Shinta Kusuma Wardhani
1606955044
Tutorial4

MENGGUNAKAN DATABASE DAN MELAKUKAN DEBUGGING DALAM PROJECT SPRING BOOT

A. RINGKASAN MATERI

Pada tutorial4 ini mempelajari yaitu menghubungkan project dengan database MySQL dengan menggunakan library MyBatis dan Lombok. Fungi library Lombok adalah sebagai helper annotation pada project. MyBatis berfungsi melakukan koneksi dan gemerate query dengan helper annotation. Dalam tutorial ini juga mempelajari memberikan argument log pada project yang dibuat. Log tersebut berguna untuk melakukan debuggingpada Spring Boot.

B. TUTORIAL

LATIHAN MENAMBAHKAN DELETE (Implementasi pada Database)

- Pada viewall.html tambahkan link text yang akan digunakan untuk menghapus data.

```
<div th:each="student,iterationStatus: ${students}">
        <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
        <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
        <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
        <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
        <a th:href="'/student/delete/' + ${student.npm}">Delete data</a><br/>
```

- Penambahan method pada StudentMapper (method deleteStudent)

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

- Lengkapi method deleteStudent pada StudentServiceDatabase, dan panggil method pada studentMapper.

StudentServiceDatabase.java

```
@Override
public void deleteStudent (String npm)
{

log.info("student" + npm + "deleted");
studentMapper.delStudent(npm);

}
```

Pada StudentServiceDatabase terdapat log.info yang digunakan untuk memberikan argument ke log dengan menggunakan formatted String. Selanjutnya

studentMapper.java

```
void delStudent(@Param("npm")String npm);
```

- Lengkapi method delete pada StudentController.java untuk penambahan validasi jika npm mahasiswa tidak ditemukan dan ditambahkan untuk memunculkan tampilan view delete

studentController.java

Shinta Kusuma Wardhani
1606955044
Tutorial4

```
@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";
    }
}
```

Dalam kode tersebut menjelaskan jika npm ditemukan maka akan mengarahkan pada halaman view delete namun jika data npm tidak ditemukan akan mengarahkan pada halaman not-found.html.

view delete.html

Digunakan untuk menampilkan jika data npm berhasil dihapus.

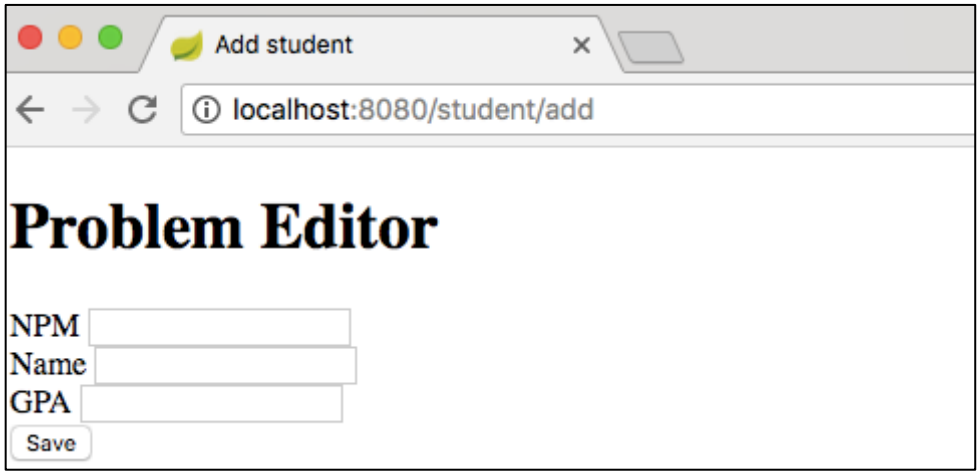
```
<html>
    <head>
        <title>Delete</title>
    </head>
    <body>
        <h2>Data berhasil dihapus</h2>
    </body>
</html>
```

