Ari Tri Wibowo Y

1506735175

APAP - C

# Tutorial 04

## Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

## 1. Yang dipelajari dalam tutorial kali ini

Dalam tutorial kali, yang dipelajari adalah operasi *delete* dan *update* yang terhubung langsung dengan data yang ada di *database* (sql). Operasi *update* kali ini, dilakukan dengan method POST. Selain itu, dipelajari juga cara menggunakan *object* sebagai *parameter*. Alasan menggunakan *object* adalah lebih praktis ketika ada banyak *parameter* pada suatu *method*.

Selain itu, hal baru yang dipelajari dalam tutorial kali ini, adalah penggunaan berbagai library eksternal yakni Lombok dan MyBatis. Dalam tutorial ini, Library Lombok berguna untuk melakukan *debugging* dengan menggunakan anotasi @SLf4j. Sementara itu library MyBatis berguna untuk menghubungkan *project* dengan *database* MySQL.

## 2. Pertanyaan dan penjelasan

1. Validasi dapat dilakukan dengan menambahkan field yang menandai bahwa attribute tersebut required atau optional.

2. Sebab jika kita mengirimkan data/nilai dengan form submit menggunakan POST, maka data/nilai variabel tidak akan tampil dalam URL. Hal ini lebih aman. Sementara itu, jika menggunakan method GET, data/nilai variabel akan terlihat dalam URL.

3. Ketika browser melakukan HTTP Request, hanya bisa melakukan satu method saja (GET, POST, PUT, dll) dalam satu waktu.

# 3. Penjelasan method pada latihan

**Method yang dibuat pada latihan menambahkan delete**

*Method deleteStudent dalam class StudentMapper*

Pada annotation delete di atas method ini, terdapat script SQL yang berfungsi untuk menghapus data mahasiswa dengan NPM tertentu pada database sql

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

*Method deleteStudent dalam class StudentServiceDatabase*

Method pada class ini berfungsi untuk memanggil method deleteStudent pada StudentMapper. Terdapat juga log yang bermanfaat ketika melakukan debugging.

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

*Method delete dalam class StudentController*

Method ini dijalankan ketika halaman /student/delete/{npm} diakses. Method akan menerima parameter berupa npm. Lalu mencari object student dengan npm yang diberikan. Apabila ditemukan, maka akan menhapus data student tersebut, jika tidak ditemukan maka akan mengembalikan halaman not-found.

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

**Method yang dibuat pada latihan menambahkan update**

*Method updateStudent pada class StudentMapper*

Pada annotation update di atas method ini, terdapat script SQL yang berfungsi untuk memperbaharui data mahasiswa pada database sql

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

*Method updateStudent pada interface StudentService*

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

*Method updateStudent pada class StudentServiceDatabase*

Method pada class ini berfungsi untuk memanggil method updateStudent pada StudentMapper. Terdapat juga log yang berguna ketika melakukan debugging.

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

}
```

*Method update dalam class StudentController*

Method ini dijalankan ketika halaman student/update/{npm} diakses, method ini menerima masukan berupa npm. Kemudian akan mencari student dengan npm tersebut lalu akan disimpan dalam object student. Jika tidak ditemukan akan mengembalikan halaman not-found. Jika berhasil ditemukan maka akan menambahkan atribut student ke dalam addAttribut lalu mengembalikan halaman form-update.

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

    StudentModel student = studentDAO.selectStudent(npm);
    if(student != null) {
        //studentDAO.updateStudent(student);
        model.addAttribute("student", student);
        return "form-update";
    }
    return "not-found";
}
```

Halaman form-update merupakan halaman yang dibuat yang berisikan form agar kita dapat menginput data yang baru untuk sebuah objek student yang akan di-update datanya. Setelah mengklik tombol update, maka halaman akan di-submit dengan method POST. Langkah ini, akan ditangai oleh method updateSubmit pada class StudentController.

*Method updateSubmit dalam class StudentController*

Method ini menerima parameter berupa npm, name, gpa yang baru. Kemudian parameter yang baru itu akan dibuatkan dalam bentuk object student. Dan akan dijalankan method updateStudent lalu mengembalikan halaman success-update.

```java
@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";

        }
```

**Method yang dibuat pada latihan menggunakan objek sebagai parameter**

*Memperbaharui form*

Menambahkan th:field=”*{[nama_field]}” di setiap input label

```
<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">Update</button>
    </div>
</form>
```

*Memperbaharui method updateSubmit di kelas StudentController*

Disini parameter yang semula panjang dan tidak rapih, cukup digantikan dengan parameter berupa student

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

public String updateSubmit (StudentModel student)
        {
            //StudentModel student = new StudentModel (npm, name, gpa);
            studentDAO.updateStudent(student);
            return "success-update";

        }
```