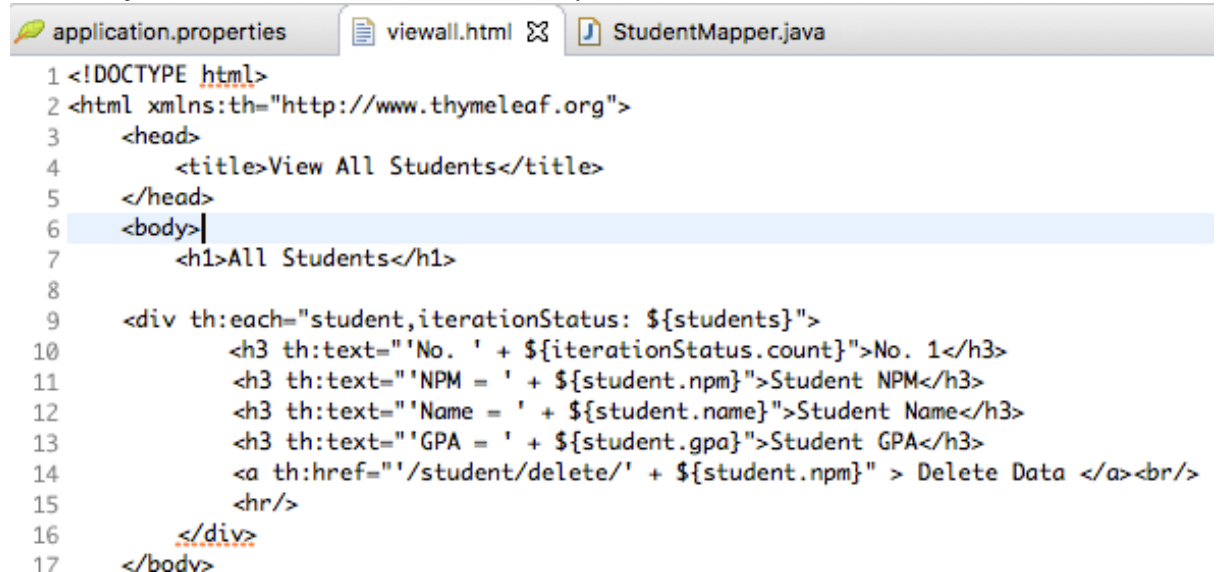


Tutorial 4 Write-Up

Latihan Menambahkan Delete

1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.
2. Pada **viewall.html** tambahkan

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



```

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <title>View All Students</title>
5   </head>
6   <body>
7     <h1>All Students</h1>
8
9     <div th:each="student, iterationStatus: ${students}">
10       <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
11       <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
12       <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
13       <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
14       <a th:href="/student/delete/" + ${student.npm}" > Delete Data </a><br/>
15       <hr/>
16     </div>
17   </body>

```

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**

```

void deleteStudent(StudentModel student);

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

```

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

```
application.properties  viewall.html  StudentMapper.java  StudentServiceDatab
33     return studentMapper.selectAllStudents ();
34 }
35
36
37 @Override
38 public void addStudent (StudentModel student)
39 {
40     studentMapper.addStudent (student);
41 }
42
43
44 @Override
45 public void deleteStudent (String npm)
46 {
47     log.info("student " + npm + " deleted");
48     studentMapper.deleteStudent(npm);
49 }
```

