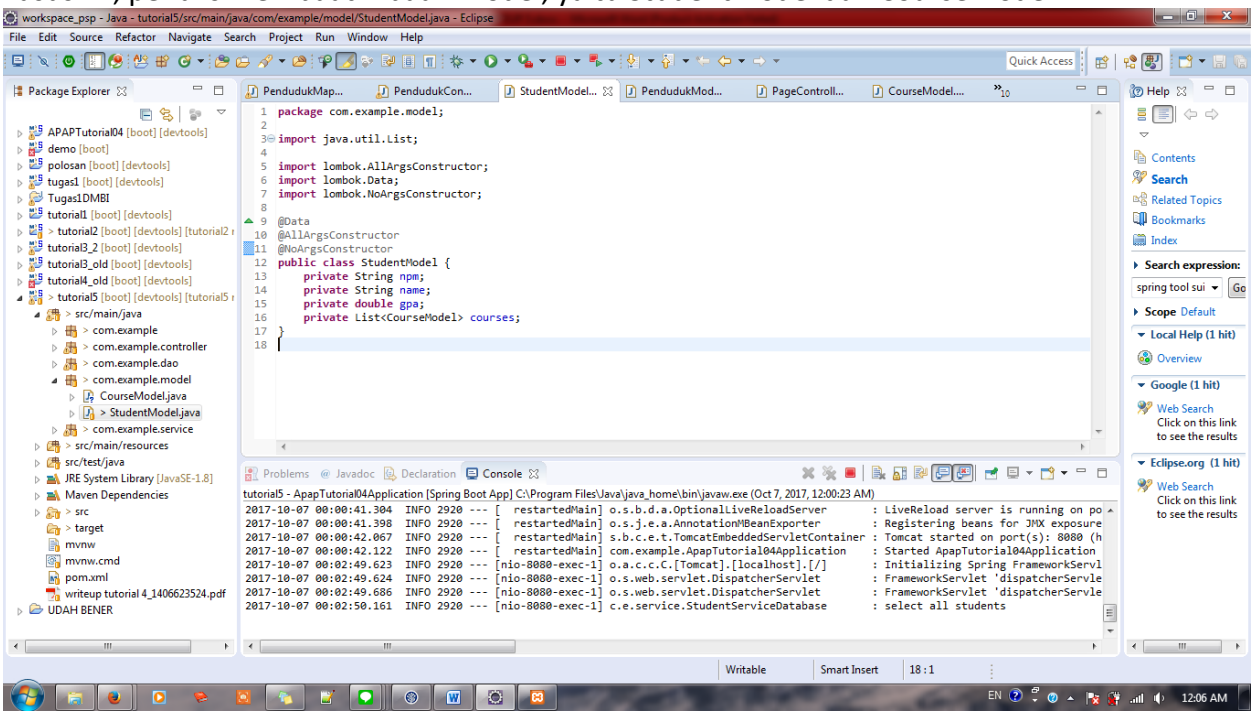
Iman Alfathan Yudhanto
1406623524
APAP-A

