

MODUL OBJECT ORIENTED PROGRAMMING 2

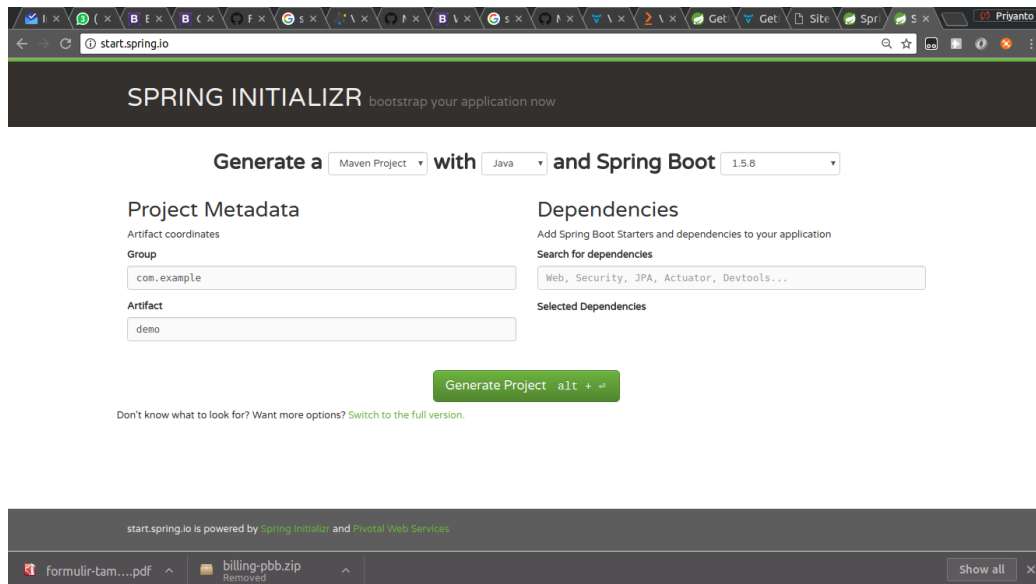
tamami

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 :

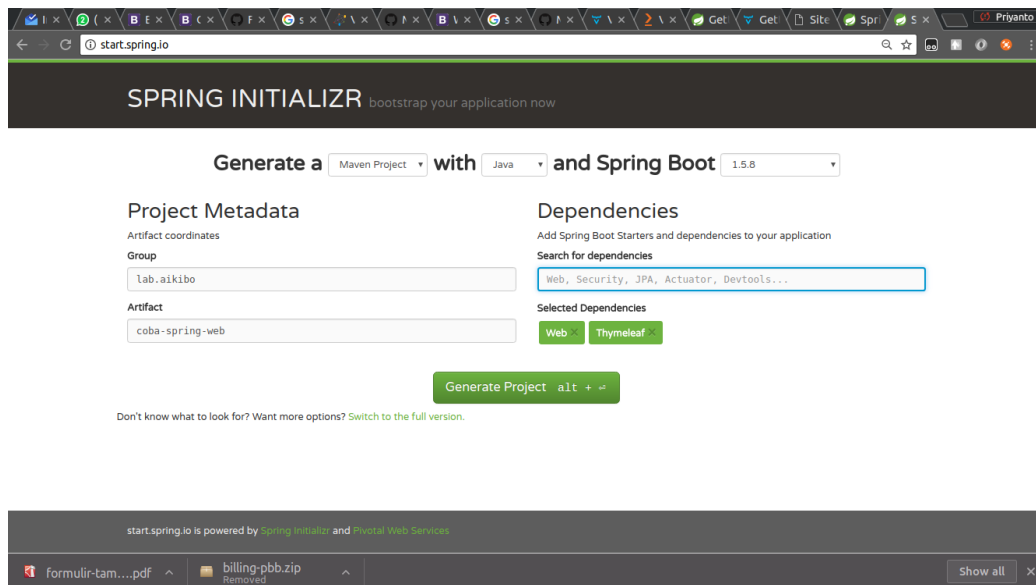
- **Group** yang diisi dengan nama atau identitas organisasi, instansi, atau tim yang mengembangkan aplikasi. Pengisian disini biasanya akan dijadikan nama paket dalam kandar (*folder*) kode sumber (*source code*).
- **Artifact** yang diisi dengan nama aplikasinya. Sebaiknya jangan menggunakan titik 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.

