WriteUp - Tutorial 4 | Luthfi Abdurrahim | 1406557535 | kelas APAP - A

Latihan

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.

   Jawab : Validasi bisa diimplementasikan di Controller / back-end dengan memeriksa masing-masing value pada object yang dikirimkan dari form. Validasi penting untuk memastikan data yang dimasukan sesuai dengan format database (seperti tipe data dll) dan logika pemrograman yang dibuat

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: method POST dapat dikatakan lebih aman jika dibandingkan dengan method GET. Karena pada method POST, data yang dikirimkan dari *client-side* tidak dapat dibaca secara terang-terangan oleh user pada *link url* secara langsung. Untuk penangan di header atau body method controller perlu penanganan berbeda untuk jenis method form yang berbeda

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

   Jawab: bisa, tinggal menambahkan array pada method values, seperti ini contohnya :

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

   Sumber : https://stackoverflow.com/questions/16602929/requestmethod-post-and-get-in-the-same-controller

Buat sebuah file write-up. Jelaskan apa saja hal yang Anda pelajari dari tutorial ini.

Pada Tutorial ini saya sangat lama dalam hal debugging hingga konfigurasi environment dalam koneksi ke database dan menggunakan library luar. Hal ini sangat menguras waktu sekitar lima jam lebih, sehingga pada akhirnya saya menanyakan kepada ahlinya dan alhasil bisa berhasil dalam waktu satu jam. Dengan error tersebut, experience dan pengetahuan saya semakin luas karena tidak terpaku pada tutorial saja dan bisa bereksplorasi lebih. Hal yang saya pelajari yaitu, mengkoneksi ke database sql, melakukan CRUD ke database, menggunakan library luar dan menyambungkannya dengan spring boot, hingga menggunakan cara singkat dalam form dengan spring boot, yaitu dengan menerapkan objek nya.

Jawab pertanyaan pada bagian Pertanyaan dan penjelasannya. Cantumkan juga penjelasan

Anda terhadap hal-hal barikut:

- Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan

  - **Pada StudentMapper.java**

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

  - **Pada StudentServiceDatabse.java**

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

  - **Pada StudentController.java**

```
@RequestMapping("/student/delete/{npm}")
public String delete (
        Model model,
        @PathVariable(value = "npm") String npm)
{
    if (studentDAO.selectStudent (npm) != null) {
        model.addAttribute ("student", studentDAO.selectStudent (npm));
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

  - **Penjelasan:**
    Request yang dilakukan oleh *user* akan diterima dahulu pada *StudentController.java* . Melalui penerimaan pada parameter npm, lalu method pada controller akan memastikan apakah terdapat data siswa dengan npm pada parameter itu. Jika tidak tersedia maka akan mengembalikan halaman not found. Namun, jika ditemukan, maka akan mengembalikan halaman delete beserta informasi mengenai siapa yang di delete. Sebelum mengembalikan halaman delete, program akan mengeksekusi method delete student dengan query sesuai yang tertera pada studentMapper.java serta men-select student yang akan didelete untuk ditampilkan di halaman delete.

- Method yang Anda buat pada Latihan Menambahkan Update, jelaskan

  - **Pada StudentMapper.java**

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

- **Pada StudentServiceDatabase.java**

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

- **Pada StudentController.java**

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

    StudentModel students = studentDAO.selectStudent(npm);
    if(students==null) {
        return "not-found";
    }
    model.addAttribute("students",students);

    return "form-update";
}
```

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

}
```

- **Penjelasan:**

  Request yang dilakukan oleh *user* akan diterima dahulu pada StudentController.java . Melalui penerimaan parameter npm, gpa dan name. method *updateSubmit* pada controller kemudian memanggil object *StudentModel* sesuai parameter. Kemudian akan mengeksekusi method *updateStudent* dengan parameter object yang dipanggil. Method tersebut akan mengeksekusi query yang ada pada StudentMapper.java. Setelah selesai akan mengembalikan halaman success-update.

- Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan

- **Pada StudentMapper.java**

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

- **Pada StudentServiceDatabase.java**

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

- **Pada StudentController.java**

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

- **Penjelasan:**

  Penjelasan sama dengan penerapan update sebelumnya, namun perbedaan hanya terletak pada StudentController.java yang dimana objek yang menjadi parameter menerima parameter object StudentModel. Parameter tersebut didapatkan dari halaman view / *client-side* yang mengirimkan object studentModel. Parameter tersebut lalu digunakan untuk memanggil method updateStudent pada studentDAO dan mengeksekusi query update. Kemudian mengembalikan halaman success-update yang menandakan proses update berhasil.