Ringkasan

Pada tutorial 3, dipelajari bagaimana membuat model dan service dengan konsep MVC dalam *project* Spring Boot. Dalam tutorial ini, diberikan latihan untuk membuat fitur CRUD (Create, Read, Update, Delete) dengan menggunakan data pada *list of objects* tertentu. Kemudian, pada tutorial ini juga digunakan materi pada tutorial sebelumnya mengenai path variable. Adapun inti dari tutorial kali ini adalah untuk memahami bagaimana konsep MVC bekerja.

Tutorial – Method Add

Jalankan program dan buka

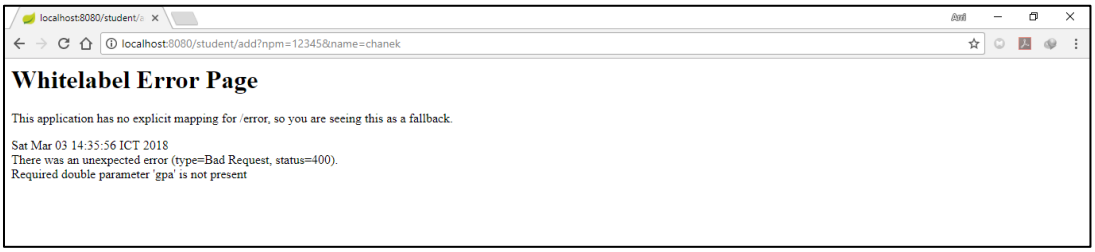
1. localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43
   Pertanyaan 1: apakah hasilnya? Jika *error*, tuliskan penjelasan Anda.

   Tidak ada error, berikut hasilnya (lihat Gambar 1).

Gambar 1. Hasil Pertanyaan Ke-1

2. localhost:8080/student/add?npm=12345&name=chanek
   Pertanyaan 2: apakah hasilnya? Jika *error*, tuliskan penjelasan Anda.

   Hasilnya *error* sebab parameter yang diperlukan oleh *method* “add” ada 3 yaitu “npm”, “name” & “gpa”. Semua parameter tersebut wajib diisi jika memanggil method “add”, seperti yang terlihat pada Gambar 2 di mana ada pesan *error* “Required double parameter 'gpa' is not present”.



Gambar 2. Hasil Pertanyaan Ke-2

Tutorial – Method View by NPM

Jalankan program dan buka

3. 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?

   Tidak ada *error*¸ hasilnya dapat dilihat pada Gambar 3.

Gambar 3. Hasil Pertanyaan Ke-3

4. localhost:8080/student/view?npm=12345
   Pertanyaan 4: apakah data Student tersebut muncul? Jika tidak, mengapa?

   Hasilnya *error*, karena *method* "view by npm" mengembalikan *object* yang bernilai *null* dan dalam halaman html yang dibuat tidak ada penanganan untuk menampilkan *object* yang bernilai *null* tersebut. *Object* tersebut bisa bernilai *null* sebab tidak ada data *student* dengan npm yang dicari pada *list of student* yang dipakai oleh program. Untuk hasilnya dari tutorial ini, bisa dilihat pada Gambar 4.

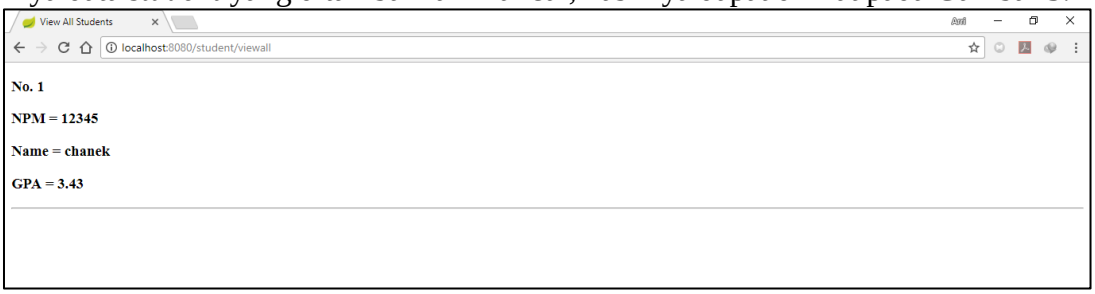
Gambar 4. Hasil Pertanyaan Ke-4

Tutorial – Method View All

Jalankan program dan buka

5. localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43 lalu buka localhost:8080/student/viewall,
   Pertanyaan 5: apakah data Student tersebut muncul?

