

## **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? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

**Jawab:** Model attribute is already optional. An @ModelAttribute on a method argument indicates the argument will be retrieved from the model. If not present in the model, the argument will be instantiated first and then added to the model.

In fact, we can add a boolean in StudentModel

Boolean objExists = false;

Then we can do something like this:

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

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 retrieve a response from an **url** and POST send also some content data to that url. When submitting form, data is collected in a standard and sent, also this time, to the url. GET is for retrieving / viewing information and POST for creating / editing information. The POST is most likely used in submitting form because GET method appends name/value pairs to the URL. Unfortunately, the length of a URL is limited, so this method only works if there are only a few parameters. The URL could be truncated if the form uses a large number of parameters, or if the parameters contain large amounts of data. Also, parameters passed on the URL are visible in the address field of the browser so it is not the best place for a password to be displayed.
- Yes, we have to change several things. If we don't, the parameters won't be passed in the url. The url read the word "submit" as the parameter instead. Therefore, the form won't be sent and the data won't be saved.

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

**Jawab:** Yes, but it is not the ideal way. We can create one method accepting both types of request, then check what type of request you received, is it of type "GET" or "POST", once we come to know that, do respective actions and then call one method which does common task for both request Methods ie GET and POST. In spring developer we can use both RequestMethod.POST and RequestMethod.GET at same controller just making an array of method like this:

```
@RequestMapping(value = "/student/update/{npm}", method = {RequestMethod.POST, RequestMethod.GET})
public String update(Model model, @PathVariable(value = "npm") String npm
,HttpServletRequest request
){
```

Implement the code here.

```
}
```

### **Lesson Learned :**

- Menggunakan logging
- Cara mengakses database dengan Spring
- Menambah, menghapus, dan memanipulasi data di database dengan Mapper dan Controller
- Parse object dari view untuk digunakan di controller

### **Penjelasan method yang dibuat :**

- Method menambahkan delete
- StudentMapper:

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

Query sql untuk menghapus data student di database dengan npm yang didapat dari parameter npm dari controller

- StudentController:

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

```
    }  
    studentDAO.deleteStudent (npm);  
  
    return "delete";  
}
```

Student dengan npm tersebut akan dimasukkan ke variable student.

Pertama akan dicek jika npm nya null berarti tidak ada mahasiswa dan jika student bernilai null berarti tidak ada mahasiswa dengan npm yang diberikan. Lalu akan masuk if pertama yang akan redirect page not-found.html. Jika ada, maka akan di delete dari database dan redirect ke page delete.html

- Method menambahkan Update
- StudentMapper

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

Query update nama dengan gpa student dengan npm yang didapat dari object student

- StudentController

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

Sama seperti delete jika npm null dan nilai student null berarti tidak ditemukan mahasiswa di database dan di return ke page not-found.html

Jika ada student, maka akan di direct ke form-update dimana kita akan diberikan form untuk mengubah nama dan gpa.

```
@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";  
  
}
```

Dari parameter yang diberikan akan dibuat StudentModel student dan akan menjalankan query update di StudentMapper

- Method menggunakan Object sebagai parameter

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

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

Untuk menggunakan object sebagai parameter harus digunakan ModelAttribute. ModelAttribute digunakan ketika ingin binding data dari request dan ditambahkan ke model secara implicit.

## Latihan Menambahkan Delete

1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.

2. Pada viewall.html tambahkan Delete Data

Di dalam div dengan th:each

3. Tambahkan method deleteStudent yang ada di class StudentMapper

a. Tambahkan method delete student yang menerima parameter NPM.

Tambahkan annotation delete di atas dan SQL untuk menghapus @Delete("[LENGKAPI]")  
Lengkapi script SQL untuk menghapus mahasiswa dengan NPM tertentu.

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

4. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

a. Tambahkan log untuk method tersebut dengan cara menambahkan log.info ("student " + npm + " deleted");

b. Panggil method delete student yang ada di Student Mapper

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

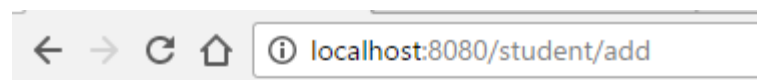
5. Lengkapi method delete pada class StudentController

- a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found
- b. Jika berhasil delete student dan tampilkan view delete • Hint: lakukan select student terlebih dahulu dengan NPM, kurang lebih mirip dengan method view

