

Ringkasan Materi

Model digunakan sebagai representasi data yang akan digunakan dalam program, baik itu tabel pada database, atau hanya sebatas kumpulan variable local yang digunakan dalam bentuk class, atau juga merupakan representasi JavaScript Object Notation (JSON) untuk API.

Service merupakan kumpulan helper yang digunakan dalam implementasi di Controller, digunakan untuk menghubungkan antara Controller dengan Model. Service dapat juga dibuat bentuk interface nya agar dapat diimplementasi dalam berbagai cara implementasi.

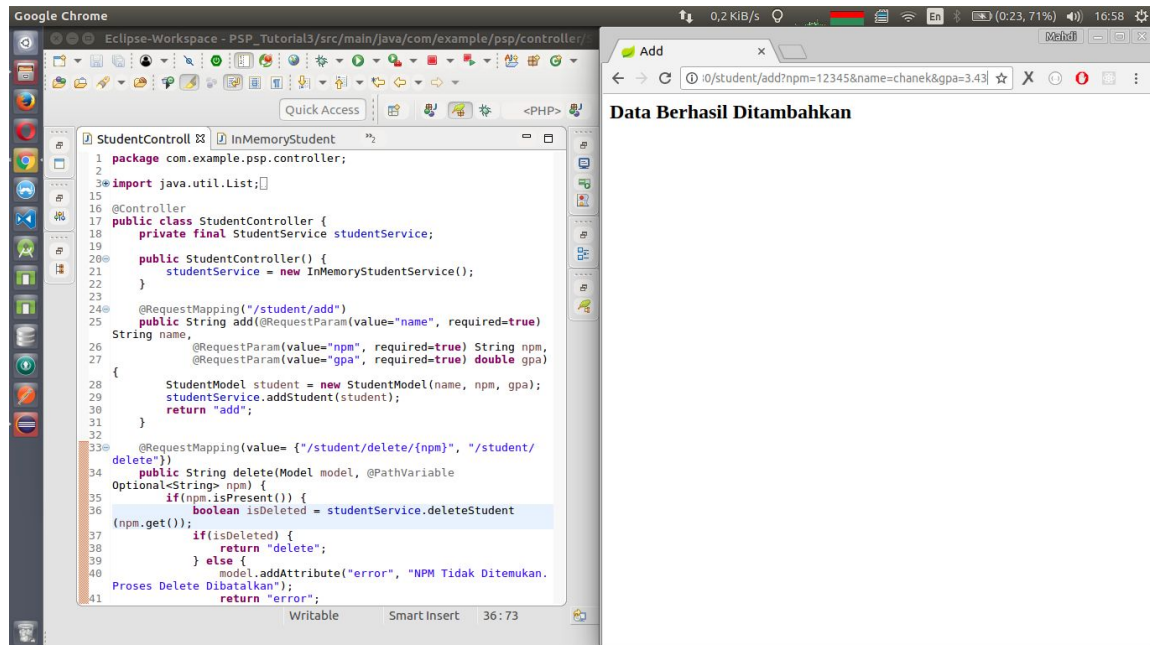
Tanpa Service pun sebuah program Spring Boot dapat berjalan, hanya saja semua implementasi helper antara Controller dan Model harus dilakukan dalam Controller, dan hal ini menghilangkan modularity.

Untuk melakukan implementasi `@PathVariable` dengan variable yang Optional, maka routing harus memiliki semua kemungkinan yang ada, baik jika variable tersebut ada, ataupun tidak ada. Hal ini harus dilakukan agar jika variable tersebut tidak ada, program tidak langsung throw error *CMIW.

