FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA

Tutorial 4
CSIM603026 – Arsitektur dan Pemrograman Aplikasi Perusahaan
Semester Ganjil 2017/2018

Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot

Versi: 1, 26 September 2017

Requirement

Pada tutorial kali ini Anda akan menggunakan dua library eksternal, yaitu MyBatis dan Lombok. Berikut langkah yang perlu anda lakukan untuk mengerjakan tutorial:

1. Unduh terlebih dahulu berkas APAPTutorial04.zip yang ada di Scele.
2. Bagi yang tidak menggunakan Eclipse:
    a. Unzip berkas tersebut dan jalankan command maven seperti biasa, library-library tersebut sudah dimasukkan ke depedencies maven di berkas pom.xml.
3. Bagi yang menggunakan Eclipse:
    a. Unduh terlebih dahulu lombok.jar di https://projectlombok.org/downloads/lombok.jar
    b. Tekan 2 kali atau jalankan di command line dengan java -jar lombok.jar
    c. Lakukan instalasi pada Eclipse Anda
    d. Buka Eclipse Anda, lakukan file > import > Existing Projects Into Workspace > Root directory diisi dengan hasil unzip berkas APAPTutorial4> Finish
4. Sebelum menjalankan program Anda, jalankan XAMPP atau database server Anda.
5. Import query SQL berikut ke dalam database MySQL Anda menggunakan phpMyAdmin atau MySQL shell. Buatlah sebuah database terlebih dahulu.

```
create database eaap default character set utf8;
grant all privileges on eaap .* to 'eaap_user'@'%' identified by 'eaap_pwd';
grant all privileges on eaap .* to 'eaap_user'@'localhost' identified by 'eaap_pwd';
flush privileges;
use eaap;

CREATE TABLE `student` (
  `npm` varchar(20) NOT NULL,
  `name` varchar(45) DEFAULT NULL,
  `gpa` double DEFAULT NULL,
  PRIMARY KEY (`npm`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6. Setelah mengikuti langkah-langkah tersebut, anda sudah bisa mengikuti tutorial selanjutnya

Penjelasan

Slf4j

Untuk melakukan debugging pada Spring Boot dapat dilakukan dengan melakukan System.out.println dan pesan Anda akan tercetak di console. Cara yang lebih baik dan biasa digunakan di *enterprise* adalah dengan menggunakan logging. Salah satu library yang biasa digunakan adalah Slf4j.

Pada tutorial ini Anda akan menggunakan Slf4j yang ada pada library eksternal Lombok. Untuk menggunakannya, cukup menambahkan anotasi @Slf4j di atas kelas yang ingin diberikan log. Dengan memberikan anotasi tersebut, otomatis terdapat sebuah variabel bernama **log** yang dapat digunakan. Method-method yang biasa digunakan adalah **log.info, log.debug,** dan **log.error.** Nantinya jika program Anda sudah cukup besar, Anda bisa memfilter log berdasarkan jenisnya agar lebih mudah dibaca. Biasakan untuk menggunakan method-method ini untuk melakukan debugging program selanjutnya.

Untuk memberikan argumen ke log, gunakan formatted string dengan {} dan berikan argumen di belakang dipisahkan dengan koma. Contoh:

```
log.info("Student {} name updated to {}", student.getNPM(), student.getName());
```

Latihan Menambahkan Delete

1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.
2. Pada viewall.html tambahkan

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

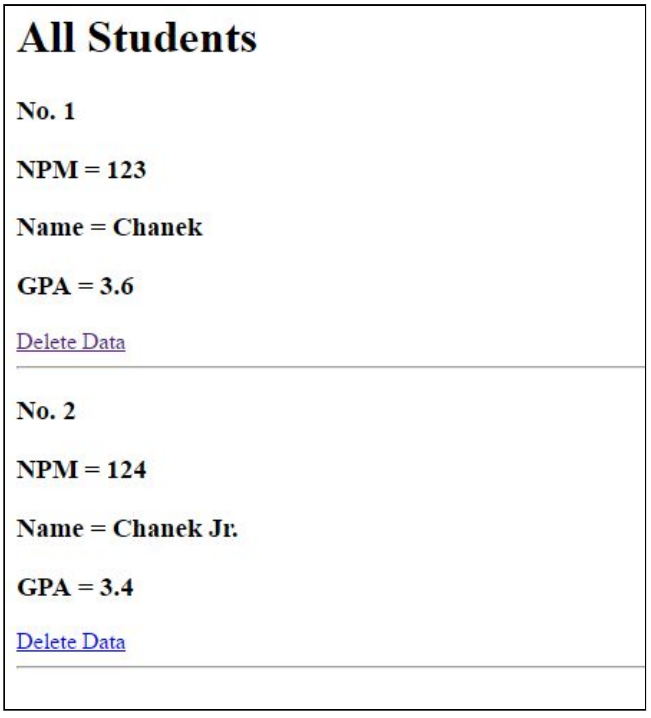
   Di dalam div dengan **th:each**
3. Tambahkan method **deleteStudent** yang ada di class **StudentMapper**
   a. Tambahkan method delete student yang menerima parameter NPM.
   a. Tambahkan annotation delete di atas dan SQL untuk menghapus

```
@Delete("[LENGKAPI]")
```

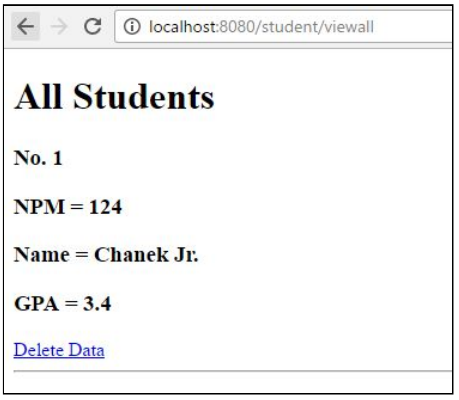
   Lengkapi script SQL untuk menghapus mahasiswa dengan NPM tertentu.
   Hint: Delete dalam SQL "DELETE FROM table_name [WHERE Clause]"
4. Lengkapi method **deleteStudent** yang ada di class **StudentServiceDatabase**
   a. Tambahkan log untuk method tersebut dengan cara menambahkan

```
log.info ("student " + npm + " deleted");
```

   b. Panggil method delete student yang ada di Student Mapper
5. Lengkapi method **delete** pada class **StudentController**
   a. Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found
   b. Jika berhasil delete student dan tampilkan view delete
      - **Hint**: lakukan select student terlebih dahulu dengan NPM, kurang lebih mirip dengan method view
6. Jalankan Spring Boot app dan lakukan beberapa insert
7. Contoh tampilan View All



8. Contoh tampilan delete NPM 123



9. Contoh tampilan jika dilakukan delete NPM 123 yang kedua kalinya

Latihan Menambahkan Update

Selama ini Anda masih menggunakan method GET untuk melakukan masukan. Pada update kali ini, Anda akan mencoba menggunakan method POST.

1. Tambahkan method **updateStudent** pada class **StudentMapper**
    a. Parameternya adalah StudentModel student
    b. Annotationnya adalah @Update
    c. Lengkapi SQL *update*-nya
    Hint: Query SQL untuk update

```
UPDATE table_name
SET column1=value1, column2=value2
WHERE id=some_id;
```

2. Tambahkan method **updateStudent** pada interface **StudentService**
3. Tambahkan implementasi method **updateStudent** pada class **StudentServiceDatabase.** Jangan lupa tambahkan logging pada method ini.
4. Tambahkan link Update Data pada **viewall.html**
5. Copy view form-add.html menjadi **form-update.html**.
    a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.
    b. Ubah action form menjadi /student/update/submit
    c. Ubah method menjadi post
    d. Untuk input npm ubah menjadi

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

    readonly agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada.
    Input name ubah menjadi

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

    Lakukan hal yang sama untuk GPA.

6. Copy view success-add.html menjadi **success-update.html**.
    a. Ubah keterangan seperlunya
7. Tambahkan method **update** pada class **StudentController**
    a. Request mapping ke /student/update/{npm}
    b. Sama halnya seperti delete, lakukan validasi.
    c. Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update
8. Tambahkan method **updateSubmit** pada class **StudentController**
    a. Karena menggunakan post method maka request mappingnya adalah sebagai berikut:

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

b. Header methodnya adalah sebagai berikut:

```
public String updateSubmit (
@RequestParam(value = "npm", required = false) String npm,
@RequestParam(value = "name", required = false) String name,
@RequestParam(value = "gpa", required = false) double gpa)
```

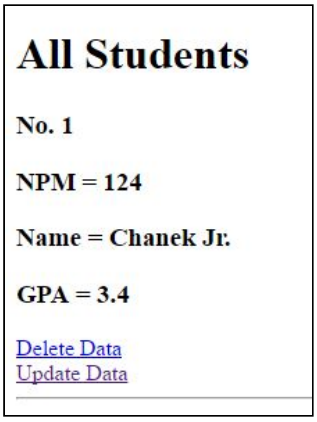
c. Lengkapi method dengan memanggil method update Student dan kembalikan view success-update

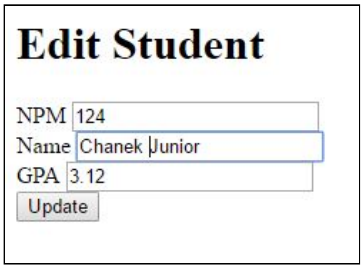
9. Jalankan Spring Boot dan coba test program Anda.

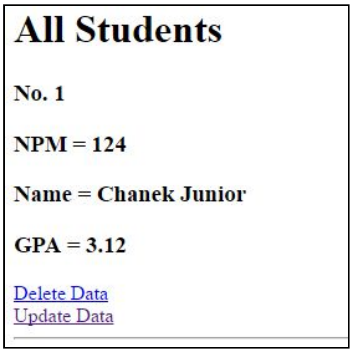
Latihan Menggunakan Object Sebagai Parameter

a. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.
b. SpringBoot dan Thymeleaf memungkinkan agar method **updateSubmit** menerima parameter berupa model **StudentModel**. **Metode ini lebih disarankan dibandingkan menggunakan RequestParam.**
c. Cara lengkapnya silakan ikuti pada link Handling Form berikut: (https://spring.io/guides/gs/handling-form-submission/)
d. Tahapannya kurang lebih sebagai berikut:
   - Menambahkan th:object="${student}" pada tag **<form>** di view
   - Menambahkan th:field="*{[nama_field]}" pada setiap input
   - Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel
   - Tes lagi aplikasi Anda.

Contoh Tampilan







Pertanyaan

Beberapa pertanyaan yang perlu Anda jawab:

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.

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?

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

Silakan tulis jawaban Anda pada write-up. Anda bisa menambahkan kode dan screenshot yang mendukung jawaban Anda.

Deliverables

Deliverables untuk tutorial kali ini adalah:

1. **File Project**
   Buat sebuah project baru pada organization /apap-2017 dengan format nama tutorial4_NPM, contoh tutorial4_1501234567. Push project Anda ke repository GitHub tersebut.

2. **Write-up**
   Buat sebuah file write-up. Jelaskan apa saja hal yang Anda pelajari dari tutorial ini. Jawab pertanyaan pada bagian Pertanyaan dan penjelasannya. Cantumkan juga penjelasan Anda terhadap hal-hal barikut:
   - Method yang Anda buat pada Latihan Menambahkan Delete, jelaskan
   - Method yang Anda buat pada Latihan Menambahkan Update, jelaskan
   - Method yang Anda buat pada Latihan Menggunakan Object Sebagai Parameter, jelaskan

   Kami rekomendasikan Anda menggunakan format pdf agar dapat lebih leluasa dalam menuangkan penjelasan Anda dengan dukungan screenshot dan kode. Masukkan file writeup ke folder project. Pastikan file write-up juga di-push ke repository.

Deadline

30 September 2017, 23:59:59

Penalti

Penalti:

- Keterlambatan
  Penalti keterlambatan sebesar -10 poin akan ditambahkan setiap 10 menit keterlambatan