Penjelasan setiap Pertanyaan

1. Validasi form perlu dilakukan. Validasi pertama perlu dilakukan di view. Validasi kedua dilakukan di backend yaitu di controller. Jika menggunakan Object sebagai parameter pada POST maka validasi bisa dilakukan dengan menambah method untuk validasi Request. Selain itu bisa juga menggunakan dependency spring boot untuk validasi request.
2. Form input biasanya memiliki beberapa data field yang akan diparsing di controller. Jika menggunakan method GET semua field tersebut akan di tampilkan di request URL nya. Jika kita punya 10 field, jika menggunakan method GET maka request URL akan sangat panjang. Jika form berisi file, maka dengan method GET tidak bisa karena ada batasan data lengthnya sedangakan POST tidak ada. Dari sisi security data yang kita submit akan lebih aman karena tidak tersimpan di cache jika menggunakan POST, menggunakan mehtod GET sebaliknya.
   Di controller juga kita perlu menambahkan request method GET atau POST. Untuk implementasi di body method, tidak ada perbedaan antara GET dan POST.

3. Satu method mungkin saja menerima request method GET dan POST, namun bukan merupakan implementasi yang baik. Seharusnya satu method spesifik untuk satu request saja.

Penjelasan setiap Method

1. Method Delete
   - Implementasi pada StudentController

```
@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";
    }

}
```

   Method ini menerima request dengan parameter npm. Pertama dipastikan dulu apakah student dengan npm tersebut ada di database. Maka di gunakan method

selectStudent yang mereturn object StudentModel. Jika ada berarti dipanggil method deleteStudent di service StudentService interface yang implementasi menthod nya di StudentServiceDatabase.
Jika npm tidak ditemukan maka di return view not found untuk memberikan informasi bahwa student dengan npm tersebut tidak ada di database.

- Implementasi pada StudentMapper

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

Class ini merupakan dao yang akan berhubungan dengan transaksi ke database. Method ini akan melakukan query delete data dari table student berdasarkan npm (Parameter method).

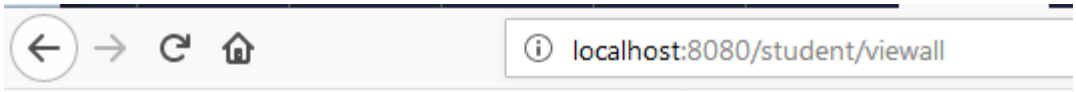
- Implementasi pada StudentServiceDatabase

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

Class ini merupakan impmentasi dari interface StudentService yang berfungsi sebagai perantara antara DAO dan controller. Pada method ini hanya memanggil method deleteStudent di DAO dan juga melakukan proses log. Menerima parameter npm dari Object student yang akan di hapus.