Jawaban Pertanyaan

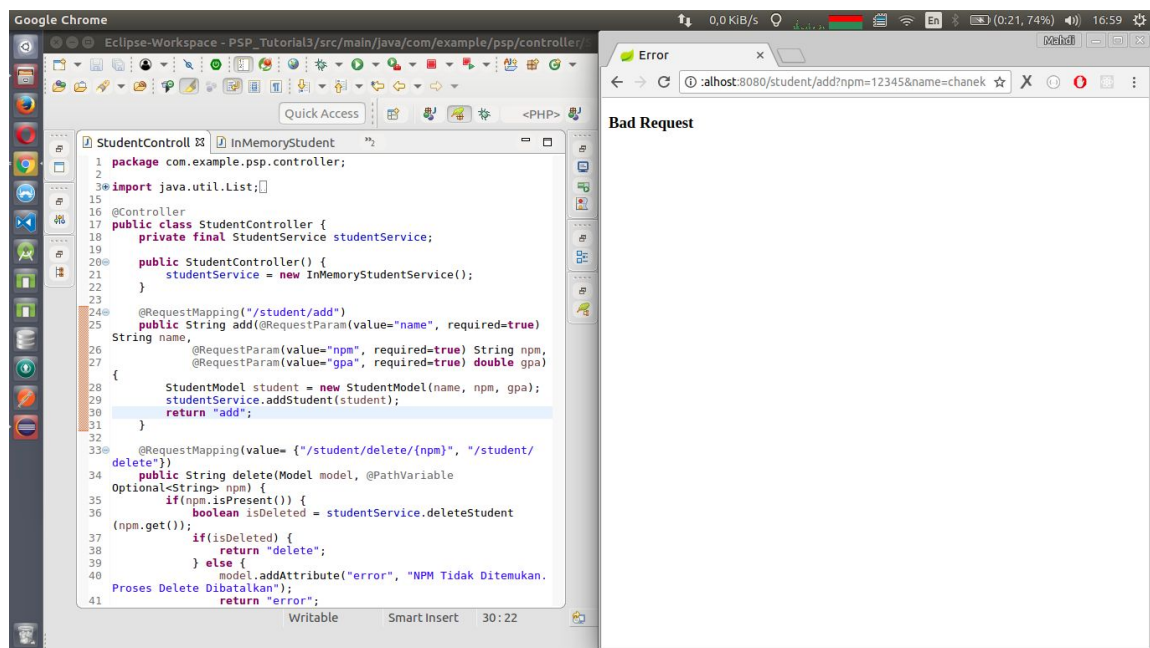
1. Jalankan "localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43"

Data berhasil ditambahkan tanpa adanya masalah



2. Jalankan "localhost:8080/student/add?npm=12345&name=chanek"

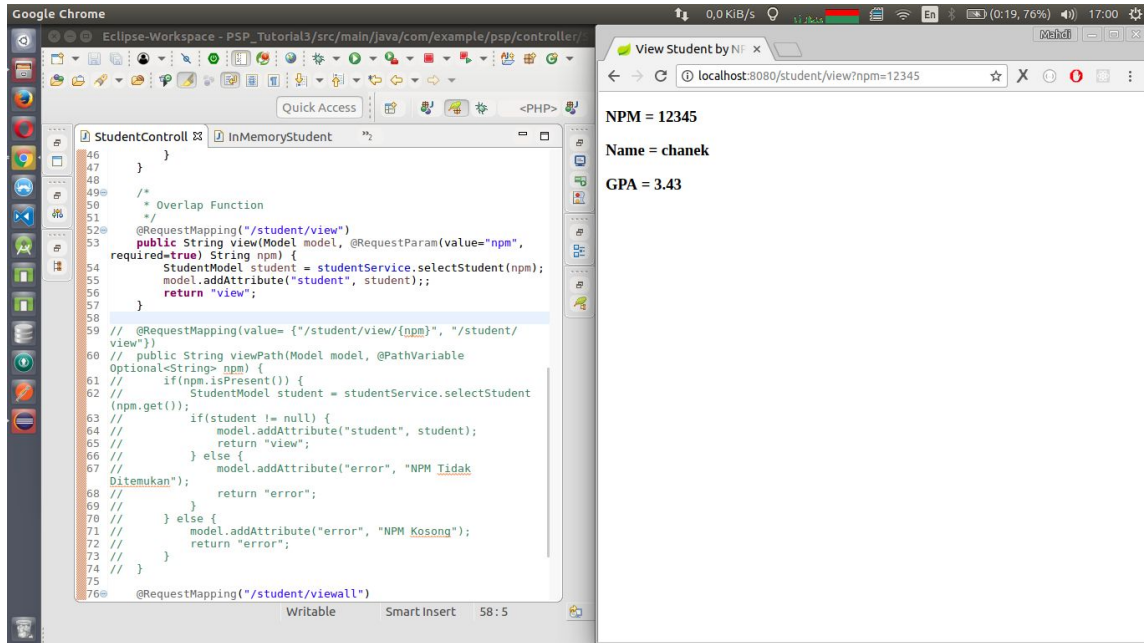
Menghasilkan "Bad Request" karena seluruh parameter bersifat Required, dan parameter ?npm tidak terpenuhi.



