TUGAS TUTORIAL 4

ARSITEKTUR DAN PEMROGRAMAN APLIKASI ENTERPRISE

Disusun oleh :

ANDESTA PUTRA

NPM. 1706106633

PROGRAM SARJANA 1 EKSTENSI SISTEM INFORMASI

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

2018

A. Ringkasan Materi

Tutorial ini berfokus pada pemanfaatan database server ke dalam sistem yang kita buat. Dengan menggunakan database server maka data yang kita inputkan ke dalam sistem akan di simpan ke dalam database. Untuk dapat melakukannya kita akan menggunakan dua library eksternal yaitu MyBatis dan Lombok. Library MyBatis digunakan untuk menghubungkan sistem yang kita buat dengan database server MySQL, sedangkan library Lombok sendiri digunakan untuk helper annotation yang akan kita buat ke dalam sistem.

B. Latihan

1. Menambahkan Method Delete

Langkah pertama yang dapat kita lakukan untuk menambahkan method delete ke dalam sistem ialah dengan membuat method deleteStudent pada class StudentMapper. Class StudentMapper sendiri merupakan class yang berisikan anotasi query database yang dipanggil melalui sebuah method. Pada class ini kita dapat menambahkan script sebagai berikut.

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

Maksud dari script diatas ialah kita akan menggunakan anotasi delete untuk memanggil query delete data dari table student dengan npm yang dituju. Untuk dapat memanggil method tersebut, maka kita perlu membuat method deleteStudent pada class StudentServiceDatabase, dimana class ini memiliki fungsi sebagai penghubung antara controller dengan class studentMapper yang memiliki akses ke database. Script dari method deleteStudent adala sebagai berikut.

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

Maksud dari script diatas ialah method deleteStudent menerima satu parameter yaitu npm, kemudian terdapat log.info ketika kita memanggil method deleteStudent tersebut. Setelah itu maka program akan memanggil method deleteStudent dari class studentMapper yang sebelumnya telah kita buat untuk mengeksekusinya kedalam database.

Selanjutnya kita akan membuat method delete pada student untuk pemanggilan method deleteStudent yang ada pada class StudentServiceDatabase ini. Adapun script nya adalah sebagai berikut.

```
@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";
    }
}
```

Method diatas memiliki anotasi RequestMapping yang digunakan sebagai routing url. Sehingga ketika pada view kita melakukan aksi menuju ke url (“/student/delete/{npm}”) maka secara otomatis controller akan menangkap url tersebut dan mengarahkannya kepada requestMapping yang cocok, pada kasus ini adalah mengarah ke method delete yang telah kita buat. Adapun parameter yang diterima ialah model, kemudian npm. Didalam method tersebut kita akan melakukan pencarian objek student dengan npm yang telah dituju kemudian menyimpannya kedalam object baru yang bernama student. Proses pencarian ini dilakukan dengan menggunakan pemanggilan method selecStudent(npm) yang terdapat pada class StudentService yang berisikan anotasi untuk menuju ke class StudentMapping dengan anotasi select sehingga program akan mencari data student dengan npm tertentu di dalam database yang kita miliki.

Selanjutnya akan dilakukan pengecekkan dimana ketika student ditemukan atau tidak bernilai null maka program akan memanggil method delelteStudent(npm) yang sebelumnya telah kita panggil kemudian memanggil view delete yang berisi bahwa delete student berhasil.

Akan tetapi jika student kosong maka nilai npm akan disimpan kedalam atribut model dengan nama npm dan akan ditampilkan pada view not-found bahwa student dengan npm tersebut tidak berhasil ditemukan/kosong.

Dari proses tersebutdapat kita ambil kesimpulan bahwa cara kerja dari sistem adalah kurang lebih sebagai berikut.

View => Controller menangkap route url => StudentServiceDatabase = > StudentMapper melakukan pengolahan database.

2. Menambahkan Method Update

Untuk method update perlakuan yang kita lakukan tidak jauh berbeda. Alur yang digunakan tetap seperti ini.

View => Controller menangkap route url => StudentServiceDatabase = > StudentMapper melakukan pengolahan database.

Pada class studentMapper kita buat method sebagai berikut.

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

Method ini berisikan query untuk melakukan update student dengan nama, gpa, serta npm yang telah diinputkan.

Kemudian kita buat method updateStudent pada class StudentServiceDatabase sebagai berikut.

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

Method ini akan memanggil method updateStudent pada class studentMapper yang telah kita buat sebelumnya.