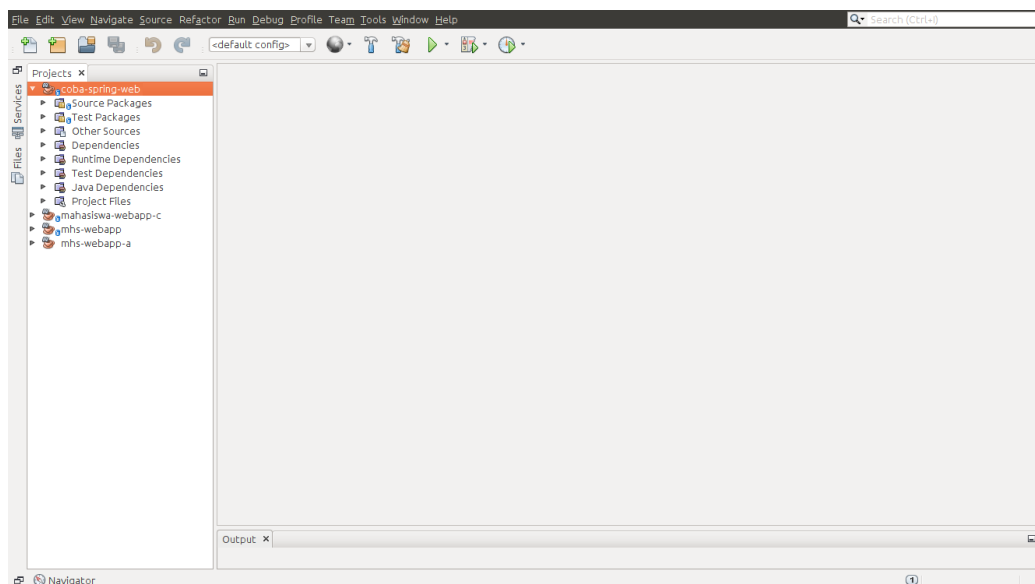
1.1 Uji Coba Pertama

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 :

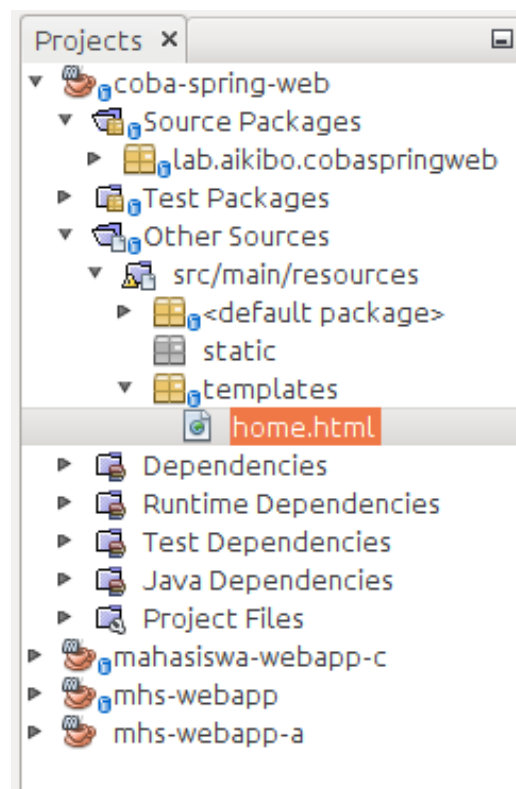
Gambar 1.3: Struktur Direktori *Project*

- 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 :

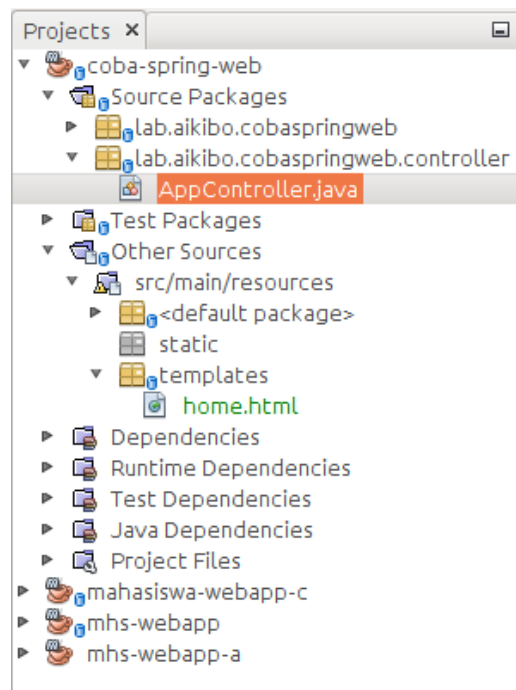
```
1 <html>
2   <head>
3     <title>Aplikasiku</title>
4   </head>
5   <body>
6     <h1>Selamat Datang</h1>
7   </body>
8 </html>
9
```

