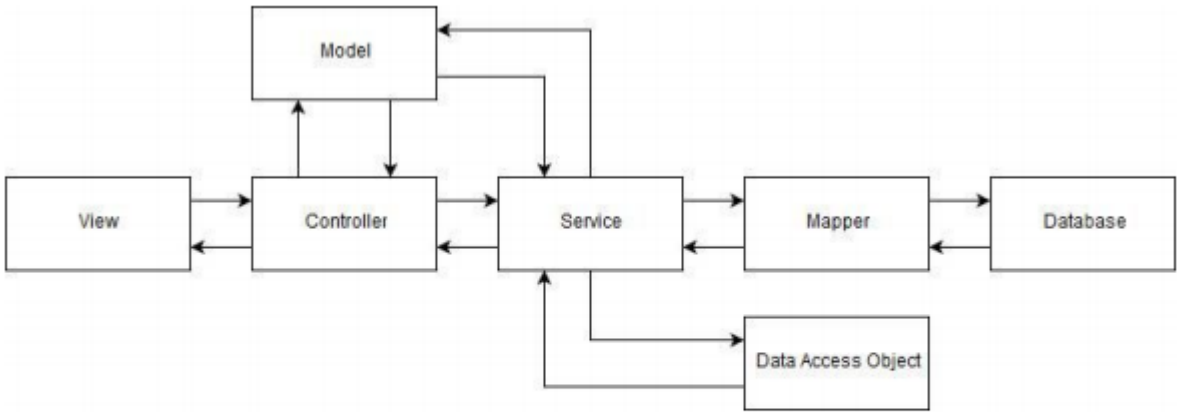
# Tutorial 3
## Menggunakan Model dan Service pada Project Spring Boot

A. Ringkasan dari materi tutorial

Pada tutorial 3 ini pembahasan fokus pada model dan service dari implementasi konsep MVC (Model View Controller). Model adalah sebuah objek yang merepresentasikan dan menyimpan informasi terhadap suatu hal. Contoh dari model adalah objek mahasiswa. Objek mahasiswa ini memiliki nama, alamat, tanggal lahir, nomor pokok mahasiswa, dsbnya. Model digunakan untuk merepresentasikan hal-hal tersebut dalam atribut yang dimiliki sebuah model.

Service adalah suatu layer yang menjadi mediator antara controller dan database. Pada service layer, disimpan business logic yang digunakan untuk mengolah data yang terdapat dalam database. Pengolahan ini meliputi kalkulasi data yang diambil dari database, manipulasi user input kedalam database, dsbnya. Contohnya adalah melakukan kalkulasi IPK sebuah model mahasiswa. Tutorial ini membahas komponen view dan controller beserta kaitannya dengan komponen model dan service.




* tanda panah merepresentasikan alur interaksi dan data antar komponen

Gambar 1. Kaitan antar komponen MVC di Spring Boot

B. Hasil jawaban dari setiap poin pada bagian tutorial

Jalankan program dan buka
localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43

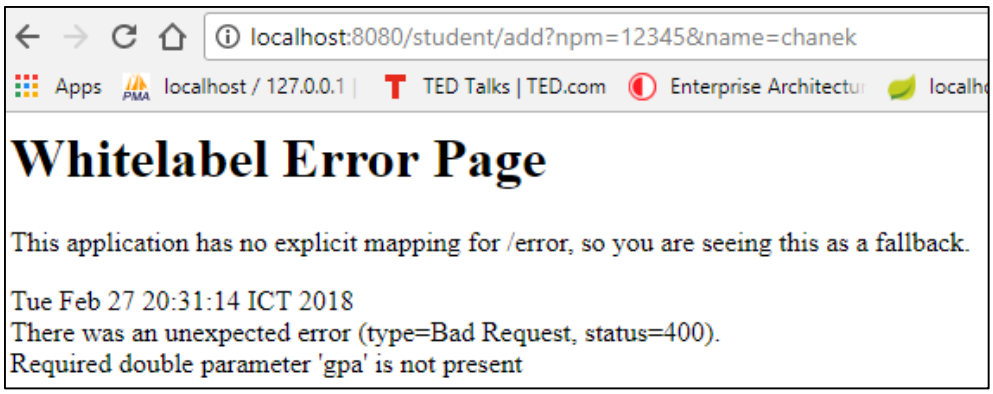
**Pertanyaan 1**: apakah hasilnya? Jika error, tuliskan penjelasan Anda.

