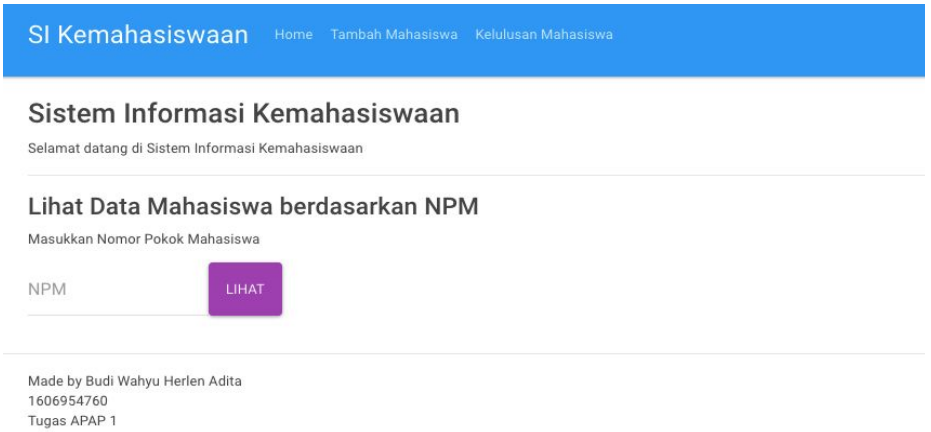
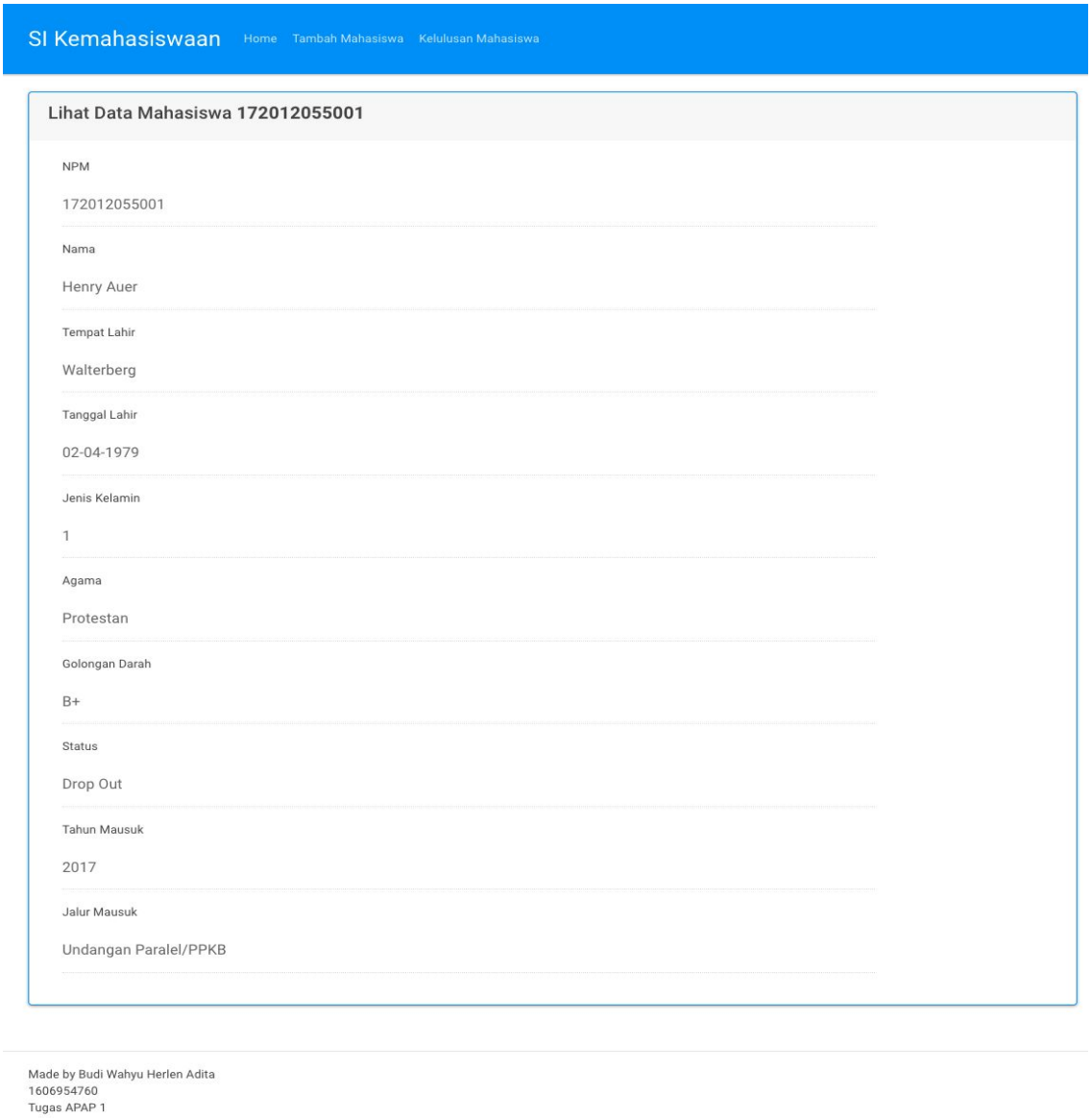
Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