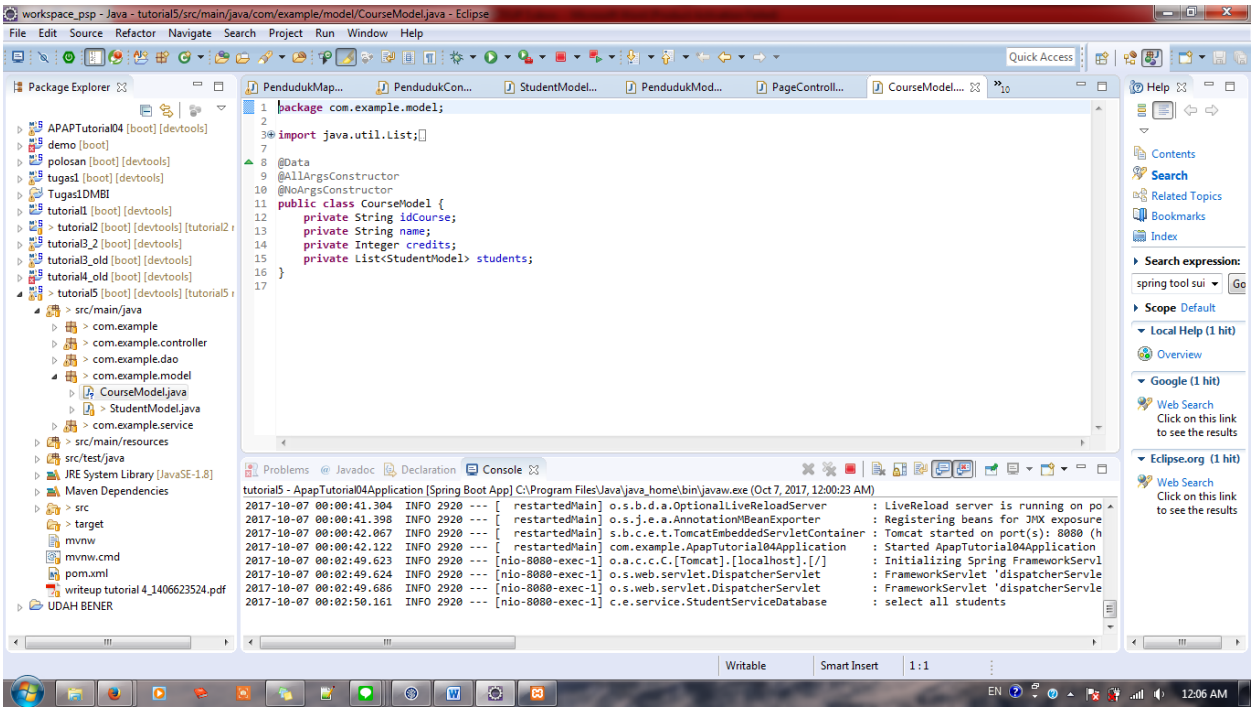
Write up-Tutorial 5

1. Method yang Anda ubah pada Latihan Merubah SelectAllStudents, jelaskan.

1. Hal pertama yang dilakukan adalah membuat model dari database yang terkait. Pada kasus ini, penulis membuat 2 buah model, yaitu StudentModel dan CourseModel.



Tampilan kodingan dari StudentModel.



Tampilan kodingan dari CourseModel

Pada kedua model ini ditambahkan Arraylist of Object karena sifat dari databasenya.

2. Memodifikasi StudentMapper (terutama pada bagian sql) untuk melakukan pengambilan Course dari database.

```
@Select("select course.id_course,"
        + "course.name, "
        + "course.credits "
        + "from studentcourse "
        + "join course on studentcourse.id_course= course.id_course "
        + "join student on studentcourse.npm= student.npm")
List<CourseModel> selectCourses2(@Param("npm") String npm);

@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> selectAllStudents2();
```

Tampilan dari bagian mapper yang dimodifikasi.

Penulis memodifikasi bagian mapper tersebut agar bisa mengambil course model. Cara pada method selectCourses2 pada *screenshot* di atas, carnya mirip seperti yang dituliskan pada tutorial ketika menampilkan view secara detail beserta course yang diambil.

