

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Latihan Menambahkan Delete

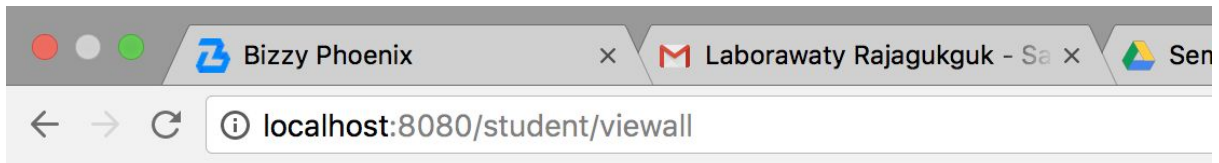
Berikut merupakan method Delete pada StudentController. Method ini sudah mengimplementasikan penggunaan database. Pada method object dihapus berdasarkan npm.

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

Pada file viewall.html, ditambahkan code yang akan menampilkan sebuah link dan link tersebut akan terhubung kepada method delete diatas.

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

Berikut merupakan tampilan viewall yang sudah memiliki link delete.



All Students

No. 1

NPM = 123

Name = Chanek

GPA = 3.6

[Delete Data](#)

No. 2

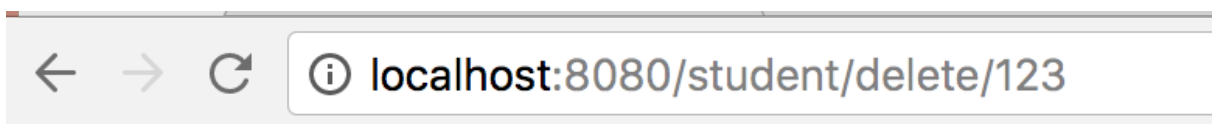
NPM = 124

Name = Chanek Jr.

GPA = 3.4

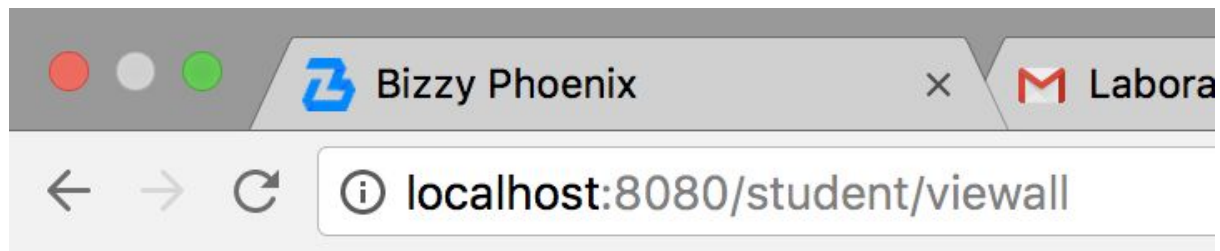
[Delete Data](#)

Ketika klik link Delete Data pada data dengan NPM = 123



Data berhasil dihapus

Setelah menghapus data tersebut, maka data tidak akan ditampilkan lagi pada viewall.



All Students

No. 1

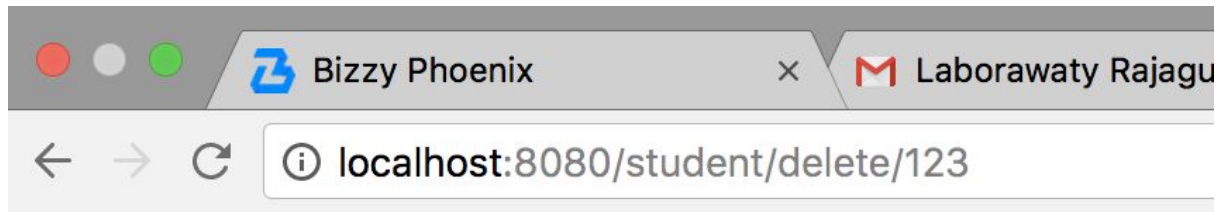
NPM = 124

Name = Chanek Jr.

