Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Latihan Menambahkan Delete

Latihan berikut ini yaitu menghapus data student yang ada dalam database dengan memberikan parameter berupa npm. Sehingga data student yang dihapus berdasarkan npm yang diberikan sebagai parameter.

1. Tambahkan method delete student pada class StudentMapper berisi anotasi Delete dan syntax delete pada sql dengan parameter npm. Hasilnya dapat dilihat pada gambar di bawah ini.

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

2. Tambahkan log untuk method tersebut dan pemanggilan fungsi delete student pada class StudentMapper dengan cara menambahkan kode seperti pada gambar di bawah ini.

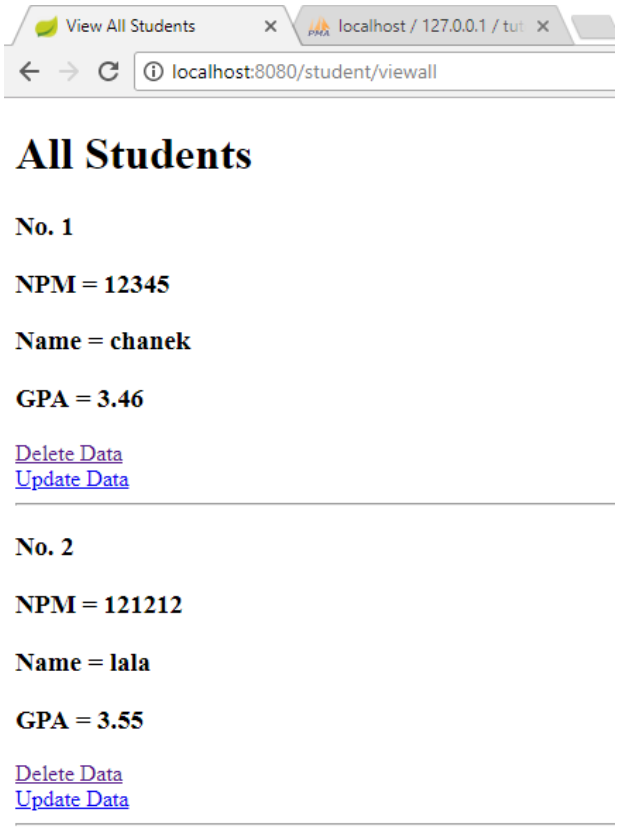
```
@Override
public void deleteStudent (String npm)
{
    log.info("student " + npm + " deleted");
    studentMapper.deleteStudent(npm);
}
```

3. 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
   Kodenya dapat dilihat seperti pada gambar di bawah ini dimana method berisi pemanggilan delete student pada class StudentMapper. Pertama perlu memanggil method selectStudent untuk mengecek apakah student dengan npm tersebut ada atau tidak. Jika ada maka student akan dihapus dan menampilkan view delete berisi notifikasi berhasil dan jika tidak memanggil view not-found yang menyatakan bahwa student tidak ada.

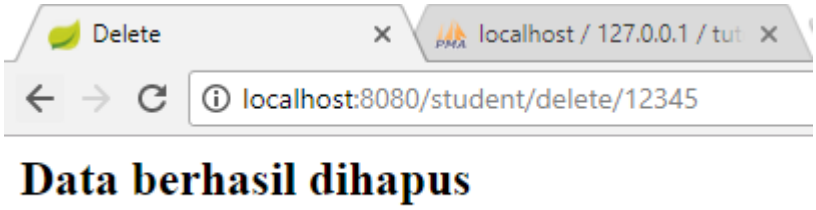
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    studentDAO.deleteStudent (npm);
    StudentModel student = studentDAO.selectStudent (npm);

