

# WRITE UP

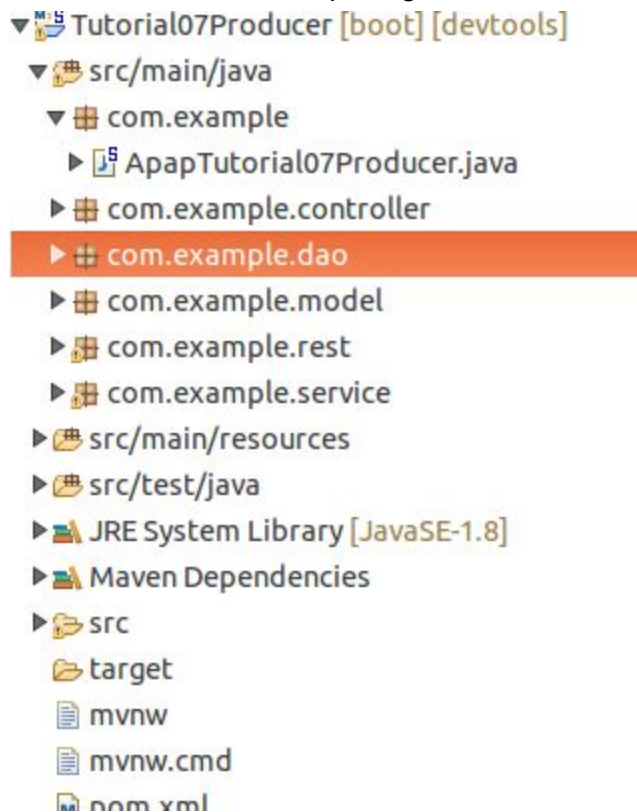
## Tutorial 7 - Membuat Web Service Menggunakan Spring Boot Framework

Pada tutorial kali ini, kita diajarkan cara membuat REST API beserta konsumernya menggunakan Spring MVC. Tutorial ini dibagi menjadi 2 tahapan, mulai dari membuat API (Producer) dan membuat Client (Consumer). Secara singkatnya, API yang melakukan interaksi dengan backend mulai dari database, struktur data, data-data internal, dan lain-lain. API atau Producer juga yang menerima request dari client terhadap suatu service tertentu. Client atau Consumer disini berperan sebagai jembatan utama antara sistem dengan pengguna. Consumer akan memanggil API yang tersedia oleh Producer, kemudian menerima response kembalian API tersebut dalam format JSON yang kemudian dapat diolah kembali menjadi response kembalian yang dapat ditampilkan kepada pengguna.

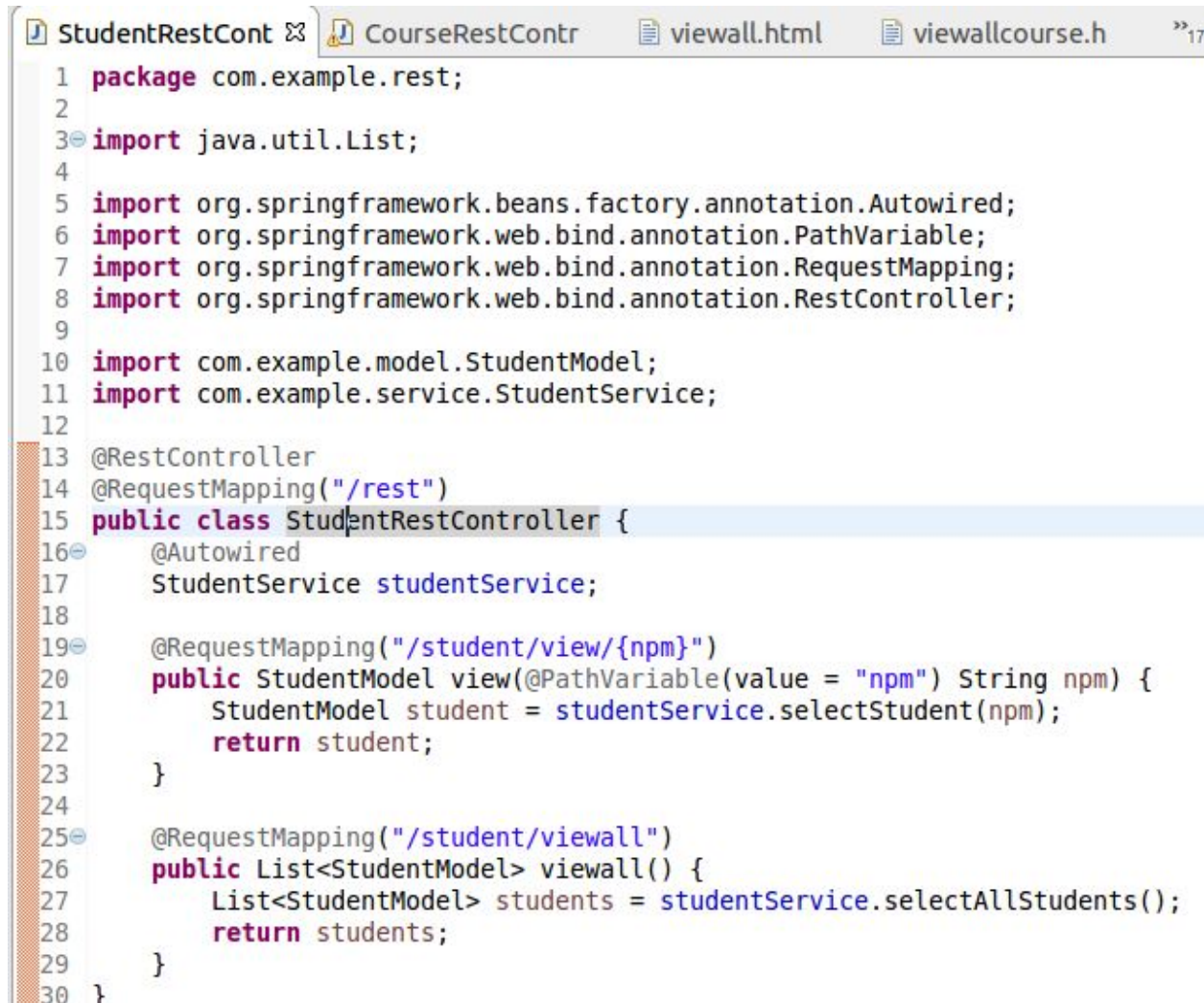
Tutorial ini memanfaatkan Tutorial5 sebelumnya untuk membuat producer, dan Tutorial6 sebelumnya untuk membuat consumer.

## Producer

Import berkas tutorial 5 untuk dapat digunakan kembali sebagai Producer di tutorial 7 kali ini.

