Sukrian Syahnel Putra
1306402066
ADPAP - B

Write-Up Tutorial 4: Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Ringkasan

Di dalam tutorial ini saya mempelajari cara mengintegrasikan aplikasi di Project Spring Boot agar dapat menggunakan database, salah satu contohnya adalah MySQL. Selain itu, saya juga mempelajari cara melakukan debugging pada aplikasi.

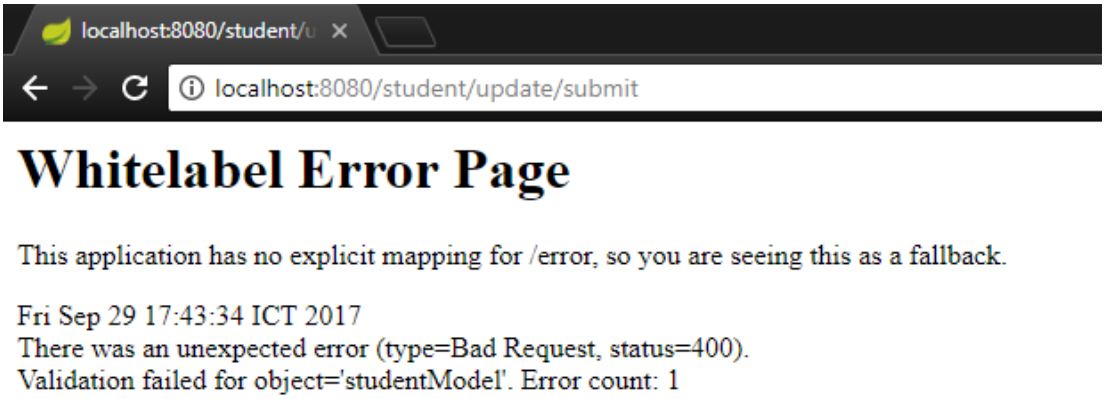
Proses debugging yang dipelajari pada tutorial ini menggunakan library @Sl4fj yang dimiliki oleh Project Lombok. Debugging yang dipelajari pada tutorial ini adalah dengan menggunakan fitur logging yang tersedia di Project Lombok. Cara menggunakan debugging dengan Project Lombok yaitu dengan menambahkan @Sl4fj pada kelas yang ingin di-debug, lalu menggunakan variabel bernama log yang menggunakan tiga jenis method (log.info, log.debug, log.delete).

Untuk mengintegrasikan aplikasi dengan database, digunakan anotasi @Mapper. Mapper berfungsi menjalankan query SQL yang terasosiasi dengan method yang bersangkutan. Misalnya, untuk menjalankan method yang menghapus data digunakan anotasi @Delete, untuk update digunakan anotasi @Update, dll. Anotasi Mapper diletakkan di interface Mapper pada package DAO (Data Access Object) yang berfungsi menjembatani aplikasi dan database.

Di dalam tutorial ini hal-hal yang telah saya pelajari antara lain mempelajari cara membuat log debugging, membuat form dengan method POST, menggunakan parameter object.

Jawaban dari setiap pertanyaan

1. Validasi input untuk sebuah objek tetap diperlukan guna menghindari kemungkinan adanya bad request yang muncul. Hal ini karena ketika saya mencoba melakukan update dengan mengosongkan nama dan GPA muncul error sebagai berikut.



   Error diatas menunjukkan bahwa validasi masih diperlukan untuk request dengan Object sebagai parameter. Untuk mencegah munculnya error yang tidak diinginkan, di dalam Spring terdapat anotasi @Valid yang dapat digunakan untuk memvalidasi data yang dimasukkan oleh user. Di Javax terdapat anotasi-anotasi constraint seperti @NotNull, @Size dan @Min untuk memvalidasi input yang dimasukkan oleh user. Constraint seperti @NotNull dan @Size diterapkan ke dalam class dari Object, sementara @Valid diimplementasikan di dalam Controller.
2. POST dan GET memiliki beberapa perbedaan sebagai berikut (diffen.com):

Sukrian Syahnel Putra
1306402066
ADPAP - B

a. Parameter pada POST tidak disimpan dalam history browser, sementara parameter GET disimpan karena termasuk bagian dari URL.
b. POST tidak dapat di-bookmark, sementara GET dapat di-bookmark.
c. Request resubmit form pada POST mengakibatkan sistem untuk mengingatkan user apakah ia ingin resubmit data, sementara request GET dieksekusi ulang, namun mungkin tidak di-resubmit ke server jika sudah ada.
d. Encoding pada GET adalah application/x-www-form-urlencoded, sementara pada POST encoding adalah application/x-www-form-urlencoded dan multipart/form-data (multipart encoding untuk data biner).
e. Parameter GET terbatas kepada apa saja yang dapat dimasukkan ke URL line, sementara parameter POST dapat mengirimkan parameter yang lebih luas, termasuk melakukan upload file.
f. GET lebih mudah diretas oleh script kiddies, sementara POST lebih sulit untuk diretas.
g. GET terbatas pada karakter ASCII, sementara POST tidak memiliki Batasan karakter, termasuk dapat mengirimkan data biner.
h. GET dapat di-cache, sementara POST tidak.