Berikut adalah proses delete student

- Tampilan viewall



No. 1

NPM = 124

Name = Chanek

GPA = 3.5

Delete Data
Update Data

No. 2

NPM = 1606954735

Name = Arri Kurniawan

GPA = 4.0

Delete Data
Update Data

No. 3

NPM = 1606954741

Name = Ayu N

GPA = 3.2

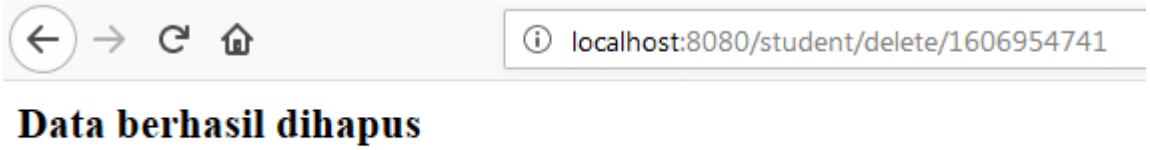
Delete Data
Update Data

- Tampilan delete dengan parameter npm yang tidak valid

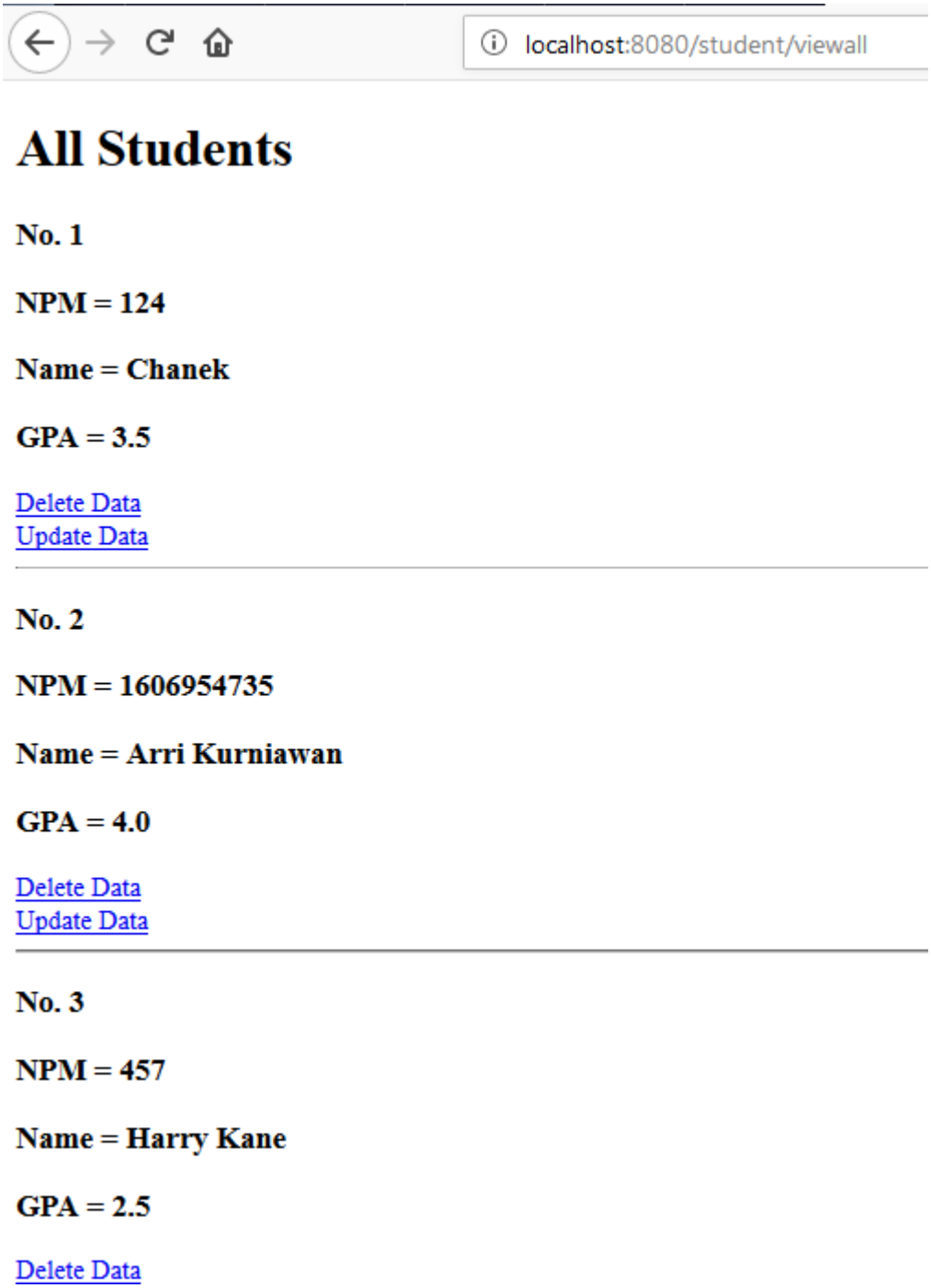
Student not found

NPM = 11

- Tampilan delete dengan parameter yang valid dan berhasil delete



- Tampilan viewall setelah delete



2. Method Update

- Implementasi pada StudentController

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

}
```

Method GET ini akan menampilkan form untuk melakukan update terhadap existing data. Namun terlebih dahulu di cek apakah object student dengan npm yang diberikan ada didatabase dengan memanggil selectStudent method. Jika ada maka akan ditampilkan form-update view dengan data student yang di parsing ke view. Di view <form action="/student/update/submit" method="post"> ketika form di submit maka data akan di post ke StudentController.
Jika npm tidak ada di database maka akan di return view not found.

Kemudian di StudentController ditambahkan method yang akan memproses 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 menerima request POST dengan parameter npm, name dan gpa. Kemudian dalam method dicreate sebuah object student baru berdasarkan data request param. Dipanggil method updateStudent di service StudentService interface yang implementasi menthod nya di StudentServiceDatabase.
Kemudian akan menampilkan view success-update.

- Implementasi pada StudentServiceDatabase

