

## Tutorial 4

### Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

#### A. Ringkasan dari materi tutorial

Hal yang dipelajari dari tutorial kali ini adalah bagaimana menggunakan database dan melakukan debugging dalam project Spring Boot. Dimana pada tutorial ini saya menggunakan database di MySQL. Terdapat pula library yang digunakan yaitu Lombok, MyBatis, dan mySQL. Salah satu fungsi Library Lombok yaitu sebagai helper annotation pada project. Library MyBatis memiliki fungsi untuk menghubungkan project dengan MySQL. MyBatis membantu Anda untuk melakukan koneksi dan generate query dengan helper annotation. Selanjutnya mempelajari application.properties dimana file ini berisi konfigurasi aplikasi. Konfigurasi yang bisa tambahkan sangat bervariasi. Mulai dari port, database config, sampai API Key dari third-party service yang gunakan. Untuk melakukan debugging pada Spring Boot dapat dilakukan dengan melakukan System.out.println dan pesan akan tercetak di console. Cara yang lebih baik dan biasa digunakan di enterprise adalah dengan menggunakan logging. Salah satu library yang biasa digunakan adalahSlf4j.

#### B. 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.

**Jawaban:**

Ya validasi diperlukan karena digunakan untuk melakukan pengecekan apakah terdapat data yang diinputkan atau tidak. Selain itu validasi juga untuk melakukan pengecekan ke database untuk memastikan bahwa format inputan pada form sesuai dengan data yang ada di database atau jika tidak maka akan terjadi error atau diberikan notice. Cara melakukan validasi adalah dengan menggunakan model. Dimana bisa ditambahkan annotations @NotNull sebagai tanda bahwa atribut tidak boleh null lalu pada controller ditambahkan @valid dan bindingResult pada parameter objectnya. Pada body tambahkan kondisi untuk validasi menggunakan bindingResult.hasErrors(). Apabila validasinya error maka akan kembali ke halaman form dan jika validasi berhasil maka akan melakukan aksi selanjutnya.

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 menggunakan POST method dibanding GET method karena jika menggunakan GET method, data yang diinput akan tampil di URL dan tentunya hal

tersebut tidaklah efektif. Bayangkan jika data yang diinputkan banyak maka semua data tersebut akan tampil di URL dan hal tersebut tidak aman. Penanganan jika menggunakan method POST dengan method lain akan berbeda pada header methodnya ditambahkan method = RequestMethod.POST pada RequestMapping, seperti berikut:

Contoh method POST

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

    return "success-update";
}
```

Contoh method GET

```
@RequestMapping("/student/add/submit")
public String addSubmit (
    @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-add";
}
```

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

**Jawaban:**

Satu method dapat terdiri lebih dari satu jenis request method seperti GET dan POST tetapi **tidak dapat dijalankan bersamaan**.

#### C. Method pada Latihan Menambahkan Delete

Pada viewall.html ditambahkan seperti kotak merah di bawah dimana delete merefer ke npm student yang dipilih.

**viewall.html**

```

<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>
    <a th:href="/student/update/' + ${student.npm}">Update Data</a><br/>
    <a th:href="/student/delete/' + ${student.npm}">Delete Data</a><br/>
</div>

```

Pada class StudentMapper tambahkan method deleteStudent dimana terdapat perintah yang menuju ke database yaitu delete dari table student dimana sesuai dengan npm yang dipilih.

### StudentMapper

```

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

```

Pada method deleteStudent di class StudentServiceDatabase tambahkan log dan memanggil method deleteStudent pada class StudentMapper.

### StudentServiceDatabase

