A. Latihan Menambahkan Delete

1. Pada viewall.html tambahkan

```
<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/>
    <hr/>
</div>
```

2. Tambahkan method deleteStudent yang ada di class StudentMapper

```
@Mapper
public interface StudentMapper
{
    @Select("select npm, name, gpa from student where npm = #{npm}")
    StudentModel selectStudent (@Param("npm") String npm);

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

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


}
```

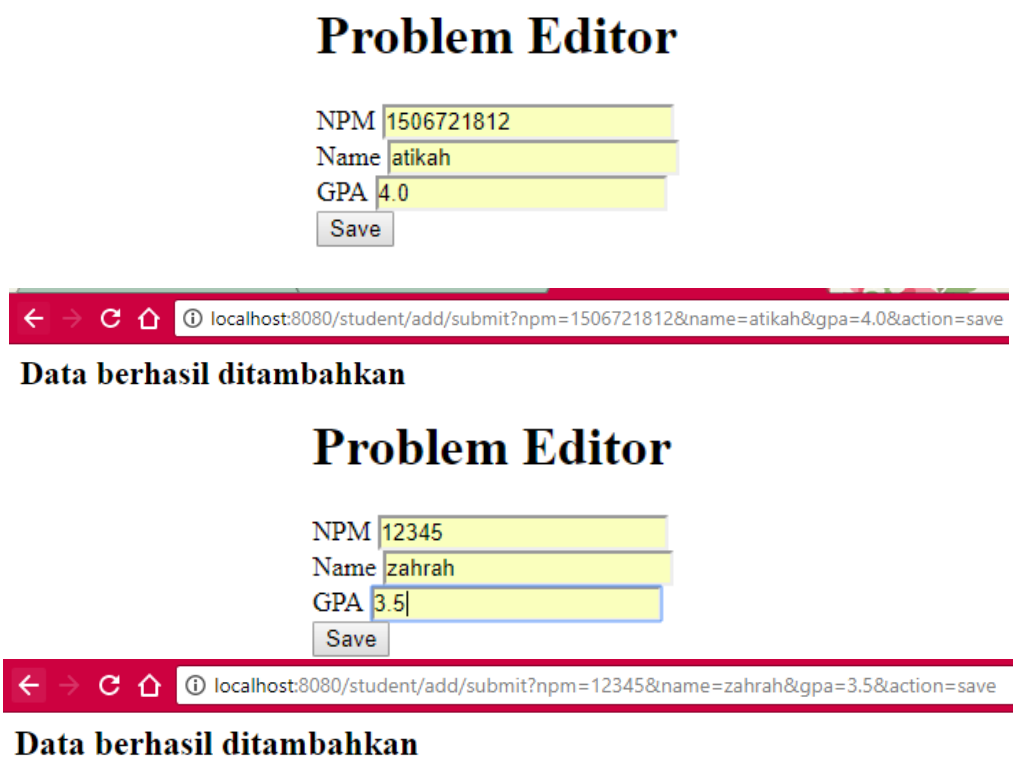
3. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

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

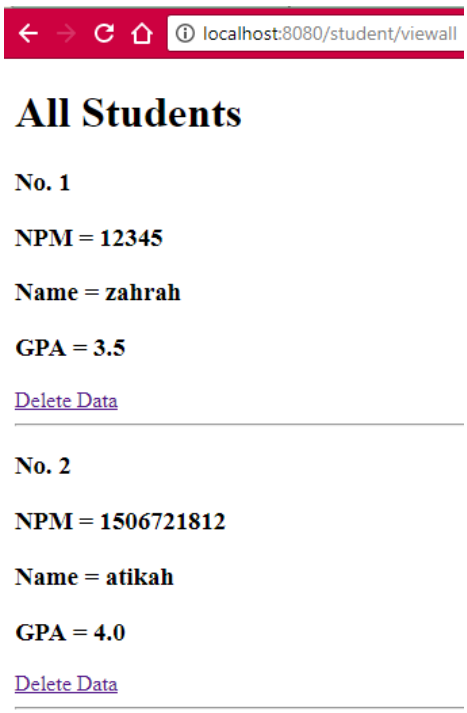
4. Lengkapi method delete pada class StudentController

