

Nama : Shafa Maharani Pradinar

NPM : 1606955031

Ada beberapa hal yang saya pelajari pada tutorial 4, yaitu cara menghubungkan project pada SpringBoot dengan database MySQL, latihan menambahkan Delete, latihan menambahkan Update dan latihan menggunakan Object sebagai parameter. Untuk project pada Springboot dengan MySQL bisa menggunakan Lombok atau MyBatis yang berguna untuk melakukan koneksi dan generate query dengan helper annotation. Selain itu, diharuskan memiliki XAMPP sebagai server untuk local dalam pembuatan database dengan MySQL. Jika diperhatikan pada folder **src/main/resource**, ada sebuah file bernama **application.properties**. File ini berisi konfigurasi aplikasi. File ini kosong awalnya dan dapat ditambahkan konfigurasi yang diinginkan. Konfigurasi yang ditambahkan bisa bervariasi. Mulai dari port, database config, sampai API Key dari third-party service yang digunakan. Berikut tampilan **application.properties**.

```
application.properties
1# port aplikasi Anda diakses, pada contoh ini aplikasi Anda berada di localhost:1234
2server.port=1234
3
4# konfigurasi untuk koneksi mysql
5spring.datasource.platform=mysql
6spring.datasource.driver-class-name=com.mysql.jdbc.Driver
7
8# sesuaikan NAMA_DATABASE dengan nama database Anda, contoh: jdbc:mysql://localhost:3306/tutorial4db
9spring.datasource.url=jdbc:mysql://localhost:3306/[NAMA_DATABASE]
10
11# sesuaikan dengan username phpmyadmin Anda,
12spring.datasource.username=[USERNAME_PHPMYADMIN]
13
14# sesuaikan dengan password phpmyadmin Anda,
15spring.datasource.password=[PASSWORD_PHPMYADMIN]
16
17spring.datasource.initialize=false
```

Lalu mempelajari cara memberikan argumen log pada project yang dibuat. Log ini berguna untuk melakukan debugging pada Spring Boot. Untuk memberikan argumen log menggunakan **Slf4j** yang berada di library eksternal Lombok. Penggunaannya bisa dengan cara menambahkan anotasi **@Slf4j** diatas kelas yang ingin diberikan log.

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</a><br/>
```

Tampilan dibawah merupakan tampilan menambahkan code untuk memanggil controller student dan method delete yang akan ditampilkan dan dijalankan fungsinya pada view all.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>View All Students</title>
</head>
<body>
<h1>All Students</h1>
<div th:each="student, iterationStatus: ${students}">
<a th:href="/student/delete/" + ${student.npm}> Delete Data</a><br/>
<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>
<hr/>
</div>
</body>
</html>
```

3. Tambahkan method deleteStudent yang ada di class StudentMapper

a. Tambahkan method delete student yang menerima parameter NPM. Jadi, kita melakukan delete data berdasarkan primary key yang ada di table Student yaitu field NPM.

b. Tambahkan annotation delete di atas dan SQL untuk menghapus data berdasarkan NPM. Syntax SQL menjelaskan bahwa data akan dihapus dari table Student, dimana data yang akan dihapus adalah data dengan NPM yang dimaksud.

```
@Delete("[LENGKAPI]")
```

```
package com.example.dao;

import java.util.List;

import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Delete;

import com.example.model.StudentModel;

@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 (String npm);
}
```

Tampilan diatas merupakan query untuk select, insert dan delete pada database.

4. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

a. Tambahkan log untuk method tersebut dengan cara menambahkan

```
log.info ("student " + npm + " deleted");
```



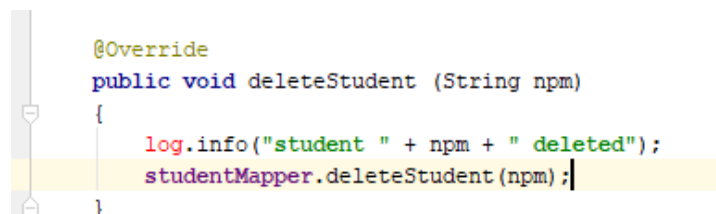
```

19
20
21  @Override
22  public StudentModel selectStudent (String npm)
23  {
24      log.info ("select student with npm {}", npm);
25      return studentMapper.selectStudent (npm);
26  }
27
28
29  @Override
30  public List<StudentModel> selectAllStudents ()
31  {
32      log.info ("select all students");
33      return studentMapper.selectAllStudents ();
34  }
35
36
37  @Override
38  public void addStudent (StudentModel student) { studentMapper.addStudent (student); }
39
40
41  @Override
42  public void deleteStudent (String npm)
43  {
44      log.info("student " + npm + " deleted");
45  }
46
47
48
49
50 }

```

Log berguna untuk melakukan debugging pada Spring Boot. Pada method tersebut terdapat parameter NPM yang bertipe data String dan akan menerima masukan berupa String juga. Lalu di method deleteStudent terdapat log sebagai info dan memanggil variable NPM dan value nya akan sesuai dengan masukan parameter.

b. Panggil method delete student yang ada di Student Mapper



```

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

```

Selanjutnya memanggil method deleteStudent yang sudah dibuat di StudentMapper dengan mengisi parameter NPM di method tersebut. Value akan sesuai isinya dengan masukan parameter NPM pada method deleteStudent yang ada pada StudentServiceDatabase. Pemanggilan method tersebut mengartikan bahwa query SQL untuk menghapus data pada StudentMapper akan dijalankan.

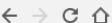
5. Lengkapi method delete pada class StudentController

- Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found.
- Jika berhasil delete student dan tampilkan view delete

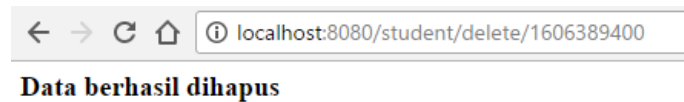
```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);
    if (student != null) {
        studentDAO.deleteStudent(npm);
        return "delete";
    } else {
        return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

6. Jalankan Spring Boot app dan lakukan beberapa insert

7. Contoh tampilan View All. Pada awalnya terdapat 4 data Student.

 localhost:8080/student/viewall	
All Students	
Delete Data	
No. 1	
NPM = 1606389400	
Name = Akhda	
GPA = 3.99	
Delete Data	
No. 2	
NPM = 160682893	
Name = Hendi	
GPA = 3.85	
Delete Data	
No. 3	
NPM = 1606954981	
Name = Gita	
GPA = 3.99	
Delete Data	
No. 4	
NPM = 1606955031	
Name = shafa	

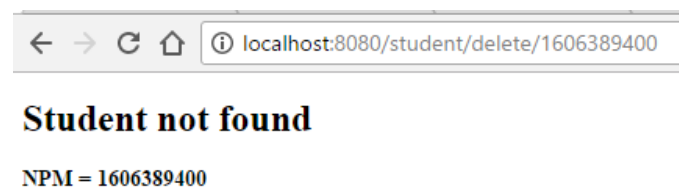
8. Lalu data dengan NPM 1606389400 dihapus. Berikut tampilan setelah data dengan NPM 1606389400 dihapus.



Tampilan dibawah menunjukkan bahwa data dengan NPM 1606389400 berhasil dihapus.



9. Contoh tampilan jika dilakukan delete NPM 1606389400 untuk kedua kalinya. Data tidak ditemukan di database karena memang sudah terhapus.



Latihan Menambahkan Update

Selama ini Anda masih menggunakan method GET untuk melakukan masukan. Pada update kali ini, Anda akan mencoba menggunakan method POST.

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

```
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Update;

import com.example.model.StudentModel;

@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 (String npm);

