

Latihan Menambahkan Delete

1. Pada viewall.html tambahkan

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

```
viewall.html
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</a><br/>
15      <a th:href="/student/update/" + ${student.npm}">Update</a><br/>
16      <hr/>
17    </div>
18  </body>
19</html>
```

2. Tambahkan method deleteStudent yang ada di class StudentMapper

- Tambahkan method delete student yang menerima parameter NPM.
- Tambahkan annotation delete di atas dan SQL untuk menghapus @Delete("[LENGKAPI]")

```
StudentMapper.java
1 package com.example.dao;
2
3 import java.util.List;
4
5 @Mapper
6 public interface StudentMapper
7 {
8     @Select("select npm, name, gpa from student where npm = #{npm}")
9     StudentModel selectStudent (@Param("npm") String npm);
10
11     @Select("select npm, name, gpa from student")
12     List<StudentModel> selectAllStudents ();
13
14     @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
15     void addStudent (StudentModel student);
16
17     @Delete("DELETE FROM student WHERE npm = #{npm}")
18     void deleteStudent (@Param("npm") String npm);
19 }
```

3. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

- Tambahkan log untuk method tersebut dengan cara menambahkan log.info ("student " + npm + " deleted");
- Panggil method delete student yang ada di Student Mapper

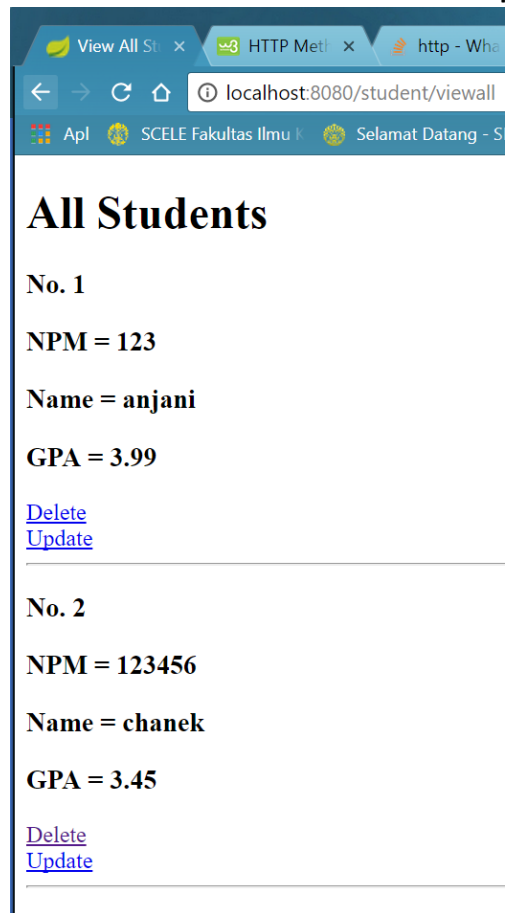
```
StudentServiceDatabase.java
24     Log.info ("select student with npm {}", npm);
25     return studentMapper.selectStudent (npm);
26 }
27
28 @Override
29 public List<StudentModel> selectAllStudents ()
30 {
31     Log.info ("select all students");
32     return studentMapper.selectAllStudents ();
33 }
34
35 @Override
36 public void addStudent (StudentModel student)
37 {
38     studentMapper.addStudent (student);
39 }
40
41 @Override
42 public void deleteStudent (String npm)
43 {
44     Log.info ("student " + npm + " deleted");
45     studentMapper.deleteStudent(npm);
46 }
```

4. Lengkapi method delete pada class StudentController

- a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found

```
StudentController.java
93
94 @RequestMapping("/student/delete/{npm}")
95 public String delete (Model model, @PathVariable(value = "npm") String npm)
96 {
97     // studentDAO.deleteStudent(npm);
98     StudentModel students = studentDAO.selectStudent(npm);
99     if (students == null){
100         return "not-found";
101     } else {
102         studentDAO.deleteStudent(npm);
103         return "delete";
104     }
105 }
```

b. Jika berhasil delete student dan tampilkan view delete



All Students

No. 1

NPM = 123

Name = anjani

GPA = 3.99

[Delete](#)