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

    return "delete";
}
```

6. Jalankan Spring Boot app dan lakukan beberapa insert

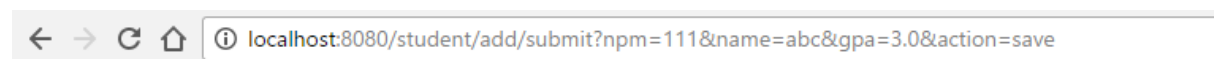


## Add Student

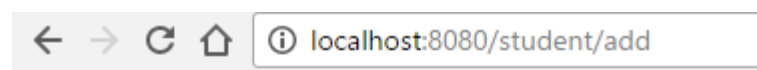
NPM

Name

GPA



**Data berhasil ditambahkan**



## Add Student

NPM

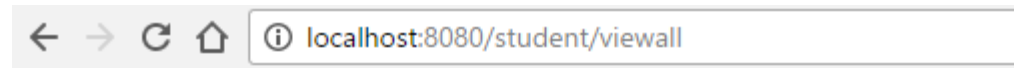
Name

GPA



**Data berhasil ditambahkan**

7. Contoh tampilan View All



## All Students

**No. 1**

**NPM = 000**

**Name = zzz**

**GPA = 1.1**

[Delete Data](#) [Update Data](#)

**No. 2**

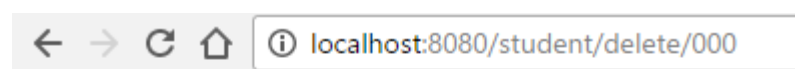
**NPM = 111**

**Name = abc**

**GPA = 3.0**

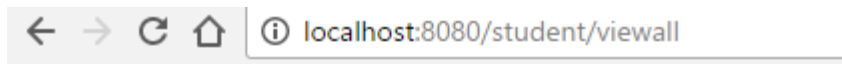
[Delete Data](#) [Update Data](#)

Delete No.1 npm 000 Name zzz



**Data berhasil dihapus**

Data di hapus. View All:



## All Students

**No. 1**

**NPM = 111**

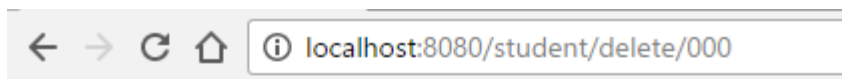
**Name = abc**

**GPA = 3.0**

[Delete Data](#) [Update Data](#)

---

9. Contoh tampilan jika dilakukan delete NPM 000 yang kedua kalinya



## Student not found

**NPM = 000**

## Latihan Menambahkan Update

1. Tambahkan method updateStudent pada class StudentMapper

a. Parameternya adalah StudentModel student

b. Annotationnya adalah @Update

c. Lengkapi SQL update-nya

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

2. Tambahkan method updateStudent pada interface StudentService

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

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

```
@Override
public void updateStudent (StudentModel student){

    Log.info("student " + student.getNpm() + " updated");
    studentMapper.updateStudent(student);
}
```

4. Tambahkan link Update Data pada viewall.html

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

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

a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.

b. Ubah action form menjadi /student/update/submit

c. Ubah method menjadi post



```
<!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">Edit 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>
```

Active

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

a. Ubah keterangan seperlunya

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

7. Tambahkan method update pada class StudentController

a. Request mapping ke /student/update/{npm}

b. Sama halnya seperti delete, lakukan validasi.

c. Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update

```
@RequestMapping("/student/update/{npm}")
public String update(Model model, @PathVariable(value = "npm") String npm){

    StudentModel student = studentDAO.selectStudent(npm);
    model.addAttribute("student", student);
    if(npm.equals(null) || student==null){

        return "not-found";
    }
    return "form-update";

}
```

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){
    StudentModel student = new StudentModel(npm, name, gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Hasil view:

View All

[←](#) [→](#) [↻](#) [🏠](#) [📄](#) localhost:8080/student/viewall

## All Students

**No. 1**

**NPM = 111**

**Name = abc**

**GPA = 3.0**

[Delete Data](#) [Update Data](#)

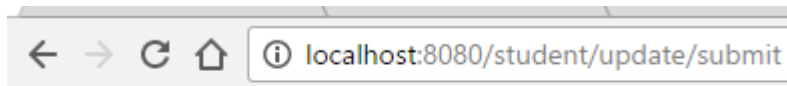
---

Update data No.1

## Edit Student

NPM	<input type="text" value="111"/>
Name	<input type="text" value="zzz"/>
GPA	<input type="text" value="3.0"/>
<input type="button" value="Update"/>	

Klik update



## Data berhasil diubah

Nama berubah

## All Students

No. 1

**NPM = 111**

**Name = zzz**

**GPA = 3.0**

[Delete Data](#) [Update Data](#)

---

## Latihan Menggunakan Object Sebagai Parameter

kurang lebih sebagai berikut:

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

```
<!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">Edit Student</h1>

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

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

</body>
</html>
```

Activate Win

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

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

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

- Tes lagi aplikasi Anda.

View all

# All Students

No. 1

NPM = 111

Name = zzz

GPA = 3.9

[Delete Data](#) [Update Data](#)

---

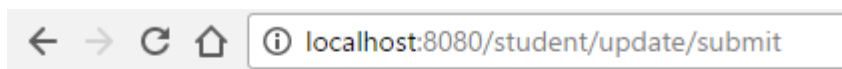
Ubah data No.1 gpa menjadi 2.7



## Edit Student



NPM	<input type="text" value="111"/>
Name	<input type="text" value="zzz"/>
GPA	<input type="text" value="2.7"/>
<input type="button" value="Update"/>	

Klik update



## Data berhasil diubah

Hasil view all

  localhost:8080/student/viewall

# All Students

**No. 1**

**NPM = 111**

**Name = zzz**

**GPA = 2.7**

[Delete Data](#) [Update Data](#)

---