Tutorial 4

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Write-up:

Pada tutorial ke-4 kali ini, saya mempelajari bagaimana melakukan manipulasi data-data yang disimpan pada database dalam project Spring Boot. Pada proses manipulasi data-data tersebut dapat dilakukan dengan memanfaatkan method POST. Selain itu, saya juga memahami lebih dalam mengenai konsep method GET dan POST dan bagaimana memanfaatkan *log* yang dapat digunakan sebagai alat untuk *debugging*.

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.

   Validasi perlu dilakukan jika *field* yang diisi memang dibutuhkan untuk ada, contohnya seperti nilai-nilai yang nantinya menjadi nilai atribut-atribut yang merupakan *primary key* di database. Validasi dapat dilakukan dengan menggunakan objek yang ada pada parameter. Sebelum dilakukan operasi lebih lanjut pada objek tersebut, nilai-nilai yang dirasa perlu ada dapat dicek terlebih dahulu ada atau tidak nilainya. Jika nilainya tidak ada, dapat dialihkan kembali ke halaman form untuk diisi kembali oleh *user*.

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?**
   Karena jika menggunakan method GET, seluruh nilai-nilai atribut akan ditampilkan melalui URL yang mana bukan merupakan hal yang tepat dilakukan jika form tersebut mengandung *field-field* dengan data-data yang sangat privasi, contohnya seperti *password.* Jika ingin menggunakan method POST, maka perlu dilakukannya RequestMapping method yang perlu ditambahkan dengan nilai "`RequestMethod.POST`". Sedangkan jika ingin menggunakan method GET, nilainya menjadi "`RequestMethod.GET`".

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

   Hal ini dapat dilakukan dengan menggunakan method POST. Hal ini disebabkan karena method GET menerima input hanya dari *query string*. Sedangkan method POST menerima input dari *request body* dan juga *query string*, yang menyebabkan method POST merupakan superset dari method GET. Sehingga ketika menggunakan method POST, secara tidak langsung juga menggunakan method GET karena method POST juga menerima input melalui *query string.*

Penjelasan Method-Method Pada Latihan Menambahkan Delete

1. Method **deleteStudent** pada class **StudentMapper**

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

   Method deleteStudent pada kelas StudentMapper ini pada awalnya menerima parameter npm yang digunakan sebagai penanda objek student mana yang akan dihapus. Selanjutnya karena menggunakan database, maka diperlukan *query* untuk menghapus objek student tersebut pada database. Karena npm yang digunakan sebagai penanda, maka parameter npm yang telah didapatkan sebelumnya diletakkan pada *where clause*. Sehingga, *query*-nya menjadi "`DELETE FROM student where npm = #{npm}`".

2. Method **deleteStudent** pada class **StudentServiceDatabase**

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

   Pada method deleteStudent di kelas StudentServiceDatabase ini, method menerima npm sebagai penanda objek student mana yang akan dihapus. Pada method ini juga ditambahkan log yang digunakan untuk *debugging* dan penanda jika objek student tersebut memang benar-benar telah terhapus. Setelah itu, method ini mengirim npm yang didapatkan pada parameter dan memanggil method deleteStudent yang telah dibuat sebelumnya pada kelas StudentMapper, untuk dilakukan penghapusan objek student di database.

