Tutorial 4 - Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Latihan Menambahkan Delete

1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.

2. Pada viewall.html tambahkan

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

3. Tambahkan method deleteStudent yang ada di class StudentMapper

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

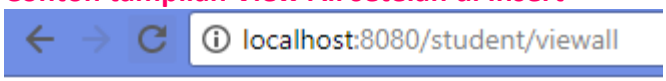
4. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

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

5. Lengkapi method delete pada class StudentController

```
@RequestMapping(value = {"/student/delete", "/student/delete/{npm}"})
public String delete (@PathVariable Optional<String> npm, Model model)
{
    if (!npm.isPresent() || studentDAO.selectStudent (npm.get()) == null) {
        String npm2 = npm.isPresent() ? npm.get() : "Tidak Ada";
        model.addAttribute ("npm", npm2);
        return "not-found";
    } else {
        studentDAO.deleteStudent (npm.get());
        return "delete";
    }
}
```

Contoh tampilan View All setelah di Insert



All Students

No. 1

NPM = 1506689490

Name = EJF

GPA = 3.7

Delete Data

No. 2

NPM = 1506721756

Name = Gheafany Widyatika Putri

GPA = 3.4

Delete Data

Contoh tampilan jika dilakukan delete NPM 123

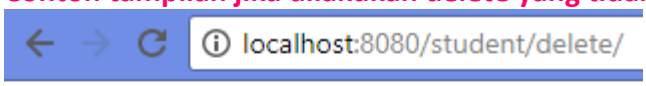
Data berhasil dihapus

Contoh tampilan jika dilakukan delete NPM 123 yang kedua kalinya

Student not found

NPM = 1506689490

Contoh tampilan jika dilakukan delete yang tidak memasukan NPM di URL



Student not found

NPM = Tidak Ada

Penjelasan method Delete:

Ketika URL /student/delete atau /student/delete/{npm} dipanggil, maka akan dilakukan pengecekan, jika NPM tidak terdapat pada URL maka akan menampilkan halaman notfound.html yang akan mem-*print* "Student Not Found ; NPM = Tidak ada", sama halnya dengan NPM yang dimasukan tidak ada di dalam *database,* maka akan menampilkan halaman notfound.html yang akan mem-*print* "Student Not Found ; NPM = (sesuai dengan NPM yang dimasukan dalam URL" sedangkan jika NPM ada maka akan dicari objek *student* yang memiliki NPM tersebut, jika ada maka objek student tersebut akan dihapus dengan mengeksekusi Query method *delete* dan akan menampilkan halaman delete.html

Latihan Menambahkan Update

Selama ini Anda masih menggunakan method GET untuk melakukan masukan. Pada update kali ini, Anda akan mencoba menggunakan method POST.

1. Tambahkan method updateStudent pada class StudentMapper

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

2. Tambahkan method updateStudent pada interface StudentService

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

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

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

4. Tambahkan link Update Data pada viewall.html

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

5. Copy view form-add.html menjadi form-update.html

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

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

</body>

</html>
```

6. Copy view success-add.html menjadi success-update.html

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

7. Tambahkan 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";
    }
}
```

8. Tambahkan method updateSubmit pada class StudentController

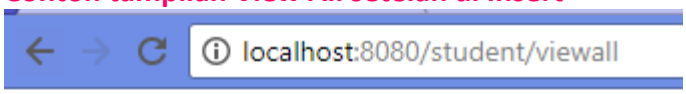
```
@RequestMapping (value = "/student/update/submit" , method = RequestMethod.POST)
public String updateSubmit (
        @RequestParam ( value = "npm" , required = false ) String npm ,
        @RequestParam ( value = "name" , required = false ) String name ,
        @RequestParam ( value = "gpa" , required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

    StudentModel student = new StudentModel(npm, name, gpa);
    studentDAO.updateStudent(student);
    return "success-update";

}
```

Penjelasan Method Update:
Ketika URL /student/update/{npm} dipanggil, method *update* akan mengecek apakah terdapat objek *student* dengan NPM tersebut di dalam *database*, jika tidak ada maka akan mengembalikan notfound.html, namun jika NPM tersebut ada di dalam *database* maka form-update.html untuk mulai mengisikan perubahan kemudian data-data *student* akan dikirim melalui model. Apabila tombol *submit* telah ditekan maka akan masuk kedalam method updateSubmit yang kemudian akan return success-update untuk menampilkan halaman success-update.html