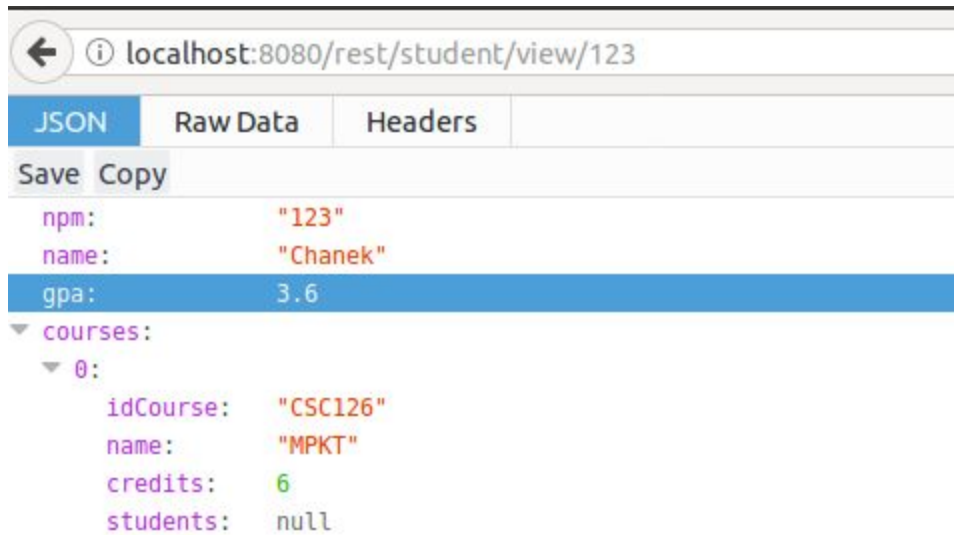


Buat package baru `com.example.rest`, atau sesuaikan dengan package Anda masing-masing.  
Buat class baru yaitu **`StudentRestController.java`** pada package **`com.example.rest`**

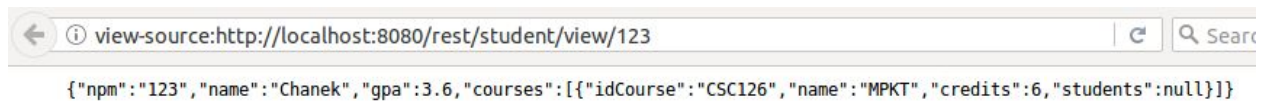
The image is a screenshot of an IDE window showing the code for a Java class named `StudentRestController`. The window has several tabs at the top: `StudentRestCont` (active), `CourseRestContr`, `viewall.html`, and `viewallcourse.h`. The code is as follows:

```
1 package com.example.rest;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.PathVariable;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import com.example.model.StudentModel;
11 import com.example.service.StudentService;
12
13 @RestController
14 @RequestMapping("/rest")
15 public class StudentRestController {
16     @Autowired
17     StudentService studentService;
18
19     @RequestMapping("/student/view/{npm}")
20     public StudentModel view(@PathVariable(value = "npm") String npm) {
21         StudentModel student = studentService.selectStudent(npm);
22         return student;
23     }
24
25     @RequestMapping("/student/viewall")
26     public List<StudentModel> viewall() {
27         List<StudentModel> students = studentService.selectAllStudents();
28         return students;
29     }
30 }
```

Untuk mencobanya langsung, silakan jalankan program Anda dan buka halaman **`localhost:8080/rest/student/view/123`** atau sesuai dengan NPM yang ada di data Anda.



Perhatikan bahwa ternyata browser yang saya gunakan dapat membaca format data JSON untuk ditampilkan dengan baik ke mata pengguna, seperti inilah tampilan raw nya.



## LATIHAN

### LATIHAN 1

**Buatlah service untuk mengembalikan seluruh student yang ada di basis data. Service ini mirip seperti method viewAll di Web Controller. Service tersebut di-mapping ke `"/rest/student/viewall"`.**

Untuk melakukannya, kita hanya perlu lengkapi implementasi pada StudentRestController

```
.2
.3 @RestController
.4 @RequestMapping("/rest")
.5 public class StudentRestController {
.6     @Autowired
.7     StudentService studentService;
.8
.9     @RequestMapping("/student/view/{npm}")
.10    public StudentModel view(@PathVariable(value = "npm") String npm) {
.11        StudentModel student = studentService.selectStudent(npm);
.12        return student;
.13    }
.14
.15    @RequestMapping("/student/viewall")
.16    public List<StudentModel> viewall() {
.17        List<StudentModel> students = studentService.selectAllStudents();
.18        return students;
.19    }
.20 }
```

Simpelnya, kita hanya perlu panggil studentService yang sudah dibuat pada tutorial 5 lalu, kemudian kita reuse method selectAllStudents() yang kemudian responsenya akan kita return sebagai list.

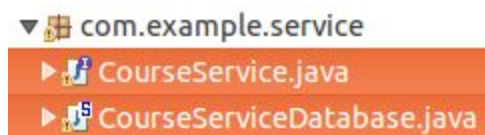
Begini lah hasil tampilan keluarannya.



## LATIHAN 2

**Buatlah service untuk class Course. Buatlah controller baru yang terdapat service untuk melihat suatu course dengan masukan ID Course (view by ID) dan service untuk melihat semua course (view all).**

