Hafizha Zuharah Az Zahra
1506689673
APAP - A

TUTORIAL 4

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Latihan Menambahkan Delete

1. Implementasi method delete menggunakan database.
2. Pada viewall.html tambahkan:

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

   Di dalam div dengan **th:each**

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**
   a. Tambahkan method delete student yang menerima parameter NPM.

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

   b. Tambahkan annotation delete di atas dan SQL untuk menghapus.

```
@Delete("DELETE FROM student WHERE npm = #{npm}")
```

4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**

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

   a. Tambahkan log untuk method tersebut dengan cara menambahkan

```
log.info("student " + npm + " deleted");
```

   b. Panggil method delete student yang ada di StudentMapper

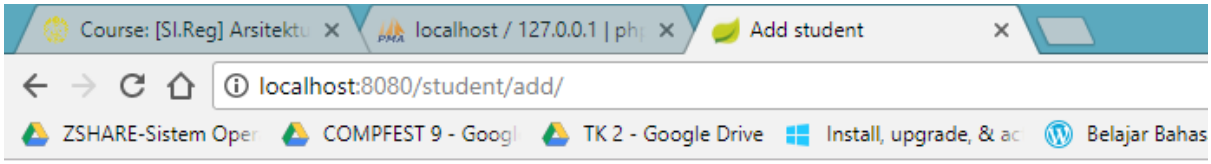
```
studentMapper.deleteStudent(npm);
```

5. Lengkapi method **delete** pada class **StudentController**
   a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found
   b. Jika berhasil delete student dan tampilkan view delete

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

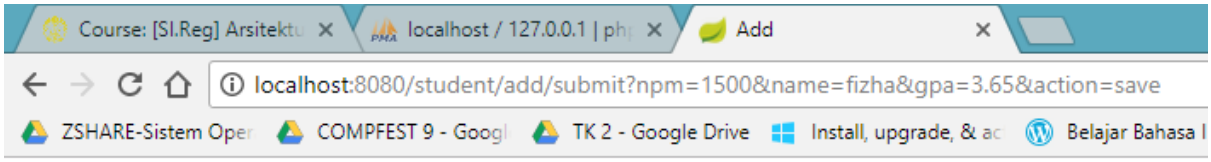
6. Jalankan Spring Boot app dan lakukan beberapa insert
   insert npm 1500, name Fizha, gpa 3.65

