WRITE-UP

Pada tutorial keempat kali ini, saya mempelajari hal baru terkait menghubungkan database (phpMyAdmin) dengan project pada Project Spring Boot dan cara melakukan Debugging pada Project tersebut. Pada awal tutorial saya diajarkan untuk memasukan (*import*) file ke dalam Spring boot dan membuat serta mengubungkan suatu database ke dalam project spring boot. Secara garis besar tutorial kali ini merupakan implementasi lanjutan dari tutorial ketiga kemarin. Seperti melakukan perubahan terhadap method *delete* yang sebelumnya menggunakan array list menjadi database. Serta membuat method baru untuk mengupdate data nama dan gpa mahasiswa.

Selain itu saya mempelajari hal baru terkait penggunaan Object Sebagai Parameter untuk menggantikan Request Parameter. Hal tersebut bertujuan untuk mengurangi penggunaan parameter dari suatu method. Jika suatu form memliki banyak field maka tentu akan mempersulit kita untuk membuat parameter dan tidak terlihat rapih pada kodingan kita. Selain itu, saya juga mengingat kembali penggunaan method GET dan POST yang telah diajarkan pada mata kuliah PPW.

Penjelasan method Delete pada Latihan

Method delete pada tutorial kali ini merupakan modifikasi dari method delete pada tutorial sebelumnya. Perbedannya adalah penggunaan database dari phpMyadmin. Pertama, kita harus menambahkan link delete Data Pada “viewall.html”. Seperti gambar dibawah ini:

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

Kemudian, Untuk membuat method delete, kita harus membuat syntax sql untuk delete dan method deleteStudent pada interface studentMapper.java . Seperti gambar dibawah ini:

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

Setelah itu, kita membuat method deleteStudent yang ada di class StudentServiceDatabase, berisikan log info dan pemanggilan method deleteStudent yang berada di interface Student Mapper. Seperti gambar dibawah ini:

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

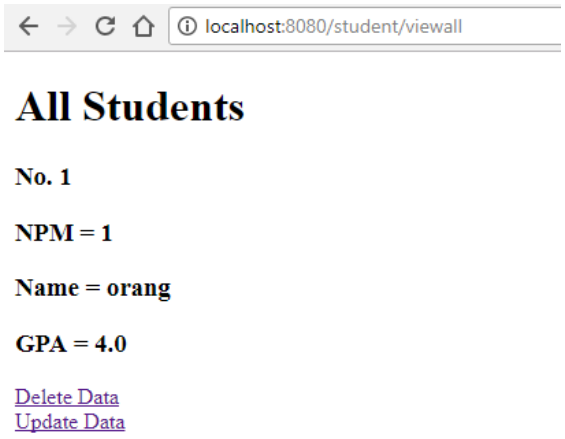
Kemudian, kita mengimplementasikan method delete pada class StudentController.

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

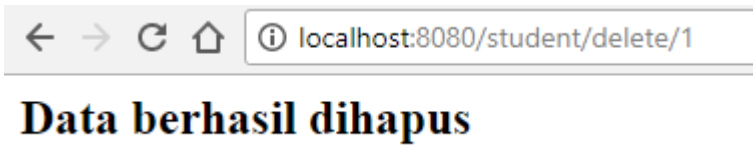
Method tersebut menerima request mapping dari http://localhost:8080/student/delete/{npm}. Method tersebut akan membuat objek student yang memiliki npm sama dengan input npm dari request mapping. Ketika objek student tersebut ada, maka akan dipanggil method deleteStudent dan menampilkan halaman "delete.html". Jika tidak ada, maka akan menampilkan halaman "not-found.html".

Tampilan di Localhost :

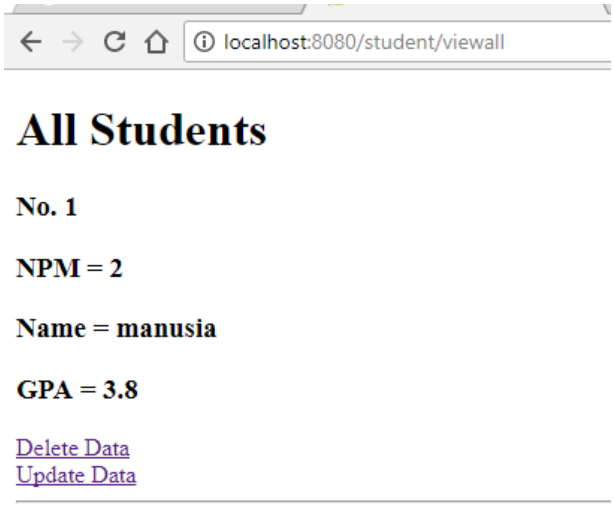
Tampilan pada viewall.



Setelah klik tulisan "Delete data".



Di cek melalui viewall, data mahasiswa yang memliki npm1 sudah tidak ada.