1. Fitur Aplikasi
    a. Tampilkan Data Mahasiswa Berdasarkan NPM
    - Berikut tampilan **Home**



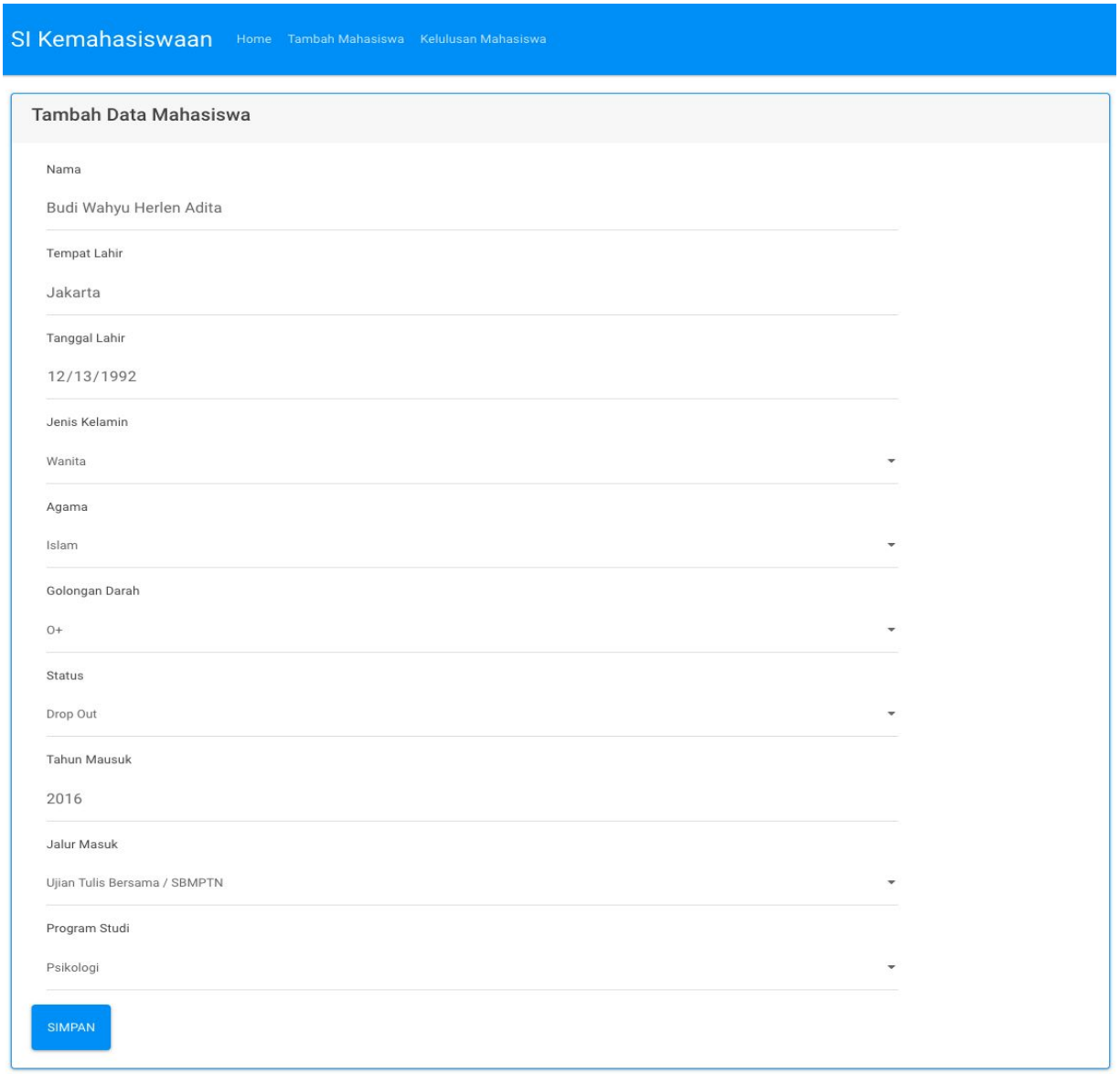
    - Berikut **hasil pencarian** berdsarkan **npm** mahasiswa