Jika npm gagal ditemukan akan mengarahkan pada halaman not-found.html

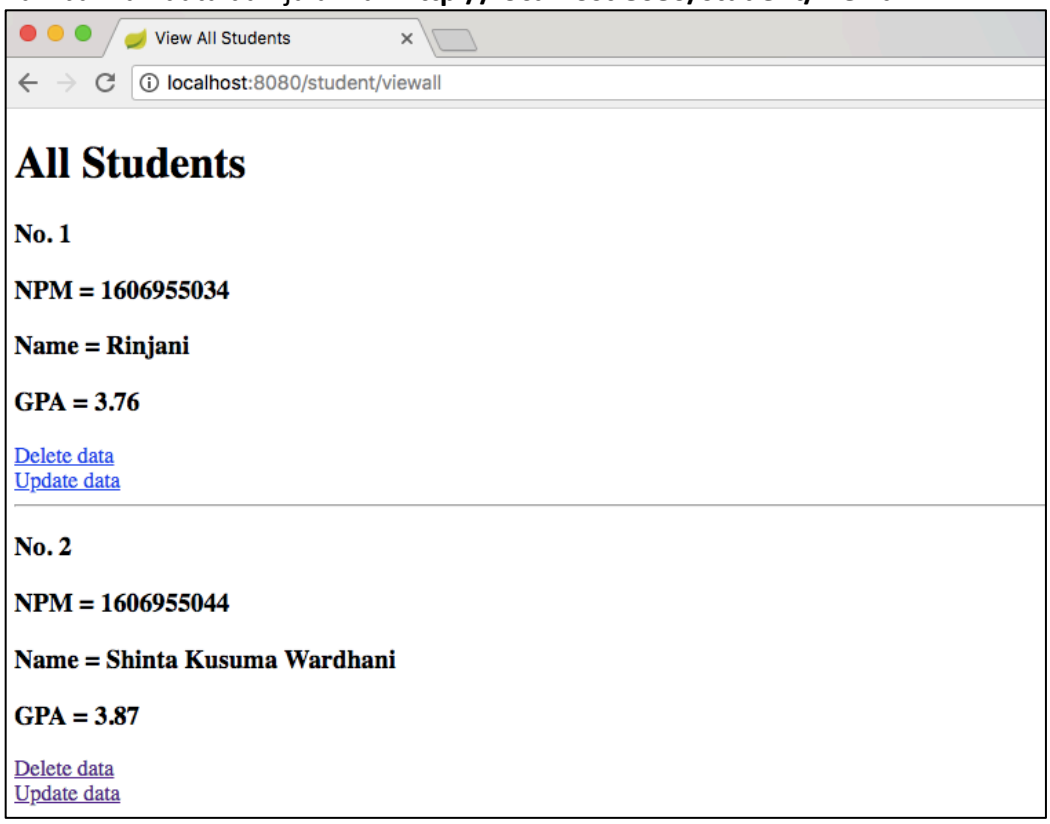
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Student not found</title>
    </head>
    <body>
    <h1>Student not found</h1>
        <h3 th:text="'NPM = ' + ${npm}">Student NPM</h3>
    </body>
</html>
```

- Menjalankan Spring boot app dan lakukan insert.
  Jalankan **localhost:8080/student/add** dan tambahkan data student.

Shinta Kusuma Wardhani
1606955044
Tutorial4

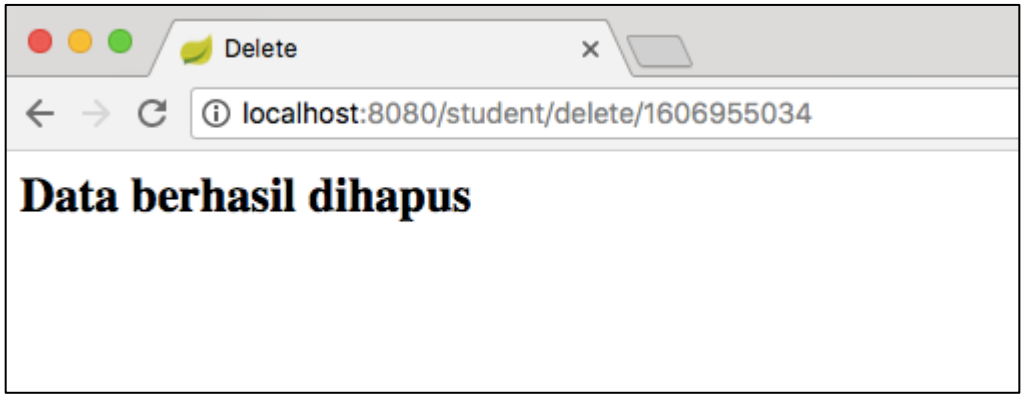


Tambahkan data dan jalankan **http://localhost:8080/student/viewall**

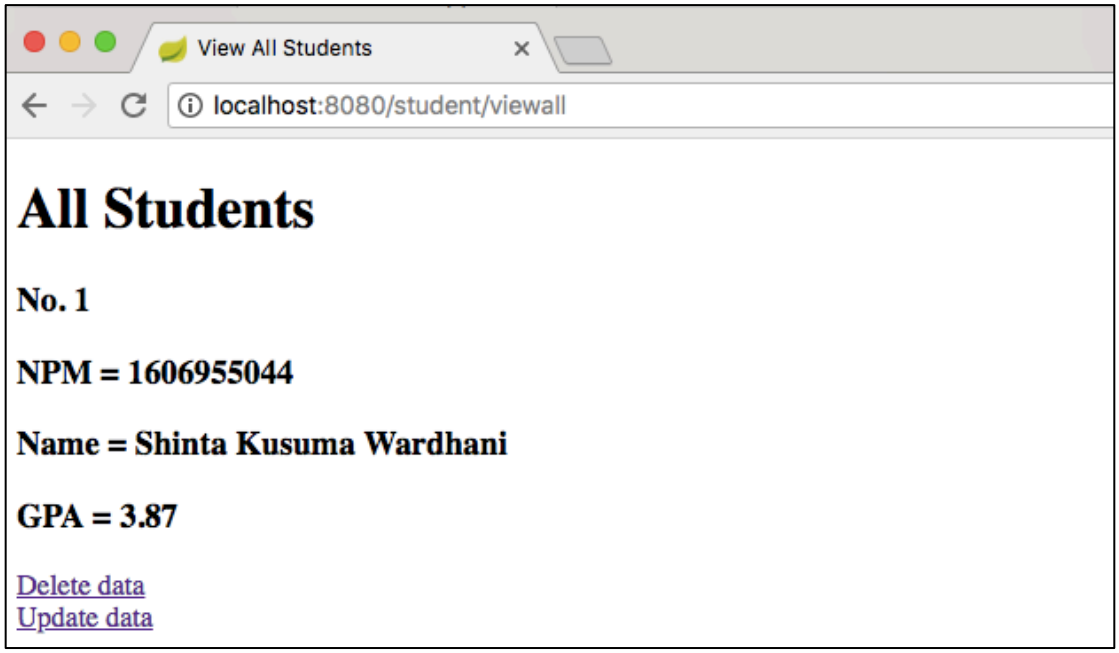


Pilih “Delete Data” maka akan muncul tampilan seperti pada gambar berikut.

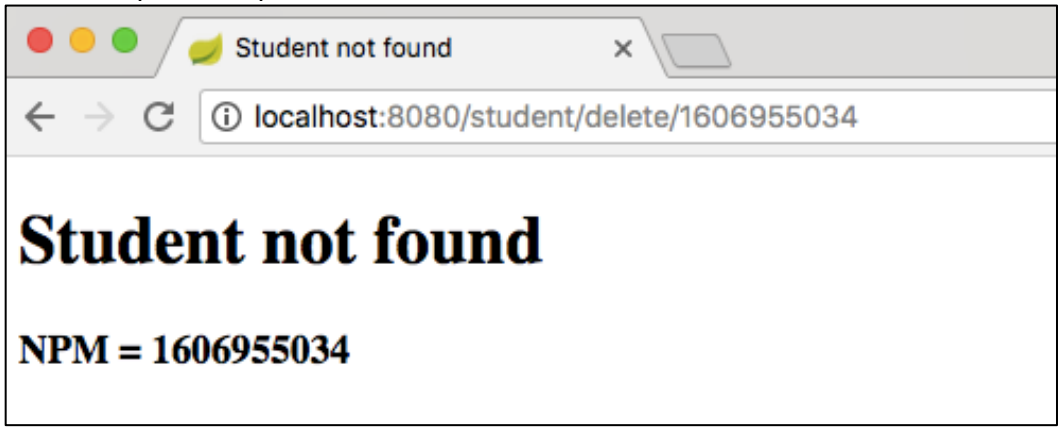
Shinta Kusuma Wardhani
1606955044
Tutorial4



Jalankan **localhost:8080/student/viewall** maka data yang telah dihapus tidak dapat muncul kembali.



Jika kita akan menghapus data yang telah dihapus (npm tidak ditemukan) maka akan muncul seperti tampilan berikut



Shinta Kusuma Wardhani
1606955044
Tutorial4

LATIHAN MENAMBAHKAN UPDATE

- Tambahkan method updateStudent pada class StudentMapper

studentMapper

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

Ditambahkan method untuk update berdasarkan npm yang dipilih kemudian akan disimpan pada StudentModel.

- Pada interface StudentService tambahkan method updateStudent

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

- Pada class StudentServiceDatabase tambahkan implementasi method updateStudent

