A. Ringkasan dari yang Saya pelajari pada tutorial ini

Pada tutorial 7 ini saya belajar bagaimana caranya membuat *web service* menggunakan *spring boot framework*. Jadi, pada tutorial ini membuat *web service* dengan 2 tutorial sebelumnya yaitu tutorial 5 dan 6. Tutorial 5 sebagai *Service Producer (layer backend)* dan tutorial 6 sebagai *service consumer (frontend)*.

Pada tutorial ini, *service producer* tidak menghasilkan tampilan yang bagus untuk pengguna, namun menghasilkan data berupa *object-object* yang akan digunakan untuk *service consumer* nantinya. *Service consumer* tidak perlu mengakses *database*, karena ia bisa mengolah data dari yang dihasilkan oleh *service producer*.

Untuk berkomunikasi dengan *service consumer*, *service producer* menyediakan *web service* yang dapat dikonsumsi oleh *service consumer*. *Web service* merupakan sebuah URL yang akan mengembalikan data dalam representasi yang telah disepakati seperti JSON atau XML. Dalam tutorial ini data yang dikembalikan adalah dalam bentuk JSON yang selanjutnya digunakan oleh *service customer*.

Service producer akan mengakses data dari *database* dan menghasilkan data-data *student* maupun *course* dalam bentuk JSON. Selanjutnya dari data JSON tersebut akan diolah di *service consumer* untuk menghasilkan suatu tampilan yang dapat dimengerti oleh pengguna.

B. Penjelasan terhadap setiap latihan pada tutorial ini

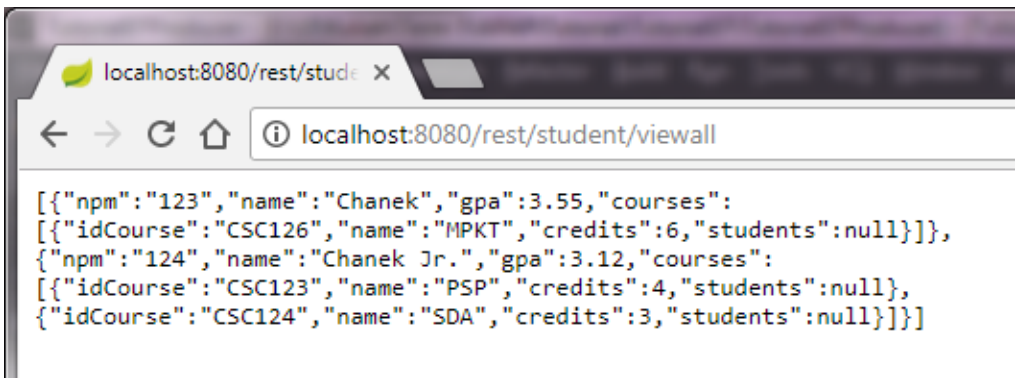
1. **Latihan 1** – *Student Viewall* pada *service producer*

   Untuk membuat *service* untuk mengembalikan seluruh *student* yang ada di basis data dan *service* tersebut di- *mapping* ke "/rest/student/viewall" adalah dengan cara menambahkan *method* pada class StudentRestController.java sebagai berikut :

```
@RequestMapping("/student/viewall")
public List<StudentModel> viewAll (Model model) {
    List<StudentModel> students = studentService.selectAllStudents();
    return students;
}
```

   Pada *method* tersebut ketika dilakukan akses ke halaman /rest/student/viewall, maka akan dijalankan *method selectAllStudents* yang akan mengambil seluruh data semua *student* di *database*, lalu mengembalikan JSON yang berisi seluruh *student* yang ada di *database*.

   Hasil ketika dijalankan sebagai berikut :



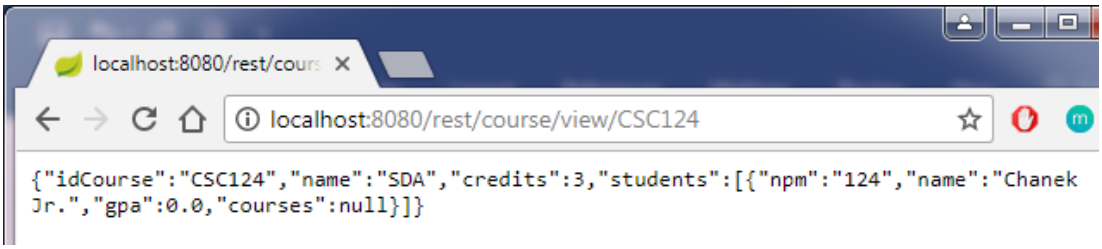
2. **Latihan 2** – *view by id* dan *viewall course* pada *service producer*

