Fajar Bardan Salam / 1506689401
APAP-A

Tutorial 4 : Menggunakan Database dan Melakukan Debuggin dalam Project Spring Boot

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/>
```

application.properties | 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}">
            <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/delete/' + ${student.npm}"> Delete Data</a><br/>
            <hr/>
        </div>
    </body>
</html>
```

3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**

viewall.html | StudentMapper.java

```
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(@Param("npm") String npm);
}
```

4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**

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

5. Lengkapi method **delete** pada class **StudentController**

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

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

6. Jalankan Spring Boot app dan lakukan beberapa insert
7. Contoh tampilan View All
   (Note : tidak dapat ditampilkan karena Lombok yang terinstall pada laptop saya error dan tetap tidak bisa dijalankan)

8. Contoh tampilan delete NPM 123
   (Note : tidak dapat ditampilkan karena Lombok yang terinstall pada laptop saya error dan tetap tidak bisa dijalankan)

9. Contoh tampilan jika dilakukan delete NPM 123 yang kedua kalinya
   (Note : tidak dapat ditampilkan karena Lombok yang terinstall pada laptop saya error dan tetap tidak bisa dijalankan)

Fajar Bardan Salam / 1506689401
APAP-A

Latihan Menambahkan Update

1. Tambahkan method **updateStudent** pada class **StudentMapper**

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

2. Tambahkan 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);
}
```

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

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

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

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

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

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

<body>

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

</html>
```

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

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diupdate!</h2>
    </body>
</html>
```

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

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

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

8. Tambahkan method **updateSubmit** pada class **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)
    {
        studentDAO.updateStudent(new StudentModel(npm, name, gpa));
        return "success-update";
    }
}
```

9. Jalankan Spring Boot dan coba test program Anda.
   (Note : tidak dapat ditampilkan karena Lombok yang terinstall pada laptop saya error dan tetap tidak bisa dijalankan)

Latihan Menggunakan Object Sebagai Parameter

1. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.

2. SpringBoot dan Thymeleaf memungkinkan agar method **updateSubmit** menerima parameter berupa model **StudentModel** . **Metode ini lebih disarankan dibandingkan menggunakan RequestParam.**

3. Cara lengkapnya silakan ikuti pada link Handling Form berikut:
   (*https://spring.io/guides/gs/handling-form-submission/*)

4. Tahapannya kurang lebih sebagai berikut:

- Menambahkan th:object=”${student}” pada tag **<form>** di view

```
<form th:object="${student}" action="/student/update/submit" method="post">
    <div>
```

- Menambahkan th:field=”*{[nama_field]}” pada setiap input

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

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

- Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel

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

    return "success-update";
}
```

- Tes lagi aplikasi Anda.
  (Note : tidak dapat ditampilkan karena Lombok yang terinstall pada laptop saya error dan tetap tidak bisa dijalankan)

Fajar Bardan Salam / 1506689401
APAP-A

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.
   Jawab : Untuk melakukan validasi tersebut, kita harus mengecek property dari objek null atau tidak, dapat dihandle pada controllernya, gpa tidak boleh kosong. Apabila gpa kosong maka textbox akan mengembalikan String, sedangkan gpa yang diset bertipe double sehingga akan terjadi error. Validasi yang diperlukan ialah mengecek apakah student.npm tidak ada isinya sehingga input tidak bisa ditinggalkan jika 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?
   Jawab : GET diakses melalui URL sedangkan POST dapat dikatakan lebih aman dan dapat merubah database. GET method biasanya digunakan untuk mengambil data pada database, sehingga biasanya form submit menggunakan POST dibandingkan method GET. Perlu adanya penangan apabila menggunakan method yang berbeda, karena untuk setiap request method memiliki parameter yang berbeda (*RequestParameter.GET* dan *RequestParameter.POST*)

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Jawab : Mungkin saja terjadi

- Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan
→

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

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

Method ini mempunyai parameter npm, kemudian npm yang telah diinput dicek terlebih dahulu melalui method selectStudent pada object studentDAO. Jika object students tidak null (berarti terisi atau tersedia pada database), maka method akan menjalankan method deleteStudent dan menampilkan halaman *delete.* Sedangkan apabila object student null (tidak ada isinya) maka akan menampilkan halaman *not-found.*

viewall.html StudentMapper.java

```java
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(@Param("npm") String npm);
}
```

Method ini akan menjalakan SQL DELETE setelah method delete dari Controller dijalankan.

- Method yang Anda buat pada Latihan Menambahkan Update, jelaskan
→

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

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

Sama halnya seperti method delete diatas, method ini memiliki parameter npm, kemudian npm yang telah diinput dicek terlebih dahulu melalui method selectStudent pada object studentDAO. Jika object students tidak null (berarti terisi atai tersedia pada database), maka method akan menambahkan object student sebagai atribut pada model, dan kemudian akan menampilkan halaman *form-update*. Sedangkan apabila object student null (tidak ada isinya) maka akan menampilkan halaman *not-found*.

```java
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);
}
```

Method ini akan menjalakan SQL UPDATE setelah method update dari Controller dijalankan.

- Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan
→

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

    return "success-update";
}
```

Method ini memanggil object student dari class StudentModel sebagai parameternya. Kemudian akan memanggil method updateStudent pada class studentDAO dan menampilkan halaman *success-update*

Write Up Tutorial 4 :

Pada tutorial 4 ini, saya mempelajari bagaimana Spring Boot dapat melakukan koneksi dengan database. Saya juga mengetahui beberapa hal penting pada tutorial ini, yaitu Menambahkan Delete, Menambahkan Update, serta Menggunakan Object sebagai Parameter. Pada bagian ketiga atau menggunakan object sebagai parameter, saya melihat bahwa ada cara untuk men-submit form melalui method POST tanpa harus mendefinisikan tiap atribut dari form tersebut.

Note : pada tutorial ini saya mengalami kendala pada instalasi Lombok, dimana berkali-kali Lombok.jar tidak terinstall dengan baik. Pada saat instalasi terlihat biasa saja dan berjalan normal sebagaimana teman-teman yang lain melakukan instalasi, namun Lombok yang terinstall di laptop saya tidak dapat berfungsi ketika program akan di running. Mohon maaf apabila writeup ini tidak didukung dengan screenshot yang memadai.