```
@Override
public void updateStudent(StudentModel student) {
log.info("student {} name updated to {}", student.getNpm(), student.getName() +"updated");
studentMapper.updateStudent(student);

}
```

- Pada viewall.html tambahkan link untuk Update Data

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

    <div th:each="student,iterationStatus: ${students}">
            <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
            <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
            <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
            <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
            <a th:href="'/student/delete/' + ${student.npm}">Delete data</a><br/>
            <a th:href="'/student/update/' + ${student.npm}">Update data</a><br/>

            <hr/>
        </div>
    </body>
</html>
```

- Buat form-update.html dengan copy form-update.html

Shinta Kusuma Wardhani
1606955044
Tutorial4

```
<!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="${student.n
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:value="${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">Update</button>
        </div>
    </form>
</body>
</html>
```

Dengan mengubah info- info yang diperlukan seperti title, page-header dan ubah action form menjadi "/student/update/submit" dan method menjadi POST.

- Copy view success-add dan rename menjadi success-update dan ubah info- info yang diperlukan.

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

```

- Tambahkan method update pada class StudentController.

studentController

```
@RequestMapping("/student/update/{npm}")
public String updateSt (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 tersebut memiliki validasi sama halnya dengan method delete jika npm tidak ditemukan maka akan mengarahkan pada halaman not-found. Jika terdapat npm yang ditemukan maka akan diarahkan ke view form-update.

