

TUTORIAL 4 APAP

- Tutorial kali ini membahas mengenai “**Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot**”.

Pada tutorial ini, menggunakan berbagai *dependency* yaitu **Web, Thymeleaf, DevTools, Lombok, My Batis dan MySQL**. **Lombok** berfungsi sebagai *helper annotation*, **My Batis** berfungsi menghubungkan *project* dengan MySQL, dan **MyBatis** membantu untuk melakukan koneksi dan *generate query* dengan *helper annotation*.

Debugging pada SpringBoot dapat dilakukan dengan menggunakan salah satu *library Lombok* yaitu **@Slf4j** yang otomatis terdapat sebuah *variable* bernama **log** yang dapat digunakan seperti ***log.info***, ***log.debug***, dan ***log.error***, yang dapat digunakan untuk memfilter log berdasarkan jenisnya.

Pada tutorial kali ini, akan diimplementasikan *method-method* seperti *delete* dan *update* menggunakan *database* dengan menambahkan *method* yang menerima sebuah parameter dengan *annotation* beserta script SQL untuk menghapus atau meng-*update* di kelas **StudentMapper.java**. Kelas **StudentMapper.java** sebagai *datamapper*, sehingga pada tutorial kali ini jika kita ingin menampilkan student dari *database*, maka *client* akan memanggil sebuah *method* untuk mencari *student* yang ditampilkan.

Sementara jika kita ingin melakukan *update* maka *client* meminta *mapper* untuk menyimpan *domain object*, lalu *mapper* akan menarik data *domain object* yang diubah dan mengirimkannya ke *database*. Menambahkan *method* pada kelas *interface* **StudentService.java**, serta menambahkan implementasi *method* tersebut pada kelas **StudentServiceDatabase.java**, lalu, menghubungkan *method* tersebut ke halaman html.

TUTORIAL 4 APAP

1. Delete

localhost:8080/student/viewall

All Students

No. 1
NPM = 1506689435
Name = Isabella Rizka
GPA = 3.4
[Delete Data](#)
[Update Data](#)

No. 2
NPM = 1506689534
Name = Irena Gita
GPA = 3.5
[Delete Data](#)
[Update Data](#)

localhost:8080/student/delete/1506689435

Data berhasil dihapus

localhost:8080/student/delete/1506689435

Student not found

NPM = 1506689435

2. Update

localhost:8080/student/update/1506689534

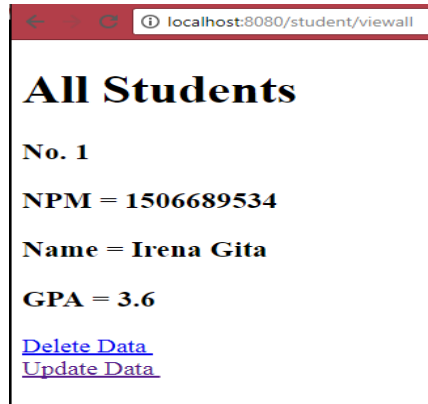
Update Student

NPM
Name
GPA

localhost:8080/student/update/submit

Data berhasil di ubah

TUTORIAL 4 APAP



localhost:8080/student/viewall

All Students

No. 1

NPM = 1506689534

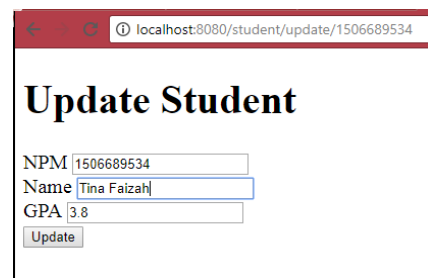
Name = Irena Gita

GPA = 3.6

[Delete Data](#)

[Update Data](#)

3. Update dengan Object Sebagai Parameter



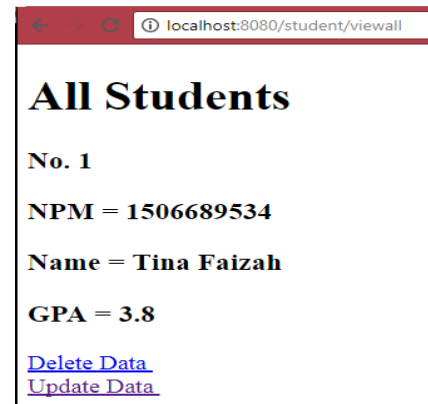
localhost:8080/student/update/1506689534