Contoh tampilan View All setelah di Insert



All Students

No. 1

NPM = 150668457

Name = Rosa

GPA = 3.97

Delete Data
Update Data

No. 2

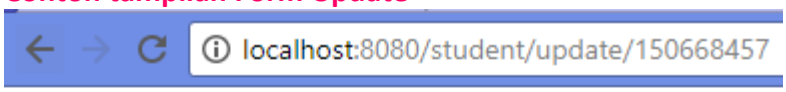
NPM = 1506721756

Name = Gheafany Widyatika Putri

GPA = 3.4

Delete Data
Update Data

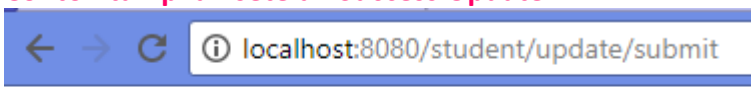
Contoh tampilan Form Update



Problem Editor

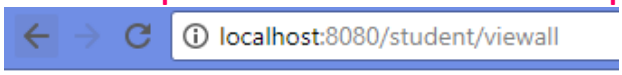
NPM 150668457
Name Rosalinda
GPA 3.5
Save

Contoh tampilan setelah Success Update



Data berhasil diperbaharui

Contoh tampilan View All setelah dilakukan update



All Students

No. 1

NPM = 150668457

Name = Rosalinda

GPA = 3.5

Delete Data
Update Data

No. 2

NPM = 1506721756

Name = Gheafany Widyatika Putri

GPA = 3.4

Delete Data
Update Data

Latihan Menggunakan Object Sebagai Parameter

1. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.

2. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel . Metode ini lebih disarankan dibandingkan menggunakan RequestParam.

3. A. Menambahkan th:object=”${student}” pada tag <form> di view kemudian menambahkan th:field=”*{[nama_field]}” pada setiap input

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

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

</body>
```

B. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

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

    return "success-update";

}
```

Penjelasan method:

Pada method ini, mengubah penggunaan RequestParam agar menjadi object, pada kasus ini yaitu StudentModel, pada form-update.html memberikan th:object=”${student}” pada tag <form>, th:field=”*{[nama_field]} pada setiap input, mengubah param updateSubmit dengan StudentController sehingga menerima parameter berupa StudentModel menggunakan @ModelAttribute.

Pertanyaan

Beberapa pertanyaan yang perlu dijawab:

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:

   Ya validasi perlu dilakukan dengan cara menambahkan anotasi @NotNull pada properti yang bertipe String(npm & name), dan menambahkan anotasi @NotEmpty pada properti double (gpa). Kita perlu menggunakan anotasi @Valid pada object parameter untuk menghindari error parsing pada form, lalu tambahkan objek class BindingResult untuk melakukan *error checking* pada saat proses validasi.

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:

   Pada *form submit* akan lebih aman menggunakan POST karena URL POST tidak tersimpan di dalam *browser* sehingga menghindari pengaksesan URL berkali-kali yang dapat menyebabkan *server down*. Ya perlu, kita perlu menghapus parameter method pada @RequestMapping karena defaultnya adalah method GET. Setiap parameter method memiliki RequestMethod memiliki parameter yang berbeda (RequestParameter.GET dan RequestParameter.POST)

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

   Jawab:

   Hal tersebut mungkin terjadi. Beberapa caranya antara lain melakukan pengecekan *request type* pada *method body controller* atau bisa juga dengan cara menambahkan method = { RequestMethod.GET, RequestMethod.POST } pada parameter @RequestMapping

Hal yang saya pelajari dari tutorial ini

Pada tutorial kali ini, saya belajar bagaimana cara mengkoneksikan spring framework dengan *database* dan belajar bagaimana cara merangkai aplikasi dengan prinsip OOP serta tak lupa dengan MVC. Ternyata kita juga dapat menggunakan objek sebagai parameter dalam menerima http request.