Pada tutorial ini kita mempelajari bagaimana menghubungan ke database mysql, kemudian menggunakan library MyBatis dan Lombok. Kegunaan dari MyBatis yaitu melakukan koneksi dan generate query dengan annotation helper. Kemudian kegunaan dari Lombok yaitu helper annotation pada project. Selanjutnya kita belajar log, log ini berguna untuk melakukan debugging.

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?

   Dengan cara mengambil data dari object student , seperti student.getName(), student.getGpa(), student.getNpm(). Ini dilakukan pada controller, saat implementasi method updateSubmit.

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

   Diperlukan ketika diperlukan, contohnya ketika column dari tablenya tidak bisa kosong.

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method?
   Jawab:
   Menurut saya dengan menggunakan method POST, data yang dikirimkan lebih aman dibanding menggunakan GET. Dikarenakan dengan method GET data yang di*parsing* tertera pada url nya, sehingga datanya bisa dilihat (tidak aman), sedangkan POST tidak. Contohnya *username* dan *password*.

   Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?
   Tidak ada penanganan berbeda pada *header* atau *body* method di controllernya.

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Bisa, dikarenakan para @RequestMapping bisa memiliki 2 jenis method GET dan POST.

4. Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan.

   Membuat method delete pada StudentController dengan parameter npm, kemudian implementasi method tersebut, melakukan checking terlebih dahulu, apakah student dengan npm yang akan didelete ada atau tidak, jika ada melakukan pemanggilan method deleteStudent return “delete”, jika tidak return “not-found”.

```
@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 ( s: "npm", npm);
        return "not-found";
    }

}
```

Mempuat method delete student pada StudentMapper dengan parameter npm, kemudian, membuat queray untuk delete student berdasarkan npm yang di*parsing*.

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

Membuat method deleteStudent dengan parameter npm pada StudentService.

```
void deleteStudent (String npm);
```

Membuat method deleteStudent dengan parameter npm pada StudentServiceDatabase, dimana, dalam implementasinya menampilkan log, dan melakukan pemanggilan method deleteStudent pada studentMapper.

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

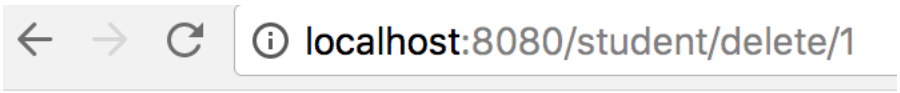
Membuat view delete.html.

```
<html>
    <head>
        <title>Delete</title>
    </head>
    <body>
        <h2>Data berhasil dihapus</h2>
    </body>
</html>
```

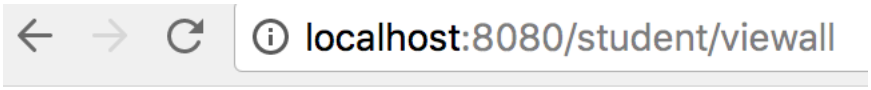
Membuat link delete pada viewall

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

Demo:



Data berhasil dihapus



All Students

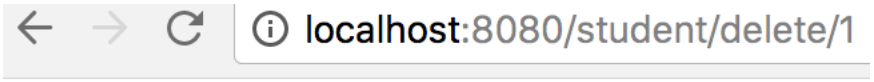
No. 1

NPM = 1234

Name = Novi

GPA = 3.0

Delete data Update data



Student not found

NPM = 1

5. Method yang Anda buat pada Latihan Menambahkan Update, jelaskan

Tambahkan method update pada class StudentController. Dalam method tersebut terdapat kode untuk memvalidasi data student dengan NPM tersebut ada, jika ada maka akan me- return halaman form-update, jika tidak ada maka akan me-return halaman not-found.

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

Tambahkan method updateSubmit pada class StudentController. Method tersebut untuk menyimpan data student yang sudah dimasukkan pada form.

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

Membuat sebuah method abstract updateStudent pada class interface StudentMapper. Gunakan anotasi @Update untuk mengeksekusi query meng-update data dalam database berdasarkan NPM yang diberikan. Parameter method tersebut menggunakan objek dari student dan menggunakan @Param untuk menangkap data dengan inisal "student".

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

Tambahkan method abstract updateStudent pada class interface StudentService. Pada parameter method tersebut menerima sebuah objek student

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

Pada class StudentServiceDatabase implementasikan method updateStudent dari class interface StudentService. Dalam method tersebut gunakan method updateStudent dari studentMapper untuk melakukan penghapusan data dari database.

```java
@Override
public void updateStudent(StudentModel student){
    log.info("Student" + student.getNpm() + "update");
    studentMapper.updateStudent(student);
}
```

Membuat link update pada viewall

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

Berikut adalah halaman form-update untuk mengubah data student berdasarkan NPM, halaman ini muncul ketika NPM yang diberikan valid.

```
<form action="/student/update/submit" th:object="${student}" method="post">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${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="update">Update</button>
    </div>
</form>
```

Berikut adalah halaman success-update, halaman ini muncul ketika data berhasil diubah dan disimpan kedalam database.

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

6. Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan

Pada class StudentController tambahkan sebuah method updateSubmitObject (tanpa mengubah method updateSubmit). Pada method ini mengganti paramternya , dimana parameternya menerima student dengan type data studentmodel. Kemudian

pemanggilan method updateStudent dengan parameter student diambil dari parameter updateSubmit.

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

Berikut adalah halaman form-update untuk mengubah data student berdasarkan NPM, halaman ini muncul ketika NPM yang diberikan valid.

```
<form action="/student/update/submit" th:object="${student}" method="post">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${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="update">Update</button>
    </div>
</form>
```