Mayanggita Rachmaryza Indra
150672974

Tutorial 4 ADPAP

Pertanyaan:

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.
   Validasi pada form POST bisa dilakukan menggunakan @PostMapping pada Controller, dimana validasi tetap dilakukan, namun tidak langsung pada form-nya.

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?
   Menurut saya, form submit biasanya menggunakan POST method dibandingkan dengan GET method. Hal ini dilakukan untuk melindungi data-data yang ada.

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

6.3. *@RequestMapping* – a fallback for all requests

To implement a simple fallback for all requests using a particular HTTP method – for example, for a GET:

```
@RequestMapping(value = "*", method = RequestMethod.GET)
@ResponseBody
public String getFallback() {
    return "Fallback for GET Requests";
}
```

Or even for all requests:

```
@RequestMapping(
  value = "*",
  method = { RequestMethod.GET, RequestMethod.POST ... })
@ResponseBody
public String allFallback() {
    return "Fallback for All Requests";
}
```

Sumber: http://www.baeldung.com/spring-requestmapping

Mayanggita Rachmaryza Indra
150672974

Latihan Menambahkan Delete

1. Menambahkan 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><br/>
            <hr/>
        </div>
    </body>
```

2. Menambhakan method deleteStudent di class StudentMapper

```
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. Melengkapi method deleteStudent di class StudentServiceDatabase

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

Mayanggita Rachmaryza Indra
150672974

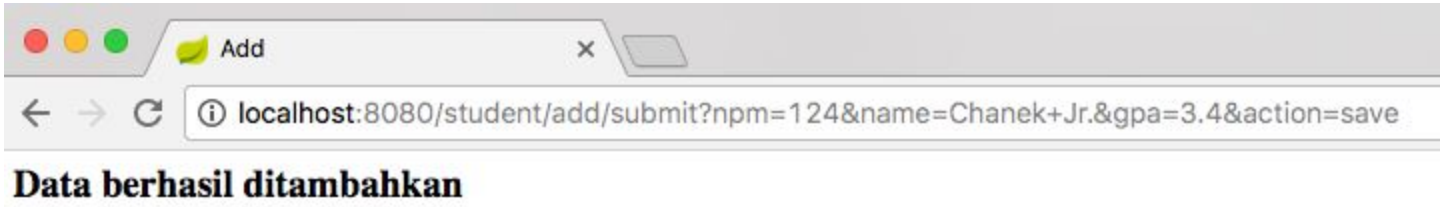
4. Melengkapi method delete di class StudentController

