Anindito Izdihardian Wibisono
1506757466
APAP – C

Dalam tutorial kali ini, saya belajar mengenai implementasi database kedalam project Spring Boot serta penggunaan Lombok dan MyBatis. Lombok merupakan library eksternal Java yang mempermudah penulisan kode dengan fitur-fitur seperti membuat metode getter/setter secara otomatis lewat anotasi @Data, sementara MyBatis digunakan untuk menambah logging guna mempermudah debugging dan tracing kode yang bermasalah. Untuk implementasi database, saya belajar cara untuk menyiapkan metode dasar operasi terhadap database SQL seperti SELECT, UPDATE, dan DELETE. Implementasi tersebut terbagi di beberapa file java yang ada pada project.

LATIHAN 1: METODE DELETE

Metode yang terdapat pada StudentMapper:

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

Metode yang terdapat pada StudentServiceDatabase:

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

Metode yang terdapat 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";
          }
    }
```

Saat pengguna mengklik tombol delete data pada student/viewall, controller lewat method delete menerima request untuk menghapus data entri yang dipilih. Metode mengecek dulu apakah NPM yang dimasukkan ada pada database. Jika ada, maka dijalankan metode delete pada StudentServiceDatabase dan mapper mengirim view konfirmasi bahwa data terhapus. Jika tidak, mapper mengirim view bahwa NPM yang dimasukkan tidak ditemukan.

LATIHAN 2: METODE UPDATE

Metode yang terdapat pada StudentMapper:

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

Metode yang terdapat pada StudentServiceDatabase:

```
@Override
    public void updateStudent (String npm, String name, double gpa)
    {
     log.info("student " + npm + " updated");
     studentMapper.updateStudent(npm, name, gpa);
    }
```

Metode yang terdapat 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";
          }
    }
```

```
@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) {
     studentDAO.updateStudent(npm, name, gpa);
     return "success-update";
    }
```

Saat pengguna mengklik tombol update data, metode update pada controller menerima request update, lalu mengecek keberadaan NPM yang menjadi masukan. Jika ada, maka form update ditampilkan; jika tidak, maka halaman “not found” ditampilkan, mirip seperti pada metode delete diatas. Metode update meneruskan atribut object student yang diambil kepada form update untuk selanjutnya diubah pengguna. Setelah pengguna selesai dan menyimpan data, metode updateSubmit menerima masukan pada form sebagai parameter dan meneruskannya ke metode updateStudent pada StudentServiceDatabase, yang menjalankan kueri SQL sesuai masukan.

LATIHAN 3: UPDATE DENGAN OBJECT SEBAGAI PARAMETER

Perubahan dilakukan pada metode updateSubmit seperti berikut:

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

Disini, metode menerima object penuh student sebagai parameter, bukan atribut-atribut individual yang dikirim oleh view. Tujuannya adalah untuk merampingkan data yang dikirim, terutama apabila object memiliki sangat banyak atribut yang ingin diubah.

PERTANYAAN TUTORIAL

1. Dengan cara mengecek apakah field yang diisi bernilai null, karena field kosong pada form akan mengembalikan null.

2. Karena metode GET kurang aman dibanding POST untuk pengiriman data. Metode GET akan menyebabkan data yang dikirim tercantum di URI request, dimana URI ini bisa tercatat oleh logger atau webcrawler. Mengenai form yang dikirim menggunakan method berbeda, perlu penanganan berbeda pula. Di header, gunakan anotasi @RequestHeader; di body, gunakan anotasi @RequestParam.

3. Bisa, dengan menspesifikasikan method yang diinginkan setelah anotasi @RequestMapping. Untuk kasus ini contoh metodenya adalah sebagai berikut:

```
@RequestMapping(value = "/example", method = { RequestMethod.GET, RequestMethod.POST })
public String exampleMethod(
// isi parameter
) {
     // isi logic kode
}
```