

MODUL OBJECT ORIENTED PROGRAMMING 2

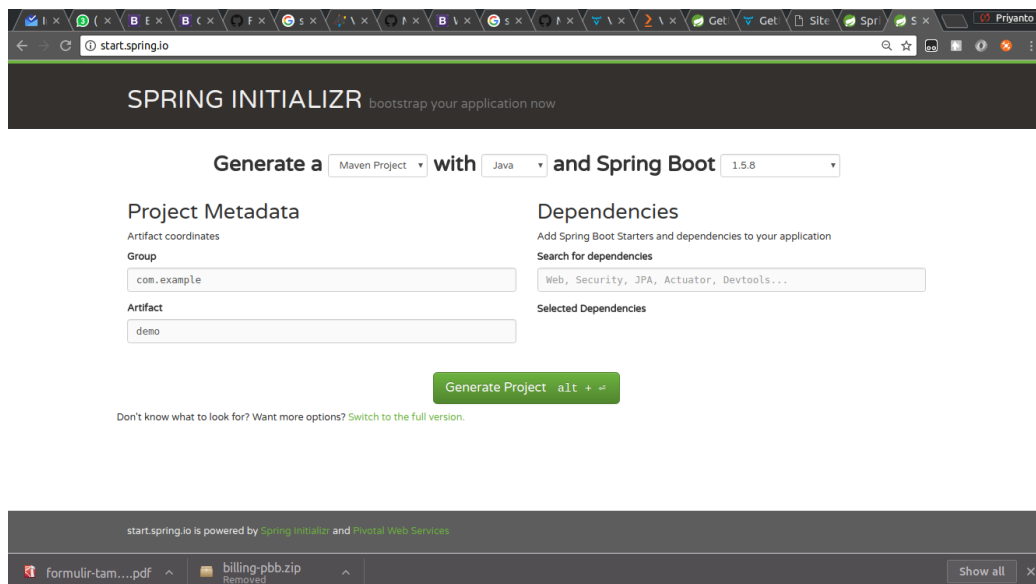
tamami

Bab 1

CRUD WITH SPRING AND THYMELEAF

Spring adalah sebuah *framework* yang memudahkan kita membangun sebuah aplikasi baik itu aplikasi berbasis desktop, web, ataupun *mobile*.

Cara paling mudah untuk memulai menggunakan framework Spring adalah dengan mengunjungi situs <https://spring.io>, dan saat akan memulai sebuah *project* kita dapat menggunakan rangka *project* yang disediakan oleh Spring di halaman <http://start.spring.io>, tampilan halaman web-nya akan terlihat seperti pada gambar 1.1 :



Gambar 1.1: Tampilan start.spring.io

Yang perlu diisi dari form yang ada pada website `start.spring.io` adalah sebagai berikut :

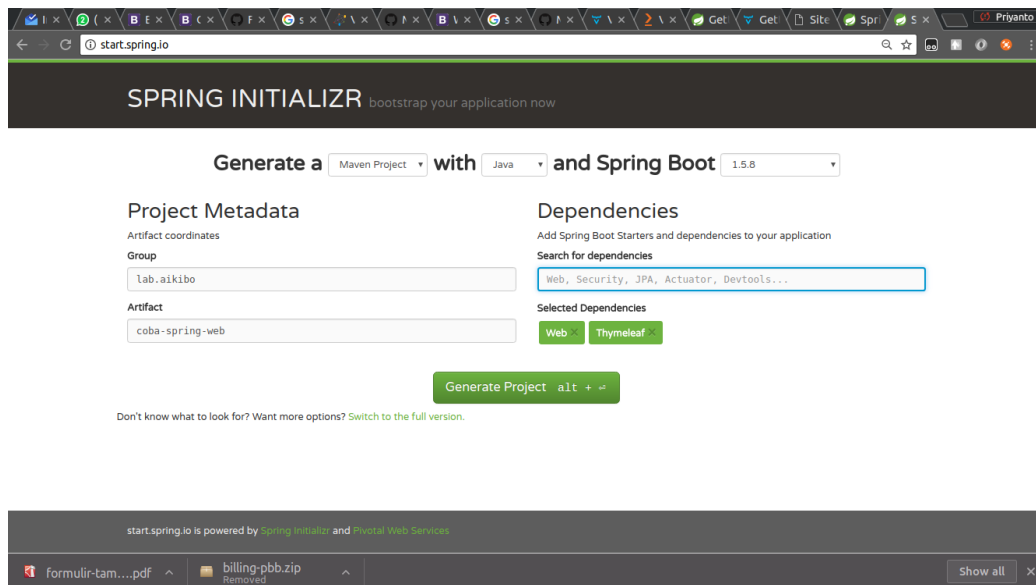
- **Group** yang diisi dengan nama atau identitas organisasi, instansi, atau tim yang mengembangkan aplikasi. Pengisian disini biasanya akan dijadikan nama paket dalam kandar (*folder*) kode sumber (*source code*).
- **Artifact** yang diisi dengan nama aplikasinya. Sebaiknya jangan menggunakan titik disini, isikan hanya dengan huruf dan angka.
- **Dependencies** yang diisi dengan pustaka-pustaka (*library*) yang nantinya akan digunakan dalam *project*.

Setelah itu klik tombol **Generate Project** atau tekan tombol **Ctrl + Enter** di tombol *keyboard*. Nanti kita akan mendapat sebuah *file* terkompres dengan ekstensi `zip`, ekstraklah *file* ini untuk selanjutnya kita buka dalam IDE Netbeans.

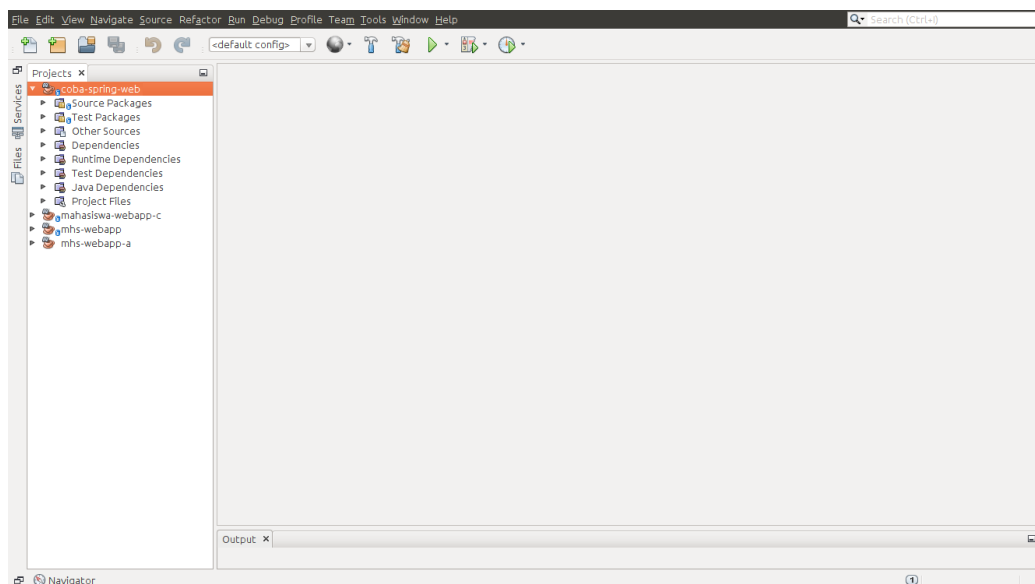
1.1 Uji Coba Pertama

Untuk pertama kalinya, kita akan mencoba bagaimana cara kerja Spring Framework dalam membangun sebuah aplikasi web dengan konsep MVC (*Model-View-Controller*). Berikut langkahnya :

1. Buat kerangka *project* dari laman `start.spring.io` dengan isian pustakanya adalah **Web** dan **Thymeleaf** seperti gambar 1.2 :

Gambar 1.2: Membuat Rangka *Project*

- Setelah *file* diekstrak, bukalah dengan Netbeans sehingga terlihat struktur *folder* seperti pada gambar 1.3 :

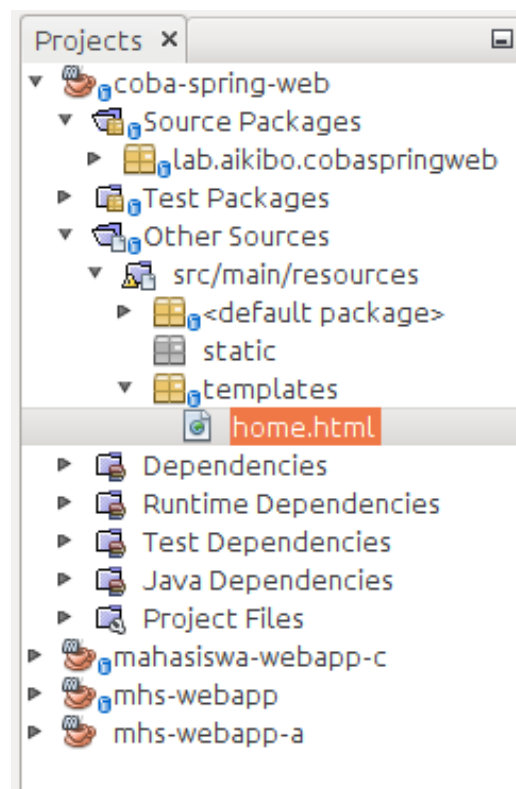
Gambar 1.3: Struktur Direktori *Project*

- Buatlah sebuah *file* `html` sebagai bagian dari *View* dalam konsep MVC

di bagian **Other Sources**, di bagian **templates** berikan nama bebas, misal kita berikan nama **home.html** dengan isian sebagai berikut :

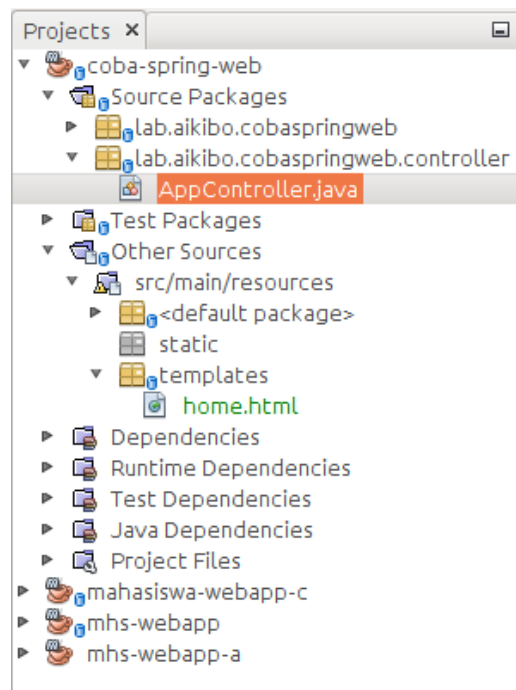
```
1 <html>
2   <head>
3     <title>Aplikasiku</title>
4   </head>
5   <body>
6     <h1>Selamat Datang</h1>
7   </body>
8 </html>
9
```

File **home.html** di Netbeans akan disimpan seperti gambar 1.4 :



Gambar 1.4: Posisi *File home.html*

4. Berikutnya kita buat sebuah *controller* yang menghubungkan antara aplikasi *backend* dengan *frontend*, biasanya untuk *controller* ini dibuatkan dalam sebuah paket tersendiri dalam **Source Packages** dengan nama **controller**. Buatlah sebuah *file controller* disana, misalkan kita beri nama **AppController** seperti terlihat pada gambar 1.5 :

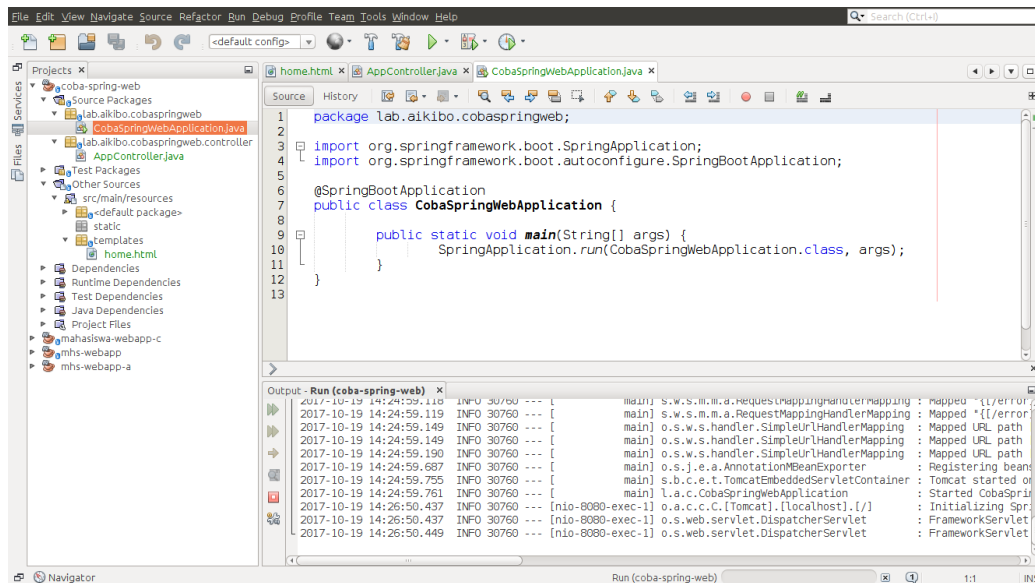
Gambar 1.5: Posisi *File* `AppController.java`

Isi kode dari *file* `AppController.java` adalah sebagai berikut :

```
1 package lab.aikibo.cobaspringweb.controller ;
2
3 import org.springframework.stereotype.Controller ;
4 import org.springframework.web.bind.annotation .
5     RequestMapping ;
6
7 /**
8  * @author tamami <tamami.oka@gmail.com>
9  */
10 @Controller
11 public class AppController {
12
13     @RequestMapping("/home")
14     public void index() {}
15
16 }
17
```

Perhatikan bahwa parameter dari `@RequestMapping` harus sama dengan nama *file* `html` yang kita buat sebelumnya.

5. Jalankan aplikasinya dengan tombol F6 di *keyboard*, kemudian pilih saja Main Class yang sudah disediakan, atau jalankan *main file* yang dibentuk oleh Spring, carilah sebuah kelas yang didalamnya ada anotasi `@SpringBootApplication` dan memiliki sebuah *method* `main` seperti gambar 1.6 :

Gambar 1.6: *Main Class*

Klik kanan lalu pilih Run File atau tekan tombol Shift + F6 di *keyboard*.

6. Bukalah *browser*, lalu akses ke alamat `localhost:8080`, sehingga seharusnya akan muncul tampilan seperti gambar 1.7 :



Gambar 1.7: Halaman Yang tampil Pertama Kali

Sampai sini, kita telah melihat bahwa konsep MVC diimplementasikan dalam Spring Web secara utuh dan terlihat strukturnya, selanjutnya kita akan coba menampilkan isi dari basis data ke jendela web aplikasi kita.

1.2 Tampilkan Data

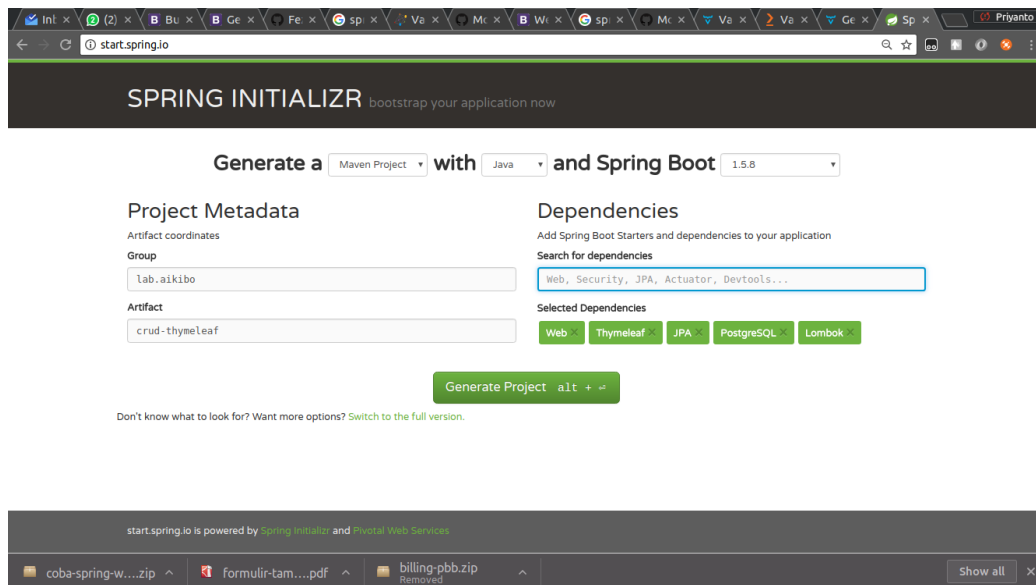
Sebelum kita tampilkan datanya, datanya harus disiapkan terlebih dahulu. Struktur data yang perlu dibentuk adalah sebagai berikut :

Kolom	Tipe	Keterangan
nim	varchar(7)	not null primary key
nama	varchar(30)	
jurusan	varchar(50)	

Berikan data *sample* untuk kita lakukan uji coba awal, yaitu menampilkan data di halaman aplikasi *web* yang kita bangun.