```

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

```

Pada method delete di class StudentController lakukan select student terlebih dahulu dengan NPM kemudian tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found dan jika berhasil delete student dan tampilkan view delete.

### StudentController

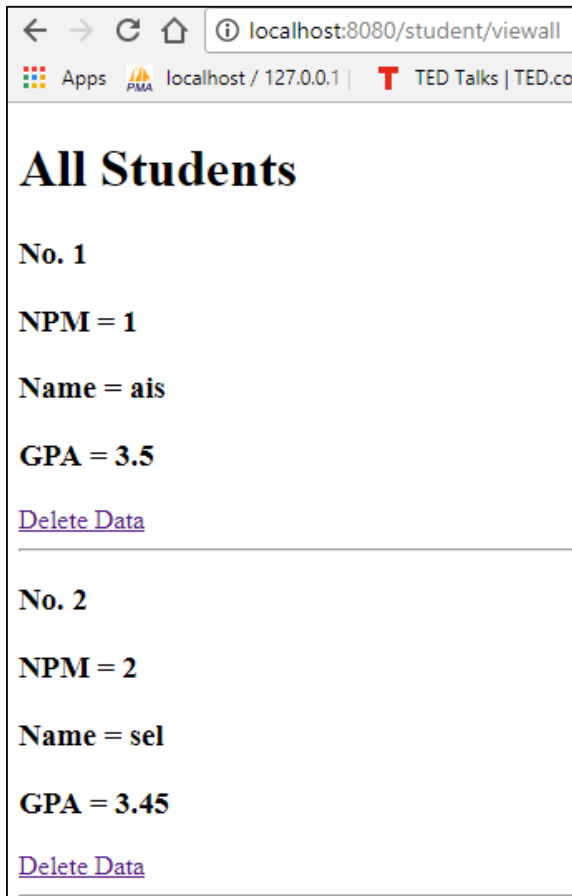
```

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

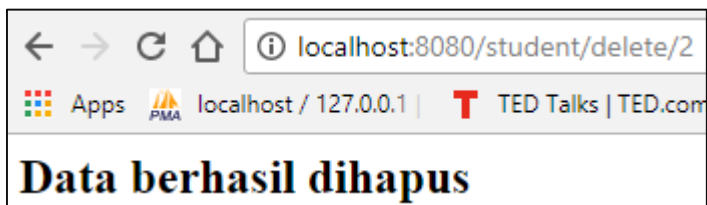
    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}

```

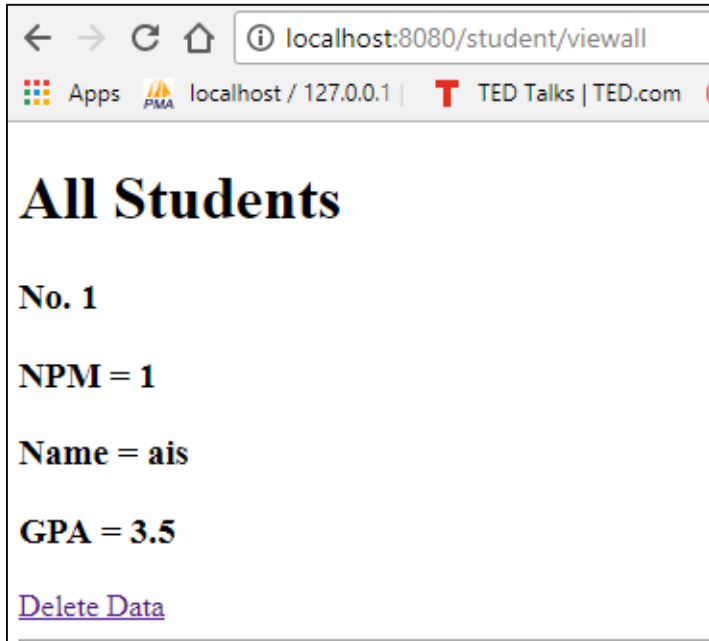
Selanjutnya jalankan Spring Boot app dan lakukan beberapa insert di database kemudian tampilkan viewall seperti di bawah ini (sesuai dengan isi database).



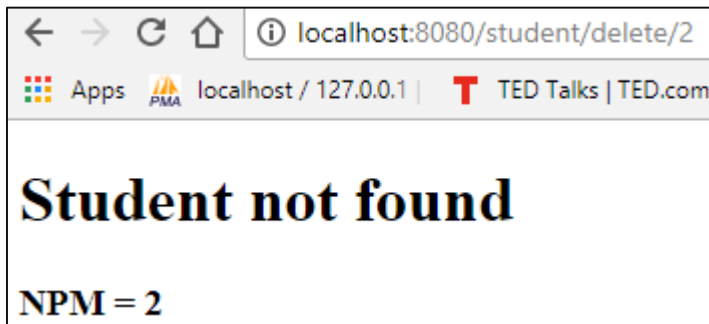
Klik data dengan NPM 2 maka akan muncul tampilan bahwa data berhasil dihapus yang mengacu pada method delete di controller.



Selanjutnya buka kembali viewall maka data yang tersisa hanya satu data student dengan NPM 1.



Kita dapat melakukan pengecekan dengan memanggil kembali NPM student yang telah dihapus sebelumnya di URL, maka akan keluar tampilan view not found.



#### D. Method pada Latihan Menambahkan Update

Pada fitur update menggunakan method POST. Pertama Tambahkan method updateStudent pada class StudentMapper dengan parameter StudentModel student, Annotationnya adalah @Update, lalu tambahkan query untuk update yaitu kita update pada tabel student lalu kita set akan merubah isi name dan gpa sesuai dengan npm yang dipilih.

##### StudentMapper

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

Method updateStudent pada interface StudentService untuk menghubungkan dengan class StudentServiceDatabase.

##### StudentService

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

Selanjutnya tambahkan method updateStudent pada class StudentServiceDatabase dengan menambahkan argumen log untuk melakukan update ke student yang diambil npm dan namanya dan memanggil method updateStudent pada studentMapper.

##### StudentServiceDatabase