Membuat 2 service baru yang sebelumnya belum ada di tutorial 5





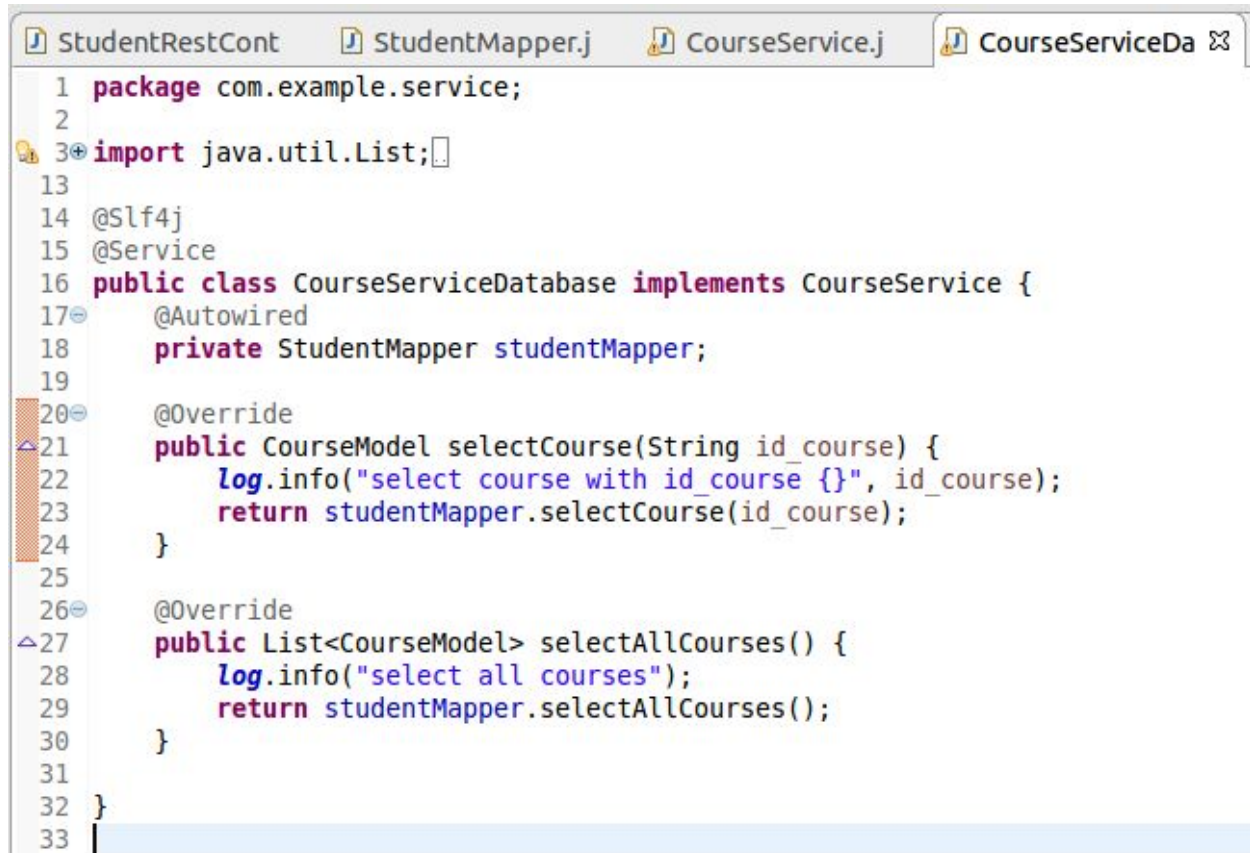
Membuat dua buah query ke database pada Mapper. Satu untuk select course by id, satu lagi untuk select semua course yang ada.

```
StudentRestCont StudentMapper.j CourseService.j CourseServiceDa »17
53 @Results(value = {
54     @Result(property = "idCourse", column = "id_course")
55 })
56 List<CourseModel> selectCourses(@Param("npm") String npm);
57
58 @Select("select id_course, name, credits from course where id_course = #{id_course}")
59 @Results(value = {
60     @Result(property = "idCourse", column = "id_course"),
61     @Result(property = "name", column = "name"),
62     @Result(property = "credits", column = "credits"),
63     @Result(property = "students", column = "id_course",
64         javaType = List.class,
65         many = @Many(select = "selectStudents"))
66 })
67 CourseModel selectCourse(@Param("id_course") String idCourse);
68
69 @Select("select id_course, name, credits from course")
70 @Results(value = {
71     @Result(property = "idCourse", column = "id_course"),
72     @Result(property = "name", column = "name"),
73     @Result(property = "credits", column = "credits"),
74     @Result(property = "students", column = "id_course",
75         javaType = List.class,
76         many = @Many(select = "selectStudents"))
77 })
78 List<CourseModel> selectAllCourses();
```

Mengisi interface CourseService

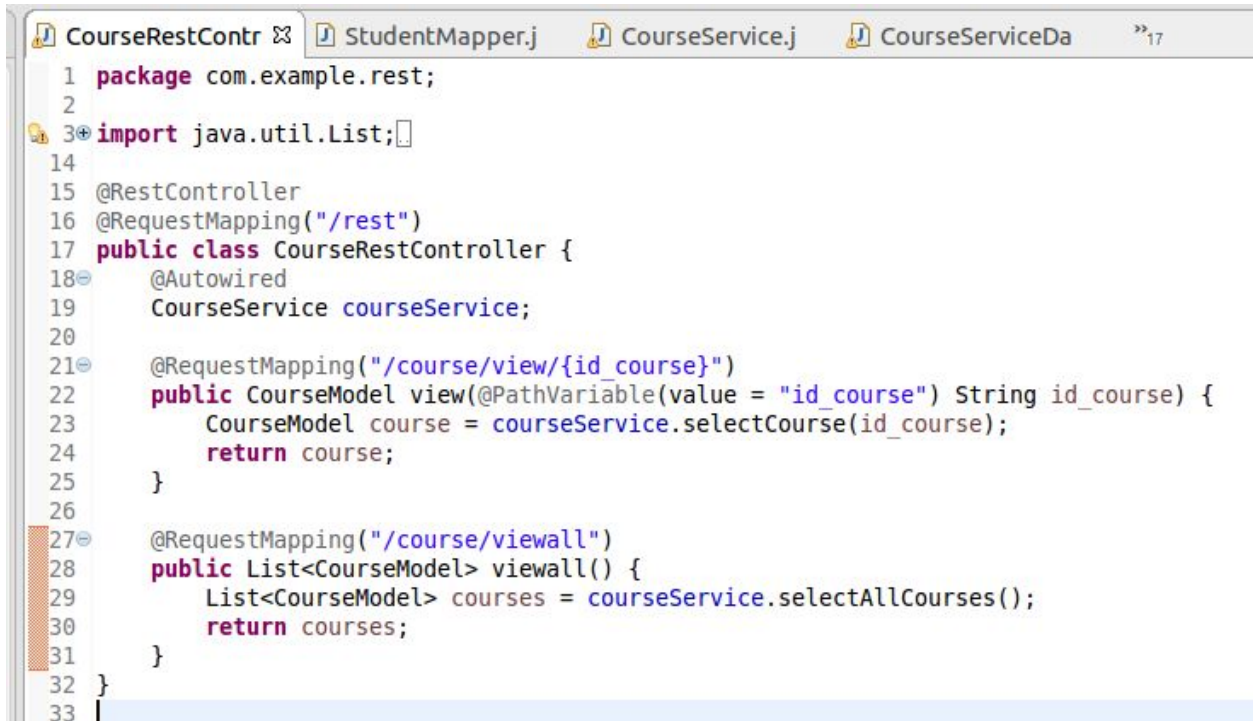
```
StudentRestCont StudentMapper.j CourseService.j
1 package com.example.service;
2
3 import java.util.List;
4
5
6
7
8 public interface CourseService
9 {
10     CourseModel selectCourse(String id_course);
11
12
13     List<CourseModel> selectAllCourses ();
14 }
15
```

Mengisi Implementasi dari CourseService ke CourseServiceDatabase



```
1 package com.example.service;
2
3 import java.util.List;
4
5 @Slf4j
6 @Service
7 public class CourseServiceDatabase implements CourseService {
8     @Autowired
9     private StudentMapper studentMapper;
10
11     @Override
12     public CourseModel selectCourse(String id_course) {
13         log.info("select course with id_course {}", id_course);
14         return studentMapper.selectCourse(id_course);
15     }
16
17     @Override
18     public List<CourseModel> selectAllCourses() {
19         log.info("select all courses");
20         return studentMapper.selectAllCourses();
21     }
22 }
23
24
25
26
27
28
29
30
31
32
33
```

Buat class CourseRestController sebagai controller yang akan menerima request lalu mengirim response ke client. Letakkan pada package com.example.rest. Panggil implementasi dari courseService untuk menampilkan data yang diinginkan.