Pertama, yang dilakukan adalah membuat *class* pada *package rest* yaitu CourseRestController.java. Lalu di dalam CourseRestController.java berisi seperti berikut :

```
@RestController
@RequestMapping("/rest")
public class CourseRestController {

    @Autowired
    CourseService courseService;

    @RequestMapping("/course/view/{id}")
    public CourseModel view (@PathVariable(value = "id") String id) {
        CourseModel course = courseService.selectCourse (id);
        return course;
    }
```

Pada *method* tersebut menyatakan bahwa ketika dilakukan akses ke halaman /rest/course/view/{id}, maka dia akan menjalankan *method selectCourse* yang mencari *course* berdasarkan id yang diberikan, lalu akan mengembalikan JSON yang berisi data *course* yang sesuai id yang diberikan dari *database*.

Hasil ketika dijalankan sebagai berikut :



Kedua, untuk *viewall*, karena pada CourseService.java belum ada *method* untuk mengambil data semua *course* maka buat *method* dulu pada CourseService sebagai berikut :

```
List<CourseModel> selectAllCourses ();
```

Lalu, buat implementasinya pada CourseServiceDatabase sebagai berikut :

```
@Override
public List<CourseModel> selectAllCourses ()
{
    log.info ("select all students");
    return courseMapper.selectAllCourses ();
}
```

SelectAllCourse masih merah karena pada CourseMapper.java belum terdapat *method* tersebut, maka selanjutnya buat *method* selectAllCourse pada CourseMapper yang akan mengambil data semua *course*. Method pada CourseMapper sebagai berikut :

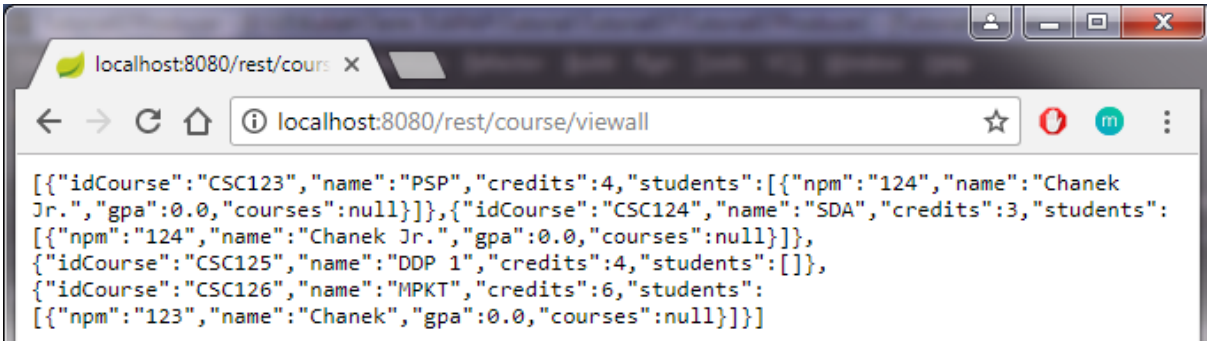
```
@Select("select id_course, name, credits from course")
@Results(value = {
        @Result(property="idCourse", column="id_course"),
        @Result(property="name", column="name"),
        @Result(property="credits", column="credits"),
        @Result(property="students", column="id_course",
                javaType = List.class,
                many=@Many(select="selectStudents"))
})
List<CourseModel> selectAllCourses ();
```

Selanjutnya pada pada CourseRestController.java ditambahkan *method* sebagai berikut :

```
@RequestMapping("/course/viewall")
public List<CourseModel> viewAll (Model model) {
    List<CourseModel> courses = courseService.selectAllCourses();
    return courses;
}
```

Pada *method* tersebut menyatakan bahwa ketika dilakukan akses ke halaman /rest/course/viewall, maka dia akan menjalankan *method selectAllCourse* yang mengambil data semua *course*, lalu akan mengembalikan JSON yang berisi data semua *course* yang ada.