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 :

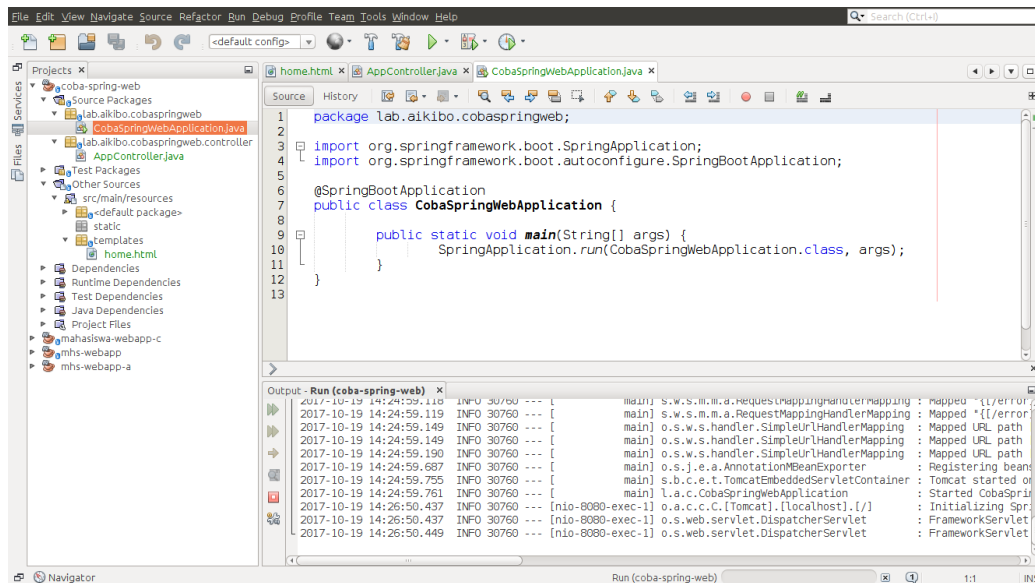
Gambar 1.5: Posisi *File* `AppController.java`

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

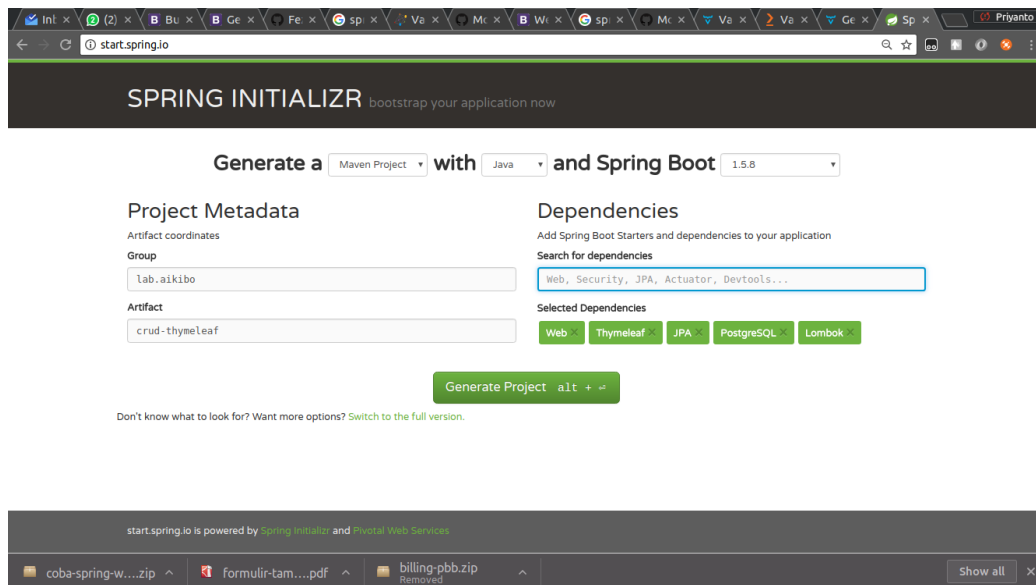
Sebelum kita tampilkan datanya, datanya harus disiapkan terlebih dahulu. Struktur data yang perlu dibentuk adalah sebagai berikut :

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.

