1. Ringkasan Materi

Pada tutorial 4 digunakan tambahan 3 tools yaitu Lombok, MyBatis dan XAMPP. Libraty Lombok digunakan untuk helper anotasi pada project yang dibuat. MyBatis digunakan untuk koneksi, mapping, dan generate query dengan bantuan helper anotasi. XAMPP digunakan sebagai database (phpMyAdmin) yang akan diakses oleh project. Sudah digunakan form juga pada tutorial ini.

2. Jawaban 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? (Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend)

   Jawab :

   Cara melakukan validasi adalah sebagai berikut.

   Dengan memanfaatkan model yang telah dibuat. Model adalah cerminan struktur sebuah table pada database. Pada model tersebut bisa ditambahkan dengan standar validation annotations seperti `@NotNull` yang tidak akan mengijinkan sebuah attribute bernilai null. Kemudian pada controller yang menerima submit ditambahkan @Valid pada parameter Object dan tambahkan parameter Object `bindingResult`.

   Kemudian pada body tambahkan kondisi untuk melakukan validasi dengan menggunakan code "bindingResult.hasErrors()" jika hasil validasi error maka akan kembali ke halaman form dan jika validasi berhasil maka akan melakukan aksi selanjutnya.

   Validasi diperlukan karena form berhubungan langsung dengan database. Jika pada struktur table database terdapat field not null maka harus dipastikan bahwa pada form telah diinputkan nilai untuk field tersebut agar tidak terjadi error atau ketidaksesuaian data. Selain validasi optional atau required juga perlu untuk dilakukan validasi tipe data dan lenght yang diinputkan pada form harus sama dengan tipe data dan lenght pada struktur table database.

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 :

   Form submit biasanya berisi dengan nilai yang cukup panjang untuk setiap fieldnya sehingga akan sangat tidak efisien jika menggunakan method GET dimana aka tertampil nilai yang diinputkan pada url. Selain itu berikut beberapa kelebihan jika menggunakan method POST:

   1. Lebih aman karena data yang dikirim tidak terlihat
   2. Dapat mengirim data dalam jumlah besar.

3. Dapat mengirim berbagai jenis data seperti gambar, file, dll, tidak harus teks

   Perlu penanganan yang berbeda pada header method di controller dengan menambahkan method = RequestMethod.POST pada RequestMapping.

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

   Jawab : Tidak bisa.

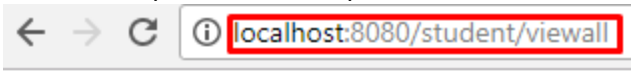
3. Penjelasan Fitur Delete

a. Tambah link Delete.

Untuk dapat menjalankan fitur Delete digunakan link pada fitur viewall yang langsung men-direct ke method delete. Berikut adalah tag html yang ditambahkan pada viewall.html.

```
<a th:href="'/student/delete/' + ${student.npm}" > Delete Data</a><br/>
```

Berikut tampilan hasil dari penambahan code.



All Students

No. 1

NPM = 12345

Name = ada1

GPA = 3.1

Delete Data

b. Tambah method deleteStudent pada StudentMapper.

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

Pada interface StudentMapper ditambahkan nama method yang digunakan untuk melakukan penghapusan data student berdasarkan npm.

Anotasi @Delete digunakan untuk mapping query delete. Didalam anotasi tersebut terdapat query sql untuk melakukan delete berdasarkan npm.

c. Tambahkan method deleteStudent pada interface StudentService

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

Mendeklarasikan nama method yang dapat diakses oleh interface StudentService.

d. Tambahkan method deleteStudent pada StudentServiceDatabase

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

Merupakan implementasi method deleteStudent yang telah dideklarasikan di interface StudentService. Method ini menerima parameter npm. Didalam method terdapat log.info yang digunakan untuk menyimpan data log untuk transaksi yang dilakukan. Method tersebut kemudian memanggil method deleteStudent pada studentMapper.

e. Tambah method delete pada StudentController

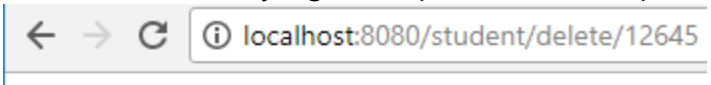
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        studentDAO.deleteStudent (npm);
        model.addAttribute ("student", student);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }

