:href="'/student/update/' + ${student.npm}" > Update Data</a><br/>
5 <a th:href="'/student/delete/' + ${student.npm}" > Delete Data</a></div>
```

Membuat method **updateStudent** pada **StudentMapper** beserta query updatenya. Parameter berupa StudentModel (mirip dengan addStudent). Query dibangun dari table stuent, dengan kondisi npm adalah npm yang diinput pengguna. Baru kita set npm, name, dan gpa dengan data yang diinput.

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

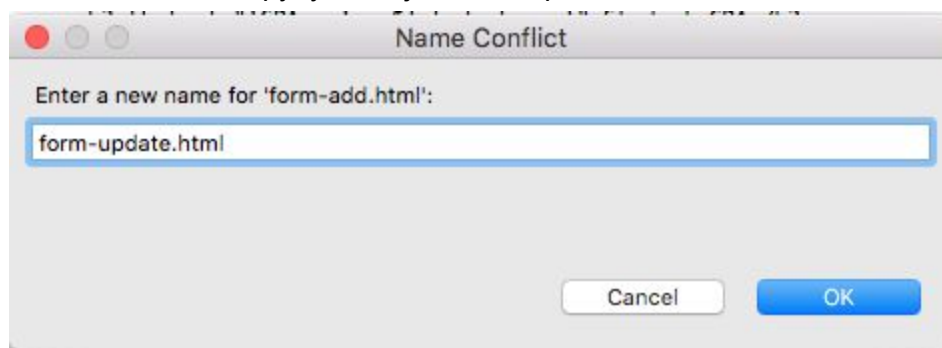
Menambahkan method **updateStudent** dengan parameter StudentModel di **StudentService** (hanya menambah method di interface)

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

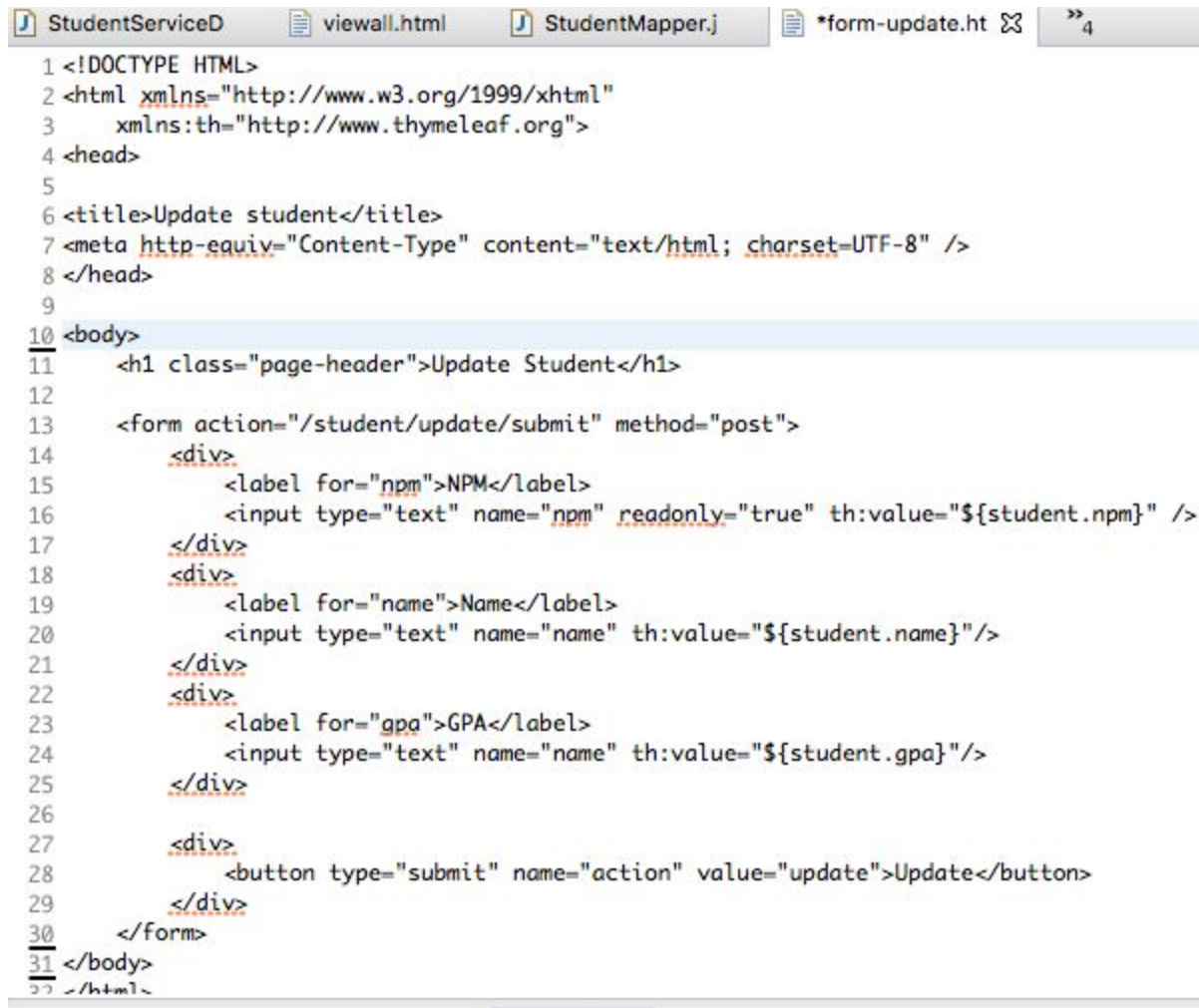
Implementasi method **updateStudent** di **StudentServiceDatabase** beserta log nya. Method ini hanya tinggal panggil studentMapper updateStudent.