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

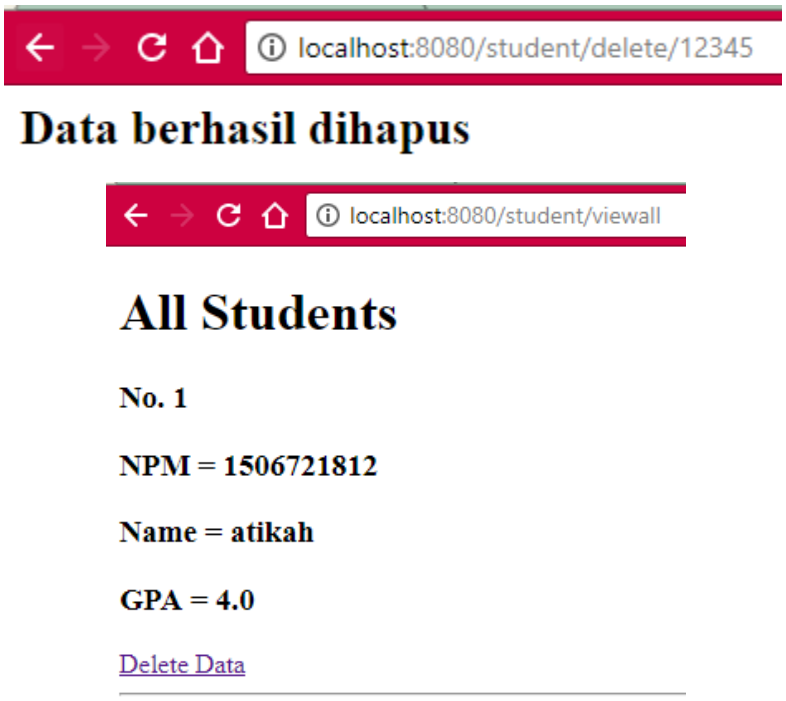
5. Jalankan Spring Boot app dan lakukan beberapa insert



6. Contoh tampilan View All



7. Contoh tampilan delete NPM 12345



8. Contoh tampilan jika dilakukan delete NPM 123 yang kedua kalinya

B. Latihan Menambahkan Update

1. Tambahkan method updateStudent pada class StudentMapper

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

2. Tambahkan method updateStudent pada interface StudentService

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

3. Tambahkan implementasi method updateStudent pada class StudentServiceDatabase. Jangan lupa tambahkan logging pada method ini.

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

4. Tambahkan link Update Data pada viewall.html

```
<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. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll. Ubah action form menjadi /student/update/submit. Ubah method menjadi 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="update">Update</button>
</div>
```

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

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

7. Tambahkan method update pada class StudentController. Request mapping ke /student/update/{npm}. Sama halnya seperti delete, lakukan validasi. Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-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("npm",npm);
        return "not-found";
    }
        studentDAO.updateStudent(student);
        return "form-update";
}
```

8. Tambahkan method updateSubmit pada class StudentController. Lengkapi method dengan memanggil method update Student dan kembalikan 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 = new StudentModel (npm, name, gpa);
    studentDAO.updateStudent(student);

    return "success-update";
}
```

9. Jalankan Spring Boot dan coba test program Anda.

C. Latihan Menggunakan Object Parameter

```
  @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";
  } */

  public String updateSubmit(StudentModel student) {
      studentDAO.updateStudent(student);
      return "success-update";
  }
```

D. Pertanyaan

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Jawab : Apabila ingin melakukan validasi pada form POST dapat dilakukan dengan menambahkan @PostMapping pada Controller. Validasi tetap dilakukan tetapi tidak langsung pada formnya.
Source : https://spring.io/guides/gs/validating-form-input/

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 : Form submit biasanya menggunakan POST method daripada GET method karena agar data tidak tampil secara eksplisit di URL. Tujuannya untuk melindungi data. Perlu penanganan yang berbeda di header method.
3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Jawab : Tidak bisa. Kecuali mempunyai 2 forms atau memili satu hal yang bisa disubmit dengan post via ajax dan submit form yang lainnya dengan get.

E. Apa yang dipelajari pada tutorial kali ini

Tuturial kali ini saya belajar tentang bagaimana Spring Boot terkoneksi dengan database pada mysql, bagaimana cara kerja dari form pada Spring Boot yang dari method get maupun method post. Saya juga bisa memahami cara submit form melalu method POST tanpa harus mendefinisikan masing-masing atribut dari form satu per satu dengan menggunakan object sebagai parameter

F. Penjelasan Kode

1. Method yang Anda buat pada Latihan Menambahkan Delete

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

Method ini akan menjalankan SQL DELETE yang ditrigger dengan eksekusi method delete dari Controller.

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

Mengambil dari npm dari URL lalu apabila npm yang dicari tidak ada pada database maka direct ke not-found.html. Apabila npm yang dimasukkan ada pada database maka direct ke delete.html

2. Method yang Anda buat pada Latihan Menambahkan Update

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

Method ini akan menjalankan SQL UPDATE yang ditrigger dengan eksekusi method update dari Controller.

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

    return "success-update";
}
```

Mengambil dari npm dari URL lalu apabila npm yang dicari tidak ada pada database maka direct ke not-found.html. Apabila npm yang dimasukkan ada pada database maka direct ke form-update.html. lalu pada method updateSubmit memasukkan ke form-update dan apabila klik “update” maka akan direct ke success-update

3. Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter

```
  @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";
  } */

  public String updateSubmit(StudentModel student) {
      studentDAO.updateStudent(student);
      return "success-update";
  }
```

Method ini menerima parameter object student dari class StudentModel lalu akan memanggil method updateStudent() pada class studentDAO dan menampilkan halaman “success-update”