5. Lengkapi method delete pada class StudentController

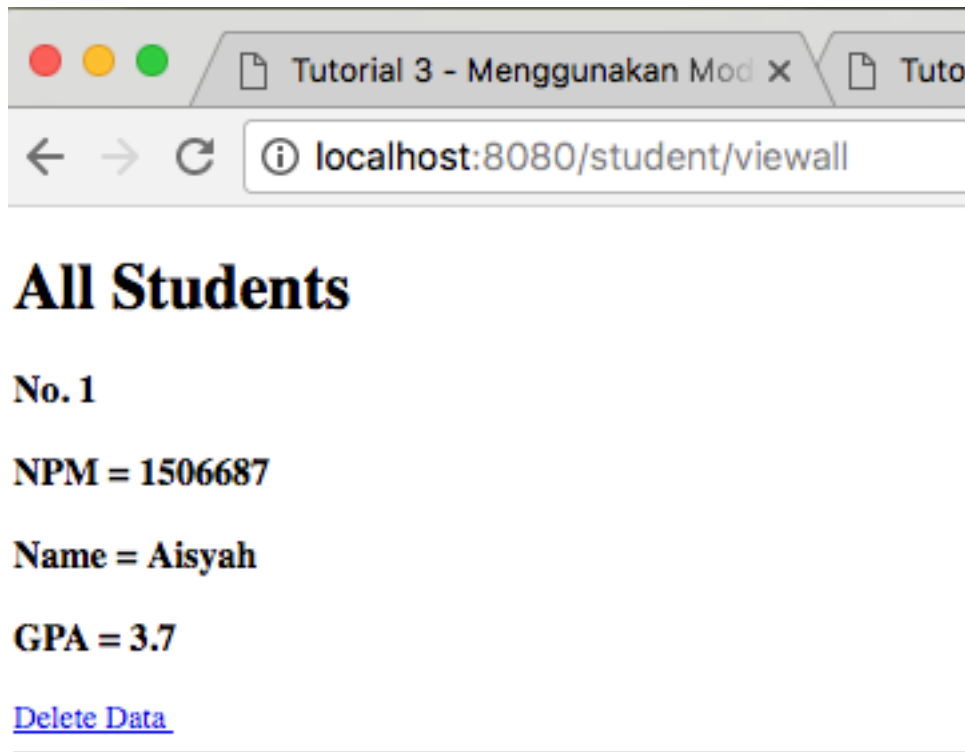
```
application.properties  viewall.html  StudentMapper.java  StudentServiceDatabase.java  StudentControl
APAPTutorial04/src/main/resources/
application.properties
88 }
89
90
91 @RequestMapping("/student/delete/{npm}")
92 public String delete (Model model, @PathVariable(value = "npm") String npm)
93 {
94     for(int i = 0; i < studentDAO.selectAllStudents().size(); i++){
95         if(studentDAO.selectAllStudents().get(i).getNpm().equalsIgnoreCase(npm)){
96             studentDAO.deleteStudent (npm);
97             return "delete";
98         }
99     }
100
101     return "not-found";
102 }
103
```

6. Jalankan **Spring Boot app** dan lakukan beberapa insert
7. Tampilan **viewAll**

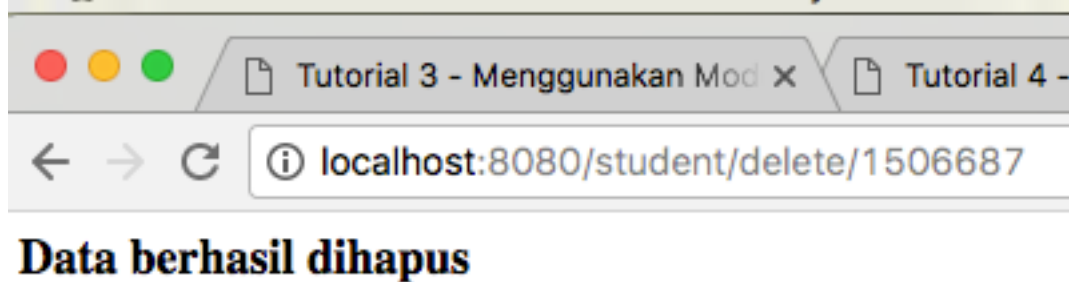
Tutorial 3 - Menggunakan Mod x Tutorial 4 - Menggunakan Date x localhost:8000 / localhost | ph x