**Jawaban:** Hasilnya tidak error. Data berhasil ditambahkan karena URL sudah sesuai dengan inputan di StudentController yaitu terdapat npm, name, dan gpa untuk kemudian data tersebut akan disimpan dalam objek student dan studentService akan memanggil method addStudent lalu return add untuk mengakses halaman add.

localhost:8080/student/add?npm=12345&name=chanek

**Pertanyaan 2**: apakah hasilnya? Jika error, tuliskan penjelasan Anda

**Jawaban**: Hasilnya error karena pada StudentController terdapat paramater gpa sedangan URL tidak memanggil gpa.




Jalankan program dan buka
localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43 lalu buka
localhost:8080/student/view?npm=12345

**Pertanyaan 3**: apakah data Student tersebut muncul? Jika tidak, mengapa?

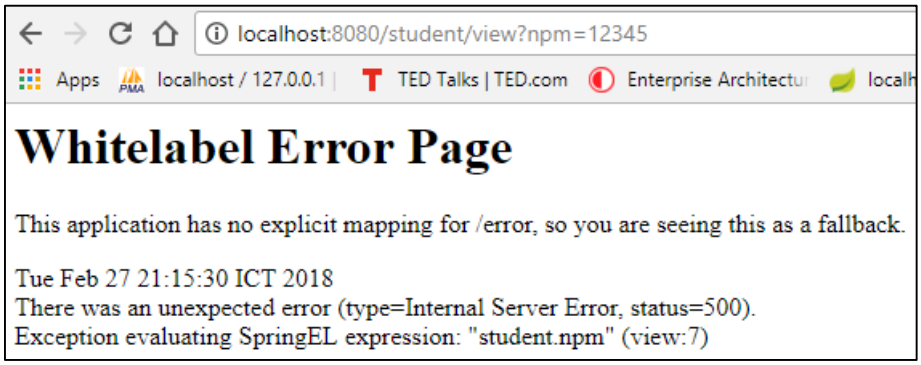
**Jawaban**: data student dapat muncul karena url yang pertama terdapat add dimana data student dimasukkan, kemudian pada url yang kedua dipanggil view dan npm student yang sama seperti yang ditambahkan sebelumnya yaitu 12345 maka akan ditampilkan halaman view berupa data student sesuai dengan npm yang dituliskan.



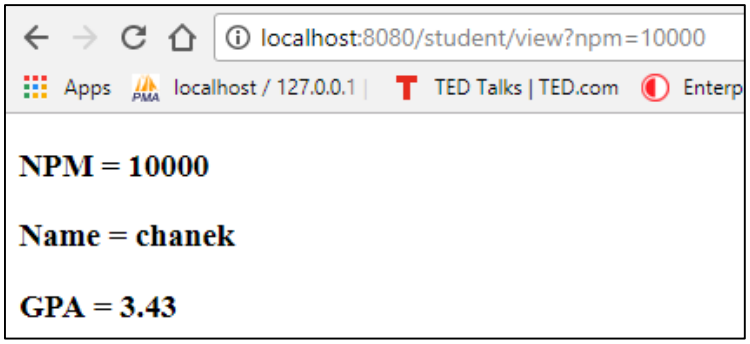
Coba matikan program dan jalankan kembali serta buka
localhost:8080/student/view?npm=12345

**Pertanyaan 4**: apakah data Student tersebut muncul? Jika tidak, mengapa?

**Jawaban**: data student tidak muncul karena data student yang di add sebelumnya dan masuk ke StudentList sudah dilakukan view sehingga ketika sudah dipanggil kemudian dipanggil lagi maka data sudah tidak tersimpan.



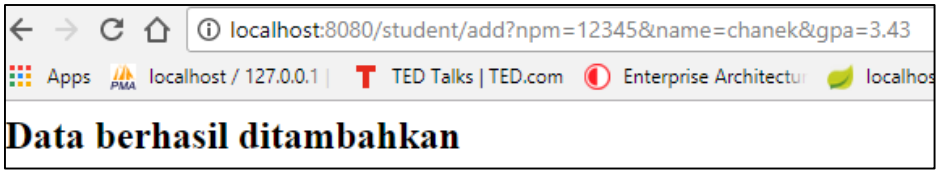

Coba tambahkan data Student lainnya dengan NPM yang berbeda.

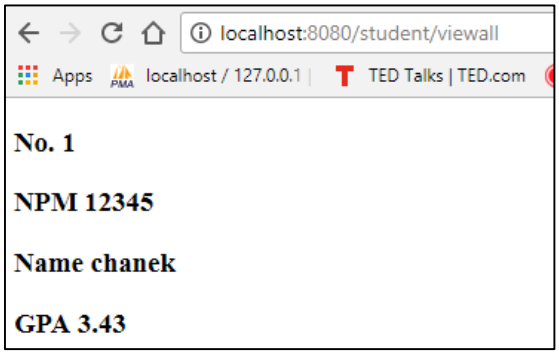



Jalankan program dan buka localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43 lalu buka localhost:8080/student/viewall

**Pertanyaan 5**: apakah data Student tersebut muncul?

**Jawaban**: data student muncul karena sama seperti method view sebelumnya namun disimpan dalam objek student yang berbentuk list dimana dapat di add dan disimpan banyak data student di dalamnya. Kemudian return viewall dan mengakses halaman viewall.

