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? Apakah validasi diperlukan?
   Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.
   Validasi input yang bersifat opsional dan input yang bersifat required dapat dilakukan di kode back-end. Validasi tersebut dapat berupa pengecekan conditional dengan menggunakan if (di cek apakah field A kosong atau tidak).
   Validasi input pastinya dirasa cukup penting untuk dilakukan, hal ini untuk mencegah terjadinya error ketika pemrosesan perintah dari user dan ketika data yang dimasukkan oleh user akan ditambahkan ke dalam database.

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?
   Menurut saya, penggunaan POST method lebih aman daripada penggunaan GET method. Hal ini disebabkan oleh pengiriman data menggunakan POST method langsung dikirim ke resource yang dituju tanpa ditampilkan di URL. Data tersebut disertakan di dalam body dari request tersebut. Hal ini dilakukan untuk meminimalisasi serangan dari luar atau penyalahgunaan data yang dikirim melalui request tersebut.
   Pada header controller dapat penanganan yang berbeda. Karena @RequestMapping secara default memetakan semua operasi HTTP, seperti GET, POST, dan sebagainya, pada header controller method ybs harus diberikan keterangan seperti berikut.

```
@RequestMapping(value="/student/update/submit", method =
RequestMethod.POST)
```

   Selain itu, ada cara lain untuk penanganan penggunaan POST method. Header diatas dapat diubah menjadi seperti berikut.

```
@PostMapping("/student/update/submit")
```

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Ya, hal tersebut memungkinkan. Ada sebuah array ($_REQUEST) yang dapat menggabungkan nilai GET dan POST. Namun, hal tersebut tidak disarankan. Ada cara yang lebih baik yang dapat dilakukan, yaitu menangani kedua request tersebut secara manual di kode back-end.

LESSON LEARNED

Hal yang saya pelajari pada tutorial kali ini adalah saya mempelajari bagaimana menggunakan database dalam arsitektur MVC dan saya juga belajar mengenai debugging dalam project spring menggunakan Slfj4 untuk menampilkan variabel log.

LATIHAN MENAMBAHKAN DELETE

Langkah pertama yang perlu dilakukan adalah menambahkan method deleteStudent pada StudentMapper.

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

Langkah selanjutnya adalah melengkapi method deleteStudent pada StudentServiceDatabase yang telah di-override.

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

Baris log.info merupakan salah satu cara untuk melakukan debugging. String pada log.info tersebut akan ditampilkan pada console eclipse ketika method ybs dijalankan.

Kemudian, melengkapi method delete pada StudentController beserta dengan validasi data student pada database.

```
@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";
  }
}
```

Pada if pertama, dilakukan pengecekan apakah ada data student dengan npm yang disertakan pada @PathVariable. Jika npm tersebut ada, data student dengan npm tersebut akan dihapus. Jika tidak ada, akan ditampilkan halaman not-found.

Terakhir, tambahkan link untuk menghapus pada halaman viewall.html.

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

Selanjutnya, aplikasi dapat dijalankan dan dapat menghapus data berdasarkan NPM dari student ybs.

LATIHAN MENAMBAHKAN UPDATE

Langkah pertama yang perlu dilakukan sama dengan latihan sebelumnya, yaitu menambahkan method updateStudent pada StudentMapper.

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

Selanjutnya adalah menambahkan method updateStudent pada StudentService dan melengkapi method tersebut pada StudentServiceDatabase.

```
public interface StudentService {
...
  void updateStudent(StudentModel student);
...
}
```

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

Kemudian, tambahkan link untuk menghapus pada halaman viewall.html.

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

Langkah selanjutnya adalah membuat halaman form-update dengan cara mengkopi halaman form-add dengan mengubah beberapa detail seperti penjelasan soal.

```
<!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">Problem Editor</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="update">
        Update</button>
    </div>
  </form>
</body>

</html>
```

Ada penambahan atribut `th:object` pada tag `<form>`, `th:field` pada seluruh tag `<input>`, dan `readonly` pada tag `<input>` milik NPM.

Berikutnya adalah tambahkan method update pada StudentController beserta dengan validasi data student pada database.

```
@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";
  }
```

```
}
```

Pada if pertama, dilakukan pengecekan apakah ada data student dengan npm yang disertakan pada @PathVariable. Jika npm tersebut ada, data student dengan npm tersebut akan dihapus. Jika tidak ada, akan ditampilkan halaman not-found.

Terakhir, tambahkan method updateSubmit pada StudentController.

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

Penggunaan `method = RequestMethod.POST` pada header bertujuan untuk menerima request pada halaman form-update.html yang berupa method POST. Jika pada tutorial sebelumnya menerima data dalam bentuk @PathVariable, pada tutorial kali ini, penerimaan data dalam bentuk object menggunakan anotasi @ModelAttribute.

Selanjutnya, aplikasi dapat dijalankan dan dapat mengubah seluruh data dari student ybs.