Nama : Awalia Mitha
NPM : 1506689521

Tutorial 5
APAP B

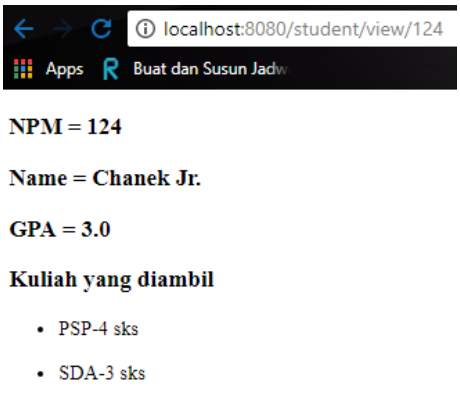
Menggunakan *Database* serta Relasi *Database* dalam *Project Spring Boot*

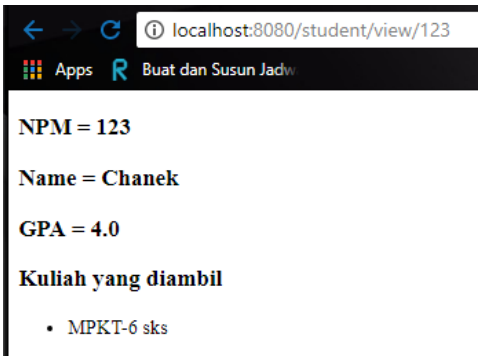
1. *Write-up*

Pada tutorial 5 kali ini, saya mempelajari fungsi annotasi *@Result @Many.* Dengan bertambahkan *entity* pada basis data, ada kondisi untuk menggunakan annotasi tersebut. Setiap ingin menambahkan model baru, berarti harus membuat *class* model juga di *package* model. Lalu *interface* StudentMapper harus disesuaikan dengan kondisi sekarang. Karena pada tutorial kali ini butuh menampilkan mata kuliah yang diambil mahasiswa dengan npm tertentu, maka perlu diubah *query* pada *StudentMapper.* Menggunakan *join* antara tabel *studentcourse* dengan *course* di basis data. Kemudian, perlu ditambahkan *array list of courses* pada objek *student* dan *array list of student* pada objek *course.* Bagaimana cara mengisi *list of courses* pada objek *student?* dengan *method selectCourses.* Nah karna mengambil data dari *method lain* digunakan annotasi *@Many* –juga annotasi *@Result* untuk memetakan data dari *database* ke atribut pada model *student.* Tapi sebenarnya saya masih ragu terkait materi ini, kapan dibutuhkannya kedua annotasi tersebut, dalam kondisi seperti apa dan solusi yang bagaimana.

2. *Method SelectAllStudents*

**Sebelumnya, pada tahap tutorial dilakukan tes *view student* dan mata kuliah yang dia ambil berdasarkan npm tertentu. Berikut *screenshoot* nya.



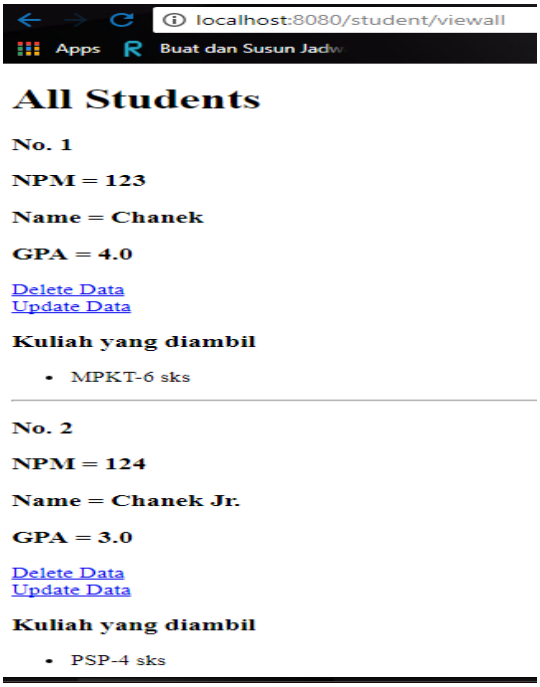


- *Method SelectAllStudents*

Method diubah pada *interface StudentMapper.*

```
@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 ();
```

Prosesnya sama seperti *method selectStudent.* Hanya saja *method selectAllStudents* hasilnya akan disimpan pada *array list StudentModel.* Untuk *select* semua *student,* langsung *select* semua kolom dari tabel *student.* Nah, di objek *Student* sudah ada atribut npm, *name, gpa* dan *list of courses.* Bagaimana cara mengisi semua atribut tersebut? Gunakan annotasi *@Result,* berfungsi untuk memetakan yang di *databse* ke atribut yang ada di model *student. Column* diisi dengan nama kolom di *database,* hasilnya dipetakan ke *property* (sesuai nama di objek *student*) yang ada di *student* model. Lalu semua data disimpan pada *array list StudentModel.* Untuk bagian *column = npm* dan *property=courses.* Gunakan annotasi *@Many,* karena kita menggunakan *method* lain (*selectCourses*) di dalam *method selectAllStudent* ini. Npm yang didapatkan dari *method selectCourses* tersebut dipetakan ke atribut *courses (*ini *arrayList of courses).* Berikut *screenshoot* nya.