Hafizha Zuharah Az Zahra
1506689673
APAP - A



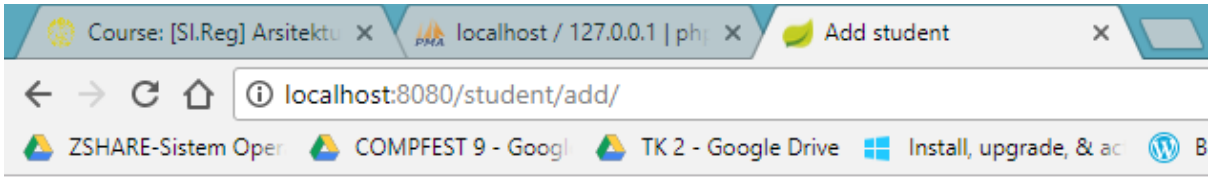
Problem Editor

NPM 1500
Name fizha
GPA 3.65
Save



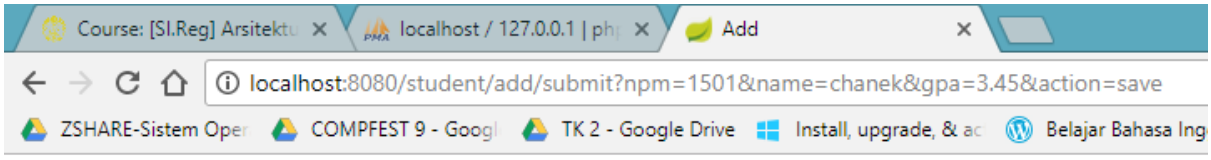
Data berhasil ditambahkan

insert npm 1501, name chanek, gpa 3.45



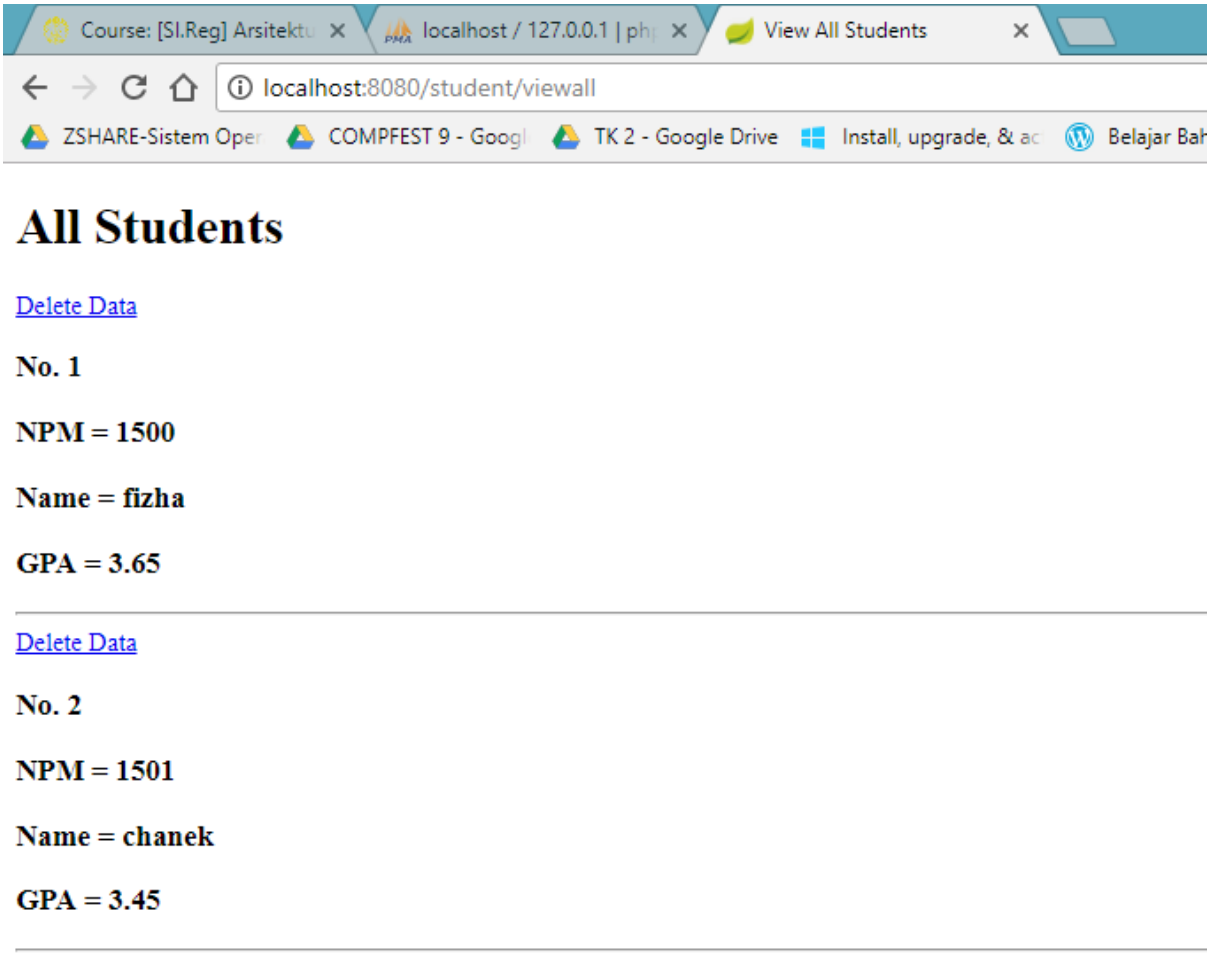
Problem Editor

NPM 1501
Name chanek
GPA 3.45
Save

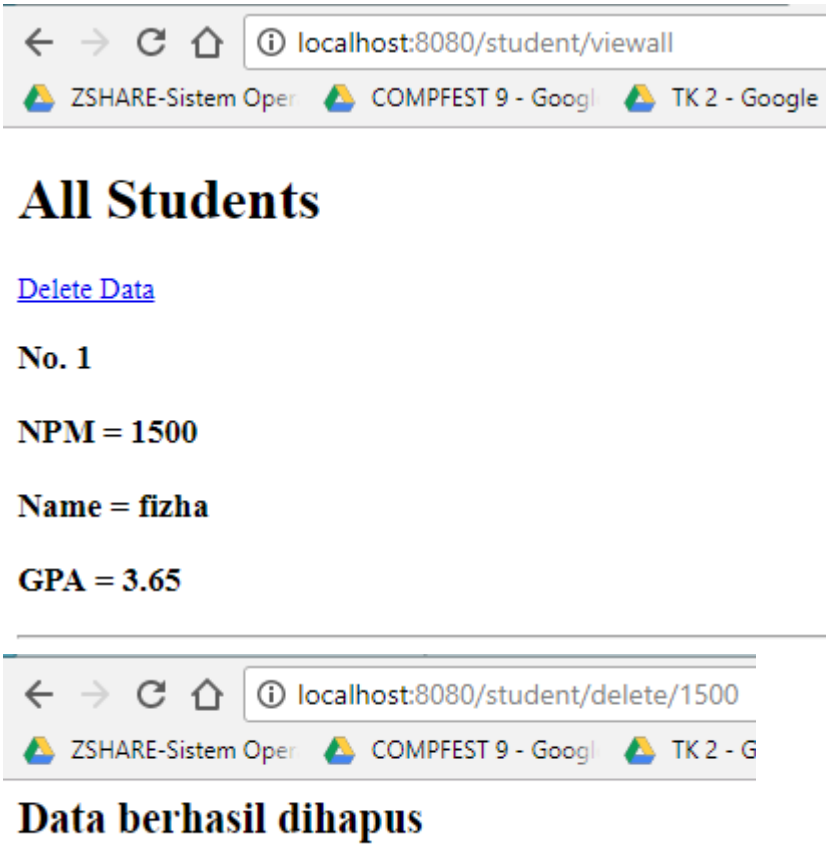


Data berhasil ditambahkan

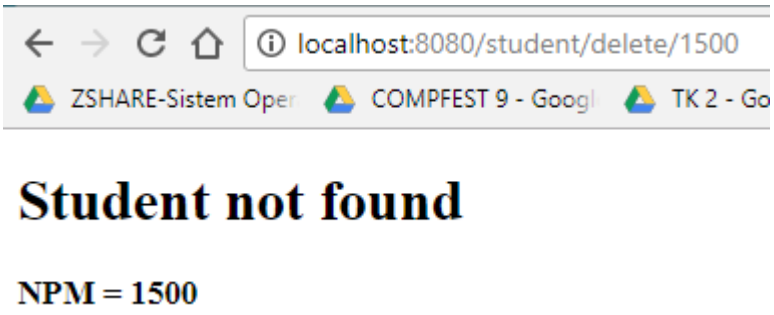
7. Contoh tampilan View All



8. Contoh tampilan delete NPM 1500



9. Contoh tampilan jika dilakukan delete NPM 1500 yang kedua kalinya



Latihan Menambahkan Update

Pada update kali ini, kita akan mencoba menggunakan method POST.

1. Tambahkan method **updateStudent** pada class **StudentMapper**
   a. Parameternya adalah StudentModel student
   ```
   void updateStudent (StudentModel student);
   ```
   b. Annotationnya adalah @Update
   ```
   @Update(
   ```
   c. Lengkapi SQL *update*-nya
   ```
   @Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}")
   ```

2. Tambahkan method **updateStudent** pada interface **StudentService**

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

3. Tambahkan implementasi method **updateStudent** pada class **StudentServiceDatabase**.

```
@Override
public void updateStudent (StudentModel student)
{
    log.info("student " + student.getNpm() + "'s information has been updated");
    studentMapper.updateStudent(student);
}
```

4. Tambahkan link Update Data pada **viewall.html**

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