3. Jalankan

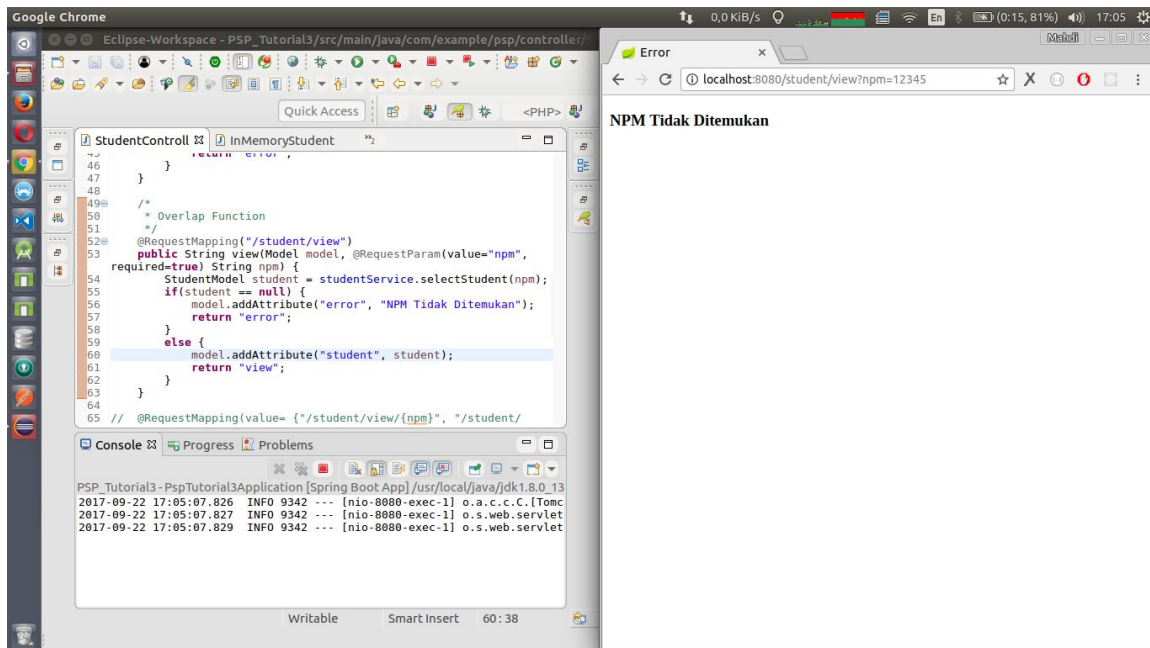
“localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43 lalu buka localhost:8080/student/view?npm=12345”

Data student tersebut muncul dengan normal.