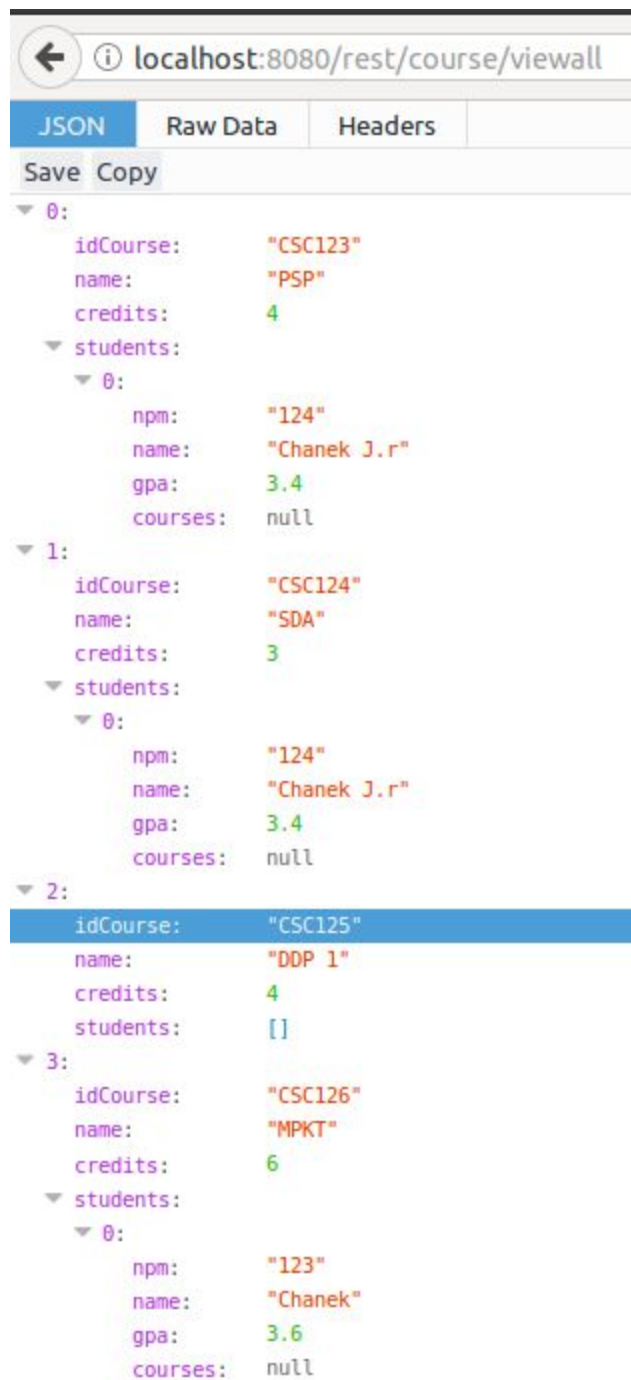


```
1 package com.example.rest;
2
3 import java.util.List;
4
14
15 @RestController
16 @RequestMapping("/rest")
17 public class CourseRestController {
18     @Autowired
19     CourseService courseService;
20
21     @RequestMapping("/course/view/{id_course}")
22     public CourseModel view(@PathVariable(value = "id_course") String id_course) {
23         CourseModel course = courseService.selectCourse(id_course);
24         return course;
25     }
26
27     @RequestMapping("/course/viewall")
28     public List<CourseModel> viewall() {
29         List<CourseModel> courses = courseService.selectAllCourses();
30         return courses;
31     }
32 }
33 |
```

Prinsip kerjanya mirip dengan yang student sebelumnya. Hasilnya akan seperti ini pula.

Untuk View All:





Untuk tampilan view.

← ⓘ localhost:8080/rest/course/view/CSC123

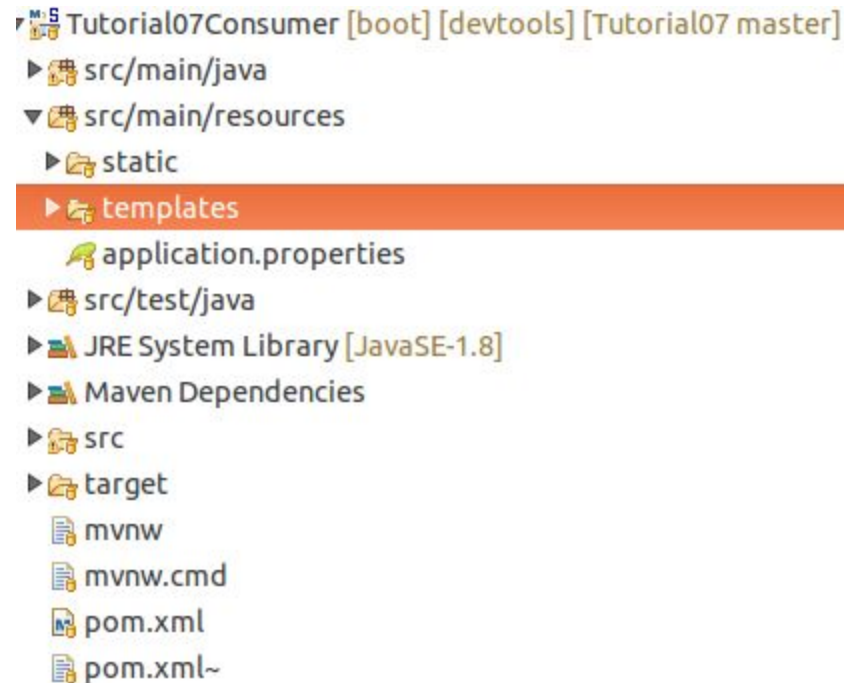
JSON Raw Data Headers

Save Copy

```
{  "idCourse": "CSC123",  "name": "PSP",  "credits": 4,  "students": [    {      "npm": "124",      "name": "Chanek J.r",      "gpa": 3.4,      "courses": null    }  ]}
```