```
@Override
public void updateStudent (StudentModel student)
{
    log.info("update student "+ student.getNpm());
    studentMapper.updateStudent(student);
}
```

Class ini merupakan impmentasi dari interface StudentService yang berfungsi sebagai perantara antara DAO dan controller. Pada method ini hanya memanggil method updateStudent di DAO dan juga melakukan proses log. Menerima parameter object StudentModel yang akan di update datanya.

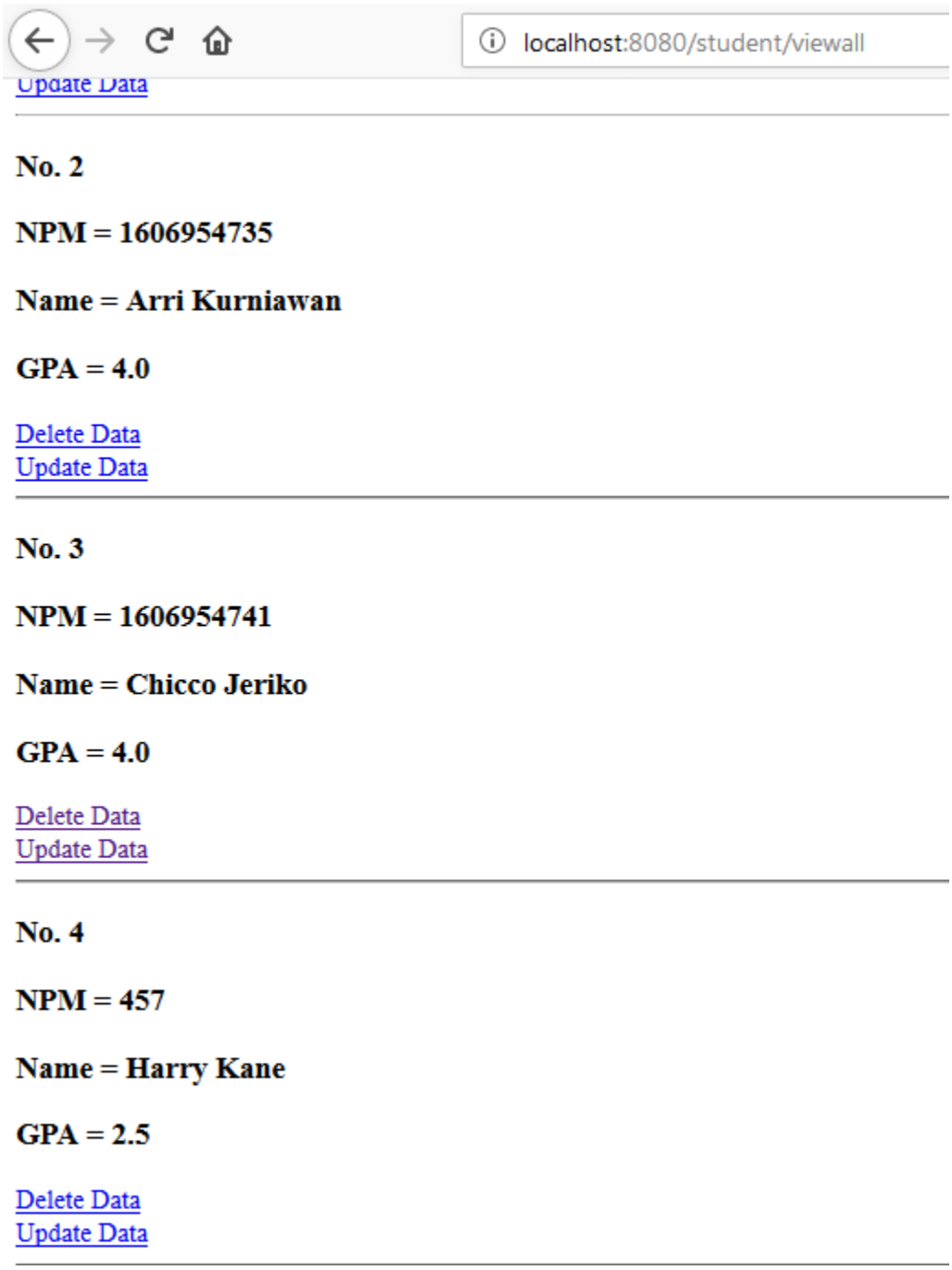
- Implementasi pada StudentMapper

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

Class ini merupakan dao yang akan berhubungan dengan transaksi ke database. Method ini akan melakukan query update data dari table student berdasarkan npm (Parameter method). Data yang diupdate adalah name dan gpa berdasarkan npm.

Berikut adalah proses update student

- Tampilan viewall

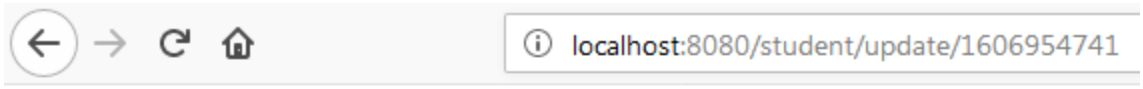