4. Jalankan “localhost:8080/student/view?npm=12345”

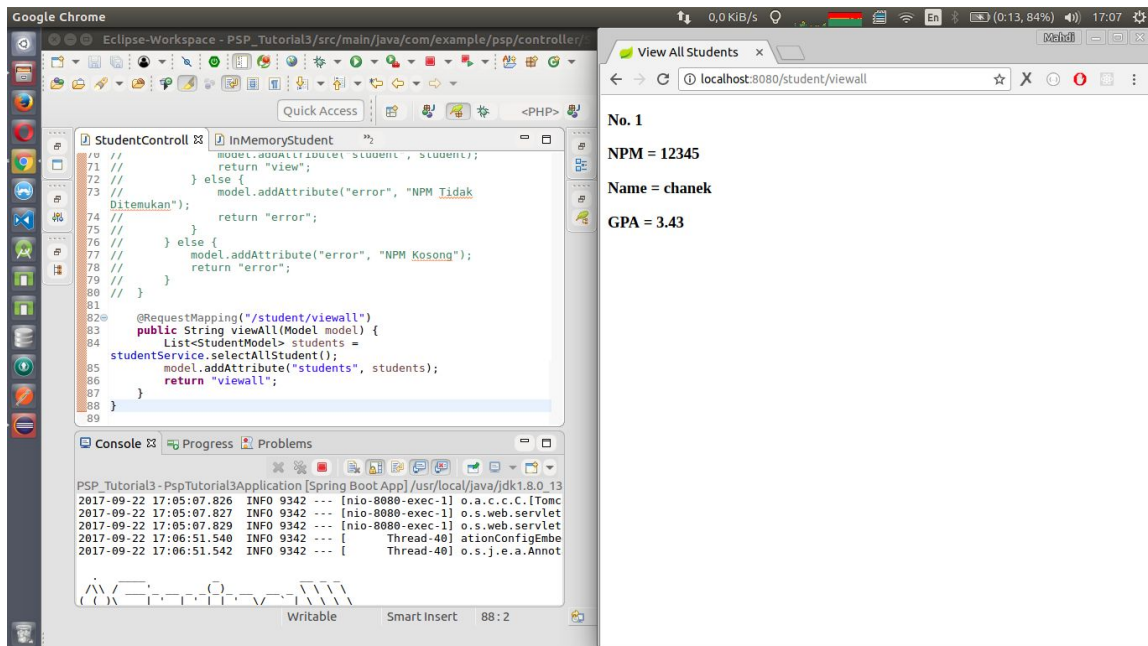
Data tidak muncul, hal ini dikarenakan List of Student Model akan di reset setiap kali program di build ulang. Sehingga belum ada student dengan NPM=12345.



5. Jalankan

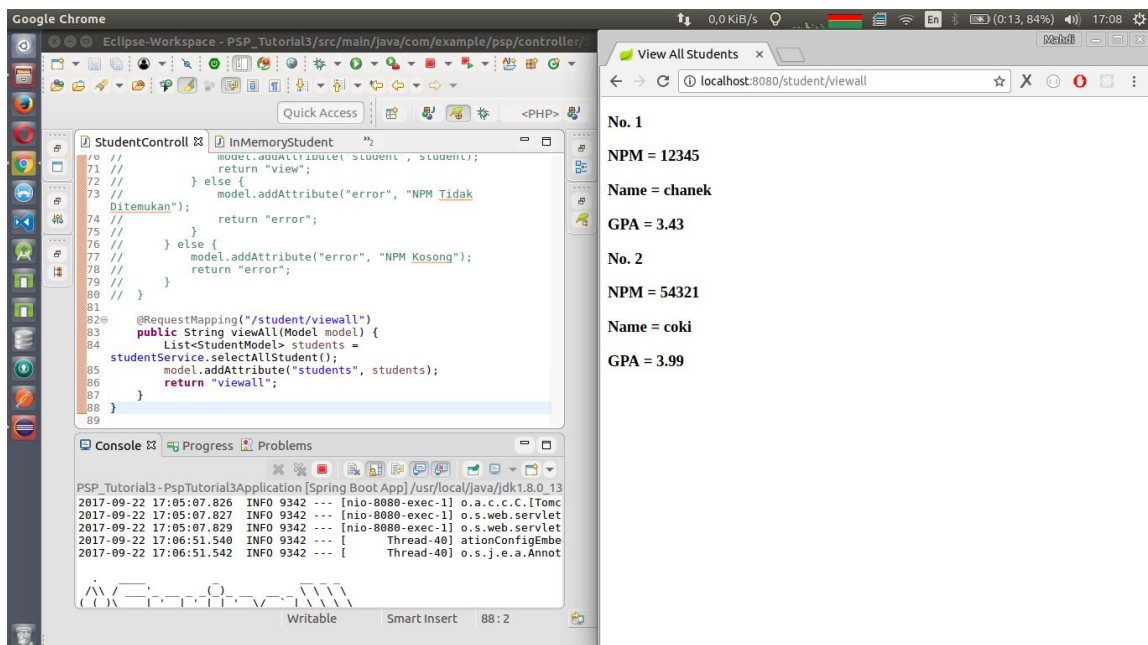
“localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43 lalu buka localhost:8080/student/viewall”

Data student dengan NPM=12345 muncul.