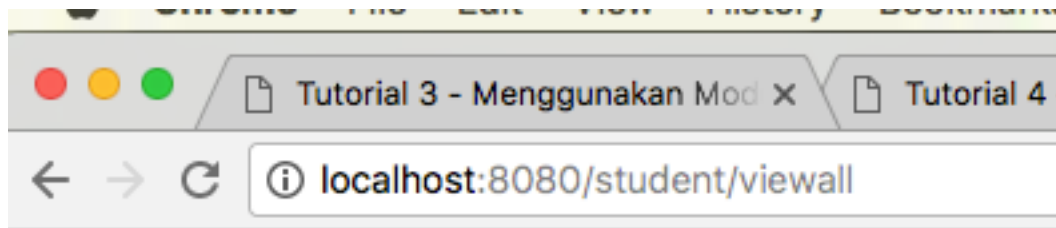
localhost:8080/student/add/submit?npm=1506687&name=Aisyah&gpa=3.70&action=save

Data berhasil ditambahkan



8. Contoh tampilan **delete NPM**

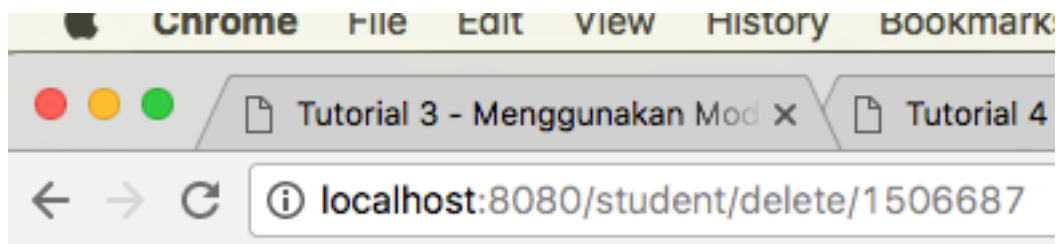




All Students

(data student hilang)