    @Update("update student set name = #{name}, gpa = #{gpa} where npm = #{npm} ")
    void updateStudent (String npm);
}
```

Tambahkan annotation Update di atas dan query SQL untuk mengupdate data nama dan GPA berdasarkan NPM.

2. Tambahkan method updateStudent pada interface StudentService

3. Tambahkan implementasi method updateStudent pada class StudentServiceDatabase.

Jangan lupa tambahkan logging pada method ini.

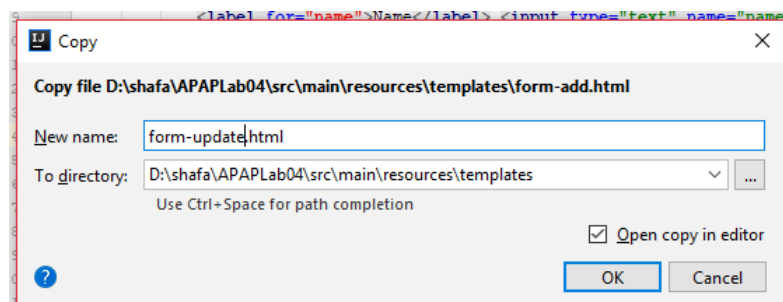
4. Tambahkan link Update Data pada viewall.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>View All Students</title>
</head>
<body>
<h1>All Students</h1>

<div th:each="student, iterationStatus: ${students}">
  <a th:href="/student/delete/" + ${student.npm}> Delete Data</a><br/>
  <a th:href="/student/update/" + ${student.npm}> Update Data</a><br/>
  <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>
  <hr/>
</div>
</body>
</html>
```

Menambahkan code untuk memanggil controller student dan method update yang akan ditampilkan dan dijalankan fungsinya pada view all.

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



- Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.


```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Edit Student</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
  <h1 class="page-header">Edit Student</h1>

  <form action="/student/update/submit" method="get">
    <div>
      <label for="npm">NPM</label> <input type="text" name="npm" />
    </div>
    <div>
      <label for="name">Name</label> <input type="text" name="name" />
    </div>
    <div>
      <label for="gpa">GPA</label> <input type="text" name="gpa" />
    </div>
    <div>
      <button type="submit" name="action" value="save">Save</button>
    </div>
  </form>
</body>
```

b. Ubah action form menjadi /student/update/submit

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Edit Student</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
  <h1 class="page-header">Edit Student</h1>

  <form action="/student/update/submit" method="get">
    <div>
      <label for="npm">NPM</label> <input type="text" name="npm" />
    </div>
    <div>
      <label for="name">Name</label> <input type="text" name="name" />
    </div>
    <div>
      <label for="gpa">GPA</label> <input type="text" name="gpa" />
    </div>
    <div>
      <button type="submit" name="action" value="save">Save</button>
    </div>
  </form>
</body>
```

```
<head>

<title>Edit Student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

<h1 class="page-header">Edit Student</h1>

<form action="/student/update/submit" method="get">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" />
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" />
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" />
  </div>
  <div>
    <button type="submit" name="action" value="save">Update</button>
  </div>
</form>
</body>
</html>
```

c. Ubah method menjadi post

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>

<title>Edit Student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

<h1 class="page-header">Edit Student</h1>

<form action="/student/update/submit" method="post">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" />
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" />
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" />
  </div>
  <div>
    <button type="submit" name="action" value="save">Update</button>
  </div>
</form>
</body>
</html>
```

Method post berguna untuk mengirimkan data atau nilai langsung ke action untuk ditampilkan, tanpa menampilkan pada URL.

d. Untuk input npm ubah menjadi

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>

<title>Edit Student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

<h1 class="page-header">Edit Student</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">Update</button>
  </div>
</form>

</body>
```

Readonly berfungsi agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada.

Input name ubah menjadi

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>

<title>Edit Student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

<h1 class="page-header">Edit Student</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">Update</button>
  </div>
</form>

</body>
```

th:value digunakan untuk mengisi input dengan name student yang sudah ada.