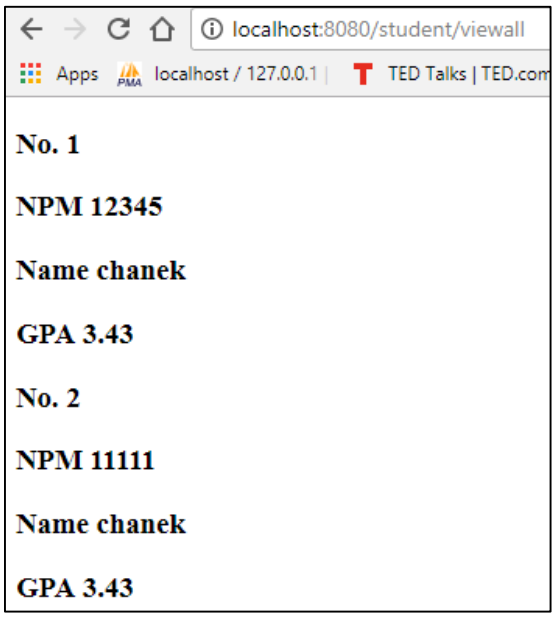


Coba tambahkan data Student lainnya dengan NPM yang berbeda, lalu buka localhost:8080/student/viewall,

**Pertanyaan 6**: Apakah semua data Student muncul?

**Jawaban**: semua data student muncul karena objek student yang berbentuk list dapat menampung banyak data di dalamnya dan halaman view all melakukan iterasi student untuk menampilkan seluruh data yang ada di dalam list tersebut.


C. Method selectStudent yang Anda implementasikan

```
public StudentModel selectStudent(String npm) {
        //Method ini menerima NPM mahasiswa dan mengembalikan object
Student
        //dengan NPM tersebut. Return null jika tidak ditemukan
        for (int i=0; i<studentList.size(); i++) {
            if (studentList.get(i).getNpm().equals(npm)) {
                return studentList.get(i);
            }
        }
        return null;
    }
```

D. Penjelasan fitur delete yang Anda buat pada bagian latihan

Latihan

1. Pada StudentController tambahkan sebuah method view Student dengan menggunakan Path Variable. Misalnya, Anda ingin memasukkan data seorang Student yang memiliki NPM 14769, untuk melihat data yang baru dimasukkan tersebut dapat mengakses halaman localhost:8080/student/view/14769.

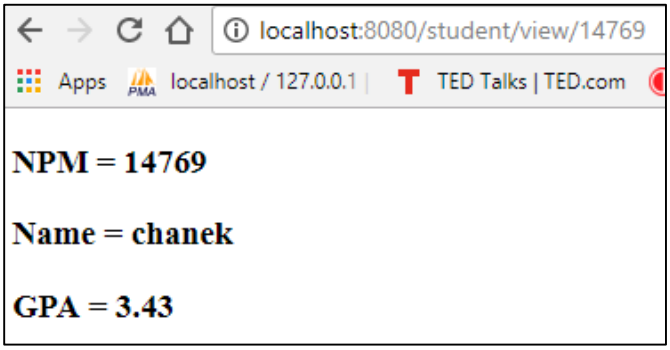
   Pertama buat terlebih dahulu method view student dengan menggunakan path variable. Pada path variable masukkan npm karena nantinya yang akan dipanggil di url adalah npm nya. Lalu buat kondisi jika url yg dimasukkan terdapat npm maka akan masuk ke kondisi lagi di dalamnya dimana jika npm sesuai dengan npm student yang telah di add sebelumnya maka akan mengakses halaman view sedangkan jika npm tidak sesuai maka akan ke halaman viewOther, begitu juga jika npm nya kosong.

   **StudentController.java**

```
//Latihan nomor 1
@RequestMapping(value = {"/student/view/", "/student/view/{npm}"})
public String greetingPath(@PathVariable String npm, Model model) {
    StudentModel student = studentService.selectStudent(npm);
    if(student != null) {
        if (student.getNpm().equals(npm)) {
            model.addAttribute("student", student);
            return "view";
        }else {
            model.addAttribute("npm", "apap");
            return "viewOther";
        }
    }else {
        return "viewOther";
    }
}
```

   Memasukkan data seorang Student yang memiliki NPM 14769


   Selanjutnya melihat data yang baru dimasukkan tersebut dapat mengakses halaman localhost:8080/student/view/14769.