   Iya data *student* yang ditambahkan muncul, hasilnya dapat dilihat pada Gambar 5.



Gambar 5. Hasil Pertanyaan Ke-5

6. Coba tambahkan data Student lainnya dengan NPM yang berbeda, lalu buka localhost:8080/student/viewall,
   Pertanyaan 6: Apakah semua data Student muncul?

   Semua data muncul, hasilnya dapat dilihat pada Gambar 6.

Gambar 6. Hasil Pertanyaan Ke-6

Penjelasan Method selectStudent

Method selectStudent yang telah Saya buat dapat dilihat pada Gambar 7.

```
@Override
public StudentModel selectStudent(String npm) {
    for (StudentModel student : studentList) {
        if (npm.equals(student.getNpm()))
            return student;
    }
    return null;
}
```

Gambar 7. Method selectStudent

Cara kerja dari *Method* selectStudent adalah sebagai berikut:

1. Melakukan pencarian secara satu per satu data student pada list studentList dengan menggunakan for looping
2. Kemudian, dilakukan validasi untuk mencocokan nilai dari parameter npm dengan nilai npm pada setiap object student
3. Jika, nilai parameter npm sama dengan nilai npm dari salah satu object student maka method langsung mengembalikan object student tersebut
4. Jika, setelah dilakukan for looping tidak ada ditemukan data student yang memiliki nilai npm sesuai dengan parameter maka method mengembalikan nilai null

Penjelasan fitur *delete*

Untuk membuat fitur *delete*, hal-hal yang perlu dilakukan adalah sebagai berikut.

1. Membuat method deletePath pada bagian controller (isi dari method tersebut dapat dilihat pada Gambar 8).

```
    public String viewAll(Model model) {
        List<StudentModel> students = studentService.selectAllStudents();
        model.addAttribute("students", students);
        return "viewall";
    }

    @RequestMapping(value = {"/student/view", "/student/view/{npm}"})
    public String viewPath(@PathVariable Optional<String> npm, Model model) {
        if (npm.isPresent()) {
            StudentModel student = studentService.selectStudent(npm.get());
            model.addAttribute("student", student);
        } else {
            model.addAttribute("student", null);
        }
        return "viewpath";
    }

    @RequestMapping(value = {"/student/delete", "/student/delete/{npm}"})
    public String deletePath(@PathVariable Optional<String> npm, Model model) {
        String msg = "";
        if (npm.isPresent()) {
            String result = studentService.deleteStudent(npm.get());
            if (result.equals("success")) {
                msg = "Student dengan npm \"" + npm.get() + "\" telah berhasil dihapus";
                model.addAttribute("result", msg);
                return "successpath";
            } else {
                msg = "NPM kosong atau tidak ditemukan dan proses delete dibatalkan";
                model.addAttribute("result", msg);
                return "errorpath";
            }
        } else {
            msg = "NPM kosong atau tidak ditemukan dan proses delete dibatalkan";
            model.addAttribute("result", msg);
            return "errorpath";
        }
    }
}
```

Gambar 8. Method deletePath (Controller)

2. Membuat interface deleteStudent pada bagian service (lihat Gambar 9).

```
package com.example.tutorial3.service;

import java.util.List;

public interface StudentService {
    StudentModel selectStudent(String npm);

    List<StudentModel> selectAllStudents();

    void addStudent(StudentModel student);

    String deleteStudent(String string);
}
```

Gambar 9. StudentService (Interface)

