TUTORIAL 4 WRITE UP

Pada tutorial ini saya mempelajari bagaimana cara mengolah program yang menggunakan database pada SpringBoot framework. Operasi yang dilakukan antara lain adalah delete dan update. Saya juga sedikit belajar mengenai penggunaan objek sebagai parameter yang seharusnya mempermudah pemrograman (misalnya tidak perlu me-request parameter banyak-banyak).

Pertanyaan

1. Asumsi saya dapat menggunakan method khusus yang disediakan oleh SpringBoot sendiri atau membuat method sendiri. Ya, validasi tetap perlu.
2. Karena method GET memiliki banyak kekurangan seperti panjang URL yang terbatas dan input yang akan terlihat pada URL sehingga kurang aman. Sepertinya penggunaan keduanya tidak jauh berbeda (masih menggunakan model.
3. Tidak.

Method Delete

```
@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";
    }
}
```

Pada StudentController.java, diambil student yang memiliki npm sesuai dengan input. Apabila student tersebut ada maka jalankan method deleteStudent dan lanjut ke delete.html, jika tidak tampilkan not-found.html.

Method 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 ("student", student);
        return "not-found";
    }
}

@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";
}
```

Pertama dicek dulu apakah student yang ingin di-update ada. Jika ada, lanjut ke halaman form-update.html. Halaman menampilkan data student yang ingin diubah. Selanjutnya dibuat objek student yang baru dengan data baru (kecuali npm) yang “menimpa” objek student yang lama. Setelah selesai, tampilkan halaman success-update.html.

Menggunakan Object sebagai Parameter

```
@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 ("student", student);
        return "not-found";
    }
}

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