

a. Ringkasan dari materi yang Anda telah pelajari pada tutorial kali ini

Saya mempelajari bagaimana melakukan beberapa method menggunakan request method POST, menggunakan objek sebagai parameter untuk membuat input lebih praktis, serta menggunakan database pada program sebagai penyimpan data.

b. Hasil jawaban dari Pertanyaan

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.

Jika kita ingin melakukan validasi, dapat dihandle di method-method yang berhubungan dengan input, untuk memeriksa apakah input null atau tidak.

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?

Menurut saya, form submit lebih menggunakan POST karena memang digunakan untuk menginput atau mengubah data, dibandingkan GET yang digunakan untuk mengambil atau melihat data. Ya, kita perlu melakukan penanganan berbeda di method controller, dimana kita perlu menulis di header method = RequestMethod.POST jika ingin menggunakan POST.

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

Tidak memungkinkan. Karena kita hanya dapat memasukkan satu kata kerja saat melakukan request.

c. Method yang dibuat

1. Delete

- a. Mengedit viewall.html untuk me-*redirect* link ke halaman berisi delete.html atau not-found.html, tergantung dari hasil mapper /student/delete.

```
<div th:each="student, iterationStatus: ${students}">  
    <a th:href="'/student/delete/' + ${student.npm}">Delete Data</a>  
</div>
```

- b. Menambahkan method deleteStudent di StudentMapper.java yang dihubungkan ke database.

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

- c. Melengkapi method deleteStudent pada StudentServiceDatabase.java untuk memanggil method deleteStudent.

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

- d. Melengkapi method delete di StudentController.java. Jika student ada di database, maka bisa di 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";
    }
}
```

2. Update

- a. Mengedit viewall.html untuk me-redirect link ke halaman berisi form-update.html atau not-found.html, tergantung dari hasil mapper /student/update.

```
<a th:href="/student/update/" + ${student.npm}">Update Data</a>
```

- b. Menambahkan method updateStudent pada StudentMapper yang dihubungkan ke database.

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

- c. Menambahkan method updateStudent pada StudentServiceDatabase.java untuk memanggil method updateStudent.

```
@Override
public void updateStudent(StudentModel student) {
    // TODO Auto-generated method stub
    log.info("student updated");
    studentMapper.updateStudent(student);
}
```

- d. Membuat form-update.html sebagai halaman yang akan mengupdate sebuah data.

- e. Membuat halaman success-update.html yang di-redirect setelah selesai melakukan update di form-update.html
- f. Menambahkan method update di StudentController.java. Jika student ada di database, maka bisa diupdate.

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

- g. Menambahkan method updateSubmit, untuk meng-update student kemudian meredirect ke halaman html, di StudentController.java.

```
@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";
}
```

3. Object Sebagai Parameter

- a. Menambahkan th:object dan th:field pada form di form-update.html.