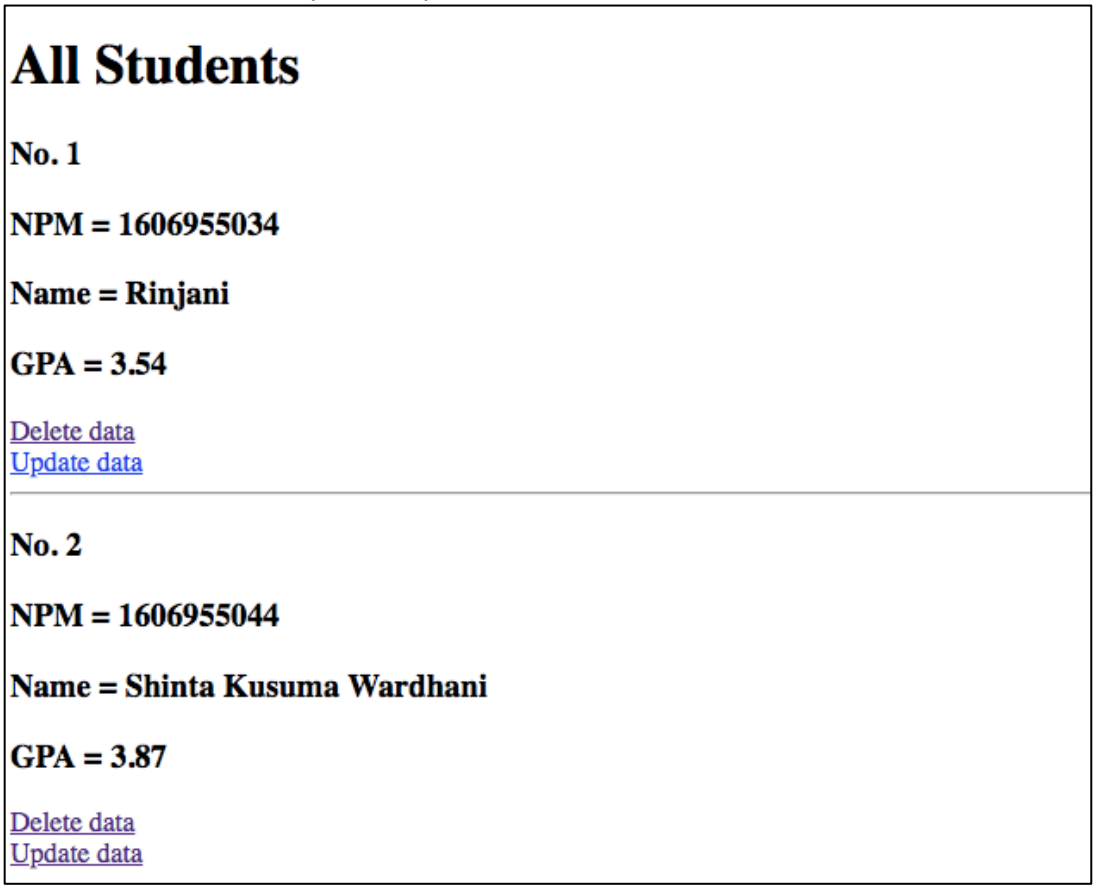
- Tambahkan method updateSubmit pada StudentController.

```
@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 tersebut berfungsi untuk menerima data NPM, nama dan GPA yang akan dilakukan pada halaman form-update. Selanjutnya akan dipanggil method updateStudent yang mempunyai object student, jika update berhasil akan mengembalikan pada view success-update untuk menandakan bahwa update data student berhasil disimpan.

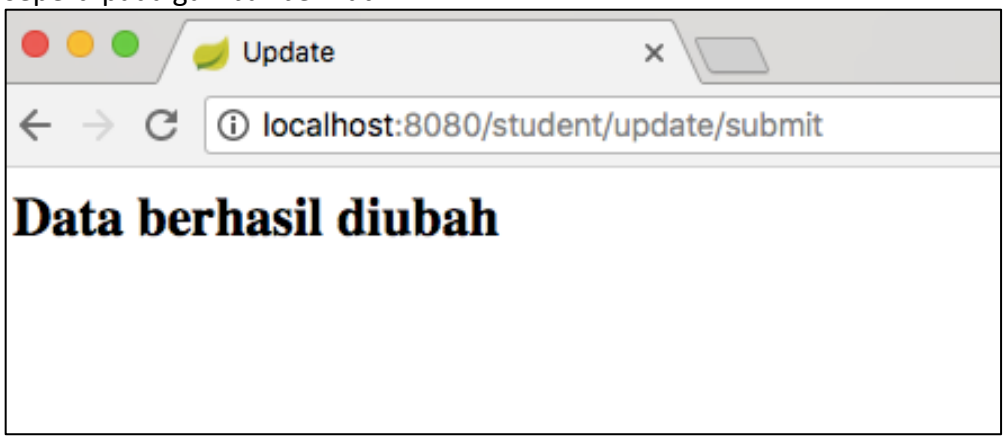
- Jalankan localhost:8080/student/viewall untuk melihat data student



Pilih Update data maka akan muncul tampilan seperti berikut

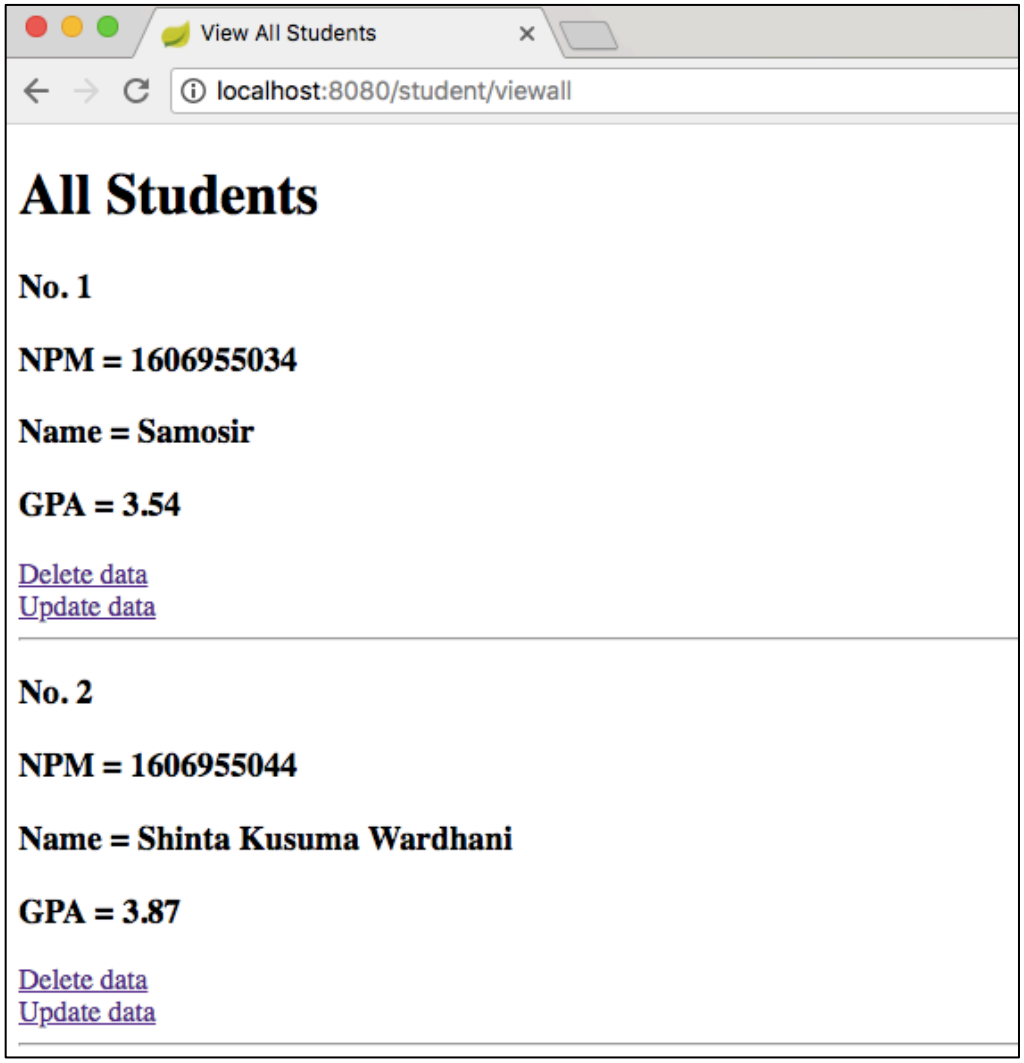
Shinta Kusuma Wardhani
1606955044
Tutorial4

Setelah diubah pada parameter nama dan pilih Update maka akan muncul tampilan seperti pada gambar berikut



Jalankan **localhost:8080/student/update/viewall** maka akan muncul tampilan seperti berikut

Shinta Kusuma Wardhani
1606955044
Tutorial4



Data mahasiswa yang bernama Rinjani telah berubah menjadi Samosir. Hal tersebut menandakan bahwa update berhasil dilakukan.

LATIHAN MENGGUNAKAN OBJECT SEBAGAI PARAMETER

- Pada studentController tambahkan method updateSubmit

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

- Pada form-update.html tambahkan seperti pada gambar berikut

Shinta Kusuma Wardhani
1606955044
Tutorial4

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

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

th:object digunakan untuk mendeklarasikan sebuah object yang akan mengumpulkan data mahasiswa. Sedangkan th:field digunakan untuk mengekspresikan field seperti npm, nama, gpa sesuai dengan object student.