}
```

Method delete menerima parameter berupa npm yang kemudian dengan parameter tersebut dilakukan pencarian data student berdasarkan parameter npm dengan menggunakan method selectStudent. Jika data student ditemukan maka kemudian memanggil method deleteStudent yang ada pada class StudentService dan mengembalika string delete yang berarti menuju ke view delete.html. Jika data student tidak ditemukan maka akan mengembalikan string not-found yang berarti menuju view not-found.html

Berikut adalah hasil yang tertampil ketika menampilkan view delete.html

Data berhasil dihapus

Berikut adalah hasil yang tertampil ketika menampilkan view not-found.html



Student not found

NPM = 12645

4. Penjelasan Fitur Update

a. Tambah method updateStudent pada StudentMapper

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

Pada interface StudentMapper ditambahkan nama method yang digunakan untuk melakukan update data student dengan parameter model studentModel.
Anotasi @Update digunakan untuk mapping query update. Didalam anotasi tersebut terdapat query sql untuk melakukan update dengan berdasarkan npm dan data yang diset adalah name dan gpa.

b. Tambah method updateStudent pada interface StudentService

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

Mendeklarasikan nama method updateStudent yang dapat diakses oleh interface StudentService.

c. Tambah method updateStudent pada StudentServiceDatabase

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

Merupakan implementasi method deleteStudent yang telah dideklarasikan di interface StudentService. Method ini menerima parameter object model StudentModel. Didalam method terdapat log.info yang digunakan untuk menyimpan data log untuk transaksi yang dilakukan. Method tersebut kemudian memanggil method updateStudent pada studentMapper.

d. Tambah link Update pada viewall.html

```
<a th:href="'/student/update/' + ${student.npm}" > Update Data</a><br/>
```

Untuk dapat menjalankan fitur Update digunakan link pada fitur viewall yang langsung men-direct ke method update pada studentController. Parameter yang digunakan adalah npm student.
Berikut adalah tampilan view-nya,

All Students

No. 1

NPM = 12345

Name = chanek

GPA = 3.46

Delete Data
Update Data

---

e. Tambahkan view form-update.html pada templete.
View ini akan menampilkan form yang otomatis terisi dengan data student yang dikirimkan oleh controller. Data yang tertampil pada controller dapat diubah hanya name dan gpa sementara nim hanya read only.

f. Tambahkan view success-update.html pada templete.

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diubah</h2>
    </body>
</html>
```

Tampilan pada view ini akan muncul ketika data berhasil diubah.

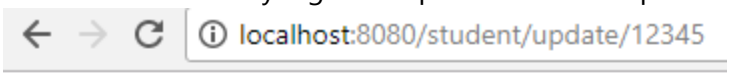
g. Tambah method update pada StudentController

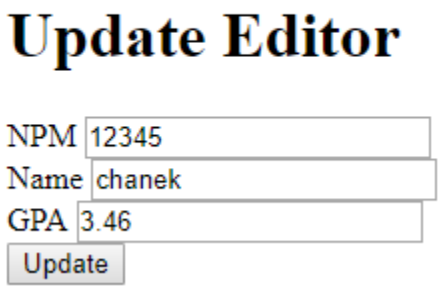
```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        model.addAttribute ("student", student);
        return "form-update";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

Method delete menerima parameter berupa npm yang kemudian dengan parameter tersebut dilakukan pencarian data student berdasarkan parameter npm dengan menggunakan method selectStudent. Jika data student ditemukan maka kemudian mengembalikan string form-update yang berarti menuju ke view form-update.html. Jika

data student tidak ditemukan maka akan mengembalikan string not-found yang berarti menuju view not-found.html

Berikut adalah hasil yang tertampil ketika menampilkan view form-update.html





h. Tambah method updateSubmit pada 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";
}
```

Method update menerima parameter berupa npm, name, dan gpa yang telah di update yang dikirimkan melalui form-update.html. Kemudian parameter tersebut di deklarasikan sebagai object StudentModel dan memanggil method updateStudent yang ada pada studentService dengan parameter object studentModel tersebut. Kemudian mengembalikan string success-update yang berarti menuju view success-update.html

Berikut adalah hasil yang tertampil ketika menampilkan view success -update.html

5. Penjelasan Firut Update dengan parameter model

Sama seperti pada nomor 5 hanya ada beberapa yang perlu ditambahkan yaitu pada form-update.html dan studentController.

a. Tambahkan th:object pada tag form form-update.html

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

Hal ini bertujuan untuk mendeklarasikan bahwa yang akan di post ke controller updateSubmit adalah bertipe object dengan nama student.

b. Tambahkan th:value dan th:field pada tag input form-update.html

```
<input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="*{npm}" />
<input type="text" name="name" th:value="${student.name}" th:field="*{name}"/>
<input type="text" name="gpa" th:value="${student.gpa}" th:field="*{gpa}"/>
```

Masing-masing field dideklarasikan nama field-nya dengan th:field baik pada field npm, nama, dan gpa. Kemudian dengan th:value digunakan untuk mendeklarasikan dimana nilai setiap field disimpan yaitu pada object student yang telah dideklarasikan.

c. Tambahkan method updateSubmit yang akan menerima parameter object

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

    return "success-update";
}
```

Terdapat anotasi @ModalAtrribut untuk menerima parameter model dengan dengan tipe data object StudentModel. Kemudian langsung memanggil method updateStudent dengan parameter student tersebut pada studentService.