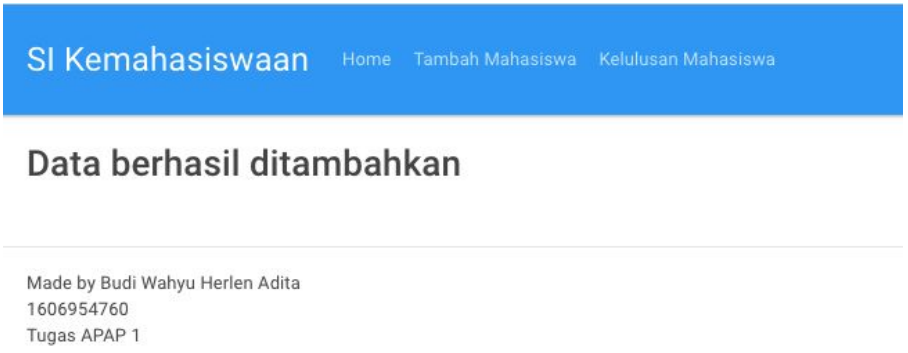
Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

b. Menambahkan Mahasiswa Baru di Suatu Program Studi

- Berikut tampilan menambahkan data mahasiswa. Untuk beberapa field seperti tanggal lahir, jenis kelamin, golongan darah, status, jalur masuk dan program studi menggunakan selection.