Data yang disiapkan untuk modul ini kebetulan menggunakan basis data PostgreSQL, yang nantinya akan ada perbedaan konfigurasi bila menggunakan basis data yang lain, namun jangan khawatir karena informasi untuk konfigurasi basis data yang lain seperti MySQL cukup mudah ditemukan di internet.

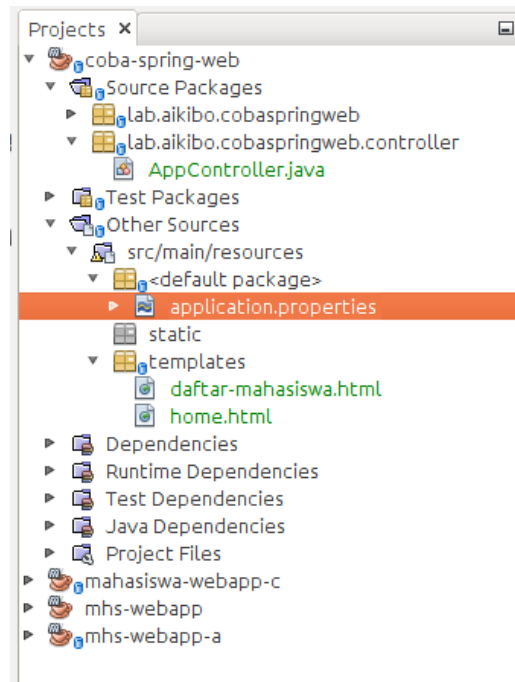
Untuk mempermudah langkah kita membangun sebuah aplikasi ini, kita kunjungi lagi alamat `start.spring.io` dengan beberapa pustaka seperti pada gambar 1.8 :

Gambar 1.8: *Generate Project* Untuk CRUD

Seperti sebelumnya, *file zip* yang telah diunduh kita ekstrak dan buka dengan IDE Netbeans / Sublime / IDE / *editor* lainnya.

Berikut langkah bagaimana kita dapat menampilkan data tabel dari basis data yang kita buat sebelumnya ke halaman aplikasi web yang kita buat :

1. Pertama kita buat konfigurasi koneksi ke basis data agar Spring Data JPA dapat melakukan akses data secara penuh ke sistem basis data yang kita gunakan. Konfigurasi ini ada pada *file application.properties* di dalam bagian **Other Sources** pada `<default package>`. Gambar ?? menunjukkan lokasi *file application.properties*.

Gambar 1.9: Lokasi *File* `application.properties`

Untuk yang menggunakan basis data PostgreSQL, isi konfigurasinya adalah sebagai berikut :

```
1 spring.datasource.url = jdbc:postgresql://localhost:5432/phb
2 spring.datasource.username = dev
3 spring.datasource.password = rahasia
4 spring.datasource.driver-class-name = org.postgresql.Driver
5
6 spring.jpa.database-platform = org.hibernate.dialect.
  PostgreSQL9Dialect
7
```

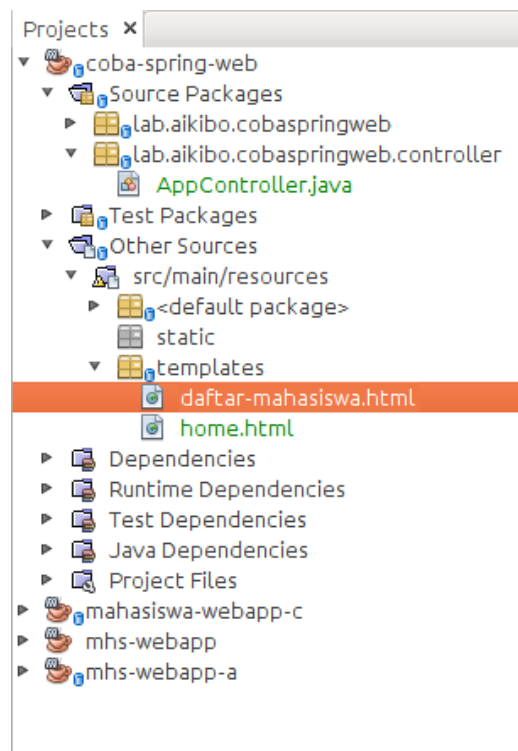
Format untuk `url` sendiri yang biasanya berubah adalah bagian `phb`, karena ini adalah nama *database* yang digunakan, bila menggunakan nama yang lain, silahkan diubah. Hal lain yang perlu disesuaikan tentu saja adalah bagian `username` dan `password`.

Isi konfigurasi bila kita menggunakan MySQL akan menjadi seperti berikut :

```
1 spring.datasource.url = jdbc:mysql://localhost:3306/
  mahasiswa
2 spring.datasource.username = dev
3 spring.datasource.password = rahasia
```

```
4 spring.datasource.driver-class-name = com.mysql.jdbc.Driver
5
6 spring.jpa.database-platform = org.hibernate.dialect.
  MySQLDialect
7
```

2. Selanjutnya kita buat dahulu *file html* sebagai *user interface* yang akan menampilkan isi dari tabel *mahasiswa*, *file* ini harus ditempatkan dalam pada bagian *Other Sources* di dalam *folder templates* seperti terlihat pada gambar 1.10 berikut :



Gambar 1.10: Lokasi *File* *daftar-mahasiswa.html*

Isi dari *file* *daftar-mahasiswa.html* ini adalah sebagai berikut :

```
1 <html xmlns:th="http://www.thymeleaf.org">
2   <head>
3     <title>Daftar Mahasiswa</title>
4     <meta charset="UTF-8" />
5     <meta name="viewport"
6       content="width=device-width, initial-scale
7       =1.0"/>
  </head>
```

```

8      <body>
9          <h1>Daftar Mahasiswa</h1>
10
11          <table border="1">
12              <thead>
13                  <tr>
14                      <th>NIM</th>
15                      <th>NAMA</th>
16                      <th>JURUSAN</th>
17                  </tr>
18              </thead>
19              <tbody>
20                  <tr th:each="mhs : ${daftarMahasiswa}">
21                      <td th:text="${mhs.nim}"></td>
22                      <td th:text="${mhs.nama}"></td>
23                      <td th:text="${mhs.jurusan}"></td>
24                  </tr>
25              </tbody>
26          </table>
27      </body>
28 </html>
29

```

Perhatikan pada baris ke-20, disana ada deklarasi variabel **mhs** yang nantinya akan diisi oleh setiap nilai yang ada pada variabel **daftarMahasiswa**. Variabel **daftarMahasiswa** sendiri sebetulnya akan dikirimkan dari *controller* di *server*.

Nilai dari masing-masing **daftarMahasiswa** itu sebetulnya adalah sebuah objek yang nantinya dititipkan ke variabel **mhs** yang kemudian pada baris ke-21 sampai ke-23 akan ditampilkan satu-satu berdasarkan nama propertinya, yaitu **nim**, **nama**, dan **jurusan**.

- Selanjutnya adalah membuat *controller* agar *file html* yang kita buat dapat tampil di *browser*. Gunakan saja *controller* yang sudah ada, yaitu **AppController** dengan tambahan **RequestMapping** baru sehingga kodenya menjadi terlihat seperti berikut ini :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
4 import org.springframework.beans.factory.annotation.
    Autowired;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.ui.Model;
7 import org.springframework.web.bind.annotation.
    RequestMapping;
8

```

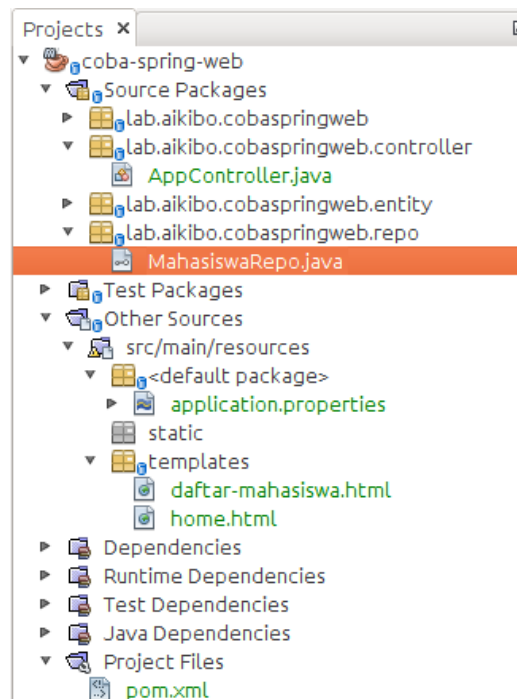
```
9  /**
10  *
11  *  @author tamami <tamami.oka@gmail.com>
12  */
13  @Controller
14  public class AppController {
15
16      @Autowired
17      private MahasiswaRepo mhsRepo;
18
19      @RequestMapping("/home")
20      public void index() {}
21
22      @RequestMapping("/daftar-mahasiswa")
23      public void getDaftarMahasiswa(Model model) {
24          model.addAttribute("daftarMahasiswa", mhsRepo.
findAll());
25      }
26
27  }
28
```

Perhatikan bahwa di kelas `controller` ini, ada anotasi baru yang kita gunakan, yaitu `Autowired` dimana ini adalah *feature* dari Spring untuk *dependency injection* yang secara otomatis akan membuat kita instan dari *interface* `MahasiswaRepo`.

Perhatikan juga pada baris ke-22, dimana kita membentuk *mapping* baru untuk `daftar-mahasiswa`, kemudian pada *method* `getDaftarMahasiswa` pada baris ke-23 memiliki sebuah parameter `model` yang merupakan pengait data antara *backend service* dengan *frontend service*.

Yang terakhir adalah pada baris ke-24 dimana pada `model` ditambahkan atribut `daftarMahasiswa` sebagaimana dibutuhkan sebelumnya pada *file* `daftar-mahasiswa.html`, yang isinya diambilkan dari basis data dengan memanggil *repository* kelas hasil turunan / pewarisan dari `JpaRepository`.

4. Membuat *file interface* `MahasiswaRepo` yang kita simpan pada *package* tersendiri. Gambar ?? menunjukkan lokasi dari *package repo* berada.



Gambar 1.11: Lokasi *Packgae* Repo
??

5. Kemudian membuat sebuah *interface* didalamnya untuk melakukan operasi *database*. Disini kita beri nama `MahasiswaRepo.java` dengan isi kodenya adalah sebagai berikut :

```

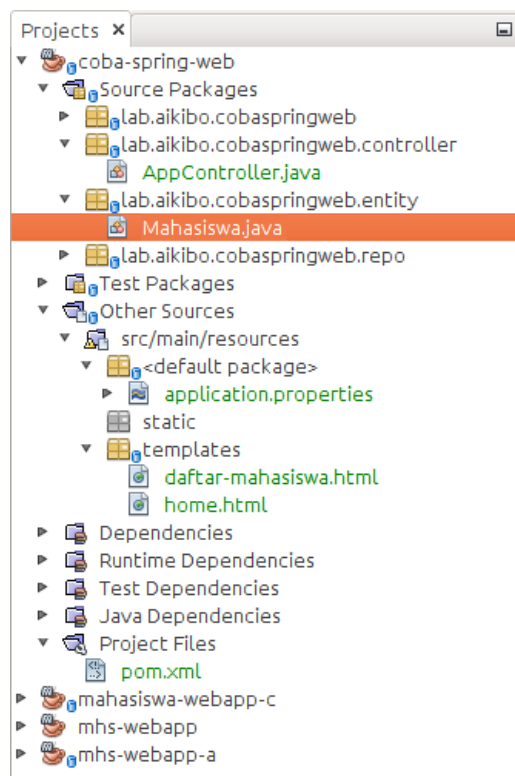
1 package lab.aikibo.cobaspringweb.repo;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import org.springframework.data.jpa.repository.JpaRepository
5     ;
6 import org.springframework.stereotype.Repository;
7 /**
8  *
9  * @author tamami <tamami.oka@gmail.com>
10  */
11 @Repository
12 public interface MahasiswaRepo extends JpaRepository<
13     Mahasiswa, String> {
14 }
15

```

Yang perlu di perhatikan adalah pada bagian deklarasi `JpaRepository`, disana membutuhkan 2 (dua) parameter, yaitu kelas entitasnya, kelas yang berfungsi sebagai tampungan data dari basis data, kemudian yang kedua adalah tipe data *key* dari tabel `mahasiswa`. Kebetulan yang menjadi *key* di tabel `mahasiswa` adalah `nim` dengan tipe data `String`.

6. Membuat kelas entitas untuk menampung data dari tabel `mahasiswa`. Nama *file* atau nama kelas harap diperhatikan bahwa harus sama dengan nama tabel di basis data, perbedaannya adalah pada nama kelas, huruf awalnya harus huruf kapital, sedangkan sisanya adalah huruf kecil biasa.

Kelas entitas ini kita beri nama `Mahasiswa` dengan penempatan pada lokasi *package* tersendiri agar lebih mudah kita pelihara aplikasinya. Lokasi dari *file* `Mahasiswa.java` dapat dilihat pada gambar 1.12 :



Gambar 1.12: Lokasi *File* `Mahasiswa.java`

Isi kode dari *file* `Mahasiswa.java` ini adalah sebagai berikut :

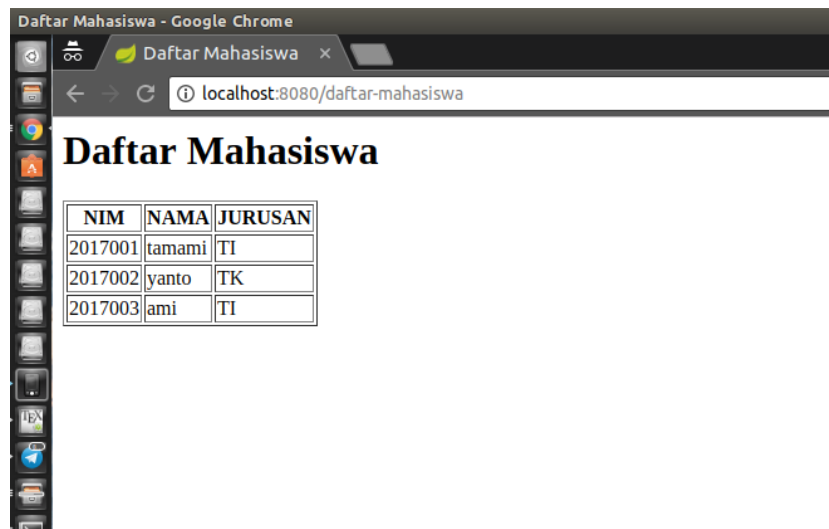
```
1 package lab.aikibo.cobaspringweb.entity;
```

