**Julio Jaya Handoyo**
**1506757440**
**APAP – A**

Pada tutorial ini, saya mempelajari cara berinteraksi dengan *database* menggunakan program yang pada tutorial sebelumnya belum digunakan serta melakukan debugging dalam project spring boot, dan menggunakan objek ketimbang atribut sebagai parameter. Melalui beberapa library yang dipakai pada tutorial ini, saya dapat menambah query disertai dengan anotasi perintah query tersebut langsung diatas method. Selanjutnya, saya juga mempelajari cara debugging yang biasa digunakan untuk skala enterprise, yaitu menggunakan logging ketimbang system.out.println(). Selanjutnya, penggunaan object sebagai parameter tentu sangat menghemat waktu ketimbang memakai atribut object tersebut yang mungkin jumlahnya akan sangat banyak.

**Pertanyaan**

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

Sumber: https://docs.oracle.com/javaee/7/api/javax/validation/constraints/package-summary.html
https://spring.io/blog/2009/11/17/spring-3-type-conversion-and-validation/

Terdapat dua cara melakukan validasi, yaitu menggunakan javax.validation.constraints dan @valid. @valid dapat dilakukan untuk melakukan validasi objek terhadap atribut-atribut yang dimasukkan dan mengecek apakah sesuai dengan atribut objek tersebut. Selanjutnya, menggunakan javax.validation.constraints. Untuk mengatasi masalah required tinggal menambahkan anotasi @NotNull diatas field yang diinginkan.

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?

Form submit biasanya menggunakan POST method karena untuk menghindari adanya data-data (yang seharusnya bersifat rahasia dan milik pengisi) yang muncul pada url dan pengisi form tidak bisa mengisi melalui url saja. Ya, perlu penanganan berbeda pada @RequestMapping kalau menggunakan POST method hal yang harus ditambah method = RequestMethod.POST, sedangkan jika menggunakan GET method hal yang harus ditambah method = RequestMethod.GET.

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

Ketika menggunakan spring mvc, satu method mungkin dapat menerima lebih dari satu jenis request method.
https://www.intertech.com/Blog/spring-mvc-handler-methods/

**Latihan Menambahkan Delete**
Pada latihan ini, saya mengikuti langkah-langkah yang sudah diberikan pada file tutorial untuk mengimplementasikan method delete menggunakan database.
Pertama, saya menambahkan link untuk menghapus data(di-highlight) pada view 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/delete/' + ${student.npm}">Delete Data</a><br/>
    <a th:href="'/student/update/' + ${student.npm}">Update Data</a><br/>
    <hr/>
</div>
```

Kedua, saya menambahkan method deleteStudent yang bertugas untuk menghapus data dari databse pada class StudentMapper

```
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(StudentModel student);

}
```

Ketiga, saya melengkapi method deleteStudent pada StudentServiceDatabase

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

Terakhir, saya melengkapi method delete pada controller StudentController

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

**Latihan Menambahkan Update**

Pada latihan ini, saya mengikuti langkah-langkah yang sudah diberikan pada file tutorial. Pertama, saya menambahkan method updateStudent pada class StudentMapper.

```java
@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(StudentModel student);

}
```

Kedua, saya juga menambahkan method updateStudent pada interface StudentService.

```java
public interface StudentService
{
    StudentModel selectStudent (String npm);


    List<StudentModel> selectAllStudents ();


    void addStudent (StudentModel student);


    void deleteStudent (String npm);

    void updateStudent(StudentModel student);
}
```

Ketiga, saya mengimplementasikan method updateStudent pada StudentServiceDatabase, serta menambahkan logging pada method ini.

```java
    @Override
    public void updateStudent(StudentModel student) {
        // TODO Auto-generated method stub
        log.info("student " + student.getNpm() + " updated");
        studentMapper.updateStudent(student);
    }
```

Keempat, saya menambahkan link update data pada viewall.html agar bisa diakses ketika melihat semua data student

```
<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>
```

Kelima, saya membuat view form-update.html

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

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

Keenam, saya membuat view success-update.html untuk memastikan jika data berhasil di-update

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Memperbarui data berhasil</h2>
    </body>
</html>
```

Ketujuh, saya membuat method update pada controller StudentController.

```
@RequestMapping("/student/update/{npm}")
public String updatePath(Model model, @PathVariable(value = "npm") String npm) {
    StudentModel student = studentDAO.selectStudent(npm);
    if(student != null) {
        studentDAO.updateStudent(student);
        model.addAttribute("student", student);
        return "form-update";
    }
    model.addAttribute("npm", npm);
    return "not-found";
}
```

Terakhirm saya menambahkan method updateSubmit pada class StudentController

```
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";
}
```

**Latihan Menggunakan Object sebagai Parameter**

Pada latihan ini, saya mengikuti langkah-langkah yang sudah diberikan pada file tutorial. Pertama, saya menambahkan th:object=”${student}” pada tag <form> di view form-update.html. Kedua, menambahkan th:field=”*{[nama_field]}” pada setiap input.

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

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

Selanjutnya, saya mengubah method updateSubmit pada StudentController sehingga method tersebut hanya menerima parameter berupa StudentModel. Bisa dilihat perbedaannya, jika method sebelumnya harus membuat objek baru untuk menerima parameter npm, nama, dan gpa kemudian meng-updatenya, sedangkan setelah parameternya diubah menjadi student, method updateStudent dapat langsung meng-update student yang menjadi parameter.

```
//    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(StudentModel student) {
        studentDAO.updateStudent(student);
        return "success-update";
    }
```