WRITE UP TUTORIAL 4

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

(**Raditya Nurfadillah – 1506728831**)

Latihan Menambahkan Delete

- Method deleteStudent di class StudentMapper

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

Penjelasan:
Method ini berfungsi untuk melakukan perintah sql Delete ke Database dengan menyamakan npm dari parameter method tersebut dengan npm yang ada di database.

- Method deleteStudent di class StudentServiceDatabase

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

Penjelasan:
Method ini berfungsi sebagai trigger untuk method deleteStudent yang ada di class StudentMapper dan juga berfungsi sebagai debugger karena menggunakan log yang ada di lombok.

- Method delete di class 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);
       return "delete";
    } else {
       model.addAttribute ("npm", npm);
       return "not-found";
    }
}
```

Penjelasan:
Method ini berfungsi sebagai penerima jika ada request mapping “/student/delete/{npm}”. Jika student dengan npm tidak ada, maka akan mengembalikan view not-found. Sedangkan jika student ditemukan, method ini akan memanggil method deleteStudent yang ada di class StudentServiceDatabase dan menampilkan view delete.

Latihan Menambahkan Update

- Method updateStudent pada class StudentMapper

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

  Penjelasan:
  Method ini berfungsi untuk melakukan perintah sql Update ke Database dengan menyamakan npm dari student di parameter method tersebut dengan npm yang ada di database.

- Method updateStudent pada interface StudentService

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

  Penjelasan:
  Method ini berada di dalam interface yang bertujuan untuk mengurangi coupling dari method tersebut.

- Method updateStudent pada class StudentServiceDatabase

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

  Penjelasan:
  Method ini berfungsi sebagai trigger untuk method updateStudent yang ada di class StudentMapper dan juga berfungsi sebagai debugger karena menggunakan log yang ada di lombok.

- Method update pada class 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";
	}
}
```

  Penjelasan:
  Method ini berfungsi sebagai penerima jika ada request mapping “/student/update/{npm}”. Jika student dengan npm tidak ada, maka akan mengembalikan view not-found. Sedangkan jika student ditemukan, method ini akan menambahkan attribute student dengan npm yang sesuai dengan parameter method tersebut kedalam model dan mengembalikan view form-update.

- Method updateSubmit pada class 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";
}
```

  Penjelasan:
  Method ini berfungsi untuk mengupdate student dengan menggunakan method POST dan mengembalikan view success-update.

Latihan Menggunakan Object sebagai Parameter

- Method updateSubmit pada class StudentController

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

  Penjelasan:
  Method ini mempunyai fungsi yang sama seperti updateSubmit yang sebelumnya hanya berbeda di parameter method ini yang berupa objek tidak seperti method sebelumnya yang berupa request param. Hal ini dilakukan agar pada saat parameter yang dibutuhkan method tersebut sudah terlalu banyak, tidak menimbulkan banyaknya line di code tersebut.

Pertanyaan

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.
   Validasi seperti itu menurut saya tidak perlu dilakukan di sector backend. Tetapi validasi tersebut masih dapat dilakukan dengan menambahkan @PostMapping di Controller. Berikut adalah link untuk mengetahui lebih jelas cara dari validasi tersebut: https://spring.io/guides/gs/validating-form-input/

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?
   Method POST lebih sering digunakan pada form submit agar keamanan data dapat lebih terjamin sebab jika menggunakan GET, data tersebut dapat terlihat secara eksplisit. Dan untuk penanganan di header atau body method di controller tentu akan berbeda untuk GET dan POST.

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 jenis request method yaitu antara GET atau POST dan bukan dua-duanya.

Lesson Learned

Pelajaran yang telah saya dapatkan di tutorial kali ini antara lain menggunakan debugger untuk mengetahui dimana error yang sedang terjadi. Kedua saya juga dapat mengetahui bagaimana cara untuk mengkoneksikan database dan class Controller yang ada. Dan yang terakhir saya juga dapat mengetahui bagaimana cara menggunakan objek sebagai parameter yaitu dengan mengubah beberapa hal yang diperlukan di dalam view yang bersangkutan.