6. Berdasarkan Pertanyaan 5, tambahkan lagi student lainnya

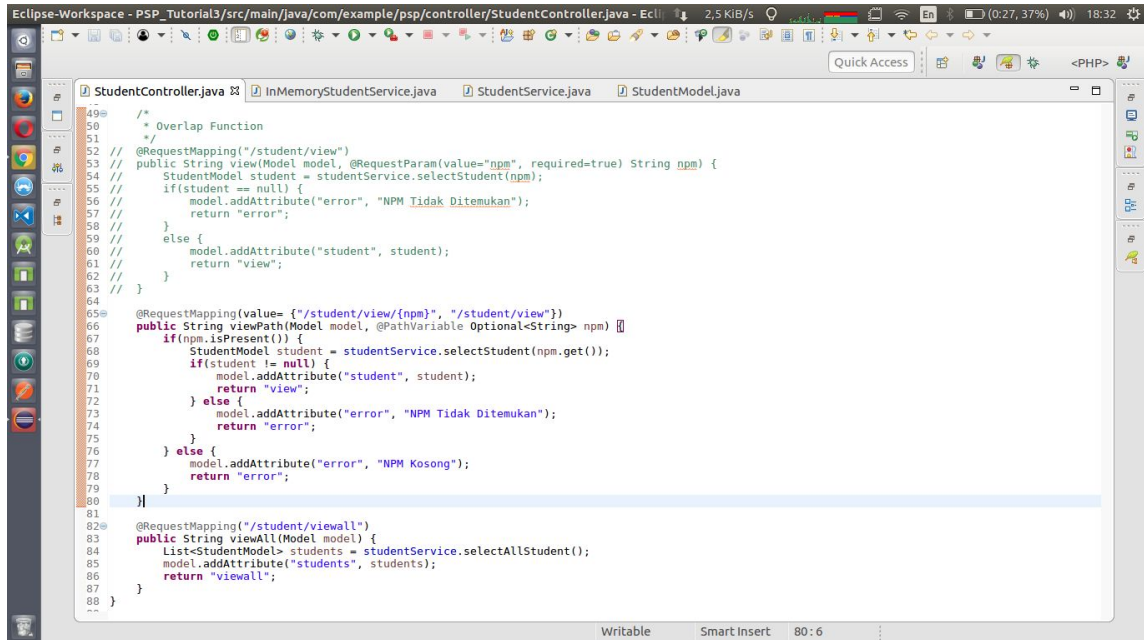
Data keseluruhan student muncul secara berurutan sesuai dengan saat masuknya data kedalam List of Student.