```
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import lombok.Getter;
6 import lombok.Setter;
7
8 /**
9  *
10  * @author tamami <tamami.oka@gmail.com>
11  */
12 @Entity
13 public class Mahasiswa {
14     @Id
15     @Getter @Setter
16     private String nim;
17
18     @Getter @Setter
19     private String nama;
20
21     @Getter @Setter
22     private String jurusan;
23 }
24
```

Kode dari kelas **Mahasiswa** ini tampak biasa saja, ada tambahan anotasi **Entity** sebagai penanda bahwa ini adalah kelas entitas, ada anotasi **Id** yang melekat pada properti **nim** yang tujuannya adalah memberikan tanda bahwa *primary key* di tabel basis data akan disimpan disini, kemudian ada beberapa anotasi **Getter** dan **Setter** yang diambilkan dari pustaka **lombok** agar kode program yang kita buat lebih bersih dan lebih mudah untuk dibaca.

7. Sampai dengan langkah ini sebetulnya sudah selesai, tinggal dilakukan pengujian pada *browser* dengan terlebih dahulu melakukan *running* pada *project* ini.

Nantinya di *browser* harus melakukan akses ke `localhost:8080/daftar-mahasiswa`, sehingga akan muncul tampilan aplikasi web seperti pada gambar 1.13 berikut :

A screenshot of a Google Chrome browser window. The title bar says "Daftar Mahasiswa - Google Chrome". The address bar shows "localhost:8080/daftar-mahasiswa". The page content has a heading "Daftar Mahasiswa" followed by a table with three columns: NIM, NAMA, and JURUSAN. The table contains three rows of data.

NIM	NAMA	JURUSAN
2017001	tamami	TI
2017002	yanto	TK
2017003	ami	TI

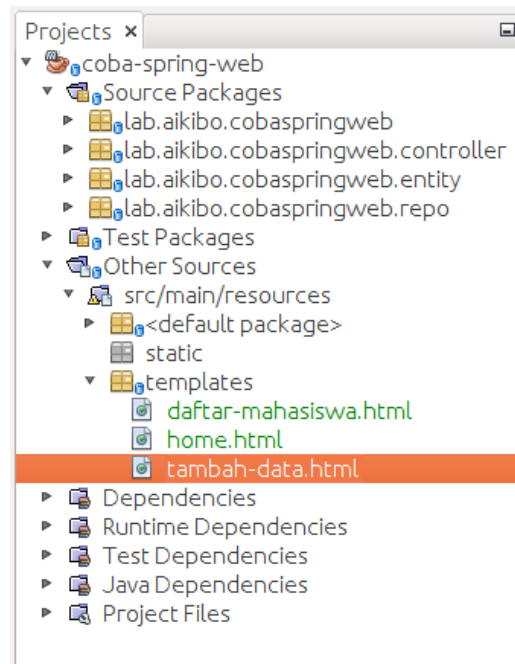
Gambar 1.13: Hasil Keluaran Daftar Mahasiswa

Sampai sini usaha kita untuk menampilkan data dari tabel ke aplikasi web telah berhasil.

1.3 Penambahan Data

Sekarang waktunya kita berikan fasilitas untuk menambahkan data pada aplikasi web yang kita bangun. Langkah-langkah yang diperlukan adalah sebagai berikut :

1. Membuat sebuah halaman `html` untuk perekaman data dan simpan pada bagian `Other Sources` pada *folder templates* seperti ditunjukkan oleh gambar 1.14 berikut ini :

Gambar 1.14: Lokasi *File* tambah-data.html

Isi dari *file* ini adalah sebagai berikut :

```

1 <html xmlns:th="http://www.thymeleaf.org">
2 <head>
3   <title>Formulir Entry Data</title>
4 </head>
5
6 <body>
7   <h1>Tambah Data</h1>
8
9   <form th:action="@{/tambah-data}" th:object="${mhs}"
    method="post">
10     <table>
11       <tr>
12         <td>NIM</td>
13         <td><input type="text" th:field="*{nim}" /> </td>
14       </tr>
15       <tr>
16         <td>NAMA</td>
17         <td><input type="text" th:field="*{nama}" /></td>
18       </tr>
19       <tr>
20         <td>JURUSAN</td>
21         <td><input type="text" th:field="*{jurusan}" /></td>
22       </tr>

```

```

23     </table>
24     <button type="submit">Simpan</button>
25 </form>
26 </body>
27 </html>
28

```

Pada baris ke-9 di bagian `th:action` adalah *url* tujuan dikirimkannya data yang terdapat pada formulir ini, kemudian `th:object` adalah nama variabel atau identitas objek yang akan dikirim ke *back-end*, sedangkan pada bagian `method` adalah cara atau metode yang digunakan untuk mengirimkan informasi ke *back-end*.

Data-data yang dikirim akan disiapkan dalam variabel-variabel atau parameter-parameter pada bagian `th:field`.

- Langkah selanjutnya adalah menambahkan sebuah tombol pada *file* `daftar-mahasiswa.html` dimana skenarionya, pada saat tombol ini diklik, nantinya akan diarahkan ke halaman / formulir `tambah-data.html`, kemudian *user* / pengguna diberikan kesempatan untuk memasukkan data-data yang diperlukan, kemudian, apabila *user* / pengguna melakukan simpan data, halaman akan dikembalikan ke `daftar-mahasiswa.html` dengan data baru ikut tampil dalam daftar.

Berikut adalah penambahan isi kode dari file `daftar-mahasiswa.html` :

```

1 <html xmlns:th="http://www.thymeleaf.org">
2   <head>
3     <title>Daftar Mahasiswa</title>
4     <meta charset="UTF-8" />
5     <meta name="viewport"
6       content="width=device-width, initial-scale
7       =1.0" />
8   </head>
9   <body>
10     <h1>Daftar Mahasiswa</h1>
11
12     <table border="1">
13       <thead>
14         <tr>
15           <th>NIM</th>
16           <th>NAMA</th>
17           <th>JURUSAN</th>
18         </tr>
19       </thead>
20       <tbody>
21         <tr th:each="mhs : ${daftarMahasiswa}">

```

```

21         <td th:text="${mhs.nim}"></td>
22         <td th:text="${mhs.nama}"></td>
23         <td th:text="${mhs.jurusan}"></td>
24     </tr>
25 </tbody>
26 <tfoot>
27     <tr>
28         <td><a href="/tambah-data.html">
29             Tambah
30         </a></td>
31     </tr>
32 </tfoot>
33 </table>
34 </body>
35 </html>
36

```

3. Selanjutnya agar *file* tersebut dapat muncul di *browser*, maka kita perlu menambahkan *controller* sebuah *mapping* yang mengarahkan ke `tambah-data.html`, berikut adalah hasil perubahan kode `AppController` :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod
    ;
12
13 /**
14  *
15  * @author tamami <tamami.oka@gmail.com>
16  */
17 @Controller
18 public class AppController {
19
20     @Autowired
21     private MahasiswaRepo mhsRepo;
22
23     @RequestMapping("/home")

```

```
24     public void index() {}
25
26     @RequestMapping("/daftar-mahasiswa")
27     public void getDaftarMahasiswa(Model model) {
28         model.addAttribute("daftarMahasiswa", mhsRepo.
findAll());
29     }
30
31
32     @RequestMapping(value = "/tambah-data",
33                     method = RequestMethod.GET)
34     public void getTambahData(
35         @ModelAttribute("mhs")Mahasiswa mhs,
36         BindingResult binding) {
37     }
38
39 }
40
```

Perhatikan pada baris ke-32 bahwa *mapping* yang disiapkan adalah untuk *url* `/tambah-data`, dengan `RequestMethod.GET`.

4. Melakukan pemeriksaan tampilan dengan menjalankan dan mengaksesnya melalui *browser*. Nantinya akan muncul jendela seperti gambar 1.15 berikut :



The screenshot shows a web browser window with the title 'Formulir Entry Data'. The address bar shows 'localhost:8080/tambah-data.html'. The page content is titled 'Tambah Data' and contains three input fields labeled 'NIM', 'NAMA', and 'JURUSAN'. Below these fields is a button labeled 'Simpan'.

Gambar 1.15: Tampilan Formulir Tambah Data

Formulir tersebut hanya berupa tampilan yang belum memiliki respon

apapun.

5. Agar tombol **Simpan** dapat berfungsi, kita harus siapkan sebuah *mapping* dengan *url* `/tambah-data` dengan metode `RequestMethod.POST`. Berikut adalah kode tambahan untuk `AppController` yang telah kita buat :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod
    ;
12
13 /**
14  *
15  * @author tamami <tamami.oka@gmail.com>
16  */
17 @Controller
18 public class AppController {
19
20     @Autowired
21     private MahasiswaRepo mhsRepo;
22
23     @RequestMapping("/home")
24     public void index() {}
25
26     @RequestMapping("/daftar-mahasiswa")
27     public void getDaftarMahasiswa(Model model) {
28         model.addAttribute("daftarMahasiswa", mhsRepo.
    findAll());
29     }
30
31     @RequestMapping(value = "/tambah-data", method =
    RequestMethod.GET)
32     public void getTambahData(@ModelAttribute("mhs")
    Mahasiswa mhs,
33         BindingResult binding) {
34     }
35

```

```
36     @RequestMapping(value = "/tambah-data",  
37         method = RequestMethod.POST)  
38     public String saveTambahData(  
39         @ModelAttribute("mhs") Mahasiswa mhs,  
40         BindingResult binding) {  
41         System.out.println(mhs.getNim());  
42         mhsRepo.save(mhs);  
43         return "redirect:/daftar-mahasiswa";  
44     }  
45 }  
46 }  
47 }
```

Pada baris ke-37 kita melihat bahwa *method* ini akan dieksekusi saat ada permintaan **post** pada *url* `/tambah-data`, pada baris ke-41 hanya untuk memastikan bahwa data yang dikirim dari *front-end* atau *user-interface* telah sampai ke *back-end*.

Perintah pada baris ke-42 adalah perintah inti, karena pada saat *request post* terjadi pada *url* `/tambah-data`, maka data yang terkirim harus tersimpan dalam basis data, untuk hal ini, kita menggunakan Spring Data JPA Repository untuk melakukan simpan data.

Setelah semua selesai dikerjakan, maka kembalikan tampilan yang ada menjadi daftar mahasiswa kembali dengan perintah seperti pada baris ke-43.

Setelah diuji, seharusnya data yang baru telah tersimpan dalam basis data dan data baru secara otomatis akan muncul pada daftar mahasiswa.

Sampai sini fasilitas untuk menambahkan data ke dalam tabel di basis data selesai.

1.4 Pengubahan Data

Kali ini kita akan memberikan sebuah fasilitas agar *user* dapat mengubah data yang telah tersimpan dalam basis data. Langkahnya adalah sebagai berikut :

1. Pada *file* `daftar-mahasiswa.html`, kita berikan pilihan atau tombol ubah untuk mengubah data yang ada, berikut perubahan kode yang terjadi pada *file* tersebut :

```

1 <html xmlns:th="http://www.thymeleaf.org">
2   <head>
3     <title>Daftar Mahasiswa</title>
4     <meta charset="UTF-8" />
5     <meta name="viewport"
6       content="width=device-width, initial-scale
7     =1.0"/>
8   </head>
9   <body>
10    <h1>Daftar Mahasiswa</h1>
11
12    <table border="1">
13      <thead>
14        <tr>
15          <th>NIM</th>
16          <th>NAMA</th>
17          <th>JURUSAN</th>
18        </tr>
19      </thead>
20      <tbody>
21        <tr th:each="mhs : ${daftarMahasiswa}">
22          <td th:text="${mhs.nim}"></td>
23          <td th:text="${mhs.nama}"></td>
24          <td th:text="${mhs.jurusan}"></td>
25          <td>
26            <a href="@{/edit(nim=${mhs.nim})}">
27              Ubah
28            </a>
29          </td>
30        </tr>
31      </tbody>
32      <tfoot>
33        <tr>
34          <td><a href="/tambah-data.html">Tambah</
35          a></td>
36        </tr>
37      </tfoot>
38    </table>
39  </body>
40</html>

```

Pada baris ke-25 kita memberikan identitas untuk tiap baris yang muncul di tabel, sehingga tombol **Ubah** manapun yang diklik akan memberikan informasi data mahasiswa dengan NIM mana yang akan diubah.

Nantinya *url* yang akan muncul saat tombol **Ubah** diklik akan berbentuk seperti ini `/edit?nim={nim}`

2. Langkah berikutnya adalah menyiapkan *controller* untuk alamat yang diinginkan dari *edit* data tersebut, berikut adalah perubahan kode yang terjadi di kelas **AppController** :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod
    ;
12 import org.springframework.web.bind.annotation.RequestParam;
13
14
15 /**
16  *
17  * @author tamami <tamami.oka@gmail.com>
18  */
19 @Controller
20 public class AppController {
21
22     @Autowired
23     private MahasiswaRepo mhsRepo;
24
25     @RequestMapping("/home")
26     public void index() {}
27
28     @RequestMapping("/daftar-mahasiswa")
29     public void getDaftarMahasiswa(Model model) {
30         model.addAttribute("daftarMahasiswa", mhsRepo.
            findAll());
31     }
32
33     @RequestMapping(value = "/tambah-data", method =
        RequestMethod.GET)
34     public void getTambahData(@ModelAttribute("mhs")
        Mahasiswa mhs,
35         BindingResult binding) {
36     }
37
38     @RequestMapping(value = "/tambah-data", method =

```

```

39 RequestMethod.POST)
    public String saveTambahData(@ModelAttribute("mhs")
    Mahasiswa mhs,
40        BindingResult binding) {
41        System.out.println(mhs.getNim());
42        mhsRepo.save(mhs);
43        return "redirect:/daftar-mahasiswa";
44    }
45
46    @RequestMapping(value = "/edit", method = RequestMethod.GET)
47    public void getEditData(@RequestParam(name = "nim", required =
    false) String nim,
48        @ModelAttribute("mhs") Mahasiswa mahasiswa, BindingResult
    binding) {
49        Mahasiswa mhs = mhsRepo.findByNim(nim);
50        mahasiswa.setNim(mhs.getNim());
51        mahasiswa.setNama(mhs.getNama());
52        mahasiswa.setJurusan(mhs.getJurusan());
53    }
54 }
55

```

Method tersebut akan merespon dengan tampilnya halaman `edit.html`, parameter `nim` yang dikirimkan oleh *client* akan dicari datanya seperti pada baris ke-49, kemudian hasil pencarian akan disimpan dalam variabel `mhs` yang dikembalikan ke halaman `edit.html`.

3. Membuat halaman `edit.html` untuk memberikan kesempatan *user* melakukan perubahan data, berikut isi kode dari `edit.html` :

```

1 <html xmlns:th="http://www.thymeleaf.org">
2 <head>
3     <title>Formulir Ubah Data</title>
4 </head>
5
6 <body>
7     <h1>Ubah Data</h1>
8
9     <form th:action="@{/edit}" th:object="${mhs}" method="post">
10         <table>
11             <tr>
12                 <td>NIM</td>
13                 <td><input type="text" th:field="*{nim}" disabled="
    true" /> </td>
14                 <td><input type="hidden" th:field="*{nim}" /></td>
15             </tr>
16             <tr>
17                 <td>NAMA</td>

```

```

18         <td><input type="text" th:field="*{nama}" /></td>
19     </tr>
20     <tr>
21         <td>JURUSAN</td>
22         <td><input type="text" th:field="*{jurusan}" /></td>
23     </tr>
24 </table>
25 <button type="submit">Simpan</button>
26 </form>
27 </body>
28 </html>
29

```

Kode ini sama seperti kode untuk *entry* data, hanya saja pada saat data dikirimkan akan menuju ke *url /edit* dengan metode *post*.

4. Melakukan perubahan pada kelas **AppController** agar dapat menangani metode *post* pada *url /edit*, berikut perubahan kode yang terjadi pada kelas **AppController** :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod
    ;
12 import org.springframework.web.bind.annotation.RequestParam;
13
14
15 /**
16  *
17  * @author tamami <tamami.oka@gmail.com>
18  */
19 @Controller
20 public class AppController {
21
22     @Autowired
23     private MahasiswaRepo mhsRepo;
24
25     @RequestMapping("/home")

```

```
26     public void index() {}
27
28     @RequestMapping("/daftar-mahasiswa")
29     public void getDaftarMahasiswa(Model model) {
30         model.addAttribute("daftarMahasiswa", mhsRepo.
findAll());
31     }
32
33     @RequestMapping(value = "/tambah-data", method =
RequestMethod.GET)
34     public void getTambahData(@ModelAttribute("mhs")
Mahasiswa mhs,
35         BindingResult binding) {
36     }
37
38     @RequestMapping(value = "/tambah-data", method =
RequestMethod.POST)
39     public String saveTambahData(@ModelAttribute("mhs")
Mahasiswa mhs,
40         BindingResult binding) {
41         System.out.println(mhs.getNim());
42         mhsRepo.save(mhs);
43         return "redirect:/daftar-mahasiswa";
44     }
45
46     @RequestMapping(value = "/edit", method = RequestMethod.
GET)
47     public void getEditData(@RequestParam(name = "nim",
required = false) String nim,
48         @ModelAttribute("mhs") Mahasiswa mahasiswa,
BindingResult binding) {
49         Mahasiswa mhs = mhsRepo.findByNim(nim);
50         mahasiswa.setNim(mhs.getNim());
51         mahasiswa.setNama(mhs.getNama());
52         mahasiswa.setJurusan(mhs.getJurusan());
53     }
54
55     @RequestMapping(value = "/edit",
56         method = RequestMethod.POST)
57     public String saveEditData(
58         @ModelAttribute("mhs") Mahasiswa mhs,
59         BindingResult binding) {
60         mhsRepo.save(mhs);
61         return "redirect:/daftar-mahasiswa";
62     }
63 }
64 }
65 }
```

Pada kode tersebut, sama seperti metode *get* di atasnya, memiliki `ModelAttribute` dan `BindingResult`, jadi perintahnya apabila metode ini dieksekusi, maka langsung saja simpan perubahannya pada basis data seperti perintah pada baris ke-60, dan kembalikan tampilannya ke `/daftar-mahasiswa` agar perubahan langsung dapat ditampilkan.

Sampai sini operasi perubahan data pada basis data selesai.

1.5 Penghapusan Data

Pada bagian ini akan kita bahas bagaimana cara menghapus sebuah data di basis data melalui aplikasi web yang telah kita bangun. Secara konsep akan mirip seperti saat kita memberikan fasilitas perubahan data, ada identitas unik untuk tiap baris yang kita kirimkan ke *backend service*, kebetulan yang menjadi identitas unik ini adalah NIM dari tiap mahasiswa, sehingga NIM ini akan kita lewatkan atau kita jadikan salah satu parameter *request* yang dikirim ke *backend service*.

Langkah-langkahnya adalah sebagai berikut :

1. Pertama kita ubah terlebih dahulu *file* `daftar-mahasiswa.html` sehingga kodenya menjadi terlihat seperti ini :