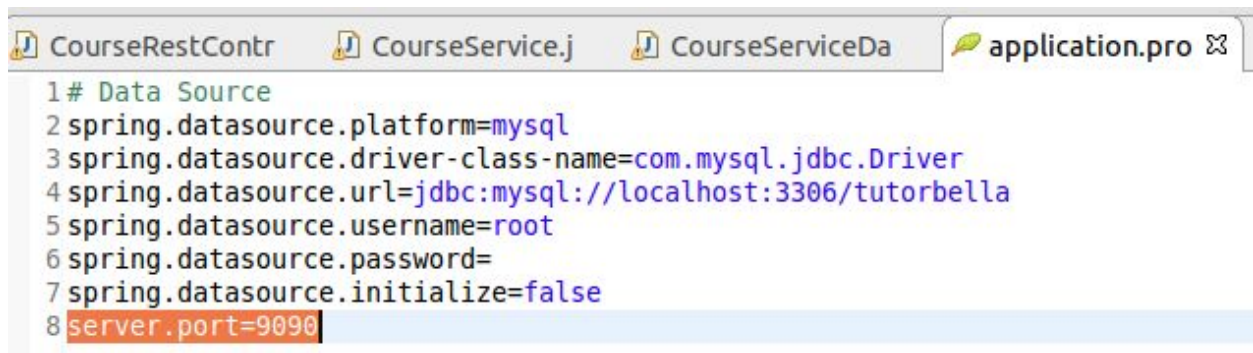
## Consumer

Import berkas-berkas yang ada di tutorial 6 untuk Service Consumer ini untuk keperluan Tutorial 7 sebagai consumer



Karena Service Producer dan Service Consumer harus dijalankan secara bersamaan, ubah berkas application.properties

Nantinya Aplikasi *Service Producer* akan dijalankan di localhost:8080 sedangkan *Service Consumer* akan dijalankan di localhost:9090



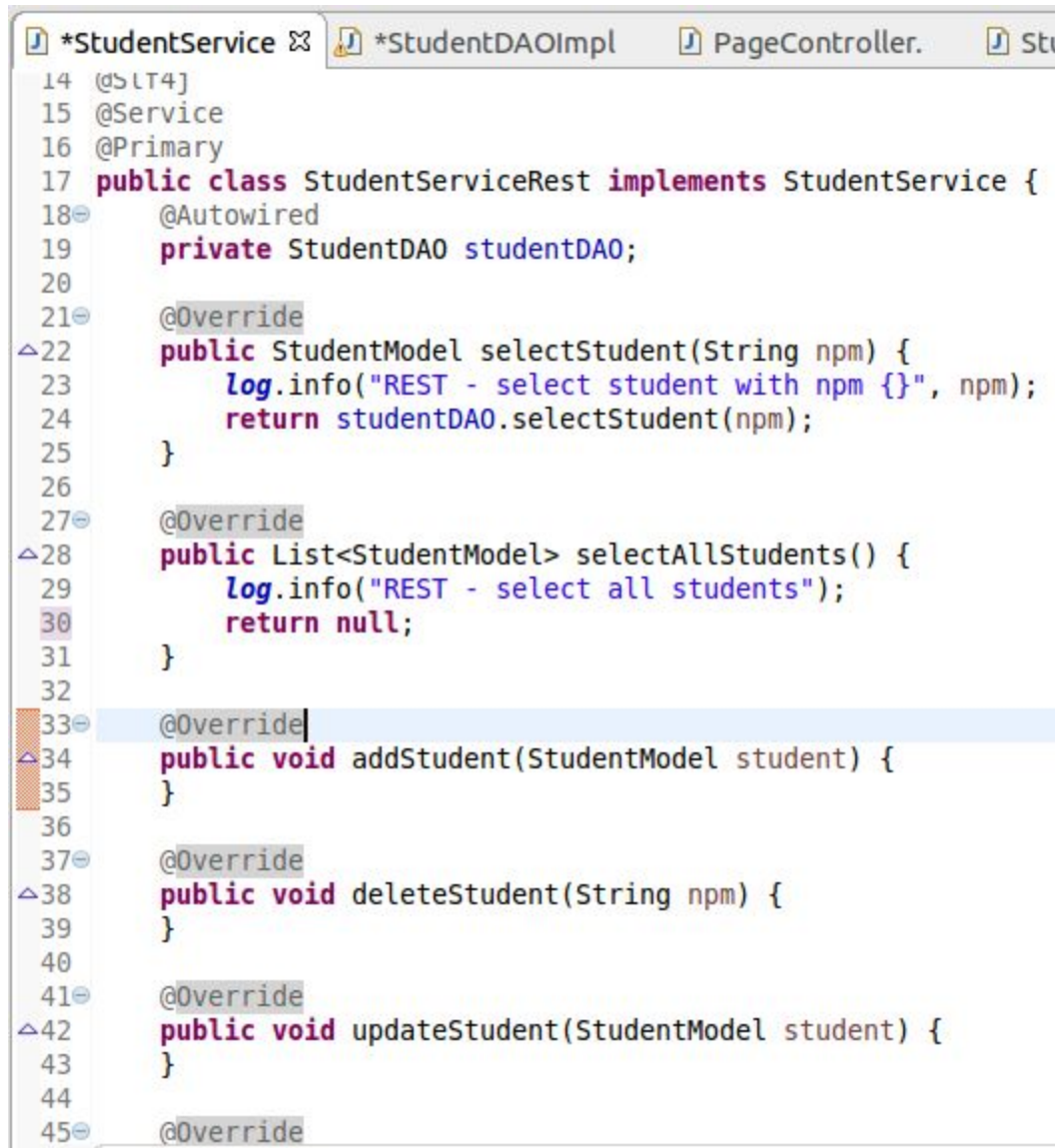
Di DAO tambahkan interface dengan nama StudentDAO

```
CourseRestContr  application.pro  PageController.  StudentDAO.java
1 package com.example.dao;
2
3 import java.util.List;
4
5
6
7 public interface StudentDAO {
8     StudentModel selectStudent(String npm);
9
10    List<StudentModel> selectAllStudents();
11 }
```

Selanjutnya kita akan membuat implementasi kelas StudentDAO tersebut dengan nama kelas StudentDAOImpl.java. Isinya adalah sebagai berikut

```
application.pro  *StudentDAOImpl  PageController.  StudentDA
1 package com.example.dao;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import org.springframework.web.client.RestTemplate;
8
9 import com.example.model.StudentModel;
10
11 @Service
12 public class StudentDAOImpl implements StudentDAO {
13     @Autowired
14     private RestTemplate restTemplate;
15
16     @Override
17     public StudentModel selectStudent(String npm) {
18         StudentModel student = restTemplate.getForObject(
19             "http://localhost:8080/rest/student/view/" + npm,
20             StudentModel.class);
21         return student;
22     }
23 }
```

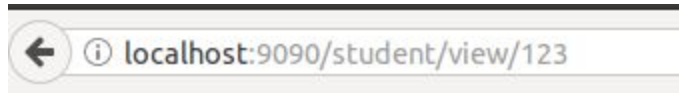
Selanjutnya, kita ingin mengubah agar StudentService mengambil data dari *web service* bukan dari *database*. Kita tidak perlu menghapus class **StudentServiceDatabase**. Karena *scalable system*, kita cukup menambahkan class baru yaitu **StudentServiceRest** yang mengimplement StudentService di *package service*.