GPA = 3.4

[Delete Data](#)

Ketika melakukan delete untuk kedua kalinya (Ketikkan url : delete/123), maka akan ditampilkan seperti gambar dibawah ini. Hal ini terjadi karena kita memanggil method delete pada object yang tidak ada di database (NPM tidak ditemukan).



Student not found

NPM = 123

Latihan Menambahkan Update

Berikut langkah langkah untuk menambahkan fitur update.
Pada studentMapper tambahkan import annotation yang dibutuhkan.

```
import org.apache.ibatis.annotations.Update;
```

Lalu tambahkan method updateStudent pada studentMapper. Method ini berupa query untuk melakukan update data student pada database.

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

Pada studentService tambahkan deklarasi method update.

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

Pada studentServiceDatabase tambahkan method yang mengimplementasikan method updateStudent diatas.

```
@Override  
public void updateStudent (StudentModel student)  
{
```

```
log.info("student " + student.getNpm() + " updated");
studentMapper.updateStudent(student);
}
```

Pada viewall.html, tambahkan link yang akan terhubung pada method updateStudent.

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

Berikut code pada form-update.html

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem Editor</h1>

    <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>

</body>

</html>
```

Berikut code pada file success-update.html.

```

<html>
  <head>
    <title>Update</title>
  </head>
  <body>
    <h2>Data berhasil diubah</h2>
  </body>
</html>

```

Kemudian tambahkan method update pada class StudentController. Method ini akan mencari object sesuai npm yang telah diberikan pada parameter. Kemudian akan ditampilkan view update data.

```

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

```

Untuk melengkapi method update, ditambahkan Method updateSubmit pada StudentController. Method ini akan mengirimkan data sebagai parameter. Data tersebut akan di set pada object student berdasarkan npmnya. Dalam method ini digunakan setter yang telah diimplementasikan pada StudentModel. Setelah perubahan data diset atau disimpan maka akan ditampilkan view success-update.

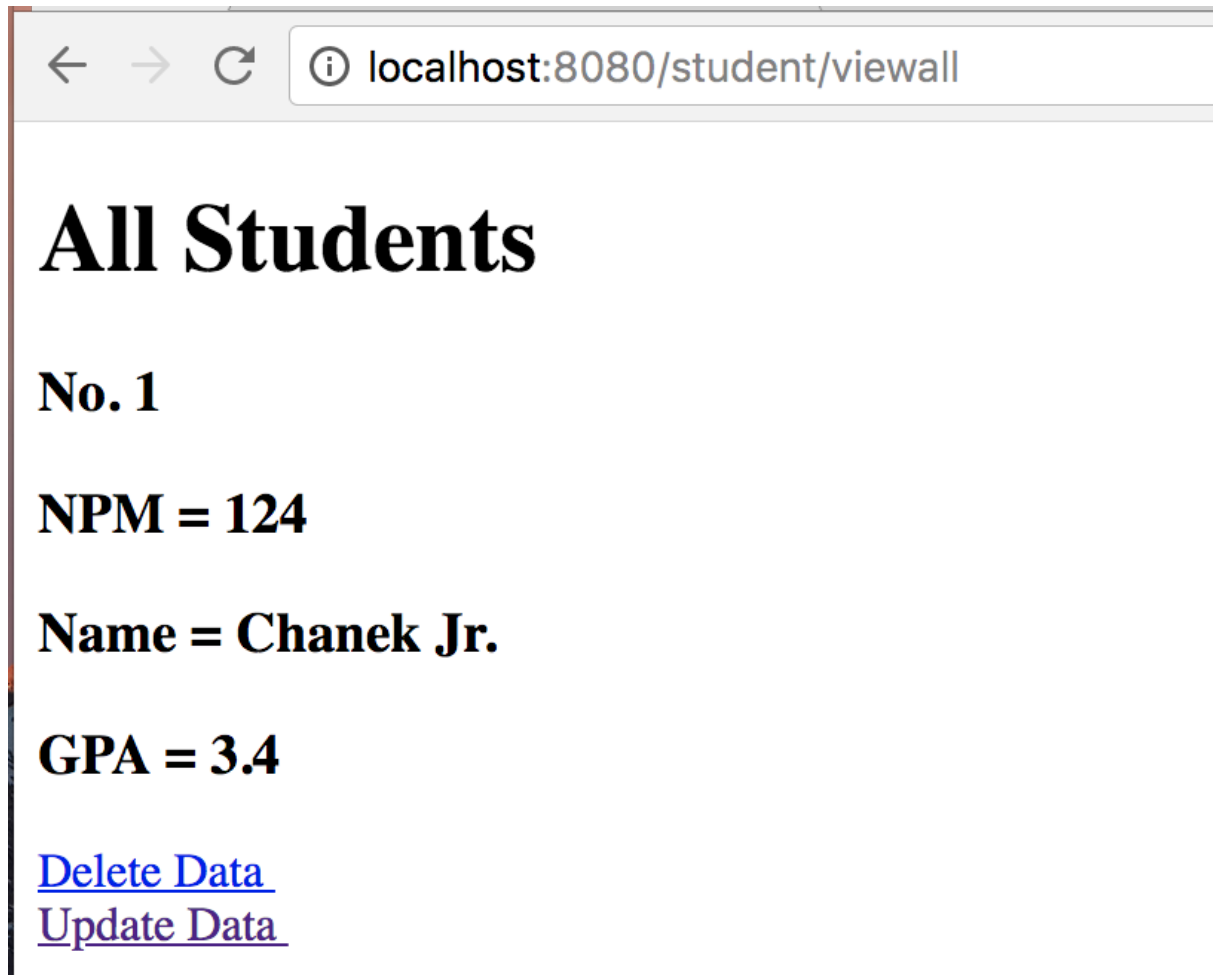
```