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

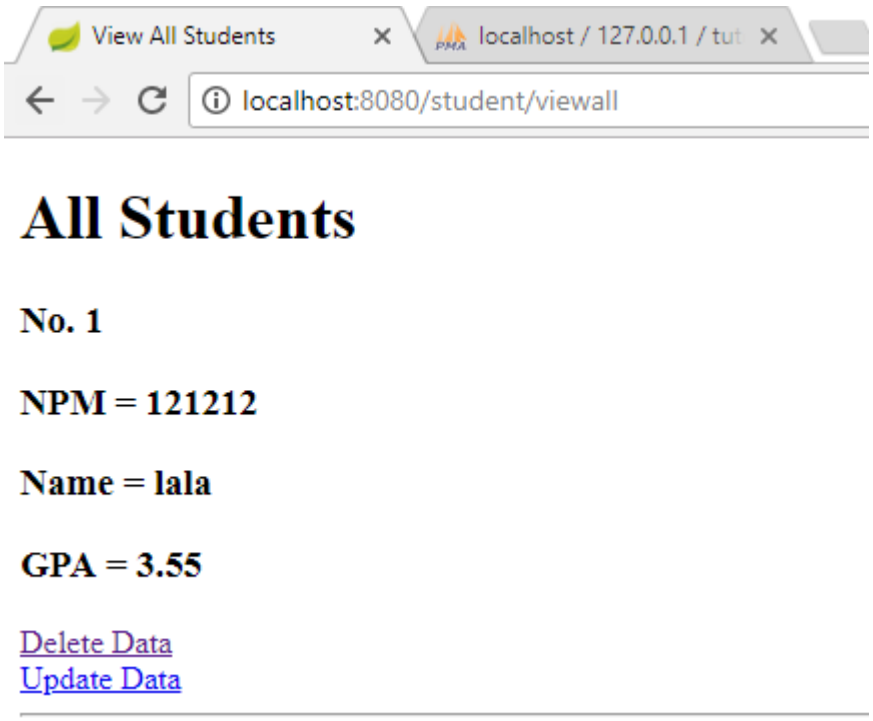
Berikut ini merupakan hasil dari pemanggilan method delete student.

Contoh tampilan view all

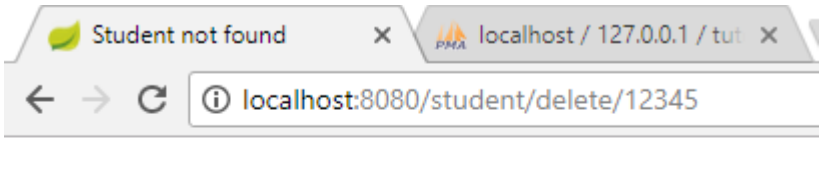


Contoh tampilan delete student dengan npm 12345 yang terdapat pada database sehingga data berhasil dihapus.





Contoh tampilan jika dilakukan delete NPM 12345 yang kedua kalinya dimana npm tersebut datanya sudah dihapus sebelumnya.



Latihan Menambahkan Update

1. Tambahkan method updateStudent pada class StudentMapper
   a. Parameternya adalah StudentModel student
   b. Annotationnya adalah @Update
   c. Lengkapi SQL update-nya Hint: Query SQL untuk update
   Pada class StudentMapper terdapat anotasi Update dan syntax update pada sql dengan parameter npm. Terdapat juga deklarasi method updateStudent dengan parameter object StudentModel. Hasilnya dapat dilihat pada gambar di bawah ini.

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

2. Tambahkan method updateStudent pada interface StudentService dengan parameter object StudentModel. Hasilnya dapat dilihat pada gambar di bawah ini.

```
public interface StudentService
{
    StudentModel selectStudent (String npm);


    List<StudentModel> selectAllStudents ();


    void addStudent (StudentModel student);


    void deleteStudent (String npm);


    void updateStudent(StudentModel student);
}
```

3. Tambahkan implementasi method updateStudent pada class StudentServiceDatabase. Jangan lupa tambahkan logging pada method ini. Method tersebut berisi pemanggilan method yang terdapat pada StudentMapper seperti gambar di bawah ini.

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

4. Tambahkan link Update Data pada viewall.html dengan menambahkan parameter npm pada link href seperti pada gambar di bawah ini.

```
<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>
```

5. Copy view form-add.html menjadi form-update.html menjadi seperti pada gambar di bawah ini.

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

6. Copy view success-add.html menjadi success-update.html untuk notifikasi bahwa proses update berhasil dilakukan seperti gambar di bawah ini.

