Nama : Andre Ramadhani

NPM : 1506689484

Kelas : APAP – A

Write-Up Tutorial 4

Pada tutorial kali ini, saya mempelajari tentang *debugging* serta menyimpan data di *database*.

Pertanyaan

1. 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.

   Validasi dapat dilakukan di *controller*. Jika data yang masuk tidak sesuai dengan kebutuhan (misalnya kosong atau panjang tidak sesuai), maka return *error message*. Validasi diperlukan ketika ada data yang memiliki bentuk spesifik dan bentuk tersebut harus dipenuhi.

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?

   Jika form disubmit dengan method GET, maka semua data (parameter) yang dikirim akan terlihat di URL browser. Sangat tidak baik apabila kita mengirimkan informasi sensitif atau form yang memiliki field yang sangat banyak dengan method GET, maka digunakan method POST. Jika form disubmit dengan method berbeda, perlu ada penanganan di header method untuk melakukan spesifikasi dari request.

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

   Tentu saja. Method dapat menerima lebih dari satu jenis request, namun perlu penanganan khusus untuk setiap jenis request.

Method Delete

Untuk melakukan delete, beberapa method yang dibuat adalah

- **Delete pada class StudentMapper**

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

Method ini berasal dari interface yang nantinya akan diimplementasikan pada StudentServiceDatabase. Pada method ini, terdapat anotasi berupa SQL query yang akan dieksekusi di database.

- **Delete pada class StudentServiceDatabase**

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

Method ini menjalankan SQL query yang ada pada StudentMapper.

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

Method ini menerima request dari browser dan mengembalikan page HTML. Jika NPM mahasiswa tidak ditemukan, maka return not found. Masukkan data ke database dan return page sukses jika sebaliknya.

Output

All Students

No. 1

NPM = 131848

Name = wasabi-chan

GPA = 3.41

Delete Data
Update Data

No. 2

NPM = 141414

Name = freya

GPA = 4.0

Delete Data
Update Data

Data berhasil dihapus

All Students

No. 1

NPM = 131848

Name = wasabi-chan

GPA = 3.41

Delete Data
Update Data

Student not found

NPM = 99999

Method Update

Untuk membuat fungsi update, method-method yang diperlukan adalah:

- **Update pada StudentMapper**

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

Method ini berisi SQL query untuk mengupdate data yang ada pada database.

- **Update pada StudentServiceDatabase**

```
public void updateStudent (String npm, String name, double gpa) {
    log.info("student " + npm + " updated");
    studentMapper.updateStudent(npm, name, gpa);
}
```

Method yang digunakan untuk menjalankan SQL query dari fungsi update.

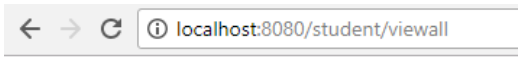
- **Update pada StudentController**

```
@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";
    }
}

@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";
}
```

Method pertama melakukan validasi terkait NPM mahasiswa. Jika ditemukan, maka server mengeluarkan page form untuk update. Return page not found jika sebaliknya

Method kedua menerima request dari page form update. Parameter-parameter yang masuk kemudian akan diteruskan ke bagian service. Method ini me-return page sukses update.

Output



All Students

No. 1

NPM = 131311

Name = wasabi-chan

GPA = 3.55

Delete Data
Update Data

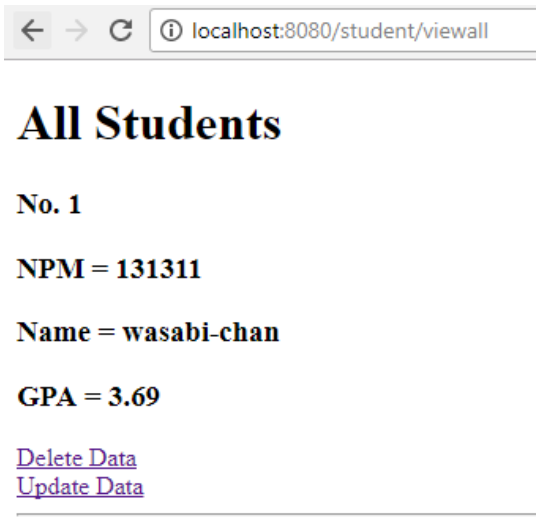
Student not found

NPM = 131818181

Update Data

NPM 131311
Name wasabi-chan
GPA 3.55
Update

Data berhasil diubah



Object sebagai Request Parameter

```
@RequestMapping(value = "student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@ModelAttribute StudentModel student) {
    studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
    return "success-update";
}
```

Implementasi method ini hampir sama dengan method update sebelumnya, hanya parameter pada method diganti menjadi objek. Untuk melakukan ini, dibutuhkan operasi Thymeleaf yang diimplementasikan pada HTML. Method ini akan mengeluarkan output yang sama dengan method update yang sebelumnya.