Update Student

NPM

Name

GPA



localhost:8080/student/viewall

All Students

No. 1

NPM = 1506689534

Name = Tina Faizah

GPA = 3.8

[Delete Data](#)

[Update Data](#)

A. Menambahkan Delete

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

- Untuk menghapus student dari database, maka client akan memanggil method diatas paka kelas **StudentMapper.java** yang berperan sebagai mapper. Method ini akan menghapus data student dari tabel student ketika npm student yang akan dihapus sama dengan npm student yang tersimpan di dalam database.

TUTORIAL 4 APAP

@Override

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

- Pada kelas **StudentServiceDatabase.java** menggunakan logging untuk melakukan debugging pada Spring Boot dengan menambahkan anotasi @Slf4j pada kelas tersebut sehingga otomatis terdapat variable log yang dapat digunakan. Pada method delete student terdapat log.info yang berfungsi untuk mencatat log dan memberitahukan ke sistem logging, dimana catatan log akan diformat secara independen oleh masing-masing handlers yang menanganinya. Log info diatas berisi pesan student dengan npm tertentu telah dihapus dan jika ada maka pesan tersebut akan ditafsirkan sebagai format dan nilai akan diteruskan ke bentuk pesan yang sebenarnya. Setelah itu akan dipanggil method delete student yang ada di kelas **StudentMapper.java**

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

- Pada kelas **StudentController.java** ditambahkan anotasi @RequestMapping diatas method delete yang menandakan jika ada request HTTP pada path `/student/delete/{npm}`, maka method delete pada kelas tersebut akan dipanggil dengan mempassing suatu data dari URL menggunakan path variable npm. Untuk menghapus data student yang telah tersimpan di database maka data student tersimpan akan dipilih dengan memanggil method `studentDAO.selectStudent (npm)`; yang datanya akan disimpan dalam variable student. Jika student dengan npm yang akan dihapus ada/ tidak sama dengan null maka akan dipanggil method `studentDAO.deleteStudent (npm)`; yang akan mengembalikan isi dari halaman **delete.html**, sementara jika student dengan npm yang ingin dihapus tidak ada maka digunakan `model.addAttribute ("npm", npm)`; untuk menyimpan nilai npm pada request HTTP dan digunakan untuk halaman **not-found.html** yang akan menampilkan npm tersebut di browser.

TUTORIAL 4 APAP

```
StudentController.java
1 package com.example.controller;
2
3 import java.util.List;
16
17 @Controller
18 public class StudentController
19 {
20     @Autowired
21     StudentService studentDAO;
22 }
```

- Pada kelas StudentController.java menggunakan anotasi @Autowired yang berfungsi untuk menginisiasikan sebuah instance dari StudentService pada kelas tersebut dengan nama studentDao.

B. Menambahkan Update

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

- Untuk mengupdate data student dari database, maka client akan memanggil method diatas paka kelas **StudentMapper.java** yang berperan sebagai mapper. Method ini akan mengubah data student dari tabel student dengan mengatur nama student yang disimpan ke nama yang baru, serta gpa student yang telah disimpan ke nilai gpa yang baru ketika memenuhi kondisi npm yang akan diubah sama dengan npm yang disimpan didalam database.

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

- Method diatas juga perlu ditambahkan didalam kelas interface **StudentService.java** supaya method tersebut dapat diimplementasikan oleh kelas **StudentServiceDatabase.java**

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

- Pada kelas **StudentServiceDatabase.java** juga menggunakan logging untuk melakukan debugging pada Spring Boot . Pada method update student terdapat log info yang berisi pesan student telah diupdate dan jika ada maka pesan tersebut akan ditafsirkan sebagai format dan nilai akan diteruskan ke bentuk pesan yang sebenarnya. Setelah itu akan dipanggil method update student yang ada di kelas **StudentMapper.java**