[Update](#)

No. 2

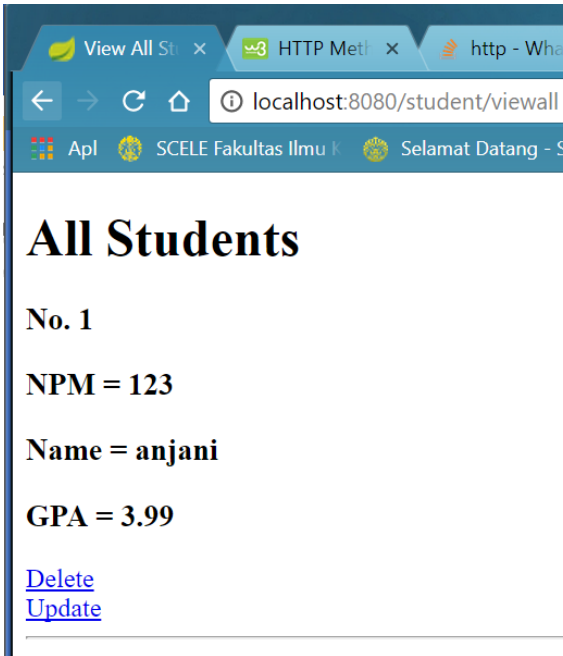
NPM = 123456

Name = chanek

GPA = 3.45

[Delete](#)

[Update](#)



All Students

No. 1


NPM = 123

Name = anjani

GPA = 3.99


[Delete](#)

[Update](#)



Data berhasil dihapus

Ketika data dihapus untuk kedua kali nya



Student not found

NPM = 12345

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

```
StudentMapper.java
1 package com.example.dao;
2
3 import java.util.List;
4
5 @Mapper
6 public interface StudentMapper
7 {
8     @Select("select npm, name, gpa from student where npm = #{npm}")
9     StudentModel selectStudent (@Param("npm") String npm);
10
11     @Select("select npm, name, gpa from student")
12     List<StudentModel> selectAllStudents ();
13
14     @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
15     void addStudent (StudentModel student);
16
17     @Delete("DELETE FROM student WHERE npm = #{npm}")
18     void deleteStudent (@Param("npm") String npm);
19
20     @Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm =#{npm}")
21     void updateStudent(StudentModel student);
22 }
23
24
25
26
27
28
29
30
31
32
```

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

```
StudentService.java
1 package com.example.service;
2
3 import java.util.List;
4
5
6
7 public interface StudentService
8 {
9     StudentModel selectStudent (String npm);
10
11     List<StudentModel> selectAllStudents ();
12
13     void addStudent (StudentModel student);
14
15     void deleteStudent (String npm);
16
17     void updateStudent(StudentModel student);
18 }
19
```

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

```
StudentServiceDatabase.java
23 {
24     Log.info ("select student with npm {}", npm);
25     return studentMapper.selectStudent (npm);
26 }
27
28 @Override
29 public List<StudentModel> selectAllStudents ()
30 {
31     Log.info ("select all students");
32     return studentMapper.selectAllStudents ();
33 }
34
35 @Override
36 public void addStudent (StudentModel student)
37 {
38     studentMapper.addStudent (student);
39 }
40
41 @Override
42 public void deleteStudent (String npm)
43 {
44     Log.info ("student " + npm + " deleted");
45     studentMapper.deleteStudent(npm);
46 }
47
48 @Override
49 public void updateStudent(StudentModel student) {
50     Log.info("student " + student + "update" );
51     studentMapper.updateStudent(student);
52 }
53 }
54
```

4. Tambahkan link Update Data pada viewall.html

```
viewall.html
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</a><br/>
15      <a th:href="/student/update/" + ${student.npm}">Update</a><br/>
16    </div>
17  </body>
18</html>
```