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?

   Jawab :

   Untuk melakukan validasi digunakan model yang sebelumnya sudah dibuat. Model merupakan tampilan struktur pada sebuah tabel, pada model tersebut dapat ditambahkan standar validation seperti @valid pada parameter object dan penambahan parameter Object bindingResult. Kemudian pada body tambahkan kondisi untuk validasi dengan menggunakan code “bindingResult.hasErrors()” jika hasil validasi error maka akan mengarahkan kembali ke halaman form dan jika proses validasi berhasil akan mengarahkan pada halaman selanjutnya. Validasi diperlukan karena form tersebut nantinya akan menyimpan data dalam database agar tidak terjadi kesalahan input.

2. Menurut anda, mengapa form submit biasanya menggunakan POST method disbanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form post dikirim menggunakan method berbeda?

   Jawab:

   POST method digunakan karena lebih aman dibandingkan dengan GET method, karena data yang dikirimkan tidak terlihat method ini sering dapat digunakan pada data yang memiliki tingkat sensifitas tinggi (password). Selain itu method POST tidak membatasi panjang karakter dari data yang diinputkan, namun jika menggunakan GET hanya dapat menampung maksimal 2047 data. Perlu penanganan yang berbeda pada header method di controller dengan menambahkan RequestMethod.POST pada RequestMapping

Shinta Kusuma Wardhani
1606955044
Tutorial4

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

   Jawab:

   Bisa, namun tidak bisa dijalankan untuk pemanggilannya harus satu persatu.