Nama : Yudistira Ramadhan
Kelas : APAP-B
NPM : 1506689635

Tutorial 4

Hal yang dipelajari

Pada tutorial kali ini saya mempelajari bagaimana menghubungkan aplikasi dengan database. Hal lain yang saya pelajari adalah penggunaan Slf4j dalam penggunaan variable log dan bagaimana cara penggunaan method Post dan GET dengan menggunakan parameter bebas atau object.

Method Delete

Method delete berfungsi untuk menghapus data student dari database. Untuk melakukan fungsi delete, method delete perlu dibuat di beberapa kelas, yaitu pada kelas StudentController, StudentService, StudentServiceDatabase, dan StudentMapper. Method delete pada controller berfungsi untuk mengelola HTTP request. Pertama method ini melakukan pengecekan berdasarkan NPM yang di input dari parameter apakah student dengan NPM tersebut terdapat di database atau tidak. Bila NPM tidak ada maka akan menampilkan halaman not-found. Bila NPM ada, maka method delete yang berada pada pada controller melakukan trigger ke method delete yang berada pada DAO dan akan menghapus data tersebut dari database.

Screenshoot method delete pada 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";
}
```

Method delete pada StudentMapper berguna untuk menginisialisasi fungsi query yang dibutuhkan untuk menghapus data dari database.

Screenshoot method delete pada StudentMapper

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

Nama : Yudistira Ramadhan
Kelas : APAP-B
NPM : 1506689635

Method delete pada StudentService menjalankan perintah method delete yang bersifat void

Screenshoot method delete pada StudentService

```
void deleteStudent (String npm);
```

Method delete pada StudentServiceDatabase menjalankan perintah method delete yang dibuat pada kelas StudentMapper

Screenshoot method delete pada StudentServiceDatabase

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

Method Update

Method update berfungsi untuk mengubah data nama dan gpa dari student. Terdapat beberapa method update untuk melakukan update data, yaitu pada kelas StudentController, StudentMapper, StudentService, dan StudentServiceDatabase. Pada kelas StudentController method update berguna untuk mengelola HTTP Request dan menerima parameter yang dikirimkan. Pertama method ini melakukan pengecekan NPM pada database, bila NPM tersebut valid maka dapat melakukan update. Bila tidak valid maka akan menampilkan halaman not-found. Method ini mentrigger method yang berada pada DAO. Untuk melakukan eksekusi pada database dibuat method updateSubmit yg berada pada StudentController.

Screenshoot method update pada StudentController

```
@RequestMapping("/student/update/{npm}")
public String update(@PathVariable(value = "npm") String npm, Model model) {

    StudentModel student = studentDAO.selectStudent (npm);

    if(student == null) {
        model.addAttribute("npm", npm);
        return "not-found";
    }
    model.addAttribute("student", student);
    return "form-update";
}
```

Nama : Yudistira Ramadhan
Kelas : APAP-B
NPM : 1506689635

Screenshoot method updateSubmit pada 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){

    studentDAO.updateStudent(npm, name, gpa);

    return "success-update";
```

Screenshoot method update pada StudentMapper

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

Screenshoot method update pada StudentService

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

Screenshoot method update pada StudentServiceDatabase

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

Method updateSubmit dengan Object

Method update ini melakukan fungsi update tetapi dengan parameter berupa object yaitu StudentModel. Metode ini sangat berguna apabila terdapat banyak parameter yang perlu di inisialisasikan. Dengan menggunakan object kita bisa langsung menerima semua parameter yang sudah dibuat di method constructionnya.

Screenshoot method update dengan menggunakan object sebagai parameternya

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

Nama : Yudistira Ramadhan
Kelas : APAP-B
NPM : 1506689635

Jawaban: Validasi diperlukan. Untuk memvalidasi input dapat dilakukan ketika mendeclare sebuah variable pada kelas object tersebut dengan menambahkan anotasi. Contoh anotasi untuk melakukan validasi pada variable di class object adalah @Size(), @NotNull, @Min().

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method disbanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?
   Jawaban: salah satu alasan dalam penggunaan method Post dibandingkan dengan method GET adalah dari sisi keamanannya. Karena dengan menggunakan method POST data disimpan dalam body request sehingga data tidak terlihat langsung. Berbeda dengan penggunaan method GET dimana data ditampilkan pada URLnya. Agar method POST dapat dieksekusi perlu penambahan attribute RequestMethod.POST supaya ketika terdapat HTTP yang merequest POST dapat di terima.

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Jawaban: Memungkinkan, dengan mendefinisikan sekaligus attribute-attribute method yang diinginkan. Contohnya dengan RequestMethod.GET, RequestMethod.POST