```
@Override
public void updateStudent(StudentModel student)
{
    log.info("Student {} name update to", student.getNpm(), student.getName());
    studentMapper.updateStudent(student);
}
```

Lalu pada viewall.html tambahkan link update data yang berfungsi menerima request.

##### viewall.html

```
<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>
    <a th:href="'/student/update/' + ${student.npm}" >Update Data</a><br/>
    <a th:href="'/student/delete/' + ${student.npm}" >Delete Data</a><br/>
    <hr/>
</div>
```

Selanjutnya buat form-update.html dan sesuaikan tittle, page-header, tombol menjadi update. Lalu action form menjadi /student/update/submit dan method menjadi POST dan input npm menjadi seperti di bawah ini dimana readonly agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada. Begitu juga dengan name dan gpa.

#### form-update.html

```
<title>Update 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" th:object="${student}">
    <div>
      <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="*{npm}" />
    </div>
    <div>
      <input type="text" name="name" th:value="${student.name}" th:field="*{name}" />
    </div>
    <div>
      <input type="text" name="gpa" th:value="${student.gpa}" th:field="*{gpa}" />
    </div>

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

Lalu buat success-update.html untuk tampilan apabila data berhasil diupdate.

#### success-update.html

```
<html>
  <head>
    <title>Update</title>
  </head>
  <body>
    <h2>Data berhasil di-update</h2>
  </body>
</html>
```

Selanjutnya tambahkan method update pada class StudentController. Dengan request mapping ke /student/update/{npm} lalu buat validasi jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update.