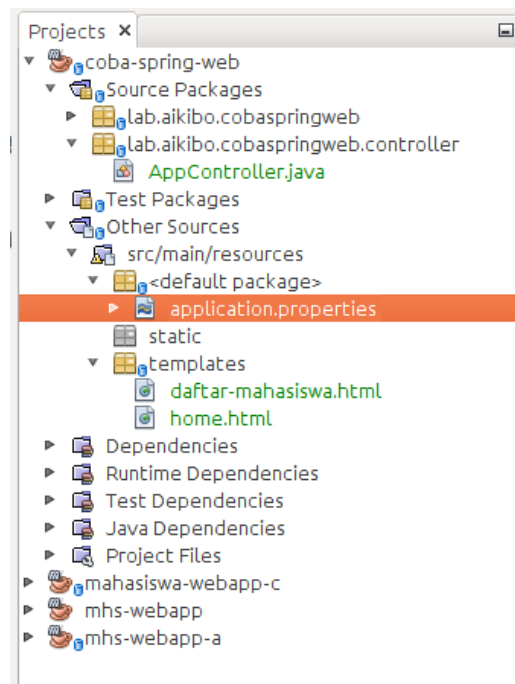
Untuk mempermudah langkah kita membangun sebuah aplikasi ini, kita kunjungi lagi alamat `start.spring.io` dengan beberapa pustaka seperti pada gambar 1.8 :

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 :

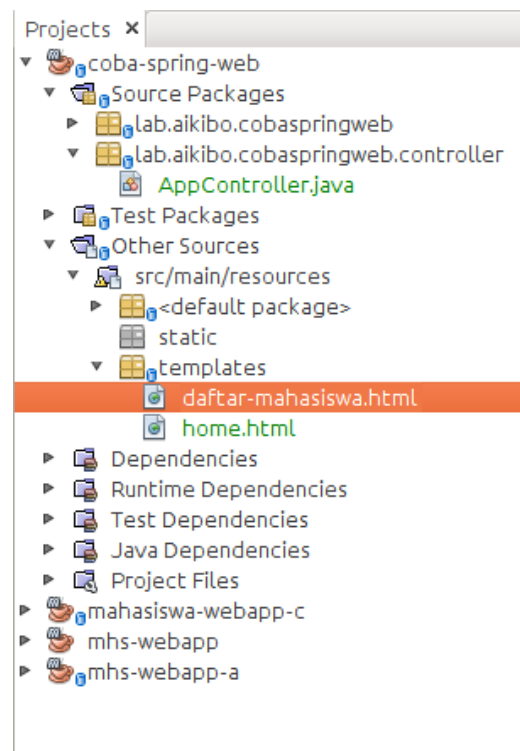
```
1 spring.datasource.url = jdbc:mysql://localhost:3306/
   mahasiswa
2 spring.datasource.username = dev
3 spring.datasource.password = rahasia
```

```

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

```

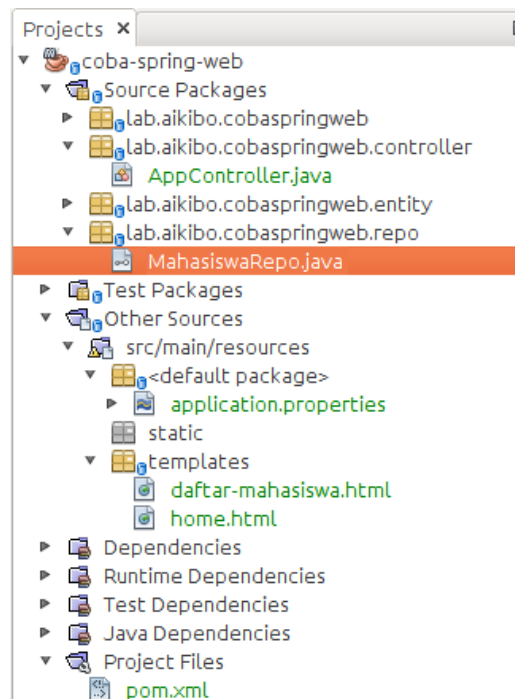
```
9  /**
10  *
11  *  @author tamami <tamami.oka@gmail.com>
12  */
13  @Controller
14  public class AppController {
15
16      @Autowired
17      private MahasiswaRepo mhsRepo;
18
19      @RequestMapping("/home")
20      public void index() {}
21
22      @RequestMapping("/daftar-mahasiswa")
23      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 :