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

    if(student == null) {
        model.addAttribute("npm", npm);
        return "not-found";
    }else {
        studentDAO.deleteStudent (npm);
        return "delete";
    }
}
```

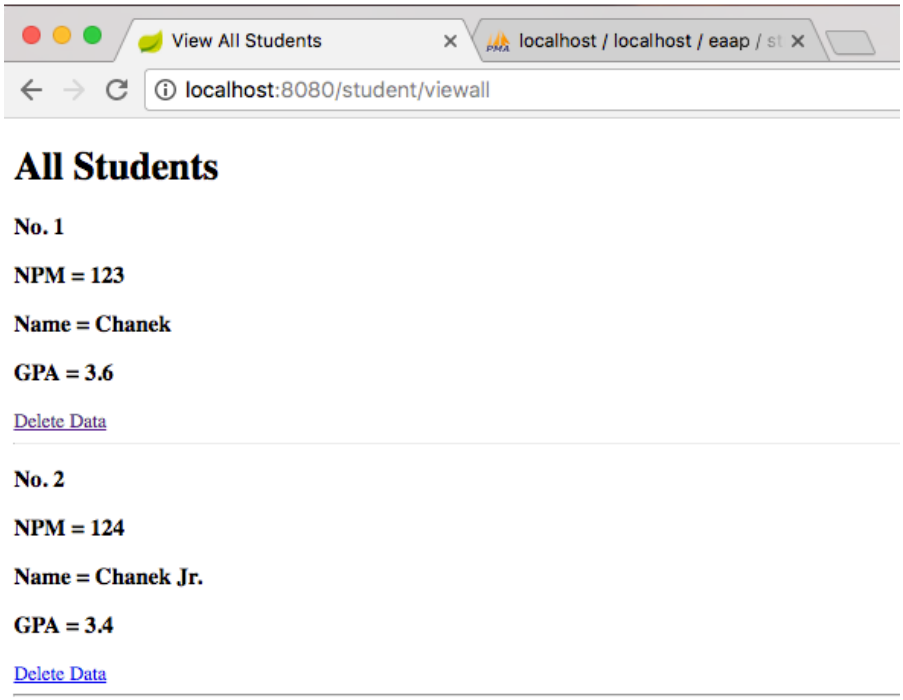
Penjelasan:
Method ini memiliki parameter npm sebagai @pathVariable, kemudian akan menjalankan method selectStudent() dengan parameter npm tersebut. Apabila object studentDAO nya null, maka akan menampilkan “not-found”, namun jika bukan null, maka akan menjalankan method deleteStudent() dengan parameter npm yang tadi.

Hasil:
1. Tambah 2 student terlebih dahulu

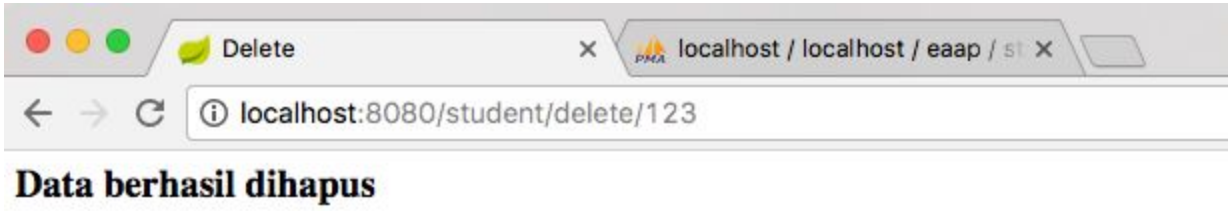


Mayanggita Rachmaryza Indra
150672974

2. Melihat semua student yang ada

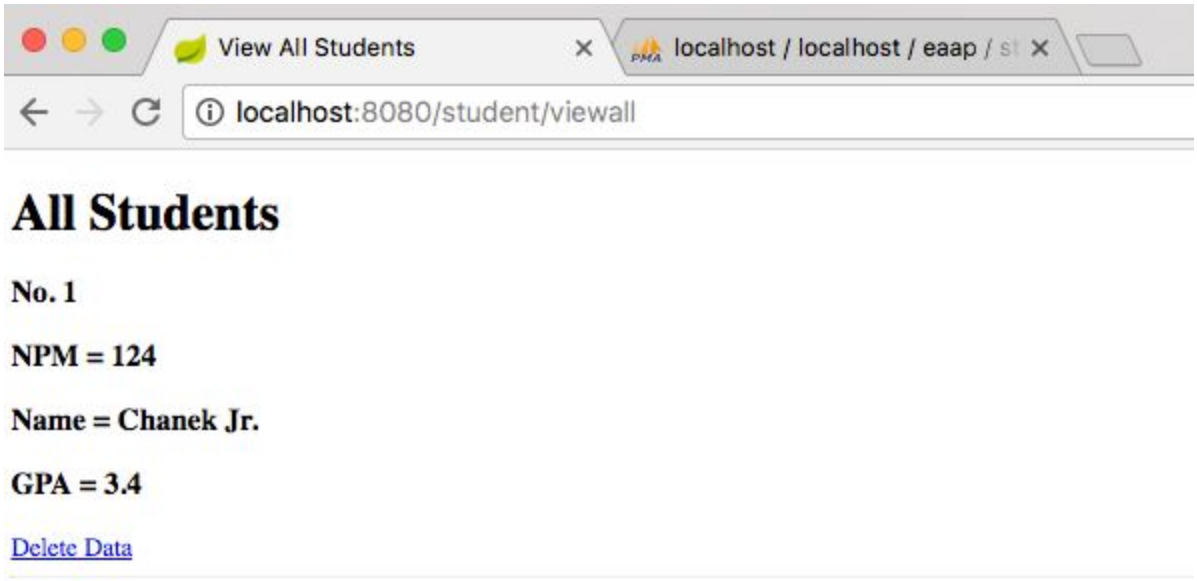


3. Menghapus student dengan npm=123

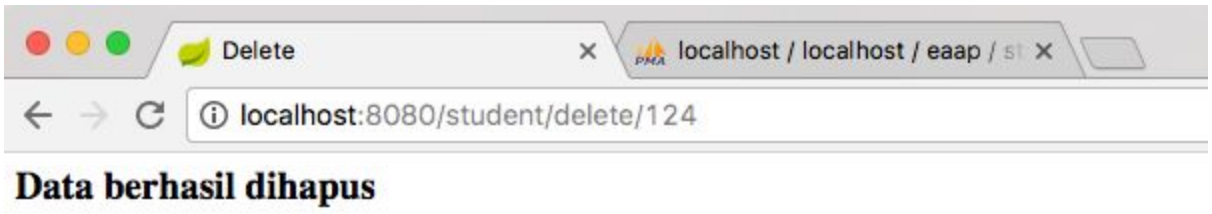


4. Dapat terlihat bahwa hanya terdapat 1 sisa student, yaitu student dengan npm = 124

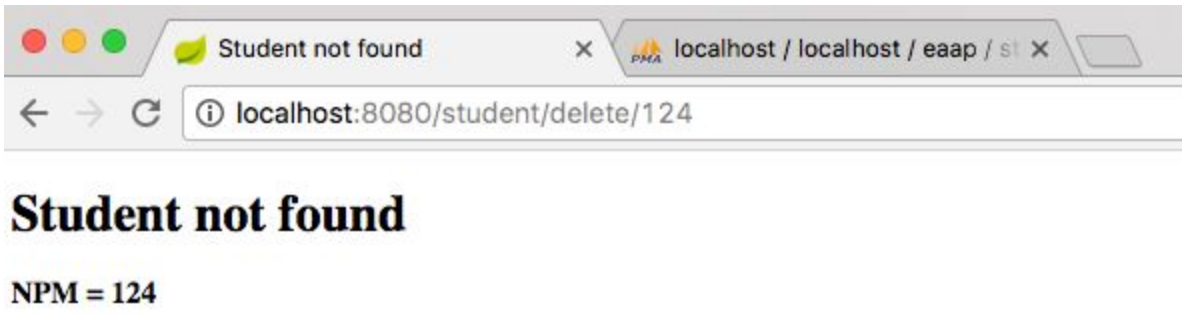
Mayanggita Rachmaryza Indra
150672974



5. Menghapus student dengan npm=124



6. Karena sudah terhapus, maka saat kita mencoba menghapus student dengan npm = 124, muncul seperti ini (Student not found)



Mayanggita Rachmaryza Indra
150672974

Latihan menambahkan Update

1. Menambahkan method updateStudent pada interface StudentService

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

2. Mengimplementasi method updateStudent pada class StudentServiceDatabase

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

3. Menambahkan link Update Data pada viewall.html

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

4. Membuat form-update.html

5. Membuat succes-update.html