**StudentController**

```

@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";
    }
}

```

Tambahkan method updateSubmit pada class StudentController karena menggunakan post method maka request mappingnya seperti di kotak merah bawah ini dengan header method seperti di bawah ini. Lalu memanggil method update Student dan kembalikan view success-update

**StudentController**

```

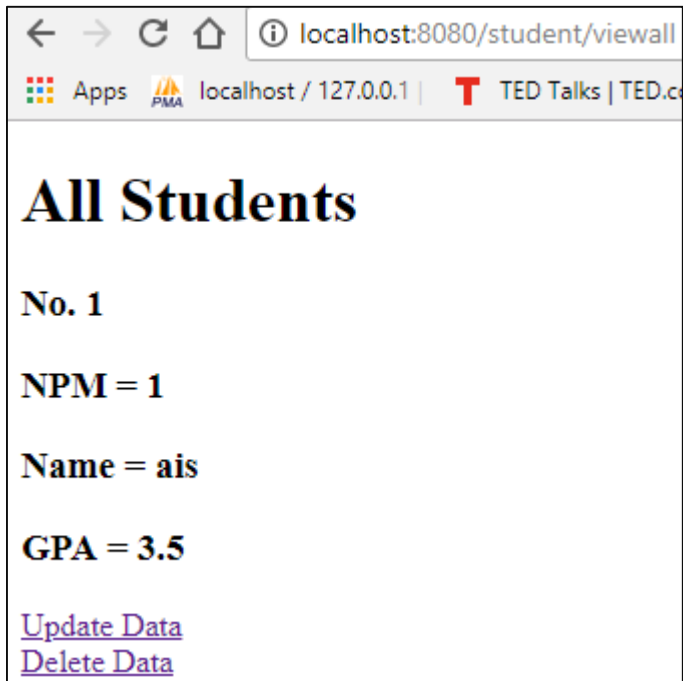
@RequestMapping(value="/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
    @RequestParam(value = "npm", required = false) String npm,
    @RequestParam(value = "name", required = false) String name,
    @RequestParam(value = "gpa", required = false) double gpa)
{
    StudentModel student = new StudentModel (npm, name, gpa);
    studentDAO.updateStudent (student);

    return "success-update";
}

```



Jalankan spring boot dan panggil viewall kembali maka akan tampil seperti di bawah ini. Lalu klik Update Data.



localhost:8080/student/viewall

Apps PMA localhost / 127.0.0.1 | TED Talks | TED.com

## All Students

No. 1

NPM = 1

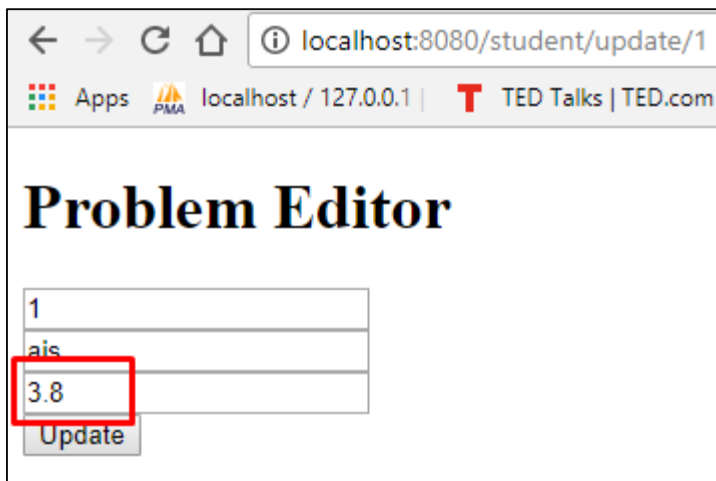
Name = ais

GPA = 3.5

[Update Data](#)

[Delete Data](#)

Selanjutnya ubah GPA dari 3.5 menjadi 3.8 lalu klik button update.



localhost:8080/student/update/1

Apps PMA localhost / 127.0.0.1 | TED Talks | TED.com

## Problem Editor

1
ais
3.8

Update

Maka akan tampil view update berhasil dilakukan.

