Tutorial 4 Writeup

Arsitektur dan Pemrograman Aplikasi Perusahaan

Nama : Komang Adelia Nala Ratih

NPM : 1406557541

Kelas : B

Jelaskan apa saja hal yang Anda pelajari dari tutorial ini.

Saya mempelajari cara untuk melakukan debugging menggunakan Lombok, menghubungkan database menggunakan Spring, serta cara untuk melakukan operasi CRUD yang terhubung ke database. Selain itu, saya juga lebih memahami konsep penggunaan method POST dan GET.

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.

Cara melakukan validasinya adalah dengan mengecek pada objek yang ada pada parameter, apakah nilai-nilai yang harus ada pada database sudah tersedia. Validasi pada backend dapat dilakukan dengan menambahkan anotasi tambahan ataupun melakukan restriction.
Ya, validasi diperlukan apabila data tersebut harus ada, misalnya pada atribut yang merupakan primary key pada database.

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?

Method POST lebih baik digunakan dibanding method GET untuk pengiriman data yang bersifat rahasia, misalnya password, sedangkan method GET lebih baik digunakan untuk mengirimkan data yang tidak bersifat rahasia, seperti id_mahasiswa dan id_halaman karena datanya akan ditampilkan pada url. Selain itu, method POST dapat mengirimkan data yang tidak terbatas, sedangkan method GET tidka boleh lebih dari 2047 karakter.

Ya, butuh penanganan yang berbeda. RequestMapping pada POST menggunakan RequestMethod.POST, sedangkan pada GET menggunakan RequestMethod.GET.

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

Ya, satu method dapat menerima lebih dari satu jenis request method.

Jelaskan method yang Anda buat pada Latihan Menambahkan Delete.

1. Menambahkan **link** pada **viewall.html** untuk melakukan operasi delete.

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

2. Menambahkan method **deleteStudent** yang ada di class **StudentMapper**. Method ini menerima parameter student. Query ini bekerja dengan memilih student yang memiliki npm yang dimasukkan, lalu melakukan operasi delete pada student tersebut.

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

3. Melengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**. Method ini akan memanggil method selectStudent dengan npm sebagai parameternya. Kemudian, SQL untuk melakukan selectStudent akan dijalankan. Setelah melakukan selectStudent, SQL delete akan dijalankan untuk menghapus student dari database, dengan objek student yang sudah didapatkan pada method selectStudent sebagai parameternya. Log.info akan menulis log bahwa student berhasil dihapus.

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

4. Melengkapi method **delete** pada class **StudentController.** Method ini akan menerima npm dari student yang ingin dihapus. Kemudian, method selectStudent akan dijalankan untuk melihat apakah student dengan npm yang diberikan ada di database. Apabila student tersebut ada di database, method deleteStudent akan dijalankan, kemudian menampilkan halaman delete. Namun apabila student tersebut tidak ada di database, halaman not-found akan ditampilkan.

```
@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 {
        return "not-found";
    }
}
```

Jelaskan method yang Anda buat pada Latihan Menambahkan Update.

1. Menambahkan method **updateStudent** pada class **StudentMapper**. Method ini menerima parameter student. Query ini bekerja dengan memilih student yang memiliki npm yang dimasukkan, lalu melakukan operasi update dengan mengubah data nama dan gpa dari student tersebut.

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

2. Menambahkan method **updateStudent** pada interface **StudentService**. Method ini dimasukkan agar kelas yang mengimplementasi interface StudentService mengimplementasi method tersebut.

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

3. Mengimplementasi method **updateStudent** pada class **StudentServiceDatabase.** Method ini akan memanggil method updateStudent pada StudentMapper, kemudian menjalankan SQL update untuk mengubah data student pada database. Log.info akan menulis log bahwa student berhasil diubah.

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

4. Menambahkan **link** Update Data pada **viewall.html** untuk melakukan operasi update.

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

5. Menambahkan halaman **form-add.html** yang berisi form untuk memasukkan data yang ingin diubah. Kolom NPM pada form ini akan berisi npm student yang ingin diubah datanya. Fungsi read-only dilakukan pada kolom ini agar data NPM dari student tidak

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

dapat diubah.

6. Menambahkan halaman **success-update.html** untuk memberikan notifikasi bahwa proses update berhasil dilakukan.
7. Menambahkan method **update** pada **StudentController.** Method ini akan menerima npm dari student yang ingin diubah datanya. Kemudian, method selectStudent akan dijalankan untuk melihat apakah student dengan npm yang diberikan ada di database. Apabila student tersebut ada di database, data dari objek student akan diambil, lalu menampilkan halaman form-update. Namun apabila student tersebut tidak ada di database, halaman not-found akan ditampilkan.

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

8. Menambahkan method **updateSubmit** pada class **StudentController**. Method ini berfungsi untuk melakukan submit terhadap data-data yang telah dimasukkan pada form update untuk kemudian diubah di database. Method ini menerima parameter npm, nama, dan gpa, sebagai data-data yang ingin diubah pada objek student. Awalnya, method ini akan memanggil method selectStudent untuk melakukan pencarian student yang ingin diubah datanya dengan parameter npm. Kemudian, data nama dan gpa pada student tersebut akan diubah dengan melakukan set. Kemudian, method updateStudent yang terdapat pada kelas StudentServiceDatabase akan dipanggil untuk diteruskan ke kelas StudentMapper untuk melakukan perubahan. Setelah itu, halaman success-update akan ditampilkan untuk memberikan notifikasi bahwa proses perubahan berhasil dilakukan.

```
@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 = studentDAO.selectStudent(npm);
        student.setName(name);
        student.setGpa(gpa);
studentDAO.updateStudent(student);
 return "success-update";
}
```

Jelaskan method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter.

1. Menambahkan th:object=”${student}” pada tag <form> di view.
2. Menambahkan th:field=”*{[nama_field]}” pada setiap input.
3. Mengubah method **updateSubmit** pada **StudentController** yang hanya menerima parameter berupa StudentModel. Method ini berfungsi untuk melakukan submit terhadap data-data yang telah dimasukkan pada form update untuk kemudian diubah di database. Method ini menerima objek student sebagai parameternya. Karena parameter yang diterima adalah sebuah objek, data-data yang terdapat pada objek tersebut dapat diambil secara keseluruhan, sehingga tidak perlu menyebutkan atributnya satu per satu. Method ini akan menjalankan method updateStudent yang terdapat pada kelas StudentServiceDatabase untuk kemudian diteruskan ke kelas StudentMapper, dimana data-data pada student akan diubah. Selanjutnya, halaman success-update akan ditampilkan untuk memberikan notifikasi bahwa proses pengubahan data berhasil dilakukan.

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