5. Copy view form-add.html menjadi form-update.html.
 - a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.
 - b. Ubah action form menjadi /student/update/submit
 - c. Ubah method menjadi post

```
StudentServiceDatabase.java form-update.html
1<!DOCTYPE HTML>
2<html xmlns:th="http://www.thymeleaf.org">
3<head>
4
5<title>Update student</title>
6</head>
7
8<body>
9    <h1 class="page-header">Update Student</h1>
10
11    <form action="/student/update/submit" method="post" th:object="${student}">
12        <div>
13            <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${students.npm}"/>
14        </div>
15
16        <div>
17            <label for="name">NAME</label> <input type="text" name="name" th:value="${students.name}"/>
18        </div>
19
20        <div>
21            <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${students.gpa}"/>
22        </div>
23        <div>
24            <button type="submit" name="action" value="save">Update</button>
25        </div>
26    </form>
27
28</body>
29</html>
--
```

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

```
success-update.html
1<html>
2    <head>
3        <title>Add</title>
4    </head>
5    <body>
6        <h2>Data berhasil diperbaharui</h2>
7    </body>
8</html>
```

7. Tambahkan method update pada class StudentController
 - a. Request mapping ke /student/update/{npm}
 - b. Sama halnya seperti delete, lakukan validasi.
 - c. Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update

```
StudentController.java
106
107 @RequestMapping("/student/update/{npm}")
108 public String update (Model model, @PathVariable(value = "npm") String npm){
109     StudentModel students = studentDAO.selectStudent(npm);
110     model.addAttribute("students", students);
111
112     if (students == null){
113         return "not-found";
114     } else {
115         return "form-update";
116     }
117 }
118
```

8. Tambahkan method updateSubmit pada class StudentController
 - a. Karena menggunakan post method maka request mappingnya adalah sebagai berikut:
 - b. Lengkapi method dengan memanggil method update Student dan kembalikan view success-update

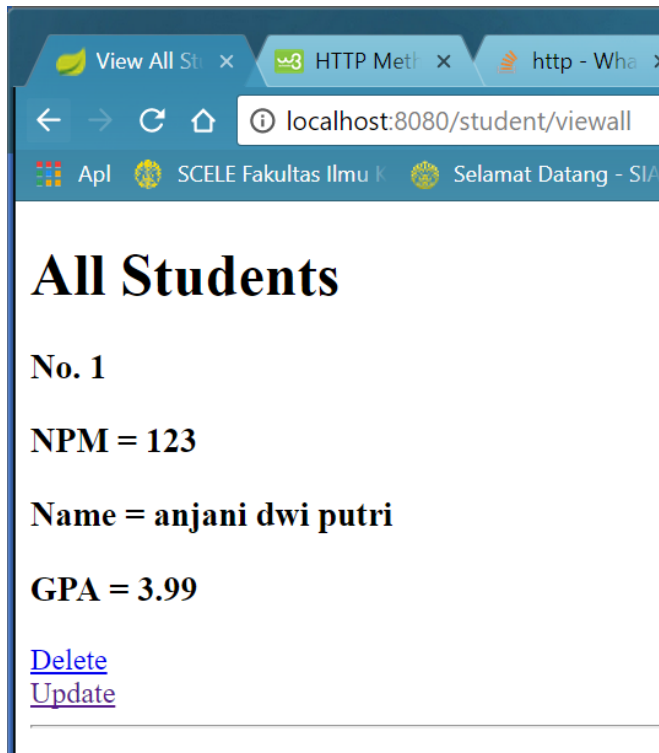
```
StudentController.java
119 @RequestMapping("/student/update/submit")
120 public String updateSubmit (@RequestParam(value = "npm", required = false) String npm,
121     @RequestParam(value="name", required = false) String name,
122     @RequestParam(value="gpa", required=false) double gpa) {
123
124     StudentModel students = new StudentModel(npm, name, gpa);
125
126     studentDAO.updateStudent(students);
127     return "success-update";
128 }
129
```

9. Jalankan Spring Boot dan coba test program Anda

The first screenshot shows a web browser at `localhost:8080/student/viewall` displaying the 'All Students' page. It lists student details for 'No. 1' with NPM = 123, Name = anjani, and GPA = 3.99. There are links for 'Delete' and 'Update'.

The second screenshot shows the 'Edit Student' page at `localhost:8080/student/update/123`. It contains input fields for NPM (123), NAME (anjani dwi putri), and GPA (3.99), along with an 'Update' button.

