Rahmat Chan
EKstensi
1606954962

Pelajaran yang diambil pada tutorial 4

Pada tutorial kali ini terdapat pembelajaran baru seperti penggunaan library Lombok, MyBatis, dan MySQL. Library- library ini membantu SpringBoot berhubungan dengan database MySQL. Library Lombok berfungsi sebagai helper annotation, library MySQL berfungsi untuk menghubungkan project yang dibuat dengan database MySQL, sedangkan library MyBatis berfungsi untuk memembantu melakukan koneksi dan generate query dengan helper annotation.

Pada latihan ini juga merupakan lanjutan dari tutorial 3 dimana pada tutorial 3 menyimpan data pada list, sedangkan pada tutorial ini menyimpan data didatabase langsung. Untuk dapat koneksi ke database dapat mengedit file “application.properties”.

selain itu untuk debugging semua hal-hal yang dikerjakan dapat disimpan log-nya dengan menambahkan library s1f4j.

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?

   Cara untuk memvalidasi input pada form POST adalah dengan menambahkan @Valid dan object BindingResult. Pada method form-update juga dibuat kondisi jika kondisi null terjadi maka akan memanggil halaman not-found.html, jika kondisi null tidak terjadi maka akan data student akan diupdate. Selain itu, juga diperlukan notasi @NotNull pada variable name dan gpa. Validasi diperlukan untuk menghindari apabila users tidak menginputkan name dan gpa pada saat melakukan update data mahasiswa di browser.

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?

   Karena method POST lebih aman dibandingkan dengan method GET. Pada saat user menginputkan data dan menekan tombol update, maka di url tidak menampilkan data apa saja yang baru diupdate. Berbeda dengan menthod GET, pada method GET data yang baru diubah akan ditampilkan pada URL. Perlu, caranya dengan menambahkan method = RequestMethod.POST pada anotasi @RequestMapping

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

   Bisa, tetapi harus satu persatu menjalankannya, tidak bisa sekaligus.

Rahmat Chan
EKstensi
1606954962

Penjelasan Method Penambahan Delete

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

Berfungsi untuk menghapus data student di database dengan inputan berupa NPM pada url atau bisa juga dengan langsung mengklik link Delete Data di browser.

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

Method diatas ada pada class studentServiceDatabase, pada method ini ditambahkan log ketika delete student serta memanggil methodStudent yang ada pada class studentMapper.

```
@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 {

            return "not-found";
        }

}
```

Method diatas merupakan controller untuk menghapus data pada database student. Pada method diatas dilakukan validasi untuk mengecek apakah NPM yang diinputkan di url ada didatabase atau tidak, Jika npm ditemkuk maka akan dipanggil method deleteStudent pada class StudentService da mengembalikan halaman delete.html. Jika data NPM tidak ditemukan maka akan mengembalikan halaman not-found.html.

Penjelasan Method Penambahan Update

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

Rahmat Chan
EKstensi
1606954962

Merupakan query SQL yang berfungsi untuk meng-update data name dang pa pada database student dengan melihat berdasarkan npm.

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

Method diatas ada pada class studentServiceDatabase, pada method ini ditambahkan log ketika update database student serta memanggil methodStudent yang ada pada class studentMapper.

```
@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";
    }

}
```

Sama seperti method delete, pada mehod update juga diperlukan validasi untuk mengecek apakah NPM yang diinputkan ada atau tidak didatabase. Jika ada maka akan mengembalikan halaman form-update.html, jika tidak ada maka akan mengembalikan halaman not-found.html.

```
@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 diatas berfungsi untuk menerima data yang ada pada StudentModel yang akan diinputkan pada halaman form-update. Data tersebut nantinya akan dimasukan kedalam objek student yang baru dibuat. Selanjutnya, akan dipanggil method updateStudent yang berisi objek student pada class StudentService. Setelah itu akan mengembalikan halaman success-update.html yang menandakan proses update data mahasiswa telah sukses dilakukan.

Rahmat Chan
EKstensi
1606954962

Penjelasan Penggunaa Object Sebagai Parameter

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

{
        studentDAO.updateStudent (student);
        return "success-update";

    }

}
```

Pada method sama seperi method updateSubmit yang menggunakan @RequestParam, bedanya pada inputan yang diterima oleh method ini dalam bentuk objek student yang dikirim dari halaman form-update.htlm. Hal berikutnya yang dilakukan menambahkan ekspresi th:object=”${student}” dan th:field=”*{nama_fileld}” pada halaman form-update.html, lebih jelasnya seperti pada gambar dibawah ini.

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

Ekspresi th:object=”${student}” pada form diatas berguna untuk mendeklarasikan sebuah objek yang akan mengumpulkan data mahasiswa yang akan diupdate. Sedangkan ekspresi th:field=”*{nama_field}” digunakan untuk mengekspresikan field nama, npm, dan gpa yang

sesuai dengan objek Student.