Enrico Jano Fiandi – 1506689490
ADPAP – C

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? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Ya validasi tetap diperlukan, berikut adalah cara untuk melakukan validasi input untuk object parameter :
Langkah pertama saya menambahkan anotasi *@NotNull* pada properti gpa dan menambahkan anotasi *@NotEmpty* pada properti npm dan name. Anotasi *@NotNull* berfungsi untuk memastikan value dari gpa tidak boleh null, dan anotasi *@NotEmpty* berfungsi untuk memastikan value dari npm dan name tidak boleh null dan tidak boleh kosong, karena jika textbox pada field npm dan name kosong akan mengembalikan string kosong yang dimana juga menggunakan *@NotNull* akan tetap disimpan, namun juga menggunakan *@NotEmpty* akan dilarang.

```
1  package com.example.model;
2
3  import javax.validation.constraints.NotNull;
4
5  import org.hibernate.validator.constraints.NotEmpty;
6
7  import lombok.AllArgsConstructor;
8  import lombok.Data;
9  import lombok.NoArgsConstructor;
10
11 @Data
12 @AllArgsConstructor
13 @NoArgsConstructor
14 public class StudentModel
15 {
16     @NotEmpty
17     private String npm;
18     @NotEmpty
19     private String name;
20     @NotNull
21     private double gpa;
22
23 }
24
```

Langkah kedua adalah menambahkan Anotasi *@Valid* pada parameter StudentModel student untuk menjalankan proses validasi dan menambahkan parameter BindingResult bindingResult untuk mengecek apakah terdapat error dalam proses validasi. Jika error maka akan menampilkan not-found.html

```java
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(
        @Valid StudentModel student,
        BindingResult bindingResult,
        Model model)
{

    if(bindingResult.hasErrors()){
        return "not-found";
    }

    StudentModel studentTemp = studentDAO.selectStudent(student.getNpm());
    if(studentTemp == null){
        model.addAttribute("npm", student.getNpm());
        return "not-found";
    }

    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?

Form submit akan melakukan penambahan resource baru atau melakukan perubahan resource pada server-side. Jika menggunakan method GET, parameter-parameternya akan tersimpan didalam *browser history* dan hal tersebut dapat dimanfaatkan oleh orang-orang yang tidak bertanggung jawab untuk menyerang aplikasi kita, jika menggunakan method POST, paramater-parameter tidak akan tersimpan dan menjadi lebih aman dari serangan orang-orang yang tidak bertanggung jawab. Method GET juga dapat di bookmark, jadi orang-orang yang tidak bertanggung jawab dapat melakukan *spamming* terhadap url tersebut yang dapat menyebabkan *website down* akibat terlalu banyak *request* ke server, sedangkan POST tidak dapat dibookmark sehingga aman dari *spamming*.

Ya perlu, kita perlu mengubah method pada *@RequestMapping* menjajadi RequestMethod.GET , atau menghapus parameter method dari *@RequestMapping* karena defaultnya sudah GET.

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

Ya mungkin, ada beberapa cara untuk melakukannya, pertama bisa dengan menambahkan
method = { RequestMethod.GET, RequestMethod.POST }
pada parameter *@RequestMapping*
atau dengan menggunakan *Class HttpServletRequest* dan melakukan pengecekan *request type* pada method *body controller*.

Penjelasan menambahkan method Delete

Method delete pada controller

```
@RequestMapping(value={"/student/delete","/student/delete/{npm}"})
public String delete (Model model, @PathVariable(value = "npm") Optional<String> npm)
{
    if(!npm.isPresent()){
        model.addAttribute("npm", null);
        return "not-found";
    }
    StudentModel student = studentDAO.selectStudent(npm.get());

    if(student != null){
         studentDAO.deleteStudent (npm.get());
         return "delete";
    }