Fitur Baru

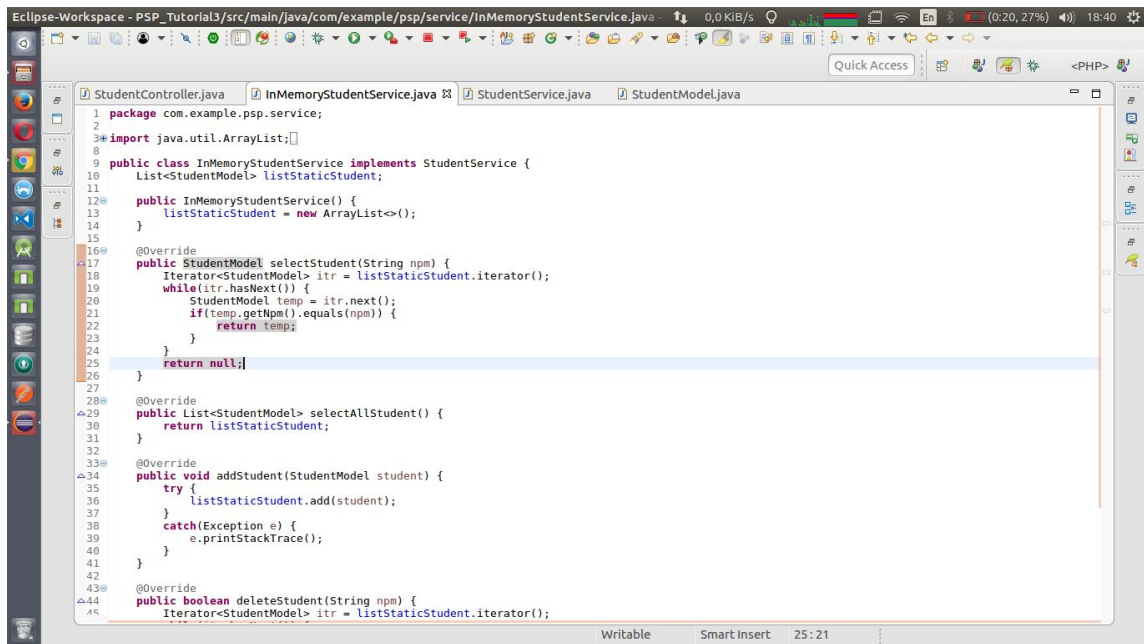
1. Select Student dengan path variable.

Method pada Controller dibuat dengan dua buah routing, “/student/view/{npm}” dan “/student/view”. Kemudian melakukan handling apakah terdapat path variable npm atau tidak. Jika path variable npm ada, maka akan dicek apakah npm tersebut terdaftar atau tidak dengan menggunakan method pada Service.



```
49 //  
50 /*  
51  * Overlap Function  
52  */  
53 // @RequestMapping("/student/view")  
54 // public String view(Model model, @RequestParam(value="npm", required=true) String npm) {  
55 //     StudentModel student = studentService.selectStudent(npm);  
56 //     if(student == null) {  
57 //         model.addAttribute("error", "NPM Tidak Ditemukan");  
58 //         return "error";  
59 //     }  
60 //     else {  
61 //         model.addAttribute("student", student);  
62 //         return "view";  
63 //     }  
64 // }  
65  
66 @RequestMapping(value= {""/student/view/{npm}", "/student/view"})  
67 public String viewPath(Model model, @PathVariable Optional<String> npm) {  
68     if(npm.isPresent()) {  
69         StudentModel student = studentService.selectStudent(npm.get());  
70         if(student != null) {  
71             model.addAttribute("student", student);  
72             return "view";  
73         } else {  
74             model.addAttribute("error", "NPM Tidak Ditemukan");  
75             return "error";  
76         }  
77     } else {  
78         model.addAttribute("error", "NPM Kosong");  
79         return "error";  
80     }  
81 }  
82  
83 @RequestMapping("/student/viewall")  
84 public String viewAll(Model model) {  
85     List<StudentModel> students = studentService.selectAllStudent();  
86     model.addAttribute("students", students);  
87     return "viewall";  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }
```