9. Tampilan jika dilakukan **delete NPM** yang kedua kalinya

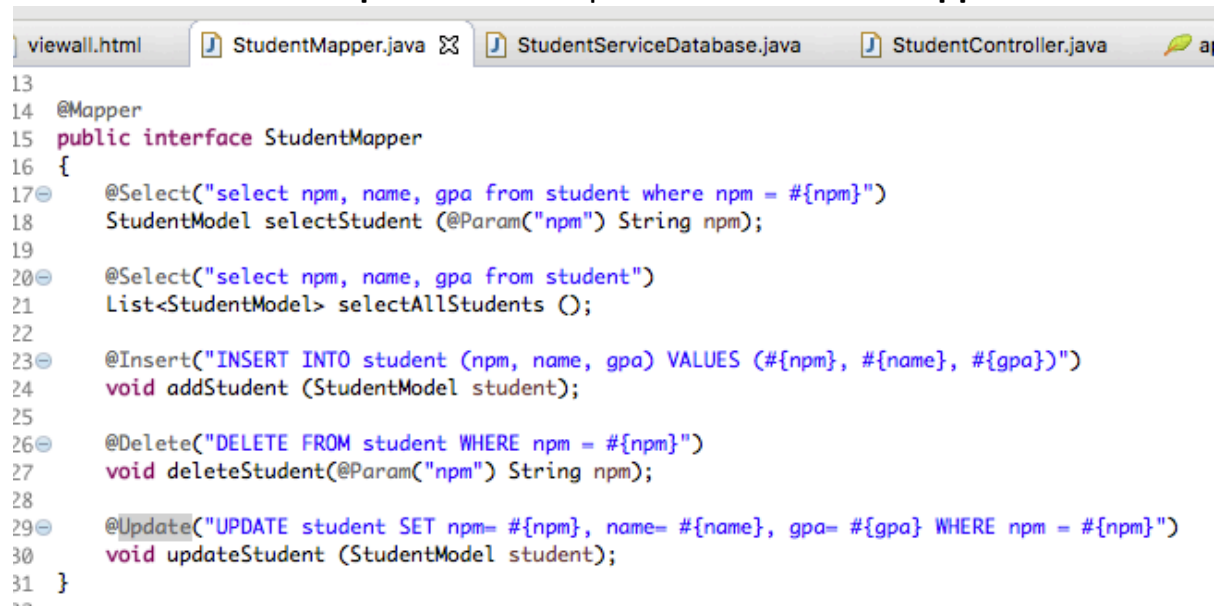


Student not found

NPM = 1506687

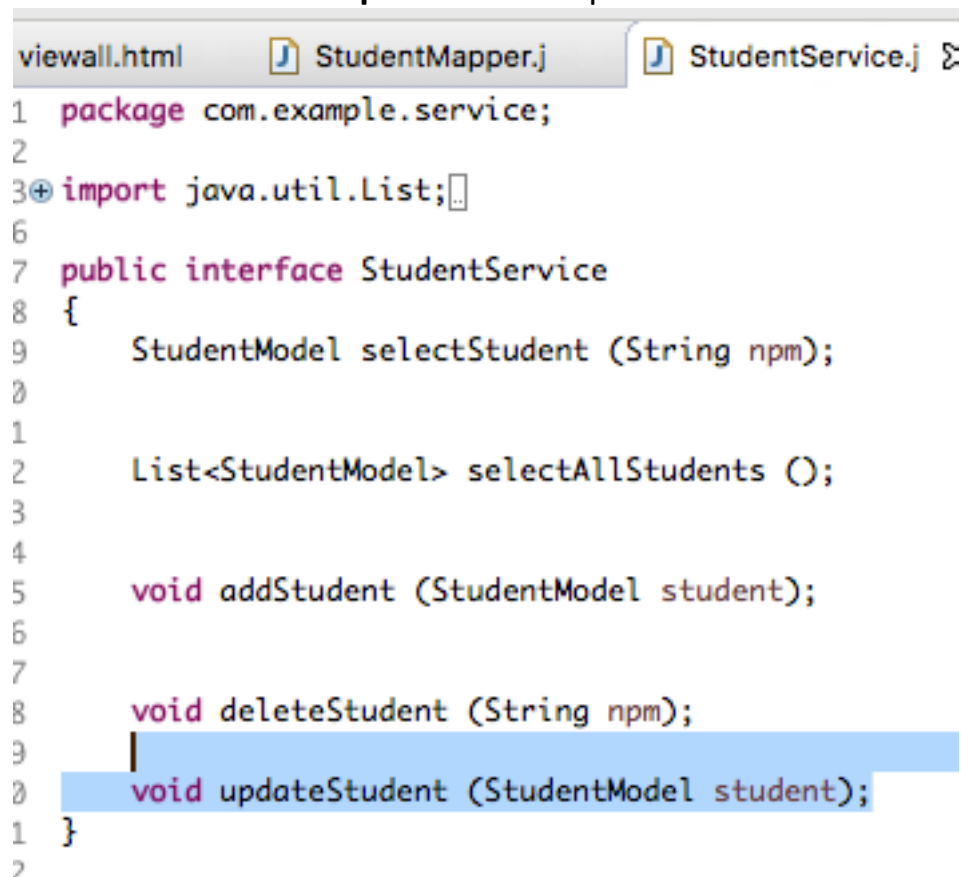
Latihan Menambahkan Update

1. Tambahkan method **updateStudent** pada class **StudentMapper**



```
viewall.html StudentMapper.java StudentServiceDatabase.java StudentController.java
13
14 @Mapper
15 public interface StudentMapper
16 {
17     @Select("select npm, name, gpa from student where npm = #{npm}")
18     StudentModel selectStudent (@Param("npm") String npm);
19
20     @Select("select npm, name, gpa from student")
21     List<StudentModel> selectAllStudents ();
22
23     @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
24     void addStudent (StudentModel student);
25
26     @Delete("DELETE FROM student WHERE npm = #{npm}")
27     void deleteStudent(@Param("npm") String npm);
28
29     @Update("UPDATE student SET npm= #{npm}, name= #{name}, gpa= #{gpa} WHERE npm = #{npm}")
30     void updateStudent (StudentModel student);
31 }
```