```
42 @Override
43 public void updateStudent(StudentModel student) {
44     log.info("student updated with data to (" + student.getName() + ", " + student.getNpm() + "
45         + student.getGpa() + ")");
46     studentMapper.updateStudent(student);
47 }
```

Copy form-add.html dan copynya menjadi form-update.html.



Mengubah info-info yang diperlukan seperti, info-info yang diperlukan seperti title, page-header, tombol menjadi update dll. action form menjadi /student/update/submit. Ubah method menjadi post. Terakhir, menambahkan th:value untuk setiap input yang ada (sebagai nilai dari value dari data sebelum diupdate).

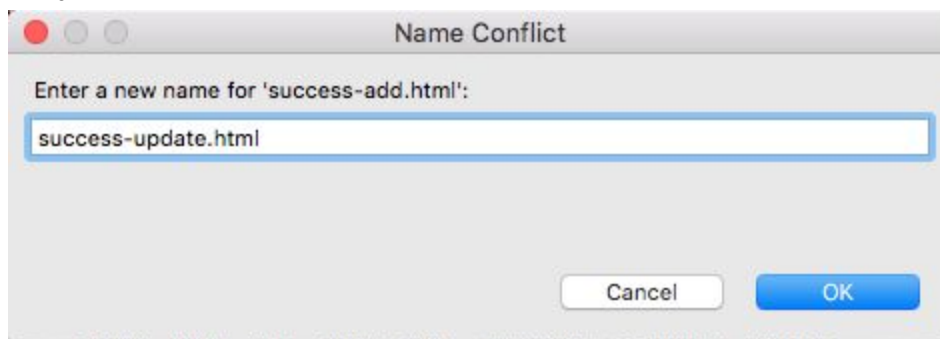


```

1 <!DOCTYPE HTML>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3   xmlns:th="http://www.thymeleaf.org">
4 <head>
5
6 <title>Update student</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8 </head>
9
10 <body>
11   <h1 class="page-header">Update Student</h1>
12
13   <form action="/student/update/submit" method="post">
14     <div>
15       <label for="npm">NPM</label>
16       <input type="text" name="npm" readonly="true" th:value="${student.npm}" />
17     </div>
18     <div>
19       <label for="name">Name</label>
20       <input type="text" name="name" th:value="${student.name}" />
21     </div>
22     <div>
23       <label for="gpa">GPA</label>
24       <input type="text" name="name" th:value="${student.gpa}" />
25     </div>
26
27     <div>
28       <button type="submit" name="action" value="update">Update</button>
29     </div>
30   </form>
31 </body>
32 </html>

```

Copy file success-add.html menjadi success-update.html serta mengubah informasi “add” related menjadi update.



Melakukan penambahan method **update** di **StudentController** beserta validasi jika npm tidak ditemukan. Sebelumnya ada validasi student ada, sama seperti saat ingin delete.

```
35 @RequestMapping("/student/update/{npm}")
36 public String update (
37     Model model,
38     @RequestParam(value = "npm", required = false) String npm)
39 {
40     // get student untuk validasi
41     StudentModel student = studentDAO.selectStudent (npm);
42     if (student == null) {
43         model.addAttribute ("npm", npm);
44         return "not-found";
45     } else {
46         model.addAttribute("student", student);
47         return "form-update";
48     }
49 }
```

Menambahkan method **updateSubmit** pada **StudentController** menggunakan post method beserta success-update. Tinggal panggil updateStudent dari service (studentDAO)

