



Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

NAMA : RIANDANI GITA LARASATI
NPM : 1606954981
KELAS : SI EKSTENSI

Latihan Menambahkan Delete

Pada tutorial 4 ini saya belajar bagaimana mengimplementasikan method delete dengan menggunakan database. Untuk mendukung implementasi tersebut, saya diharuskan untuk install Lombok sebagai library eksternal dan merupakan helper annotation pada project ini. Selain itu, saya memakai MySQL untuk database-nya, lalu memastikan XAMPP sudah terinstall dan dapat dijalankan.

Setelah Lombok dan XAMPP sudah di-install, maka yang saya lakukan selanjutnya adalah membuat database pada phpmyadmin. Database yang saya buat saya beri nama db_apap dan membuat table yang dinamakan student beserta field-field sesuai dengan ketentuan tutorial project ini.

Setelah itu, saya membuat project baru pada springboot dan import template yang telah disediakan. Pada template tersebut belum terkoneksi dengan database. Berikut ini akan dijelaskan langkah mengintegrasikan ke database.

```
# Data Source
#port aplikasi anda diakses, di contoh ini aplikasi berada di localhost:1234
server.port=8080

#konfigurasi untuk koneksi mysql
spring.datasource.platform=mysql
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

#sesuaikan nama_database dengan nama database yg dipunya
spring.datasource.url=jdbc:mysql://localhost:3306/db_apap

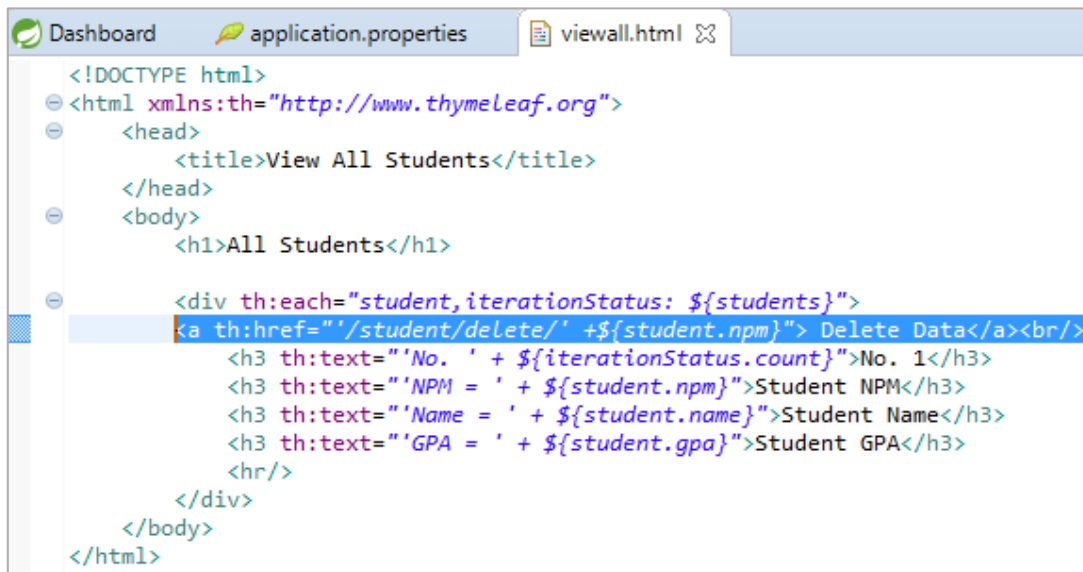
#sesuaikan dengan username phpmyadmin,
spring.datasource.username=root

#sesuaikan dengan password phpmyadmin,
spring.datasource.password=

spring.datasource.initialize=false
```

Melengkapi code seperti di atas pada **application.properties**. Server port 8080 sesuai dengan port yang terdapat pada XAMPP. Untuk setiap code sudah dijelaskan pada comment seperti diatas. Localhost:3306/db_apap sesuai dengan database yang sudah saya buat sebelumnya.