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



```
viewall.html StudentMapper.j StudentService.j
1 package com.example.service;
2
3 import java.util.List;
4
5 public interface StudentService
6 {
7     StudentModel selectStudent (String npm);
8
9     List<StudentModel> selectAllStudents ();
10
11     void addStudent (StudentModel student);
12
13     void deleteStudent (String npm);
14     void updateStudent (StudentModel student);
15 }
```

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

method ini.

```
viewall.html StudentMapper.j StudentService.j StudentServiceD
{
    studentMapper.addStudent (student);
}

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

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

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

```
viewall.html StudentMapper.j StudentService.j StudentServiceD StudentCo
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3     <head>
4         <title>View All Students</title>
5     </head>
6     <body>
7         <h1>All Students</h1>
8
9         <div th:each="student, iterationStatus: ${students}">
10             <h3 th:text="'No. ' + ${iterationStatus.count}'>No. 1</h3>
11             <h3 th:text="'NPM = ' + ${student.npm}'>Student NPM</h3>
12             <h3 th:text="'Name = ' + ${student.name}'>Student Name</h3>
13             <h3 th:text="'GPA = ' + ${student.gpa}'>Student GPA</h3>
14             <a th:href="/student/delete/' + ${student.npm}" > Delete Data </a><br/>
15             <a th:href="/student/update/' + ${student.npm}" > Update Data </a><br/>
16             <hr/>
17         </div>
18     </body>
```

5. Copy view **form-add.html** menjadi **form-update.html**.

```
viewall.html StudentMapper.j StudentService. StudentServiceD form-update.htm »2
1 <!DOCTYPE HTML>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3   xmlns:th="http://www.thymeleaf.org">
4 <head>
5
6 <title>Update student</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8 </head>
9
10 <body>
11
12   <h1 class="page-header">Update Student</h1>
13
14   <form th:object="${student}" action="/student/update/submit" method="post">
15     <div>
16       <label for="npm">NPM</label> <input th:field="*{npm}" type="text" name="npm" readonly="true" th:value="${stuc
17     </div>
18     <div>
19       <label for="name">Name</label> <input th:field="*{name}" type="text" name="name" th:value="${student.name}" /
20     </div>
21     <div>
22       <label for="gpa">GPA</label> <input th:field="*{gpa}" type="text" name="gpa" th:value="${student.gpa}"/>
23     </div>
24     <div>
25       <button type="submit" name="action" value="save">Update</button>
26     </div>
27   </form>
28
29 </body>
30
31 </html>
32
--
```

6. Copy view **success-add.html** menjadi **success-update.html**

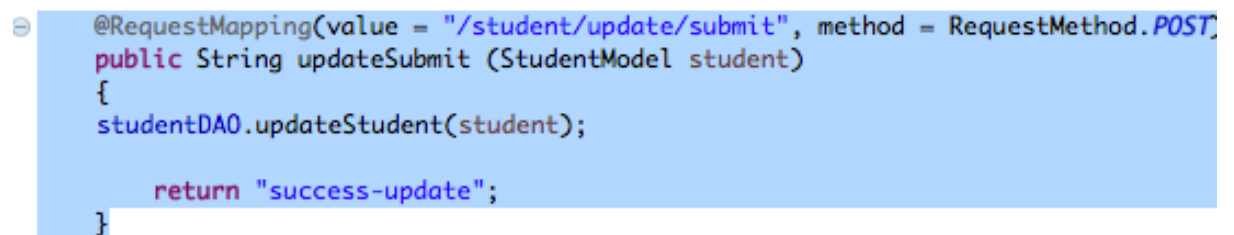
```
viewall.html StudentService. StudentServiceD form-update.htm success-update.
1 <html>
2   <head>
3     <title>Update</title>
4   </head>
5   <body>
6     <h2>Data berhasil diperbaharui</h2>
7   </body>
8 </html>
```

7. Tambahkan method update pada class **StudentController**