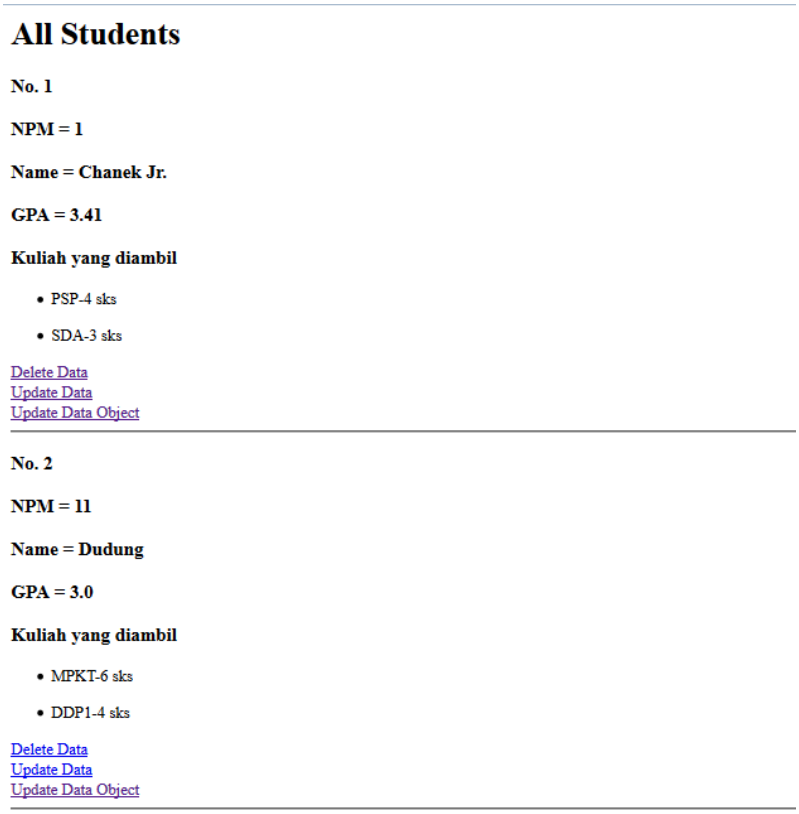
3. Memodifikasi front-end pada bagian student agar bisa menampilkan course yang diambil oleh student.

```
<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 />
        <a th:href="'/student/updateObject/' + ${student.npm}"> Update Data Object </a><br />
        <hr />
    </div>
</body>
</html>
```

Tampilan pada front-end

Tampilan tersebut berisi fungsi thymeleaf yang berfungsi untuk menghubungkan backend dengan front end. Pada bagian bawah iteration student, terdapat iteration untuk mengambil course.

4. Program dapat ditampilkan



Berikut adalah tampilan jika program berhasil dilakukan.

2.Buatlah view pada halaman http://localhost:8080/course/view/{id} untuk Course sehingga dapat menampilkan data course beserta Student yang mengambil.

Hal yang pertama dilakukan adalah membuat service, mapper, service interface, dan front-end untuk course.

```
1  package com.example.service;
2
3  import com.example.model.CourseModel;
5
6  public interface CourseService {
7      CourseModel selectCourse(String idCourse);
8
9  }
```

Tampilan dari Course Service

```java
package com.example.service;

import java.util.List;

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

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

import lombok.extern.slf4j.Slf4j;

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

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

Tampilan dari CourseServiceDatabase.

```java
package com.example.dao;


import java.util.List;

@Mapper
public interface CourseMapper {

    @Select("select student.npm,"
            + "name "
            + "from studentcourse "
            + "join student on studentcourse.npm= student.npm "
            + "where studentcourse.id_course = #{id_course}")
    List<StudentModel> selectStudents(@Param("id_course") String idCourse);

    @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="selectStudents"))
    })
    CourseModel selectCourse(@Param("id_course") String idCourse);

}
```

Tampilan dari CourseMapper.

Pada bagian mapper, cara pengambilan database hamper mirip pada tutorial cara mengambil course saat menampilkan student dengan npm tertentu. Yang penulis lakukan hanyalah membalik cara tersebut.

Hal yang dipelajari:

Pada tutorial ini, penulis belajar cara menggunakan, mengambil, dan mengkoneksi database dengan aplikasi springboot. Selain memberikan gambaran umum mengenai penggunaan database, tutorial ini juga mengajarkan penulis menggunakan anotasi query-query untuk melakukan pengambilan database seperti @delete (untuk membantu query delete melakukan penghapusan), @select(untuk membantu query select dan membantu dalam menyeleksi data yang diinginkan), @result (memberikan hasil dari database yang diambil dan disesuaikan dengan model), dan @update(membantu query update untuk melakuan update data). Yang menarik bagi penulis adalah penggunaan @Many yang berfungsi untuk menghubungkan 2 query dari mapper yang sama.