

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792

TUTORIAL 4

1. Hasil Ringkasan Materi

Tujuan Tutorial 4 ini hampir mirip dengan tutorial 3 minggu kemarin, yaitu menambahkan dan menghapus data ke/ dari Objek List. Namun, Perbedaannya adalah pada tutorial ini ada tambahan fungsi update dan juga data yang diinput, update dan delete disimpan ke dalam database menggunakan method post dengan parameter static dan Object pada variabel RequestParam. Koneksi ke dalam database menggunakan satu file bernama application properties yang diset port, alamat database, username, password dan lain-lain.

Hal yang paling ditekankan pada tutorial ini adalah perbandingan penggunaan parameter static dengan parameter berbentuk object. Sangat terlihat efisiensinya pada file control dimana parameter yang dikirimkan dari view merupakan sebuah objek dimana di dalam nya ada variabel variabel lain (seperti npm, nama dan gpa). Sehingga penggunaan RequestParam dapat digantikan dengan sanagat efisien.

2. Pertanyaan

- a. 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.

Jawaban :

Cara untuk melakukan validasi input dari form post menggunakan parameter objek tanpa menggunakan attribute required adalah dengan menambah sintaks *@Valid* dan objek *BindingResult* pada parameter kondisi method. Selain itu menambahkan sintaks *@NotNull* pada konstruktor *name* dan *gpa* seperti gambar berikut

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
    @Valid @ModelAttribute StudentModel student, BindingResult bindingresult)
{
    if(bindingresult.hasErrors()) {
        return "not-found";
    }else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
}
```

```
@NotNull
private String name;

@NotNull
private double gpa;
```

- b. 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?

Jawaban :

Alasan utamanya adalah mengenai keamanan data, dimana untuk method *post* tidak menampilkan data yang dikirim ataupun diambil pada URL request seperti pada method *get*. Untuk penanganannya perlu ditambahkan sintaks `method = RequestMethod.POST` pada anotasi *@RequestMapping* di file Controller.

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792

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

Jawaban :

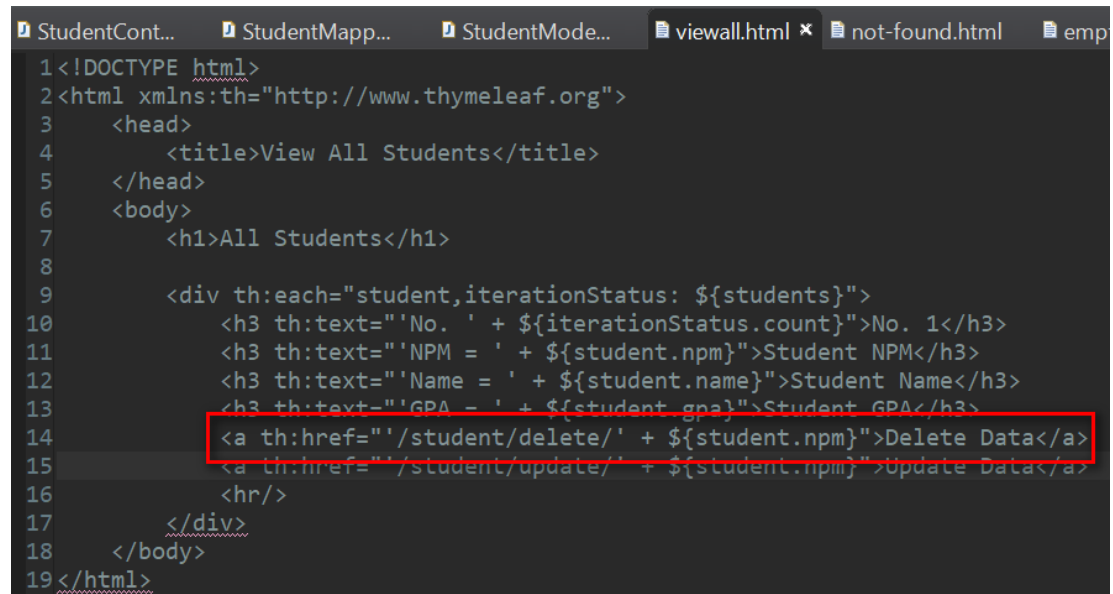
Bisa, tetapi saat menjalankannya harus satu per satu.

3. Method Delete

Pada method ini akan dilakukan penghapusan data yang sudah ada di dalam database melalui file view dengan menambahkan link untuk menghapus pada setiap data yang ditampilkan.

Beberapa sintaks yang ditambahkan antara lain,

a. File view-all.html

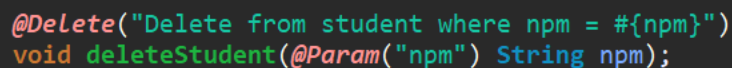


```
1<!DOCTYPE html>
2<html xmlns:th="http://www.thymeleaf.org">
3  <head>
4    <title>View All Students</title>
5  </head>
6  <body>
7    <h1>All Students</h1>
8
9    <div th:each="student,iterationStatus: ${students}">
10      <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
11      <h3 th:text="'NPM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

Sintaks di atas untuk menambahkan link delete dan mengarahkannya ke @RequestParam yang sudah diset.

b. Class StudentMapper

Pada kelas ini ditambahkan query untuk melakukan delete data ke database berdasarkan value yang dikirimkan dari file viewall.html ketika tombol delete ditekan. Kemudian mengembalikan parameter ke method deleteStudent di dalam file StudentController. Sintaksnya adalah :



```
@Delete("Delete from student where npm = #{npm}")
void deleteStudent(@Param("npm") String npm);
```

c. Class StudentServiceDatabase

Pada kelas ini ditambahkan sintaks berikut untuk menyimpan log pada compiler pada setiap stepnya. Juga digunakan untuk memanggil method deleteStudent pada Class StudentMapper.