```
viewall.html StudentServiceD StudentControll form-update.htm success-
10
11 @RequestMapping("/student/delete/{npm}")
12 public String delete (Model model, @PathVariable(value = "npm") String npm)
13 {
14     for(int i = 0; i< studentDAO.selectAllStudents().size(); i++){
15         if(studentDAO.selectAllStudents().get(i).getNpm().equalsIgnoreCase(npm)){
16             studentDAO.deleteStudent (npm);
17             return "delete";
18         }
19     }
20
21     return "not-found";
22 }
23
24 @RequestMapping("/student/update/{npm}")
25 public String update (Model model, @PathVariable(value = "npm") String npm)
26 {
27     StudentModel student = studentDAO.selectStudent (npm);
28
29     if (student != null) {
30         model.addAttribute("student", student);
31         return "form-update";
32     } else {
33         model.addAttribute ("npm", npm);
34         return "not-found";
35     }
36 }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

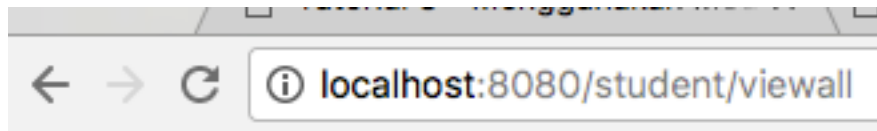
8. Tambahkan method **updateSubmit** pada class **StudentController**



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

    return "success-update";
}
```

9. Jalankan **Spring Boot** dan coba test program Anda



All Students

No. 1

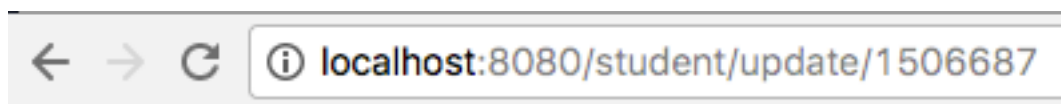
NPM = 1506687

Name = Ais

GPA = 3.5

[Delete Data](#)

[Update Data](#)

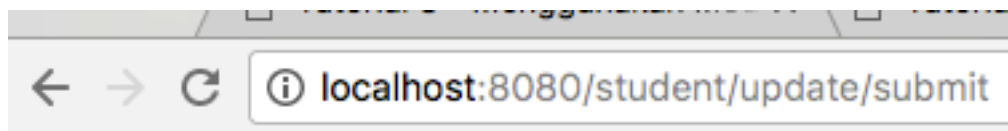


Update Student

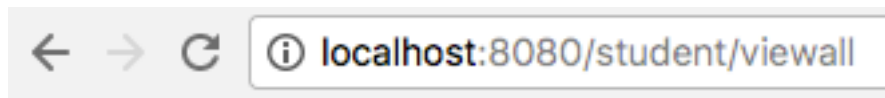
NPM

Name

GPA



Data berhasil diperbaharui



All Students

No. 1

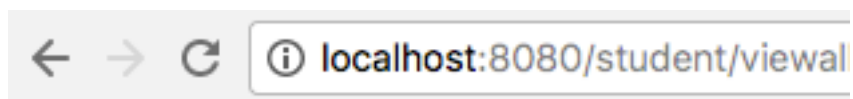
NPM = 1506687

Name = AisBaru

GPA = 3.9

[Delete Data](#)

[Update Data](#)



All Students

No. 1

NPM = 1506687

Name = AisBaru

GPA = 3.9

[Delete Data](#)

[Update Data](#)

Latihan Menggunakan Object Sebagai Parameter

1. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.
2. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakan RequestParam.
3. Cara lengkapnya silakan ikuti pada link Handling Form berikut:
(<https://spring.io/guides/gs/handling-form-submission/>)
4. Tahapannya kurang lebih sebagai berikut:
 - a. Menambahkan th:object="\${student}" pada tag di view

- b. Menambahkan th:field="*{[nama_field]}" pada setiap input