```
1 <html xmlns:th="http://www.thymeleaf.org">
2   <head>
3     <title>Daftar Mahasiswa</title>
4     <meta charset="UTF-8" />
5     <meta name="viewport"
6       content="width=device-width, initial-scale
7     =1.0"/>
8   </head>
9   <body>
10    <h1>Daftar Mahasiswa</h1>
11
12    <table border="1">
13      <thead>
14        <tr>
15          <th>NIM</th>
16          <th>NAMA</th>
17          <th>JURUSAN</th>
18        </tr>
19      </thead>
20      <tbody>
21        <tr th:each="mhs : ${daftarMahasiswa}">
22          <td th:text="${mhs.nim}"></td>
23          <td th:text="${mhs.nama}"></td>
24          <td th:text="${mhs.jurusan}"></td>
```

```

24         <td><a th:href="@{/edit(nim=${mhs.nim})}">Ubah</a></td>
25         <td>
26             <a th:href="@{/delete(nim=${mhs.nim})}">
27                 Hapus
28             </a>
29         </td>
30     </tr>
31 </tbody>
32 <tfoot>
33     <tr>
34         <td><a href="/tambah-data.html">Tambah</a></td>
35     </tr>
36 </tfoot>
37 </table>
38 </body>
39 </html>
40

```

2. Karena *url* yang dituju belum didefinisikan di *controller*, maka kita akan mengubah kode kelas *AppController* yang kita miliki menjadi seperti ini :

```

1 package lab.aikibo.cobaspringweb.controller;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import lab.aikibo.cobaspringweb.repo.MahasiswaRepo;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod;
12 ;
13 import org.springframework.web.bind.annotation.RequestParam;
14
15 /**
16  *
17  * @author tamami <tamami.oka@gmail.com>
18  */
19 @Controller
20 public class AppController {
21

```

```
22     @Autowired
23     private MahasiswaRepo mhsRepo;
24
25     @RequestMapping("/home")
26     public void index() {}
27
28     @RequestMapping("/daftar-mahasiswa")
29     public void getDaftarMahasiswa(Model model) {
30         model.addAttribute("daftarMahasiswa", mhsRepo.
31         findAll());
32     }
33
34     @RequestMapping(value = "/tambah-data", method =
35     RequestMethod.GET)
36     public void getTambahData(@ModelAttribute("mhs")
37     Mahasiswa mhs,
38     BindingResult binding) {
39
40     }
41
42     @RequestMapping(value = "/tambah-data", method =
43     RequestMethod.POST)
44     public String saveTambahData(@ModelAttribute("mhs")
45     Mahasiswa mhs,
46     BindingResult binding) {
47         System.out.println(mhs.getNim());
48         mhsRepo.save(mhs);
49         return "redirect:/daftar-mahasiswa";
50     }
51
52     @RequestMapping(value = "/edit", method = RequestMethod.
53     GET)
54     public void getEditData(@RequestParam(name = "nim",
55     required = false) String nim,
56     @ModelAttribute("mhs") Mahasiswa mahasiswa,
57     BindingResult binding) {
58         Mahasiswa mhs = mhsRepo.findByNim(nim);
59         mahasiswa.setNim(mhs.getNim());
60         mahasiswa.setNama(mhs.getNama());
61         mahasiswa.setJurusan(mhs.getJurusan());
62     }
63
64     @RequestMapping(value = "/edit", method = RequestMethod.
65     POST)
66     public String saveEditData(@ModelAttribute("mhs")
67     Mahasiswa mhs,
68     BindingResult binding) {
69         mhsRepo.save(mhs);
70         return "redirect:/daftar-mahasiswa";
71     }
72 }
```

```
61
62     @RequestMapping("/delete")
63     public String deleteData(
64         @RequestParam(name = "nim", required = true) String nim) {
65         mhsRepo.delete(nim);
66         return "redirect:/daftar-mahasiswa";
67     }
68
69 }
70
```

Perintahnya cukup sederhana, dari parameter NIM yang dikirimkan oleh *browser*, kita cukup memanggil *method delete* milik *JpaRepository* berdasarkan *primary key*, kemudian mengembalikan halaman *browser* ke *url /daftar-mahasiswa*.

Cukup 2 (dua) perubahan tersebut, kita sudah menyelesaikan fasilitas hapus data pada aplikasi perekaman data mahasiswa yang kita bangun.

Sekarang lengkap sudah fasilitas yang ada pada aplikasi web perekaman data mahasiswa, untuk selanjutnya dapat dikembangkan atau diimplementasikan pada soal UTS.

Bab 2

CRUD WITH SPRING ANGULAR

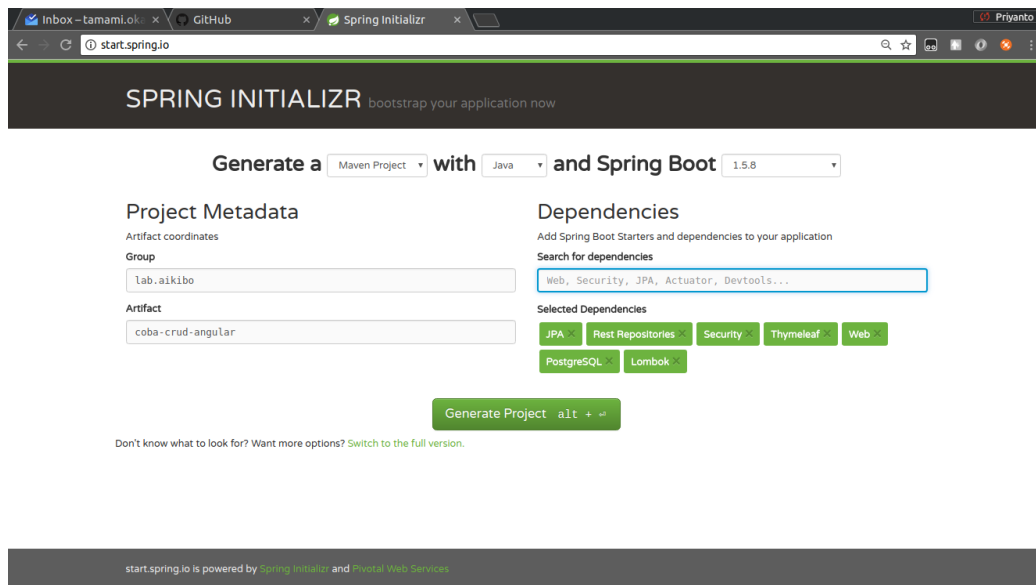
2.1 Memulai

Untuk membangun *project* dengan Spring dan AngularJS, lebih mudahnya kita mengawali dengan membuat kerangka dari <http://start.spring.io>.

Beberapa pustaka yang perlu kita sertakan adalah sebagai berikut :

- JPA. Yang nantinya kita butuhkan untuk memudahkan melakukan operasi manipulasi basis data.
- Rest Repositories. Ini digunakan *backend service* mengirimkan pesan atau data ke *front-end service* Angular.
- Security. Ini adalah paket atau modul yang disediakan *Framework* Spring untuk menjaga / mengotorisasi akses *user*.
- Thymeleaf. Digunakan untuk melakukan *templating* halaman `html`.
- Web. Digunakan untuk implementasi MVC pada aplikasi web yang kita bangun.
- PostgreSQL. Digunakan sebagai *driver* basis data yang menghubungkan Java dengan basis data postgresql.
- Lombok. Digunakan untuk menjadikan kode yang kita buat lebih bersih.

Pada halaman start.spring.io akan terlihat seperti gambar 2.1 berikut :

Gambar 2.1: *Generate Project* Untuk Spring Angular

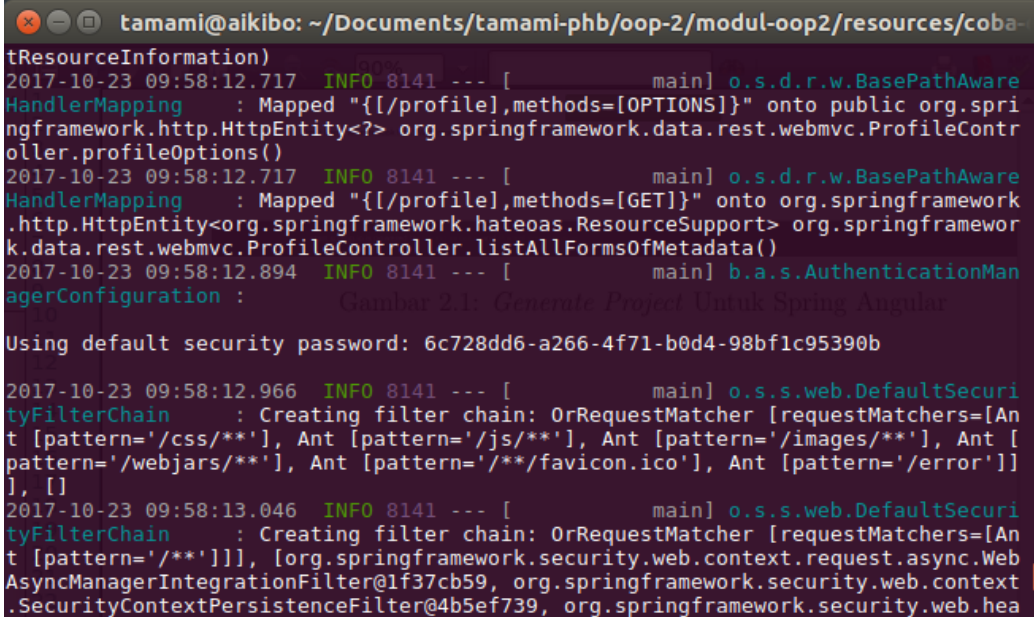
Selanjutnya kita ubah terlebih dahulu *file* `application.properties` dengan konfigurasi koneksi basis data, karena kita telah menambahkan Spring JPA dalam konfigurasi Maven kita. Berikut adalah isi dari *file* konfigurasi `application.properties` :

```

1 spring.datasource.url = jdbc:postgresql://localhost:5432/phb
2 spring.datasource.username = dev
3 spring.datasource.password = rahasia
4 spring.datasource.driver-class-name = org.postgresql.Driver
5
6 spring.jpa.database-platform=org.hibernate.dialect.
   PostgreSQL9Dialect
7
8 spring.jackson.serialization.indent_output = true

```

Setelah pengisian konfigurasi basis data, kita dapat melakukan tes langsung terhadap Spring Security, jalankan dengan Maven dan perhatikan informasi yang muncul di layar seperti pada gambar 2.2 berikut :



```

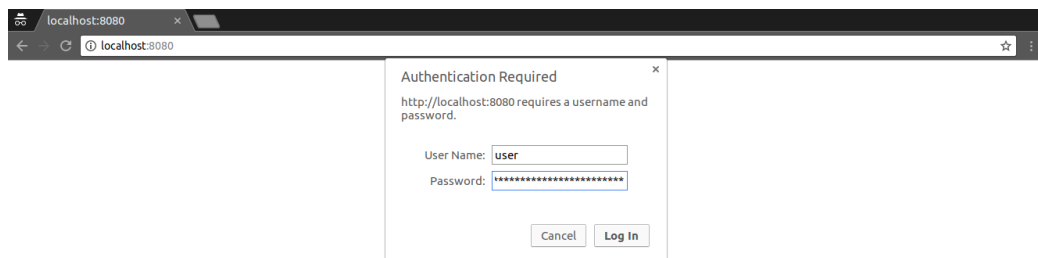
tamami@aikibo: ~/Documents/tamami-phb/oop-2/modul-oop2/resources/coba-
tResourceInformation)
2017-10-23 09:58:12.717 INFO 8141 --- [main] o.s.d.r.w.BasePathAware
HandlerMapping : Mapped "{[/profile],methods=[OPTIONS]}" onto public org.spr
ingframework.http.HttpEntity<?> org.springframework.data.rest.webmvc.ProfileContr
oller.profileOptions()
2017-10-23 09:58:12.717 INFO 8141 --- [main] o.s.d.r.w.BasePathAware
HandlerMapping : Mapped "{[/profile],methods=[GET]}" onto org.springframework
k.data.rest.webmvc.ProfileController.listAllFormsOfMetadata()
2017-10-23 09:58:12.894 INFO 8141 --- [main] b.a.s.AuthenticationMan
agerConfiguration :
Using default security password: 6c728dd6-a266-4f71-b0d4-98bf1c95390b

2017-10-23 09:58:12.966 INFO 8141 --- [main] o.s.s.web.DefaultSecuri
tyFilterChain : Creating filter chain: OrRequestMatcher [requestMatchers=[An
t [pattern='/css/**'], Ant [pattern='/js/**'], Ant [pattern='/images/**'], Ant [
pattern='/webjars/**'], Ant [pattern='/**/favicon.ico'], Ant [pattern='/error']]
], []
2017-10-23 09:58:13.046 INFO 8141 --- [main] o.s.s.web.DefaultSecuri
tyFilterChain : Creating filter chain: OrRequestMatcher [requestMatchers=[An
t [pattern='/**']]], [org.springframework.security.web.context.request.async.Web
AsyncManagerIntegrationFilter@1f37cb59, org.springframework.security.web.context
.SecurityContextPersistenceFilter@4b5ef739, org.springframework.security.web.he

```

Gambar 2.2: *Default Password Spring Security*

Bukalah *browser* sehingga akan memberikan tampilan seperti gambar 2.3 berikut ini :

Gambar 2.3: Isian *User* dan *Password*

Lalu isikan informasi *username* dengan *user* dan *password* dengan karakter panjang yang diberikan Spring Security di gambar 2.2.

Setelah itu akan diberikan halaman Rest *default* berupa JSON seperti terlihat pada gambar 2.4 berikut :



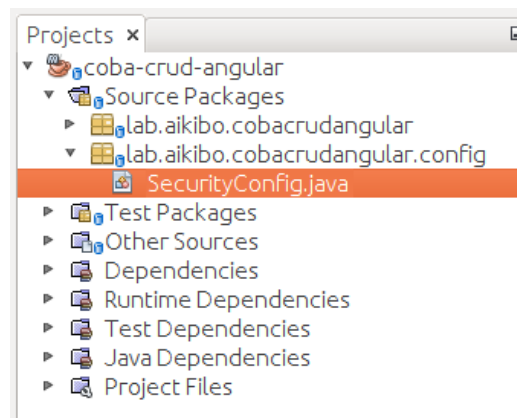
Gambar 2.4: *Default Page* Setelah *Login*

Sampai kondisi ini, Spring JPA sudah berhasil terhubung dengan sistem basis data dan *security* telah berjalan dengan baik.

2.2 Merubah *Username* dan *Password*

Lalu bagaimana agar *username* dan *password* dapat kita tentukan sendiri? Berikut adalah langkahnya :

1. Menambahkan kelas `SecurityConfig`. Pada paket `config` seperti terlihat pada gambar 2.5 berikut :

Gambar 2.5: Lokasi *File SecurityConfig.java*

Kode dari kelas **SecurityConfig** sendiri adalah seperti kode berikut :