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792

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

d. Class StudentController

Kelas ini berfungsi untuk menerima Mapping Request dari file view pada saat ditekan tombol delete data. Parameter yang diterima berupa npm untuk selanjutnya dilakukan pengecekan ke objek student yang menyimpan semua data ada di dalam database. Jika npm yang dikirimkan ada di database, proses delete akan dilakukan.

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

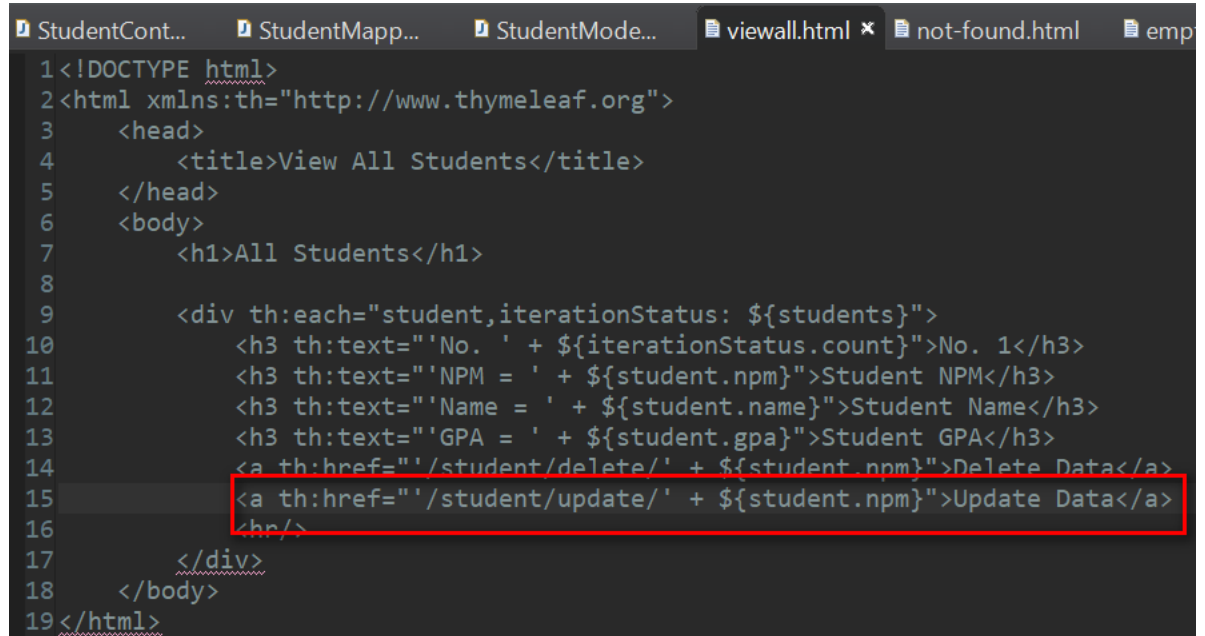
    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

4. Method Update

Method update ini digunakan untuk melakukan perubahan data nama dan gpa dari data mahasiswa yang sudah ada di dalam database. Beberapa perubahan kode yang diperlukan untuk menjalankan fungsi ini antara lain,

a. File viewall.html

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792



```
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>
15      <a th:href="'/student/update/' + ${student.npm}">Update Data</a>
16    </div>
17  </body>
18</html>
```

Sintaks di atas untuk menambahkan link update dan mengarahkannya ke @RequestParam yang sudah diset.

- b. Method updateStudent pada interface StudentService
Memberikan method interface updateStudent.

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

- c. Method updateStudent pada StudentServiceDatabase
Method yang ditambahkan berfungsi untuk menyimpan log

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

- d. Method Update pada StudentWrapper

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

Fungsinya adalah menyimpan query dan mengirimkan objek student ke controller.

- e. Penambahan form-update

Fungsi nya adalah tempat menampilkan data yang sudah dipilih untuk diupdate, merubah data dan mengirimkan value yang sudah dirubah ke controller dengan kolom npm yang tidak dapat dirubah sama sekali.

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792

```
<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:
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="*{nam
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="*{gpa}"/>
        </div>
    </form>

```

f. Method update pada controller

Berfungsi untuk menangkap value yang dikirimkan oleh data yang dipilih untuk diupdate (berupa npm) dan mengarahkan pada halaman form-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";
    }
}
```

g. Method updateSubmit pada controller

Berfungsi untuk menangkap value parameter npm, gpa, name yang dikirimkan oleh form-update (data yang sudah diupdate) untuk kemudian dilakukan eksekusi ke query dengan menggunakan method POST.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@RequestParam(value = "npm", required = false) String npm,
    @RequestParam(value = "name", required = false) String name,
    @RequestParam(value = "gpa", required = false) double gpa) {
    StudentModel student = new StudentModel(npm, name, gpa);
    studentDAO.updateStudent(student);

    return "success-update";
}
```

5. Method menggunakan Object sebagai Parameter

Perbedaannya jika menggunakan RequestParameter per field data adalah jika terlalu banyak deklarasi variabel yang dilakukan update/input akan lebih banyak juga. Sehingga akan membuat baris kode tidak efisien. Untuk itu parameter GPA disimpan ke dalam objek student bertipe data StudentModel untuk kemudian dimasukkan dieksekusi oleh controller menggunakan query yang ada dengan parameter yang otomatis sudah termapping.

Nama : Fadly Muhammad Ridho
Kelas : Ekstensi
NPM : 1606954792

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
    @Valid @ModelAttribute StudentModel student, BindingResult bindingresult)
{
    if(bindingresult.hasErrors()) {
        return "not-found";
    }else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
}
```