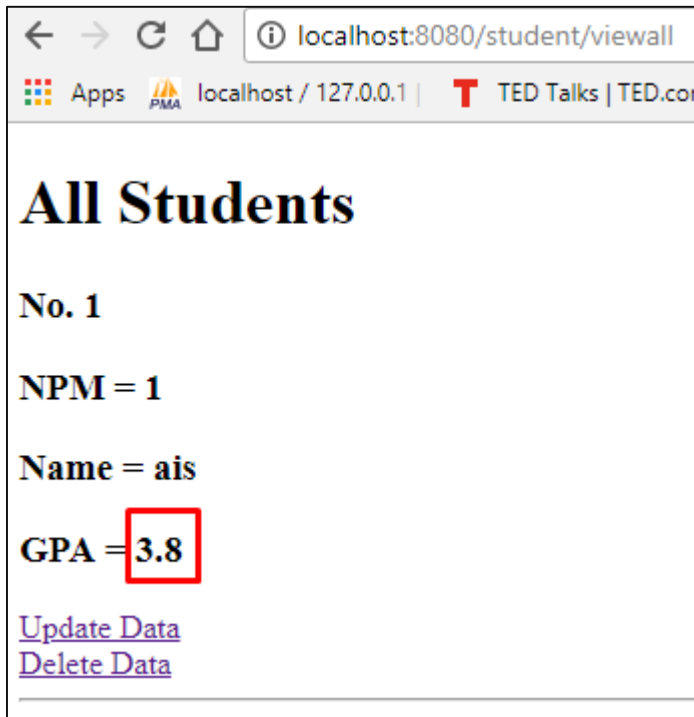


localhost:8080/student/update/submit

Apps PMA localhost / 127.0.0.1 | TED Talks | TED.com

## Data berhasil di-update

Lalu kita panggil kembali method viewall pada URL maka data student tersebut sudah ter-update.



#### E. Method pada Latihan Menggunakan Object Sebagai Parameter

Pada tutorial sebelumnya masih menggunakan RequestParam untuk handle form submit. Sehingga ada banyak parameter pada method. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakan RequestParam.

#### StudentController

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

    return "success-update";
}
```

Menambahkan `th:object="${student}"` pada tag `<form>` di view form-update digunakan untuk mengisi input object dengan student, menambahkan `th:field="*{[nama_field]}"` pada setiap input digunakan untuk mengisi input field dengan npm, name, dan gpa.

**form-update**

```

<form action="/student/update/submit" method="post" th:object="${student}">
  <div>
    <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="**{npm}" />
  </div>
  <div>
    <input type="text" name="name" th:value="${student.name}" th:field="**{name}" />
  </div>
  <div>
    <input type="text" name="gpa" th:value="${student.gpa}" th:field="**{gpa}" />
  </div>

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

```

Selanjutnya tes kembali aplikasi dengan klik Update data.

← → ↻ 🏠 localhost:8080/student/viewall

Apps PMA localhost / 127.0.0.1 | TED Talks | TED.com

## All Students

No. 1

NPM = 1

Name = ais

GPA = 3.8

[Update Data](#)

[Delete Data](#)

Kemudian ganti GPA lalu klik button update.

← → ↻ 🏠 localhost:8080/student/update/1

Apps PMA localhost / 127.0.0.1 | TED Talks | TED.com

## Edit Student

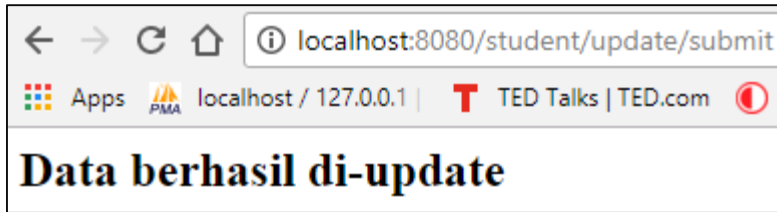
1

ais

3.95

Update

Akan tampil view seperti di bawah ini jika update berhasil.



Dan kita panggil kembali viewall dan data gpa sudah ter-update.