Beberapa pertimbangan yang telah disebutkan tadi menjadi penyebab mengapa POST lebih sering digunakan untuk submisi dibandingkan GET. Akan tetapi, penanganan kedua method terbilang sama untuk setiap jenis form, hanya perlu mengganti method saja di form.

3. Satu method hanya dapat menerima satu jenis request method. Hal ini dikarenakan terdapat perbedaan yang telah disebukan di nomor 2 mengenai kedua jenis method, sehingga satu method hanya dapat menggunakan satu request method.

Latihan menambahkan Delete

Dalam latihan menambahkan Delete, terdapat implementasi method di beberapa class dan interface di dalam aplikasi.

Pertama-tama, saya menambahkan line berikut pada div dengan th:each di template viewall.html.

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

Implementasi selanjutnya pada aplikasi dibuat pada interface StudentMapper dengan method deleteStudent sebagai berikut.

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

Method deleteStudent sekedar merupakan kerangka yang juga mengikuti method deleteStudent di interface StudentService yang telah disediakan, sehingga method ini juga me-return void (tidak me-return apa-apa). Pada anotasi @Delete, tersedia query MySQL yang berfungsi menghapus data student dengan NPM yang bersangkutan, karena NPM pada sistem ini merupakan primary key sehingga dapat melakukan delete secara akurat. Method ini menerima parameter String NPM dari user.

Implementasi selanjutnya adalah pada method deleteStudent di class StudentServiceDatabase, sebagai berikut.

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

```
    }
```

Ini adalah implementasi method kerangka deleteStudent pada interface StudentService. Pertama kali dilakukan logging untuk memastikan pada sistem apakah student dengan NPM bersangkutan berhasil dihapus yang kemudian mencatatkan hasilnya di konsol. Jika tertulis pada konsol, maka operasi penghapusan student berdasarkan NPM telah berjalan dengan baik. Method ini memanggil method deleteStudent pada studentMapper untuk menjalankan query yang berfungsi menghapus student dengan NPM yang sesuai.

Implementasi selanjutnya adalah pada class StudentController dengan menambahkan method delete sebagai berikut.

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
   StudentModel student = studentDAO.selectStudent(npm);
   if(student == null) {
          model.addAttribute("npm", npm);
          return "not-found";
   }
     studentDAO.deleteStudent(npm);

     return "delete";
}
```

Method ini menerima input npm dari user dengan jenis PathVariable. RequestMapping pada method ini adalah /student/delete/[npm student]. Untuk validasi, pertama saya memanggil StudentModel yang bersangkutan untuk memastikan apakah ada student tersebut di dalam database. Pada if clause yang mengecek jika tidak ada student yang bersangkutan, maka atribut npm dimasukkan dan sistem akan segera me-return halaman not-found yang ada di templates untuk memberitahukan user. Jika ada student tersebut, maka tidak akan masuk if clause dan langsung memerintahkan studentDAO untuk melakukan penghapusan student yang bersangkutan lalu me-return halaman delete di templates.

Latihan menambahkan Update

Untuk latihan menambahkan Update, ada beberapa method yang ditambahkan dan diimplementasi pada beberapa class dan interface.

Pertama, saya menambahkan method updateStudent pada interface StudentMapper, seperti tertera di bawah ini.

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

Query SQL di atas berfungsi mencari data dengan NPM yang sesuai untuk kemudian diperbarui nama saja, gpa saja, atau kedua data tersebut. Method ini menerima parameter tipe Object berupa StudentModel.

Selanjutnya, pada interface StudentService saya menambahkan method updateStudent yang menerima parameter berupa Object StudentModel, seperti tertera di bawah ini.

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

Method kosong ini nantinya digunakan untuk kemudian diimplementasikan pada method updateStudent di class StudentServiceDatabase.

Sukrian Syahnel Putra
1306402066
ADPAP - B
Selanjutya, saya mengimplementasikan method updateStudent pada class StudentServiceDatabase, seperti tertera di bawah ini.

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

Pada method ini, dilakukan logging terlebih dahulu untuk memastikan apakah data operasi update berhasil dijalankan. Selanjutnya, method ini kemudian memerintahkan studentMapper untuk melakukan update terhadap data student.

Pada template viewall.html, saya menambahkan line berikut untuk meng-update data.

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

Kemudian saya meng-copy isi dari form-add.html untuk kemudian dijadikan form-update.html. Saya mengubah setiap metode sesuai keperluan, seperti mengubah form method menjadi POST, action menjadi /student/update/submit, dan mengganti line setiap input bracket menjadi sebagai berikut.