```

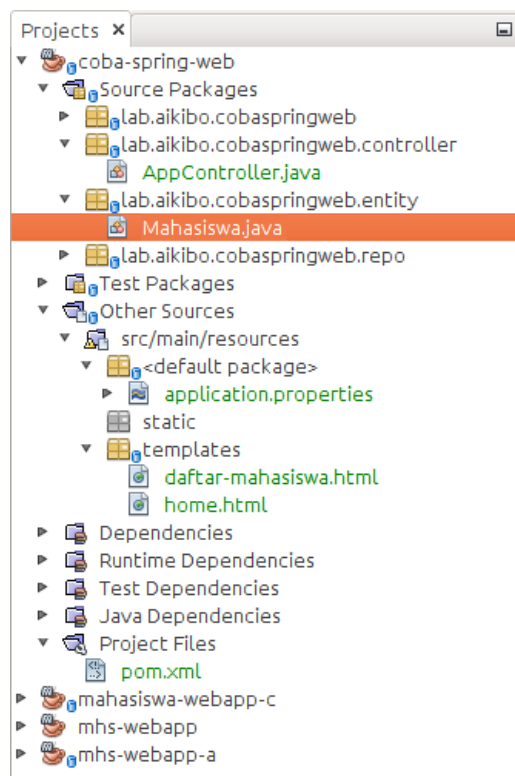
1 package lab.aikibo.cobaspringweb.repo;
2
3 import lab.aikibo.cobaspringweb.entity.Mahasiswa;
4 import org.springframework.data.jpa.repository.JpaRepository
5     ;
6 import org.springframework.stereotype.Repository;
7 /**
8  *
9  * @author tamami <tamami.oka@gmail.com>
10  */
11 @Repository
12 public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