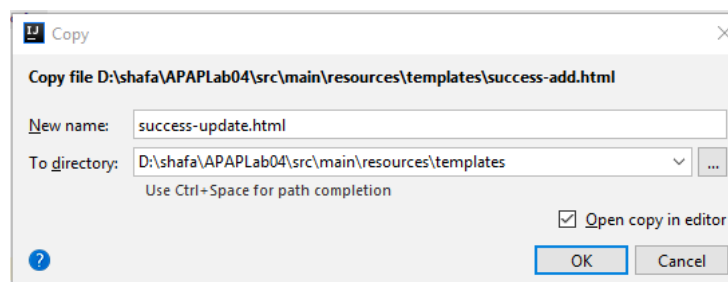
Lakukan hal yang sama untuk GPA.

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Edit Student</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
  <h1 class="page-header">Edit Student</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">Update</button>
    </div>
  </form>
</body>
```

th:value digunakan untuk mengisi input dengan gpa student yang sudah ada.

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

a. Ubah keterangan seperlunya



```
<html>
  <head>
    <title>Edit</title>
  </head>
  <body>
    <h2>Data berhasil diedit</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 ( "student", student);
        return "form-update";
    }else{
        return "not-found";
    }
}
```

8. Tambahkan method updateSubmit pada class StudentController

- Karena menggunakan post method maka request mappingnya adalah sebagai berikut:

```
@RequestMapping("/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 = newStudentModel(npm, name, gpa);
    studentDAO.updateStudent (student);
    return "success-update";
}
```

b. Header methodnya adalah sebagai berikut:

```
@RequestMapping("/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 = newStudentModel(npm, name, gpa);
    studentDAO.updateStudent (student);
    return "success-update";
}
```

c. Lengkapi method dengan memanggil method update Student dan kembalikan view success-update