```

1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.security.config.annotation.web.
   configuration.WebSecurityConfigurerAdapter;
5 import org.springframework.security.config.annotation.web.
   servlet.configuration.EnableWebMvcSecurity;
6 import org.springframework.beans.factory.annotation.
   Autowired;
7 import org.springframework.security.config.annotation.
   authentication.builders.AuthenticationManagerBuilder;
8
9 @Configuration
10 @EnableWebMvcSecurity
11 public class SecurityConfig extends
   WebSecurityConfigurerAdapter {
12
13     @Autowired
14     public void configureGlobal(AuthenticationManagerBuilder
   auth) throws Exception {
15         auth.inMemoryAuthentication()
16             .withUser("tamami")
17             .password("rahasia")
18             .roles("ADMIN");
19     }
20 }
21
22

```

Dengan menambahkan kelas **SecurityConfig** tersebut, halaman *login*

akan berubah bila kita coba jalankan aplikasinya, halaman *login* akan terlihat seperti gambar ?? berikut :



Gambar 2.6: *Username dan Password Custom*

Isikan dengan *username* dan *password* seperti pada kelas `SecurityConfig` pada baris ke-16 dan ke-17. Perhatikan pula bahwa pada halaman login ada kode `csrf` sebagai pengaman.

2. Sampai sini, langkah konfigurasi *security* untuk aplikasi *web* yang kita bangun masih statis, *username* dan *password* masih harus *hard-code*.

2.3 *Username dan Password Dari DB*

Selanjutnya kita akan bangun sistem *security* dengan *username* dan *password* yang dinamis yang datanya diambilkan dari sistem basis data.

Berikut adalah langkah-langkahnya :

1. Buatlah struktur tabelnya terlebih dahulu, karena struktur tabel untuk Spring Security akan mengikuti pola aturan tertentu, berikut adalah beberapa tabel yang perlu dibuat :
 - a. Tabel *User*. Nantinya dari tabel ini nama *user* dan *password* akan dicocokkan. Struktur tabelnya dengan nama table **users** adalah sebagai berikut :

Kolom	Tipe	Keterangan
username	varchar(30)	not null primary key
password	varchar(200)	not null
enabled	boolean	

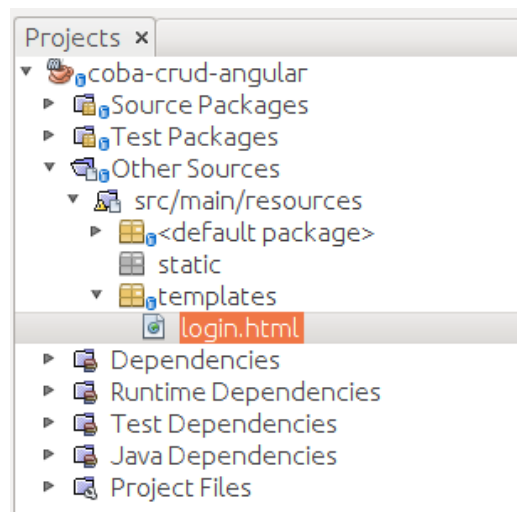
- b. Tabel *Role*. Nantinya daftar akses akan ditempatkan dalam tabel ini. Struktur tabelnya dengan nama tabel **roles** adalah sebagai berikut :

Kolom	Tipe	Keterangan
id	varchar(36)	not null primary key
role	varchar(200)	not null

- c. Tabel *User-Role*. Nantinya masing-masing *user* akan memiliki kewenangan tertentu berdasarkan *role* yang diberikan atau dipasangkan pada tabel ini. Struktur tabelnya dengan nama tabel **user_role** adalah sebagai berikut :

Kolom	Tipe	Keterangan
username	varchar(30)	not null
id_role	varchar(36)	not null

2. Selanjutnya kita buat halaman login dengan tambahan Bootstrap untuk mempercantiknya. Lokasi dari *file login.html* disimpan di dalam *folder templates* seperti terlihat pada gambar 2.7 berikut ini :



Gambar 2.7: Lokasi *file login.html*

Isi dari *file login.html* ini adalah sebagai berikut :

```

1 <html>
2   <head>
3     <title>Login</title>
4
5     <link rel="stylesheet" href="https://maxcdn.
bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.
min.css" integrity="sha384-
PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
" crossorigin="anonymous">
6     <script src="https://code.jquery.com/jquery-3.2.1.
slim.min.js" integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
7     <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
vFJXuSjphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
" crossorigin="anonymous"></script>
8     <script src="https://maxcdn.bootstrapcdn.com/bootstrap
/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
" crossorigin="anonymous"></script>
9   </head>
10
11  <body>
12    <div class="container">
13      <h1 div="card-title">Login</h1>
14      <form id="post" method="post" th:action="@{/login
}">
15        <div class="card">
16          <div class="card-body">
17            <div class="container">
18              <div class="row">
19                <div class="col"><input name="
username" type="text" class="form-control" placeholder="
Username"/></div>
20              </div>
21              <div class="row">
22                <div class="col"><input name="
password" type="password" class="form-control"
placeholder="Password"/></div>
23              </div>
24              <div class="row">
25                <div class="col">
26                  <input type="submit" class="
btn btn-primary" value="Masuk" />
27                </div>
28              </div>
29            </div>
30          </div>

```



```

31         </div>
32     </form>
33 </div>
34 </body>
35 </html>
36

```

3. Mengubah kelas `SecurityConfig` agar melakukan *override* terhadap *method* `configureGlobal` dan menambahkan *method* `configure(HttpSecurity)`. Berikut adalah perubahan kode dari kelas `SecurityConfig` :

```

1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.security.config.annotation.web.
    configuration.WebSecurityConfigurerAdapter;
5 import org.springframework.security.config.annotation.web.
    servlet.configuration.EnableWebMvcSecurity;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.security.config.annotation.web.
    builders.HttpSecurity;
8 import org.springframework.security.config.annotation.
    authentication.builders.AuthenticationManagerBuilder;
9 import javax.sql.DataSource;
10 import org.springframework.security.web.csrf.
    CsrfTokenRepository;
11 import org.springframework.security.web.csrf.
    HttpSessionCsrfTokenRepository;
12 import lab.aikibo.cobacrudangular.util.CsrfHeaderFilter;
13 import org.springframework.security.config.annotation.web.
    builders.HttpSecurity;
14 import org.springframework.security.web.csrf.CsrfFilter;
15
16 @Configuration
17 @EnableWebMvcSecurity
18 public class SecurityConfig extends
    WebSecurityConfigurerAdapter {
19
20     private static final String SQL_LOGIN =
21         "select username, password, enabled from users " +
22         "where username = ?";
23
24     private static final String SQL_PERMISSION =
25         "select u.username, r.role as authority " +
26         "from users u " +
27         "join user_role ur on u.username = ur.username " +
28         "join roles r on r.id = ur.id_role " +
29         "where u.username = ?";

```

```
30
31 @Autowired
32 private DataSource ds;
33
34 @Autowired
35 public void configureGlobal(AuthenticationManagerBuilder
36     auth) throws Exception {
37     auth.jdbcAuthentication()
38         .dataSource(ds)
39         .usersByUsernameQuery(SQL_LOGIN)
40         .authoritiesByUsernameQuery(SQL_PERMISSION);
41 }
42
43 @Override
44 protected void configure(HttpSecurity http) throws Exception {
45     http
46         .authorizeRequests()
47         .anyRequest().authenticated()
48         .and()
49         .formLogin().loginPage("/login").permitAll()
50         .defaultSuccessUrl("/")
51         .and()
52         .logout()
53         .and()
54         .addFilterAfter(new CsrfHeaderFilter(),
55             CsrfFilter.class)
56         .csrf()
57         .csrfTokenRepository(csrfTokenRepository());
58 }
59
60 private CsrfTokenRepository csrfTokenRepository() {
61     HttpSessionCsrfTokenRepository tokenRepo =
62         new HttpSessionCsrfTokenRepository();
63     tokenRepo.setHeaderName("X-XSRF-TOKEN");
64     return tokenRepo;
65 }
66 }
67 }
```

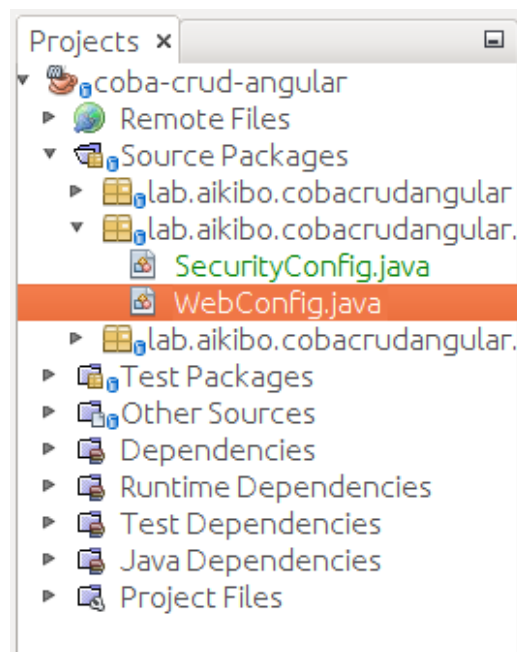
Pada baris ke-36, kita sudah mulai melakukan otentikasi *username* dan *password* dari sistem basis data.

Pada baris ke-43 adalah tempat kita mengatur halaman mana yang boleh dibuka oleh umum, dan mana yang hanya boleh setelah *login* bahkan pada *method* ini pun dapat diberikan akses lebih spesifik terhadap *user* mana yang boleh melakukan akses ke *url* tertentu.

Pada baris ke-56 adalah cara atau mekanisme yang dilakukan Spring

untuk menyampaikan CSRF dari *user* yang melakukan akses, dimana implementasinya dilakukan pada *method* di baris ke-59, yaitu dengan cara menyematkan atau menempelkan informasi CSRF di *header* tiap *request*.

4. Karena kita membuat halaman *login* khusus yang kita desain sendiri, maka halaman ini harus didaftarkan terlebih dahulu, mirip seperti saat kita buat aplikasi dengan Thymeleaf sebelumnya yang memerlukan *mapping*, namun kali ini dengan cara yang berbeda. Kali ini kita akan membuat sebuah kelas dengan nama `WebConfig` yang menjadi tempat untuk mendaftarkan setiap *View* yang kita buat. Letak *file* dari `WebConfig` ini adalah seperti pada gambar 2.8 berikut :



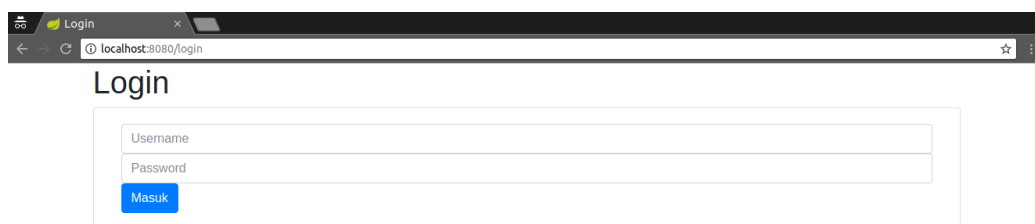
Gambar 2.8: Lokasi *File* `WebConfig`

Isi kode dari *file* ini adalah sebagai berikut :

```
1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.
    ViewControllerRegistry;
5 import org.springframework.web.servlet.config.annotation.
    WebMvcConfigurerAdapter;
6
```

```
7 @Configuration
8 public class WebConfig extends WebMvcConfigurerAdapter {
9
10     @Override
11     public void addViewControllers(ViewControllerRegistry
12         registry) {
13         registry.addViewController("/login").setViewName("login
14     ");
15 }
16 }
```

5. Lalu melakukan pemeriksaan aplikasi, seharusnya aplikasi akan menampilkan halaman *login* yang telah kita bangun sendiri seperti pada gambar 2.9 berikut :



Gambar 2.9: Halaman *Login* Yang Dibangun Sendiri

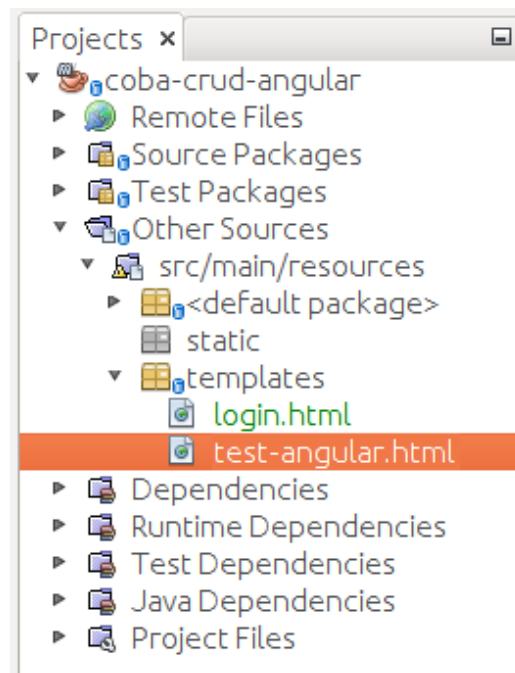
Dan apabila berhasil *login* maka akan muncul halaman seperti pada gambar 2.10 berikut :

Gambar 2.10: Halaman Setelah *Login* Berhasil

2.4 Mencoba Angular

Karena kita sudah berhasil membuat bagian *login* berjalan dengan semestinya, kali ini kita akan mencoba mengawali pembahasan Angular dengan cara sederhana. Kita akan membuat sebuah halaman interaktif yang dapat langsung berubah ketika *user* melakukan aksi di sebuah halaman. Berikut adalah langkahnya :

1. Kita akan membuat sebuah halaman `html` terlebih dahulu, misalkan kita beri nama `test-angular.html`. Lokasi dari *file* ini adalah seperti pada gambar 2.11 berikut :

Gambar 2.11: Lokasi *File* test-angular.html

Isi dari kode programnya adalah sebagai berikut :

```

1 <html>
2 <head>
3   <title>Percobaan Angular</title>
4
5   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
6     integrity="sha384-
7     PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
8     " crossorigin="anonymous"/>
9   <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
    KJ3o2DKtIkvYIK3UENzmM7KCkRr/
    rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
    anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
    vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
    " crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
    alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
    " crossorigin="anonymous"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
  
```

```

10 </head>
11 <body ng-app="">
12   Isikan nama <input type="text" ng-model="nama" />
13   <br />
14   Selamat datang , {{ nama }}
15 </body>
16 </html>
17

```

Yang perlu diperhatikan adalah pada baris ke-9, kita menambahkan *script* untuk AngularJS, agar AngularJS dapat kita gunakan. Kemudian pada baris ke-12, ada parameter `ng-model` yang isinya adalah pembentukan sebuah variabel pada AngularJS dengan nama `nama`. Nantinya variabel ini akan dipanggil / dibaca pada baris ke-14.

2. Registrasikan halaman ini di `WebConfig`, sehingga kelas `WebConfig` menjadi seperti ini :

```

1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.
   ViewControllerRegistry;
5 import org.springframework.web.servlet.config.annotation.
   WebMvcConfigurerAdapter;
6
7 @Configuration
8 public class WebConfig extends WebMvcConfigurerAdapter {
9
10    @Override
11    public void addViewControllers(ViewControllerRegistry
       registry) {
12        registry.addViewController("/login").setViewName("login
       ");
13        registry.addViewController("/test-angular")
14            .setViewName("test-angular");
15    }
16
17 }
18

```

Sehingga apabila ada *request* ke *url* `/test-angular` akan di *respon* dengan halaman `test-angular.html`.

3. Melakukan uji coba aplikasi, seharusnya apabila alamat *url* diarahkan ke `localhost:8080/test-angular`, nantinya akan aplikasi akan meminta informasi *login*, dan setelah berhasil *login*, nanti akan dibawa ke halaman seperti pada gambar 2.12 berikut :

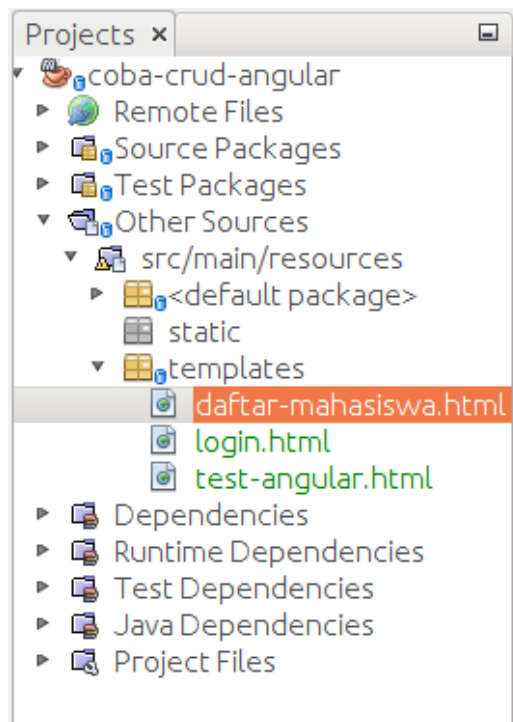
Gambar 2.12: Halaman `test-angular.html`

Bila *user* mengisikan nama ditempat yang disediakan, aplikasi dapat langsung melakukan respon interaktif tanpa harus melakukan *request* ke *server*.

2.5 Tampilkan Data

Sekarang kita coba tampilkan isi dari basis data ke halaman *web* kita dengan Angular, berikut langkah-langkahnya :

1. Yang pertama kita lakukan adalah membuat tampilan *front-end* terlebih dahulu, yaitu daftar mahasiswa dalam sebuah tabel, kita sebut saja nama *file* untuk ini adalah `daftar-mahasiswa.html`, lokasi *file* dapat dilihat seperti pada gambar 2.13 berikut :

Gambar 2.13: Lokasi *File* daftar-mahasiswa.html

Isi dari *file* daftar-mahasiswa.html ini adalah sebagai berikut :