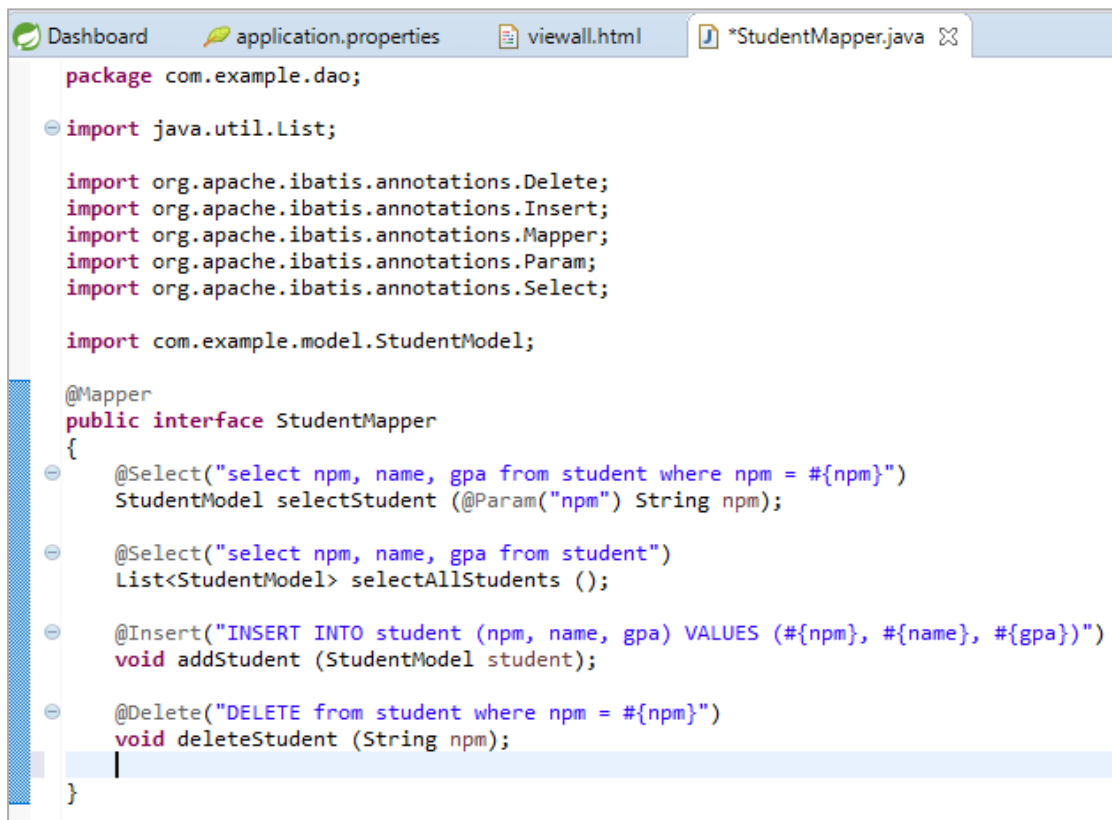
1. Menambahkan code untuk memanggil method delete yang akan ditampilkan dan dijalankan fungsinya pada view all.



```
<!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>
```

2. Menambahkan method **deleteStudent** yang ada di class **StudentMapper**
 - a. Menambahkan method **deletestudent** yang menerima parameter npm, yaitu kita akan melakukan delete berdasarkan primary key yaitu field npm yang akan di-select. Npm tersebut akan mewakili field yang lainnya.
 - b. Menambahkan annotation delete di atas dan SQL untuk menghapus data berdasarkan npm. Query SQL tersebut menjelaskan bahwa data akan dihapus dari tabel student dimana data yang akan dihapus adalah data dengan npm yang dikehendaki.



```
package com.example.dao;

import java.util.List;

import org.apache.ibatis.annotations.Delete;
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 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);
}
```

3. Melengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**
 - a. Menambahkan log untuk method tersebut dengan cara menambahkan code seperti berikut ini.

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

Method tersebut disertai dengan parameter npm bertipe data String, yaitu akan menerima masukan npm yang bertipe data String juga. Di dalam method tersebut terdapat log sebagai info, dan memanggil variable npm, yang nanti value-nya akan sesuai dengan masukan paramater.

- b. Panggil method **deleteStudent** yang ada di **StudentMapper**

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

Setelah itu, memanggil method **deleteStudent** yang telah dibuat pada **StudentMapper** dengan mengisi parameter npm pada method tersebut, value akan sesuai dengan masukan parameter npm pada method **deleteStudent** yang ada pada **StudentServiceDatabase**. Dengan memanggil method tersebut, maka query SQL untuk menghapus data pada **StudentMapper** akan dijalankan.