5. Copy view form-add.html menjadi **form-update.html**
   a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.

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

   b. Ubah action form menjadi /student/update/submit

```
<form action="/student/update/submit"
```

   c. Ubah method menjadi post

```
method="post"
```

   d. Untuk input npm ubah menjadi

```
<div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" />
</div>
```

   readonly agar npm tidak dapat diubah, **th:value** digunkan untuk mengisi input dengan npm student yang sudah ada.
   Input name ubah menjadi

```
<label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" />
```

   Input gpa ubah menjadi

```
<label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" />
```

6. Copy view success-add.html menjadi **success-update.html**.
   a. Ubah keterangan seperlunya

```
<html>
    <head>
        <title>Updated</title>
    </head>
    <body>
        <h2>Data berhasil diperbaharui</h2>
    </body>
</html>
```

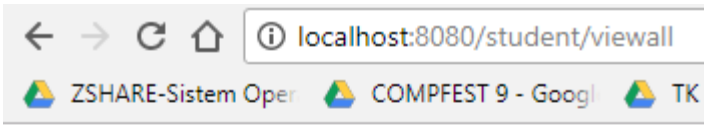
7. Tambahkan method **update** pada class **StudentController**
   a. Request mapping ke /student/update/{npm}
   b. Lakukan validasi

c. Jika student dengan npm tidak ada, tampilkan view not-found. Jika ada tampilkan view form-update
*validasi dilakukan dengan select student di database. jika npm tidak bernilai dan student tidak ada, maka akan return not-found. Jika ada, maka add attribute student ke model agar nilai attribute student bisa diambil di form-update.