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

7. Tambahkan method update pada class StudentController yang melakukan pengecekan terlebih dahulu apakah student yang akan di update ada di database atau tidak dengan memanggil method selectStudent. Jika student ada maka form update akan ditampilkan berisi data student dengan npm yang ada di parameter. Jika tidak ada maka akan memanggil view not-found. Kodenya dapat dilihat pada gambar di bawah ini.

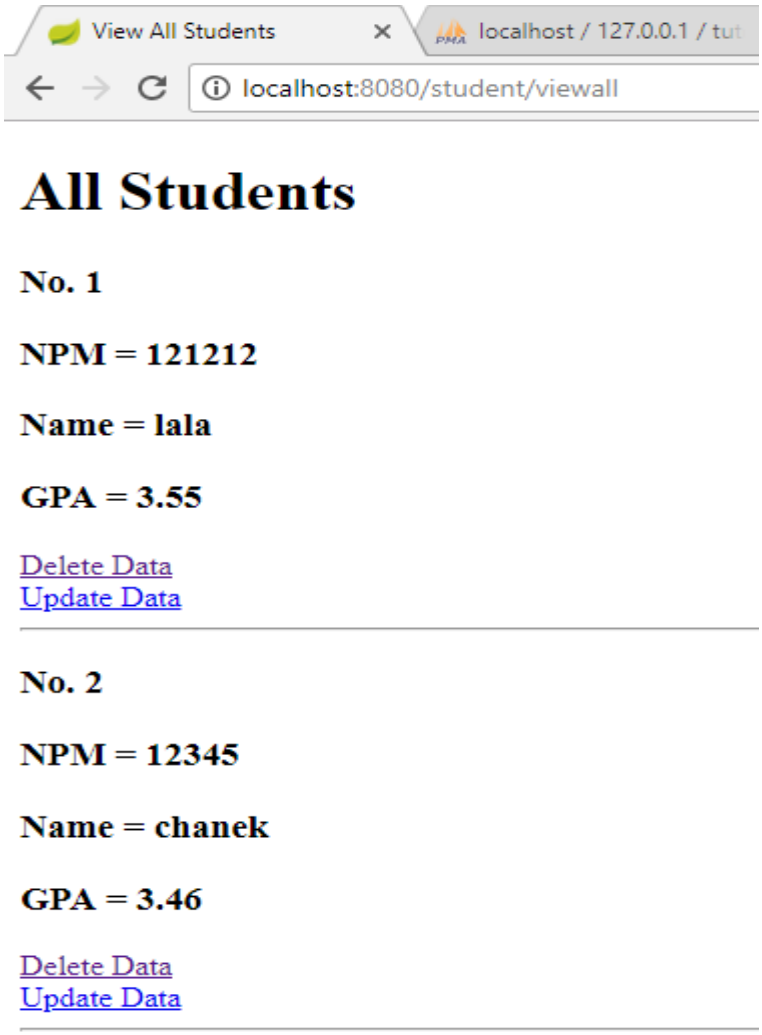
```
@RequestMapping("/student/update/{npm}")
public String update(Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);
    if (student != null) {
        studentDAO.updateStudent(student);
        model.addAttribute ("student", student);
 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

8. Tambahkan method updateSubmit pada class StudentController sebagai method yang menangkap data baru student dan menyimpannya ke database dengan memanggil method updateStudent. Pertama perlu memanggil method selectStudent untuk mengambil object student yang akan diubah datanya. Kemudian nilai student akan diubah dengan methos setter dengan nilai parameter yang diberikan dari form-update. Kemudian memanggil method updateStudent pada studentMapper untuk update data student dan memanggil view success-update seperti pada gambar berikut.

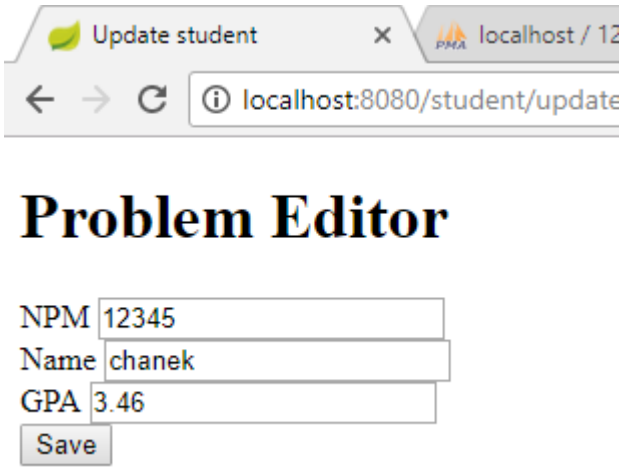
```
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@RequestParam (value="npm", required=false) String npm,
        @RequestParam (value="name", required=false) String name,
        @RequestParam (value="gpa", required=false) double gpa)
{
    StudentModel student = studentDAO.selectStudent(npm);
    student.setNpm(npm);
    student.setName(name);
    student.setGpa(gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Berikut ini merupakan hasil dari pemanggilan method update student.

