Menggunakan Model dan Service pada Project Spring Boot

Versi: 2 **(20 September 2017)** **perubahan ditandai dengan tinta merah*

Outline:

- Membuat model dengan konsep MVC dalam *project* Spring Boot
- Membuat *service* dengan fungsi *create* dan *read* data melalui konsep MVC

Requirements:

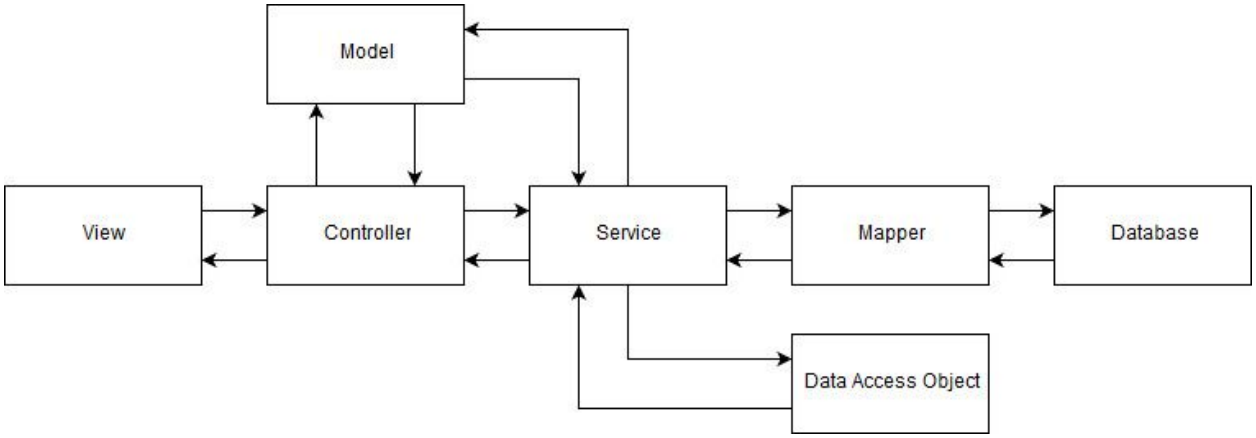
Sebelum memulai tutorial ini pastikan komputer Anda sudah terinstall JDK, Eclipse, Maven, dan STS.

Pendahuluan

Pada tutorial ini, Anda akan melanjutkan implementasi konsep *model-view-controller* (MVC) menggunakan Spring Boot. Komponen yang menjadi fokus dalam tutorial ini adalah *model* dan *service. Model* merupakan 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.

Pada tutorial sebelumnya, Anda juga telah menjumpai komponen *view* dan *controller* yang menggunakan teknologi Thymeleaf dan Spring Boot. Tutorial ini akan membahas kedua komponen tersebut beserta kaitannya dengan komponen *model* dan *service*. Anda akan memperdalam pengetahuan mengenai komponen *view* dan *controller* serta penggunaan komponen *model* dan *service* melalui pembuatan fungsi *create* dan *read* data (*method* '*add*' dan '*view*') yang disajikan dalam tutorial ini.



* tanda panah merepresentasikan alur interaksi dan data antar komponen

Gambar 1. Kaitan antar komponen MVC di Spring Boot

Tutorial

Perhatian: Jangan copy-paste kode pada pdf ini ke eclipse/IDE Anda. Kode Anda tidak akan berjalan atau dikenali oleh compiler karena karakter tidak memiliki encoding yang sama.

Pendahuluan

1. Pada Eclipse klik File > New > Other > **Spring Boot** > Spring Starter Project, tekan Next
2. Beri nama *project*, *description*, *package,* dll. **Beri nama package com.example.tutorial3.**
3. Pilih **Web**, **Thymeleaf**, dan **DevTools** pada bagian *dependency*

Membuat *Class Model*

Salah satu komponen dalam MVC adalah *model*. Anda akan membuat *model* berupa *class* Student yang berisi variabel name (String), npm (String), dan gpa (double).

1. Klik kanan pada Project > New > Package. Buatlah package **com.example.tutorial3.model**
2. Buatlah class **StudentModel** dengan spesifikasi seperti di atas pada package tersebut.

```
package com.example.tutorial3.model;

public class StudentModel {
    private String name;
    private String npm;
    private double gpa;

    public StudentModel(String npm, String name, double gpa){
        this.name = name;
        this.npm = npm;
        this.gpa = gpa;

    }
}
//Jangan lupa untuk menambahkan Setter Getter untuk setiap property
```

3. Tambahkan *method constructor*, *setter*, dan *getter*.

Membuat *Service*

1. Klik kanan pada Project > New > Package. Buatlah package **com.example.tutorial3.service**
2. Buatlah *interface* **StudentService.java** pada *package* tersebut dengan isi sebagai berikut:

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

import java.util.List;

import com.example.tutorial3.model.StudentModel;

public interface StudentService {
    StudentModel selectStudent(String npm);

    List<StudentModel> selectAllStudents();

    void addStudent(StudentModel student);
}
```

Service ini merupakan *interface* yang mendefinisikan *method-method* apa saja yang dapat dilakukan untuk memanipulasi kelas Student. Dari *method-method* di atas, Anda dapat melihat data Student berdasarkan NPM-nya, melihat seluruh data Student, dan menambahkan Student baru.

3. Pada package yang sama, buat class InMemoryStudentService yang meng-*implements* StudentService dengan isi sebagai berikut:

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

import java.util.ArrayList;
import java.util.List;
import com.example.tutorial3.model.StudentModel;

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

    @Override
    public StudentModel selectStudent(String npm) {

        // Implement
        return null;
    }

    @Override
    public List<StudentModel> selectAllStudents() {
            return studentList;
    }

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

    }
}
```

Pada kelas tersebut Anda mengimplementasikan StudentService dengan **InMemoryStudentService**. Pada tutorial kali ini Anda akan menggunakan static List (studentList) untuk menyimpan seluruh data mahasiswa.