```
4 <head>
5
6 <title>Update student</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8 </head>
9
10 <body>
11
12 <h1 class="page-header">Update Student</h1>
13
14 <form th:object="${student}" action="/student/update/submit" method="post">
15     <div>
16         <label for="npm">NPM</label> <input th:field="*{[npm]}" type="text" name="npm" readonly="true" th:value="${s
17     </div>
18     <div>
19         <label for="name">Name</label> <input th:field="*{[name]}" type="text" name="name" th:value="${student.name}
20     </div>
21     <div>
22         <label for="gpa">GPA</label> <input th:field="*{[gpa]}" type="text" name="gpa" th:value="${student.gpa}"/>
23     </div>
24     <div>
25         <button type="submit" name="action" value="save">Update</button>
26     </div>
27 </form>
28
29 </body>
30
31 </html>
```

- c. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

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.





Untuk melakukan validasi pada form POST dapat dilakukan dengan menambahkan @PostMapping pada Controller. Validasi tetap dilakukan tetapi tidak langsung pada formnya.

Sumber: <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?
Form submit biasanya menggunakan POST method dibanding 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?

Tidak bisa. Sumber: <https://stackoverflow.com/questions/6918991/submit-post-and-get-variables-in-one-form>

-  You cannot do a post and a get in the same form you even said that in your question.
-  1 You have to either:
-  1. Have 2 forms
-  2. Have one thing submit with a post via ajax and then submit the other form with a get

- Jelaskan method yang Anda buat pada Latihan Menambahkan Delete


Parameter NPM pada method ini berfungsi sebagai @PathVariable, lalu npm yang telah diinputkan tersebut terlebih dahulu method selectStudent() pada object studentDAO.

Halaman “not-found” akan ditampilkan apabila object student tersebut null atau tidak ada. Sebaliknya, jika student tersebut tidak null, dimana artinya npm tersebut ada pada database, maka method deleteStudent() akan dijalankan berdasarkan npm pada object studentDAO

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

- Jelaskan Method yang Anda buat pada Latihan Menambahkan Update Parameter NPM pada method ini berfungsi sebagai **@PathVariable**, lalu npm yang telah diinputkan tersebut terlebih dahulu method **selectStudent()** pada object **studentDAO**.

Halaman “not-found” akan ditampilkan apabila object student tersebut null atau tidak ada. Sebaliknya, jika student tersebut tidak null, dimana artinya npm tersebut ada pada database, maka method ini akan menambahkan object student sebagai atribut pada model, lalu memanggil view “form-update”



```
viewall.html StudentServiceD StudentControll form-update.htm success-
10
11 @RequestMapping("/student/delete/{npm}")
12 public String delete (Model model, @PathVariable(value = "npm") String npm)
13 {
14     for(int i = 0; i< studentDAO.selectAllStudents().size(); i++){
15         if(studentDAO.selectAllStudents().get(i).getNpm().equalsIgnoreCase(npm)){
16             studentDAO.deleteStudent (npm);
17             return "delete";
18         }
19     }
20
21     return "not-found";
22 }
23
24 @RequestMapping("/student/update/{npm}")
25 public String update (Model model, @PathVariable(value = "npm") String npm)
26 {
27     StudentModel student = studentDAO.selectStudent (npm);
28
29     if (student != null) {
30         model.addAttribute("student", student);
31         return "form-update";
32     } else {
33         model.addAttribute ("npm", npm);
34         return "not-found";
35     }
36 }
37
38
39
40
```

```
@Update("UPDATE student SET npm= #{npm}, 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.

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

Pendefinisian method ini terletak pada interface

- Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
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”

Lesson Learned:

Pada tutorial 4 kali ini, saya dapat memahami bagaimana Spring Boot terkoneksi dengan database MySQL. Selain itu, saya juga bias mengerti bagaimana cara kerja dari form pada Spring Boot (Baik itu melalui method POST atau GET). Saya dapat mengetahui pula cara untuk submit form melalui method POST tanpa harus mendefinisikan masing-masing atribut dari masing-masing form, yaitu menggunakan Object sebagai Parameter.