3. Membuat method deleteStudent pada bagian model (isi dari method dapat dilihat pada Gambar 10).

```
import java.util.List;

public abstract class InMemoryStudentService implements StudentService {
    private static List<StudentModel> studentList = new ArrayList<StudentModel>();

    @Override
    public StudentModel selectStudent(String npm) {
        for (StudentModel student : studentList) {
            if (npm.equals(student.getNpm()))
                return student;
        }
        return null;
    }
    @Override
    public List<StudentModel> selectAllStudents() {
        return studentList;
    }

    @Override
    public void addStudent(StudentModel student) {
        studentList.add(student);
    }

    @Override
    public String deleteStudent(String npm) {
        String result = "";
        for (StudentModel student : studentList) {
            if (npm.equals(student.getNpm())) {
                studentList.remove(student);
                result = "success";
                break;
            }
        }
        return result;
    }
}
```

Gambar 10. Method deleteStudent (Model)

4. Membuat file successpath.html untuk menampilkan informasi ketika fitur *delete* berhasil dilakukan (lihat Gambar 11).

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

Gambar 11. successpath.html (view)

5. Membuat file successpath.html untuk menampilkan informasi ketika fitur *delete* berhasil dilakukan (lihat Gambar 12).

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Error Page</title>
    </head>
    <body>
        <h3 th:text="${result}">Error Messages</h3>
    </body>
</html>
```

Gambar 12. errorpath.html (view)

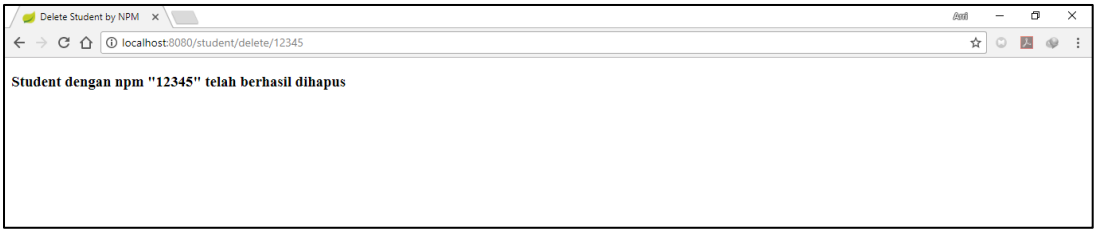
Cara kerja dari fitur ini adalah sebagai berikut.

1. Jika, parameter npm diisi dan data student berhasil dihapus maka akan menampilkan halaman successpath.html
2. Jika, parameter npm diisi dan data student tidak berhasil dihapus makan akan menampilkan halaman errorpath.html

3. Jika, paremeter npm tidak diisi maka akan menampilkan halaman errorpath.html

Cara kerja dari method deleteStudent hampir sama dengan cara kerja method selectStudent, dimana dalam method deleteStudent jika data student sudah ditemukan maka akan dilakukan penghapusan data tersebut dari list.

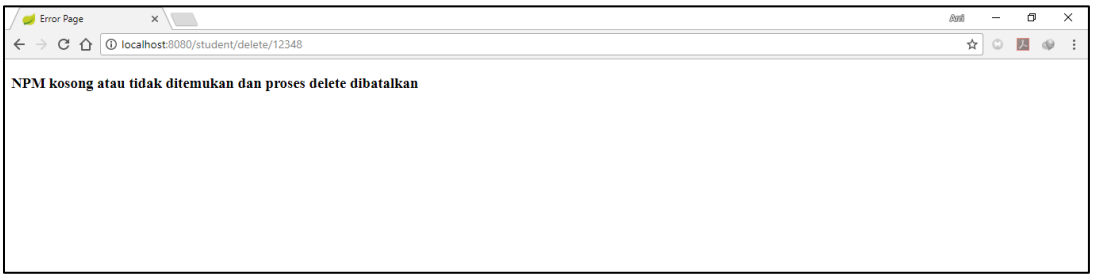
Adapun hasil dari fitur *delete* adalah sebagai berikut (dengan menggunakan data seperti pada Gambar 6). Hasil fitur delete yang berhasil dilakukan dapat dilihat pada Gambar 13 dan kondisi data setelah dilakukan penggunaan fitur ini dapat dilihat pada Gambar 14.



Gambar 13. Hasil dari fitur delete yang berhasil

Gambar 14. Kondisi Data setelah dilakukan delete

Adapun hasil error atau fitur delete yang tidak berhasil dapat dilihat pada Gambar 15.



Gambar 15. Hasil dari fitur delete yang gagal