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value="npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);

    if (npm != null && student != null) {
        model.addAttribute("student", student);
        return "form-update";
    } else {
        return "not-found";
    }

}
```

8. Tambahkan method **updateSubmit** pada class **StudentController**
   a. Karena menggunakan post method maka request mappingnya adalah sebagai berikut.
   ```
   @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
   ```
   b. Header methodnya adalah sebagai berikut.
   *mengambil semua nilai yang dipost di form-update (npm, name, gpa)
   ```
   public String updateSubmit (
           @RequestParam(value = "npm", required = false) String npm,
           @RequestParam(value = "name", required = false) String name,
           @RequestParam(value = "gpa", required = false) double gpa)
   {
   ```
   c. Lengkapi method dengan memanggil method update Student dan kembalikan view success-update
   *select student dengan npm yang ada (menandakan npm student yg diupdate) lalu ubah nilai yang berubah.
   ```
   StudentModel student = studentDAO.selectStudent(npm);
   student.setName(name);
   student.setGpa(gpa);
   studentDAO.updateStudent(student);

   return "success-update";
   ```
9. Jalankan Spring Boot dan coba test program
   view all students

Hafizha Zuharah Az Zahra
1506689673
APAP - A



All Students

No. 1

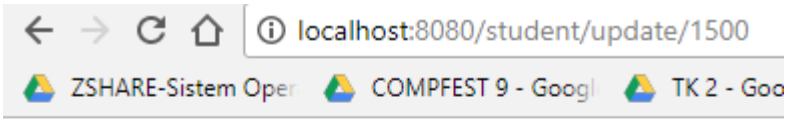
NPM = 1500

Name = fizha

GPA = 3.88

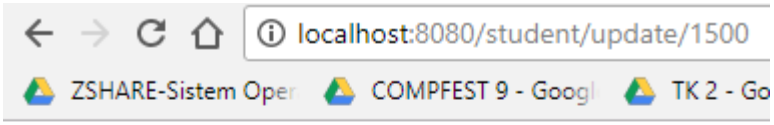
Delete Data
Update Data

update name = fizhasi, gpa = 3.98



Problem Editor

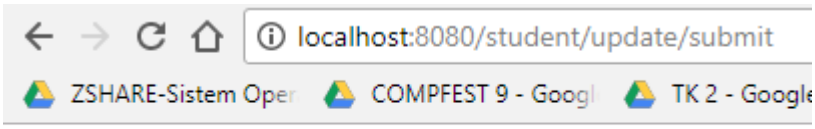
NPM 1500
Name fizha
GPA 3.88
Save



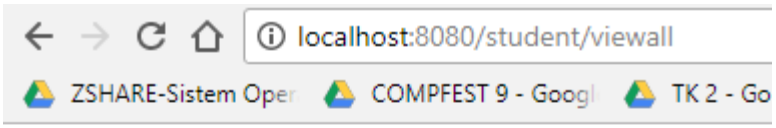
Problem Editor

NPM 1500
Name fizhasi
GPA 3.98
Save

Student updated



Data berhasil diperbaharui



All Students

No. 1

NPM = 1500