Langkah selanjutnya adalah membuat method update untuk memanggil form update agar kita dapat memperbaharui data student yang ada.

```
@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";
    }
}
```

Method ini akan melakukan pengecekan student dengan npm yang diinputkan. Setelah itu apabila student ditemukan maka kita akan memanggil form-update dengan data student yang dilemparkan ke form tersebut sehingga nantinya di setiap field pada view form-update telah terisi data mahasiswa yang telah tersimpan pada database.

Kemudian kita buat form update yang bedanya pada form ini th:field=”${student.namafiel}” untuk menampilkan data yang telah ada sebelumnya.

Setelah itu kita setting action form update tersebut ke url /student/update/submit.

Kemudian pada controller kita buat route yang akan menangkap url tersebut dan mengarahkannya ke method updateSubmit seperti berikut.

```
@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";
}
```

Pada method tersebut berisikan parameter dari setiap field student. Kemudian kita akan membuat objek baru dengan data tersebut dan emanggil method updateStudent pada class StudentServiceDatabase yang sebelumnya telah kita buat.

3. Membuat Objek Sebagai Parameter

Latihan terakhir pada lab kali ini ialah membuat objek sebagai parameter dari method updateSubmit yang telah kita buat.

Untuk membuatnya, hal yang pertama kita lakukan ialah merubah struktur form update yang telah kita buat sebelumnya yaitu dengan menambahkan th:object=”${student}“ pada tag form dan th:field=”*{name_field}” pada setiap field pada form tersebut.

```
<form th:object="${student}" action="/student/update/submit" method="post">
    <div>
        <label for="npm">NPM</label> <input th:field="*{npm}" type="text" name="npm" readonly="true" th:value="${student.npm}"/>
    </div>
    <div>
        <label for="name">Name</label> <input th:field="*{name}" type="text" name="name" th:value="${student.name}"/>
    </div>
    <div>
        <label for="gpa">GPA</label> <input th:field="*{gpa}" type="text" name="gpa" th:value="${student.gpa}"/>
    </div>

    <div>
        <button type="submit" name="action" value="save">Save</button>
    </div>
</form>
```

Selanjutnya adalah kita akan mengganti parameter pada method updateSubmit.

```
@RequestMapping(value = "/student/update/submit", method =  RequestMethod.POST)
public String updateSubmit(@ModelAttribute StudentModel student){

    studentDAO.updateStudent (student);

    return "success-update";
}
```

Caranya cukup mudah yaitu kita hanya perlu mengganti parameter sebelumnya dengan objek student dan untuk RequestMethod menggunakan POST. Kemudian karena sudah berbentuk objek, maka kita dapat langsung mengirimnya ke method updateStudent untuk melanjutkan proses update data student tersebut.

C. 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?

   Secara default, apabila kita menggunakan objek sebgai parameter, maka semua inputan akan bersifat opsional, karena walaupun kita tidak memasukkan name student, objek dari student tetap akan dibuat.
   Kemudian apabila kita ingin melakukan validasi inputan yang required, maka kita dapat melakukan pengecekkan pada controller yang kita gunakan dengan menggunakan package BindingResult. Package ini akan membantu kita untuk melakukan validasi dengan menggunakan method hasErrors dimana nilai bindingResult.hasErrors() akan bernilai true apabila kita tidak mengisi semua field pada form update.

```
@RequestMapping(value = "/student/update/submit", method =  RequestMethod.POST)
public String updateSubmit(@ModelAttribute StudentModel student, BindingResult bindingResult){

    if(bindingResult.hasErrors())
        return "errorForm";

    studentDAO.updateStudent (student);

    return "success-update";
}
```

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?

   Pada sebuah form biasa menggunakan post apabila data yang diinputkan bersifat privacy sehingga tidak ingin ditampilkan begitu saja d url seperti apabila kita menggunakan method GET. Kemudian untuk membuat membuat form menjadi metode get kita hanya perlu merubah form atribut method pada form menjadi get, kemudian pada header controller kita perlu merubah requestMethod mejadi get.

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

   Menurut percobaan yang saya lakukakn sebuah method hanya dapat menerima satu metode saja, karena ini berkaitan dengan header atau request method yang dipakai.

```
@GetMapping(value = "/student/update/submit")
@PostMapping(value = "/student/update/submit")
```

   Misalkan kita membuat dua header pada satu method maka yang akan dibaca tetap header paling atas sehingga yang terbaca oleh sistem adalah metode get saja. Pada percobaan ini saya menggunakan metode post dan muncul error dikarenakan method hanya menerima request metode get saja.