A screenshot of an IDE window showing the implementation of the StudentService interface. The window has multiple tabs: \*StudentService, \*StudentDAOImpl, PageController, and St... The code is in Java and implements the StudentService interface using Spring annotations. The class is named StudentServiceRest and implements the StudentService interface. It has a private field studentDAO of type StudentDAO, which is annotated with @Autowired. There are five methods implemented: selectStudent, selectAllStudents, addStudent, deleteStudent, and updateStudent. Each method is annotated with @Override. The selectStudent method logs the request and returns the result from studentDAO. The selectAllStudents method logs the request and returns null. The addStudent method is currently empty. The deleteStudent and updateStudent methods are also currently empty. The code is as follows:

```
14 @SLT4]
15 @Service
16 @Primary
17 public class StudentServiceRest implements StudentService {
18     @Autowired
19     private StudentDAO studentDAO;
20
21     @Override
22     public StudentModel selectStudent(String npm) {
23         log.info("REST - select student with npm {}", npm);
24         return studentDAO.selectStudent(npm);
25     }
26
27     @Override
28     public List<StudentModel> selectAllStudents() {
29         log.info("REST - select all students");
30         return null;
31     }
32
33     @Override
34     public void addStudent(StudentModel student) {
35     }
36
37     @Override
38     public void deleteStudent(String npm) {
39     }
40
41     @Override
42     public void updateStudent(StudentModel student) {
43     }
44
45     @Override
```

Jalankan kedua project Spring Boot untuk *Service Producer* dan *Service Consumer* Tersebut. Pastikan *service producer* sudah berjalan dengan menjalankan **localhost:8080/rest/student/view/123**. Untuk menguji *service consumer* buka **localhost:9090/student/view/123**





- [Home](#)
- [Daftar Mahasiswa](#)
- [Daftar Course](#)

**NPM = 123**

**Name = Chanek**

**GPA = 3.6**

**Kuliah yang diambil**

- MPKT-6 sks

---

**Copyright © Mata Kuliah APAP**

Console Producer

```
Console
Tutorial07Producer - ApapTutorial07Producer [Spring Boot App] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Nov 4, 2017, 4:01:26 PM)
2017-11-04 16:14:13.231 INFO 9244 --- [nio-8080-exec-3] c.example.service.CourseServiceDatabase : select all courses
2017-11-04 16:14:24.575 INFO 9244 --- [nio-8080-exec-5] c.example.service.CourseServiceDatabase : select all courses
2017-11-04 16:14:39.627 INFO 9244 --- [nio-8080-exec-7] c.example.service.CourseServiceDatabase : select all courses
2017-11-04 16:14:40.826 INFO 9244 --- [nio-8080-exec-8] c.example.service.CourseServiceDatabase : select course with id_course CSC123
2017-11-04 18:28:57.944 INFO 9244 --- [nio-8080-exec-10] c.e.service.StudentServiceDatabase : select student with npm 123
2017-11-04 18:33:11.458 INFO 9244 --- [nio-8080-exec-2] c.e.service.StudentServiceDatabase : select all students
2017-11-04 18:39:39.890 INFO 9244 --- [nio-8080-exec-4] c.example.service.CourseServiceDatabase : select all courses
2017-11-04 18:40:22.193 INFO 9244 --- [nio-8080-exec-5] c.example.service.CourseServiceDatabase : select course with id_course CSC123
2017-11-04 18:48:26.732 INFO 9244 --- [nio-8080-exec-7] c.example.service.CourseServiceDatabase : select all courses
2017-11-04 18:48:37.511 INFO 9244 --- [nio-8080-exec-9] c.e.service.StudentServiceDatabase : select student with npm 123
```

Console Consumer

```
2017-11-04 16:14:24.573 INFO 9195 --- [nio-9090-exec-3] com.example.service.CourseServiceRest : REST - select all courses
2017-11-04 16:14:39.625 INFO 9195 --- [nio-9090-exec-3] com.example.service.CourseServiceRest : REST - select all courses
2017-11-04 16:14:40.821 INFO 9195 --- [nio-9090-exec-6] com.example.service.CourseServiceRest : REST - select course with id CSC123
2017-11-04 18:48:26.731 INFO 9195 --- [nio-9090-exec-5] com.example.service.CourseServiceRest : REST - select all courses
2017-11-04 18:48:37.508 INFO 9195 --- [nio-9090-exec-1] com.example.service.StudentServiceRest : REST - select student with npm 123
StudentModel(npm=123, name=Chanek, gpa=3.6, courses=[CourseModel(idCourse=CSC126, name=MPKT, credits=6, students=null)])
```

## LATIHAN

### LATIHAN 3

Implementasikan *service consumer* untuk view all Students dengan melengkapi *method* `selectAllStudents` yang ada di kelas `StudentServiceRest`

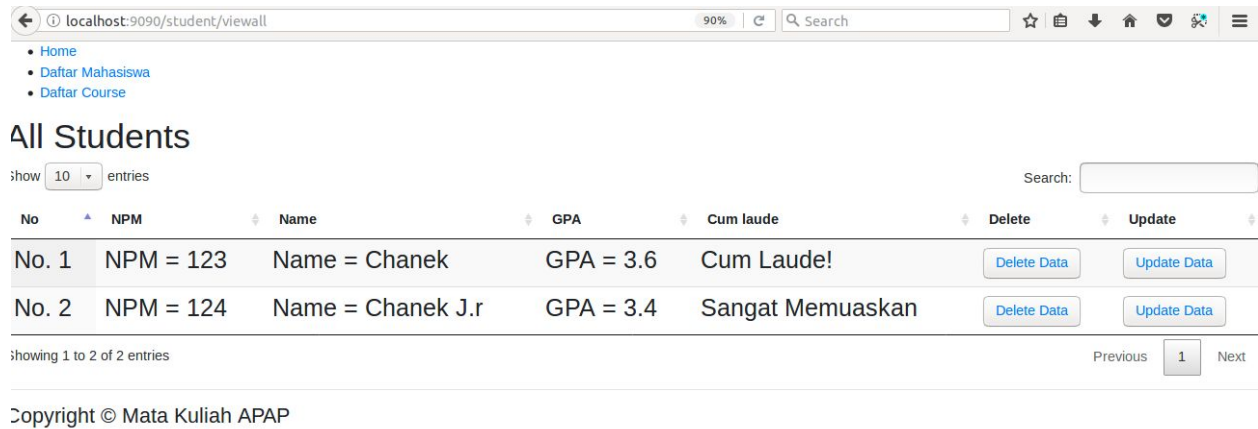
Melengkapi bagian DAO untuk select ALL students dengan cara panggil rest api untuk viewall

```
11 @Service
12 public class StudentDAOImpl implements StudentDAO {
13     @Autowired
14     private RestTemplate restTemplate;
15
16     @Override
17     public StudentModel selectStudent(String npm) {
18         StudentModel student = restTemplate.getForObject(
19             "http://localhost:8080/rest/student/view/" + npm,
20             StudentModel.class);
21         return student;
22     }
23
24     @Override
25     public List<StudentModel> selectAllStudents() {
26         List<StudentModel> students = restTemplate.getForObject(
27             "http://localhost:8080/rest/student/viewall",
28             List.class);
29         return students;
30     }
31 }
```

