WRITEUP TUTORIAL 4

A. Latihan Menambahkan Delete

1 . Source Code

- Pada StudentController.java

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

}
```

- Pada StudentMapper.java

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

- Pada StudentService.java

```
void deleteStudent (String npm);
```

- Pada StudentServiceDatabase.java

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

- Pada delete.html

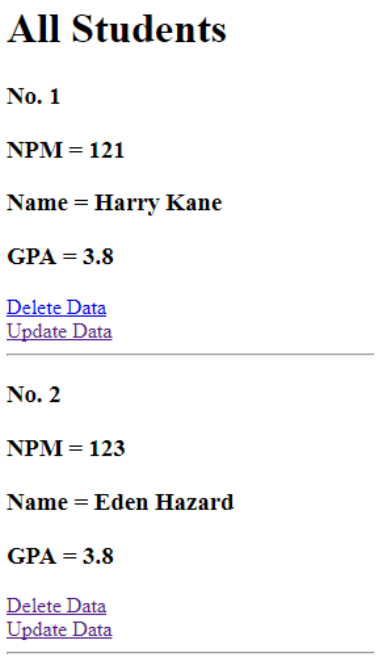
```
<html>
	<head>
		<title>Delete</title>
	</head>
	<body>
```

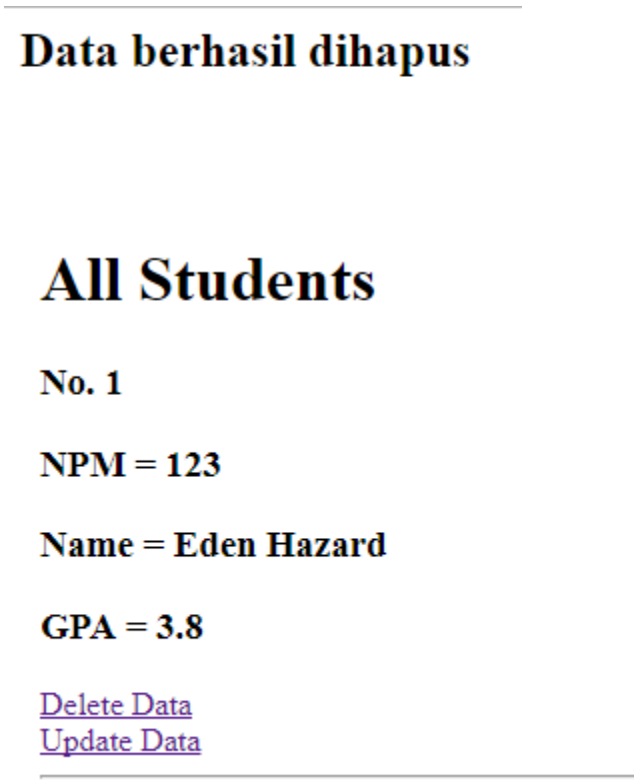
```
            <h2>Data berhasil dihapus</h2>
        </body>
</html>
```

- Pada viewall.html

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

2. Screenshot





3. Penjelasan

Method-method untuk fitur delete data ini ditambahkan pada kelas StudentController, StudentService, StudentMapper, StudentServiceDatabase, dan halaman viewall serta delete. Pada StudentController, method deleteStudent ditambahkan untuk menerima student mana yang akan dihapus. Method yang berfungsi untuk mengirimkan pesan ke SQL untuk menghapus data student yang akan dihapus berada di StudentMapper yang memetakannya ke SQL. StudentServiceDatabase menghubungkan antara Controller dan Mapper serta menuliskan log terkait aktivitas penghapusan data student tersebut.

B. Latihan Menambahkan Update

1. Source Code

- Pada StudentController.java

```
@RequestMapping("/student/update/{npm}")
 public String update (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";
   }
 }