```
@RequestMapping("/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 = newStudentModel(npm, name, gpa);
    studentDAO.updateStudent (student);
    return "success-update";
}
```

9. Jalankan Spring Boot dan coba test program Anda

Pada awalnya terdapat 3 data, lalu data dengan student name Gita dihapus



← → ↻ 🏠 ⓘ localhost:8080/student/viewall

All Students

[Delete Data](#)
[Update Data](#)

No. 1

NPM = 160682893

Name = Hendi

GPA = 3.85

[Delete Data](#)
[Update Data](#)

No. 2

NPM = 1606954981

Name = Gita

GPA = 3.99

[Delete Data](#)
[Update Data](#)

No. 3

NPM = 1606955031

Name = shafa

GPA = 3.69

Berikut tampilan bahwa data berhasil dihapus

← → ↻ 🏠 ⓘ localhost:8080/student/delete/1606954981

Data berhasil dihapus

Dan jumlah data menjadi 2

← → ↻ 🏠 ⓘ localhost:8080/student/viewall

All Students

[Delete Data](#)
[Update Data](#)

No. 1

NPM = 160682893

Name = Hendi

GPA = 3.85

[Delete Data](#)
[Update Data](#)

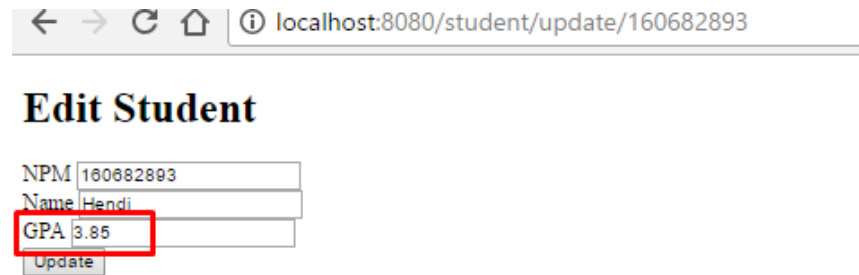
No. 2

NPM = 1606955031

Name = shafa

GPA = 3.69

Untuk edit data student, misal saya mengedit data GPA Hendi dari 3.85 menjadi 3.99



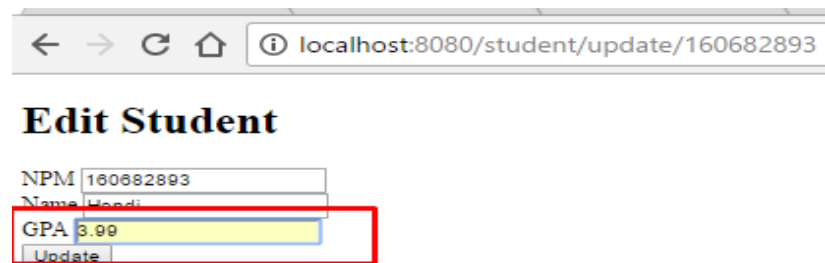
← → ↻ 🏠 ⓘ localhost:8080/student/update/160682893

Edit Student

NPM

Name

GPA



← → ↻ 🏠 ⓘ localhost:8080/student/update/160682893

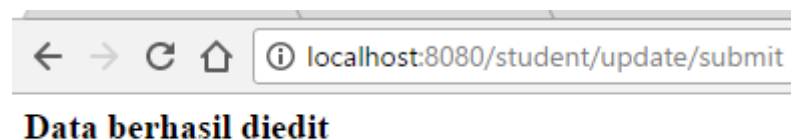
Edit Student

NPM

Name

GPA

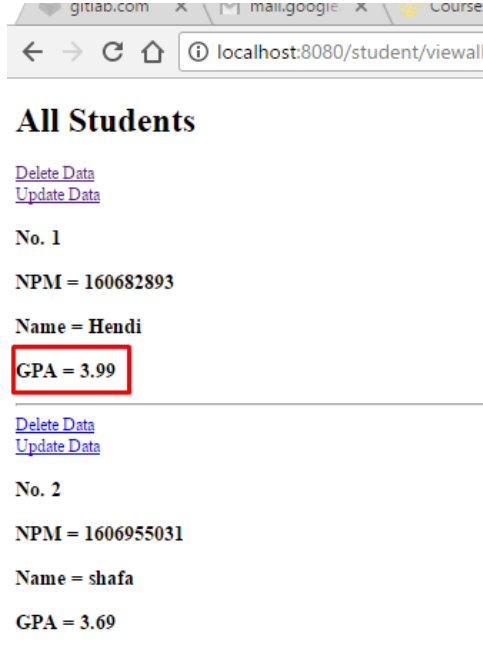
Setelah data berhasil diedit, muncul seperti tampilan dibawah ini



← → ↻ 🏠 ⓘ localhost:8080/student/update/submit

Data berhasil diedit

Data dengan nama Hendi berhasil diubah GPA nya menjadi 3.99



All Students

[Delete Data](#)
[Update Data](#)

No. 1

NPM = 160682893

Name = Hendi

GPA = 3.99

[Delete Data](#)
[Update Data](#)

No. 2

NPM = 1606955031

Name = shafa

GPA = 3.69

Latihan Menggunakan Object Sebagai Parameter

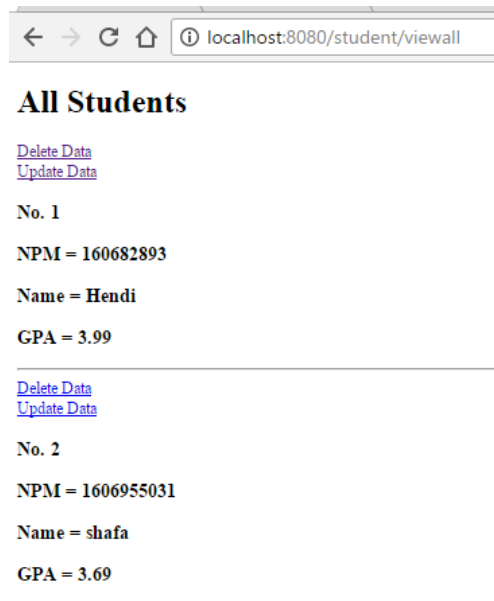
Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga adabanyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakanRequestParam.

Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

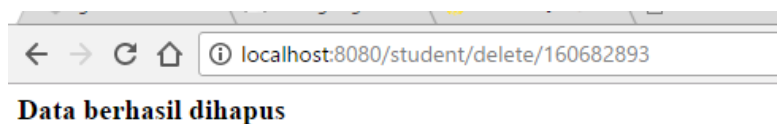
```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@ModelAttribute StudentModel student)
{
    studentDAO.updateStudent (student);
    return "success-update";
}
```

Tes aplikasi

Pada awalnya terdapat 2 data atas nama Hendi dan Shafa.



Lalu data atas nama Hendi dengan NPM 160682893 dihapus



Data yang tersisa hanya ada satu yaitu atas nama Shafa

All Students

[Delete Data](#)

[Update Data](#)

No. 1

NPM = 1606955031

Name = shafa

GPA = 3.69

Selanjutnya, data atas nama Shafa akan diubah namanya menjadi Shafa Maharani seperti pada tampilan dibawah

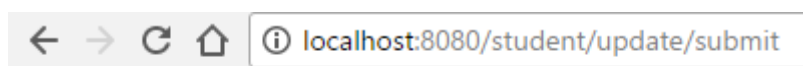
Edit Student

NPM
Name
GPA

Edit Student

