Teuku Amrullah Faisal
1506723231
APAP - C

Yang dipelajari pada tutorial ini

Method pada Latihan Menambahkan Delete

- Pada viewall.html tambahkan

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

Di dalam div dengan th:each

```
<body>
    <h1>All Students</h1>

    <div th:each="student,iterationStatus: ${students}">
        <a th:href="'/student/delete/' + ${student.npm}"> Delete Data</a><br/><br/>
        <a th:href="'/student/update/' + ${student.npm}"> Update Data</a><br/>
        <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>
        <hr/>
    </div>
</body>
```

- Tambahkan method deleteStudent yang ada di class StudentMapper
    - Tambahkan method delete student yang menerima parameter NPM.
    - Tambahkan annotation delete di atas dan SQL untuk menghapus
    - Lengkapi script SQL untuk menghapus mahasiswa dengan NPM tertentu.

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

- Lengkapi method deleteStudent yang ada di class StudentServiceDatabase
    - Tambahkan log untuk method tersebut dengan cara menambahkan
    - Panggil method delete student yang ada di Student Mapper

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

- Lengkapi method delete pada class StudentController

Teuku Amrullah Faisal
1506723231
APAP - C

- Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found
- Jika berhasil delete student dan tampilkan view delete

```
@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 {
        return "not-found";
    }

}
```

Method pada Latihan Menambahkan Update

- Tambahkan method updateStudent pada class StudentMapper
  - Parameternya adalah StudentModel student
  - Annotationnya adalah @Update
  - Lengkapi SQL *update* -nya

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

- Tambahkan method updateStudent pada interface StudentService

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

- Tambahkan implementasi method updateStudent pada class StudentServiceDatabase.
  Jangan lupa tambahkan logging pada method ini.

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

Teuku Amrullah Faisal
1506723231
APAP - C

- Tambahkan link Update Data pada viewall.html

```
<body>
    <h1>All Students</h1>

    <div th:each="student,iterationStatus: ${students}">
        <a th:href="'/student/delete/' + ${student.npm}"> Delete Data</a><br/><br/>
        <a th:href="'/student/update/' + ${student.npm}"> Update Data</a><br/>
        <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>
        <hr/>
    </div>
</body>
```

- Copy view form-add.html menjadi form-update.html .
    - Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.
    - Ubah action form menjadi /student/update/submit
    - Ubah method menjadi post
    - Untuk input npm ubah menjadi

```
<input type="text" name="npm" readonly="true" th:value="${student.npm}"/>
```

readonly agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada.

Input name ubah menjadi

```
<input type="text" name="name" th:value="${student.name}" />
```

Lakukan hal yang sama untuk GPA.

```
<h1 class="page-header">Update 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}"/>
    </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">Save</button>
    </div>
</form>
```

Teuku Amrullah Faisal
1506723231
APAP - C

- Copy view success-add.html menjadi success-update.html .
    - Ubah keterangan seperlunya

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

- Tambahkan method update pada class StudentController
    - Request mapping ke /student/update/{npm}
    - Sama halnya seperti delete, lakukan validasi.
    - Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view 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 {
        return "not-found";
    }
}
```

- Tambahkan method updateSubmit pada class StudentController
    - Karena menggunakan post method maka request mappingnya adalah sebagai berikut:

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
```

    - Header methodnya adalah sebagai berikut:

```
public String updateSubmit(
                    @RequestParam(value = "npm", required = false) String npm,
                    @RequestParam(value = "name", required = false) String name,
                    @RequestParam(value = "gpa", required = false) double gpa)
```

    - Lengkapi method dengan memanggil method update Student dan kembalikan view success-update

Teuku Amrullah Faisal
1506723231
APAP - C

```
@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) {

    studentDAO.updateStudent(new StudentModel(npm, name, gpa));

    return "success-update";
}
```

Method pada Latihan Menambahkan Object sebagai Parameter

- Tahapannya kurang lebih sebagai berikut:
    - Menambahkan th:object=”${student}” pada tag <form> di view
    - Menambahkan th:field=”*{[nama_field]}” pada setiap input
    - Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel
    - Tes lagi aplikasi Anda.

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

    studentDAO.updateStudent(student);

    return "success-update";
}
```

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.
   Diperlukan atau tidaknya suatu validasi bergantung pada penggunaannya. Untuk kasus update data pada tutorial ini, validasi diperlukan. Untuk melakukan validasi dapat dilakukan dengan menambahkan anotasi @NotNull untuk instance variable yang required dan tidak menulis anotasi pada instance variable yang opsional di class StudentModel. Kemudian, tambahkan anotasi @Valid pada parameter method updateSubmit pada class StudentController. Selengkapnya dapat dilihat pada gambar di bawah ini.

Teuku Amrullah Faisal
1506723231
APAP - C

```
public class StudentModel
{
    @NotNull
    private String npm;

    @NotNull
    private String name;

    @NotNull
    private double gpa;

}
```

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

    studentDAO.updateStudent(student);

    return "success-update";
}
```

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method?
   Data yang dikirim oleh method GET tidak boleh lebih dari 2047 karakter sedangkan method POST tidak terbatas. Method GET juga menyebabkan data yang akan dikirimkan ditampung pada URL sedangkan method POST akan langsung mengirimkan datanya ke server. Dengan menggunakan POST, data yang dikirimkan dapat dicegah penyalahgunaannya karena tidak dapat melihat datanya melalui URL.
   Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?
   ***Best practice*-nya adalah perlu penanganan berbeda untuk masalah tersebut. Hal tersebut dapat dilakukan dengan menambah method = RequestMethod.POST pada @RequestMapping. Selengkapnya dapat dilihat pada gambar di bawah ini.**

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

    studentDAO.updateStudent(student);

    return "success-update";
}
```

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