The third screenshot shows the 'Data berhasil diperbaharui' (Data successfully updated) message at `localhost:8080/student/update/submit`.



Latihan Menggunakan Object Sebagai Parameter

1. Menambahkan `th:object="${student}"` pada tag `<form>` di view

```
view.html
1<!DOCTYPE html>
2<html xmlns:th="http://www.thymeleaf.org">
3  <head>
4    <title>View Student by NPM</title>
5  </head>
6  <body>
7    <form th:object="${student}">
8      <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
9      <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
10     <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
11     </form>
12  </body>
13</html>
```

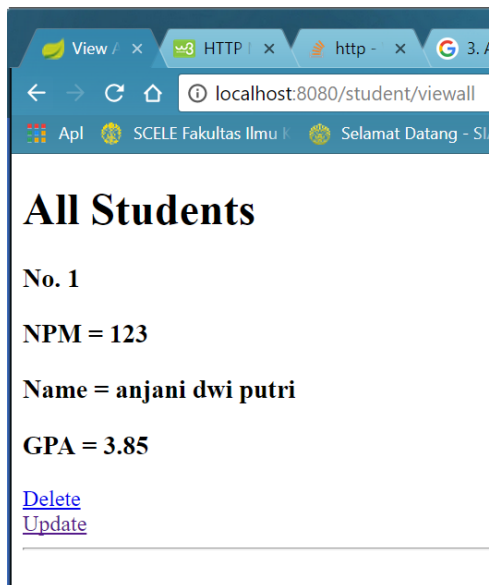
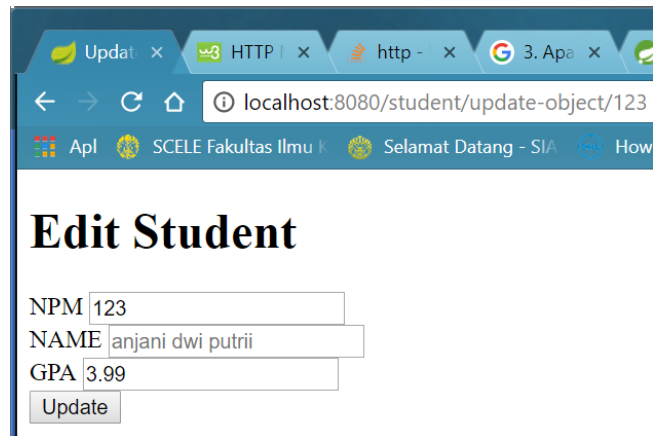
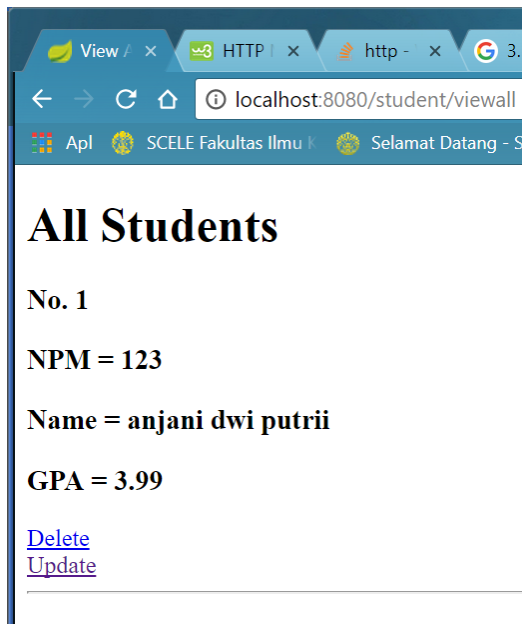
2. Menambahkan `th:field="*{[nama_field]}"` pada setiap input

```
view.html
1<!DOCTYPE html>
2<html xmlns:th="http://www.thymeleaf.org">
3  <head>
4    <title>View Student by NPM</title>
5  </head>
6  <body>
7    <form th:object="${student}">
8      <h3 th:text="'NPM = ' + ${student.npm}" th:field="*{npm}">Student NPM</h3>
9      <h3 th:text="'Name = ' + ${student.name}" th:field="*{name}">Student Name</h3>
10     <h3 th:text="'GPA = ' + ${student.gpa}" th:field="*{gpa}">Student GPA</h3>
11   </form>
12 </body>
13</html>
14
```