```

1 <html>
2 <head>
3     <title>Aplikasi Pendataan Mahasiswa</title>
4     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
      integrity="sha384-
      PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUUpPvrszuE4W1povHYgTpBfshb
      " crossorigin="anonymous"/>
5     <script src="https://code.jquery.com/jquery-3.2.1.slim.
      min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/
      rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
      anonymous"></script>
6     <script src="https://cdnjs.cloudflare.com/ajax/libs/
      popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
      vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
      " crossorigin="anonymous"></script>
7     <script src="https://maxcdn.bootstrapcdn.com/bootstrap
      /4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
      alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
      " crossorigin="anonymous"></script>
8     <script src="https://ajax.googleapis.com/ajax/libs/
      angularjs/1.6.6/angular.min.js"></script>

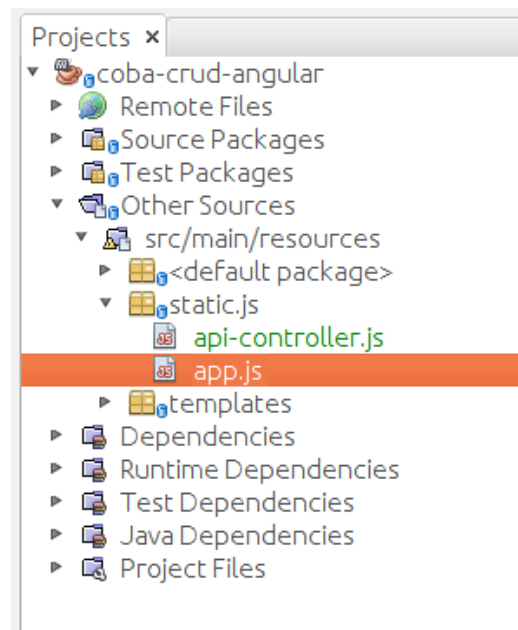
```

```

9      <script src="/js/app.js"></script>
10     <script src="/js/api-controller.js"></script>
11 </head>
12 <body ng-app="MahasiswaApp" class="container" >
13
14     <div class="jumbotron">
15         <h1>Daftar Mahasiswa</h1>
16     </div>
17
18     <div ng-controller="ApiController">
19
20         <table class="table table-striped">
21             <thead>
22                 <tr>
23                     <th scope="col">NIM</th>
24                     <th scope="col">NAMA</th>
25                     <th scope="col">JURUSAN</th>
26                 </tr>
27             </thead>
28             <tbody>
29                 <tr ng-repeat="mhs in daftarMahasiswa">
30                     <td>{{mhs.nim}}</td>
31                     <td>{{mhs.nama}}</td>
32                     <td>{{mhs.jurusan}}</td>
33                 </tr>
34             </tbody>
35         </table>
36     </div>
37 </body>
38 </html>
39

```

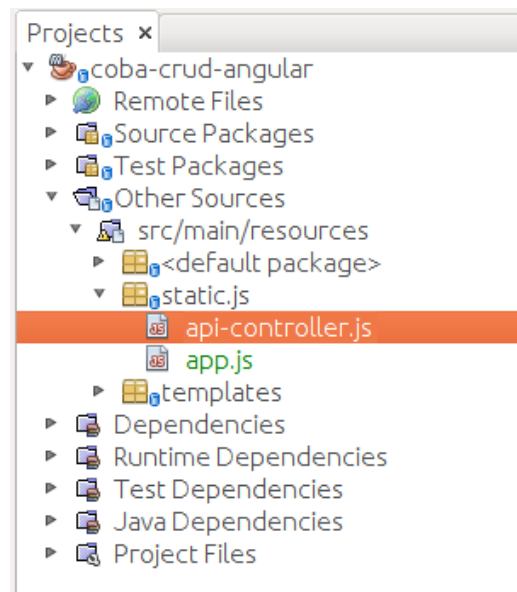
Kita coba beda satu per satu, dari baris ke-12, ada deklarasi **ng-app** dengan nilai **MahasiswaApp** yang sebetulnya diambilkan dari *app.js*, letak *file* dari **app.js** ini seperti ditunjukkan pada gambar 2.14 berikut :

Gambar 2.14: Lokasi *File* `app.js`

Sedangkan isi dari *file* `app.js` ini adalah sebagai berikut :

```
1 var app = angular.module('MahasiswaApp', []);  
2
```

Lalu pada baris ke-18 ada deklarasi `ng-controller` dengan nilai / isian `ApiController` yang sebetulnya diambilkan dari *file* `api-controller.js`. Lokasi *file* `api-controller.js` ini seperti ditunjukkan pada gambar 2.15 berikut :

Gambar 2.15: Lokasi *File* `api-controller.js`

Isi dari *file* ini adalah sebagai berikut :

```
1 app.controller('ApiController', function($scope, $http) {  
2     $scope.daftarMahasiswa = {};  
3  
4     $scope.updateDaftarMahasiswa = function() {  
5         $http.get('daftar-mahasiswa').then(sukses, gagal);  
6  
7         function sukses(response) {  
8             console.log(response);  
9             $scope.daftarMahasiswa = response.data;  
10        };  
11  
12        function gagal(response) {  
13            console.log(response);  
14        }  
15    };  
16  
17    $scope.updateDaftarMahasiswa();  
18 });  
19
```

Di *file* `api-controller` inilah nantinya *front-end* akan berkomunikasi dengan *back-end*. Pada `ApiController` milik Angular ini, pada baris ke-2 akan menyiapkan sebuah variabel `daftarMahasiswa` yang masih kosong.

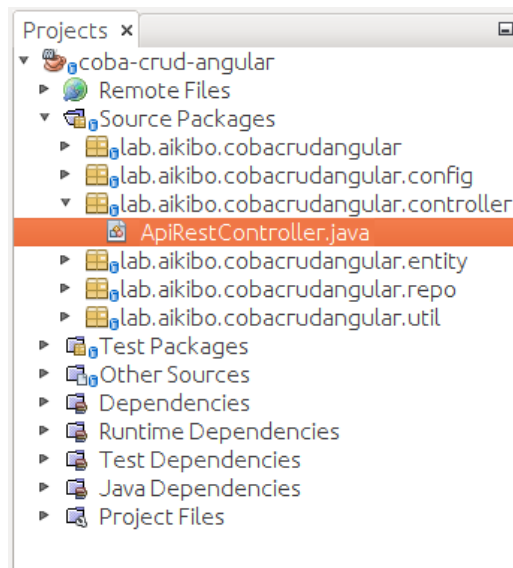
Kemudian pada baris ke-4 ada deklarasi fungsi dengan nama `updateDaftarMahasiswa`, dimana didalamnya melakukan *request* ke *server* dengan *method* `get` ke alamat `/daftar-mahasiswa`, yang diharapkan, pada saat sukses melakukan *request* dan menerima *response* yang benar, sehingga variabel `daftarMahasiswa` dapat diisi dengan data dari *server* seperti pada baris ke-9.

Setiap dilakukan *refresh* / *reload* halaman, maka akan dipanggil fungsi `updateDaftarMahasiswa` seperti pada baris ke-17.

Kita kembali lagi ke *file* `daftar-mahasiswa.html` pada baris ke-29, dimana disana ada atribut `ng-repeat` dengan isian `mhs` in `daftarMahasiswa`, nantinya Angular akan melakukan iterasi untuk setiap data pada `daftarMahasiswa` disimpan dalam variabel temporer `mhs`, yang kemudian ditampilkan di tiap baris seperti disebutkan pada baris ke-30, 31, dan 32.

Sampai sini *back-end* diharapkan dapat menyiapkan *mapping* ke *url* `/daftar-mahasiswa` dengan nilai kembalian yang didalamnya terdapat data larik dari daftar mahasiswa, untuk selanjutnya disimpan dalam variabel `daftarMahasiswa`.

2. Menyiapkan *controller* agar dapat melakukan *response* terhadap *url* `/daftar-mahasiswa`, *file controller* yang kita buat misalkan dengan nama `ApiRestController` dengan letak *file* berada seperti ditunjukkan pada gambar 2.16 berikut :



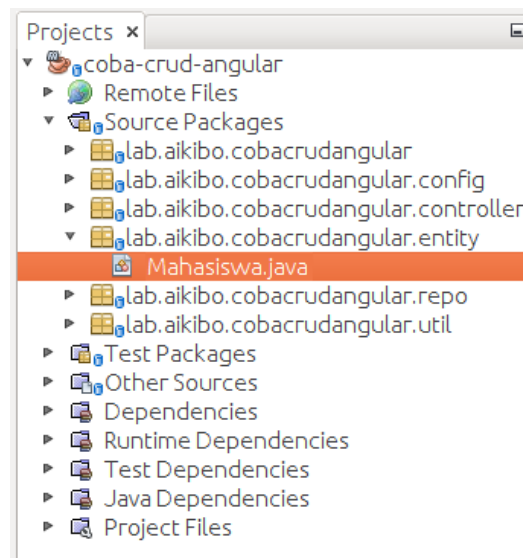
Gambar 2.16: Letak *File* `ApiRestController.java`

Isi dari kelas `ApiRestController` ini adalah sebagai berikut :

```
1 package lab.aikibo.cobacrudangular.controller;
2
3 import java.util.List;
4 import lab.aikibo.cobacrudangular.entity.Mahasiswa;
5 import lab.aikibo.cobacrudangular.repo.MahasiswaRepo;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.web.bind.annotation.
    RequestMapping;
8 import org.springframework.web.bind.annotation.
    RestController;
9
10 /**
11  *
12  * @author tamami <tamami.oka@gmail.com>
13  */
14 @RestController
15 public class ApiRestController {
16
17     @Autowired
18     private MahasiswaRepo mhsRepo;
19
20     @RequestMapping("/daftar-mahasiswa")
21     public List<Mahasiswa> getDaftarMahasiswa() {
22         return mhsRepo.findAll();
23     }
24 }
25
26
```

Hanya ada sebuah *method* dengan nama `getDaftarMahasiswa` dengan *mapping* ke url `/daftar-mahasiswa`. Isi dari *method* ini pun sangat sederhana, hanya memanggil *method* `findAll` dari `MahasiswaRepo`, objek inilah yang nantinya akan memberikan kita akses operasi terhadap basis data.

3. Membuat kelas `Mahasiswa` yang digunakan sebagai *mapping* kelas dari tabel di basis data, letak file `Mahasiswa.java` ini seperti ditunjukkan pada gambar 2.17 berikut :

Gambar 2.17: Letak *File Mahasiswa.java*

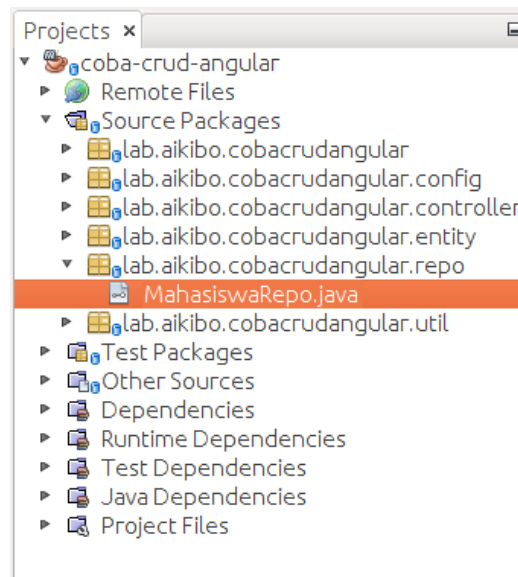
Isi dari *file Mahasiswa.java* ini adalah sebagai berikut :

```
1 package lab.aikibo.cobacrudangular.entity;  
2  
3 import javax.persistence.Column;  
4 import javax.persistence.Entity;  
5 import javax.persistence.Id;  
6 import lombok.Getter;  
7 import lombok.Setter;  
8  
9 /**  
10  *  
11  * @author tamami <tamami.oka@gmail.com>  
12  */  
13 @Entity  
14 public class Mahasiswa {  
15  
16     @Id @Getter @Setter  
17     private String nim;  
18  
19     @Column @Getter @Setter  
20     private String nama;  
21  
22     @Column @Getter @Setter  
23     private String jurusan;  
24  
25 }  
26
```

Kelas ini ditandai dengan anotasi `@Entity`, dan di dalamnya berisi atribut yang sesuai dengan nama kolom yang ada pada tabel di sistem basis data, dimana anotasi `@Id` menandakan *primary key* di sistem basis data, dan anotasi `@Column` tentunya menandakan atribut kolom di sistem basis data.

Kita juga memanfaatkan anotasi `@Getter` dan `@Setter` milik Lombok agar kode yang kita bangun lebih bersih dan mudah dibaca.

4. Selanjutnya kita membuat *file* dengan nama `MahasiswaRepo.java` yang nantinya ditugaskan untuk membuat *query* operasi terhadap sistem basis data. Letak *file* `MahasiswaRepo.java` ini seperti ditunjukkan pada gambar 2.18 berikut :



Gambar 2.18: Letak *File* `MahasiswaRepo.java`

Isi dari *file* ini juga cukup sederhana, berikut kodenya :

```
1 package lab.aikibo.cobacrudangular.repo;
2
3 import lab.aikibo.cobacrudangular.entity.Mahasiswa;
4 import org.springframework.data.jpa.repository.JpaRepository;
5     ;
6 import org.springframework.stereotype.Repository;
7
8 /**
9  *
10  * @author tamami <tamami.oka@gmail.com>
```



```

10 */
11 @Repository
12 public interface MahasiswaRepo extends JpaRepository<
    Mahasiswa, String> {
13
14 }
15

```

Interface ini ditandai dengan anotasi `@Repository` agar kita dapat melakukan *dependency injection* terhadap objek ini, yang implementasinya dapat dilihat di *file* `ApiRestController.java` pada baris ke-18, yaitu pada deklarasi objek `mhsRepo` dengan anotasi `@Autowired`.

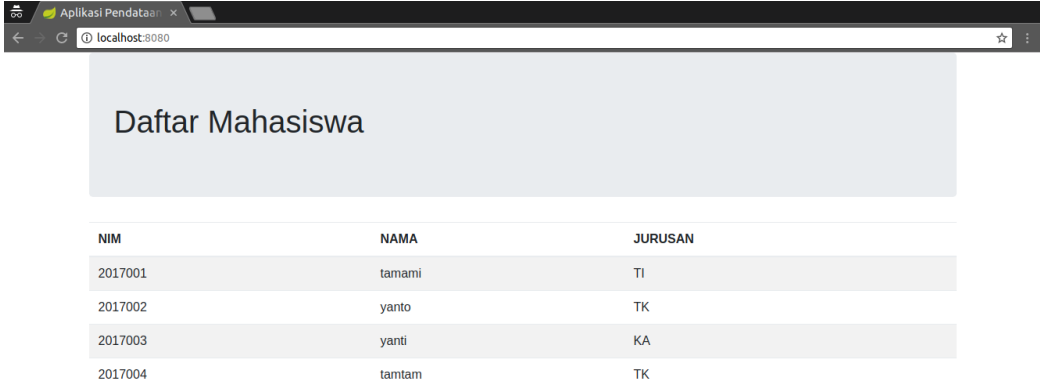
- Langkah berikutnya yang perlu dikerjakan adalah mendaftarkan halaman `daftar-mahasiswa.html` ke dalam daftar *view* pada kelas `WebConfig` dengan isi kode sebagai berikut :

```

1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.
    ViewControllerRegistry;
5 import org.springframework.web.servlet.config.annotation.
    WebMvcConfigurerAdapter;
6
7 @Configuration
8 public class WebConfig extends WebMvcConfigurerAdapter {
9
10     @Override
11     public void addViewControllers(ViewControllerRegistry
        registry) {
12         registry.addViewController("/login").setViewName("login
            ");
13         registry.addViewController("/test-angular").setViewName
            ("test-angular");
14         registry.addViewController("/")
15             .setViewName("daftar-mahasiswa");
16         registry.addViewController("/daftar-mahasiswa")
17             .setViewName("daftar-mahasiswa");
18     }
19 }
20

```

- Langkah terakhir adalah melakukan uji coba terhadap aplikasi yang kita bangun, setelah aplikasi dijalankan nantinya seperti biasa akan menampilkan halaman *login*, setelah berhasil, maka seharusnya halaman yang tampil adalah seperti pada gambar 2.19 berikut ini :



NIM	NAMA	JURUSAN
2017001	tamami	TI
2017002	yanto	TK
2017003	yanti	KA
2017004	tamtam	TK

Gambar 2.19: Halaman Daftar Mahasiswa

Sampai sini aplikasi *web* kita telah dilengkapi dengan fasilitas untuk menampilkan daftar Mahasiswa.

2.6 Tambah Data

Untuk menambahkan data, kita perlu sebuah halaman yang berbentuk seperti formulir agar *user* dapat mengisi informasi tentang data yang baru. Langkah-langkahnya adalah sebagai berikut :

1. Pertama kita berikan tombol di bawah daftar mahasiswa untuk membuka halaman formulir isian data mahasiswa baru, berikut perubahan yang kita lakukan di *file* `daftar-mahasiswa.html` :

