Lidia Natalia Manik
Ekstensi
1706106803 Tutorial 4

MENGGUNAKAN DATABASE DAN MELAKUKAN DEBUGGING

DALAM PROJECT SPRING BOOT

Membuat Database

- Membuat database dengan nama tutorial4

```
CREATE DATABASE tutorial4
```

- Membuat table dan field npm, name, dan gpa

```
CREATE table student (npm varchar(20) PRIMARY KEY, name varchar(45) null,
gpa double null
```

Latihan

1. Menambahkan Delete

Pada viewall.html > di bawah th : each tambahkan

```
<a th:href="'/student/delete/' + ${student.npm}">Delete Data</a><br>
```

Penjelasan : bagian ini berfungsi untuk membuat url yang me-refer url student/delete/[student_npm]

Pada class **StudentMapper** > tambahakan method **deleteStudent** sebagai berikut :

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

Penjelasan : bagian ini berfungsi untuk membuat query delete dan memanggil fungsi deleteStudent

Pada class **StudentServiceDatabase** > lengkapi method **deleteStudent**

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

Penjelasan : Bagian ini berfungsi untuk membuat debugging yang akan mencetak student [nomor_NPM] deleted pada console jika proses menghapus berhasil dilakukan.

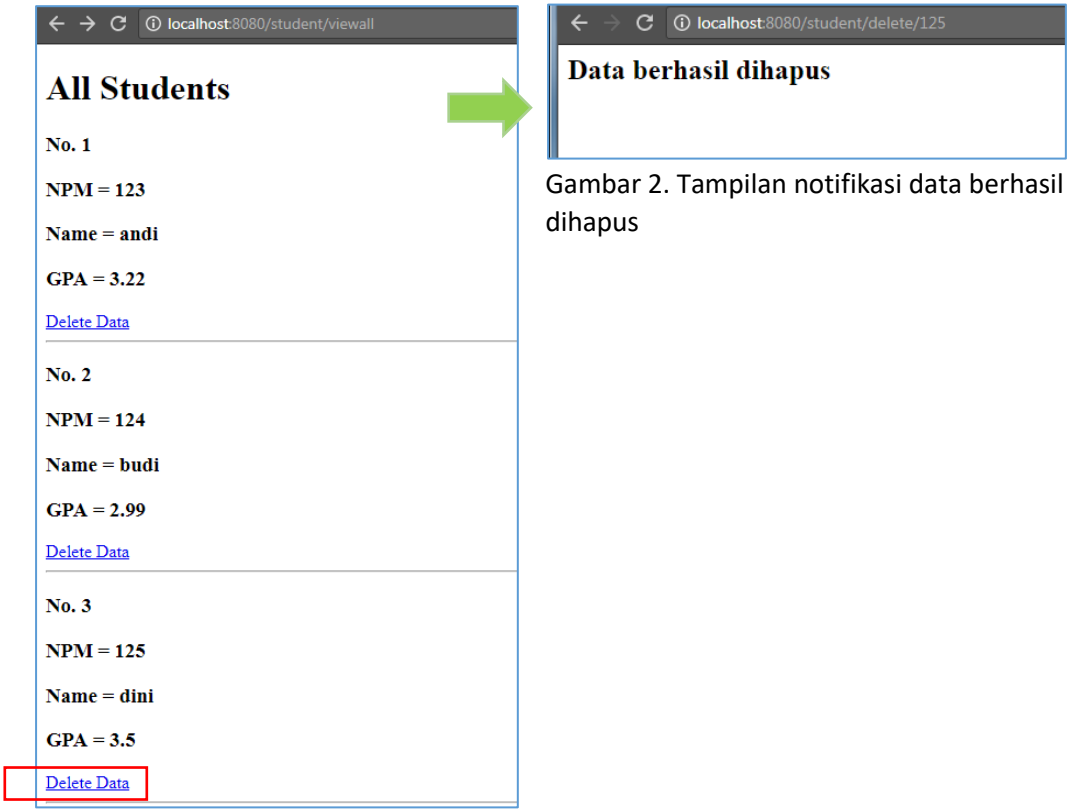
Pada class **StudentController** > lengkapi method **delete**

```
@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";
    }
}
```

Pada method ini, apabila data berhasil ditemukan, method deleteStudent akan dipanggil dan menampilkan view delete sementara jika npm tidak ditemukan, method akan menampilkan view not-found.