@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 = studentDAO.selectStudent (npm);
    student.setNpm(npm);
    student.setName(name);
}

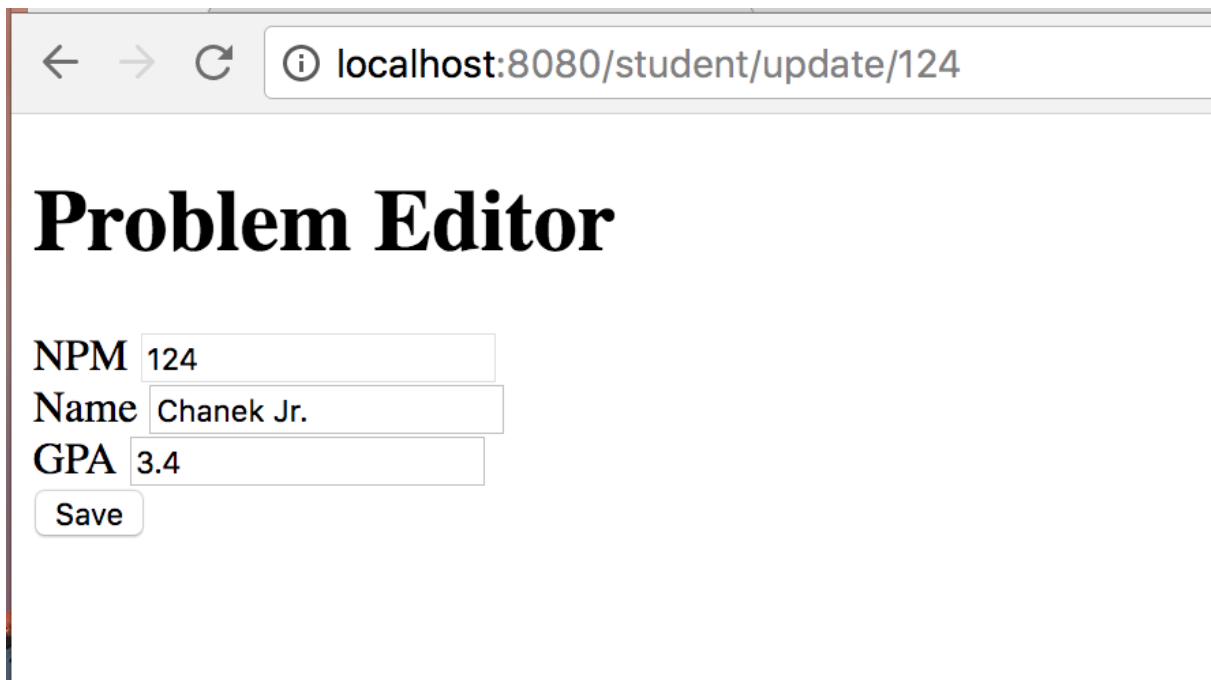
```

```
student.setGpa(gpa);  
studentDAO.updateStudent(student);  
return "success-update";  
}
```

Berikut hasil dari pemanggilan method update.



Klik pada link "Update Data" dan akan ditampilkan form update data.