    model.addAttribute("npm", npm.get());
    return "not-found";
}
```

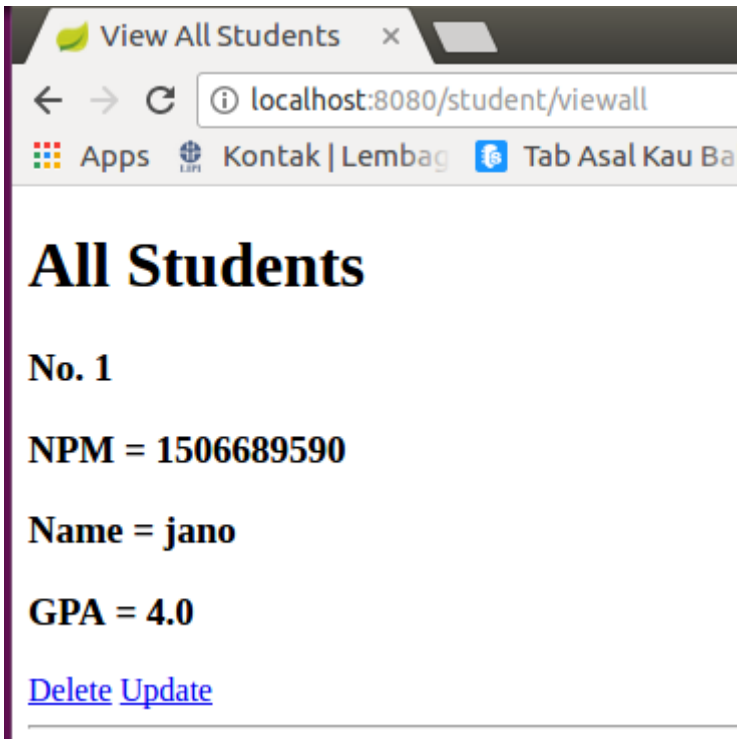
Query method delete

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

Alur method delete adalah ketika url /student/delete atau /student/delete/{npm} dipanggil, maka akan dilakukan pengecekan jika npm tidak terdapat pada url maka akan menampilkan halaman not-found.html, jika ada maka akan dicari objek student yang memiliki npm tersebut, jika ada maka objek student tersebut akan dihapus dengan mengeksekusi Query method delete dan akan menampilkan halaman delete, jika tidak ada maka akan menampilkan halaman not found.

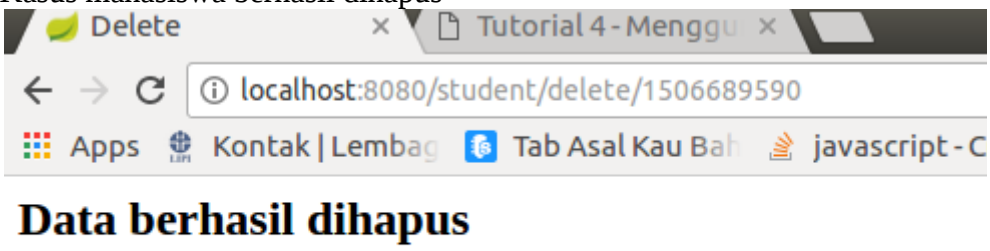
KASUS DELETE

Kondisi awal :

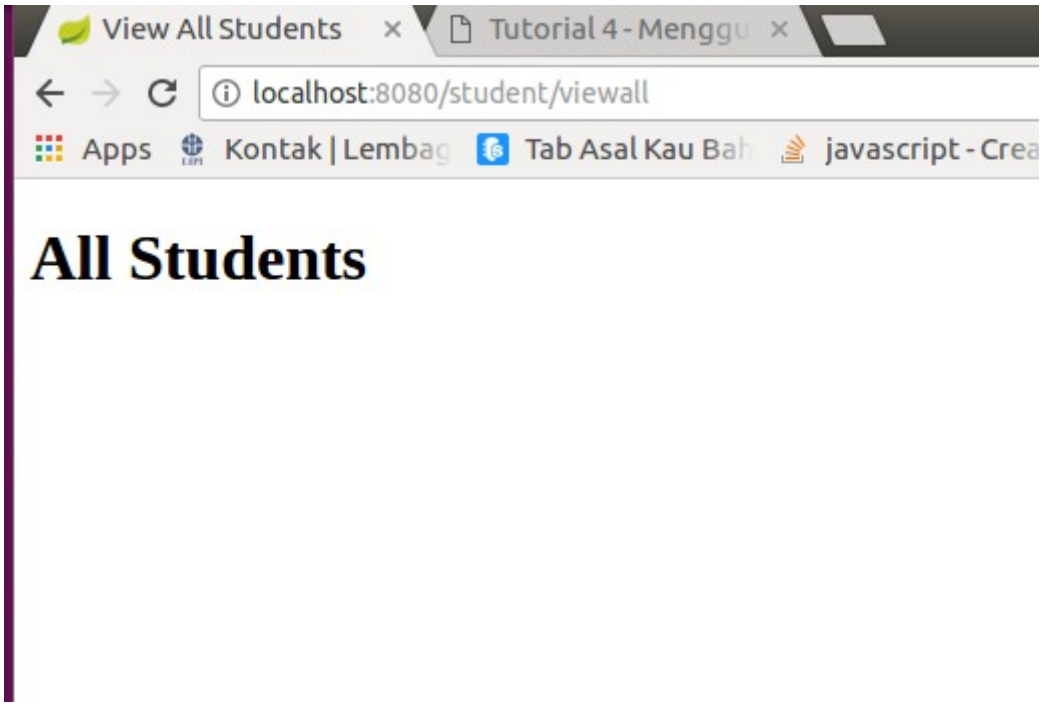


Kasus mahasiswa tidak ditemukan

Kasus mahasiswa berhasil dihapus



Kondisi akhir



Penjelasan menambahkan method Update

Method Update pada controller

- Method untuk menampilkan form update