Name = fizhasi

GPA = 3.98

Delete Data
Update Data

Latihan Menggunakan Object Sebagai Parameter

1. Pada tutorial kali ini kita tidak menggunakan RequestParam.
2. SpringBoot dan Thymeleaf memungkinkan agar method **updateSubmit** menerima parameter berupa model **StudentModel.**
3. Ubah form dalam form-update.html menjadi seperti berikut ini.
   *karena tidak memakai requestParam, maka harus define objek student terlebih dahulu. Lalu ambil nilai yang ada di field memakai th:field. Saat dipost, niai attribute model student difield akan tersimpan.

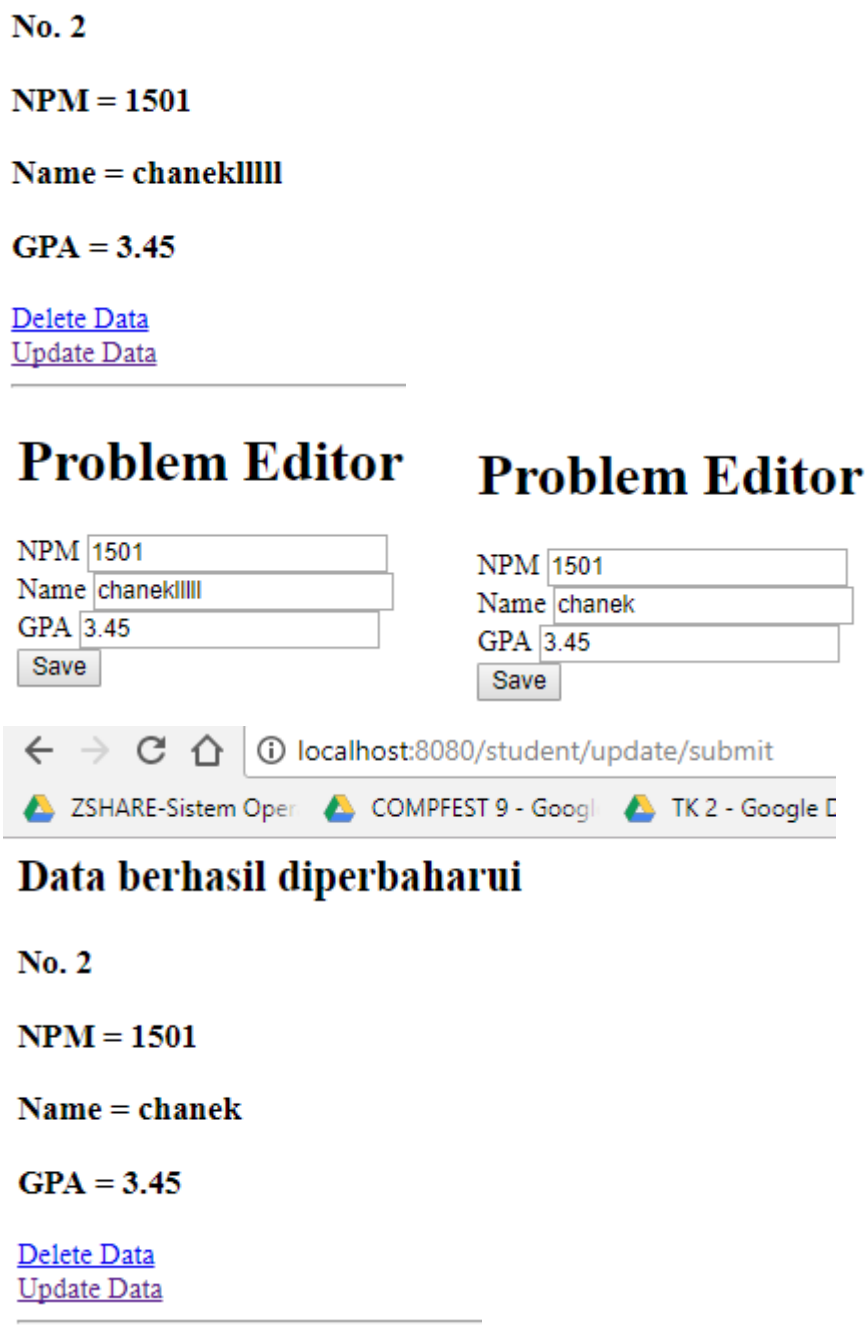
```
<form action="/student/update/submit" method="post" th:object="${student}">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:field="${student.npm}" />
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:field="${student.name}"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

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

4. Ubah method **updateSubmit**pada StudentController yang hanya menerima parameter berupa StudentModel
   *nilai objek student akan terambil semua melalui anotasi model attribute. Lalu, panggil

method update untuk mengambil nilai yang ada di model student.

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

    return "success-update";
}
```

5. Cek aplikasi



Hafizha Zuharah Az Zahra
1506689673
APAP - A

Jawaban pertanyaan

1. Kita bisa melakukan validasi input salah satunya dengan menggunakan anotasi @NotNull untuk mencegah field yang kosong saat diambil nilainya.

2. Biasanya method POST digunakan untuk memudahkan input nilai jika banyak parameter yang dibutuhkan dan juga dengan method POST akan lebih aman karena data tidak dapat terlihat pada address bar. Ya, dengan menggunakan method POST, diperlukan request method POST pada header controller.

3. Iya, controller akan memisahkan sendiri data-data yang menggunakan method berbeda.