4. Melengkapi method **delete** pada class **StudentController**
 - a. Menambahkan validasi agar jika npm mahasiswa tidak ditemukan tampilkan view **not-found**. Hal tersebut dapat diimplementasikan dengan cara, membuat method seperti screenshot di bawah ini. **@RequestMapping** diarahkan untuk menuju controller student dan method **delete** berdasarkan parameter npm yang dikehendaki.
 - b. Jika berhasil delete student dan tampilkan view **delete**. Maka saya membuat kondisi pada body method seperti berikut ini.

```
@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);
        return "delete";
    }else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

Pertama-tama menginstansiasi **StudentModel** dengan object student untuk method selectStudent yang memiliki parameter npm. Lalu, dengan menggunakan object tersebut, di cek jika value npm pada object student tersebut tidak null, maka akan membaca **@model.Attribute** yaitu anotasi yang mengikat parameter (object student) yang mengembalikan nilai ke atribut model dan kemudian memaparkannya ke tampilan web, yaitu mengembalikan view delete untuk tampilan pada web browser.

Kemudian, apabila value npm pada object student adalah null maka **@model.Attribute** akan mengikat parameter npm, lalu mengembalikan view not-found.

5. Menjalankan Spring Boot app.

Menambahkan beberapa data:

Data ke-1

The screenshot shows a web browser at the URL `localhost:8080/student/add`. The page title is "Problem Editor". It contains three input fields: "NPM" with the value "12345", "Name" with the value "Gita", and "GPA" with the value "3.99". There is a "Save" button below the fields. Below the form, a message "Data berhasil ditambahkan" (Data successfully added) is displayed. The browser's address bar shows the full URL: `localhost:8080/student/add/submit?npm=12345&name=Gita&gpa=3.99&action=save`.

Data ke-2

The screenshot shows a web browser at the URL `localhost:8080/student/add`. The page title is "Problem Editor". It contains three input fields: "NPM" with the value "12346", "Name" with the value "Bimo", and "GPA" with the value "3.95". There is a "Save" button below the fields. Below the form, a message "Data berhasil ditambahkan" (Data successfully added) is displayed. The browser's address bar shows the full URL: `localhost:8080/student/add/submit?npm=12346&name=Bimo&gpa=3.95&action=save`.

Menampilkan semua data:

← → ↻ localhost:8080/student/viewall	
All Students	
Delete Data	
No. 1	
NPM = 12345	
Name = Gita	
GPA = 3.99	
Delete Data	
No. 2	
NPM = 12346	
Name = Bimo	
GPA = 3.95	

Menghapus salah satu data, yaitu menghapus data dengan npm 12346:

← → ↻ localhost:8080/student/delete/12346
Data berhasil dihapus

Data yang tersisa tinggal 1 data:

← → ↻ localhost:8080/student/viewall	
All Students	
Delete Data	
No. 1	
NPM = 12345	
Name = Gita	
GPA = 3.99	

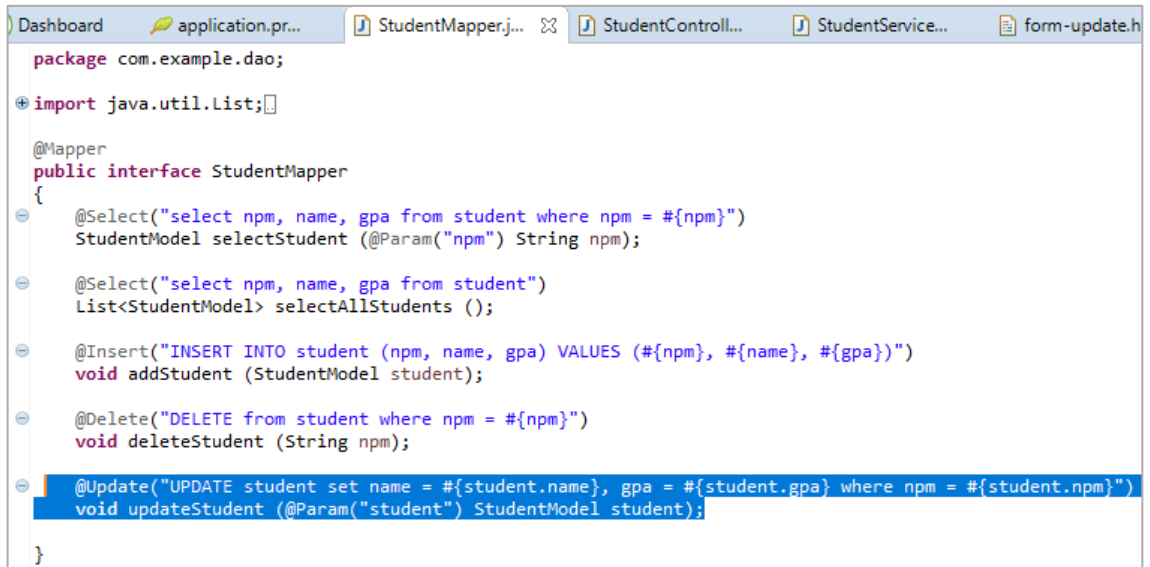
Contoh tampilan jika dilakukan delete NPM 123456 yang kedua kalinya:

← → ↻ localhost:8080/student/delete/12346
Student not found
NPM = 12346

Data dengan npm **12346** sudah tidak tersimpan lagi di database sehingga akan masuk ke kondisi **else** yaitu mengembalikan view **not-found**.

Latihan Menambahkan Update

1. Menambahkan method **updateStudent** pada class **StudentMapper** dengan parameter **StudentModel student** serta dengan anotasi **@Update**. Setelah itu menuliskan query SQL untuk mengubah data seperti di bawah ini.



