Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

Tutorial 4
Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Ringkasan dari materi

- Ketika ingin menggunakan tempat penyimpanan database maka kita perlu mengkonfigurasi database yang terhubung ke aplikasi kita dengan menambahkan informasi mulai dari IP, port, nama database yang akan digunakan
- Fungsi Library Lombok yaitu sebagai helper annotation pada project kita. Sedangkan Library MyBatis memiliki fungsi untuk menghubungkan project kita dengan MySQL. MyBatis membantu untuk melakukan koneksi dan generate query dengan helper annotation
- Untuk melakukan debugging pada Spring Boot dapat dilakukan dengan bantuan Library Slf4j. Library ini memungkinkan kita melakukan logging.
- Kelas Mapper digunakan method yang merepresentasikan operasi yang digunakan di database seperti SELECT, INSERT, DELETE, dan UPDATE. Kelas Mapper berupa interface yang akan diimplementasikan sebuah kelas Service.
- Untuk memudahkan kita dalam mengatur parameter dengan jumlah banyak yang akan digunakan untuk input suatu proses, kita dapat melemparnya dengan menggunakan object yang merangkum parameter-parameter yang diinginkan.

Latihan Menambahkan Delete

1. Tambahkan link untuk delete data student di viewall.html seperti potongan html berikut.

```
<div th:each="student,iterationStatus: ${students}">
        <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
                <h3 th:text="'NPM = ' + ${student.getNpm()}">Student NPM</h3>
                <h3 th:text="'Name = ' + ${student.getName()}">Student Name</h3>
                <h3 th:text="'GPA = ' + ${student.getGpa()}">Student GPA</h3>
                <a th:href="'/student/delete/' + ${student.getNpm()}"> Delete Data</a>
        <hr/>
</div>
```

2. Tambahkan method deleteStudent yang ada di class StudentMapper seperti berikut.

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

3. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase dengan memanggil method yang deleteStudent di StudentMapper seperti berikut.

```
@Override
public void deleteStudent (String npm)
{
        studentMapper.deleteStudent(npm);
}
```

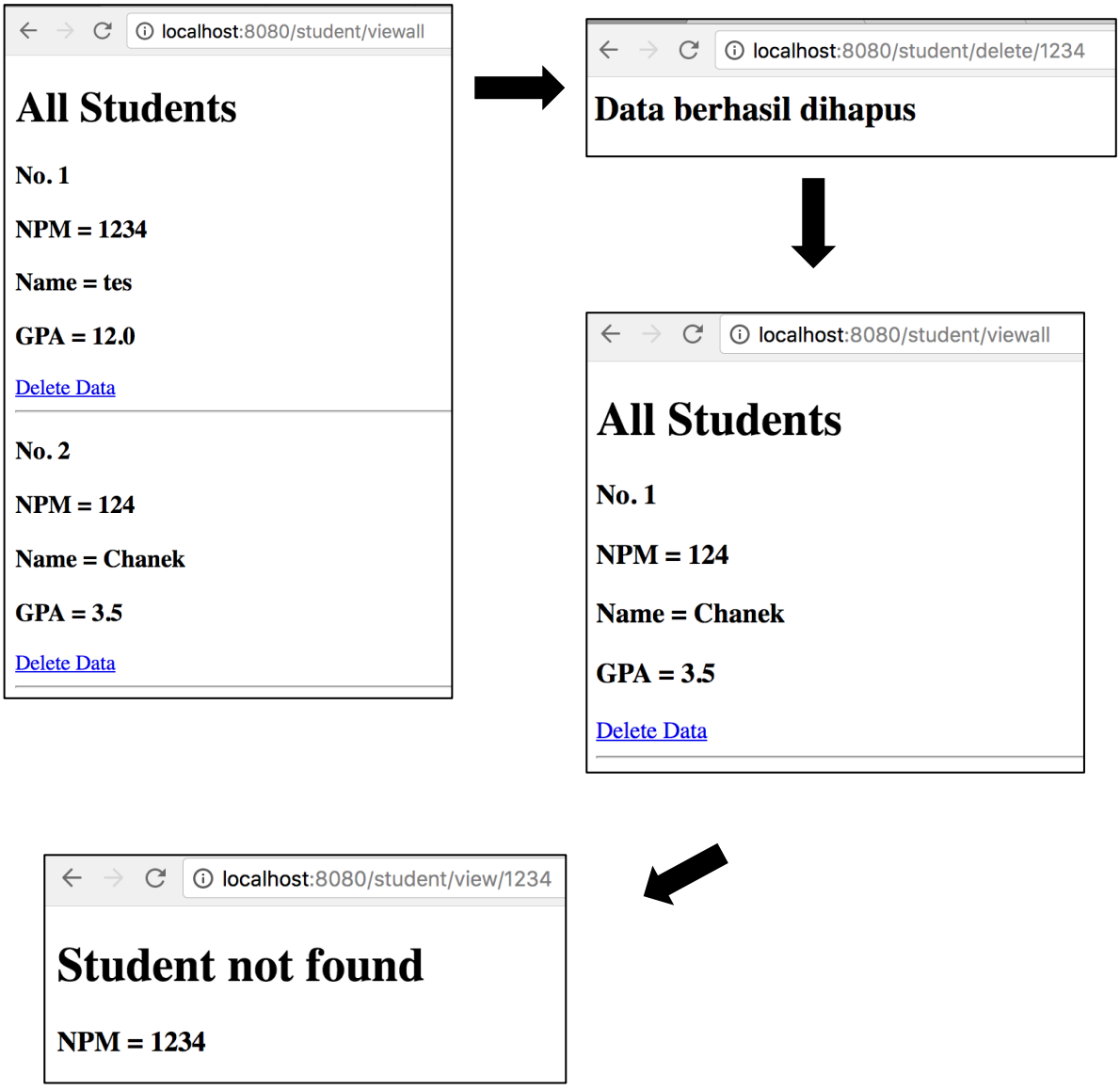
4. Lengkapi method delete pada class StudentController dengan menambahkan validasi mahasiswa berhasil di-delete dan mahasiswa tidak ditemukan seperti berikut.

Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

```
@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";
   }
}
```

5. Jalankan Spring Boot app dan lakukan beberapa insert. Berikut hasil simulasi dari implementasi method delete di atas.



Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

Latihan Menambahkan Update

1. Tambahkan method updateStudent pada class StudentMapper seperti berikut.

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

2. Tambahkan method updateStudent pada interface StudentService seperti berikut.

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

3. Implementasi method updateStudent pada class StudentServiceDatabase seperti berikut.

```
@Override
public void updateStudent(StudentModel student) {
        studentMapper.updateStudent(student);
}
```

4. Tambahkan link untuk update data student di viewall.html seperti potongan html berikut.

```
<div th:each="student,iterationStatus: ${students}">
    <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
        <h3 th:text="'NPM = ' + ${student.getNpm()}">Student NPM</h3>
        <h3 th:text="'Name = ' + ${student.getName()}">Student Name</h3>
        <h3 th:text="'GPA = ' + ${student.getGpa()}">Student GPA</h3>
        <a th:href="'/student/delete/' + ${student.getNpm()}"> Delete Data</a><br></br>
        <a th:href="'/student/update/' + ${student.getNpm()}"> Update Data</a>
    <hr/>
</div>
```

5. Tambahkan form-update.html di folder resource seperti form-add.html untuk menampung input value yang akan diupdate seperti berikut.

```
<!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" th:object="${student}">
            <div>
                  <label for="npm">NPM</label>
                  <input type="text" name="npm" readonly="true"
th:value="${student.getNpm()}”/>
            </div>
```

```
                <div>
                        <label for="name">Name</label>
                        <input type="text" name="name" th:value="${student.getName()}"/>
                </div>
                <div>
                        <label for="gpa">GPA</label>
                        <input type="text" name="gpa" th:value="${student.getGpa()}"/>
                </div>

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

</body>

</html>
```

6. Tambahkan success-update.html di folder resource seperti success-add.html untuk menampilkan informasi proses update berhasil seperti berikut.

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

7. Tambahkan method update pada class StudentController untuk mengambil dan menampilkan value dari mahasiswa yang dipilih untuk di-update di form-update.html seperti berikut, Tambahkan juga validasi jika mahasiswa tidak ditemukan.

```
  @RequestMapping("/student/update/{npm}")
  public String updatePath (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";
    }
  }
```