Melengkapi StudentServiceRest

```
14 @SUT4]
15 @Service
16 @Primary
17 public class StudentServiceRest implements StudentService {
18     @Autowired
19     private StudentDAO studentDAO;
20
21     @Override
22     public StudentModel selectStudent(String npm) {
23         log.info("REST - select student with npm {}", npm);
24         return studentDAO.selectStudent(npm);
25     }
26
27     @Override
28     public List<StudentModel> selectAllStudents() {
29         log.info("REST - select all students");
30         return studentDAO.selectAllStudents();
31     }
32 }
```

Buka localhost:9090/student/viewall



No	NPM	Name	GPA	Cum laude	Delete	Update
No. 1	NPM = 123	Name = Chanek	GPA = 3.6	Cum Laude!	<a href="#">Delete Data</a>	<a href="#">Update Data</a>
No. 2	NPM = 124	Name = Chanek J.r	GPA = 3.4	Sangat Memuaskan	<a href="#">Delete Data</a>	<a href="#">Update Data</a>

Showing 1 to 2 of 2 entries

Previous 1 Next

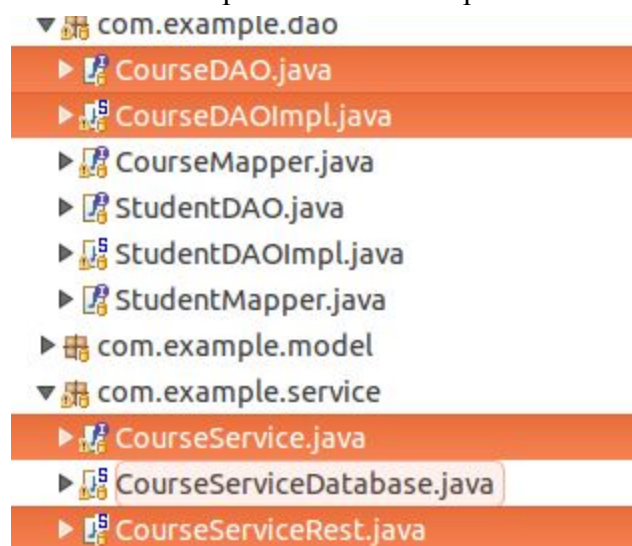
Copyright © Mata Kuliah APAP

Singkatnya, kita hanya perlu melengkapi dan merubah implementasi sebelumnya yang menggunakan database langsung, menjadi menggunakan rest api yang sudah dibuat sebelumnya. Tentunya harus dipastikan, producer dalam keadaan hidup.

#### LATIHAN 4

Implementasikan *service consumer* untuk *class CourseModel* dengan membuat *class-class DAO* dan *service baru*

Membuat beberapa kelas relevan seperti DAO dan service untuk courseModel.



Implementasi CourseDAO dan CourseDAOImpl. Caranya mirip seperti saat kita ingin implementasi untuk student.

```
StudentServiceR *StudentDAOImpl CourseDAO.java
1 package com.example.dao;
2
3 import java.util.List;
4
5 import com.example.model.CourseModel;
6
7 public interface CourseDAO {
8     CourseModel selectCourse(String npm);
9
10    List<CourseModel> selectAllCourses();
11 }
12
```

```
StudentServiceR *StudentDAOImpl CourseDAO.java *CourseDAOImpl
1 package com.example.dao;
2
3 import java.util.List;
4
5 @Service
6 public class CourseDAOImpl implements CourseDAO {
7     @Autowired
8     private RestTemplate restTemplate;
9
10    @Override
11    public CourseModel selectCourse(String id) {
12        CourseModel course = restTemplate.getForObject(
13            "http://localhost:8080/rest/course/view/" + id,
14            CourseModel.class);
15        return course;
16    }
17
18    @Override
19    public List<CourseModel> selectAllCourses() {
20        List<CourseModel> courses = restTemplate.getForObject(
21            "http://localhost:8080/rest/course/viewall",
22            List.class);
23        return courses;
24    }
25 }
```

Singkatnya, DAO tersebut akan panggil rest api yang producer sediakan.

Update CourseService pada tutorial 6 sebelumnya, agar punya method select all courses.



```
CourseDAO.java *CourseDAOImpl. CourseService.j
1 package com.example.service;
2
3 import java.util.List;
4
5
6
7
8 public interface CourseService
9 {
10     CourseModel selectCourse (String id_course);
11
12     List<CourseModel> selectAllCourses ();
13 }
14
```

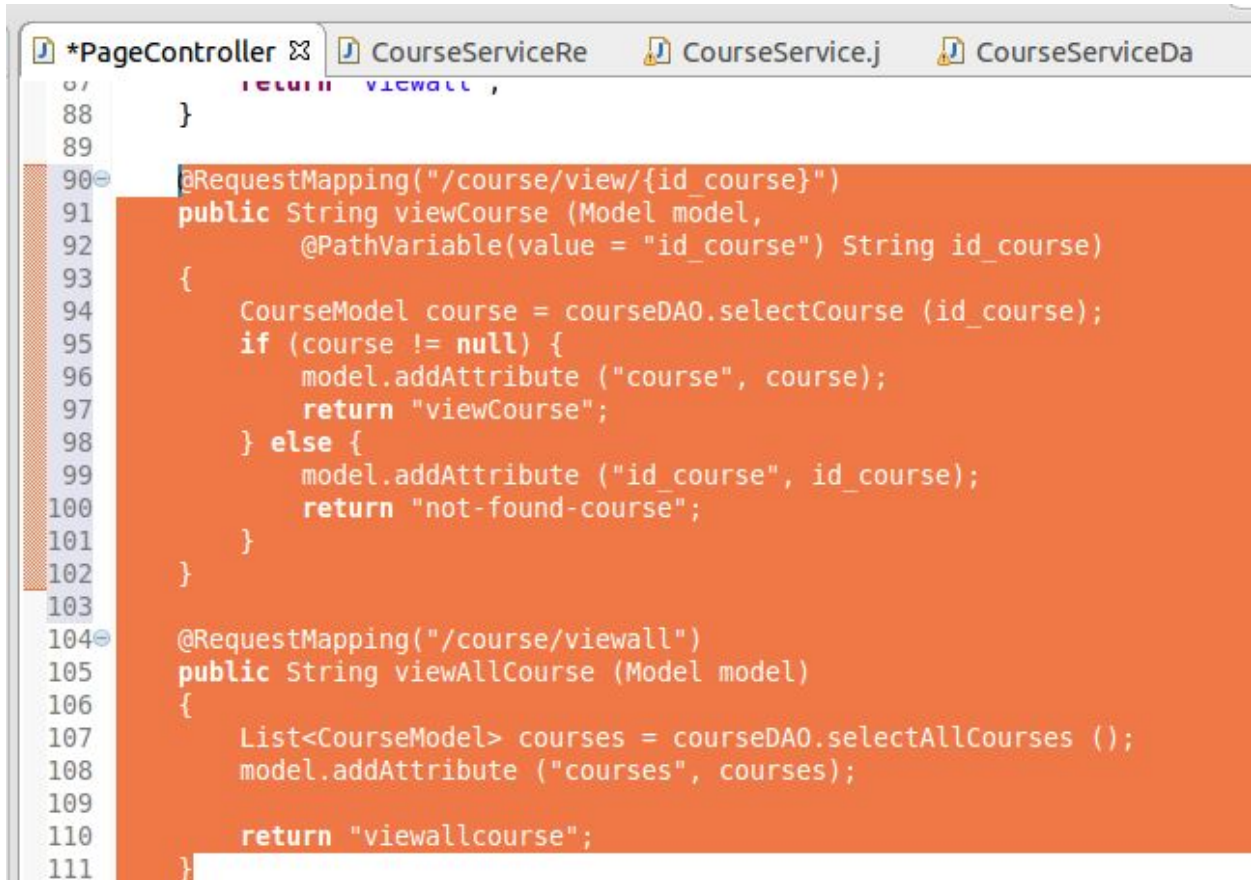
Implementasi course service.

```
*CourseDAOImpl. CourseServiceRe CourseService.j CourseSe
1 package com.example.service;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14 @Slf4j
15 @Service
16 @Primary
17 public class CourseServiceRest implements CourseService {
18     @Autowired
19     private CourseDAO courseDAO;
20
21     @Override
22     public CourseModel selectCourse(String id_course) {
23         log.info("REST - select course with id {}", id_course);
24         return courseDAO.selectCourse(id_course);
25     }
26
27     @Override
28     public List<CourseModel> selectAllCourses() {
29         log.info("REST - select all courses");
30         return courseDAO.selectAllCourses();
31     }
32 }
33
```

