PERTANYAAN

1. Menggunakan object sebagai parameter pada form POST. Berikut cara validasinya:

   Validasi dapat dilakukan dengan annotasi @Valid pada controller dan class BindingResult, selain itu juga harus menggunakan annotasi validasi seperti @NotNull, @Size(min=,max=), dll pada view sebagai parameter validasinya.

   Contoh implementasi pada controller:

```
@RequestMapping(value="/student/update/submit", method=RequestMethod.POST)
public String updateSubmit(@Valid StudentModel student, BindingResult bindingResult)
{
    if(bindingResult.hasErrors()) {
        log.error("Error parameter {}", bindingResult.getAllErrors());
        return "not-found";
    } else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
}
```

   Contoh implementasi pada view:

```
<form action="/student/update/submit" method="post" th:object="${student}">
    <div>
        <label for="npm">NPM</label>
        <input type="text" name="npm" readonly="true" th:field="${student.npm}" />
    </div>
    <div>
        <label for="name">Name</label>
        <input type="text" name="name" th:field="${student.name}"/>
        <label th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Name Error</label>
    </div>
    <div>
        <label for="gpa">GPA</label>
        <input type="text" name="gpa" th:field="${student.gpa}"/>
        <label th:if="${#fields.hasErrors('gpa')}" th:errors="*{gpa}">GPA Error</label>
    </div>

    <div>
        <button type="submit" name="action" value="save">Update</button>
    </div>
</form>
```

2. Perbedaan penganganan request POST dan GET.

   Menurut saya, submit form biasa menggunakan method POST dibandingkan method GET karena jika menggunakan method POST, data dikirimkan melalui JSON body, sedangkan jika menggunakan GET, maka akan dikirimkan dengan query di URL.

   Dengan demikian, data yang dikirimkan untuk disimpan di server lebih baik menggunakan method POST, sedangkan data request untuk suatu webpage dengan ID, seperti page untuk menampilkan suatu data per halaman, lebih baik menggunakan method GET.

   Dalam hal penganganan di controller Spring Boot, perbedaan pertama terletak pada parameter annotasi yang digunakan, jika menggunakan method POST, maka akan seperti:

   `@RequestMapping(value="/student/update/submit", method=RequestMethod.POST)`

   Sedangkan jika menggunakan method GET, maka akan seperti:

```
@RequestMapping(value="/student/add/submit", method=RequestMethod.GET)
```

Selebihnya akan ± sama.

3. Sebuah method menerima GET dan POST sekaligus.

   Tergantung pengertian “menerima GET dan POST sekaligus”.

   Jika yang dimaksud adalah sebuah method dapat menghandle kedua hal tersebut namun request yang dilakukan tetap salah satu, jawabannya BISA. Seperti pada contoh berikut:

```
@RequestMapping(value="/student/update/submit", method= {RequestMethod.POST, RequestMethod.GET})
public String updateSubmit(@Valid StudentModel student, BindingResult bindingResult)
{
    if(bindingResult.hasErrors()) {
        log.error("Error parameter {}", bindingResult.getAllErrors());
        return "not-found";
    } else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
}
```

   Sedangkan jika yang dimaksud adalah sebuah form thymeleaf mengirimkan langsung POST request dan GET request, kemudian method menghandlenya sekaligus, jawabannya TIDAK. Hal ini dikarenakan form html hanya bisa mendefine 1 buah method request yang digunakan. *CMIIW

LATIHAN IMPLEMENTASI METHOD

- Method Delete

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel archive = studentDAO.selectStudent(npm);
    if(archive == null) {
        return "not-found";
    } else {
        studentDAO.deleteStudent(npm);
        return "delete";
    }
}
```

Method delete() digunakan untuk menghapus data student di database. Jika parameter npm terpenuhi dan terdapat npm tersebut di database, maka record tersebut akan dihapus. Jika tidak terpenuhi, maka akan mengembalikan page bahwa npm tersebut tidak ada.

- Method Update

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel archive = studentDAO.selectStudent(npm);
    if(archive == null) {
        return "not-found";
    } else {
        model.addAttribute("student", archive);
        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 update() digunakan untuk mengembalikan form update jika parameter npm terpenuhi dan akan melakukan check database apakah npm tersebut terdaftar. Jika iya, maka akan dikembalikan sebuah form untuk update. Jika tidak, maka akan mengembalikan page bahwa npm tersebut tidak ada.

Method updateSubmit() digunakan untuk menerima parameter data baru dari form update dan melakukan update database. Parameter diterima dalam bentuk primitive types, sehingga dibuatkan object StudentModel baru untuk dikirim ke Mapper dan melakukan update database.

- Method Update dengan parameter Object

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel archive = studentDAO.selectStudent(npm);
    if(archive == null) {
        return "not-found";
    } else {
        model.addAttribute("student", archive);
        return "form-update";
    }
}

/*
 * Update menggunakan parameter object
 * Jika menghandle POST dan GET sekaligus
 * @RequestMapping(value="/student/update/submit", method={RequestMethod.POST, RequestMethod.GET})
 */
@RequestMapping(value="/student/update/submit", method=RequestMethod.POST)
public String updateSubmit(@Valid StudentModel student, BindingResult bindingResult)
{
    if(bindingResult.hasErrors()) {
        log.error("Error parameter {}", bindingResult.getAllErrors());
        return "not-found";
    } else {
        studentDAO.updateStudent(student);
        return "success-update";
    }
}
```

Implementasi method update() dilakukan sama persis seperti Method update dengan parameter primitive types.

Sedangkan method updateSubmit() menerima parameter StudentModel dengan annotasi @Valid dan tambahan parameter BindingResult. Annotasi @Valid digunakan untuk menandakan bahwa object tersebut harus valid berdasarkan apa yang didefinisikan pada class object tersebut. Sedangkan BindingResult digunakan untuk melakukan “catch” jika terjadi exception.

Implementasi @Valid dan Binding Result harus diikuti dengan annotasi validasi pada model seperti @NotNull untuk memastikan data tidak boleh kosong, serta @Size(min=,max=) untuk menentukan panjang karakter yang diperbolehkan.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class StudentModel
{
    private String npm;
    @Size(min=1)
    @NotNull
    private String name;
    @NotNull
    private double gpa;

}
```

Selain itu, diperlukan juga conditional thymeleaf pada view. th:if digunakan untuk melakukan check apakah input pada field tersebut valid (conditional), dan th:errors untuk listing semua error.

```
<form action="/student/update/submit" method="post" th:object="${student}">
    <div>
        <label for="npm">NPM</label>
        <input type="text" name="npm" readonly="true" th:field="${student.npm}" />
    </div>
    <div>
        <label for="name">Name</label>
        <input type="text" name="name" th:field="${student.name}"/>
        <label th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Name Error</label>
    </div>
    <div>
        <label for="gpa">GPA</label>
        <input type="text" name="gpa" th:field="${student.gpa}"/>
        <label th:if="${#fields.hasErrors('gpa')}" th:errors="*{gpa}">GPA Error</label>
    </div>

    <div>
        <button type="submit" name="action" value="save">Update</button>
    </div>
</form>
```