```
package com.example.dao;

import java.util.List;

@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 ();

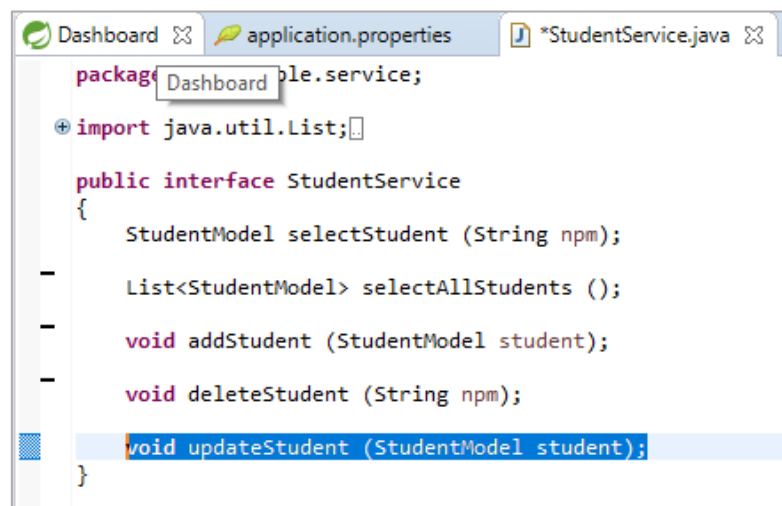
    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

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

    @Update("UPDATE student set name = #{student.name}, gpa = #{student.gpa} where npm = #{student.npm}")
    void updateStudent (@Param("student") StudentModel student);
}
```

Maksud dari query tersebut adalah data pada table student akan diubah, yaitu mengubah data field npm atau gpa atau name atau dapat pula beberapa diantaranya, dimana data tersebut disatukan berdasarkan field unik yaitu npm.

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



```
package com.example.service;

import java.util.List;

public interface StudentService
{
    StudentModel selectStudent (String npm);

    List<StudentModel> selectAllStudents ();

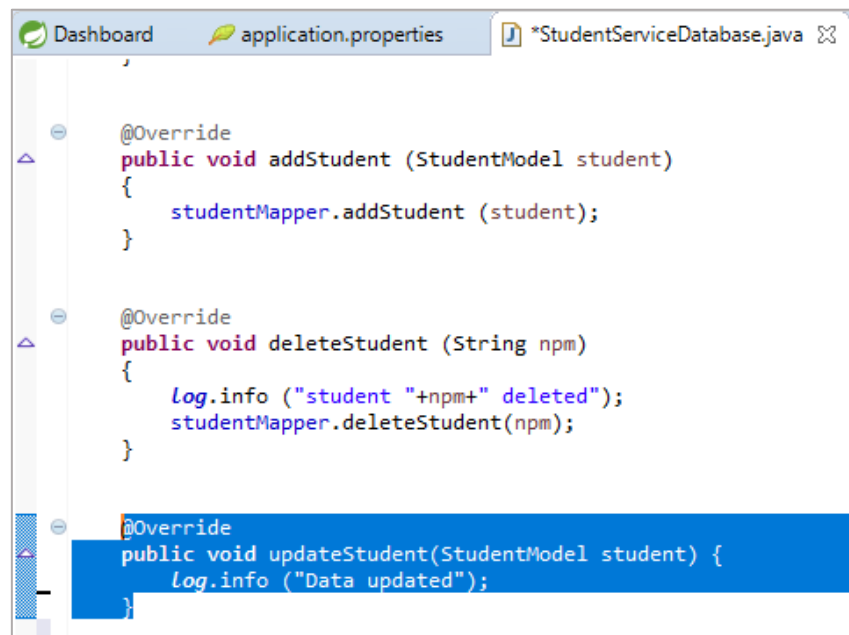
    void addStudent (StudentModel student);

    void deleteStudent (String npm);

    void updateStudent (StudentModel student);
}
```

Memanggil method **updateStudent** yang ada pada **StudentController** yang memiliki parameter student.

3. Menambahkan implementasi method **updateStudent** pada class **StudentServiceDatabase**. Pada method tersebut ditambahkan log untuk pesan bahwa data telah berhasil diubah. Seperti di bawah ini.

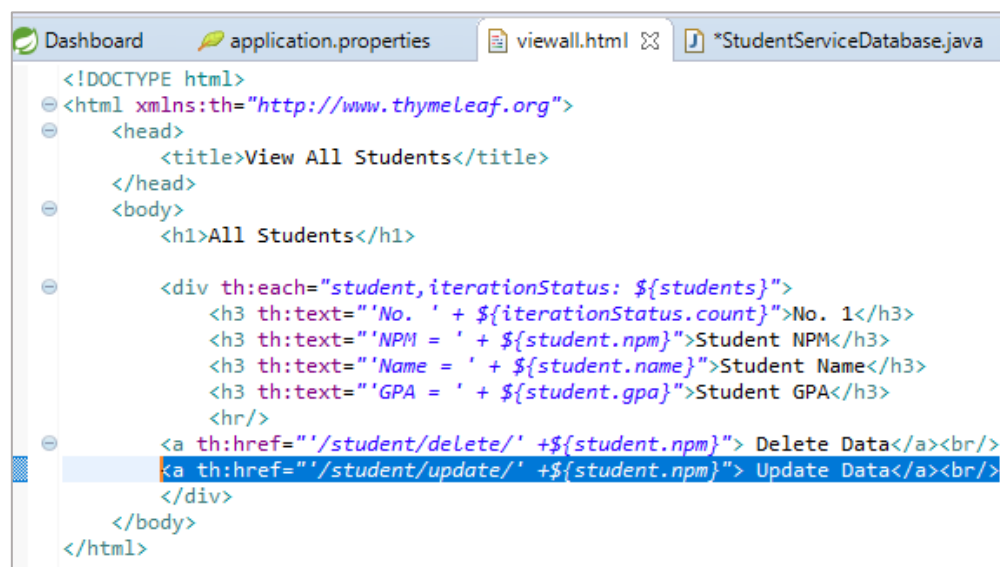


