**Muhammad Umar Farisi (1506725552) | APAP – C | Tutorial 4**

## Hal yang Dipelajari

Hal yang dipelajari pada tutorial ini adalah menghubungkan aplikasi dengan database menggunakan mybatis. Selain itu, mempelajari juga mengenai penanganan HTTP POST dengan parameter bertipe biasa dan bertipe object.

## Method pada Latihan Menambahkan Delete

Method **delete** pada kelas **StudentController** berfungsi untuk menangani HTTP request GET terhadap endpoint /student/delete/{npm}. Dalam method ini, akan mengecek apakah npm yang berada di path terdapat di database atau tidak. Jika npm tersebut ada, method deleteStudent yang ada di *object* service (studentDAO) akan dieksekusi dengan tujuan menghapus data student dari database berdasarkan npm tersebut. Jika npm tersebut tidak ada, halaman yang akan ditampilkan adalah not-found yang berisi pemberitahuan bahwa npm tersebut tidak ada di database. Berikut ini *screenshot* dari method ini:

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

    if(studentDAO.selectStudent(npm) == null) {
        return "not-found";
    }
    studentDAO.deleteStudent (npm);
    return "delete";
}
```

Method **deleteStudent** yang berada di interface **StudentService** akan diimplementasi oleh kelas StudentServiceDatabase. Berikut ini *screenshot* dari method ini:

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

Method **deleteStudent** yang berada di kelas **StudentServiceDatabase** bertujuan menghapus student berdasarkan npm yang diberikan melalui parameter method ini. Untuk melakukan hal tersebut, method ini memanggil method deleteStudent yang berada di *object* mapper (studentMapper). Berikut ini *screenshot* dari method ini:

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

Method **deleteStudent** yang berada di kelas **StudentMapper** akan menghapus student berdasarkan npm dari parameter method ini. Penghapusan tersebut berdasarkan *query* yang berada di annotation @Delete. Berikut ini *screenshot* dari method ini:

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

## Method pada Latihan Menambahkan Update

Method **update** di kelas **StudentController** berfungsi untuk menangani HTTP request GET terhadap endpoint /student/update/{npm}. Method ini mengambil student berdasarkan npmnya melalui method selectStudent dari *object* service (studentDAO). Jika student dengan npm tersebut ada di database, student tersebut akan di berikan ke form-update melalui object model dan tampilan yang akan ditampilkan adalah form-update. Jika student dengan npm tersebut tidak ada di database, tampilan yang akan ditampilkan adalah not-found yang berisi pemberitauan bahwa student dengan npm tersebut tidak ada di database. Berikut ini *screenshot* dari method ini:

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

    model.addAttribute("student", student);

    return "form-update";
}
```

Method **updateSubmit** di kelas **StudentController** berfungsi untuk menangani HTTP request POST terhadap endpoint /student/update/submit. Method ini akan menerima data student dari form-update dan mengupdatenya ke database melalui method updateStudent dari *object* service (studentDAO). Berikut ini *screenshot* dari method ini:

```
@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 **updateStudent** di kelas **StudentService** akan diimplementasi oleh kelas StudentServiceDatabase. Berikut ini *screenshot* dari method ini:

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

Method **updateStudent** di kelas **StudentServiceDatabase** berfungsi untuk mengupdate student. Untuk melakukan hal tersebut, method ini akan memanggil method updateStudent dari *object* mapper (studentMapper). Berikut ini *screenshot* dari method ini:

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

Method **updateStudent** di kelas **StudentMapper** yang berfungsi untuk mengupdate student berdasarkan query yang ada di annotation @Update. Berikut ini *screenshot* dari method ini:

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

## Method pada Latihan Menggunakan Object Sebagai Parameter

Di kelas **StudentController** membuat method **updateSubmit** yang memiliki parameter **StudentModel**. Method ini akan mengupdate data student dari form-update melewati parameter method ini. Berikut ini *screenshot* dari method ini:

```
@PostMapping("/student/update/submit")
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?
   **Jawaban :** Untuk memvalidasi nilai yang diberikan per-*field* dapat dilakukan dengan cara pemanggilan atribut (dari object yang ada di parameter) yang akan diisi *field* tersebut. Jika atribut itu kosong (string kosong), artinya adalah pengguna tidak memasukan nilai apapun ke dalam *field*. Oleh karena itu, validasi dapat dilakukan berdasarkan hal tersebut.
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?
   **Jawaban :** Data yang dikirimkan dari form submit biasanya merupakan data yang sensitif, tidak boleh diketahui seseorang, oleh karena itu biasanya form submit menggunakan method POST dimana data disimpan di body request, sedangan data yang dikirim dengan method GET di-*append* di URL-nya sehingga dapat dilihat oleh seseorang. Untuk penanganan HTTP request POST di controller, perlu menggunakan annotation @PostMapping atau @RequestMapping dengan tambahan atribut method yang bernilai RequestMethod.POST agar method tersebut dieksekusi oleh controller ketika adanya HTTP request POST.
3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   **Jawaban :** Iya, memungkinkan satu method menerima lebih dari satu jenis request dengan cara mendefinisikan atribut method di annotation @RequestMapping yang berisi jenis – jenis request method yang diinginkan. Contohnya jenis – jenis method yang diinginkan adalah GET dan POST, annotationnya akan seperti ini:
   @RequestMapping(value = “/”, method = {RequestMethod.GET, RequestMethod.POST})