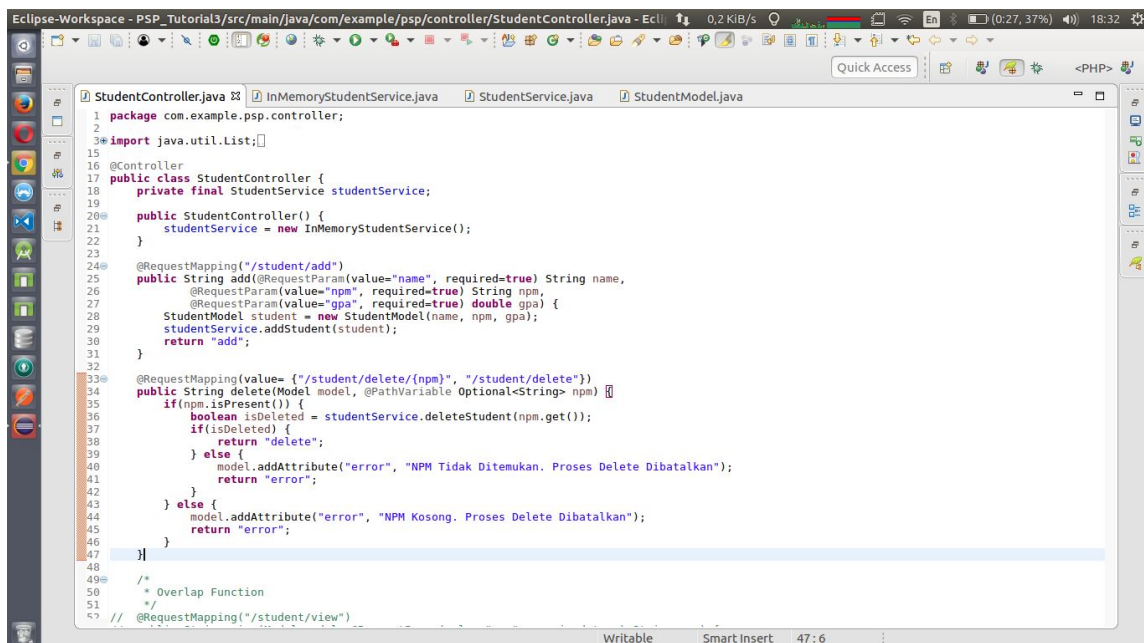

Method pada Service dibuat dengan mengimplementasikan class Iterator pada List of StudentModel untuk mengiterasi List tersebut. Jika NPM tersebut terdapat pada List of StudentModel, maka akan di return data StudentModel tersebut, jika tidak maka akan me-return null.



```
1 package com.example.psp.service;
2
3 import java.util.ArrayList;
4
5 public class InMemoryStudentService implements StudentService {
6     List<StudentModel> listStaticStudent;
7
8     public InMemoryStudentService() {
9         listStaticStudent = new ArrayList<>();
10    }
11
12    @Override
13    public StudentModel selectStudent(String npm) {
14        Iterator<StudentModel> itr = listStaticStudent.iterator();
15        while (itr.hasNext()) {
16            StudentModel temp = itr.next();
17            if (temp.getNpm().equals(npm)) {
18                return temp;
19            }
20        }
21        return null;
22    }
23
24    @Override
25    public List<StudentModel> selectAllStudent() {
26        return listStaticStudent;
27    }
28
29    @Override
30    public void addStudent(StudentModel student) {
31        try {
32            listStaticStudent.add(student);
33        } catch (Exception e) {
34            e.printStackTrace();
35        }
36    }
37
38    @Override
39    public boolean deleteStudent(String npm) {
40        Iterator<StudentModel> itr = listStaticStudent.iterator();
41        while (itr.hasNext()) {
42            StudentModel temp = itr.next();
43            if (temp.getNpm().equals(npm)) {
44                itr.remove();
45            }
46        }
47    }
48}
```