```java
@RequestMapping(value={"/student/update","/student/update/{npm}"})
public String update(@PathVariable Optional<String> npm, Model model)
{
    if(!npm.isPresent()){
        model.addAttribute("npm", null);
        return "not-found";
    }

    StudentModel student = studentDAO.selectStudent(npm.get());
    if(student != null){
        model.addAttribute("student", student);
        return "form-update";
    }
    model.addAttribute("npm", npm.get());
    return "not-found";

}
```

- Method untuk meng-*submit* 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,
        Model model)
{
    if(npm != null){
        StudentModel student = studentDAO.selectStudent(npm);
        if(student == null){
            model.addAttribute("npm", npm);
            return "not-found";
        }

        student = new StudentModel(npm, name, gpa);
        studentDAO.updateStudent(student);
        return "success-update";
    }

    model.addAttribute("npm", "not-supplied");
    return "not-found";

}
```

Query method update

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

Pada method menampilkan form update akan mengecek apakah parameter npm terdapat pada url atau tidak saat mengakses /student/update atau /student/update/{npm} , jika tidak ada maka akan menampilan halaman not found, jika ada maka akan mengecek apakah terdapat data student di database yang memiliki npm tersebut, jika tidak akan mengembalikan not found, jika ada maka akan menampilkan form-update.html dan data-data student akan dikirim melalui model.

Pada method submit form update akan menerima 3 buah parameter yaitu npm, name, dan gpa, pertama-tama akan dicek terlebih dahulu apakan npm tersebut terdapat dalam parameter atau tidak jika tidak maka akan menampilkan halaman not found jika ada maka akan dicari data student yang memiliki npm tersebut jika ada maka data student tersebut akan diupdate dengan menjalankan query method update dan akan menampilkan halaman jika success update, jika tidak ada maka akan menampilkan halaman not found

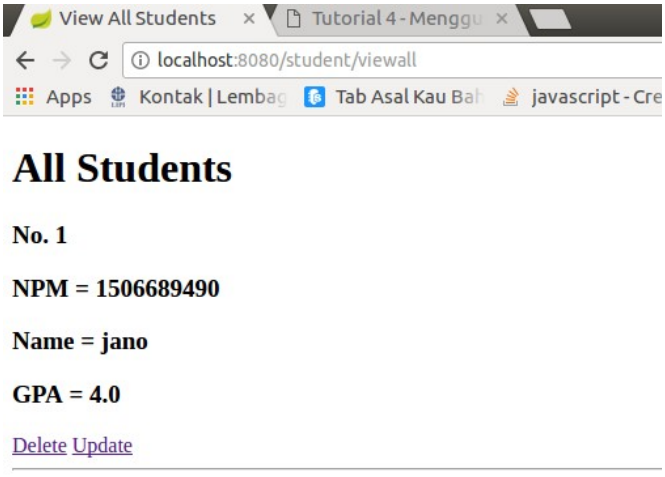
Penjelasan menggunakan objek sebagai parameter