Result page :

Tampilan viewall student dan setelah berhasil dihapus adalah sebagai berikut :



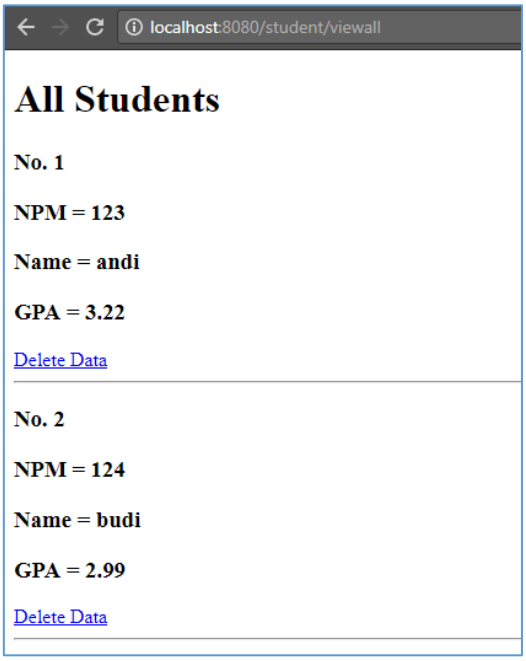
Gambar 1. Tampilan data student

Gambar 2. Tampilan notifikasi data berhasil dihapus

Pada console akan muncul tampilan berikut :

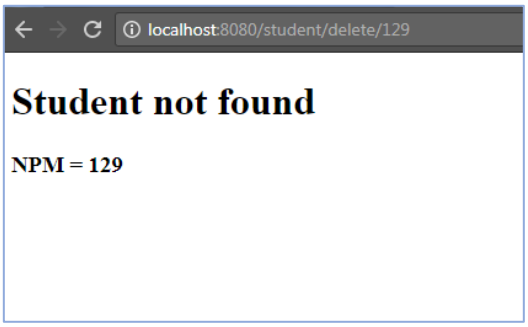
```
APAPtutorial4 - ApapTutorial04Application [Spring Boot App] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (2018/03/09 13:15:04)
·09 14:10:58.478  INFO 11808 --- [nio-8080-exec-1] c.e.service.StudentServiceDatabase       : select student with npm 124
·09 14:10:58.480  INFO 11808 --- [nio-8080-exec-1] c.e.service.StudentServiceDatabase       : student 124deleted
```

Gambar 3. Tampilan debugging delete pada console



Gambar 4. Tampilan data terbaru student

Tampilan apabila npm tidak sama dengan record yang ada di database :



Gambar 5. Tampilan page apabila student tidak ditemukan

2. Menambahkan Update

Pada viewall.html > di bawah th : each tambahkan

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

Penjelasan : bagian ini berfungsi untuk membuat url yang me-refer url student/update/[student_npm]

Pada class **StudentMapper** > tambahakan method **updateStudent** sebagai berikut :

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

Penjelasan : bagian ini berfungsi untuk membuat query update dan memanggil fungsi update Student

Pada class StudentService tambahkan method updateStudent

```
    void updateStudent (StudentModel student);
```

Pada class **StudentServiceDatabase** > lengkapi method **updateStudent**

```
    @Override
    public void updateStudent (StudentModel student)
    {
       studentMapper.updateStudent(student);
    }
```

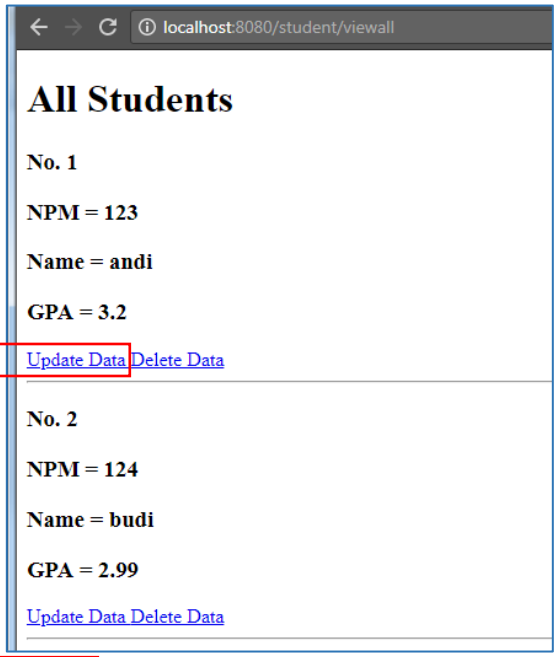
Pada class **StudentController** > lengkapi method **update**

```
    @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";
        }
    }

    @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";
    }
```

Pada method update, apabila data berhasil ditemukan, view form-update akan ditampilkan sementara jika npm tidak ditemukan, view not-found akan ditampilkan.