← → ↻ ⓘ localhost:8080/student/update/124

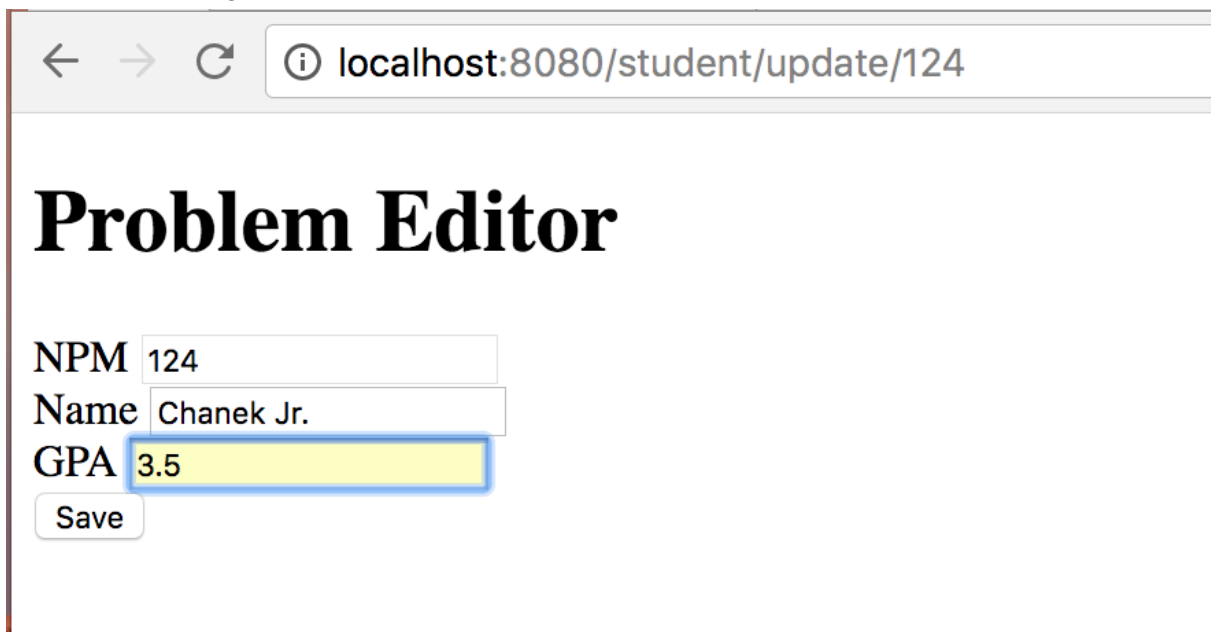
Problem Editor

NPM

Name

GPA

Lalu lakukan perubahan data.



← → ↻ ⓘ localhost:8080/student/update/124

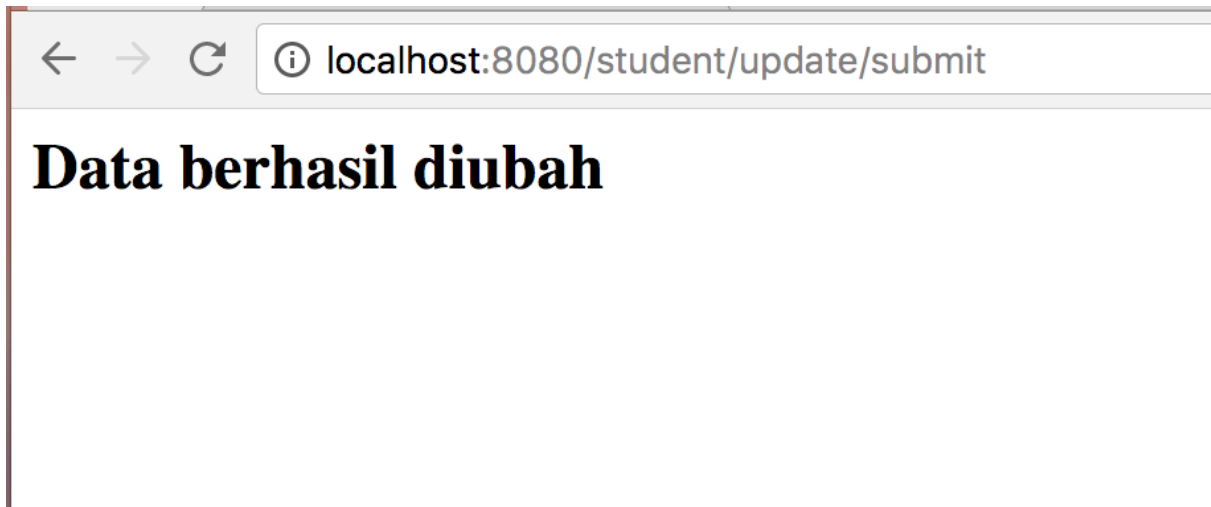
Problem Editor

NPM

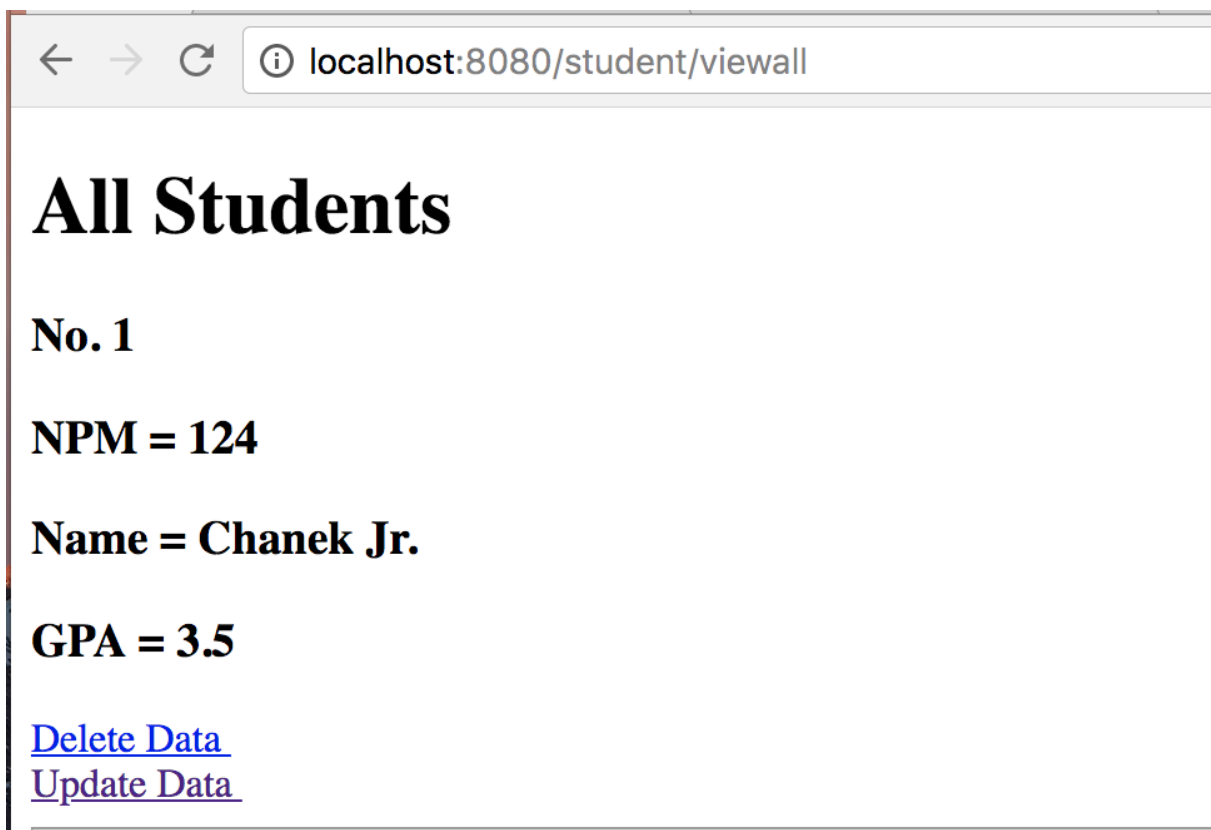
Name

GPA

Setelah data diubah, klik tombol “save”. Kemudian akan dipanggil method `updateSubmit`. Data akan diubah pada database kemudian ditampilkan pesan sukses seperti gambar dibawah ini.



Data student akan berubah sesuai dengan data terbaru.



Latihan Menggunakan object sebagai parameter

Untuk menggunakan object dalam method update maka dilakukan perubahan sebagai berikut.

Berikut code pada file form-update.