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

    if(bindingResult.hasErrors()){
        return "not-found";
    }

    StudentModel studentTemp = studentDAO.selectStudent(student.getNpm());
    if(studentTemp == null){
        model.addAttribute("npm", student.getNpm());
        return "not-found";
    }

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


}
```

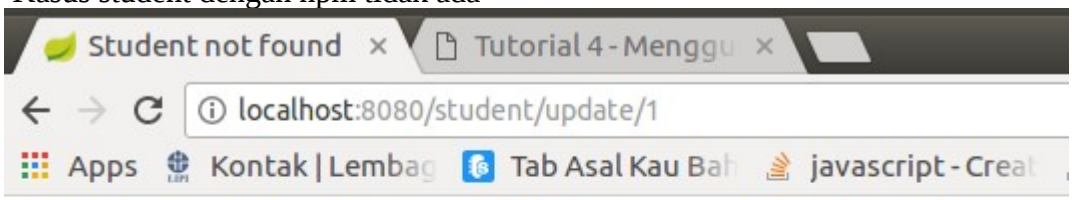
Pada method submit form update, akan menerima parameter berupa objek StudentModel dengan anotasi *@Valid* yang fungsinya telah dijelaskan pada pertanyaan nomor 1, akan dilakukan validasi input terlebih dahulu, jika terdapat error maka akan menampilkan halaman not found jika tidak maka akan