3. ***Method* yang Anda buat pada Latihan Menambahkan *View* pada *Course***

   Latihan menambahkan *view* pada *course* pertama buat *method* kosong pada *interface StudentService* dengan tipe *CourseModel* karna akan dikembalikan *course* dengan *id course* tertentu beserta mahasiswa yang mengambil *course* tersebut. Seperti ini *method* nya :

```
CourseModel viewCourse (String idCourse);
```

   Kedua, *override method* tersebut di *class StudentServiceDatabase.* Seperti ini:

```
@Override
public CourseModel viewCourse (String id_course)
{
    log.info("course with " + id_course);
    return studentMapper.selectCourse(id_course);
}
```

   return method selectCourse dengan parameter *id_course* karena kita akan

mengambil semua data *course* berdasarkan *id_course* tertentu. Bagaimana cara mengambilnya? sudah dilakukan oleh *method selectCourse* di *interface StudentMapper.*

Ketiga, karena kita pakai *method selectCourse* dan *method* tersebut butuh info *student* yang mengambil mata kuliah apa saja. Maka dibuat *method* baru yaitu *SelectCourseStudent,* ini untuk *join* tabel *student* dengan tabel *studentcourse.* Berati didapatkan data *student (npm* dan *name)* yang ambil mata kuliah apa aja.

```
@Select("SELECT * 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);

@Select("SELECT * from student join studentcourse on student.npm = studentcourse.npm where studentcourse.id_course = #{id_course}")
List<StudentModel> selectCourseStudent (@Param("id_course") String id_course);
```

Jadi, *method selectCourse* untuk *select* semua nilai di tabel *course* dengan *id_course* tertentu. Kemudian dipetakan hasilnya ke atribut yang ada di objek *Course*. Karena di objek *Course* ada atribut *list of student*. Gimana cara isinya? Perlu dibuat 1 *method* lagi yang mengembalikan *list of student* (*student* tersebut ambil mata kuliah apa saja). Ketika didapatkan *id_course* yang si mahasiswa ambil, dipetakanlah hasilnya ke atribut *students* pada objek *Course*.

Keempat, pada *class StudentController.java* implementasikan *method* viewCourse yang sudah dibuat sebelumnya pada *interface StudentService.* Berikut kode nya.

```
//method untuk menampilkan course dengan id course tertentu dan mahasiswa yang mengambilnya
@RequestMapping("/course/view/{id}")
public String viewCourse (Model model, @PathVariable(value = "id") String idCourse)
{
    CourseModel courses = studentDAO.viewCourse(idCourse);

    if (courses != null) {
        model.addAttribute ("course", courses);
        return "viewIdCourse";
    }else {
        model.addAttribute ("id_course", idCourse);
        return "not-foundCourse";
    }
}
```

karena kita butuh objek *course* maka dideklarasikan, lalu dicek jika *courses* dengan *id_course* yang diminta ada, maka panggil html *viewIdCourse.* Jika tidak ada, maka panggil html *not-foundCourse.* Berikut kode *viewIdCourse.html*

viewIdCourse.html | not-foundCourse.html

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

        <form th:objects="${course}">
        <h3 th:text="'ID = ' + ${course.idCourse}" th:field="*{id}">ID COURSE</h3>
        <h3 th:text="'Nama = ' + ${course.name}" th:field="*{name}">COURSE NAME</h3>
        <h3 th:text="'SKS = ' + ${course.credits}" th:field="*{credit}">CREDITS</h3>

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

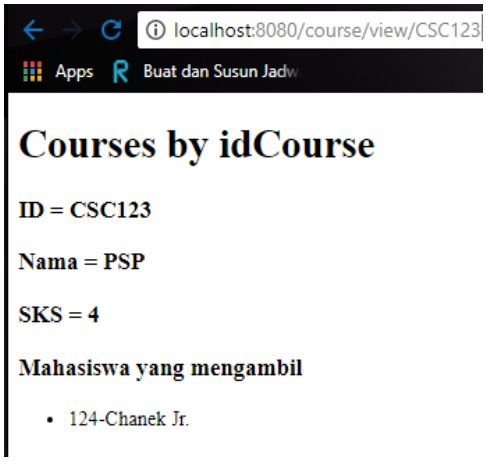
Berikut kode *not-foundCourse.html*

viewIdCourse.html | not-foundCourse.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Student not found</title>
    </head>
    <body>
        <h1>ID Course Not Found</h1>
        <h3 th:text="'ID = ' + ${id_course}"></h3>
    </body>
</html>

```

Berikut *screenshoot* jika *id_course* ditemukan



Berikut *screenshoot* jika *id_course* tidak ditemukan