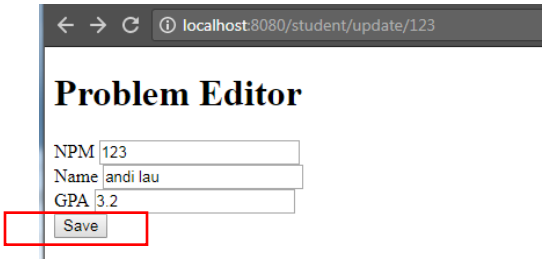
Result page :

Tampilan viewall student:

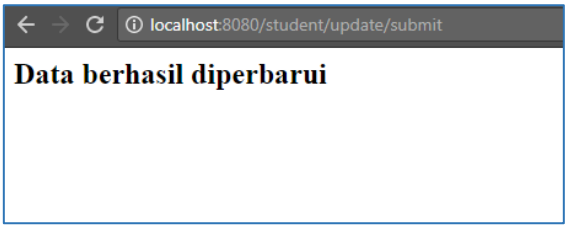


Gambar 6. Tampilan page awal

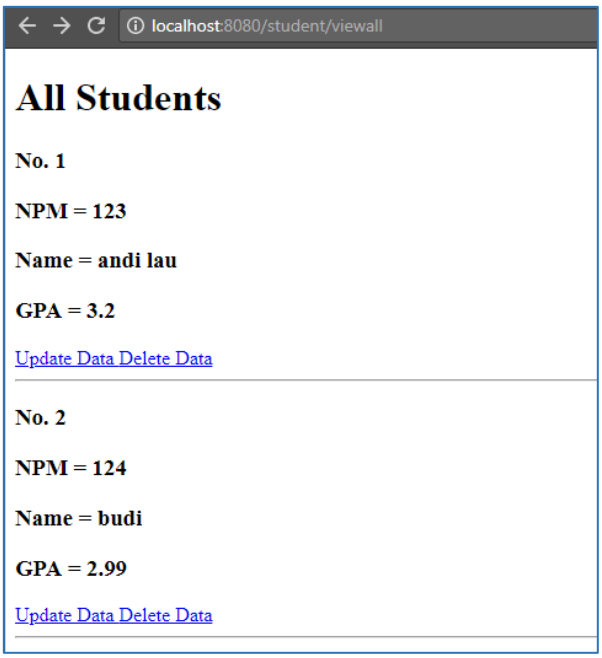
Apabila url Update Data diklik, muncul tampilan seperti gambar 7 dan jika berhasil seperti gambar 8



Gambar 7 Tampilan form-update

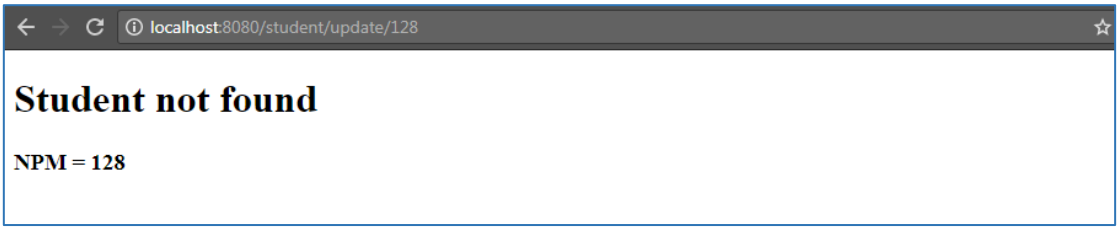


Gambar 8. Tampilan view update



Gambar 9. Tampilan viewall terbaru

Pada proses selanjutnya adalah update data student yang tidak terdapat di database. Page yang muncul adalah sebagai berikut :



Gambar 10. Tampilan view not-found update

3. Menggunakan Object Sebagai Parameter

Pada latihan 1 dan 2 , project menggunakan RequestParam untuk menghandle form submit sehingga banyak parameter pada method.

Pada latihan ini, method updateSubmit akan menerima parameter berupa model StudentModel.

