Tutorial 4 APAP

A. Method yang dibuat pada Latihan Menambahkan Delete

a. Method deleteStudent di class StudentMapper

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

Merupakan method yang berisi query untuk delete student dengan parameter npm.

b. Method deleteStudent di class StudentServiceDatabase

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

Merupakan method untuk delete student dengan memanggil method deleteStudent yang ada di class StudentMapper

c. Method delete di 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(npm);
                     return "delete";
              }
              else {
                     model.addAttribute ("npm", npm);
                     return "not-found";
              }
    }
```

Ketika ingin menghapus salah satu data student,maka method ini akan menerima paramaeter npm dan akan mengecek terlebih dahulu apakah student tersebut ada,jika ada data akan dihapus dan jika tidak ada,maka akan menampilkan halaman not-found.

B. Method dibuat pada Latihan Menambahkan Update

a. Method updateStudent di class StudentMapper

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

Merupakan method yang berisikan query untuk update student dengan parameter StudentModel student

b. Method updateStudent di interfase StudentService

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

Method ini nantinya akan di override di class StudentServiceDatabase agar nantinya bisa di implementasikan untuk mengupdate student.

c. Method updateStudent di class StudentServiceDatabase

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

    }
```

Method untuk update data student dengan memanggil method updateStudent pada class StudentMapper

d. Method update di 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 ("student", student);
                  return "form-update";
           }
           else {

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

Merupakan method untuk menampilkan update student. Jika data student ingin di update,dan student tersebut ditemukan(dengan paramater npm),maka halaman akan menampilkan form untuk mengupdate data student,jika student tidak terdaftar,maka browser akan menampilkan halaman not-found.

e. Method updateSubmit di 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";
    }
```

Setelah mengisi form-update,dan klik tombol update,maka method ini akan mengupdate data yang ada didatabase dengan memanggil method updateStudent yang telah di implemetasikan di class StudentServiceDatabase dan data student akan terupdate dengan menampilkan halaman update berhasil.

C. Method dibuat pada Latihan Menggunakan Object Sebagai Parameter

a. Method updateSubmit di class StudentController

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

Method ini menerima parameter berupa StudentModel student. Karena sudah menerima parameter berupa StudentModel student,maka kita tidak perlu mendefinisikan StudentModel student seperti pada methode updateSubmit.

D. 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.

   Dengan menggunakan object sebagai parameter pada form POST memang lebih aman,tetapi kita masih harus tetap melakukan validasi pada controller karena masih bisa menerima null,sehingga harus mengcover agar method tidak menerima null.

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?

   Karena form submit biasanya bersifat rahasia seperti saat mengupdate data student,jika menggunakan GET method maka akan menampilkan data update tersebut seperti gambar dibawah ini:

   Maka dari itu sebaiknya form submit menggunakan POST method. Pada RequestMapping di method controller,kita tidak perlu membuat method = RequestMethod.POST atau method = RequestMethod.GET jika kita sudah membuat method apa yang akan digunakan di form seperti:

```
<form action="/student/update/submit" method="post"
th:object="${student}">
```

Tetapi,jika kita membuat RequestMethod yang berbeda dari apa yang kita buat di form,maka akan seperti ini:

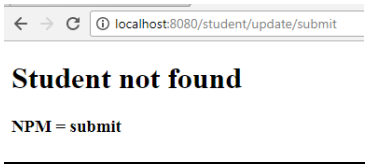
Form :

```
<form action="/student/update/submit" method="post" th:object="${student}">
```

Controller:

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

Menghasilkan:



Jadi ketika kita ingin membuat RequestMethod pada controller,maka harus sesuai dengan method apa yang kita gunakan pada form.

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

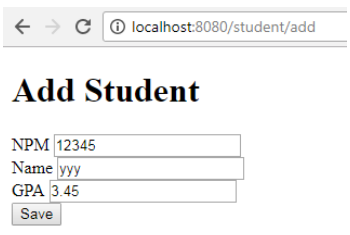
   Tidak bisa. Satu method hanya bisa menerima satu jenis method saja,yaitu GET atau POST

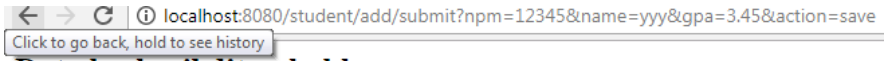
```
120
121    @RequestMapping(value ="/student/update/submit", method = RequestMethod.POST , method = RequestMethod.GET)
122    public String updateSubmit (@ModelAttribute StudentModel student) {
123        studentDAO.updateStudent(student);
124        return "success-update";
125    }
```

```
121  Multiple markers at this line
       - Duplicate attribute method in annotation @RequestMapping
       - Duplicate attribute method in annotation @RequestMapping
122
123
124        return "success-update";
```

E. Screenshot

Add student:





Data berhasil ditambahkan

Viewall:

All Students

No. 1

NPM = 12345

Name = yyy

GPA = 3.45

Delete Data
Update Data

No. 2

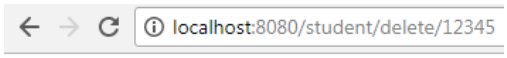
NPM = 34567

Name = aaaaaa

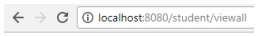
GPA = 3.45

Delete Data
Update Data

Dilakukan delete:



Data berhasil dihapus



All Students

No. 1

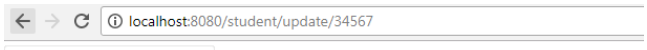
NPM = 34567

Name = aaaaaa

GPA = 3.45

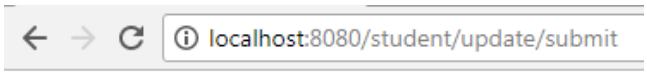
Delete Data
Update Data

Dilakukan update:



Update Student

34567 bbbbbb 3.45 Update



Data berhasil diupdate

All Students

No. 1

NPM = 34567

Name = bbbbbb

GPA = 3.45

Delete Data
Update Data