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.

   Perlu, dikarenakan belum tentu atribut yang ada pada objek valid memenuhi syarat database. Jadi, sebelum dilakukan dicek terlebih dahulu apakah tidak melanggar *constraint* yang ada. Maka dari itu, untuk mengurangi kemungkinan *constraint* yang terjadi, akan lebih baik jika validasi dilakukan pada *front end* dan *back and.*

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?

   Salah satu alasannya adalah untuk menjaga keamanan data. Dengan menggunakan GET, maka semua data yang ingin disubmit akan terlihat pada url. Jika data form yang dikirim rahasia, misalnya password, maka password itu akan terlihat. Selain itu, apabila konten form yang dikirim banyak sekali, maka url akan menjadi sangat panjang. Untuk menggunakan method POST, maka digunakan 'RequestMethod.POST' sedangkan untuk GET, gunakan 'RequestMethod.GET'.

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

   Bisa, dengan syarat form method harus POST dan action berisi link yg nantinya akan di-get. Namun, menurut analisis saya, get tidak bisa berisi hasil dari form hanya link yang dicantumkan pada action saja.

Write-up

1. Method Delete
   Pada studentMapper

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

Method ini akan menerima npm yang nantinya akan digunakan oleh query. Query yang digunakan adalah 'DELETE FROM student WHERE npm=#{npm}' dimana, data student yang memiliki npm sesuai akan dihapus dari database.

Pada studentController

```
@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";
    }
}
```

Sebelum melakukan delete, method ini akan melakukan pengecekan terlebih dahulu. Pengecekan dilakukan dengan melakukan pemanggilan method selectStudent(), jika dihasilkan null berarti pelaksanaan delete gagal. Jika tidak null, maka method deleteStudent akan dipanggil yang kemudian akan menjalakan query untuk menghapus student yang diinginkan pada database.

Pada studentService

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

Pada studentServiceDatabase

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

Pada method ini, akan menerima npm untuk menandakan student yang akan dihapus. Terdapat log untuk debugging, mengecek apakan method tersebut berhasil dipanggil atau tidak.

2. Method Update

Pada studentMapper

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

Method ini akan menerima objek sstudeny yang nantinya akan digunakan oleh query. Query yang digunakan adalah 'UPDATE student SET name=#{name}, gpa=#{gpa}

WHERE npm=#{npm}’ dimana, data student yang memiliki npm sesuai akan diganti di database sesuai dengan yang didapat dari method.

Pada studentController

```
@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";
    }
}
```

Sebelum melakukan update, maka akan dilakukan pengecekan apakah student yang ingin di-update terdapat di database atau tidak. Jika ada, maka akan mengembalikan form-update.html

```
@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";
}
```

Method ini dipanggil pada saat submit form dari halaman form-update. Di dalamnya, akan memanggil method updateStudent() yang nantinya akan menjalakan query di database.

Pada studentService

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

Pada studentServiceDatabase

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

Pada method ini, akan menerima objek student untuk menandakan student yang akan di-update. Terdapat log untuk debugging, mengecek apakan method tersebut berhasil dipanggil atau tidak.

3. Method Object Sebagai Parameter

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

    return "success-update";
}
```

Method ini, dijalankan ketika mengklik tombol update dari form-update. Method langsung menerima objek student saja tanpa harus membuatnya terlebih dahulu sehingga, langsung saja panggil method updateStudent. Selanjutnya akan menampilkan halaman success-update.

Pada form-update.html

```
<body>

    <h1 class="page-header">Update 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.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="save">Update</button>
        </div>
    </form>

</body>
```

Pada form-update sudah disesuaikan agar bisa menghasilkan suatu objek student dengan ditambahkannya th:objecr=”${student}” pada tag form. Kemudian pada setiap input diberikan penanda sebagai atribut dari objek student.

Kesimpulan:
Saya mempelajari cara mengakses database, melakukan CRUD. Membuat method yang menerima parameter sebagai suatu objek, sehingga tidak perlu menerima banyak parameter. Saya juga mempelajar bagaimana mengirim form dalam bentuk POST/GET.

Sumber:
https://stackoverflow.com/questions/2749406/post-and-get-at-the-same-time-in-php