Pada form-update.html, ubah menjadi:

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

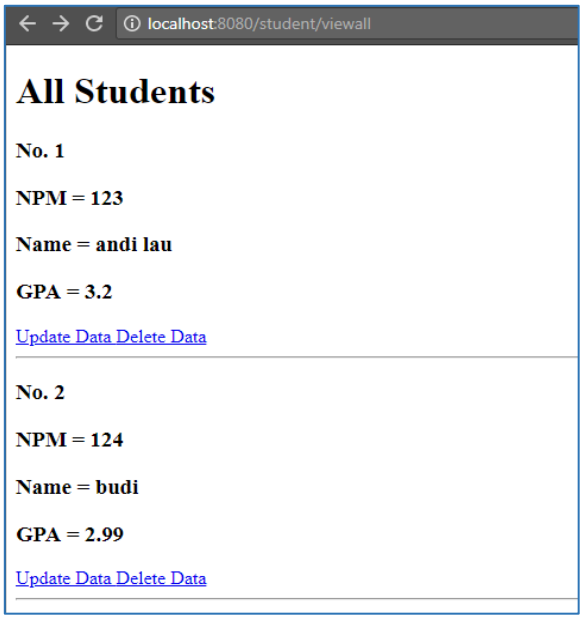
Pada class StudentController > method updateSubmit diubah menjadi :

```
    @RequestMapping (value = "/student/update/submit", method =
RequestMethod.POST)
    public String updateSubmit (@ModelAttribute ("student") StudentModel
student, BindingResult br)
    {
      if (br.hasErrors()) {
              studentDAO.updateStudent (student);
              return "success-update";
      }else {
            return "not-found";
      }
    }
```

Lidia Natalia Manik
Ekstensi
1706106803 Tutorial 4

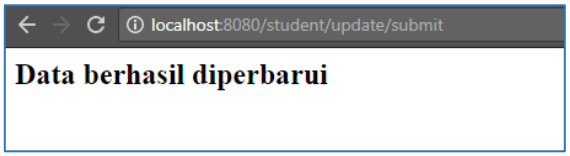
Result page :

Tampilan awal dan tampilan form adalah seperti gambar di bawah ini :

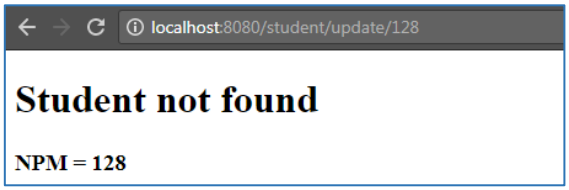


Gambar 11. Tampilan awal viewall Student

Jika data berhasil diperbarui, muncul page seperti gambar 12 , namun jika tidak muncul page 13

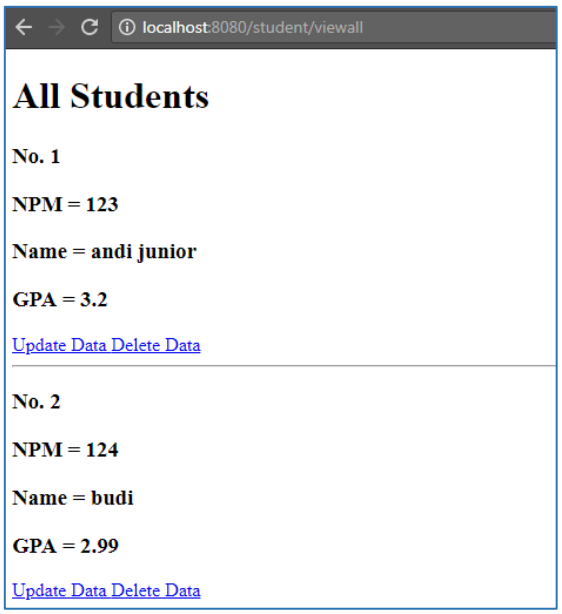


Gambar 12. Tampilan page notifikasi success



Gambar 13. Tampilan page notifikasi gagal

Lidia Natalia Manik
Ekstensi
1706106803



Gambar 14. Tampilan viewall Student terbaru

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?
   Apabila menggunakan object, validasi tidak diperlukan karena sudah validasinya required optional.

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?
   Karena method POST menyembunyikan informasi dari url sementara method GET tidak.
   Untuk penanganannya tidak ada yang berbeda baik **post** ataupun **get**. Selain itu, limit data yang dilewati lebih besar (text data , binary data) dibandingkan method get.

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Tidak. Form hanya bisa menerima satu method saja.

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

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