Writeup Tutorial 4
Sandar Klinton, APAP-A
NPM 1306404720
Yang saya pelajari dari Tutorial 4 kali ini adalah:

- Bagaimana membuat form di SpringBoot
- Bagaimana mencocokkan isi form menggunakan objek pada springboot
- Ternyata, dapat mengecek tipe request (GET,POST) di dalam controller
- Kita bisa menggunakan ThymeLeaf untuk memasukkan apapun yang berkaitan dengan model yang diantar dari controller.
- Menggunakan slf4j untuk log

Jawaban Untuk Pertanyaan Tutorial

1. Bila menggunakan objek, menurut saya validasi diperlukan dan menyesuaikan kepada tujuan form tersebut. Validasi juga bisa terkait pada model database apabila model database mempunyai syarat field(misal: integer, datetime field, not null, foreign key dll).

   Bila input OPTIONAL, tetap dicek apabila dia ada isinya atau tidak. Bila ada, maka input harus dicek lebih lanjut agar menyesuaikan kepada apa yang telah dijabarkan diatas. Bila tidak ada, tidak usah dicek lagi. Input REQUIRED sama seperti di atas, namun bila tidak ada, harus mengembalikan laman post beserta pesan bahwa field tersebut harus diisi.

   Berikut adalah contoh codenya.
   Required:
   Pada controller:

```
@RequestMapping(value = "student/update/submit", method = RequestMethod.POST)
public String updateSubmit(Model model, @ModelAttribute StudentModel student) {
    StudentModel found = studentDAO.selectStudent(student.getNpm());
    System.out.println(found.toString());
    if (found.equals(null)) {
        return "not-found";
    }else {
        found.setGpa(student.getGpa());
        anggap NAME required. tadi mau coba GPA, tapi error object, belum ngerti how to check object field if empty :D
        NAME, bila harus ada dan bernomor tidak boleh misalnya.
        if(student.getName().equals(null)||student.getName().equals("")||student.getName().matches(".*\\d+.*")) {
            model.addAttribute("message", "Nama tidak boleh kosong/berisi angka!");
            model.addAttribute("student", found);
            return "form-update";
        }else {
            found.setName(student.getName());
        }


        studentDAO.updateStudent(found);
        return "success-update";
    }
```

   Pada template:

```
<h1 class="page-header">Update Student Here.</h1>
<h4 th:text="${message}"></h4>
<form action="/student/update/submit" method="POST" th:object="${student}">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="readonly
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:field="*{nam
    </div>
    <div>
        <label for="gpa">GPA</label> <input type="text" name="gpa" th:field="*{gpa}"/
```

Update Student Here.

NPM 12334
Name omwvo1
GPA 3.333
Save

Update Student Here.

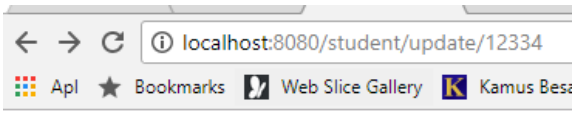
Nama tidak boleh kosong/berisi angka!

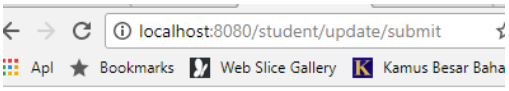
NPM 12334
Name omwvo
GPA 3.333
Save

Bila Optional:

```
@RequestMapping(value = "student/update/submit", method = RequestMethod.POST)
public String updateSubmit(Model model, @ModelAttribute StudentModel student) {
    StudentModel found = studentDAO.selectStudent(student.getNpm());
    System.out.println(found.toString());
    if (found.equals(null)) {
        return "not-found";
    }else {
        found.setGpa(student.getGpa());
        anggap NAME optional. tadi mau coba GPA, tapi error object, belum ngerti how to check object field if empty :D
        NAME, bila tidak harus ada dan bernomor tidak boleh misalnya.
        if(student.getName().matches(".*\\d+.*")) {
            model.addAttribute("message", "Nama tidak boleh berisi angka!");
            model.addAttribute("student", found);
            return "form-update";
        }else {
            found.setName(student.getName());
        }


        studentDAO.updateStudent(found);
        return "success-update";
    }
```

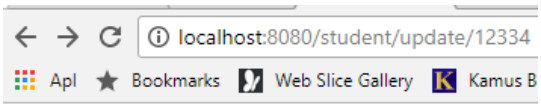




Update Student Here.

Data berhasil diupdate

NPM 12334
Name
GPA 3.333
Save



Update Student Here.

NPM 12334
Name 12
GPA 3.333
Save

Update Student Here.

Nama tidak boleh berisi angka!

NPM 12334
Name
GPA 3.333
Save

Mohon maaf, saya tidak menyertakan code ini di dalam tugas saya karena langsung dikembalikan ke kode semula, dan saya langsung melanjutkan koding saya.

2. Request GET dan Request POST adalah request yang berbeda, dimana request GET mengantar parameter dan value secara terbuka pada URL. Sedangkan pada request POST, data tidak dikirim lewat URL string. Dengan demikian, isi dari form tidak dapat dikirimkan dengan sembarang oleh siapapun lewat URL. Tidak ada yang berbeda pada pengambilan request pada controller, namun agar lebih aman request POST harus didefinisikan pada request mapping method. Request GET tidak bisa mengantarkan kelas dari client ke server, sedangkan POST bisa dengan pengaplikasian object pada form.