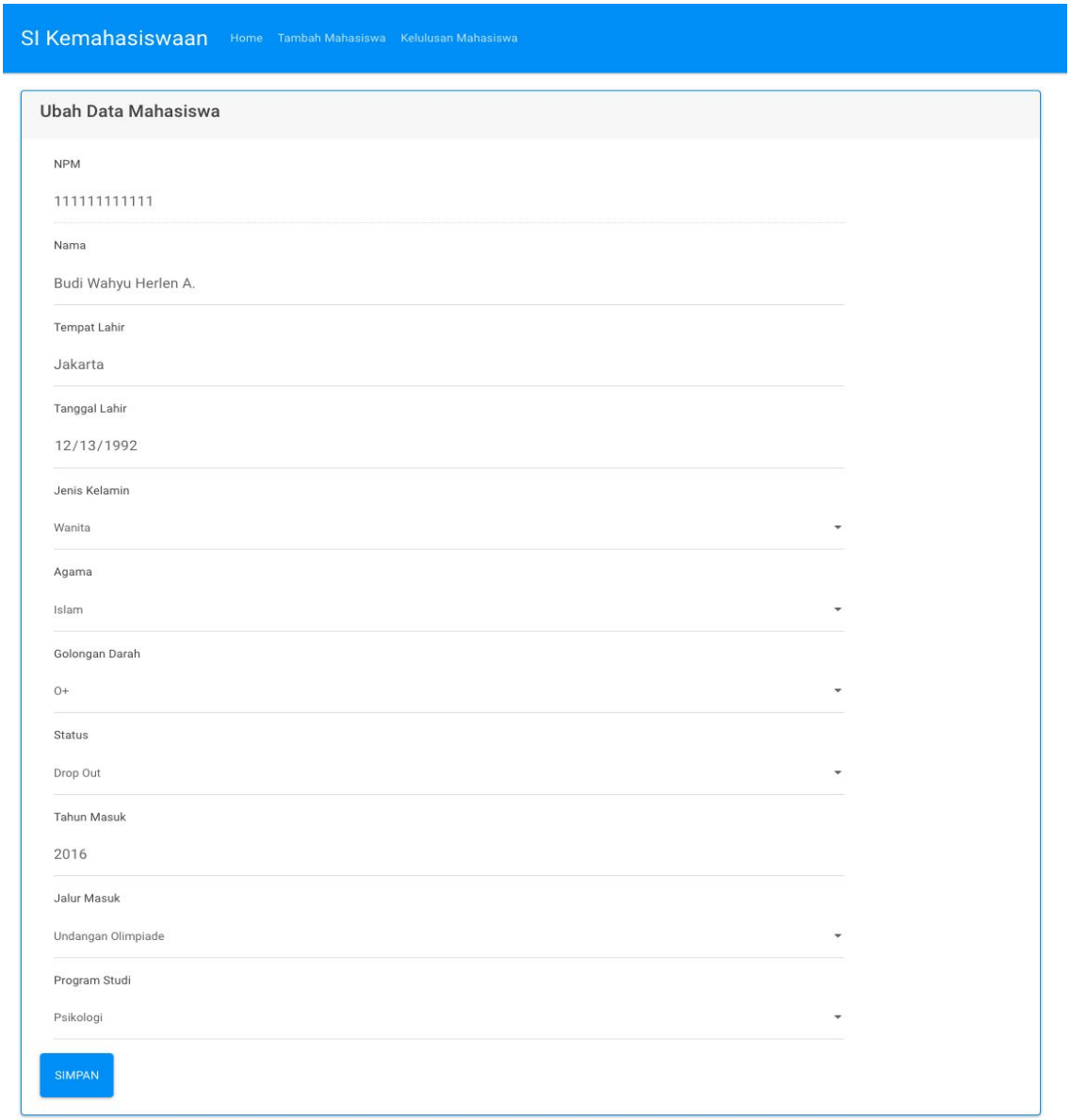


- Berikut **tampilan** jika data berhasil ditambahkan



Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

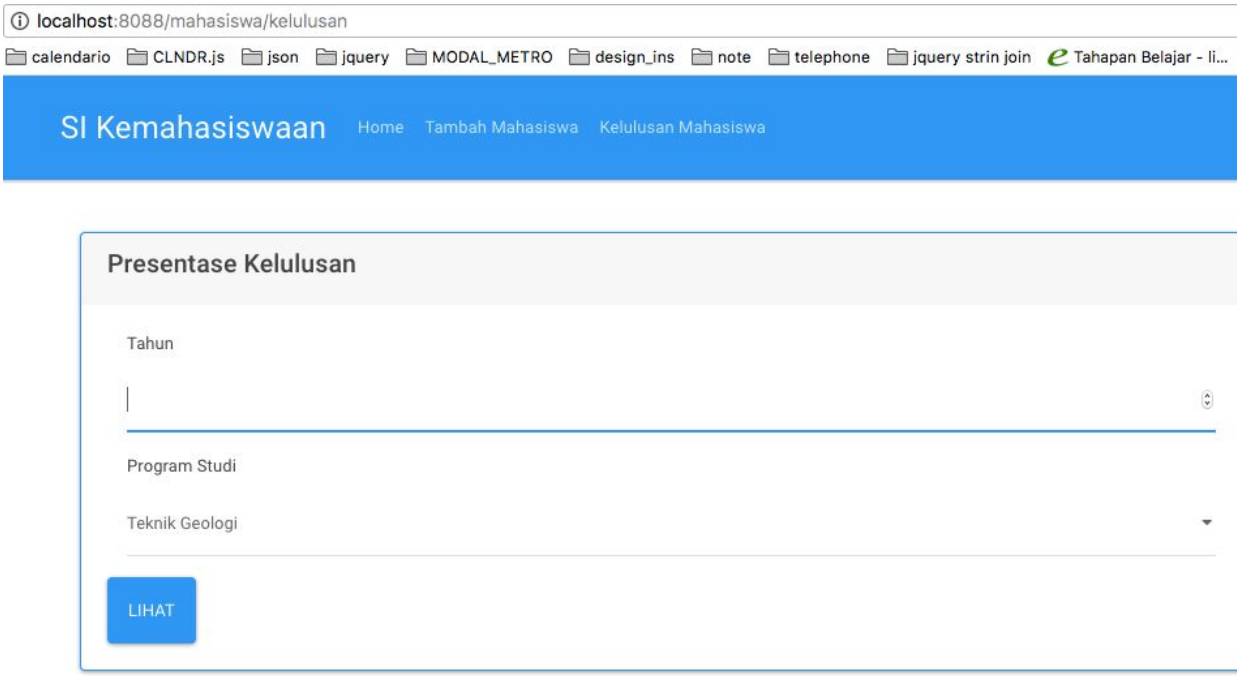
c. Mengubah Data Mahasiswa



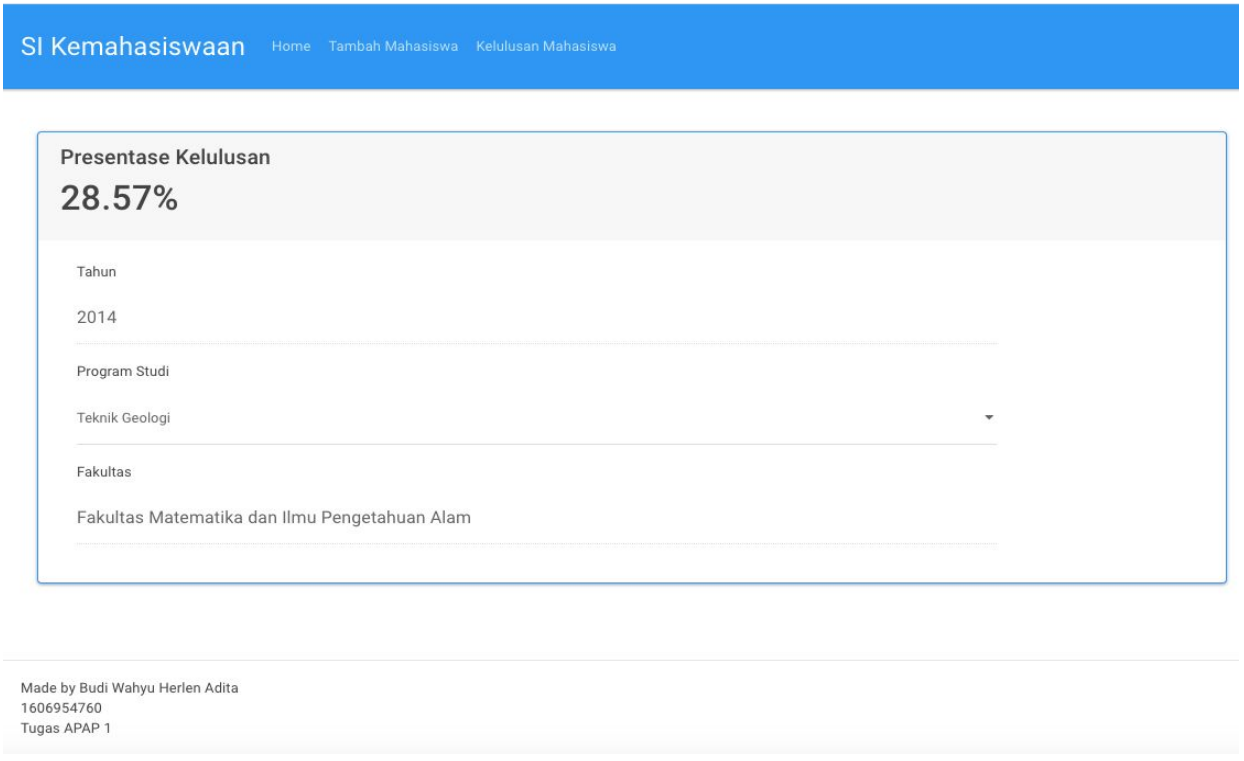
d. Menampilkan Presentase Kelulusan

- **Input** data presentase kelulusan

Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5



- **Output** presentase **kelulusan**



Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

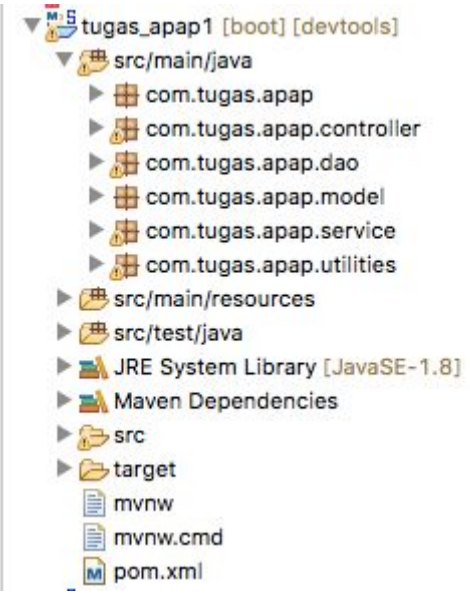
- Saya menggunakan bantuan 2 method dari **MahasiswaService**, untuk megkalkulasi persentase **kelulusan** berdasarkan **program studi** dan **tahun**.

```
@RequestMapping(value = "/mahasiswa/kelulusan", method = RequestMethod.GET)
public String graduateCalculate (
    Model model,
     @RequestParam(value = "tahun", required = false) String tahun,
     @RequestParam(value = "prodi", required = false) String id_prodi
    ) {
    if(tahun != null || id_prodi != null) {
        double totalMahasiswaLulus = mahasiswaDAO.