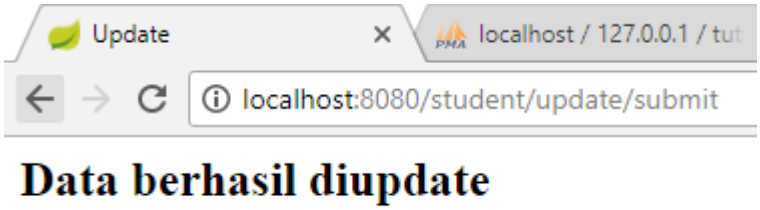
Contoh tampilan view all menampilkan semua data student



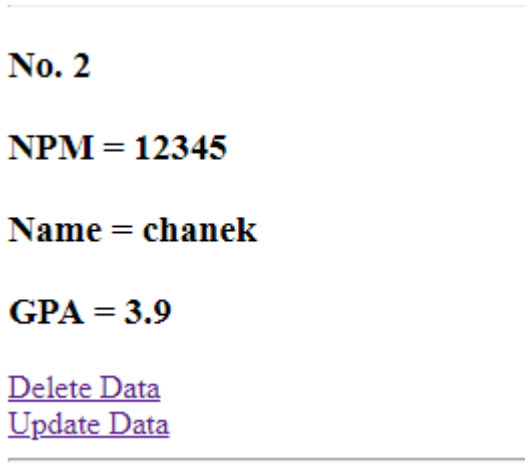
Contoh tampilan update student dengan npm 12345 yang terdapat pada database. View yang dipanggil yaitu form-update.



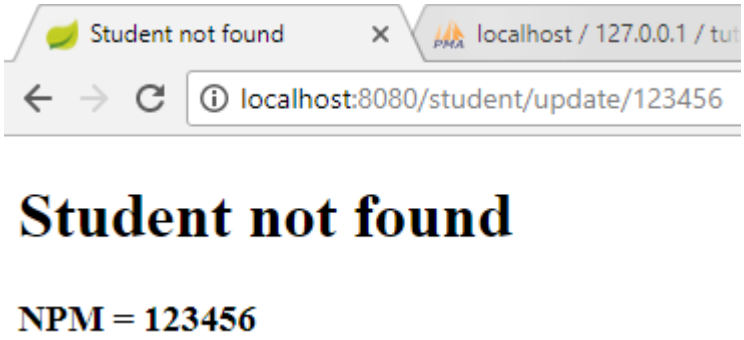
Ketika klik tombol save maka data akan disimpan dan menampilkan form update-success



Contoh tampilan view all setelah data diupdate data telah berubah.



Contoh tampilan update student dengan npm 123456 yang tidak terdapat pada database.