```
Dashboard application.properties *StudentServiceDatabase.java
@Override
public void addStudent (StudentModel student)
{
    studentMapper.addStudent (student);
}

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

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

4. Menambahkan link untuk Update Data pada viewall.html dengan memanggil method **update** yang ada pada **StudentController**.



```
Dashboard application.properties viewall.html *StudentServiceDatabase.java
<!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}">
      <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/>
      <a th:href="/student/delete/" + ${student.npm}"> Delete Data</a><br/>
      <a th:href="/student/update/" + ${student.npm}"> Update Data</a><br/>
    </div>
  </body>
</html>
```

5. Menambahkan view untuk tampilan update dengan Copy view **form-add.html** menjadi **form-update.html**.
 - a. Mengubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll seperti screenshot di bawah ini.

```

<h1 class="page-header">Update Data</h1>

<form action="/student/add/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>

```

- b. Mengubah action form menjadi **/student/update/submit** untuk mengarahkan pada method **updateSubmit** yang ada pada **StudentController**. Seperti screenshot di bawah ini.

```

<body>

  <h1 class="page-header">Update Data</h1>

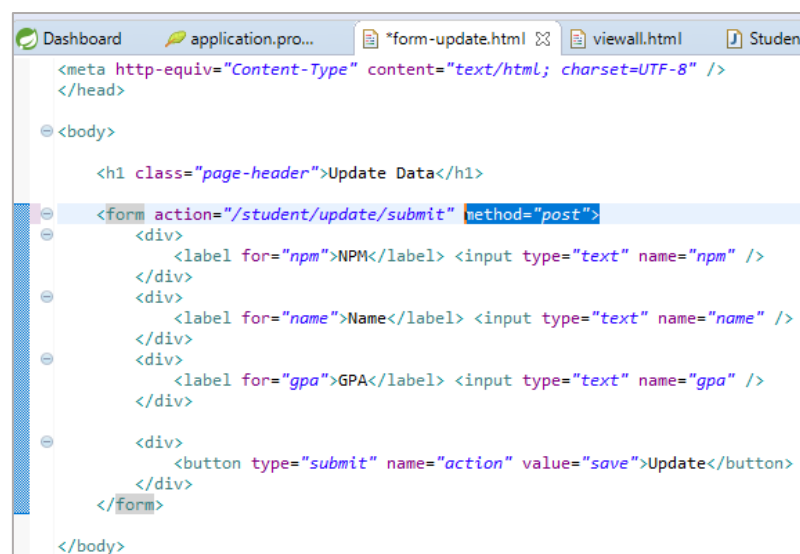
  <form action="/student/update/submit" method="get">
    <div>
      <label for="npm">NPM</label> <input type="text" name="npm" />
    </div>
    <div>
      <label for="name">Name</label> <input type="text" name="name" />
    </div>
    <div>
      <label for="gpa">GPA</label> <input type="text" name="gpa"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

    <div>
      <button type="submit" name="action" value="save">Update</button>
    </div>
  </form>

</body>

```

- c. Mengubah method pada form menjadi post seperti di bawah ini. Post tersebut maksudnya akan mengirimkan value yang diterima untuk langsung diarahkan pada action lalu disimpan, tanpa ditampilkan pada url.



```

Dashboard application.pro... *form-update.html viewall.html Studen
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
  <h1 class="page-header">Update Data</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>

```


- d. Untuk input npm ubah menjadi seperti screenshot di bawah ini. Npm dituliskan true untuk readonly agar npm tidak dapat diubah, sedangkan th:value digunakan untuk mengisi input dengan npm student yang sudah ada.

```
Dashboard application.pro... form-update.html viewall.html StudentServiceD... StudentService.j... Stud

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

<body>

  <h1 class="page-header">Update Data</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" />
    </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>
```

Input name dan gpa diubah menjadi:

```
Dashboard application.pro... *form-update.html viewall.html StudentServiceD... StudentService.j... Stud

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

<body>

  <h1 class="page-header">Update Data</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>
```

6. Copy view **success-add.html** menjadi **success-update.html**
Mengubah keterangan seperlunya, yaitu hanya mengubah pesan yang informatif.