TUTORIAL 4 APAP

```
@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")
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 kelas **StudentController.java** ditambahkan anotasi `@RequestMapping("/student/update/{npm}")` diatas method update yang menandakan jika ada request HTTP pada path `/student/update/{npm}` ,maka method update pada kelas tersebut akan dipanggil dengan mempassing suatu data dari URL menggunakan path variable npm. Untuk mengupdate data student yang telah tersimpan di database maka data student tersimpan akan dipilih dengan memanggil method `studentDAO.selectStudent (npm)`; yang datanya akan disimpan dalam variable student. Jika student dengan npm yang akan diubah ada/ tidak sama dengan null maka digunakan `model.addAttribute ("student", student)`; untuk menyimpan npm pada request HTTP dan menampilkan isi dari halaman **form-update.html** di browser. Pada **form-update.html** berisi `<form action="/student/update/submit" method="post">` yang akan menampilkan sebuah form untuk mengubah nama dan gpa student dengan npm yang tadi disimpan dari requestHTTP
- Anotasi `@RequestMapping(value= "/student/update/submit")` bahwa jika ada request HTTP pada path `/student/update/submit`, maka data student yang diubah akan disimpan di dalam variable student, selanjutnya akan dipanggil method `studentDAO.updateStudent (student)`; untuk mengubah data student yang baru di database dan menampilkan isi dari halaman `success-update.html` di browser

TUTORIAL 4 APAP

- Sementara jika student dengan npm yang ingin diubah tidak ada maka digunakan `model.addAttribute("npm", npm);` untuk menyimpan nilai npm pada request HTTP dan digunakan untuk halaman **not-found.html** yang akan menampilkan npm tersebut di browser.

C. Menambahkan Object Sebagai Parameter

Kelas 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")
public String updateSubmit(@ModelAttribute StudentModel student)
{
    StudentModel students = student;

    studentDAO.updateStudent(students);

    return "success-update";
}
```

- Pada kelas **StudentController.java** ditambahkan anotasi `@RequestMapping("/student/update/{npm}")` diatas method update yang menandakan jika ada request HTTP pada path `/student/update/{npm}` ,maka method update pada kelas tersebut akan dipanggil dengan mempassing suatu data dari URL menggunakan path variable npm. Untuk mengupdate data student yang telah tersimpan di database maka data student tersimpan akan dipilih dengan memanggil method `studentDAO.selectStudent (npm);` yang datanya akan disimpan dalam variable student. Jika student dengan npm yang akan diubah ada/ tidak sama dengan null maka digunakan `model.addAttribute ("student", student);` untuk menyimpan npm pada request HTTP dan menampilkan isi dari halaman **form-update.html** di browser.