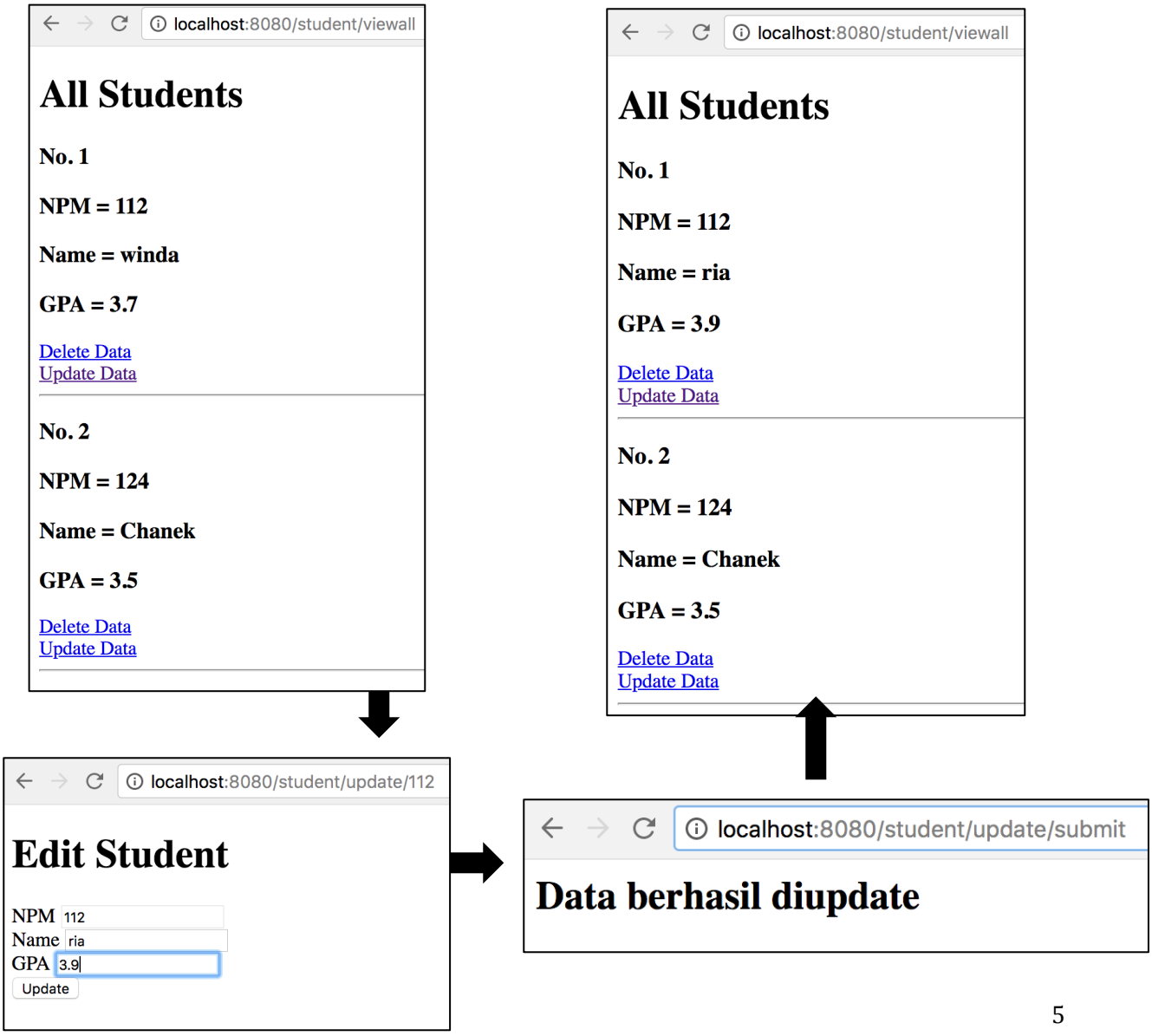
Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

8. Tambahkan method updateSubmit pada class StudentController dengan menggunakan method POST dan RequestParam npm, name, dan gpa. Method ini digunakan untuk memproses nilai update yang dimasukkan dan menyimpannya ke database seperti berikut,

```
@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";
}
```

6. Jalankan Spring Boot app dan lakukan update. Berikut hasil simulasi dari implementasi method update di atas.



Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

Latihan Menggunakan Object sebagai Parameter

Latihan ini diimplementasikan pada method update yang telah dilakukan pada latihan di atas. Akan tetapi, menggunakan RequestParam untuk menghandle form submit membuat banyak parameter pada method kita. Jika kita memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.

Berikut langkah untuk menerapkan object sebagai parameter pada method updateSubmit:

1. Comment method updateSubmit yang telah dibuat sebelumnya, lalu buat methof updateSubmit baru dengan parameter Student Model seperti berikut.

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

2. Update form-update.html sehingga dapat menampung object dari StudentModel yang akan di-update nilainya dengan Menambahkan th:object=”${student}” pada tag<form> . Nama object mahasiswa yang digunkan sama seperti nama attribute yang di-set di method updatePath.
3. Capture updated value di setiap field dengan menambahkan th:field=”*{[nama_field]}” pada setiap input.
4. Berikut contohnya form-update.html yang sudah disesuaikan berdasarkan informasi di atas.

```
<!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" th:object="${student}">
            <div>
                  <label for="npm">NPM</label>
                  <input type="text" name="npm" readonly="true" th:value="${student.getNpm()}"
th:field="*{npm}"/>
            </div>
            <div>
                  <label for="name">Name</label>
                  <input type="text" name="name" th:value="${student.getName()}"
th:field="*{name}"/>
            </div>
```

Nama : Winda Dumaria Simanjuntak
NPM : 1706107075

```
                <div>
                        <label for="gpa">GPA</label>
                        <input type="text" name="gpa" th:value="${student.getGpa()}"
th:field="*{gpa}"/>
                </div>
                <div>
                        <button type="submit" name="action" value="update">Update</button>
                </div>
        </form>
</body>
</html>
```

5. Jalankan Spring Boot app dan lakukan update. Berikut hasil simulasi dari implementasi method update menggunakan object sebagai param di atas.