 @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 StudentMapper.java

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

- Pada StudentService.java

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

- Pada StudentServiceDatabase.java

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

- 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">Update</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>
```

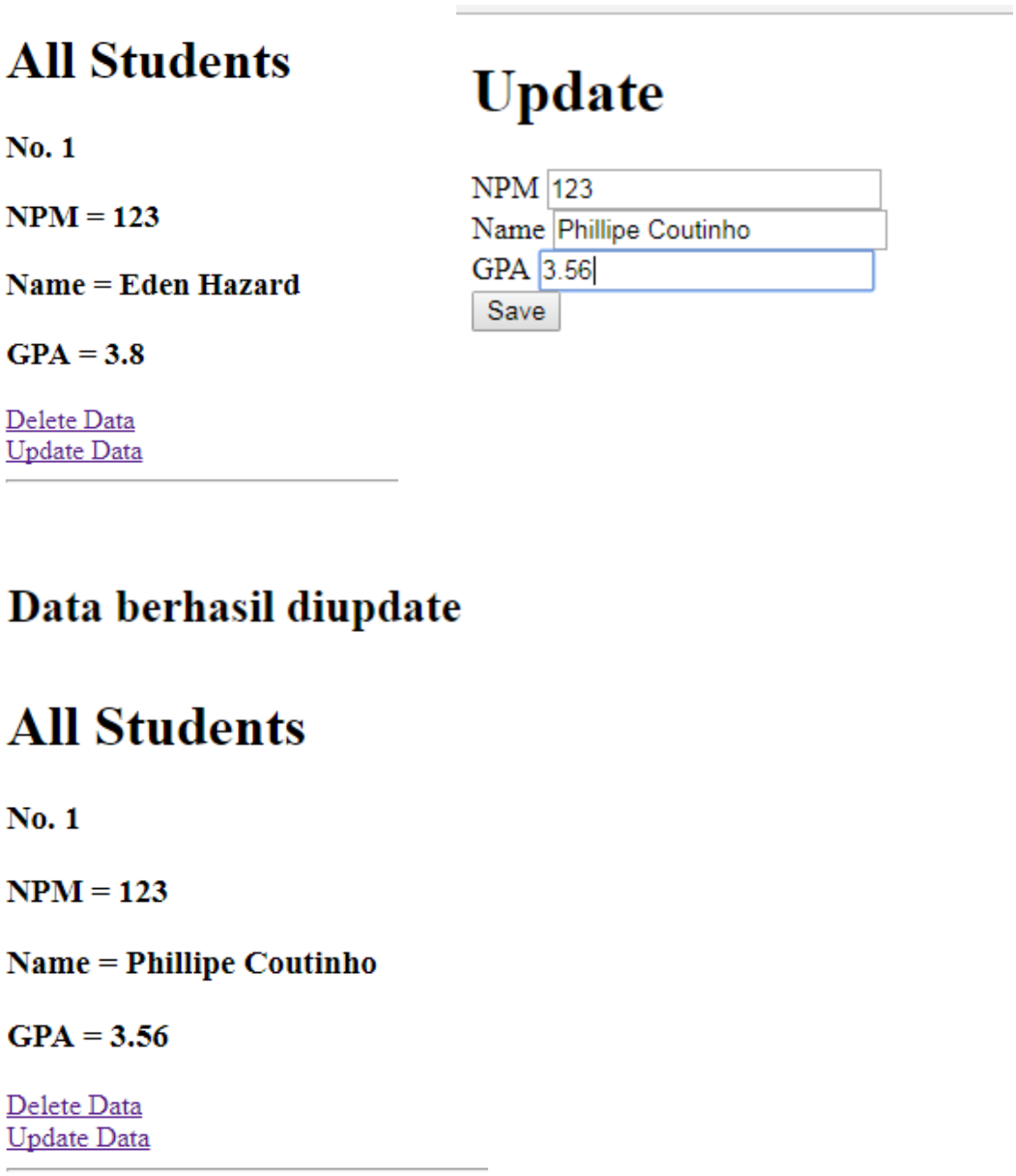
- Pada success-update.html

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

- Pada viewall.html

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

2. Screenshot



3. Penjelasan

Fitur update data student ini hamper mirip dengan fitur add student yang dimana keduanya menggunakan form. Kelas StudentController menerima string npm yang akan diperiksa apakah student yang berkaitan terdaftar atau tidak. Setelah ditemukan, nantinya akan diteruskan ke view form-update.html yang terdapat form untuk mengubah data student (kecuali npm). Setelah mengubah data, terdapat method di StudentController yang akan menerima data student yang diubah dan kemudian akan memanggil method updateStudent di StudentMapper.java melalui StudentService. Di StudentMapper inilah terdapatmethod yang menghubungkan ke Database SQL sehingga dapat melakukan query update. Setelah

berhasil diubah, maka akan diteruskan ke view yang menyatakan bahwa data berhasil diubah (success-update.html).

C. Latihan Menggunakan Object Parameter

1. Source Code

- StudentController.java

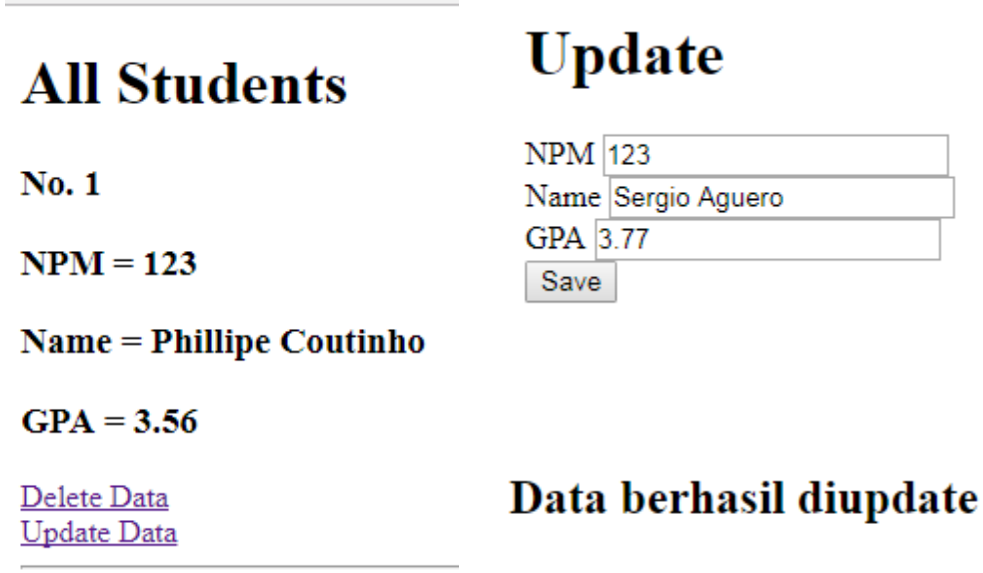
```
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(StudentModel student)
{
 //StudentModel student = new StudentModel (npm, name, gpa);
  studentDAO.updateStudent (student);
  return "success-update";
}
```

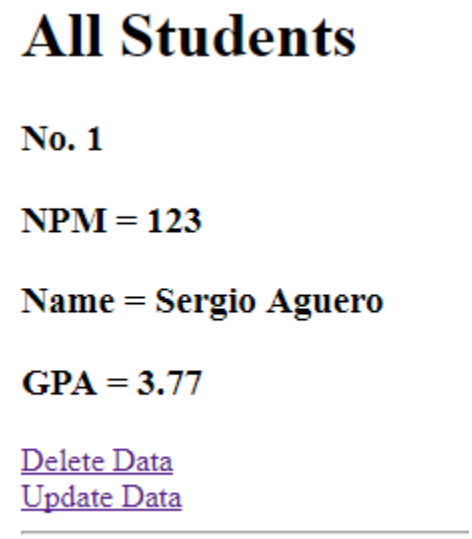
- form-update.html