Hasil ketika dijalankan sebagai berikut :



3. **Latihan 3** – Implementasi *service customer* untuk *viewall students*

Pada StudentServiceRest.java yang sebelumnya :

```
@Override
public List< StudentModel > selectAllStudents ()
{
    log.info ( "REST - select all students" );
    return null;
}
```

ubah menjadi :

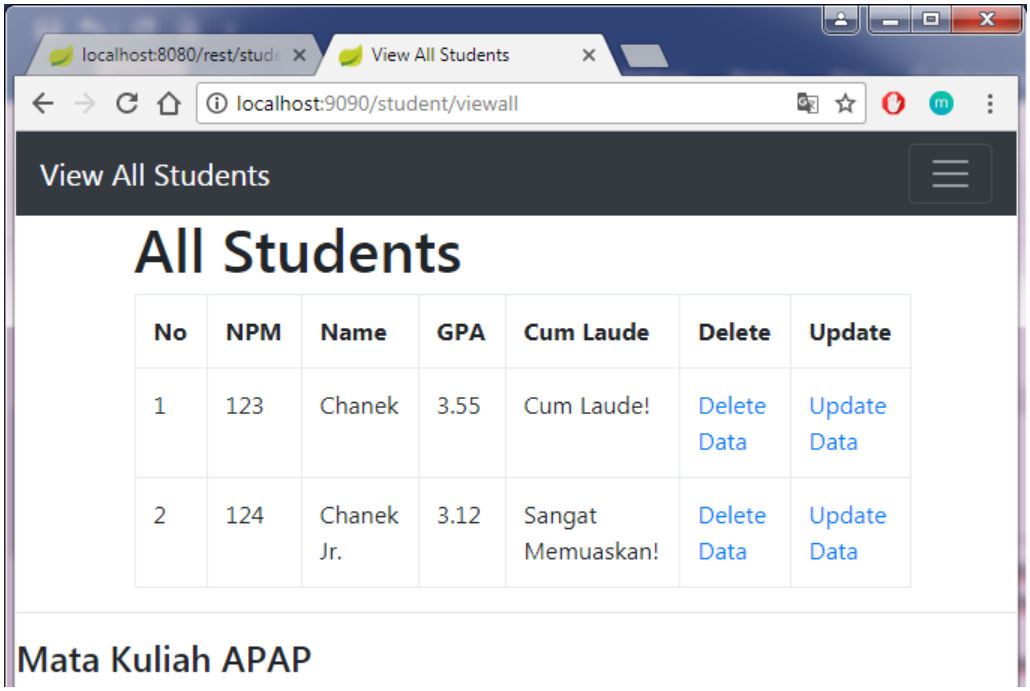
```
@Override
public List<StudentModel> selectAllStudents ()
{
    log.info ("REST - select all students");
    return studentDAO.selectAllStudents();
}
```

Lalu pada StudentDAOImpl.java implementasi *method* selectAllStudents yaitu sebagai berikut :

```
@Override
public List<StudentModel> selectAllStudents ()
{
    List<StudentModel> students =
            restTemplate.getForObject(
                    "http://localhost:8080/rest/student/viewall",
                    List.class);
    return students;
}
```

Pada *method* tersebut, dia menerima *object* yang dihasilkan dari *link* (yaitu berasal dari *service producer* berupa JSON yang berisi data semua *student*). Lalu akan mengembalikan *List* dari *students*.

Hasil ketika dijalankan sebagai berikut :



4. **Latihan 4** - Implementasi *service consumer* untuk *class* CourseModel dengan *membuat class-class* DAO dan *service* baru.

Pertama yaitu, *package* DAO membuat membuat *interface* CourseDAO, yang berisi :

```
package com.example.dao;

import ...

public interface CourseDAO {
    CourseModel selectCourse (String id);
    List<CourseModel> selectAllCourses ();
}
```

Lalu selanjutnya buat *class* CourseDAOImpl yang mengimplementasi CourseDAO, yang berisi sebagai berikut :

```
@Service
public class CourseDAOImpl implements CourseDAO {

    @Autowired
    private RestTemplate restTemplate;

    @Override
    public CourseModel selectCourse (String id)
    {
        CourseModel course =
                restTemplate.getForObject(
                        url: "http://localhost:8080/rest/course/view/"+id,
                        CourseModel.class);
        return course;
    }

    @Override
    public List<CourseModel> selectAllCourses ()
    {
        List<CourseModel> courses =
                restTemplate.getForObject(
                        url: "http://localhost:8080/rest/course/viewall",
                        List.class);
        return courses;
    }
```