```

1 <html>
2 <head>
3     <title>Aplikasi Pendataan Mahasiswa</title>
4     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
      integrity="sha384-
      PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
      " crossorigin="anonymous"/>
5     <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/
      rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
      anonymous"></script>

```

```

6      <script src="https://cdnjs.cloudflare.com/ajax/libs/
    popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
    vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
    " crossorigin="anonymous"></script>
7      <script src="https://maxcdn.bootstrapcdn.com/bootstrap
    /4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
    alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
    " crossorigin="anonymous"></script>
8      <script src="https://ajax.googleapis.com/ajax/libs/
    angularjs/1.6.6/angular.min.js"></script>
9      <script src="/js/app.js"></script>
10     <script src="/js/api-controller.js"></script>
11 </head>
12 <body ng-app="MahasiswaApp" class="container" >
13
14     <div class="jumbotron">
15         <h1>Daftar Mahasiswa</h1>
16     </div>
17
18     <div ng-controller="ApiController">
19
20         <table class="table table-striped">
21             <thead>
22                 <tr>
23                     <th scope="col">NIM</th>
24                     <th scope="col">NAMA</th>
25                     <th scope="col">JURUSAN</th>
26                 </tr>
27             </thead>
28             <tbody>
29                 <tr ng-repeat="mhs in daftarMahasiswa">
30                     <td>{{mhs.nim}}</td>
31                     <td>{{mhs.nama}}</td>
32                     <td>{{mhs.jurusan}}</td>
33                 </tr>
34             </tbody>
35             <tfoot>
36                 <tr>
37                     <td>
38                         <a class="btn btn-primary"
39                             href="/form" role="button">
40                             Tambah</a>
41                     </td>
42                 </tr>
43             </tfoot>
44         </table>
45     </div>
46 </body>
47 </html>

```

Perhatikan pada baris ke-38 dimana kita menambahkan element `a` yang melakukan *request* ke *server* dengan *url* `/form` dan menggunakan Bootstrap untuk menjadikan *link* ini seperti sebuah tombol.

- Hal berikutnya yang disiapkan adalah *file* `form.html` sebagai bahan *response* atas *request* sebelumnya. Letak *file* `form.html` ini sama seperti *file* `html` lainnya yaitu di dalam *folder* `templates`, isi kode dari *file* `form.html` ini adalah sebagai berikut :

```

1 <html>
2 <head>
3   <title>Form Entry Data</title>
4
5   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
6     integrity="sha384-
7     PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszUE4W1povHYgTpBfshb
8     " crossorigin="anonymous"/>
9   <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
10     KJ3o2DKtIkvYIK3UENzmM7KCkRr/
11     rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
12     anonymous"></script>
13   <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
14     vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
15     " crossorigin="anonymous"></script>
16   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
17     alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
18     " crossorigin="anonymous"></script>
19   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
20   <script src="/js/app.js"></script>
21   <script src="/js/form-controller.js"></script>
22 </head>
23 <body ng-app="MahasiswaApp">
24   <div class="container" ng-controller="FormController">
25     <div class="jumbotron">
26       <h1>Formulir Isian</h1>
27     </div>
28
29     <form>
30       <div class="form-group row">
31         <label class="col-sm-2 col-form-label">NIM</label>
32         <input class="form-control" type="text" placeholder
33           ="NIM" ng-model="dataMahasiswa.nim" />
34       </div>
35       <div class="form-group row">
36         <label class="col-sm-2 col-form-label">Nama</label>

```

```

26     <input class="form-control" type="text" placeholder
    ="Nama" ng-model="dataMahasiswa.nama" />
27   </div>
28   <div class="form-group row">
29     <label class="col-sm-2 col-form-label">Jurusan</
    label>
30     <input class="form-control" type="text" placeholder
    ="Usia" ng-model="dataMahasiswa.jurusan" />
31   </div>
32   <div class="form-group row">
33     <button class="btn btn-primary" ng-click="simpan()">
    Simpan</button>
34   </div>
35 </form>
36 </div>
37 </body>
38 </html>
39

```

Seperti pada *file* `daftar-mahasiswa.html`, disini kita juga menyertakan parameter `ng-app` dengan isi `MahasiswaApp` seperti pada baris ke-13 berikut :

```

13 <body ng-app="MahasiswaApp">

```

Kemudian pada baris ke-14 kita mengaitkan komponen `<div>` dengan *controller* Angular `FormController` seperti berikut :

```

14 <div class="container" ng-controller="FormController">

```

Selanjutnya untuk tiap komponen `<input>` kita kaitkan dengan `ng-model` agar *controller* dapat membaca isinya seperti pada baris ke-22, ke-26, dan ke-30 seperti berikut :

```

22 <input class="form-control" type="text" placeholder="NIM" ng
    -model="dataMahasiswa.nim" />
    :
    :
26 <input class="form-control" type="text" placeholder="NAMA"
    ng-model="dataMahasiswa.nama" />
    :
    :
30 <input class="form-control" type="text" placeholder="JURUSAN
    " ng-model="dataMahasiswa.jurusan" />

```

3. Berikutnya kita buat *controller* di *front-end* dengan nama `FormController` untuk menangani *view* atau *file* `form.html`. Lokasi dari *file* `form-controller.js` sama seperti *file* `api-controller.js`, yaitu di dalam *folder* `static/js`, isi kode dari *file* ini adalah sebagai berikut :

```

1 app.controller('FormController', function($scope, $http,
2   $window) {
3
4   $scope.dataMahasiswa = {};
5
6   $scope.simpan = function() {
7     $http.post('tambah-data', $scope.dataMahasiswa).then
8     (sukses, gagal);
9
10    function sukses(response) {
11      $window.location.href = '/';
12    };
13
14    function gagal(response) {};
15  }
16 });

```

Pada *controller* ini, kita membutuhkan komponen **\$scope** untuk melakukan *binding* data antara *file html* dengan *controller*, komponen lainnya adalah **\$http** yang nantinya kita gunakan untuk melakukan komunikasi antara *front-end* dengan *back-end*, dan **\$window** yang kita gunakan untuk membaca *url* atau mengganti halaman *browser* yang aktif.

Pada baris ke-2 kita menyiapkan sebuah variabel **dataMahasiswa** yang pada halaman **form.html** akan diisikan atau di *binding* melalui atribut **ng-model**.

Pada baris ke-4 kita membuat fungsi **simpan** yang nantinya akan dipanggil dari halaman **form.html** pada baris ke-33 seperti berikut :

```

33 <button class="btn btn-primary" ng-click="simpan()">Simpan</
    button>

```

Kemudian pada baris ke-5, *controller front-end* mulai mengkomunikasikan dengan *back-end* melalui *url tambah-data* dengan *method* POST, apabila *request* ini berhasil, maka akan dikerjakan fungsi **sukses** yang berada di bawahnya, namun bila gagal, maka akan dikerjakan fungsi **gagal**.

4. Berikutnya adalah merubah *controller* di *back-end* agar siap melakukan *response* atas *request* terhadap *url tambah-data*. Berikut adalah perubahan kode yang kita lakukan terhadap *file ApiRestController.java* :

```

1 package lab.aikibo.cobacrudangular.controller;
2
3 import java.util.List;
4 import lab.aikibo.cobacrudangular.entity.Mahasiswa;

```

```

5 import lab.aikibo.cobacrudangular.repo.MahasiswaRepo;
6 import lab.aikibo.cobacrudangular.repo.MahasiswaRepoPaging;
7 import org.springframework.beans.factory.annotation.
    Autowired;
8 import org.springframework.data.domain.Page;
9 import org.springframework.data.domain.Pageable;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.
    RequestMapping;
12 import org.springframework.web.bind.annotation.RequestMethod
    ;
13 import org.springframework.web.bind.annotation.
    RestController;
14
15 /**
16  *
17  * @author tamami <tamami.oka@gmail.com>
18  */
19 @RestController
20 public class ApiRestController {
21
22     @Autowired
23     private MahasiswaRepo mhsRepo;
24
25     @Autowired
26     private MahasiswaRepoPaging mhsRepoPaging;
27
28     @RequestMapping("/daftar-mahasiswa")
29     public List<Mahasiswa> getDaftarMahasiswa() {
30         return mhsRepo.findAll();
31     }
32
33     @RequestMapping("/daftar-mahasiswa-with-paging")
34     public Page<Mahasiswa> getDaftarMahasiswaWithPaging(
35         Pageable pageable) {
36         return mhsRepoPaging.findAll(pageable);
37     }
38
39     @RequestMapping(value = "/tambah-data",
40         method = RequestMethod.POST)
41     public void tambahData(
42         @RequestBody Mahasiswa mhs) {
43         mhsRepo.save(mhs);
44     }
45 }

```

5. Langkah berikutnya adalah menambahkan *view form.html* ke dalam

kelas `WebConfig` agar *server* dapat merespon ketika tombol tambah diklik, berikut adalah perubahan kode yang terjadi pada kelas `WebConfig` :

```
1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.
    ViewControllerRegistry;
5 import org.springframework.web.servlet.config.annotation.
    WebMvcConfigurerAdapter;
6
7 @Configuration
8 public class WebConfig extends WebMvcConfigurerAdapter {
9
10     @Override
11     public void addViewControllers(ViewControllerRegistry
        registry) {
12         registry.addViewController("/login").setViewName("login
            ");
13         registry.addViewController("/test-angular").setViewName
            ("test-angular");
14         registry.addViewController("/").setViewName("daftar-
            mahasiswa");
15         registry.addViewController("/daftar-mahasiswa").
            setViewName("daftar-mahasiswa");
16         registry.addViewController("/form").setViewName("form");
17     }
18 }
19
20
```

6. Melakukan ujicoba, seharusnya saat tombol tambah pada `daftar-mahasiswa` kita klik, maka akan muncul jendela *entry* data seperti pada gambar 2.20 berikut :



The screenshot shows a web browser window with the title 'Form Entry Data'. The address bar shows 'localhost:8080/form'. The main content area has a light blue header with the text 'Formulir Isian'. Below the header, there are four input fields: 'NIM', 'Nama', 'Jurusan', and 'Usia'. Each field has a small label above it. At the bottom of the form, there is a blue button labeled 'Simpan'.

Gambar 2.20: Tampilan Halaman *Entry* Data

7. Kemudian saat melakukan simpan data, seharusnya datanya tersimpan dan halaman kembali ke **daftar-mahasiswa** dengan penambahan data dari operasi sebelumnya.

Sampai sini usaha kita untuk memberikan fasilitas tambah data selesai.

2.7 Ubah Data

Pada pengubahan data ini, konsepnya adalah menampilkan formulir isian seperti tambah data, hanya saja isian datanya sudah terisi dengan data sebelumnya, *user* atau pengguna hanya mengganti isiannya yang dirasa perlu diperbaiki / diubah, setelah itu melakukan simpan data seperti tambah data dan layar dikembalikan ke **daftar-mahasiswa** untuk menampilkan hasil perubahannya. Berikut adalah langkahnya :

1. Melakukan perubahan pada halaman **daftar-mahasiswa** agar memiliki tombol **Ubah**. Perubahan yang dilakukan pada *file* ini adalah sebagai berikut :

```
1 <html>
2 <head>
3   <title>Aplikasi Pendataan Mahasiswa</title>
```

```

4   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
    integrity="sha384-
    PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
    " crossorigin="anonymous"/>
5   <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
    KJ3o2DKtIkVYIK3UENzmM7KChRr/
    rE9/Qpg6A7GJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
    anonymous"></script>
6   <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
    vFJXuSjphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
    " crossorigin="anonymous"></script>
7   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
    alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
    " crossorigin="anonymous"></script>
8   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
9   <script src="/js/app.js"></script>
10  <script src="/js/api-controller.js"></script>
11 </head>
12 <body ng-app="MahasiswaApp" class="container" >
13
14   <div class="jumbotron">
15     <h1>Daftar Mahasiswa</h1>
16   </div>
17
18   <div ng-controller="ApiController">
19
20     <table class="table table-striped">
21       <thead>
22         <tr>
23           <th scope="col">NIM</th>
24           <th scope="col">NAMA</th>
25           <th scope="col">JURUSAN</th>
26         </tr>
27       </thead>
28       <tbody>
29         <tr ng-repeat="mhs in daftarMahasiswa">
30           <td>{{mhs.nim}}</td>
31           <td>{{mhs.nama}}</td>
32           <td>{{mhs.jurusan}}</td>
33           <td>
34             <button class="btn btn-warning"
35               ng-click="ubah(mhs)">
36               Ubah
37             </button>
38           </td>

```

```

39         </tr>
40     </tbody>
41 </tfoot>
42 <tr>
43     <td>
44         <a class="btn btn-primary" href="/
form" role="button">Tambah</a>
45     </td>
46 </tr>
47 </tfoot>
48 </table>
49 </div>
50 </body>
51 </html>
52

```

Pada baris ke-33 sampai dengan baris ke-38 adalah tambahan kode kita agar nantinya di tiap baris data yang muncul pada *browser* akan muncul pula tombol ubah untuk mengubah data. Perhatikan bahwa pada baris ke-35 ada parameter `ng-click` yang berisi atau memanggil fungsi `ubah` yang melewatkan data `mhs` yang terpilih.

2. Mengubah *file* `api-controller.js` agar memiliki fungsi `ubah`. Fungsi `ubah` ini nantinya hanya melakukan *request* terhadap halaman `edit-form` dengan disertai parameter `nim` yang berisi NIM dari data mahasiswa terpilih. Berikut adalah perubahan kode yang kita lakukan terhadap *file* `api-controller.js` :

```

1 app.controller('ApiController', function($scope, $http,
    $window) {
2     $scope.daftarMahasiswa = {};
3
4     $scope.updateDaftarMahasiswa = function() {
5         $http.get('daftar-mahasiswa').then(sukses, gagal);
6         // $http.get('daftar-mahasiswa-with-paging').then(
        sukses, gagal);
7
8         function sukses(response) {
9             console.log(response);
10            $scope.daftarMahasiswa = response.data;
11            // console.log(response.data.content);
12            // $scope.daftarMahasiswa = response.data.content;
13        };
14
15        function gagal(response) {
16            console.log(response);
17        }
18    };

```

```

19
20     $scope.ubah = function(mhs) {
21         $window.location.href = 'edit-form?nim=' + mhs.nim;
22     };
23
24     $scope.updateDaftarMahasiswa();
25 });
26

```

Seharusnya nanti halaman yang tampil akan berubah dari `daftar-mahasiswa.html` menjadi `edit-form.html`.

- Langkah berikutnya adalah membuat halaman `edit-form.html` agar siap menampilkan *response* dari *request* pada langkah sebelumnya. Letak *file* `edit-form.html` ini sama seperti *file* `daftar-mahasiswa.html`, berada dalam *folder* `templates`. Isi kode dari *file* `edit-form.html` ini adalah seperti berikut :

```

1 <html>
2 <head>
3     <title>Formulir Ubah Data</title>
4
5     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
6         integrity="sha384-
7         PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
8         " crossorigin="anonymous"/>
9     <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
10         KJ3o2DKtIkVYIK3UENzmM7KChRr/
11         rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
12         anonymous"></script>
13     <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.3/dist/umd/popper.min.js" integrity="sha384-
14         vFJXUuSjphROIrBnz7yo7oB41mKfc8JzZqZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
15         " crossorigin="anonymous"></script>
16     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
17         alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXM3XxPqe5iKTfUKjNkCk9SaVuEZflJ
18         " crossorigin="anonymous"></script>
19     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
20     <script src="/js/app.js"></script>
21     <script src="/js/edit-form-controller.js"></script>
22 </head>
23 <body class="container" ng-app="MahasiswaApp">
24     <div class="jumbotron">
25         <h1>Formulir Ubah Data</h1>
26     </div>
27

```