                selecTotalMahasiswaByProgStudiIdAndStatus(Integer.valueOf(id_prodi), tahun, "Lulus");
        double totalMahasiswa = mahasiswaDAO.
                selecTotalMahasiswaByProgStudiId(Integer.valueOf(id_prodi), tahun);

        double resultPercent = (totalMahasiswaLulus/totalMahasiswa)*100;
        DecimalFormat formatResult = new DecimalFormat("##.00");

        model.addAttribute("result", formatResult.format(resultPercent));
        model.addAttribute("tahun", tahun);
        model.addAttribute("prodi", id_prodi);
        model.addAttribute("fakultas",
                psDAO.selectFakultasById(
                        psDAO.selectProgramStudiById(Integer.valueOf(id_prodi)).getId_fakultas()
                )
                );

    }
    model.addAttribute ("prodis", psDAO.selectAllProgramStudis());

    return "form-graduated";
}
```

e. Tampilkan Data Mahasiswa Berdasarkan Universitass, Fakultas, dan Program Studi Tertentu

Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

2. Penerapan Konsep MVC dan Layering
   Berikut tampilan project :
   Layering berpusat pada 3 hal : **Presentation** Layer, **Domain** Layer dan **Data** Layer



   a. Logical Pattern

Digunakan pattern **DAO** (Data Access Object) dan **Service Layer**.

```
package com.tugas.apap.service;

import java.util.List;

public interface MahasiswaService {
    MahasiswaModel selectMahasiswa (String npm);

    List<MahasiswaModel> selectAllMahasiswas ();

    boolean addMahasiswa (MahasiswaModel student);

    boolean updateMahasiswa (MahasiswaModel mahasiswa);

    int selectMahasiswaUrutan(String tahun_masuk, int id_prodi, String jalur_masuk);

    int selecTotalMahasiswaByProgStudiId(int id_prodi, String tahun_masuk);

    int selecTotalMahasiswaByProgStudiIdAndStatus(int id_prodi, String tahun_masuk, String status);


}
```

Pada interface ini, hampir semua yang dapat dilakukan di **DataMapper** dapat diakomodasi di interface ini, selanjutnya yang menginplementasi **interface service** tersebut di representasikan per nama tabel.

   b. Data Access Pattern

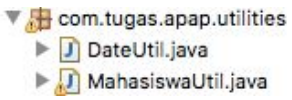
Pattern yang dipilih adalah **Data Mapper.** Dipilih **MahasiswaMapper** sebagai contoh.

Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

```
import java.util.List;

@Mapper
public interface MahasiswaMapper {
    @Select("select * from mahasiswa where npm = #{npm} ")
    MahasiswaModel selectMahasiswa (@Param("npm") String npm);

    @Select("select * from mahasiswa" )
    List<MahasiswaModel> selectAllMahasiswas ();

    @Insert("INSERT INTO mahasiswa ("

            + "npm, nama, tempat_lahir, tanggal_lahir,"
            + "jenis_kelamin, agama, golongan_darah, "
            + "status, tahun_masuk, "
            + "jalur_masuk, id_prodi) "

            + "VALUES (#{npm}, #{nama}, #{tempat_lahir}, #{tanggal_lahir}, "
            + "#{jenis_kelamin}, #{agama}, #{golongan_darah}, "
            + "#{status}, #{tahun_masuk}, #{jalur_masuk}, #{id_prodi})")
    boolean addMahasiswa (MahasiswaModel mahasiswa);

    @Update("UPDATE mahasiswa "
            + "SET "

            + "nama = #{nama},"
            + "tempat_lahir = #{tempat_lahir},"
            + "tanggal_lahir = #{tanggal_lahir},"
            + "jenis_kelamin = #{jenis_kelamin},"
            + "agama = #{agama},"
            + "golongan_darah = #{golongan_darah},"
            + "status = #{status},"
            + "tahun_masuk = #{tahun_masuk},"
```

Terlihat Mapper berisi pemanggilan query yang nantinya dapat dipakai oleh **service**.

Kemudian saya tambahkan juga package **utilities**, yang gunanya untuk menyimpan logic seputar parsing dan generate kode. Tujuannya, agar controller lebih simple dan bersih sehingga tidak terbebani oleh logic yang kompleks dan banyak.



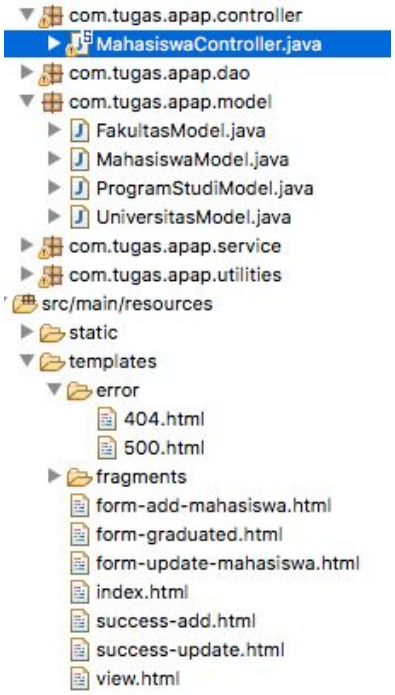
Berikut contoh kode **parsing date** yang saya kelompokkan ke dalam utilitis.

```
package com.tugas.apap.utilities;

import java.text.DateFormat;

@Slf4j
public class DateUtil {
    public static String convertDate(String dateStr) {
        String dateResult = "";
        try {
            DateFormat formatter = new SimpleDateFormat("yyyy-mm-dd");
            DateFormat formatterDateIndo = new SimpleDateFormat("dd-mm-yyyy");

            dateResult = formatterDateIndo.format(formatter.parse(dateStr));
        } catch (Exception e) {
            log.error("Error : " + e + "\nconvertDate " + dateStr);
        }
        return dateResult;

    }

}
```

Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5

c. MVC

Konsep MVC (Model View Controller) juga saya selipkan di project ini.