Pada tutorial selanjutnya, Anda akan belajar untuk menyimpan data Student pada *database*. Jika Anda ingin menggunakan *database*, apakah perlu mengubah class-class yang sudah ada? Tidak, Anda cukup membuat kelas baru yang meng-*implements* StudentService dan mengimplementasikan bagaimana koneksi ke *database*.

4. Implementasikan *method* selectStudent! *Method* ini menerima NPM mahasiswa dan mengembalikan *object* Student dengan NPM tersebut. Return null jika tidak ditemukan.

(**Silahkan tulis method selectStudent di dalam writeup setelah berhasil diimplementasi**)

Membuat Controller dan Fungsi Add

1. Klik kanan pada Project > New > Package. Buatlah *package* **com.example.tutorial3.controller**
2. Tambahkan method *add* yang menerima parameter name, npm, dan gpa dengan menggunakan request method GET. Buatlah *class* **StudentController** dengan isi sebagai berikut:

```
package com.example.tutorial3.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.example.tutorial3.service.InMemoryStudentService;
import com.example.tutorial3.service.StudentService;
import com.example.tutorial3.model.StudentModel;

@Controller
public class StudentController {
    private final StudentService studentService;

    public StudentController() {
        studentService = new InMemoryStudentService();
    }

    @RequestMapping("/student/add")
    public String add(@RequestParam(value = "npm", required = true) String npm,
                @RequestParam(value = "name", required = true) String name,
                @RequestParam(value = "gpa", required = true) double gpa) {
        StudentModel student = new StudentModel(npm, name, gpa);
        studentService.addStudent(student);
        return "add";
    }
}
```

3. Selanjutnya pada *directory* resources/templates tambahkan add.html dengan isi sebagai berikut:

```
<html>
<head>
<title>Add</title>
</head>
<body>
    <h2>Data berhasil ditambahkan</h2>
</body>
</html>
```

4. Jalankan program dan buka

localhost:8080/student/add?npm=12345&name=chanek&gpa=3.43

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

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

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

Method View by NPM

Sejauh ini, Anda sudah menambahkan data Student ke ArrayList yang ada pada *data access object* (DAO). Namun, Anda tidak dapat melihat apakah data benar-benar tersimpan. Selanjutnya Anda akan mengimplementasikan method view by NPM.

1. Pada class StudentController tambahkan *method* berikut:

```
    @RequestMapping("/student/view")
      public String view(Model model, @RequestParam(value = "npm", required = true)
String npm) {
        StudentModel student = studentService.selectStudent(npm);
        model.addAttribute("student", student);
        return "view";
    }
```

2. Selanjutnya pada *directory* resources/templates tambahkan view.html dengan isi sebagai berikut:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>View Student by NPM</title>
</head>
<body>
    <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
    <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
    <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
</body>
</html>
```

3.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?

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

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

5. Coba tambahkan data Student lainnya dengan NPM yang berbeda.

Method View All

1. Pada class StudentController tambahkan *method* berikut

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

2. Selanjutnya pada *directory* resources/templates tambahkan viewall.html dengan isi sebagai berikut:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>View All Students</title>
</head>
<body>
<div th:each="student,iterationStatus: ${students}">
<h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
<h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
<h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
<h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
<hr/>
</div>
</body>
</html>
```

3. 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?

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

   Pertanyaan 6: Apakah semua data Student muncul?

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.**

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

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.

   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.

Catatan: Jelaskanlah tahap-tahap yang Anda lakukan dalam menyelesaikan fitur latihan diatas pada *file write-up*.

Deliverables

Pada tutorial ini, ada beberapa *deliverables* yang Anda perlu kerjakan dan kumpulkan. *Deliverables* tersebut adalah sebagai berikut:

1. *Folder project* Anda yang berisi implementasi dari bagian 'Latihan' yang terdapat dalam tutorial ini. Tidak masalah jika dalam project tersebut juga ada hasil tutorial Anda.
2. *File write-up* berisi penjelasan apa yang Anda pelajari pada tutorial kali ini, yaitu:

   a. Ringkasan dari materi yang Anda telah pelajari pada tutorial kali ini

   b. Hasil jawaban dari setiap poin pada bagian tutorial (dapat didukung dengan *screenshot*)
   c. Method selectStudent yang Anda implementasikan
   d. Penjelasan fitur *delete* yang Anda buat pada bagian latihan.

   Format txt atau pdf. Masukkan dalam *folder project* dan pastikan *file write-up* juga di-*push* ke repositori GitHub.

Pengumpulan

Buat sebuah **project** di organisasi GitHub **/apap-2017** dengan format nama tutorial3_[NPM]. Contoh: **tutorial3_1501234567**. *Push* seluruh isi *folder project* Anda, termasuk *file write-up*, ke repositori *project* tersebut. Anda tidak perlu membuat *branch* baru. Cukup *push* ke *branch master* saja.

Perhatikan dalam membuat **public** project tersebut. Pastikan Anda berada di dalam organisasi GitHub **/apap-2017** terlebih dahulu.

Commit* yang akan dinilai adalah *commit* terakhir yang di *push* ke repositori sebelum *deadline. Waktu yang akan dijadikan patokan adalah waktu *commit* di *server* GitHub. Pastikan Anda tidak *push commit* lagi setelah *deadline* jika tidak ingin terkena penalti.

Deadline

23 September 2017, 23:59:59

Penalti

Penalti berlaku untuk kasus:

- Keterlambatan
  Nilai total akan ditambahkan -10 poin untuk setiap 1 jam keterlambatan