

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

- Validasi yang dilakukan dapat menggunakan custom pengecekan dari masing masing variable object.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    } else {
        model.addAttribute("npm", student.getNpm());
        return "not-found";
    }
}
```

- Berikut validasi yang dapat dilakukan, bila datanya kosong akan mengembalikan view error

```
if(student.getName() == null || student.getName() == "") {
    return "error";
}
```

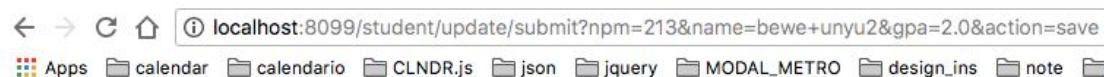
Apakah validasi diperlukan?

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

- Tergantung sesuai kebutuhan, dan dari seberapa penting pencegahan untuk menghindari error.
- Jika dari level parameter/data seperti name dan gpa tidak boleh null. Maka, validasi dibutuhkan pengecekan tiap valuenya.

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?

- Request GET biasanya digunakan untuk *retrieve* data. Sedangkan POST biasanya digunakan jika ada perubahan data.
- Karena jika menggunakan GET method, data post akan terlihat di link address. Memang tidak mengapa, tetapi jika data tersebut perlu di *secure*, munculnya data di link address sebaiknya tidak dilakukan.
- Berikut tampilan jika menggunakan GET.



The screenshot shows a web browser window with the address bar displaying 'localhost:8099/student/update/submit?npm=213&name=bewe+unyu2&gpa=2.0&action=save'. Below the address bar, there is a navigation bar with several folder icons and labels: 'Apps', 'calendar', 'calendario', 'CLNDR.js', 'json', 'jquery', 'MODAL\_METRO', 'design\_ins', 'note', and an additional folder icon.

**Data berhasil diupdate**

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
- Jika nama methodnya sama, tidak bisa.
  - Tetapi jika ingin menggunakan route yang sama bisa asalkan dibedakan nama metodenya.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    if(student.getName() == null || student.getName() == "") {
        return "error";
    }

    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    }else {
        model.addAttribute ("npm", student.getNpm());
        return "not-found";
    }
}

@RequestMapping(value = "/student/update/submit", method = RequestMethod.GET)
public String updateSubmitGET (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    if(student.getName() == null || student.getName() == "") {
        return "error";
    }

    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    }else {
        model.addAttribute ("npm", student.getNpm());
        return "not-found";
    }
}
```

○

4. Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan
- Pada StudentMapper di isi query untuk men-*delete* data student berdasarkan npm

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

- Pada StudentServiceDatabase, sesuai dengan fungsinya sebagai gate sebelum dan setelah mengakses database, akan memanggil deleteStudent di StudentMapper.

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

- Akhirnya controller untuk melakukan fungsi delete melalui service student 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";
    }
}
```

5. Method yang Anda buat pada Latihan Menambahkan Update, jelaskan

- Dibuat dahulu query untuk update dengan parameter yang ingin di ubah.

```
@Update("UPDATE student SET name = #{nama}, gpa = #{gpa} where npm = #{npm}")
void updateStudent(@Param("npm") String npm, @Param("nama") String nama, @Param("gpa") double gpa);
```

- Dipersiapkan servicenya

```
@Override
public void updateStudent(String npm, String nama, double gpa)
{
    log.info("Student " + npm + "updated");
    studentMapper.updateStudent(npm, nama, gpa);
}
```

- Sehingga service update student dapat dipanggil. Dibagian ini masih menggunakan @RequestParam sehingga butuh banyak parameter untuk mengupdate data.

```
/**
 * Method to update data student with params declared
 * @param model
 * @param npm
 * @param name
 * @param gpa
 * @return
 */

@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (Model model,
    @RequestParam(value = "npm", required = false) String npm,
    @RequestParam(value = "name", required = true) String name,
    @RequestParam(value = "gpa", required = false) double gpa)
{
    StudentModel student = studentDAO.selectStudent (npm);
    if (student != null) {
        studentDAO.updateStudent(npm, name, gpa);
        return "success-update";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

- Berikut view update.

```
<form action="/student/update/submit" method="post">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}"/>
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}"/>
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}"/>
  </div>

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

6. Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan

- Dibagian ini sudah menggunakan single object StudentModel. Sehingga parameter tidak diperlukan karena sudah dibungkus di object StudentModel. Jika membutuhkan paramaternya, tinggal memanggil variable dari object StudentModel.

```
/**
 * Well, this method is using StudentModel as the parameter
 * So, the method updateSubmit will only need 1 object StudentModel
 * And that object's variable will be used to update the selected data student
 * @param student
 * @param model
 * @return
 */
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    } else {
        model.addAttribute ("npm", student.getNpm());
        return "not-found";
    }
}
```

- Berikut tmapilan view update, terdapat perbedaan pada tag “th:object” dan “th:field”

```
<form action="/student/update/submit" th:object="${student}" method="post">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:field="*{npm}"/>
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" th:field="*{name}"/>
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" th:field="*{gpa}"/>
  </div>

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