```
<html>
    <head>
        <title>Update</title>
    </head>

    <body>
        <h2>Data berhasil diperbaharui</h2>
    </body>

</html>
```

Mayanggita Rachmaryza Indra
150672974

6. Menambahkan 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 ("npm", npm);
        return "not-found";
    } else {
        model.addAttribute("student", student);
        return "form-update";
    }
}
```

Penjelasan:
Method ini memiliki parameter npm sebagai @pathVariable, kemudian akan menjalankan method selectStudent() dengan parameter npm tersebut. Apabila object studentDAO nya null, maka akan menampilkan “not-found”, namun jika bukan null, maka akan menambahkan object student sebagai atribut pada model, kemudian memanggil “form-update”.

Mayanggita Rachmaryza Indra
150672974

Latihan menggunakan Object Sebagai Parameter

1. Menambahkan th:object=”${student}” pada tag <form> di view

```
<form th:object="${student}" action="/student/update/submit" method="post">
```

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

```
<div>
    <label for="npm">NPM</label> <input th:field="*{npm}" type="text" name="npm" readonly="true" th:value="${student.npm}
</div>
<div>
    <label for="name">Name</label> <input th:field="*{name}" type="text" name="name" th:value="${student.name}" />
</div>
<div>
    <label for="gpa">GPA</label> <input th:field="*{gpa}" type="text" name="gpa" th:value="${student.gpa}"/>
</div>
<div>
    <button type="submit" name="action" value="save">Update</button>
</div>
```

3. Ubah method updateSubmit pada studentController yang hanya menerima parameter berupa StudentModel

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

Penjelasan:
Method ini memiliki parameter object student dari class StudentModel, kemudian akan menjalankan method selectStudent() pada studentDAO dengan parameter tersebut . Lalu, akan menampilkan “success-update”.

Lesson learned:
Pada tutorial 4 ini, saya mempelajari tentang bagaimana Spring Boot terhubung dengan database (mysql) dan memahami cara kerja form pada Spring Boot (method GET mapun POST). Selain itu, saya mempelajari penggunaan Object sebagai parameter yang dapat digunakan untuk men-submit form tanpa harus menulis atributnya di form satu-persatu.