```
65 @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
66 public String updateSubmit (
67     @RequestParam(value = "npm", required = false) String npm,
68     @RequestParam(value = "name", required = false) String name,
69     @RequestParam(value = "gpa", required = false) double gpa)
70 {
71     StudentModel student = new StudentModel (npm, name, gpa);
72     studentDAO.updateStudent (student);
73
74     return "success-update";
75 }
```

Run program, dan beginilah hasilnya. (Merubah data Martha Bucks menjadi John Petrucci).

All Students

No. 1

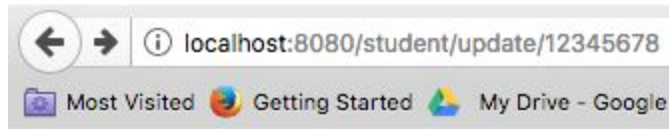
NPM = 12345678

Name = Martha Bucks

GPA = 3.65

[Update Data](#)

[Delete Data](#)

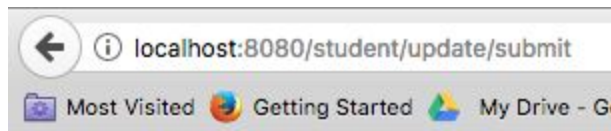


Update Student

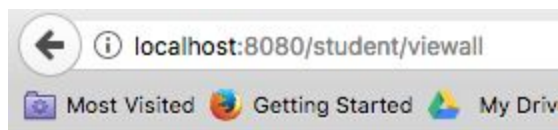
NPM

Name

GPA



Data berhasil di update



All Students

No. 1

NPM = 12345678

Name = John Petrucì

GPA = 3.88

[Update Data](#)

[Delete Data](#)

Ternyata, dari tadi ada log yang mencatat aktivitas kita, termasuk aktivitas melakukan update data martha bucks menjadi John petrucì.

```

c.e.service.StudentServiceDatabase : select student with npm 12345678
c.e.service.StudentServiceDatabase : select student with npm 12345678
c.e.service.StudentServiceDatabase : student updated with data to (John Petrucci, 12345678, 3.88)

2017-09-29 20:05:00.365 INFO 9007 --- [nio-8080-exec-7] c.e.service.StudentServiceDatabase : select student with npm 12345678
2017-09-29 20:05:03.483 INFO 9007 --- [nio-8080-exec-8] c.e.service.StudentServiceDatabase : select student with npm 12345678
2017-09-29 20:05:14.552 INFO 9007 --- [nio-8080-exec-9] c.e.service.StudentServiceDatabase : student updated with data to (John Petrucci,

```

Latihan Menggunakan Object Sebagai Parameter

Menambahkan `th:object="${student}"` pada tag di view dan Menambahkan `th:field="*{[nama_field]}"` pada setiap input

```

1 <!DOCTYPE HTML>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:th="http://www.thymeleaf.org">
4 <head>
5
6 <title>Update student</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8 </head>
9
10 <body>
11 <h1 class="page-header">Update Student</h1>
12
13 <form th:object="${student}" action="/student/update/submit" method="post">
14     <div>
15         <label for="npm">NPM</label>
16         <input th:field="*{npm}" type="text" name="npm" readonly="true" th:value="${student.npm}" />
17     </div>
18     <div>
19         <label for="name">Name</label>
20         <input th:field="*{name}" type="text" name="name" th:value="${student.name}" />
21     </div>
22     <div>
23         <label for="gpa">GPA</label>
24         <input th:field="*{gpa}" type="text" name="gpa" th:value="${student.gpa}" />
25     </div>
26
27     <div>
28         <button type="submit" name="action" value="update">Update</button>
29     </div>
30 </form>
31 </body>
32 </html>

```

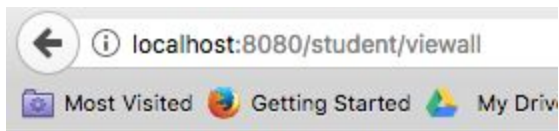
Ubah method `updateSubmit` pada `StudentController` yang hanya menerima parameter berupa `StudentModel` (menggunakan `@ModelAttribute` untuk dapat menerima object/class)