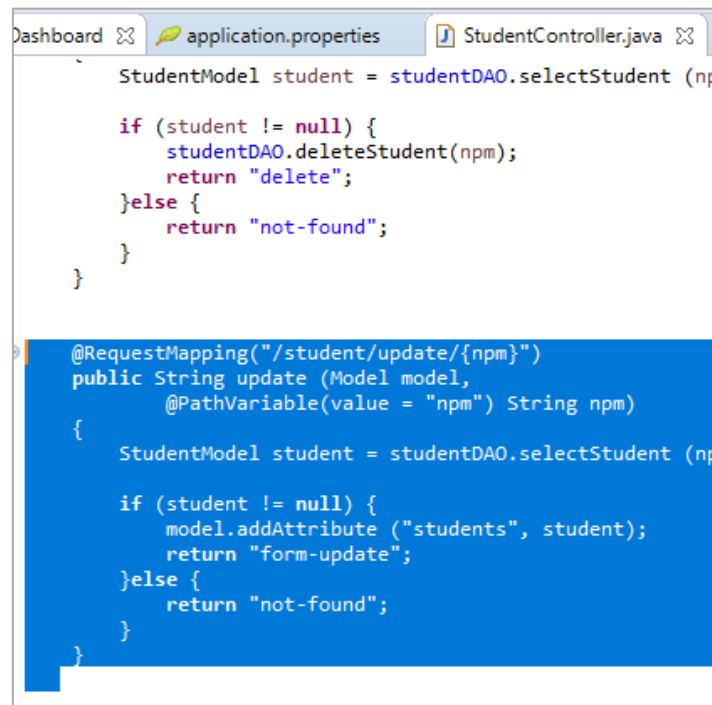
```
Dashboard application.properties success-update.html

<html>
  <head>
    <title>Add</title>
  </head>
  <body>
    <h2>Data berhasil diubah</h2>
  </body>
</html>
```

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

Menuliskan Request mapping ke controller student, method update dengan parameter {npm}. Sama halnya seperti delete, disini dilakukan validasi.

Jika student dengan npm tidak ada tampilkan view **not-found**, jika ada tampilkan view **form-update**. Seperti pada screenshot di bawah ini.



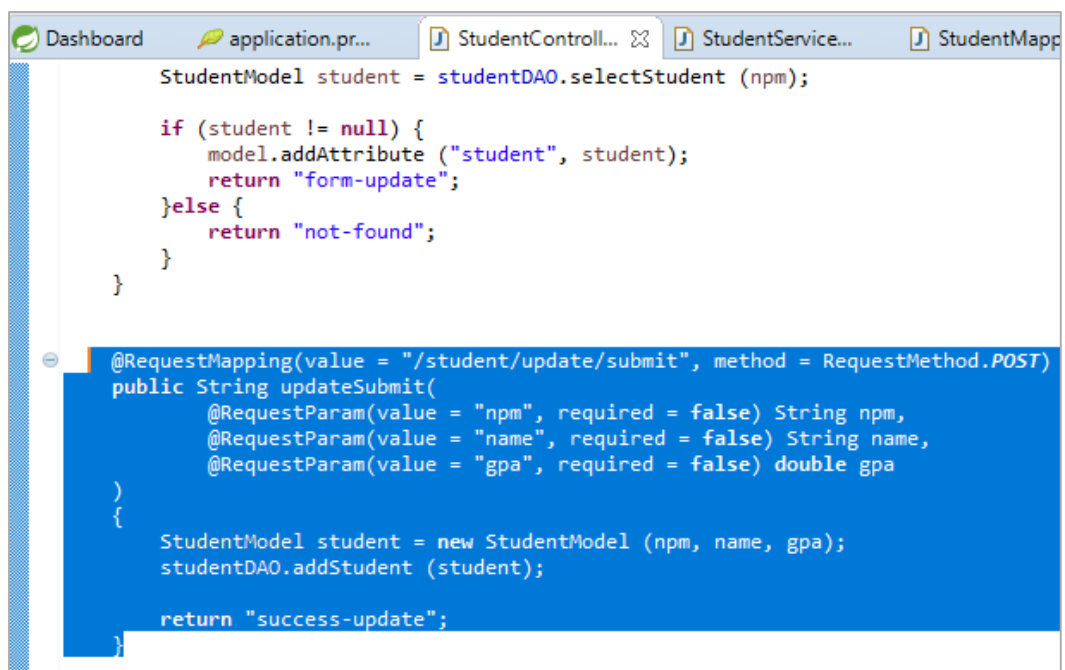
```
StudentModel student = studentDAO.selectStudent (npm);

if (student != null) {
    studentDAO.deleteStudent(npm);
    return "delete";
}else {
    return "not-found";
}

@RequestMapping("/student/update/{npm}")
public String update (Model model,
    @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        model.addAttribute ("students", student);
        return "form-update";
    }else {
        return "not-found";
    }
}
```