dicari apakah terdapat data student yang memiliki npm yang terdapat didalam objek parameter, jika ada maka data tersebut akan diupdate dengan menjalankan query method update dan akan menampilkan halaman success update jika tidak maka akan menampilkan halama not found.

KASUS UPDATE

Kondisi awal



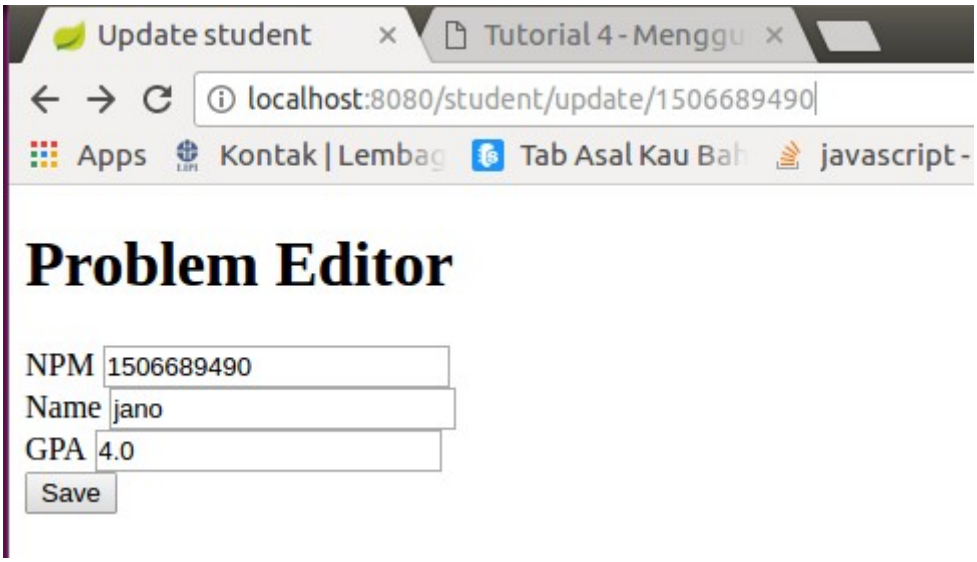
Kasus student dengan npm tidak ada



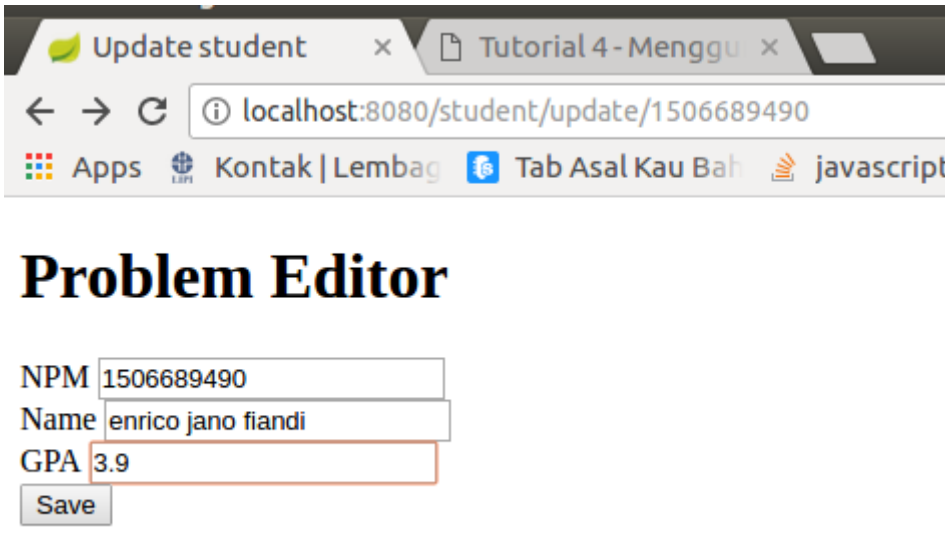
Student not found

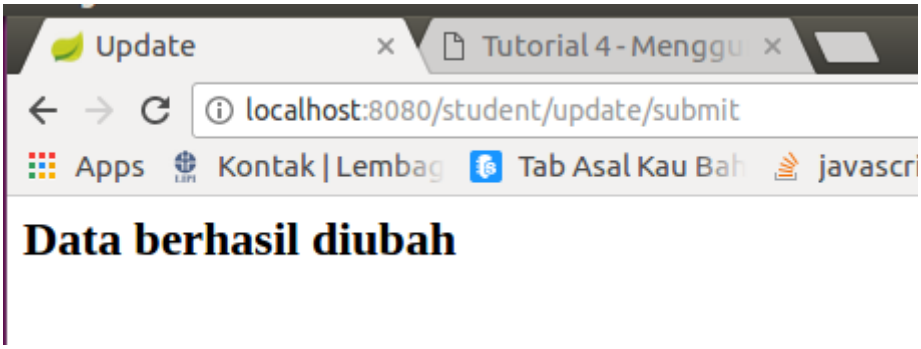
NPM = 1

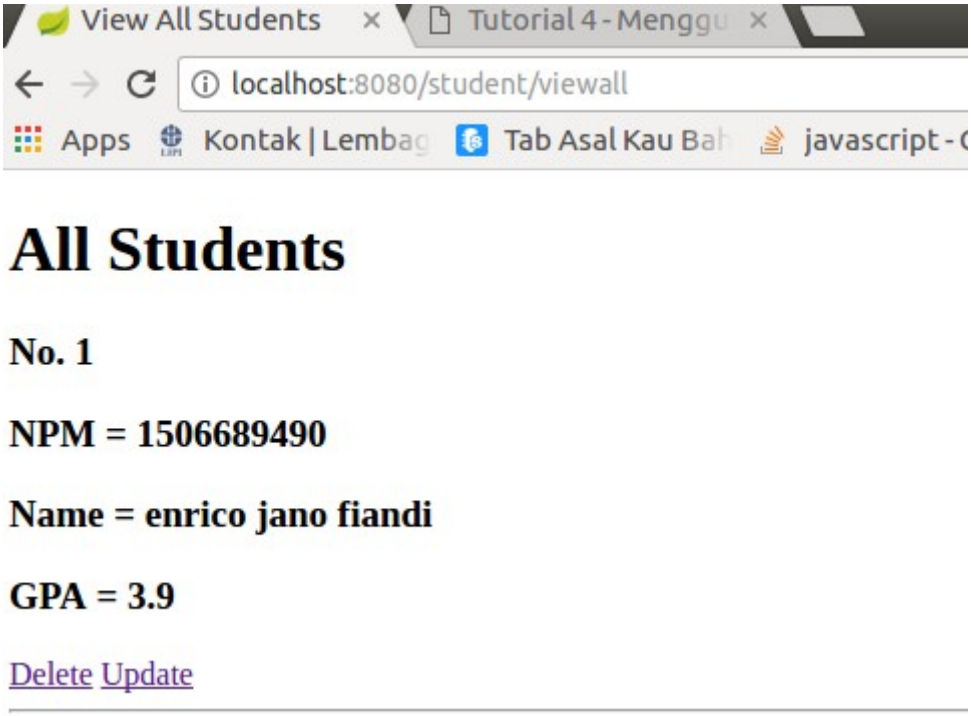
Kasus student dengan npm ada



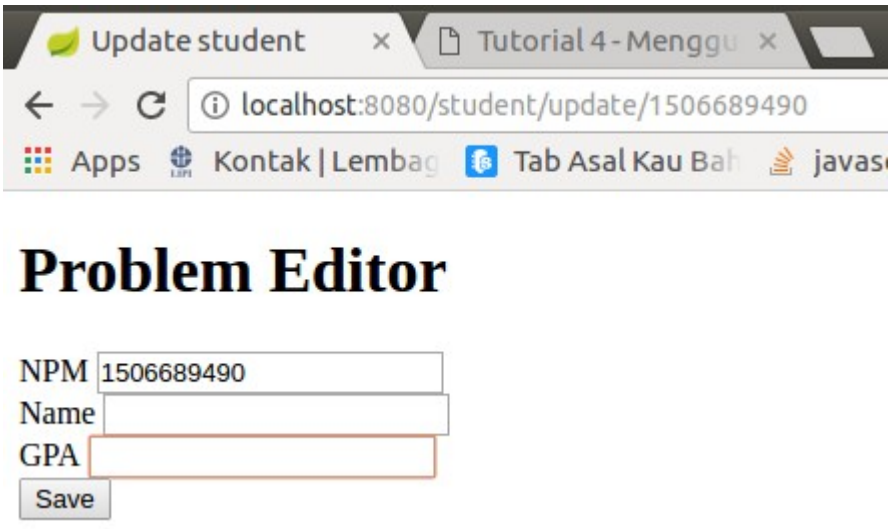
Kasus data berhasil diupdate



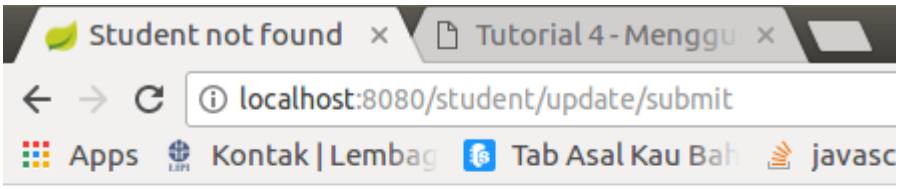




Kasus gpa dan name tidak diisi **(Validasi Objek Parameter)**



Akan menampilkan not-found (mungkin lebih baik jika dibuat halaman errorPage.html untuk kasus seperti ini, namun sementara saya menggunakan not-found untuk kasus validasi object parameter)



Student not found

NPM = null

Hal yang saya pelajari dari tutorial ini

Saya belajar bahwa kita dapat menggunakan objek sebagai parameter dalam menerima http request, dan saya juga belajar bagaimana cara melakukan validasi jika objek menjadi parameter dalam menerima HTTP Request menggunakan anotasi *@Notnull*, *@NotEmpty* dan *@Valid*. Saya juga belajar bagaimana cara mengkoneksikan spring framework dengan database dan belajar bagaimana cara merangkai aplikasi dengan prinsip OOP.