Jika nomor NPM tidak diberikan atau tidak ditemukan kembalikan halaman error yang berisi informasi bahwa nomor NPM kosong atau tidak ditemukan.

**viewOther.html**

```
<!DOCTYPE html>
<html xmlns:th="http//www.thymeleaf.org">
<head>
<title>View Student by NPM</title>
</head>
<body>
    <h3>Nomor NPM kosong atau tidak ditemukan</h3>
</body>
</html>
```


2. Tambahkan fitur untuk melakukan delete Student berdasarkan NPM. Misalnya, setelah melakukan add Student pada soal nomor 1, cobalah untuk melakukan delete data tersebut dengan mengakses halaman localhost:8080/student/delete/14769. Tampilkan sebuah halaman yang memberikan informasi bahwa data tersebut telah berhasil dihapus.

   Untuk membuat fitur delete student berdasarkan npm maka kita tambahkan method delete di StudentController.java. Method tersebut memiliki path variable npm dan request mapping ke halaman delete. Lalu select npm pada objek student yang sebelumnya telah dilakukan add student melalui studentService. Npm yang dipanggil akan di simpan dalam objek student untuk kemudian dilakukan pengecekan apakah objek student memiliki data di dalamnya atau tidak, jika ada maka akan masuk ke kondisi selanjutnya untuk dilakukan pengecekan apakah npm yang dipanggil sama dengan npm yang telah di add sebelumnya. Jika sama maka akan dilakukukan delete student melalui studentService lalu return ke delete maka akan mengakses halaman delete, jika kondisi tidak sesuai (npm tidak sesuai atau kosong) maka akan return ke deleteOther.

   **StudentController.java**

```
//Latihan nomor 2
@RequestMapping(value = {"/student/delete/", "/student/delete/{npm}"})
public String delete(@PathVariable String npm, Model model) {
    StudentModel student = studentService.selectStudent(npm);
    if(student != null){
        if (student.getNpm().equals(npm)) {
            studentService.deleteStudent(npm);
            return "delete";
        }else {
            model.addAttribute("npm", "apap");
            return "deleteOther";
        }
    }else {
        return "deleteOther";
    }

}
```

Selanjutnya pada interface StudentService tambahkan seperti di kotak merah bawah ini untuk dapat dilakukan proses delete student.

**StudentService.java**

```
public interface StudentService {
    StudentModel selectStudent(String npm); //melihat data Student berdasarkan NPM-nya

    void deleteStudent(String npm);         //delete student (Latihan)

    List<StudentModel> selectAllStudents(); //melihat seluruh data Student

    void addStudent(StudentModel student);  //menambahkan Student baru.
}
```

Pada InMemoryStudentService.java buat method yang mengimplement dari class interface StudentService. Dimana method ini menerima npm mahasiswa dan meremove objek student dengan npm tersebut.

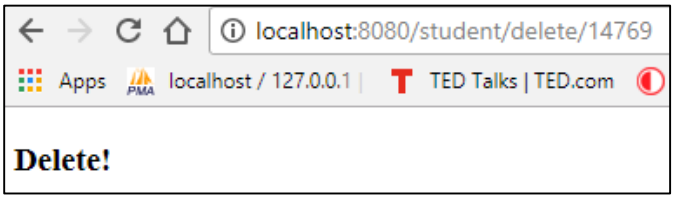
**InMemoryStudentService.java**

```
//Latihan
@Override
public void deleteStudent(String npm) {
    //Method ini menerima NPM mahasiswa dan mengembalikan object Student
    //dengan NPM tersebut.
    for (int i=0; i<studentList.size(); i++) {
        if (studentList.get(i).getNpm().equals(npm)) {
            studentList.remove(i);
        }
    }
}
```

delete.html merupakan halaman yang diakses jika kondisi pada studentController sesuai yaitu npm yang sesuai dengan objek student dan berhasil dilakukan delete.

**delete.html**

```
<!DOCTYPE html>
<html xmlns:th="http//www.thymeleaf.org">
<head>
<title>Delete Student by NPM</title>
</head>
<body>
    <h3>Delete!</h3>
</body>
</html>
```




Jika nomor NPM tidak diberikan atau tidak ditemukan kembalikan halaman error yang berisi informasi bahwa nomor NPM kosong atau tidak ditemukan dan proses delete dibatalkan.

Sedangkan deleteOther.html adalah halaman yang diakses jika tidak memenuhi kondisi yaitu npm yang diberikan kosong atau tidak sesuai dengan yang ada pada objek student.

**deleteOther.html**

```
<!DOCTYPE html>
<html xmlns:th="http//www.thymeleaf.org">
<head>
<title>Delete Student by NPM</title>
</head>
<body>
    <h3>Nomor NPM kosong atau tidak ditemukan</h3>
</body>
</html>
```