TUTORIAL 4 APAP

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

<form action="/student/update/submit" method="post" th:action="@{/student/update/submit}" 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>
  <div>
    <button type="submit" name="action" value="save">Update</button>
  </div>
</form>
```

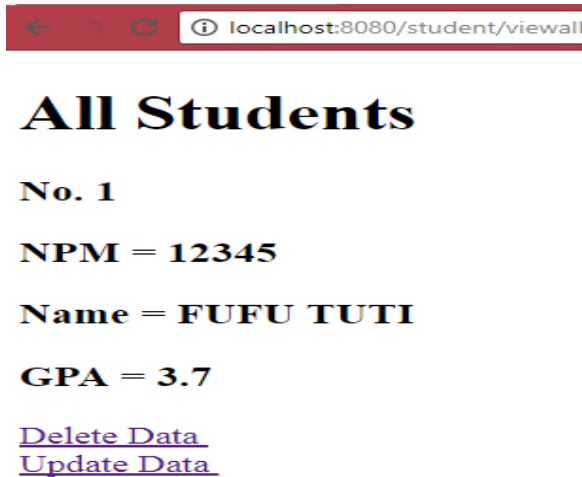
- Pada **form-update.html** diatas berisi `<form action="/student/update/submit" method="post" th:action="@{/student/update/submit}" th:object="${student}" >` , dimana **th:action** akan mengarahkan form untuk POST ke `/student/update/submit`, sementara **th:object** akan menyatakan objek yang akan digunakan untuk mengumpulkan form. Tiga form field yang dinyatakan dengan **th:field** yang sesuai dengan objek student. Objek tersebut akan mencakup controller, model dan view untuk menampilkan form.
- Anotasi `@RequestMapping(value= "/student/update/submit")` menandakan bahwa jika ada form submit dengan method POST ke request HTTP pada path `/student/update/submit`, maka akan dipanggil method `updateSubmit()` untuk menerima objek student yang ada di form. `StudentModel` merupakan sebuah `@ModelAttribute` yang terikat pada isi form yang masuk dan juga data objek student yang disubmit akan disimpan dalam variable `students` . Selanjutnya akan dipanggil method `studentDAO.updateStudent (students)`; untuk mengubah data student yang baru di database dan menampilkan isi dari halaman `success-update.html` di browser.
- Sementara jika student dengan npm yang ingin diubah tidak ada maka digunakan `model.addAttribute ("npm", npm)`; untuk menyimpan nilai npm pada request HTTP dan digunakan untuk halaman **not-found.html** yang akan menampilkan npm tersebut di browser.

Pertanyaan dan Jawaban

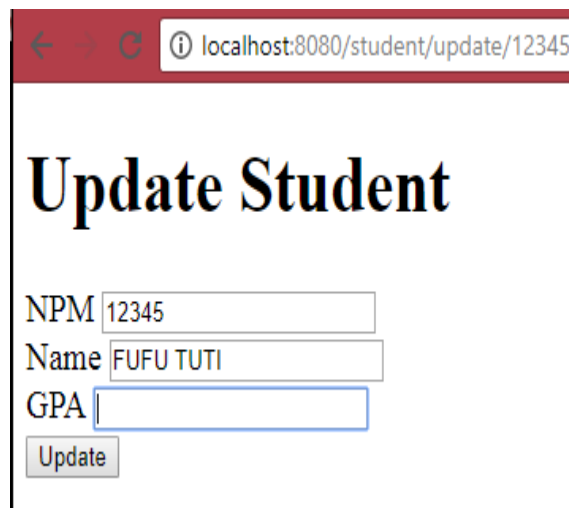
1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required seperti jika menggunakan `RequestParam`? Apakah validasi diperlukan?
Asumsikan input pada form Anda tidak menggunakan attribute `required` sehingga butuh validasi di backend.

- Jika menggunakan object sebagai parameter maka untuk input yang optional, kita dapat mendefinisikan sebuah option field dari field-field yang ada pada **form-update.html**
- Jika menggunakan object sebagai parameter pada form POST, maka untuk input yang required sudah melakukan validasi sendiri karena ketika input gpa kosong maka akan dikembalikan halaman error seperti berikut
