TUTORIAL 4

ARSITEKTUR DAN PEMROGRAMAN APLIKASI PERUSAHAAN

6 MARET 2018

MENGGUNNAKAN DATABASE DAN MELAKUKAN DEBUGGIN DALAM PROJECT SPRING BOOT

Muhammad Fajar Pamungkas – 1706106835

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.

   Jawaban :

   Object student akan tetap dibuat walaupun pengguna tidak memasukkan input yang diinginkan jika menggunakan object sebagai parameter sehingga parameter akan otomatis bersifat optional. Untuk memvalidasi input yang dimasukkan dapat menambahkan parameter BindingResult pada method updateSubmit seperti berikut atau memvalidasi inputan per attribute object dengan menggunakan percabangan kondisi if then else.

```
125     @PostMapping("/student/update/submit")
126     public String updateSubmit(@Valid @ModelAttribute("student") StudentModel student,
127         BindingResult result) {
128
129         if(result.hasErrors())
130             return "not-valid-input";
131         else
132             studentDAO.updateStudent(student);
133
134         return "success-update";
135     }
136 }
```

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?

   Dengan method POST, URL akan lebih pendek karena data yang diinputkan tidak akan ditampilkan pada URL dan panjang input tidak memiliki batas sedangkan jika menggunakan GET panjang input maksimal yaitu 2047 karakter. POST lebih sesuai untuk operasi submit karena data sensitive tidak akan terlihat pada URL. Berikut adalah baris kode jika menggunakan menggunakan method GET.

```
126     @GetMapping("/student/update/submit/")
127     public String updateSubmit(@Valid @ModelAttribute("student") StudentModel student,
128         BindingResult result) {
129
130         if(result.hasErrors())
131             return "not-valid-input";
132         else
133             studentDAO.updateStudent(student);
134
135         return "success-update";
136     }
137 }
```

3. Tidak bisa, karena browser hanya dapat mengirimkan satu method request yaitu GET atau POST atau PUT sehingga penerimaan method GET atau POST secara sekaligus pada method tidak beralasan. Walaupun RequestMapping dapat menghandle secara jenis request dengan fleksibel tapi tetap saja method hanya akan menghandle 1 jenis request dalam satu waktu.

Delete Student

Berikut adalah method deleteStudent yang ditambahkan pada interface StudentMapper dan diberikan annotation header Delete dengan perintah SQL untuk menghapus baris data student dengan npm tertentu.

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

Lalu menambahkan method deleteStudent pada class StudentServiceDatabase dimana method tersebut berisi baris kode yang berfungsi menghapus baris data student dengan npm tertentu dengan menerima parameter berupa String npm memanfaatkan method deleteStudent yang ada pada StudentMapper.

```
44      @Override
45      public void deleteStudent (String npm)
46      {
47          studentMapper.deleteStudent(npm);
48          log.info("student "+ npm + " deleted");
49      }
```

Pada controller ditambahkan method delete dimana pada method tersebut terdapat baris kode untuk memvalidasi keberadaan data yang ingin dihapus apabila data tersebut ada maka data tersebut akan terhapus jika data yang dimaksud tidak ada maka tampilan akan diarahkan pada halaman not-found.

```
97      @RequestMapping("/student/delete/{npm}")
98      public String delete (Model model, @PathVariable(value = "npm") String npm)
99      {
100         StudentModel student = studentDAO.selectStudent(npm);
101
102         if (student != null)
103             studentDAO.deleteStudent (npm);
104         else {
105             model.addAttribute("npm", npm);
106             return "not-found";
107         }
108
109         return "delete";
110     }
```

Update Student

Berikut adalah method updateStudent yang ditambahkan pada interface StudentMapper dengan annotation update berisi perintah SQL untuk mengupdate baris data student berdasarkan npm yang diinginkan.

```
29      @Update("UPDATE student set name = #{name}, gpa = #{gpa} where npm = #{npm}")
30      void updateStudent(@Param("npm") String npm, @Param("name") String name, @Param("gpa") double gpa);
31
```

Lalu menambahkan method updateStudent pada interface studentService.

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

Lalu mengimplementasikan method updateStudent pada class StudentServiceDatabase dengan memberikan baris kode yang berfungsi untuk mengupdate baris data student pada databse dengan kriteria npm tertentu memanfaatkan method updateStudent dari pada StudentMapper.

```
51    public void updateStudent (String npm, String name, double gpa)
52    {
53        studentMapper.updateStudent(npm, name,gpa);
54        log.info("student "+ npm + " update");
55    }
```

Pada controller ditambahkan method update dengan menerima parameter String NPM. Method ini berfungsi untuk menampilkan halaman update yang berisi form update yang berisi dengan data student yang dipilih untuk diupdate. Pada method ini terdapat validasi untuk memeriksa keberadaan data yang ingin diupdate, apabila data yang diinginkan terdapat pada database maka halaman yang ditampilkan adalah halaman form-update apabila sebaliknya maka halaman yang ditampilkan adalah halaman not-found.

```
112   @RequestMapping("/student/update/{npm}")
113   public String update (Model model, @PathVariable(value = "npm") String npm)
114   {
115       StudentModel student = studentDAO.selectStudent(npm);
116
117           if (student == null){
118               model.addAttribute("npm", npm);
119               return "not-found";
120           }
121
122           model.addAttribute ("student", student);
123           return "form-update";
124   }
```

Method updateSubmit berfungsi untuk menerima segala perubahan lalu melakukan segala inputan perubahan kedalam database dengan memanfaatkan method updateStudent yang ada pada class StudentService.

```
127   @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
128   public String updateSubmit(
129       @RequestParam(value = "npm", required = false) String npm,
130       @RequestParam(value = "name", required = false) String name,
131       @RequestParam(value = "gpa", required = false) double gpa)
132   {
133       studentDAO.updateStudent(npm,name,gpa);
134       return "success-update";
135   }
```

Update Student Menggunakan Object Sebagai Parameter

Mengubah method updateStudent yang ada pada StudentMapper untuk menerima parameter berupa object StudentModel seperti pada gambar dibawah

```
29    @Update("UPDATE student set name = #{student.name}, gpa = #{student.gpa} where npm = #{student.npm}")
30    void updateStudent (@Param("student") StudentModel student);
31 }
32
```

Begitu juga pada interface StudentService, method updateStudent diubah untuk menerima parameter object StudentModel seperti pada gambar dibawah.

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

Sehingga method updateStudent pada class StudentServiceDatabase juga dilakukan perubahan untuk menerima parameter object.

```
57    public void updateStudent (StudentModel student)
58    {
59        studentMapper.updateStudent(student);
60        log.info("student "+ student.getNpm() + " update");
61    }
```

Pada controller penerimaan parameter juga diubah menjadi StudentModel lalu ditambahkan anotasi ModelAttribute, Valid, beserta parameter berupa BindingResult untuk validasi input atapun untuk menghandle error ketika melakukan input data.

```
137    @GetMapping("/student/update/submit/")
138    public String updateSubmit(@Valid @ModelAttribute("student") StudentModel student,
139        BindingResult result) {
140
141        if(result.hasErrors())
142            return "not-valid-input";
143        else
144            studentDAO.updateStudent(student);
145
146        return "success-update";
147    }
```

Lalu pada view form-update ditambakan th-object=”${student}” pada tag form untuk menyatakan model objek yang digunakkan untuk menbungkus data-data inputan lalu th:field”*{namaAttribute} pada setiap field input sesuai dengan attribute-attribute yang ada pada object Student yang digunakan dengan field input.

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

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