3. Ubah method `updateSubmit` pada `StudentController` yang hanya menerima parameter berupa `StudentModel`

```
StudentController.java
130 @GetMapping("/student/update-object/{npm}")
131 public String updateObjectForm(Model model, @PathVariable(value = "npm") String npm) {
132     StudentModel students = studentDAO.selectStudent(npm);
133     model.addAttribute("students", students);
134     if (students == null){
135         return "not-found";
136     } else {
137         return "form-update-object";
138     }
139 }
140 }
141
142 @PostMapping("/student/update-project/submit")
143 public String updateObjectSubmit(@ModelAttribute StudentModel studentmodel ) {
144
145     return "success-update";
146 }
147 }
```

4. Tes lagi aplikasi Anda



1. Jelaskan apa saja hal yang Anda pelajari dari tutorial ini.

Pada tutorial ini, kita akan menggunakan database untuk memasukkan data yang di input lalu kita dapat merubah data tersebut. Data yang input adalah nama, npm, dan pa sedangkan data yang dapat diubah adalah nama dan gpa. Untuk dapat merubah data tersebut maka kita harus membuat method update yang akan di jelaskan pada pertanyaan berikutnya, lalu selain update kita juga dapat menghapus data yang telah di input.

2. Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan

Pada soal dijelaskan terlebih dahulu untuk memberikan beberapa tambahan code pada viewall.html. Sebelum membuat code di controller, dijelaskan untuk membuat method deleteStudent pada kelas StudentMapper dengan menambahkan SQL delete, lalu membuat method delete student pada StudentController.java, pada kelas tersebut saya mengimplementasikan method menerima parameter npm. Lalu mengecek npm yang ada di database apakah ada atau tidak dengan menggunakan selectStudent, jika ada maka akan menampilkan delete.html jika tidak maka akan menampilkan deleteFailed.html

Lalu implementasikan method deleteStudent pada kelas studentServiceDatabase.java dengan memanggil method deleteStudent pada kelas studentController yang berfungsi untuk menghubungkan ke database untuk menghapus data yang di inginkan.

3. Method yang Anda buat pada Latihan Menambahkan Update, jelaskan

Pada soal terlebih dahulu menambahkan method updateStudent pada kelas kelas studentMapper dengan menerima method StudentModel student, lalu menambahkan query SQL pada StudentMapper dengan anotasi @Update. Lalu menambahkan method updateStudent pada interface StudentService dan mengimplementasikan di kelas StudentServiceDatabase. Lalu setelah itu membuat method update pada kelas StudentController.

Pada kelas StudentController saya membuat 2 method update, yaitu update dimana method ini akan menerima parameter npm lalu di cek pada database apakah data npm yang diminta ada atau tidak, jika ada maka akan di *assign* ke form-update.html. pada file tersebut user akan memperbaharui, lalu akan dikembalikan ke method updateSubmit yang ada di kelas StudentController, method ini akan melakukan pengecekan terhadap input user, input user berupa npm, nama, dan gpa. Lalu jika di cek semua terisi maka method ini akan melakukan update dan dimasukkan ke dalam database.

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

Hal yang dilakukan method ini tidak jauh berbeda dengan method update yang menerima parameter StudentModel student, hanya saja pada method ini pada form-update-object.html ditambahkan `"th:placeholder="${student.npm}" "`. lalu perbedaan kedua terletak

pada method `updateObjectSubmit`, method ini hanya menerima parameter berupa `StudentModel student` dan langsung memperbaharui di database.

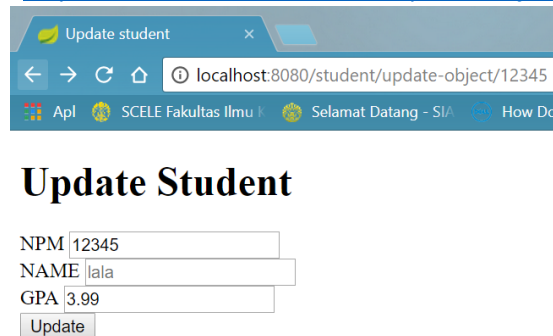
Pertanyaan Beberapa pertanyaan yang perlu Anda jawab:

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.