Latihan Menggunakan Object Sebagai Parameter

Tahapannya kurang lebih sebagai berikut:

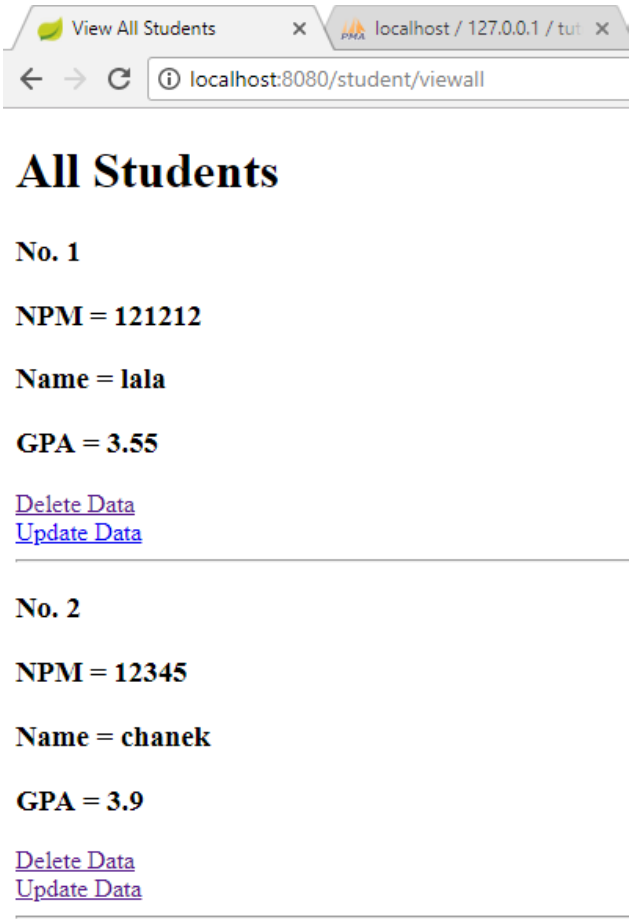
a. Menambahkan th:object=”${student}” pada tag <form> di view
b. Menambahkan th:field=”*{[nama_field]}” pada setiap input

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

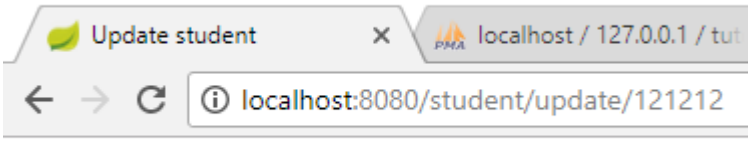
c. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel seperti gambar berikut ini.

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

d. Tes lagi aplikasi Anda.
Contoh tampilan view all menampilkan semua data student

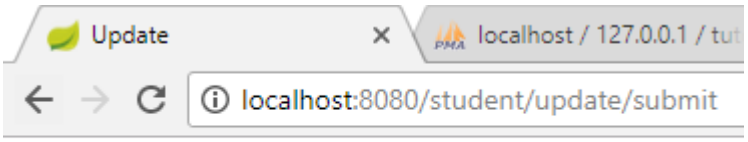


Contoh tampilan update student dengan npm 121212 yang terdapat pada database.

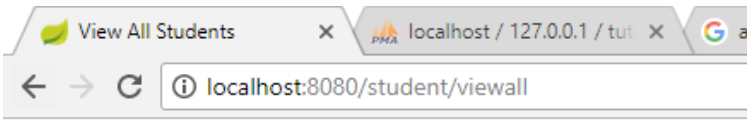


Problem Editor

NPM 121212
Name lala
GPA 3.55
Save



Data berhasil diupdate



All Students

No. 1

NPM = 121212

Name = lala

GPA = 3.75

Delete Data
Update Data

No. 2

NPM = 12345

Name = chanek

GPA = 3.9

Delete Data
Update Data

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?
   Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.
   Jawab:
   Jika menggunakan Object sebagai parameter pada form POST maka validasi ada di form view yang akan mengirim message error dan akan ditangkap oleh controller. Contohnya dengan menggunakan Thymeleaf yang menyediakan fasilitas untuk validasi pada view.

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?
   Jawab:
   Data yang dikirim melalui method POST lebih tertutup dan jauh lebih aman karena data tidak terlihat, dibandingkan dengan method GET yang HTTP client dapat mengambil informasi dari server dan menampilkan data pada url sehingga data terlihat dan juga dapat tersimpan di browser.

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