NPM
Name
GPA

Setelah klik button update, muncul pemberitahuan bahwa data berhasil diedit



Data berhasil diedit



Lalu, data nama berhasil diubah menjadi Shafa Maharani.

← → ↻ 🏠 ⓘ localhost:8080/student/viewall

All Students

[Delete Data](#)
[Update Data](#)

No. 1

NPM = 1606955031

Name = shafa maharani

GPA = 3.69

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 seperti jika menggunakan RequestParam? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Jawaban :

Cara melakukan validasi yaitu dengan memanfaatkan model yang telah dibuat. Model adalah cerminan struktur sebuah table pada database. Pada model bisa ditambahkan standar validation annotations seperti @NotNull yang tidak akan mengijinkan sebuah attribute bernilai null. Kemudian pada controller yang menerima submit ditambahkan @Valid pada parameter Object dan tambahkan parameter Object bindingResult. Lalu dibagian body tambahkan kondisi untuk melakukan validasi dengan menggunakan code "bindingResult.hasErrors()" jika hasil validasi error maka akan kembali ke halaman form dan jika validasi berhasil maka akan melakukan aksi selanjutnya.

Validasi diperlukan karena form berhubungan langsung dengan database. Jika pada struktur table database terdapat field not null maka harus dipastikan bahwa pada form telah diinputkan nilai untuk field tersebut agar tidak terjadi error atau ketidaksesuaian data.



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?

Jawaban :

Form submit biasanya berisi nilai yang cukup panjang untuk setiap fieldnya sehingga sangat tidak efisien jika menggunakan method GET dimana akan muncul nilai yang diinputkan pada URL. Selain itu berikut beberapa kelebihan jika menggunakan method POST:

1. Lebih aman karena data yang dikirim tidak terlihat
2. Dapat mengirim berbagai jenis data seperti gambar, file, dll, tidak harus teks
3. Dapat mengirim data dalam jumlah besar.

Perlu penanganan yang berbeda pada header method di controller dengan menambahkan `method = RequestMethod.POST` pada `RequestMapping`.

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

Jawaban :

Tidak bisa, karena method POST dan GET memiliki fungsi yang agak sedikit berbeda yaitu method POST mengirimkan data secara langsung. Maksudnya adalah mengirimkan sebuah data atau nilai / value langsung ke file lain. Pemakaian method POST ini digunakan untuk mengirimkan data yang penting / kredensial dan data yang orang lain tidak boleh tau atau secret data, seperti password, dan sebagainya. Itulah yang dimaksud dengan "Mengirimkan data secara langsung". Sedangkan method GET mengirimkan data tidak langsung. Maksudnya adalah kalau kita menggunakan method GET dalam membuat formulir online, pasti ketika mengisi nama, alamat, dan sebagainya pasti data tersebut akan terlihat di URL.