Pada gambar tersebut, sudah ada 2 method yaitu selectCourse(String id) dan selectAllCourse. Pada *method* selectCourse(String id), dia menerima *object* yang dihasilkan dari *link* (yaitu berasal dari *service producer* berupa JSON yang berisi data *course* dengan id tertentu). Lalu akan mengembalikan *object course* sesuai dengan id. Pada *method* selectAllCourses, dia menerima *object* yang dihasilkan dari *link* (yaitu berasal dari *service producer* berupa JSON yang berisi data semua *course*). Lalu akan mengembalikan *List* dari *course*.

Lalu untuk CourseService dibuat seperti berikut :

```
List<CourseModel> selectAllCourses ();
```

Lalu membuat *class* CourseServiceRest.java yang mengimplementasi CourseService yang berisi :

```
public class CourseServiceRest implements CourseService {

    @Autowired
    private CourseDAO courseDAO;

    @Override
    public CourseModel selectCourse (String id)
    {
        log.info ("REST - select student with npm {}", id
        return courseDAO.selectCourse(id);
    }

    @Override
    public List<CourseModel> selectAllCourses ()
    {
        log.info ("select all students");
        return courseDAO.selectAllCourses();
    }
```

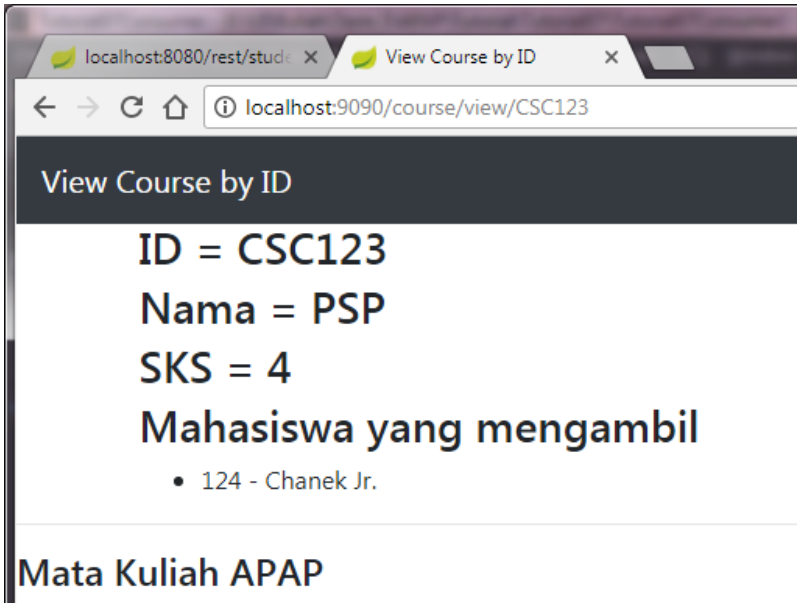
Untuk yang viewall, karena sebelumnya, yaitu pada lab 6, belum ada implementasi pada CourseController dan belum ada template view nya, maka dibuat dahulu pada controllernya sebagai berikut :

```
@RequestMapping("/course/viewall")
public String view (Model model)
{
    List<CourseModel> courses = courseDAO.selectAllCourses ();
    model.addAttribute ( s: "courses", courses);

    return "viewall-course";
}
```

Lalu pada templatenya :

```
<h1>All Courses</h1>

<div th:each="course,iterationStatus: ${courses}">
    <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
    <h3 th:text="'ID Course = ' + ${course.idCourse}">Course ID</h3>
    <h3 th:text="'Name = ' + ${course.name}">Course Name</h3>
    <h3 th:text="'Credits = ' + ${course.credits}">Course Credits</h3>
    <h3>Mahasiswa yang mengambil</h3>
    <ul th:each="student,iterationStatus: ${course.students}">
        <li th:text="${student.name} + '-' + ${student.npm}" >
            Nama Mahasiswa-NPM Mahasiswa
        </li>
    </ul>
```

Hasil ketika dijalankan untuk view course by id :



Hasil ketika dijalankan untuk viewall courses :