Fajar Bardan Salam / 1506689401
APAP-A

Tutorial 5 : Menggunakan Database serta Relasi Database dalam Project Spring Boot

MENAMBAHKAN MODEL

1. Membuka kode tutorial 04 yang sudah dikerjakan minggu lalu.
2. Karena kita menambahkan Model baru maka kita perlu menambahkan class **CourseModel** pada package com.example.model

```java
package com.example.model;

import java.util.List;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CourseModel {
    private String idCourse;
    private String name;
    private Integer credits;
    private List<StudentModel> students;
}
```

3. Karena Student memiliki daftar Course yang dia ambil, maka pada class **StudentModel** perlu ditambahkan list of CourseModel pada variabel:

```java
    private String npm;
    private String name;
    private double gpa;
    private List<CourseModel> course;
```

Kemudian, pada **StudentController** akan muncul error karena menambahkan parameter constructor baru. Ubah inisialisasi constructor menjadi:

```java
StudentModel student = new StudentModel (npm, name, gpa, null);
```

4. Selanjutnya kita akan membuat method pada **StudentMapper** yang mengembalikan list of course pada Student dengan NPM tertentu. Methodnya adalah sebagai berikut:

```java
@Select("select course.id_course, name, credits " +
        "from studentcourse join course " +
        "on studentcourse.id_course = course.id_course " +
        "where studentcourse.npm = #{npm}"
        )
List<CourseModel> selectCourses (@Param("npm") String npm);
```

Query tersebut akan melakukan join antara tabel studentcourse dengan course berdasarkan id_course dan difilter hanya pada student dengan NPM yang diberikan

5. Selanjutnya kita akan menghubungkan method selectStudent dengan method selectCourses agar saat melakukan select maka variabel List<Course> juga akan terisi. Pada class **StudentMapper** terdapat method

```
@Select("select npm, name, gpa from student where npm = #{npm}")
StudentModel selectStudent (@Param("npm") String npm);
```

Method tersebut diubah menjadi

```
@Select("select npm, name, gpa from student where npm = #{npm}")
@Results(value = {
        @Result(property="npm", column="npm"),
        @Result(property="name", column="name"),
        @Result(property="gpa", column="gpa"),
        @Result(property="courses", column="npm",
                javaType= List.class,
                many=@Many(select="selectCourses"))
}

)

StudentModel selectStudent (@Param("npm") String npm);
```

Anotasi Results digunakan untuk memetakan hasil dan query select ke class model

```
@Result(property="npm", column="npm"),
```

Variabel **property** diisi dengan nama variabel di class StudentModel, sedangkan variabel **column** diisi dengan nama kolom hasil kueri di database (parameter yang dibutuhkan). Sedangkan untuk variabel courses karena hasilnya diambil dari method lain maka querynya adalah sebagai berikut:

```
@Result(property="courses", column="npm",
        javaType= List.class,
        many=@Many(select="selectCourses"))
```

Property diset dengan variabel courses karena kita ingin mengisi variabel courses di class StudentModel. Variabel column diisi dengan npm karena kolom npm pada database akan dikirim menjadi parameter di method selectCourses. Variabel javaType menentukan class yang menjadi kembalian. Kemudian variabel many diset dengan method yang akan mengisi variabel courses yaitu selectCourses.

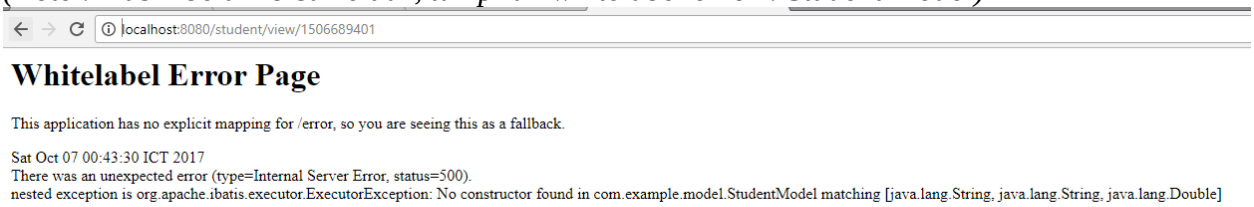
6. Selanjutnya kita akan menambahkan *view* pada **view.html** untuk menampilkan list kuliah yang diambil oleh mahasiswa seperti berikut:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View Student by NPM</title>
    </head>
    <body>
        <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
        <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
        <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>

        <h3>Kuliah yang diambil</h3>
        <ul th:each="course,iterationStatus: ${student.courses}">
            <li th:text="${course.name} + ' - ' + ${course.credits} + ' sks'">
                Nama kuliah-X SKS
            </li>
        </ul>
    </body>
</html>
```

7. Jalankan program tersebut dan buka http://localhost:8080/student/view/123 dan http://localhost:8080/student/view/124 (nomor student disesuaikan dengan database masing-masing).

8. Tampilannya kurang lebih sebagai berikut:
(Note : masih belum bisa keluar, tampilan whitelabel error : Student Model)



Latihan

1. Ubah method **selectAllStudents** pada kelas **StudentMapper** agar halaman *viewall* menampilkan semua *student* beserta daftar kuliah yang diambil.
Contoh tampilan:

```
//Method selectAllStudents yang telah diimplementasi
@Select("select npm, name, gpa from student")
@Results(value = {
        @Result(property="npm", column="npm"),
        @Result(property="name", column="name"),
        @Result(property="gpa", column="gpa"),
        @Result(property="courses", column="npm",
                javaType = List.class,
                many=@Many(select="selectCOurses"))
})
List<StudentModel> selectAllStudents ();
```

(*Tampilan belum bisa krn student model masih error*)

2. Buatlah view pada halaman **http://localhost:8080/course/view/{id}** untuk Course sehingga dapat menampilkan data course beserta Student yang mengambil.
(*Tampilan belum bisa krn student model masih error*)

Fajar Bardan Salam / 1506689401
APAP-A

Write-up

Method yang diubah pada Latihan Merubah **SelectAllStudent** :

- Menambahkan baris berikut pada viewall.html untuk menampilkan tampilan mata kuliah yang diambil oleh mahasiswa

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

        <div th:each="student,iterationStatus: ${students}">
            <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
            <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
            <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
            <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
            <h3>Kuliah yang diambil</h3>
                <ul th:each="course,iterationStatus: ${student.courses}">
                    <li th:text="${course.name} + '-' + ${course.credits} + ' sks'">
                        Nama kuliah-X SKS
                    </li>
                </ul>
            <a th:href="'/student/delete/' + ${student.npm}"> Delete Data</a><br/>
            <a th:href="'/student/update/' + ${student.npm}"> Update Data</a><br/>>
            <hr/>
        </div>
    </body>
</html>
```

- Mengubah method selectAllStudent pada class StudentMapper menjadi berikut

```
//Method selectAllStudents yang telah diimplementasi
@Select("select npm, name, gpa from student")
@Results(value = {
        @Result(property="npm", column="npm"),
        @Result(property="name", column="name"),
        @Result(property="gpa", column="gpa"),
        @Result(property="courses", column="npm",
                javaType = List.class,
                many=@Many(select="selectCOurses"))
})
List<StudentModel> selectAllStudents ();
```

Fajar Bardan Salam / 1506689401
APAP-A

Method yang diubah pada Latihan Menambahkan **View** pada **Course**:

- Saya menambahkan beberapa class/interface untuk Course sebagaimana yang telah dibuat pada Student, namun class/interface ini disesuaikan dengan fungsi yang ingin ditampilkan melalui Course, yaitu :
    - Course Controller

```java
package com.example.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import com.example.model.CourseModel;
import com.example.service.CourseService;

@Controller
public class CourseController {
    @Autowired
    CourseService courseDAO;

    @RequestMapping("/course/view/{id_course}")
    public String viewCourse (Model model,
            @PathVariable(value = "id_course") String id_course)
    {
        CourseModel course = courseDAO.selectCourse (id_course);

        if (course != null) {
            model.addAttribute ("course", course);
            return "view-course";
        } else {
            model.addAttribute ("id_course", id_course);
            return "not-found";
        }
    }
}
```

- Course Mapper

```java
package com.example.dao;

import java.util.List;

import org.apache.ibatis.annotations.Many;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Result;
import org.apache.ibatis.annotations.Results;
import org.apache.ibatis.annotations.Select;

import com.example.model.CourseModel;
import com.example.model.StudentModel;

@Mapper
public interface CourseMapper {
    @Select("select student.npm, name, gpa " +
            "from studentcourse join student " +
            "on studentcourse.npm = student.npm "
            + "where studentcourse.id_course = #{id_course}")
     List<StudentModel> selectCourseStudent (@Param("id_course") String id_course);

    @Select("select id_course, name, credits " +
            "from course " +
             "where id_course = #{id_course}")
    @Results(value = {
            @Result(property="idCourse", column="id_course"),
            @Result(property="name", column="name"),
            @Result(property="credits", column="credits"),
            @Result(property="students", column="id_course",
                    javaType = List.class,
                    many=@Many(select="selectCourseStudent"))
        }) CourseModel selectCourse (@Param("id_course") String id_course);
}
```

- Course Model

```java
package com.example.model;

import java.util.List;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CourseModel
{
    private String idCourse;
    private String name;
    private Integer credits;
    private List<StudentModel> students;
}
```

- Interface Course Service

```java
package com.example.service;

import com.example.model.CourseModel;

public interface CourseService {
    CourseModel selectCourse(String id_course);
}
```

- Course Service Database

```java
package com.example.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.dao.CourseMapper;
import com.example.model.CourseModel;

import groovy.util.logging.Slf4j;

@Slf4j
@Service
public class CourseServiceDatabase implements CourseService {
    @Autowired
    private CourseMapper courseMapper;

    @Override
    public CourseModel selectCourse(String id_course) {
        //log.info ("select course with id course {}", id_course);
        return courseMapper.selectCourse(id_course);
    }
}
```

- viewcourse.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View Course</title>
    </head>
    <body>
        <h3 th:text="'ID = ' + ${course.idCourse}"> ID </h3>
        <h3 th:text="'Nama = ' + ${course.name}"> Name </h3>
        <h3 th:text="'SKS = ' + ${course.credits}"> SKS </h3>

        <h3>Mahasiswa yang mengambil</h3>
        <ul th:each="student,iterationStatus: ${course.students}">
            <li th:text="${student.npm} + '-' + ${student.name}">
                NPM Mahasiswa-X
            </li>
        </ul>
    </body>
</html>
```

Write up : Pada tutorial 5 ini saya mempelajari bagaimana menggunakan database serta relasi database dalam project Spring Boot. Dimana pada tutorial kali ini terdapat improvisasi dari tutorial 4 sebelumnya, tambahan berupa hubungan antara **Student** dengan **Course.** Karena adanya hubungan dari kedua hal tersebut maka perlu diimplementasikan tabel baru serta halaman view pada Course. Dimana halaman view pada Course tersebut terdiri dari : *CourseController*, *CourseMapper*, *CourseModel*, *CourseService* (Interface), *CourseServiceDatabase*, dan *viewcourse.html*. Semua class/interface/html ini memiliki fungsi yang sama dengan yang ada di Student, namun yang dibuat ini adalah versi ***'course'***-nya.