`Resolved exception caused by Handler execution: org.springframework.validation.BindException:`

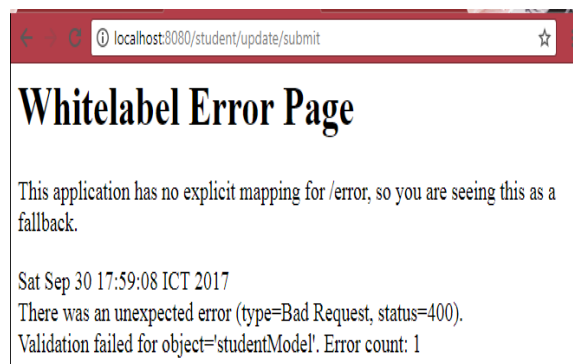
TUTORIAL 4 APAP



Gambar 1. Data Student Akan Diubah



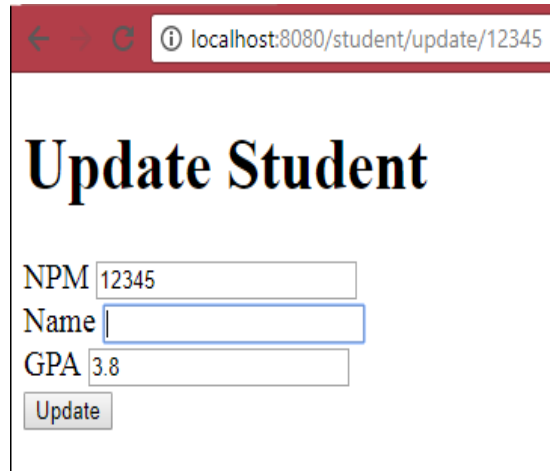
Gambar 2. GPA kosong



Gambar 3. Validation Failed

- Walaupun method POST lebih aman dibandingkan menggunakan method GET, tetapi validasi tetap diperlukan karena ada beberapa field yang tidak tervalidasi contohnya adalah field nama.

TUTORIAL 4 APAP



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

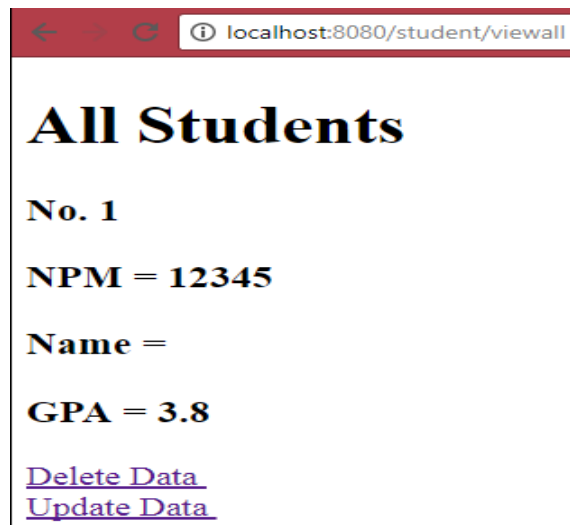
Update Student

NPM

Name

GPA

Gambar 1. Name Kosong



← → ↻ ⓘ localhost:8080/student/viewall

All Students

No. 1

NPM = 12345

Name =

GPA = 3.8

[Delete Data](#)

[Update Data](#)

Gambar 2. Name Dapat Ditampilkan Kosong

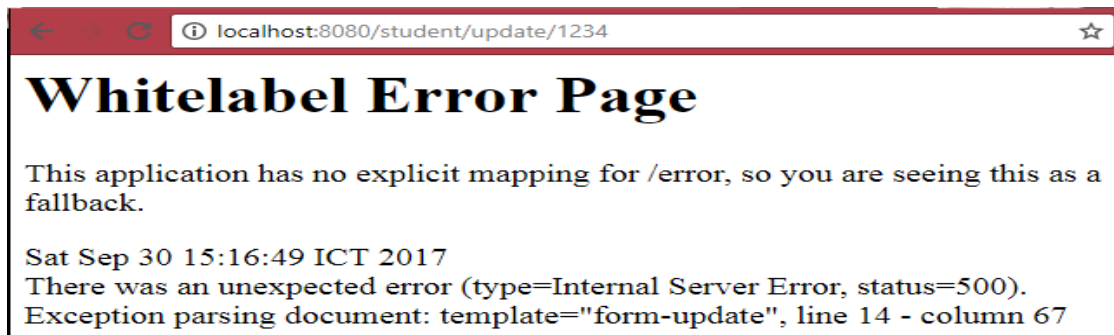
2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?
- Menurut saya karena menggunakan method POST lebih aman dibandingkan dengan GET sehingga cocok untuk digunakan pada data-data yang bersifat sensitif, karena method POST akan mengirimkan data langsung ke action untuk ditampung datanya tanpa menampilkan data tersebut pada URL, selain itu data yang dikirim dengan method POST tidak terbatas dan biasanya digunakan untuk input dari form sehingga cocok digunakan apabila kita ingin melakukan sebuah proses yang akan mengubah suatu isi database seperti add, update ataupun delete.
 - Ya perlu karena apabila menggunakan method GET maka akses data akan menggunakan link pada URL, selain itu method GET tidak memiliki body sehingga perlu dibuat annotation yaitu `@ResponseBody` pada method di kelas **StudentController.java**

TUTORIAL 4 APAP

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

```
form-update.html
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 action="/student/update/submit" method="post" method = "get" th:action="@{/student/update/submit}" th:object="${student}"
15 <div>
```

- Jika pada halaman **form-update.html** request method yaitu post dan get maka akan memunculkan pesan error seperti berikut



- Artinya suatu method tidak dapat menerima lebih dari satu request method secara bersamaan, karena attribute method telah di deklarasikan sebelumnya untuk elemen form
[org.xml.sax.SAXParseException](#): Attribute "method" was already specified for element "form".