8. Menambahkan method **updateSubmit** pada class **StudentController**. Pada paramter method ini dituliskan **@RequestParam** dengan value field npm, name, gpa. Required diisi false menandakan bahwa field tersebut tidak harus diubah.



```
StudentModel student = studentDAO.selectStudent (npm);

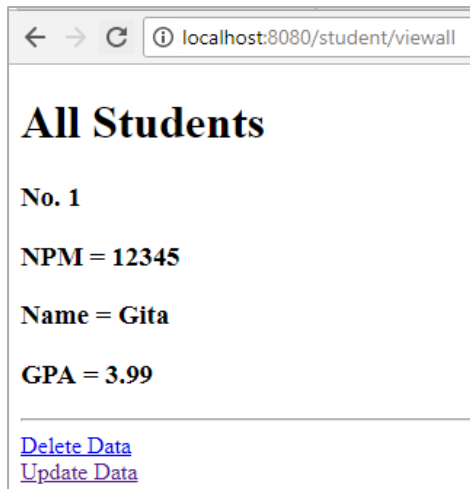
if (student != null) {
    model.addAttribute ("student", student);
    return "form-update";
}else {
    return "not-found";
}

@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.addStudent (student);

    return "success-update";
}
```

9. Jalankan Spring Boot.

Menampilkan semua data:



← → ↻ ⓘ localhost:8080/student/viewall

All Students

No. 1

NPM = 12345

Name = Gita

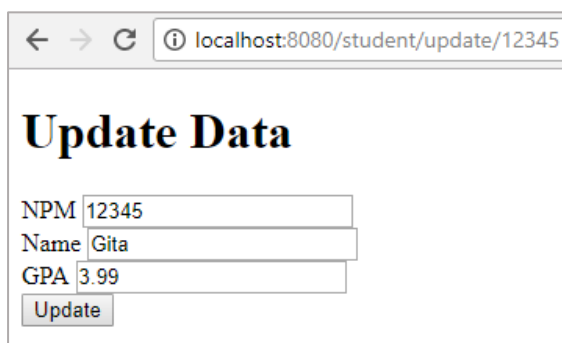
GPA = 3.99

[Delete Data](#)

[Update Data](#)

Melakukan update data:

Data awal



← → ↻ ⓘ localhost:8080/student/update/12345

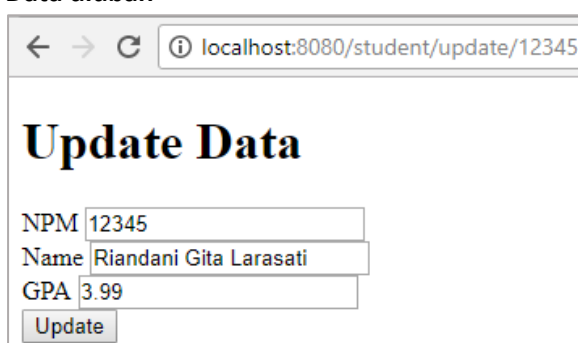
Update Data

NPM

Name

GPA

Data diubah



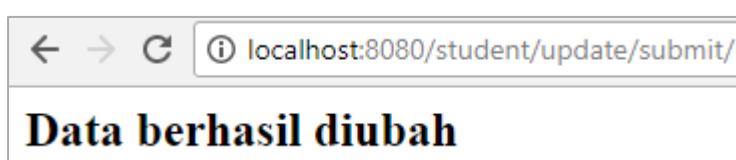
← → ↻ ⓘ localhost:8080/student/update/12345

Update Data

NPM

Name

GPA



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

Data berhasil diubah

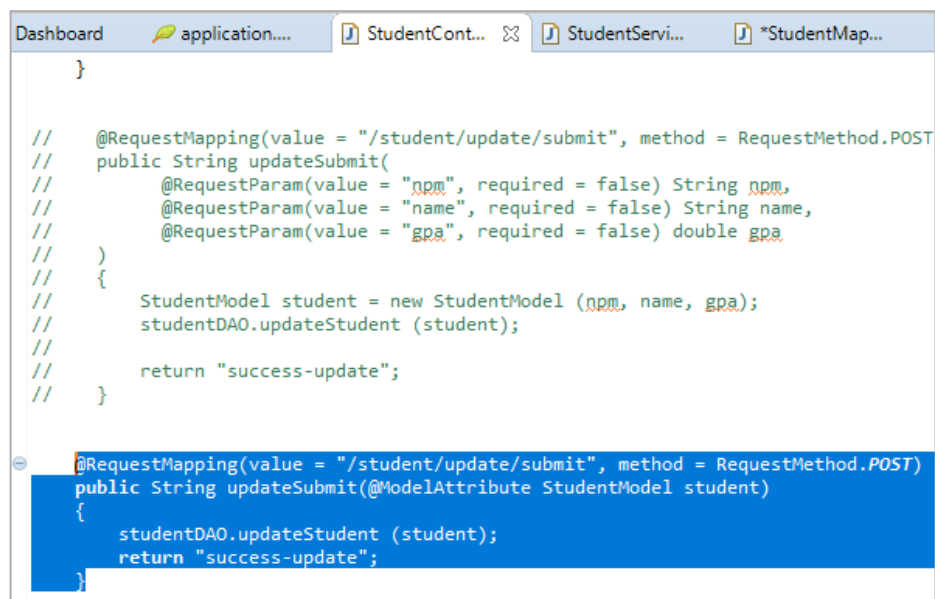
Latihan Menggunakan Object Sebagai Parameter