Jika di cek melalui view/1, data mahasiswa yang memliki npm1 sudah tidak ada.

Penjelasan method Update pada Latihan

Untuk membuat method update, kita harus membuat syntax sql untuk update dan method updateStudent pada interface studentMapper.java . Seperti gambar dibawah ini:

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

Kemudian kita mengimplementasikan method updateStudent pada class StudentServiceDatabase dengan memanggil method updatestudent yang berada di interface Student Mapper dan menambahkan logging pada method tersebut. Seperti gambar dibawah ini:

```
@Override
public void updateStudent (StudentModel student)
{
    log.info ("data mahasiswa berhasil di update");
    studentMapper.updateStudent(student);
}
```

Setelah itu kita menambahkan link Update Data pada “viewall.html”. Seperti gambar dibawah ini:

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

Kemudian kita membuat halaman baru yang bernama “form-update.html”. Halaman tersebut mirip dengan “form-add.html “. Perbedannya terletak pada action form, method post, dan tag <input> pada form, seperti gambar dibawah ini:

```
<!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">Edit Student</h1>

    <form action="/student/update/submit" method="post">
        <div>
            <input type="text" name="npm" readonly="true" th:value="${student.npm}" />
        </div>
        <div>
            <input type="text" name="name" th:value="${student.name}"/>
        </div>
        <div>
            <input type="text" name="gpa" th:value="${student.gpa}"/>
        </div>

        <div>
            <button type="submit" name="action" value="save">Update</button>
        </div>
    </form>

</body>

</html>
```

Setelah itu, kita membuat halaman baru yang bernama “success-update.html”. Halaman tersebut akan menampilkan tulisan bahwa data berhasil diupdate. Berikut ini isi html tersebut:

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

Kemudian, kita mengimplementasikan 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";
    }
}
```

Method tersebut menerima request mapping dari http://localhost:8080/student/update/{npm}. Method tersebut akan membuat objek student yang memiliki npm sama dengan input npm dari request mapping. Ketika objek student tersebut ada, maka akan menampilkan halaman "form-update.html". Jika tidak ada, maka akan menampilkan halaman "not-found.html".

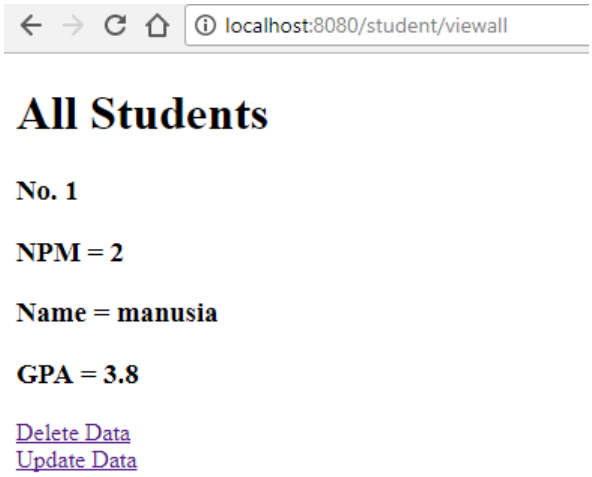
Setelah itu, kita mengimplementasikan method updateSubmit pada class StudentController.

```
@RequestMapping("/student/update/submit")
public String updateSubmit (@RequestParam(value = "npm", required =
        @RequestParam(value="name", required = false) String name,
        @RequestParam(value="gpa", required=false) double gpa) {

    StudentModel students = new StudentModel(npm, name, gpa);