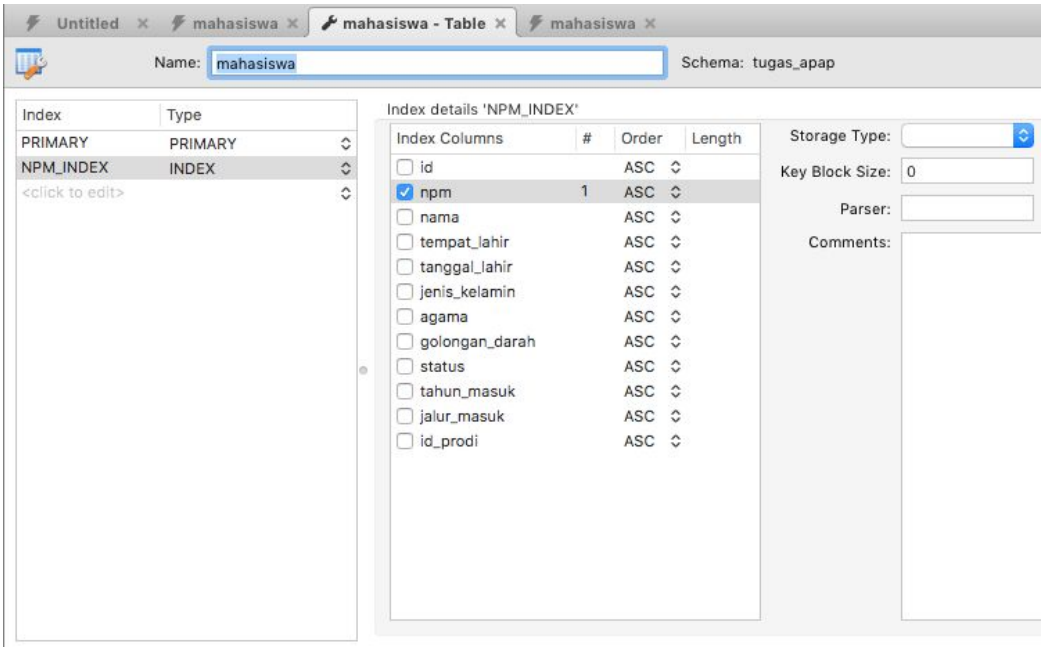
- **Model** merepresentasikan database seperti nama, field serta domainnya.
- Pada bagian **View** saya memanfaatkan template engine **thymeleaf.**
- Sedangkan Pada **Controller**, yang akan **menghubungkan model** dan **view** termasuk controller itu sendiri.

Lalu bagaimana cara controller mengakses database? karena Model disini lebih sebagai constructor.

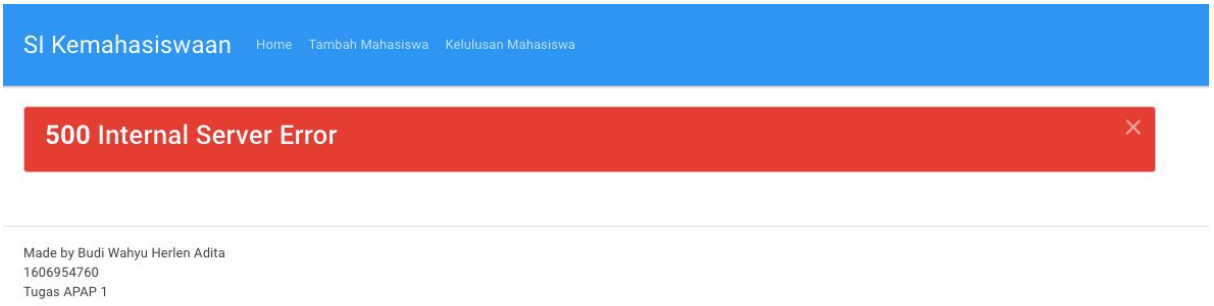
Disinilah fungsi **service**. Dengan adanya service, **organisasi domain logic** jadi fleksible, bahkan bisa didetailkan, tergantung dari kebutuhannya (apa saja fungsionalitasnya).

3. Optimasi Database dan Stress Testing
   a. Optimasi
   - Penerapan indexing pada npm

Budi Wahyu Herlen Adita
1606954760
Tugas APAP 5



4. Tambahan
    a. Menambahkan Error Page
    - Error 500



- Error 404