```

<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem 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">Save</button>
        </div>
    </form>

</body>

</html>

```

Pada Controller parameter hanya menggunakan StudentModel saja seperti dibawah ini.

```

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

```

Berikut hasil dari perubahan method update (update berjalan sama seperti sebelumnya)

← → ↻ ⓘ localhost:8080/student/viewall

All Students

No. 1

NPM = 124

Name = Chanek Jr.

GPA = 3.5

[Delete Data](#)

[Update Data](#)

Klik pada link “Update Data” dan akan ditampilkan form update data.

← → ↻ ⓘ localhost:8080/student/update/124

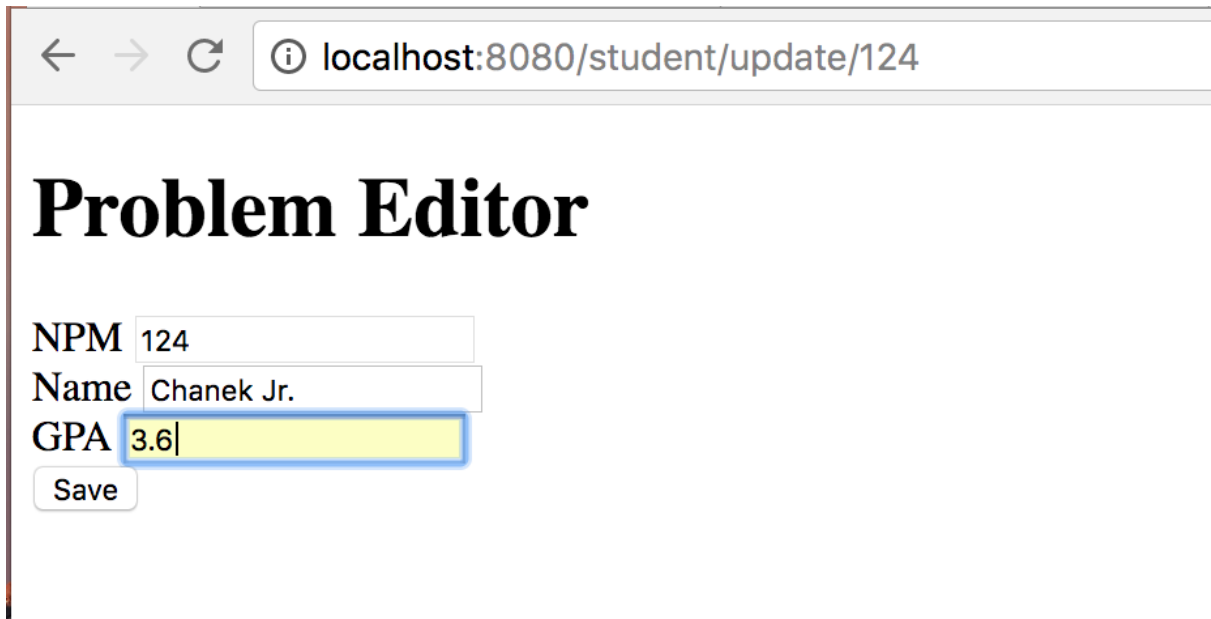
Problem Editor

NPM

Name

GPA

Lalu lakukan pengubahan data.



← → ↻ ⓘ localhost:8080/student/update/124

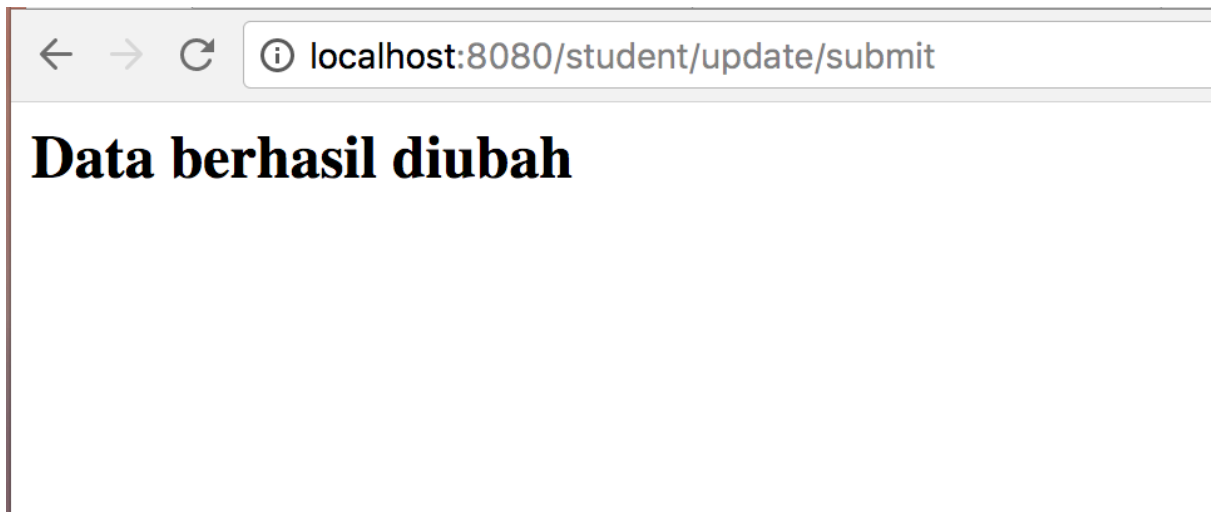
Problem Editor

NPM

Name

GPA

Klik tombol save, dan data akan berhasil diubah.



← → ↻ ⓘ localhost:8080/student/update/submit

Data berhasil diubah

Data sudah berubah.



localhost:8080/student/viewall

All Students

No. 1

NPM = 124

Name = Chanek Jr.

GPA = 3.6

[Delete Data](#)

[Update Data](#)

Menjawab Pertanyaan

1. Jika menggunakan Object sebagai parameter pada form POST maka validasi dilakukan pada halaman view (Thymeleaf). Dari view akan dikirimkan message error atau tidak, dan pada controller message tersebut akan ditangkap dan diimplementasikan action sesuai dengan message yang dikirim.
2. Menggunakan POST method dibanding GET method karena :
 - a. Metode POST lebih aman dibandingkan menggunakan metode GET karena data yang dikirim tidak terlihat (dikirim melalui body), serta parameter yang dikirim tidak disimpan pada history browser/log browser.
 - b. Dapat mengirim data dalam jumlah besar.

Penanganan berbeda diperlukan pada header method di controller seperti pada contoh dibawah ini (method post)

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

3. Satu method **tidak** mungkin untuk menerima lebih dari satu jenis request method.