```

66 @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
67 public String updateSubmit (
68     @ModelAttribute StudentModel student)
69 {
70     studentDAO.updateStudent (student);
71
72     return "success-update";
73 }
--

```

Test aplikasi



All Students

No. 1

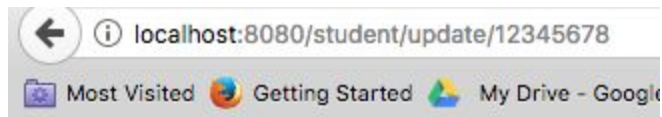
NPM = 12345678

Name = John Petrucci

GPA = 3.88

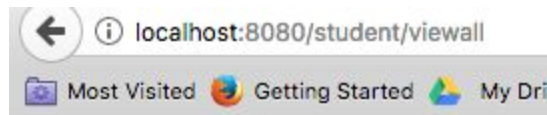
[Update Data](#)

[Delete Data](#)



Update Student

| | |
|---------------------------------------|--|
| NPM | <input type="text" value="12345678"/> |
| Name | <input type="text" value="Martha Tilaar"/> |
| GPA | <input type="text" value="4.00"/> |
| <input type="button" value="Update"/> | |



All Students

No. 1

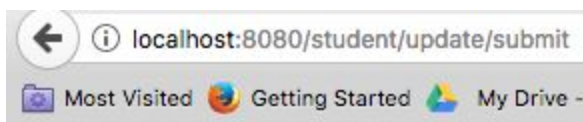
NPM = 12345678

Name = Martha Tilaar

GPA = 4.0

[Update Data](#)

[Delete Data](#)



Data berhasil di update

Hasil Log nya seperti berikut

```
2017-09-29 20:21:07.074 INFO 10576 --- [nio-8080-exec-6] c.e.service.StudentServiceDatabase : select student with npm 12345678
2017-09-29 20:21:31.892 INFO 10576 --- [nio-8080-exec-7] c.e.service.StudentServiceDatabase : student updated with data to (Martha Tilaar
2017-09-29 20:21:44.410 INFO 10576 --- [nio-8080-exec-8] c.e.service.StudentServiceDatabase : select all students

select student with npm 12345678
student updated with data to (Martha Tilaar, 12345678, 4.0)
select all students
```

Dengan demikian latihan berhasil diselesaikan.

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

Misalkan kita punya suatu kasus, dimana isian nama tidak boleh kosong (mandatory), dan isian gpa adalah boleh kosong (optional). Kita bisa lakukan validasi di backend mengenai ini

- Pertama, saat controller yang menerima POST request dari form yang disubmit, kita akan dapatkan object/class nya.
- Kita berikan pengecekan, apakah nama null. Jika iya, maka langsung tampilkan pesan error untuk pengguna
- Kita berikan pengecekan, apakah nama kosong (isEmpty), jika iya, tampilkan pesan error juga
- Jika lolos validasi tersebut, maka lanjutkan proses di method tersebut (bisa dengan validasi field lainnya, atau jalankan proses yang ingin dijalankan).

Contoh dibawah adalah kode untuk updateSubmit dengan @ModelAttribute dan validasi (nama dan npm mandatory) di backend. Kode tidak di submit di scele, ini hanya contoh untuk jawaban pertanyaan.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
    @ModelAttribute StudentModel student)
{
    if (student.getNpm() == null || student.getNpm().isEmpty()) {
        // return template error disini
    }
    if (student.getName() == null || student.getName().isEmpty()) {
        // return template error disini
    }

    studentDAO.updateStudent (student);

    return "success-update";
}
```

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?

Penggunaan method POST tidak lah lebih secure dibandingkan penggunaan method GET [1]. Walaupun method POST tidak mengekspose informasi dalam url, namun saat informasi dikirimkan melalui jaringan, seluruh informasi tetap terekspose. Meski begitu, umumnya, penggunaan method GET adalah dengan tujuan untuk membuat informasi terekspose jelas di url (untuk tujuan bookmarking, copy paste url, share url, seach engine optimization, dll).

Umumnya, saat kita akan melakukan submit form, pengguna tidak memerlukan url yang bisa di bookmark/share ketika melakukan submit, maka dari itu penggunaan method POST lebih baik. Selain itu, menurut saya pribadi, ketika data disubmit menggunakan method POST, user akan lebih merasa aman pula (itu yang saya rasakan sebagai user). Mengekspose informasi yang terlalu banyak (misal formulir pendaftaran) melalui method GET juga akan membuat URL menjadi berantakan dan optimisasi search engine menjadi jelek pula.

Iya, sepertinya butuh penanganan berbeda di controller jika form disubmit menggunakan request method yang berbeda. Hal ini terjadi di tutorial kali ini dimana kita perlu specify suatu method di controller menerima request method apa.

[1] <https://stackoverflow.com/questions/198462/is-either-get-or-post-more-secure-than-the-other>

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

Mungkin saja, terdapat cara tersendiri di Spring untuk melakukan hal tersebut. Salah satunya seperti ini:

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

Lesson Learned

Pada tutorial kali ini, saya belajar tentang tata cara penggunaan library pihak ketiga untuk implementasi spring menggunakan database yang sesungguhnya. Selain itu, saya juga jadi belajar tentang penggunaan log untuk melakukan debugging, cara yang lebih proper dari pada menggunakan System.out.println(). Dengan begini, bertambah satu lagi ilmu saya dalam membangun sistem menggunakan java dengan framework spring.