Validasi diperlukan, karena input user tidak boleh ada yang kosong. Jika salah satu input user ada yang kosong maka akan mengembalikan halaman error. Berikut contohnya

1. Berikut tampilan halaman <http://localhost:8080/student/update-object/12345> sebelum diubah



Update student

localhost:8080/student/update-object/12345

Apl SCELE Fakultas Ilmu k Selamat Datang - SIA How Do I

Update Student

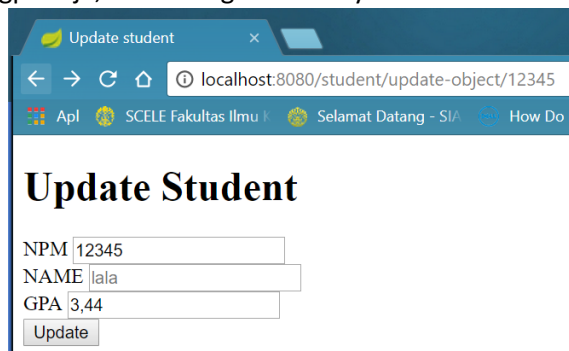
NPM 12345

NAME lala

GPA 3.99

Update

2. Saya akan mengupdate gpa saja, tidak dengan namanya



Update student

localhost:8080/student/update-object/12345

Apl SCELE Fakultas Ilmu k Selamat Datang - SIA How Do I

Update Student

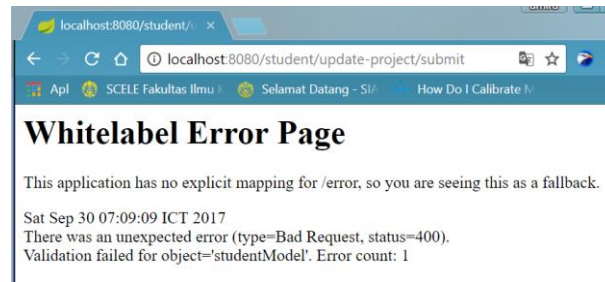
NPM 12345

NAME lala

GPA 3,44

Update

3. Berikut hasil keluaran jika saya meng-klik "Update"



4. Berikut code pada update-project-object.html, tidak ada required yang diberikan

```
StudentServiceDatabase.java StudentModel.java StudentMapper.java StudentService.java StudentController.java form-update.html form-update-object.html
1<!DOCTYPE HTML>
2<html xmlns:th="http://www.thymeleaf.org">
3<head>
4
5<title>Update student</title>
6</head>
7
8<body>
9
10<h1 class="page-header">Update Student</h1>
11
12<form action="/student/update-project/submit" method="post" th:object="${student}">
13<div>
14<label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${students.npm}" th:placeholder="${students.npm}"/>
15</div>
16
17<div>
18<label for="name">NAME</label> <input type="text" name="name" th:placeholder="${students.name}"/>
19</div>
20
21<div>
22<label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${students.gpa}" th:placeholder="${students.gpa}"/>
23</div>
24<div>
25<button type="submit" name="action" value="save">Update</button>
26</div>
27</form>
28
29</body>
30</html>
```

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method?

Menurut saya, form submit digunakan untuk memperbaharui atau menambahkan data maka menggunakan POST, form submit tidak mendapatkan data dari suatu database yang menggunakan method GET.

Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

Tidak diperlukan.

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

Kalau menurut saya tidak bisa. Contohnya ada pada method update. Pada controller saya membuat 2 method yang belaku sebagai POST dan GET, berikut contohnya

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

@PostMapping("/student/update-project/submit")
public String updateObjectSubmit(@ModelAttribute StudentModel studentmodel ) {

    return "success-update";
}
```

Pada code pertama method akan meng-GET npm dari url, lalu method kedua akan melakukan POST ke database kembali.