```
<h1 class="page-header">Update Editor</h1>
<form action="/student/update/submit" method="post" th:object="${student}">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="**{npm}"/>
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="**{name}"/>
    </div>
    <div>
        <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="**{gpa}"/>
    </div>
    <div>
        <button type="submit" name="action" value="save">Update</button>
    </div>
</form>
```

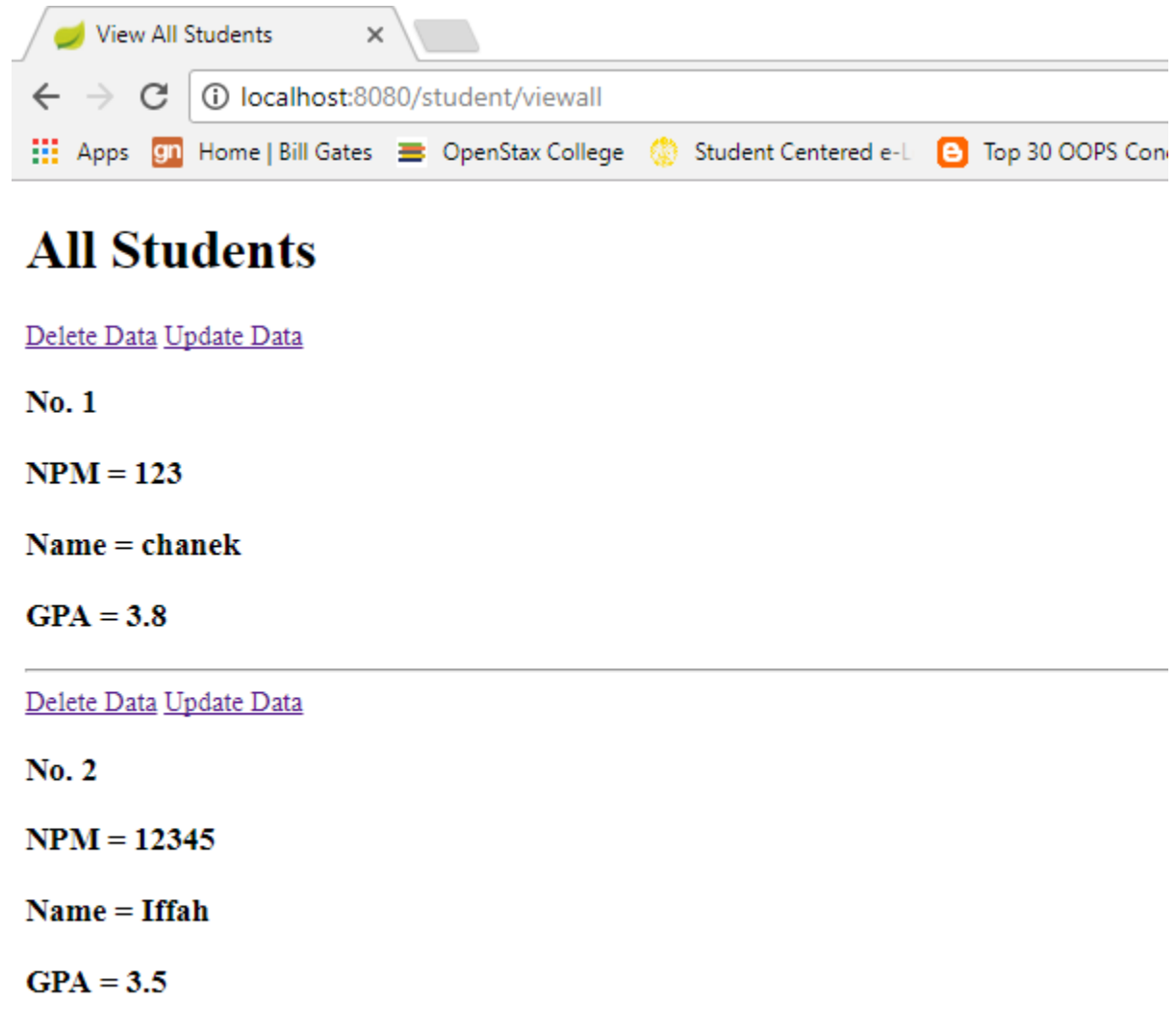
- b. Mengubah method updateSubmit di StudentController agar menerima parameter berupa StudentModel.

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

May Iffah Rizki
1506757661
APAP – A

Hasil akhir:

Viewall.html:



The screenshot shows a web browser window with a single tab titled "View All Students". The address bar displays "localhost:8080/student/viewall". Below the address bar is a navigation bar with several links: "Apps", "Home | Bill Gates", "OpenStax College", "Student Centered e-L", and "Top 30 OOPS Con". The main content area features a large heading "All Students" in a bold, serif font. Below this heading are two rows of student data, each