Lebih efisien dengan menggunakan **object** untuk parameter dibanding menggunakan **@RequestParam**. Hal tersebut karena akan jauh lebih sedikit code yang dituliskan dan dapat kita bayangkan apabila field yang digunakan lebih dari 5 atau 10 atau bahkan lebih dari itu, **tidak efisien** bila kita mendeklarasikan satu-satu menggunakan **@RequestParam**. Maka dapat diganti dengan **menuliskan sekali menggunakan object**.

Menambahkan `th:object="${student}"` pada tag di view, lalu menambahkan `th:field="*{[nama_field]}"` pada setiap input.

```
<h1 class="page-header">Edit Student</h1>
<form action="/student/update/submit/" method="post" th:object="${student}">
  <div>
    <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="*{npm}" />
  </div>
  <div>
    <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="*{name}" />
  </div>
  <div>
    <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="*{gpa}" />
  </div>
</form>
```

Mengubah method **updateSubmit** pada **StudentController** yang hanya menerima parameter berupa **StudentModel student** yang merupakan object. Object tersebut digunakan untuk menjadi masukan parameter method **updateStudent**.



```
}

// @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";
// }

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

Berikut adalah result dari tutorial project ini:

Menampilkan semua data:

← → ↻ localhost:8080/student/viewall	
All Students	
No. 1	
NPM = 12345	
Name = Riandani Gita Larasati	
GPA = 3.99	
Delete Data	Update Data
No. 2	
NPM = 12346	
Name = Bimo	
GPA = 3.8	
Delete Data	Update Data
No. 3	
NPM = 12347	
Name = Niana Guerero	
GPA = 3.6	
Delete Data	Update Data

Menghapus salah satu data dengan npm 12346:

← → ↻ localhost:8080/student/delete/12346
Data berhasil dihapus

Data terhapus:

← → ↻ localhost:8080/student/viewall	
All Students	
No. 1	
NPM = 12345	
Name = Riandani Gita Larasati	
GPA = 3.99	
Delete Data	Update Data
No. 2	
NPM = 12347	
Name = Niana Guerero	
GPA = 3.6	
Delete Data	Update Data

Mengubah salah satu data dengan npm 12345:

The image shows two browser screenshots. The top screenshot is for the URL `localhost:8080/student/update/12345` and displays a form titled "Edit Student". The form contains three input fields: "NPM" with the value "12345", "Name" with the value "Riandani Gita Larasati", and "GPA" with the value "3.99". Below these fields is an "Update" button. The bottom screenshot is for the URL `localhost:8080/student/update/submit/` and displays a message "Data berhasil diubah" (Data successfully changed).

Maka, dengan menggunakan object pun program dapat berjalan sama seperti saat menggunakan `@RequestParam` akan tetapi lebih efisien dari sisi code.

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.

Jawab:

Memanfaatkan model yang telah dibuat sebelumnya. Di dalam model tersebut dapat ditambahkan standar validation annotations seperti `@NotNull` yang tidak akan mengizinkan sebuah attribute memiliki value null. Kemudian pada controller yang menerima submit ditambahkan anotasi `@Valid` pada parameter object dan menambahkan parameter Object `bindingResult`. Kemudian pada body ditambahkan kondisi untuk melakukan validasi dengan menggunakan code `"bindingResult.hasErrors()"`. Jika hasil validasi error maka akan kembali ke form dan jika validasi berhasil maka akan melakukan aksi selanjutnya.

Ya, validasi diperlukan. 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. Selain validasi optional atau required juga penting untuk dilakukan validasi tipe data dan length yang diinputkan pada form harus sama dengan tipe data dan length pada struktur table pada database.

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:

Karena dengan menggunakan GET akan sangat tidak efisien dibanding dengan menggunakan POST melihat dari panjangnya nilai pada setiap fieldnya. Dengan menggunakan POST, maka nilai yang akan ditangkap dan ditampung tidak akan

ditampilkan pada url, dapat mengirim sebesar apapun jumlah datanya dan apapun tipe file-nya (gambar, musik, file, dll).

Ya, perlu penanganan berbeda yaitu pada header method di StudentController ditambahkan method = RequestMethod.POST pada RequestMapping.

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

Jawab:

Tidak bisa, karena GET dan POST memiliki kegunaan yang berbeda dan kegunaan tersebut tidak dapat dijalankan sekaligus. GET digunakan untuk menerima atau mengambil data sedangkan POST digunakan untuk mengirimkan data ke server.