Muhammad Farhan Mardadi
1506757743

Tutorial 4

Pada tutorial kali ini saya mempelajari bagaimana menggunakan database pada aplikasi spring boot, dengan menggunakan database saya lebih mudah melihat data yang telah dimasukkan maupun dihapus, saya juga belajar untuk melakukan query untuk mendapatkan data dengan attribute yang ada pada objek.

Saya juga belajar debugging pada tutorial kali ini, yaitu menggunakan S1F4j, method-method yang digunakan seperti log.info, log.debug dan log.error. Memfilter log sesuai dengan kategori akan memudahkan kita untuk melakukan debugging.

Method Delete

Untuk membuat fitur delete, pertama-tama saya membuat method pada interface StudentMapper dengan menggunakan query.

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

Lalu di class StudentServiceDatabase membuat method delete yang menanggil method delete di StudentMapper.

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

Lalu membuat method delete pada class StudentController, yaitu dengan mencari dahulu data mahasiswa yang ingin dihapus, apabila ditemukan maka data mahasiswa itu akan dihapus namun apabila data mahasiswa itu tidak ditemukan maka akan ditampilkan view not found.

Method Update

Untuk membuat method update ini pertama-tama membuat method pada interface StudentMapper dengan memberikan parameter StudentModel dan memberikan query untuk update.

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

Dan juga saya membuat method update pada interface StudentService yang juga menerima paratemer StudentModel..

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

Lalu membuat method updateStudent pada class StudentServiceDatabase yang tujuannya untuk memanggil method update pada class StudentMapper dan juga untuk debugging.

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

Lalu membuat form-update.html agar user dapat update menggunakan view, hanya sedikit mengedit dari form add yang ada sebelumnya. Dan juga membuat success-update.html yang menandakan update berhasil.

Kemudian membuat method update pada class StudentController yang sangat mirip dengan method delete sebelumya namun hanya kali ini mengembalikan form-update.

```
@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";
    }
}
```

Kemudian membuat method updateSubmit yang berguna untuk mengupdate data dari view yang disediakan.

Muhammad Farhan Mardadi
1506757743

Menggunakan Object sebagai Parameter

Menurut saya cara ini merupakan cara yang efesien untuk menghandle request param, apalagi jika terdapat banyak request param.

Pertama menambahkan object pada form tersebut di form-update.

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

Lalu menambahkan field pada setiap input yang dibutuhkan oleh form.

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

Dan yang terakhir dengan mengubah semua param yang ada pada method updateSubmit dengan object student.

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

Muhammad Farhan Mardadi
1506757743

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan?
   Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.
   Diperlukan atau tidak merupakan pilihan user. Untuk tutorial ini sepertinya dibutuhkan. Untuk melakukan validasi kita dapat menambahkan anotasi @NotNull pada atribut pada kelas model untuk menandakan atribut tersebut tidak boleh kosong. Lalu untuk pengecekannya dapat dilakukan di method updateSubmit dengan menambahkan anotasi @Valid pada model yang ingin divalidasi.

```
public class StudentModel
{
    @NotNull
    private String npm;

    @NotNull
    private String name;

    @NotNull
    private double gpa;

}
```

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@Valid @ModelAttribute StudentModel student)
{
    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?
   Karena apabila menggunakan method GET maka data yang dikirimkan akan terlihat di URL, maka sangat tidak aman untuk submit data. Dengan menggunakan method POST yang tidak memperlihatkan data yang dikirimkan lewat URL akan lebih aman.
   Sebaiknya diberikan penanganan yang berbeda, contohnya dengan memberikan method = RequestMethod.POST untuk menandakan bahwa method ini menggunakan method POST, seperti gambar pada No. 1.

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