```
<input type="text" name="npm" readonly="true" th:value="${student.npm}" />

<input type="text" name="name" th:value="${student.name}" />

<input type="text" name="gpa" th:value="${student.gpa}" />
```

Variabel read-only berfungsi mencegah user dari secara tidak sengaja mengubah NPM, karena NPM merupakan fixed value. Sementara itu, th:value merepresentasikan data dari student yang akan di-update.

Selanjutnya, saya membuat view success-update.html yang di-copypaste dan dimodifikasi dari add-update sebagai page target setelah data berhasil di-update.

Saya kemudian mengimplementasi method update dalam class StudentController, seperti tertera di bawah ini.

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm) {
   StudentModel student = studentDAO.selectStudent(npm);
   if(student == null) {
	model.addAttribute("npm", npm);
	return "not-found";
   }
   model.addAttribute("student", student);
   return "form-update";
}
```

Method ini menerima input npm dari user dengan jenis PathVariable. RequestMapping pada method ini adalah /student/update/[npm student]. Untuk melakukan validasi untuk mencegah update Object null, dilakukan validasi sama seperti pada method delete. Setelah validasi, method ini memerintahkan model untuk memasukkan StudentModel dari student ke dalam model, untuk kemudian diarahkan ke form-update.

Hal selanjutnya yang dilakukan adalah melengkapi method updateSubmit pada StudentController. Method yang saya implementasikan adalah sebagai berikut.

Sukrian Syahnel Putra
1306402066
ADPAP - B

```
    @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";
    }
```

Pada anotasi RequestMapping, dilengkapi value dari request mapping, yaitu /student/update/submit/ dan request method yang bertipe POST. Method ini menerima tiga request parameter, yaitu npm, name, dan GPA. Sebuah object StudentModel dibentuk dari tiga parameter ini, kemudian method memanggil studentDAO untuk melakukan update terhadap student dengan npm yang sama dengan student yang sedang di-update. Method ini mengarahkan user ke view success-update setelah menyelesaikan update pada database SQL.

Latihan menggunakan Object sebagai Parameter

Untuk menggunakan Object sebagai parameter, saya mengujikan ini pada proses update student.

Hal pertama yang dilakukan adalah menambahkan th:object pada <form> yang tersedia di form-update seperti tertera di bawah ini.

<form action="/student/update/submit" method="post" th:object="${student}">

Hal selanjutnya adalah menambahkan th:field pada masing-masing atribut dari student yang ada di <input> seperti berikut.

<input type="text" name="npm" readonly="true" th:field="*{npm}" />

<input type="text" name="name" th:field="*{name}" />

<input type="text" name="gpa" th:field="*{gpa}" />

Hal selanjutnya adalah dengan mengubah parameter yang diterima method updateSubmit pada class StudentController menjadi hanya menerima sebuah Object StudentModel, seperti di bawah ini.

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

Perbedaan antara method ini dengan method sebelumnya yang menerima tiga parameter adalah StudentModel baru tidak perlu dibuat kembali dan panjang kode menjadi lebih pendek karena hanya perlu menerima satu parameter berupa StudentModel yang telah diisikan, yang juga menghapuskan keperluan untuk meng-construct class baru untuk Object StudentModel. Panjang kode biasanya identik dengan performa aplikasi, sehingga secara teori menggunakan Object sebagai parameter tidak hanya menyebkan aplikasi berjalan sescara efisien, akan tetapi juga lebih cepat memproses perintah.

Referensi

Validating Form Input. (n.d.). Retrieved September 29, 2017, from https://spring.io/guides/gs/validating-form-input/

Sukrian Syahnel Putra
1306402066
ADPAP - B

Handling Form Submission. (n.d.). Retrieved September 29, 2017, from https://spring.io/guides/gs/handling-form-submission/

GET vs POST. (n.d.). Retrieved September 29, 2017, from http://www.diffen.com/difference/GET-vs-POST-HTTP-Requests