TUTORIAL 4 ADPAP

Pertanyaan

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan?

   Dengan cara menambahkan kondisi diawal *method* yang dipanggil dari *form* POST sebelum mengeksekusi perintah yang lainnya untuk memeriksa kelengkapan atribut yang *required* pada Object parameter. Misalnya seperti input nama yang *required:*

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

   if(student.getName().equals("")) {
         return "error";
   } else {
         studentDAO.updateStudent (student);
         return "success-update";
   }
}
```

   Menurut saya, validasi diperlukan untuk kondisi-kondisi yang tidak biasa seperti untuk memastikan apakah suatu atribut ada di dalam database atau tidak. Namun, jika validasi yang dibutuhkan hanya untuk memeriksa apakah input pada *form* sudah terisi atau belum, cukup dengan validasi di *front-end.*

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* GET dapat digunakan untuk meng-*alter* data pada *database* hanya dengan memasukkan URL tertentu. Sehingga, untuk melakukan fungsi yang dapat merubah *resource* lebih baik menggunakan *method* POST, dan untuk fungsi yang hanya meminta informasi tertentu dari *resource* dapat menggunakan *method* GET.

   Ya, perlu pengangan berbeda pada *header method controller,* yakni dengan menghilangkan "method = RequestMethod.POST" dari parameter @RequestMapping. Namun, untuk *body method* tidak diperlukan penanganan berbeda.

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

   Tidak, Karena akan terkena *exception.*

Lesson Learned

Hal-hal yang saya pelajari dari tutorial kali ini adalah perbedaan pada *method* POST dan GET, cara menggunakan *object* sebagai parameter, cara menggunakan *library* Lombok, serta bagaimana cara mengimplementasikan fungsi *update* dan *delete* dengan menggunakan *database*.

Delete Method

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
   StudentModel student = studentDAO.selectStudent(npm);

          if(student == null) {
                return "not-found";
          } else {
                studentDAO.deleteStudent (npm);
                return "delete";
          }
}
```

Method ini menerima parameter npm berupa String yang akan digunakan untuk mencari Student dengan npm tersebut pada *database.* Jika Student ditemukan, maka Student tersebut akan dihapus, dan halaman akan dialihkan ke halaman delete.html yang memberitahukan bahwa data berhasil dihapus. Namun, jika Student tidak ditemukan, maka halaman akan dialihkan ke halaman not-found.html yang memberikan informasi bahwa data tidak ditemukan serta keterangan npm yang telah dimasukkan.

Update Method

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
   StudentModel student = studentDAO.selectStudent(npm);

          if(student == null) {
                return "not-found";
          } else {
                model.addAttribute("student", student);
                return "form-update";
          }
}

@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 ini (atas) menerima input npm berupa String yang akan digunakan untuk mencari Student dengan npm tersebut pada *database.* Jika Student tidak ditemukan, maka halaman akan dialihkan ke halaman not-found.html yang memberikan informasi bahwa data tidak ditemukan serta keterangan npm yang telah dimasukkan. Namun, jika Student ditemukan, halaman akan dialihkan ke halaman form-update.html yang berisi *form* untuk mengubah atribut-atribut pada Student tersebut. Setelah *form* di-*submit,* maka akan memanggil *method* updateSubmit() (bawah) yang menerima input npm, name, gpa dari isian pada *form* sebelumnya yang digunakan untuk menginstansiasi *object* Student baru sebagai masukan *parameter* untuk pemanggilan *method update* di *class* StudentService. Setelah dilakukan *update,* maka halaman akan dialihkan ke success-update.html yang memberitahukan bahwa Student berhasil di-*update.*

Method yang Menggunakan Object Sebagai Parameter

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

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

Method ini sama dengan (2) *method update* diatas. Namun pada *method* updateSubmit() ini, hanya menerima input parameter student berupa *object* (StudentModel).