3. Method **delete** pada class **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";
    }
}
```

Proses mendapatkan nilai dari npm yang diinput melalui URL oleh *user* dikelola pada kelas StudentController ini. Nilai yang didapat pada awalnya akan dicek terlebih dahulu di database apakah ada atau tidak. Jika objek tersebut ada, maka kelas StudentController ini akan memanggil method deleteStudent yang telah dibuat pada kelas StudentServiceDatabase yang selanjutnya diteruskan ke kelas StudentMapper untuk dilakukan proses penghapusan. Setelah itu, akan ditampilkan halaman delete yang menyatakan bahwa proses penghapusan telah berhasil dilakukan. Sedangkan jika objek student dengan npm yang dimaksud tidak ada, maka halaman akan diteruskan ke halaman not-found yang menyatakan bahwa objek student dengan npm yang diinput tidak dapat ditemukan.

Penjelasan Method-Method Pada Latihan Menambahkan Update

1. Method **updateStudent** pada class **StudentMapper**

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

Method updateStudent pada kelas StudentMapper ini pada awalnya menerima parameter sebuah objek student yang akan diubah datanya. Selanjutnya karena menggunakan database, maka diperlukan *query* untuk mengubah data pada objek student tersebut pada database dengan npm sebagai patokan objek student yang akan diubah. Sehingga, *query*-nya menjadi "`UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}`".

2. Method **updateStudent** pada interface **StudentService**

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

Method ini diletakkan pada *interface* StudentService untuk selanjutnya nanti diimplementasikan oleh kelas-kelas yang meng-*implements interface* tersebut.

3. Method **updateStudent** pada class **StudentServiceDatabase**

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

Pada method updateStudent di kelas StudentServiceDatabase ini, method menerima parameter objek student yang akan diubah datanya. Pada method ini juga ditambahkan log yang digunakan untuk *debugging* dan penanda jika objek student tersebut memang benar-benar telah terubah. Setelah itu, method ini mengirim objek student yang didapatkan pada parameter dan memanggil method updateStudent yang telah dibuat sebelumnya pada kelas StudentMapper, untuk dilakukan pengubahan data objek student tersebut pada database.

4. Method **update** pada class **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";
    }
}
```

Proses mendapatkan objek student yang akan diubah data-datanya didapatkan melalui nilai npm yang diinput melalui URL oleh *user,* yang mana dikelola pada kelas StudentController ini. Nilai yang didapat pada awalnya akan dicek terlebih dahulu di database apakah ada atau tidak. Jika objek tersebut ada, maka kelas StudentController ini akan menampilkan halaman form-update dimana *user* dapat mengubah data-data dari objek student dengan npm yang dimaksud. Sedangkan jika objek student dengan npm yang dimaksud tidak ada, maka halaman akan diteruskan

ke halaman not-found yang menyatakan bahwa objek student dengan npm yang diinput tidak dapat ditemukan.

5. Method **updateSubmit** pada class **StudentController**

```
@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";
}
```

Method updateSubmit pada kelas StudentController ini berfungsi untuk meng-*submit* data-data yang sudah diubah oleh *user* untuk selanjutnya disimpan pada database. Pada awalnya method melakukan pencarian objek student yang akan diubah datanya dengan patokan parameter npm. Setelah itu, method menge-*set* nilai nama dan gpa dengan nilai baru berdasarkan data-data yang diinput oleh *user*. Setelah itu, method ini akan memanggil method updateStudent yang telah dibuat pada kelas StudentServiceDatabase yang selanjutnya diteruskan ke kelas StudentMapper untuk dilakukan proses pengubahan. Setelah itu, method menampilkan halaman success-update yang menyatakan bahwa proses pengubahan telah berhasil dilakukan.

Penjelasan Method-Method Pada Latihan Menggunakan Object Sebagai Parameter

1. Method **updateSubmit** pada class **StudentController**

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

Method updateSubmit pada kelas StudentController setelah diupdate dengan menggunakan *object* sebagai parameter menjadi seperti gambar di atas. Method sudah langsung menerima objek student yang data-datanya sudah diubah secara otomatis. Method hanya tinggal memanggil method updateStudent dengan parameter objek student yang sudah diubah data-datanya yang nantinya akan diproses untuk dilakukan perubahan di database. Setelah itu, method menampilkan halaman success-update yang menyatakan bahwa proses pengubahan telah berhasil dilakukan.

References

<https://stackoverflow.com/questions/504947/when-should-i-use-get-or-post-method-whats-the-difference-between-them> diakses pada tanggal 29 September 2017 00:57 WIB.