    studentDAO.updateStudent(students);
    return "success-update";
}
```

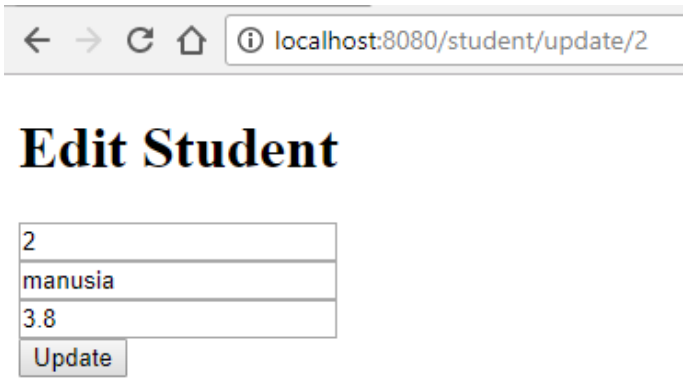
Method tersebut menggunakan post method dan menerima parameter npm, name, dan gpa. Pada method tersebut akan dibuatkan objek student yang berisikan npm, name, dan gpa sesuai paramater. Kemudian akan dipanggil method updateStudent dan mengembalikan halaman "success-update.html".

Implementasi di LocalHost :

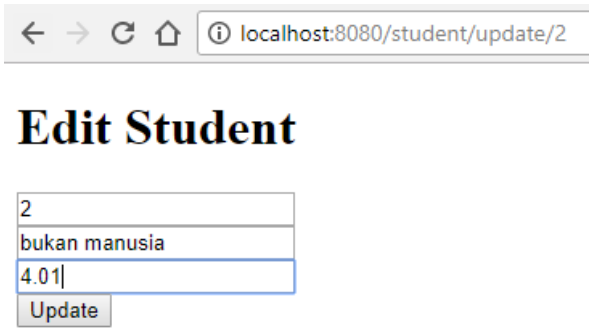
Tampilan pada viewall



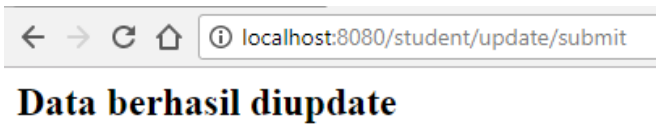
Setelah klik tulisan “Update data”.



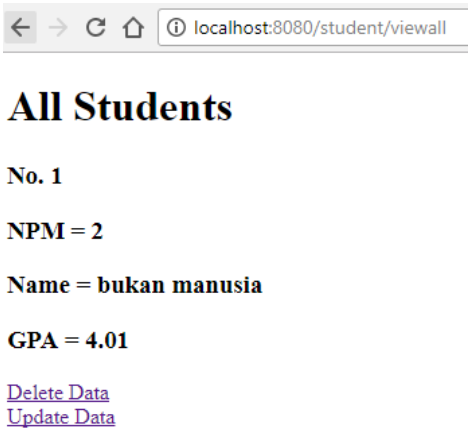
Melakukan perubahan pada nama dan gpa mahasiswa.



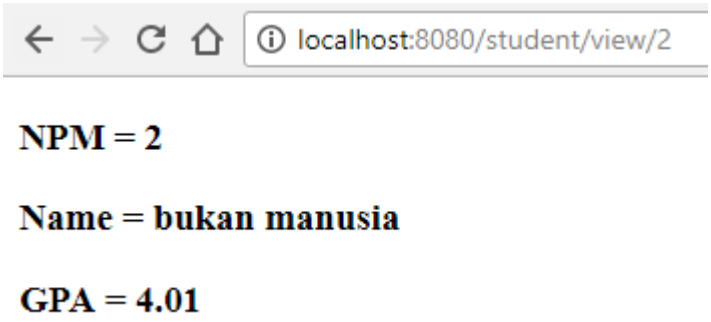
Setelah klik tombol “Update”.



Di cek melalui viewall, nama dan gpa mahasiswa yang memliki npm2 sudah berubah.



Jika di cek melalui view/2, nama dan gpa mahasiswa yang memliki npm2 sudah berubah.



Penjelasan penggunaan Object Sebagai Parameter pada Latihan

Untuk membuat parameter kita menjadi lebih *simple*, kita dapat menggunakan object (student) sebagai parameter. Pertama kita harus menambahkan th:object="${student}" pada tag <form> di view dan th:field="*{[nama_field]}" pada setiap input. Seperti gambar dibawah ini:

```
<form th:object="${student}">
    <H3 th:text="'NPM = ' + ${student.npm}" th:field="*{npm}">Student NPM</H3>
    <h3 th:text="'Name = ' + ${student.name}" th:field="*{name}">Student Name</h3>
    <h3 th:text="'GPA = ' + ${student.gpa}" th:field="*{gpa}">Student GPA</h3>
    </form>
```

Setelah itu kita mengubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel, seperti gambar dibawah ini:

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

    studentDAO.updateStudent(student);
    return "success-update";
}
```

Jawaban dari pertanyaan pada tutorial

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.

Jawab:

Untuk mencegah input yang tidak diharapkan oleh database, seperti mahasiswa tidak memiliki nama ataupun gpa, Kita dapat melakukan validasi input yang optional dan required. Dapat kita dilakukan dengan cara memberikan th:if="${#fields.hasErrors('name')}" th:errors="*{name}" pada input di html serta menambahkan anotasi @Valid pada parameter method.

Sumber:

https://spring.io/guides/gs/validating-form-input/

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?

Jawab:

Hal tersebut dikarenakan jika Form untuk submit data menggunakan GET, maka data yang disubmit akan muncul pada url. Sedangkan jika menggunakan POST, data yang disubmit tidak akan muncul di url. Tentu penggunaan GET akan lebih aman dan terhindar dari ulah para hacker yang jahat.

Perlu penanganan yang sedikit berbeda pada header di controller. Pada @RequestMapping method = RequestMethod.POST untuk POST dan method = RequestMethod.GET untuk GET

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

Mungkin saja jika suatu method dapat menerima lebih dari satu jenis request method. Pada request mapping method tersebut kita dapat menambahkan lebih dari satu method, contohnya seperti berikut:

@RequestMapping(value = "/testonly", method = { RequestMethod.GET, RequestMethod.POST }

Sumber:

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