```
<!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">Update</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}"/>
          </div>
          <div>
                <label for="name">Name</label> <input type="text"
name="name" th:field="*{name}"/>
          </div>
          <div>
                <label for="gpa">GPA</label> <input type="text" name="gpa"
th:field="*{gpa}"/>
          </div>

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

```
            </div>
        </form>

    </body>

    </html>
```

2. Screenshot





3. Penjelasan

Yang berubah dengan menggunakan Objek sebagai parameter hanya penerimaan parameter di method updateSubmit yang sekarang berupa objek Student. Method update di StudentController.java sebelumnya mengirimkan objek student dan, berbeda dengan dengan fitur update sebelumnya, di view form-update-html ditambahkan tag yang berbeda. Proses selanjutnya untuk mengubah data student tetap sama dengan latihan update sebelumnya.

D. 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.

- Untuk validasi yang required, kita dapat melakukan seperti ketika parameter menggunakan @PathVariable yang dimana terdapat pengecekan melalui sebuah method apakah data yang terkait itu null atau tidak. Contohnya :

```
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(StudentModel student)
{
        if(student != null ) {
        studentDAO.updateStudent (student);
        return "success-update";
        } else {
                //do something
                return null;
        }
      }
```

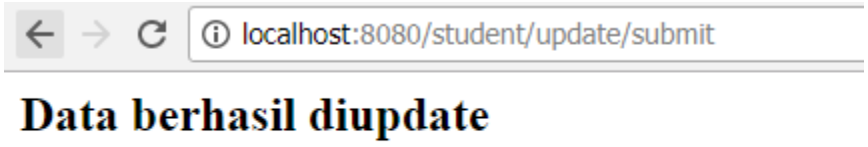
2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

- Perbedaan terletak ketika kita mengklik tombol submit dan mengirimkan data untuk disimpan di server. Action untuk form dengan HTTP Request Method GET biasanya akan menampilkan data yang terkait di URL sehingga bisa terindikasi terlihat oleh orang lain. Sedangkan POST tidak akan menampilkan sehingga akan lebih aman. Maka sebenarnya, POST lebih umum digunakan untuk menambahkan atau mengubah data ke server dibandingkan ke GET. Selain itu, tedapat limitasi di GET yang dimana bila data yang digunakan sangatlah panjang, URL yang disediakan browser belum tentu dapat menampung. Perbedaan yang signifikan pada header method untuk menggunakan POST atau GET hanyalah pada di anotasi RequestMapping dimana terdapat pernyataan bahwa method yang digunakan dispesifikasi (default apabila tidak ada pernyataan tersebut adalah RequestMethod GET).

Contoh perbedaan di url ketika memakai fitur addStudent (menggunakan GET) dan fitur updateStudent (menggunakan POST) :

addStudent :

updateStudent:



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

   Ya dapat digunakan bersamaan dengan cara seperti ini,

```
method = { RequestMethod.GET, RequestMethod.POST }
```