- Tampilan view update dengan invalid npm

Student not found

NPM = 11

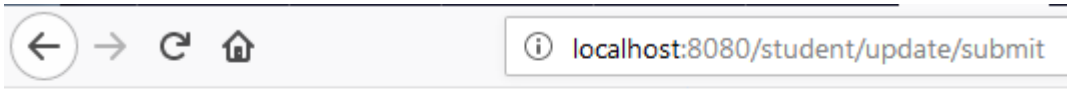
- Tampilan view form update



Update Student

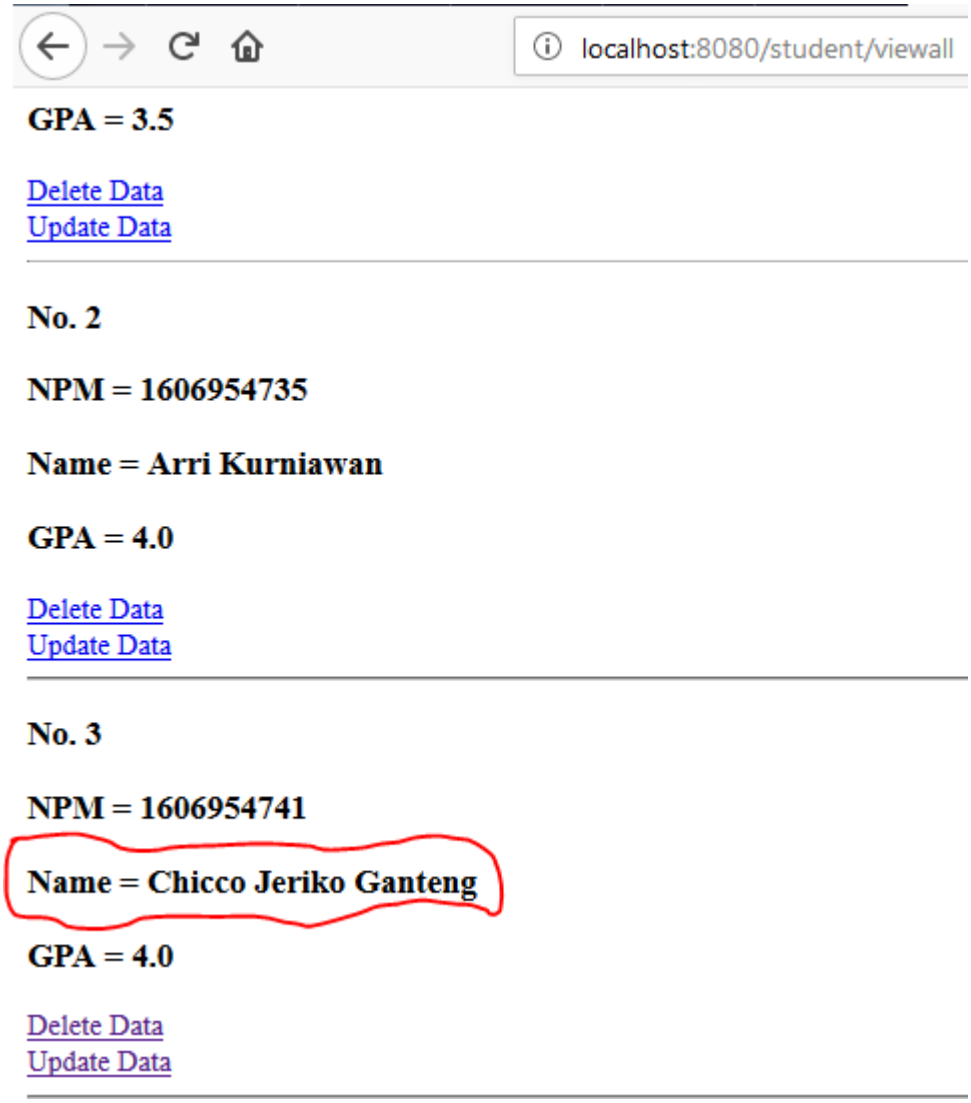
NPM 1606954741
Name Chicco Jeriko Ganteng
GPA 4.0
Update

- Tampilan view berhasil update



Data berhasil diubah

- Tampilan viewall setelah update



3. Method untuk menggunakan Object sebagai Parameter

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

    return "success-update";
}
```

Parameter method update dijadikan object student model. Di view akan ditambahkan nama field untuk setiap input dan menambah nama object yang akan di post dari view

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

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

Nama field akan menjadi nama attribute pada object student yang ada di tag form th:object=”${student}”.