Singkatnya, service tersebut akan panggil DAO yang tadi sudah kita buat, tentunya DAO akan melanjutkannya dengan memanggil request Rest API ke producer. Response akan dikirimkan ke controller yang memanggilnya.



Kita update controller kita untuk menerima request dari user, kemudian dapat menampilkan responsennya.



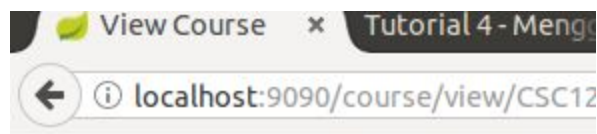
```
87         return viewall ,
88     }
89
90     @RequestMapping("/course/view/{id_course}")
91     public String viewCourse (Model model,
92         @PathVariable(value = "id_course") String id_course)
93     {
94         CourseModel course = courseDAO.selectCourse (id_course);
95         if (course != null) {
96             model.addAttribute ("course", course);
97             return "viewCourse";
98         } else {
99             model.addAttribute ("id_course", id_course);
100             return "not-found-course";
101         }
102     }
103
104     @RequestMapping("/course/viewall")
105     public String viewAllCourse (Model model)
106     {
107         List<CourseModel> courses = courseDAO.selectAllCourses ();
108         model.addAttribute ("courses", courses);
109
110         return "viewallcourse";
111     }
```

Kita buat view/templates baru untuk viewallcourse karena sebelumnya belum dibuat di tutorial6

```
*PageController  CourseServiceRe  CourseService.j  viewallcourse.h  »19
10      <table id="myTable2" class="display">
11          <thead>
12              <tr>
13                  <th>No</th>
14                  <th>Id</th>
15                  <th>Name</th>
16                  <th>Credits</th>
17                  <th>Action</th>
18              </tr>
19          </thead>
20          <tbody>
21              <div th:each="course,iterationStatus: ${courses}" th:class="${iterationStatus.
22                  <tr>
23                      <td><h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3></td>
24                      <td><h3 th:text="'Id| = ' + ${course.idCourse}"></h3></td>
25                      <td><h3 th:text="'Name = ' + ${course.name}"></h3></td>
26                      <td><h3 th:text="'Credits = ' + ${course.credits}"></h3></td>
27                  </tr>
28              </div>
29          </tbody>
30      </table>
31      <td><a th:href="'/course/view/' + ${course.idCourse}" > View Detail</a>
32      </td>
33      </tr>
34  </tbody>
35  </table>
```

Ps. mirip dengan viewall student, hanya beda field dan kolom-kolomnya saja.

Hasilnya untuk view course.



- [Home](#)
- [Daftar Mahasiswa](#)
- [Daftar Course](#)

**ID = CSC124**

**Name = SDA**

**Credits = 3**

**Mahasiswa yang mengambil**

- 124-Chanek J.r

---

**Copyright © Mata Kuliah APAP**

Hasilnya untuk viewallcourse

localhost:9090/course/viewall

90%

Search

Home

Daftar Mahasiswa

Daftar Course

## All Students

Show 10 entries

Search:

No	Id	Name	Credits	Action
No. 1	Id = CSC123	Name = PSP	Credits = 4	<a href="#">View Detail</a>
No. 2	Id = CSC124	Name = SDA	Credits = 3	<a href="#">View Detail</a>
No. 3	Id = CSC125	Name = DDP 1	Credits = 4	<a href="#">View Detail</a>
No. 4	Id = CSC126	Name = MPKT	Credits = 6	<a href="#">View Detail</a>

Showing 1 to 4 of 4 entries

Previous

1

Next

Copyright © Mata Kuliah APAP

Selesai!

## Lesson Learned

Kita jadi tau lebih banyak tentang penggunaan Spring MVC dalam dunia pemrograman perusahaan dimana segalanya menggunakan API dan ada client/consumer yang menggunakan API tersebut untuk keperluan mobile programming, frontend, dan lain-lain. Disini kita belajar menggunakan Rest service. Dalam dunia pemrograman, sekarang jadi dibagi menjadi 2 layer, yaitu producer dan consumer.

Pada tutorial kali ini, kita diajarkan cara membuat REST API beserta konsumernya menggunakan Spring MVC. Tutorial ini dibagi menjadi 2 tahapan, mulai dari membuat API (Producer) dan membuat Client (Consumer). Secara singkatnya, API yang melakukan interaksi dengan backend mulai dari database, struktur data, data-data internal, dan lain-lain. API atau Producer juga yang menerima request dari client terhadap suatu service tertentu. Client atau Consumer disini berperan sebagai jembatan utama antara sistem dengan pengguna. Consumer akan memanggil API yang tersedia oleh Producer, kemudian menerima response kembalian API tersebut dalam format JSON yang kemudian dapat diolah kembali menjadi response kembalian yang dapat ditampilkan kepada pengguna.