3. Bisa. Perhatikan kode di bawah ini.

```
//  GET dan POST dalam satu request mapping dan satu method saja.
//  Request mapping dari kedua method, yaitu update dan updateSubmit dimatikan, agar
//  requestnya hanya menjalankan method ini :D

    @RequestMapping(value = {"student/update/{npm}", "student/update/submit"}, method = {RequestMethod.POST, RequestMethod.GET})
    public String updateAndSubmit(@PathVariable Optional<String> npm,
            Model model, @ModelAttribute StudentModel student,
            HttpServletRequest request) {

        if(request.getMethod().equals("GET")) {
//          ini isi dari Method update :D
            StudentModel found = studentDAO.selectStudent(npm.get());
            if(found.equals(null)) {
                return "not-found";
            }else {
                model.addAttribute("student", found);
                return "form-update";
            }
        }else {
//          ini isi dari method updateSubmit :D
            StudentModel found = studentDAO.selectStudent(student.getNpm());
            System.out.println(found.toString());
            if (found.equals(null)) {
                return "not-found";
            }else {
                found.setGpa(student.getGpa());
                found.setName(student.getName());
                studentDAO.updateStudent(found);
//              sebenarnya lebih baik bila return method untuk membalikkan laman get, tapi ini cukup sama dengan method diatas :D
                return "success-update";
            }
        }
    }
```

Kita harus menyertakan object request berbentuk HttpServletRequest, untuk memeriksa apakah request ini mempunyai request GET atau POST. Bila GET, masukkan isi kode dari method update, dan bila POST, masukkan isi kode dari method UpdateSubmit.

Di form-update, ganti action form menggunakan thymeleaf. Ini contohnya.

```
<body>

    <h1 class="page-header">Update Student Here.</h1>
    <h4 th:text="${message}"></h4>
    <form th:action="'/student/update/'+ ${student.npm}" method="POST" th:object="${student}">
        <div>
            <label for="npm">NPM</label> <input type="text" name="npm" readonly="readonly" th:field
```

hal ini saya lakukan karena saya hanya menggunakan satu value dari request mapping. Bisa saja menambahkan value lain dari request mapping, seperti “/student/update/submit/”, tapi saya tidak lakukan.

Untuk cek metode, saya harap asdos dapat mengetes aplikasi saya. RequestMethod dari method percobaan saya ini akan saya command, agar default dari aplikasi sesuai dengan tutorial. Method ini telah saya jalankan dan berjalan sama seperti fitur update data yang dijabarkan pada tutorial.

Method dari Latihan Menambahkan Delete.

Pada studentMapper:

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

Pada studentService:

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

Pada studentServiceDatabase:

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

Pada studentController, mencari dahulu studentnya, lalu mendelete kalau ditemukan. Bila tidak ketemu, akan mengembalikan laman not-found.

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

Semua syntax untuk mengakses database menggunakan StudentDAO.

Method dari Latihan Menambahkan Update.

Method di StudentController pada awalnya meminta requestparam saja.

Pada StudentController:

```
    @RequestMapping("/student/update/{npm}")
    public String update (Model model, @RequestParam(value = "npm", required = false) String npm){
        StudentModel found = studentDAO.selectStudent(npm);
        if(found.equals(null)) {
            return "not-found";
        }else {
            model.addAttribute("student", found);
            return "form-update";
        }
    }

    @RequestMapping(value = "student/update/submit", method = {RequestMethod.POST, RequestMethod.GET})
    public String updateSubmit(Model model, @RequestParam(value = "npm", required = false) String npm,
            @RequestParam(value = "name", required = false) String name,
            @RequestParam(value = "gpa", required = false) double gpa) {
//       public String updateSubmit(Model model, @ModelAttribute StudentModel student) {
        StudentModel found = studentDAO.selectStudent(npm);

        if (found.equals(null)) {
            return "not-found";
        }else {
            studentDAO.updateStudent(npm, name, gpa);
            return "success-update";
        }

    }
```

Pada studentmapper:

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

Pada studentService:

```
  void updateStudent (String npm, String name, double gpa);
```

Pada StudentServiceDatabase:

```
    @Override
    public void updateStudent(String npm, String name, double gpa) {
        log.info("student " + npm+ " updated");
        studentMapper.updateStudent(npm, name, gpa);

    }
```

Disini karena belum menggunakan objek sebagai value dari parameter, maka parameter yang diantar ada tiga, yaitu npm, name, dan gpa. Pada saat pemanggilan studentDAO.updateStudent(npm,name,gpa), Studentmapper akan menjalankan query di dalam @update yang terhubung dengan lombok. Setelah dieksekusi, controller akan mengembalikan laman sesuai dengan apa yang dimasukkan.

Method dari Latihan menggunakan objek sebagai parameter

Ada perubahan pada keempat kelas tersebut.

Pada controller:

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

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

    StudentModel found = studentDAO.selectStudent(student.getNpm());
    System.out.println(found.toString());
    if (found.equals(null)) {
        return "not-found";
    }else {
        found.setGpa(student.getGpa());
        found.setName(student.getName());
        studentDAO.updateStudent(found);
        return "success-update";
    }

}
```

Pada studentMapper:

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

Pada studentService:

```
void deleteStudent (String npm);

void updateStudent (StudentModel student);
```

Pada StudentServiceDatabase:

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

}
```

Disini, karena form telah menandakan bahwa ia akan mengantarkan objek, maka saat disubmit, isi dari form akan dibuat menjadi objek student. Ada empat tempat yang harus diubah, yaitu pada gambar yang diatas. Metode tersebut sama dengan yang diatas, hanya beda syntax untuk akses data baru.