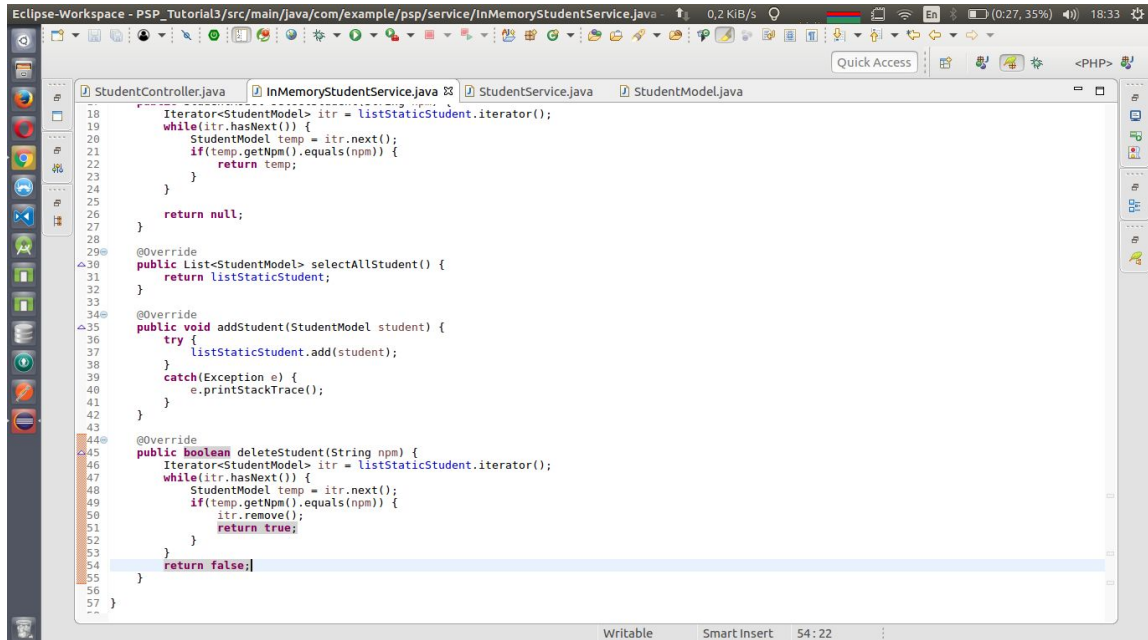
2. Delete Student berdasarkan NPM

Implementasi Delete Student pada Controller memiliki logic yang sama dengan Search Student. Mengecek apakah terdapat path variable npm atau tidak. Jika ada maka akan di cek apakah NPM tersebut terdaftar atau tidak dengan menggunakan method pada Service.



```
1 package com.example.psp.controller;
2
3 import java.util.List;
4
5 @Controller
6 public class StudentController {
7     private final StudentService studentService;
8
9     public StudentController() {
10        studentService = new InMemoryStudentService();
11    }
12
13    @RequestMapping("/student/add")
14    public String add(@RequestParam(value="name", required=true) String name,
15                    @RequestParam(value="npm", required=true) String npm,
16                    @RequestParam(value="gpa", required=true) double gpa) {
17        StudentModel student = new StudentModel(name, npm, gpa);
18        studentService.addStudent(student);
19        return "add";
20    }
21
22    @RequestMapping(value= {"{/student/delete/{npm}", "/student/delete"})
23    public String delete(Model model, @PathVariable Optional<String> npm) {
24        if (npm.isPresent()) {
25            boolean isDeleted = studentService.deleteStudent(npm.get());
26            if (isDeleted) {
27                return "delete";
28            } else {
29                model.addAttribute("error", "NPM Tidak Ditemukan. Proses Delete Dibatalkan");
30                return "error";
31            }
32        } else {
33            model.addAttribute("error", "NPM Kosong. Proses Delete Dibatalkan");
34            return "error";
35        }
36    }
37
38    /* Overlap Function
39    */
40    @RequestMapping("/student/view")
41}
```

Pada Service, implementasi dilakukan dengan me-return boolean sebagai flaging apakah data tersebut berhasil di hapus atau tidak. Jika flag tersebut true, maka data berhasil dihapus, jika tidak maka flag tersebut akan false. Implementasi dilakukan dengan Iterator seperti pada Search Student.



The screenshot shows the Eclipse IDE with the file `InMemoryStudentService.java` open. The code implements a `deleteStudent` method using an `Iterator` to traverse a list of students. The method returns a `boolean` flag indicating whether the deletion was successful.

```
18 Iterator<StudentModel> itr = listStaticStudent.iterator();
19 while(itr.hasNext()) {
20     StudentModel temp = itr.next();
21     if(temp.getNpm().equals(npm)) {
22         return temp;
23     }
24 }
25 return null;
26 }
27
28 @Override
29 public List<StudentModel> selectAllStudent() {
30     return listStaticStudent;
31 }
32
33 @Override
34 public void addStudent(StudentModel student) {
35     try {
36         listStaticStudent.add(student);
37     }
38     catch (Exception e) {
39         e.printStackTrace();
40     }
41 }
42
43
44 @Override
45 public boolean deleteStudent(String npm) {
46     Iterator<StudentModel> itr = listStaticStudent.iterator();
47     while(itr.hasNext()) {
48         StudentModel temp = itr.next();
49         if(temp.getNpm().equals(npm)) {
50             itr.remove();
51             return true;
52         }
53     }
54     return false;
55 }
56
57 }
```