```

18     <div ng-controller="EditFormController">
19 <form>
20     <div class="form-group row">
21         <label class="col-sm-2 col-form-label">NIM</label>
22         <input class="form-control"
23             type="text"
24             placeholder="NIM"
25             ng-model="dataMahasiswa.nim"
26             readonly="true"/>
27     </div>
28     <div class="form-group row">
29         <label class="col-sm-2 col-form-label">Nama</label>
30         <input class="form-control" type="text" placeholder
31             ="Nama" ng-model="dataMahasiswa.nama"/>
32     </div>
33     <div class="form-group row">
34         <label class="col-sm-2 col-form-label">Jurusan</
35         label>
36         <input class="form-control" type="text" placeholder
37             ="Usia" ng-model="dataMahasiswa.jurusan" />
38     </div>
39     <div class="form-group row">
40         <button class="btn btn-primary" ng-click="simpan()">
41             Simpan</button>
42     </div>
43 </form>
44 </div>
45 </body>
46 </html>

```

Terlihat mirip dengan *file form.html*, namun ada beberapa perbedaan bila kita teliti lebih jauh, perbedaan yang pertama adalah penggunaan *controller* `EditFormController` seperti disebutkan pada baris ke-18 berikut :

```

18 <div ng-controller="EditFormController">

```

Controller ini diambilkan dari *file edit-form-controller.js* seperti terlihat pada baris ke-11 berikut :

```

11 <script src="/js/edit-form-controller.js">

```

Kemudian menjadikan isian untuk NIM menjadi *read-only* seperti perintah pada baris ke-26 berikut :

```

26     readonly="true" />

```

4. Membuat *file* `edit-form-controller.js` agar dapat membaca *url* yang dikirimkan dan mengambil informasi NIM yang terdapat didalamnya. Letak *file* `edit-form-controller.js` ini sama seperti *controller* lainnya, yaitu berada di *folder* `static/js/`. Berikut adalah isi kode untuk *file* tersebut :

```

1 app.controller('EditFormController', function($scope,
2   $window, $http) {
3
4   $scope.updateData = function() {
5     nim = $window.location.search.split('?')[1].split
6     ('=')[1];
7     $scope.dataMahasiswa.nim = nim;
8     $http.get('/get-mahasiswa-by-nim/' + nim).then(
9       sukses, gagal);
10
11     function sukses(response) {
12       $scope.dataMahasiswa.nama = response.data.nama;
13       $scope.dataMahasiswa.jurusan = response.data.
14       jurusan;
15     };
16
17     function gagal(response) {};
18
19     $scope.simpan = function() {
20       $http.post('/simpan-edit-data', $scope.dataMahasiswa
21       ).then(sukses, gagal);
22
23       function sukses(response) {
24         $window.location.href = '/';
25       };
26
27       function gagal(response) {}
28     };
29
30     $scope.updateData();
31   });

```

Pada *controller* ini, kita membutuhkan beberapa modul seperti `$scope`, `$window`, dan `$http`. Di dalam *controller* ini, ada objek `dataMahasiswa` yang fungsinya adalah menyimpan hasil *entry* data pada halaman `edit-form.html`, kemudian ada sebuah fungsi `updateData` yang selalu dijalankan apabila `edit-form` ini dibuka, atau dilakukan *refresh*.

Isi dari fungsi `updateData` ini yaitu melakukan *parsing* dari *url request*

yang terletak di *address browser* seperti pada baris ke-5, kemudian mengisi parameter **nim** milik **dataMahasiswa** agar tampilan *input* di halaman *web* terbaru seperti pada baris ke-6.

Pada baris ke-7 melakukan *request* dengan *method* GET ke *back-end service* dengan sebuah parameter **nim**. Apabila berhasil, data **nama** dan **jurusan** dari *back-end service* akan ditampilkan dalam komponen yang semestinya.

Pada baris ke-17, kita menyiapkan fungsi **simpan** yang didalamnya (pada baris ke-18) melakukan *request* data dengan metode POST dengan variabel **dataMahasiswa** sebagai parameternya.

5. Mengubah kelas **ApiRestController** agar dapat melakukan respon terhadap *request* **/get-mahasiswa-by-nim** dengan parameter **nim**. Isi kode dari kelas **ApiRestController** adalah seperti berikut ini :

```

1 package lab.aikibo.cobacrudangular.controller;
2
3 import java.util.List;
4 import lab.aikibo.cobacrudangular.entity.Mahasiswa;
5 import lab.aikibo.cobacrudangular.repo.MahasiswaRepo;
6 import lab.aikibo.cobacrudangular.repo.MahasiswaRepoPaging;
7 import org.springframework.beans.factory.annotation.
    Autowired;
8 import org.springframework.data.domain.Page;
9 import org.springframework.data.domain.Pageable;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.
    RequestMapping;
13 import org.springframework.web.bind.annotation.RequestMethod
    ;
14 import org.springframework.web.bind.annotation.
    RestController;
15
16 /**
17  *
18  * @author tamami <tamami.oka@gmail.com>
19  */
20 @RestController
21 public class ApiRestController {
22
23     @Autowired
24     private MahasiswaRepo mhsRepo;
25
26     @Autowired
27     private MahasiswaRepoPaging mhsRepoPaging;
28

```

```

29     @RequestMapping("/daftar-mahasiswa")
30     public List<Mahasiswa> getDaftarMahasiswa() {
31         return mhsRepo.findAll();
32     }
33
34     @RequestMapping("/daftar-mahasiswa-with-paging")
35     public Page<Mahasiswa> getDaftarMahasiswaWithPaging(
36         Pageable pageable) {
37         return mhsRepoPaging.findAll(pageable);
38     }
39
40     @RequestMapping(value = "/tambah-data", method =
41         RequestMethod.POST)
42     public void tambahData(@RequestBody Mahasiswa mhs) {
43         mhsRepo.save(mhs);
44     }
45
46     @RequestMapping(value = "/get-mahasiswa-by-nim/{nim}")
47     public Mahasiswa getMahasiswaByNim(
48         @PathVariable("nim") String nim) {
49         return mhsRepo.findOne(nim);
50     }
51
52     @RequestMapping(value = "/simpan-edit-data",
53         method = RequestMethod.POST)
54     public void simpanEditData(@RequestBody Mahasiswa mhs) {
55         mhsRepo.save(mhs);
56     }

```

Pada baris ke-47, kita hanya melihat pemanggilan sebuah fungsi `findOne` dengan parameter `nim` yang dipanggil dari *interface repository* yang telah kita buat, yang artinya adalah mengambil sebuah *record* dari basis data.

Pada baris ke-50 sampai dengan baris ke-54 sebetulnya sama saja dengan *method* `tambahData` di atasnya, yaitu hanya menyimpan objek Mahasiswa ke dalam sistem basis data.

6. Mendaftarkan *view* `edit-form` ke Spring MVC pada kelas `WebConfig` agar dapat dikenali halaman `edit-form` yang dimaksud, berikut adalah hasil perubahan kode yang terjadi pada *file* `WebConfig` :

```

1 package lab.aikibo.cobacrudangular.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.web.servlet.config.annotation.
    ViewControllerRegistry;

```



```
5 import org.springframework.web.servlet.config.annotation.  
    WebMvcConfigurerAdapter;  
6  
7 @Configuration  
8 public class WebConfig extends WebMvcConfigurerAdapter {  
9  
10     @Override  
11     public void addViewControllers(ViewControllerRegistry  
        registry) {  
12         registry.addViewController("/login").setViewName("login  
            ");  
13         registry.addViewController("/test-angular").setViewName  
            ("test-angular");  
14         registry.addViewController("/").setViewName("daftar-  
            mahasiswa");  
15         registry.addViewController("/daftar-mahasiswa").  
            setViewName("daftar-mahasiswa");  
16         registry.addViewController("/form").setViewName("form");  
17  
18         registry.addViewController("/edit-form").setViewName("edit-form");  
19     }  
20 }  
21
```

7. Mencoba *feature* yang telah jadi. Seharusnya saat dipilihkan tombol Ubah untuk mengubah suatu dari basis data, maka seharusnya akan tampil informasi lengkap data mahasiswa seperti pada gambar 2.21 berikut :

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/edit-form?nim=2017004'. The page title is 'Formulir Ubah Data'. The form contains three input fields: 'NIM' with the value '2017004', 'Nama' with the value 'tamtam', and 'Jurusan' with the value 'TK'. Below the input fields is a blue button labeled 'Simpan'.

Gambar 2.21: Halaman Untuk `edit-form.html`

Sampai sini, apabila tombol disimpan maka halaman akan kembali ke `daftar-mahasiswa` dengan menyertakan hasil perubahannya.

2.8 Hapus Data

Kali ini kita akan menambahkan fasilitas untuk melakukan hapus data, skenarionya akan mirip dengan *edit* data, dimana nanti dari daftar mahasiswa, akan dilakukan *request* dengan parameter NIM. Berikut langkah kodenya :

1. Mengubah / menambahkan tombol Hapus di *file* `daftar-mahasiswa.html`. Perubahan kode yang dilakukan adalah sebagai berikut :

```

1 <html>
2 <head>
3     <title>Aplikasi Pendataan Mahasiswa</title>
4     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css"
      integrity="sha384-
      PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb
      " crossorigin="anonymous"/>
5     <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/
      rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="
      anonymous"></script>

```

```

6      <script src="https://cdnjs.cloudflare.com/ajax/libs/
    popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
    vFJXuSjphROlrBnz7yo7oB4lmKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh
    " crossorigin="anonymous"></script>
7      <script src="https://maxcdn.bootstrapcdn.com/bootstrap
    /4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-
    alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXm3XxPqe5iKTfUKjNkCk9SaVuEZflJ
    " crossorigin="anonymous"></script>
8      <script src="https://ajax.googleapis.com/ajax/libs/
    angularjs/1.6.6/angular.min.js"></script>
9      <script src="/js/app.js"></script>
10     <script src="/js/api-controller.js"></script>
11 </head>
12 <body ng-app="MahasiswaApp" class="container" >
13
14     <div class="jumbotron">
15         <h1>Daftar Mahasiswa</h1>
16     </div>
17
18     <div ng-controller="ApiController">
19
20         <table class="table table-striped">
21             <thead>
22                 <tr>
23                     <th scope="col">NIM</th>
24                     <th scope="col">NAMA</th>
25                     <th scope="col">JURUSAN</th>
26                 </tr>
27             </thead>
28             <tbody>
29                 <tr ng-repeat="mhs in daftarMahasiswa">
30                     <td>{{mhs.nim}}</td>
31                     <td>{{mhs.nama}}</td>
32                     <td>{{mhs.jurusan}}</td>
33                     <td>
34                         <button class="btn btn-warning" ng-
    click="ubah(mhs)">
35                             Ubah
36                         </button>
37                     </td>
38                     <td>
39                         <button class="btn btn-danger"
40                             ng-click="hapus(mhs)">
41                             Hapus
42                         </button>
43                     </td>
44                 </tr>
45             </tbody>
46             <tfoot>

```

```

47         <tr>
48             <td>
49                 <a class="btn btn-primary" href="/
form" role="button">Tambah</a>
50             </td>
51         </tr>
52     </tfoot>
53 </table>
54 </div>
55 </body>
56 </html>
57

```

Pada baris ke-40, ada parameter `ng-click` yang akan memanggil fungsi `hapus` dengan parameter `mhs` yang terpilih.

2. Merubah *file* `api-controller.js` agar memiliki fungsi `hapus`. Berikut adalah perubahan atau penambahan kode yang terjadi di *file* `api-controller.js` :

```

1 app.controller('ApiController', function($scope, $http,
    $window) {
2     $scope.daftarMahasiswa = {};
3
4     $scope.updateDaftarMahasiswa = function() {
5         $http.get('daftar-mahasiswa').then(sukses, gagal);
6         // $http.get('daftar-mahasiswa-with-paging').then(
        sukses, gagal);
7
8         function sukses(response) {
9             console.log(response);
10            $scope.daftarMahasiswa = response.data;
11            // console.log(response.data.content);
12            // $scope.daftarMahasiswa = response.data.content;
13        };
14
15        function gagal(response) {
16            console.log(response);
17        }
18    };
19
20    $scope.ubah = function(mhs) {
21        $window.location.href = 'edit-form?nim=' + mhs.nim;
22    };
23
24    $scope.hapus = function(mhs) {
25        $http.delete('/hapus/' + mhs.nim).then(sukses, gagal);
26
27        function sukses(response) {

```

```

28         $scope.updateDaftarMahasiswa();
29     };
30
31     function gagal(response) {}
32 };
33
34 $scope.updateDaftarMahasiswa();
35 });
36

```

3. Merubah kelas `ApiRestController` agar dapat melakukan *response* terhadap *request* `/hapus/{nim}`. Perubahan yang kita lakukan adalah sebagai berikut :

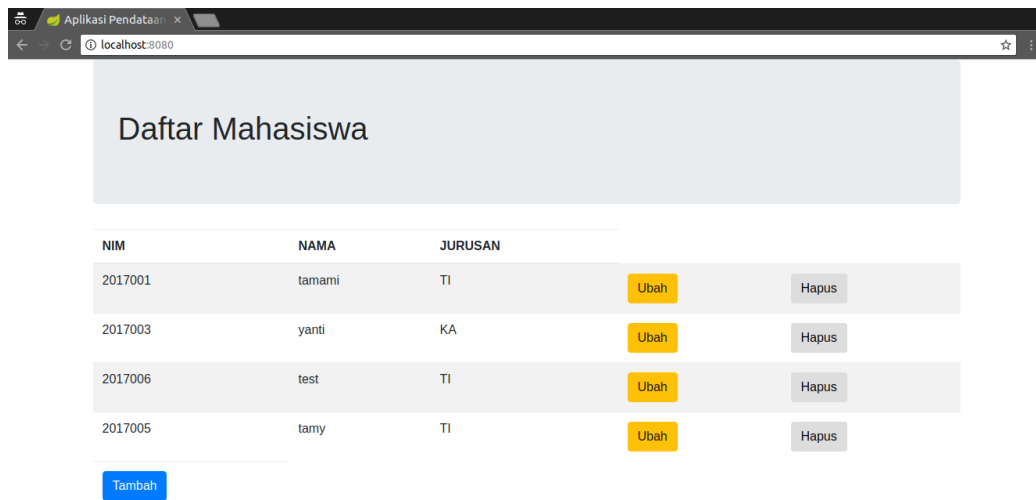
```

1 package lab.aikibo.cobacrudangular.controller;
2
3 import java.util.List;
4 import lab.aikibo.cobacrudangular.entity.Mahasiswa;
5 import lab.aikibo.cobacrudangular.repo.MahasiswaRepo;
6 import lab.aikibo.cobacrudangular.repo.MahasiswaRepoPaging;
7 import org.springframework.beans.factory.annotation.
    Autowired;
8 import org.springframework.data.domain.Page;
9 import org.springframework.data.domain.Pageable;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.
    RequestMapping;
13 import org.springframework.web.bind.annotation.RequestMethod
    ;
14 import org.springframework.web.bind.annotation.
    RestController;
15
16 /**
17  *
18  * @author tamami <tamami.oka@gmail.com>
19  */
20 @RestController
21 public class ApiRestController {
22
23     @Autowired
24     private MahasiswaRepo mhsRepo;
25
26     @Autowired
27     private MahasiswaRepoPaging mhsRepoPaging;
28
29     @RequestMapping("/daftar-mahasiswa")
30     public List<Mahasiswa> getDaftarMahasiswa() {
31         return mhsRepo.findAll();
32     }
33
34 }
35

```

```
32     }
33
34     @RequestMapping("/daftar-mahasiswa-with-paging")
35     public Page<Mahasiswa> getDaftarMahasiswaWithPaging(
36         Pageable pageable) {
37         return mhsRepoPaging.findAll(pageable);
38     }
39
40     @RequestMapping(value = "/tambah-data", method =
41         RequestMethod.POST)
42     public void tambahData(@RequestBody Mahasiswa mhs) {
43         mhsRepo.save(mhs);
44     }
45
46     @RequestMapping(value = "/get-mahasiswa-by-nim/{nim}")
47     public Mahasiswa getMahasiswaByNim(@PathVariable("nim")
48         String nim) {
49         return mhsRepo.findOne(nim);
50     }
51
52     @RequestMapping(value = "/simpan-edit-data", method =
53         RequestMethod.POST)
54     public void simpanEditData(@RequestBody Mahasiswa mhs) {
55         mhsRepo.save(mhs);
56     }
57
58     @RequestMapping(value = "/hapus/nim",
59         method = RequestMethod.DELETE)
60     public void deleteData(@PathVariable("nim") String nim) {
61         mhsRepo.delete(nim);
62     }
63 }
```

4. Melakukan uji coba. Gambar ?? adalah hasil tampilan dari aplikasi yang telah memiliki fasilitas untuk hapus data :



Gambar 2.22: Halaman Daftar Mahasiswa Dengan Fasilitas Hapus Data

Sampai sini aplikasi *web* yang telah kita bangun memiliki fasilitas yang lengkap yang terdiri dari penambahan data, pengubahan data, dan penghapusan data.