preceded by a link labeled "Delete Data Update Data". The first row shows a student with ID "No. 1", NPM "123", Name "chanek", and GPA "3.8". The second row shows a student with ID "No. 2", NPM "12345", Name "Iffah", and GPA "3.5". Each row is separated by a horizontal line.

All Students

[Delete Data Update Data](#)

No. 1

NPM = 123

Name = chanek

GPA = 3.8

[Delete Data Update Data](#)

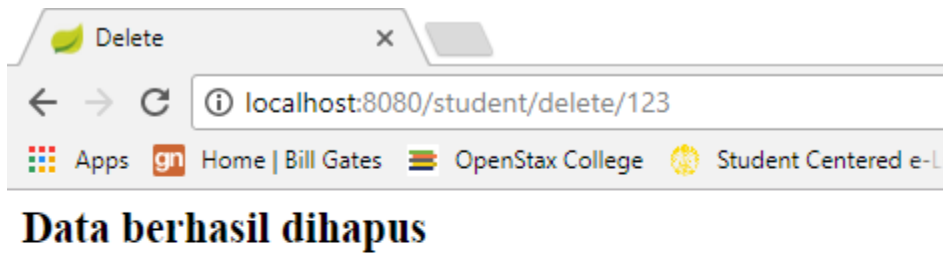
No. 2

NPM = 12345

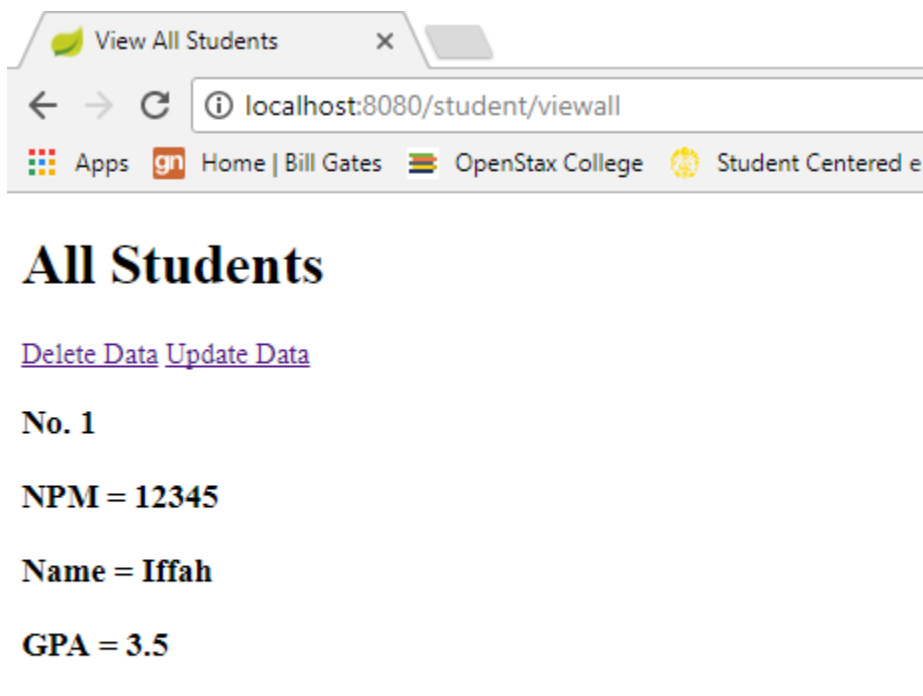
Name = Iffah

GPA = 3.5

May Iffah Rizki
1506757661
APAP – A
Menghapus data dengan npm 123:



Melihat kembali viewall.html, data dengan npm 123 sudah dihapus:

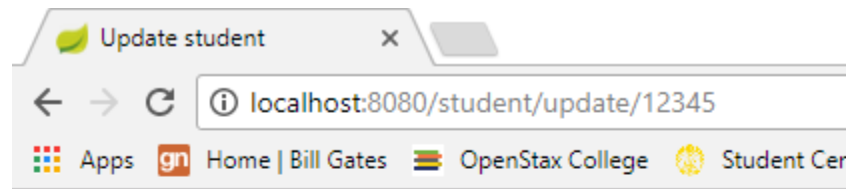


May Iffah Rizki

1506757661

APAP – A

Meng-update data dengan npm 12345, mengubah GPA menjadi 4:



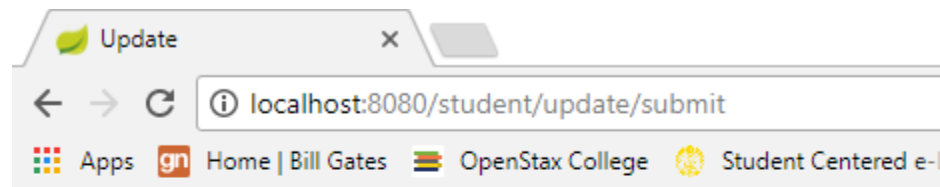
Update Editor

NPM

Name

GPA

Data berhasil diupdate:



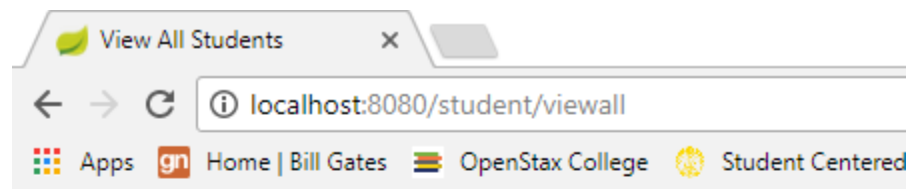
Data berhasil diupdate

May Iffah Rizki

1506757661

APAP – A

Melihat kembali viewall.html, data dengan npm 12345 sudah diupdate:



All Students

[Delete Data](#) [Update Data](#)

No. 1

NPM = 12345

Name = Iffah

GPA = 4.0