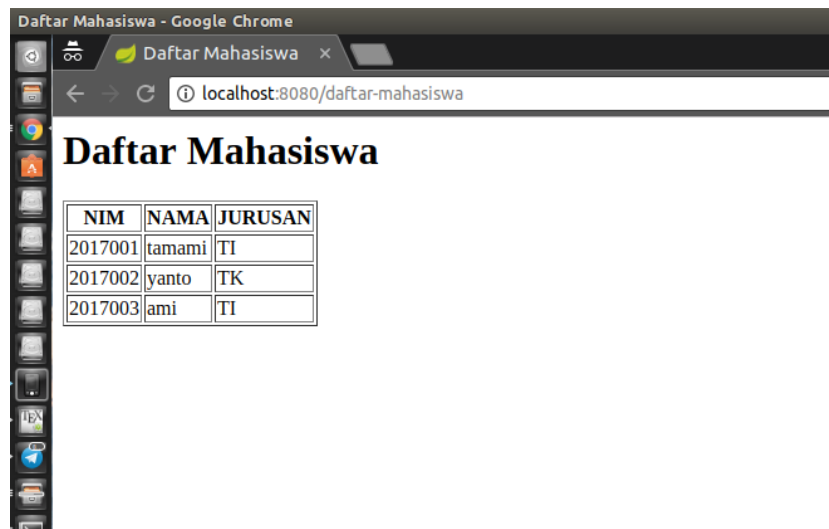
```
1 package lab.aikibo.cobaspringweb.entity;
```

```
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

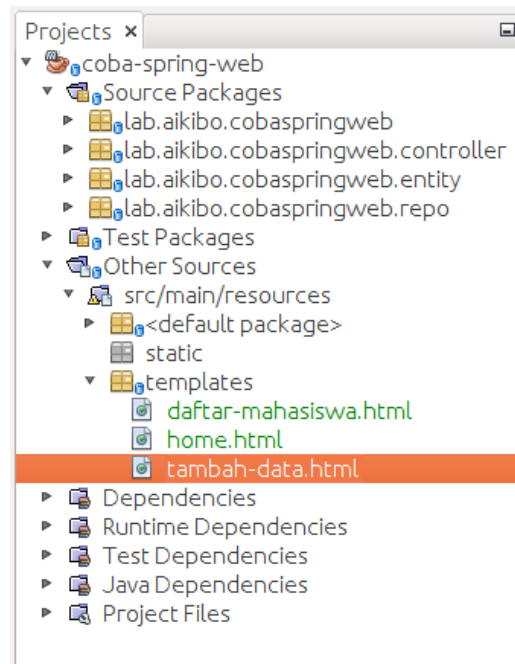
Gambar 1.13: Hasil Keluaran Daftar Mahasiswa

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}`

- 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

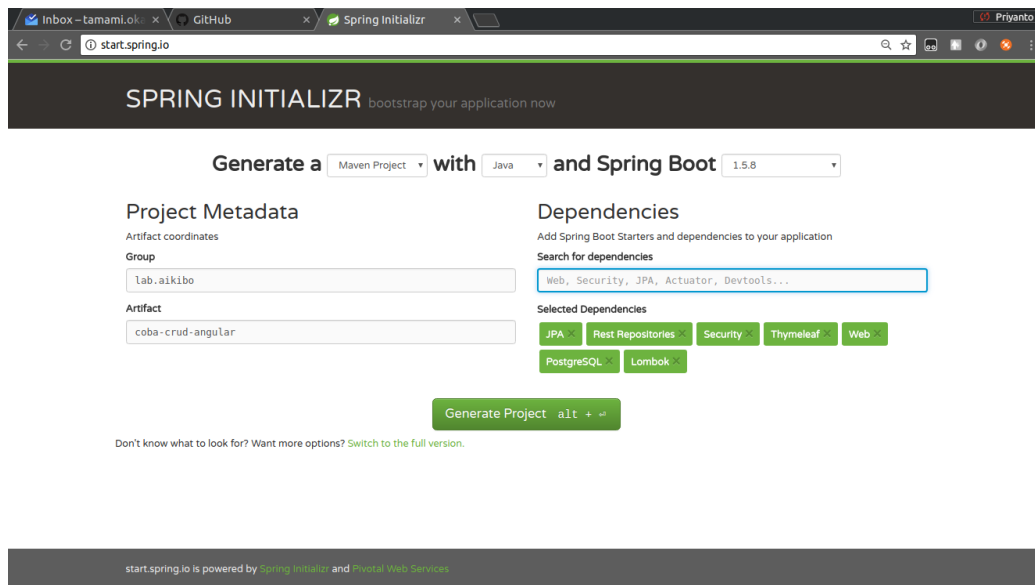
CRUD WITH SPRING ANGULAR

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

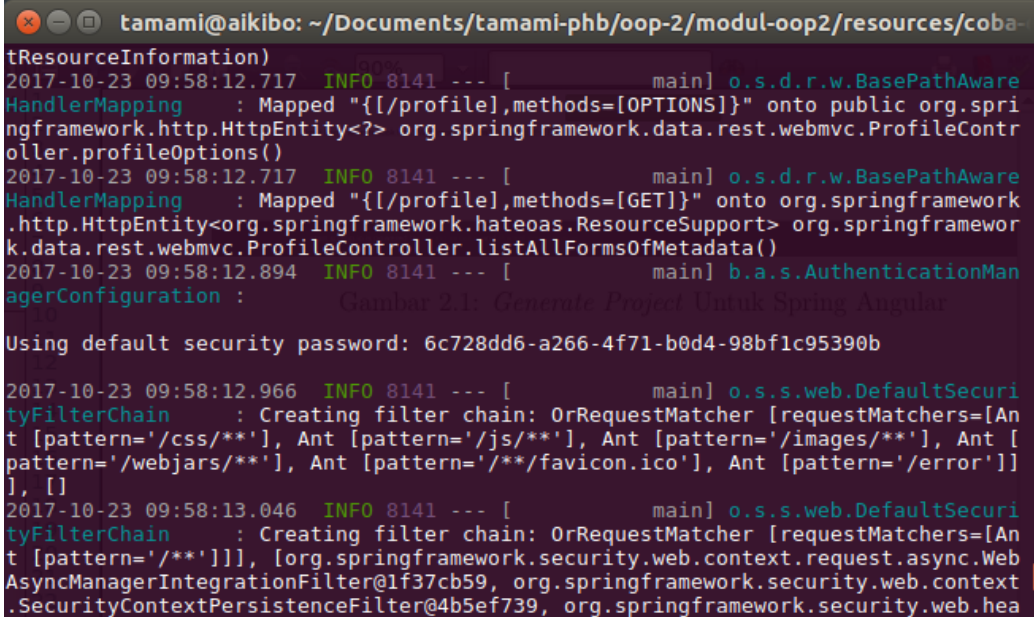
Selanjutnya kita ubah terlebih dahulu *file* `application.properties` dengan konfigurasi koneksi basis data, karena kita telah menambahkan Spring JPA dalam konfigurasi Maven kita. Berikut adalah isi dari *file* konfigurasi `application.properties` :

```

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 :



```

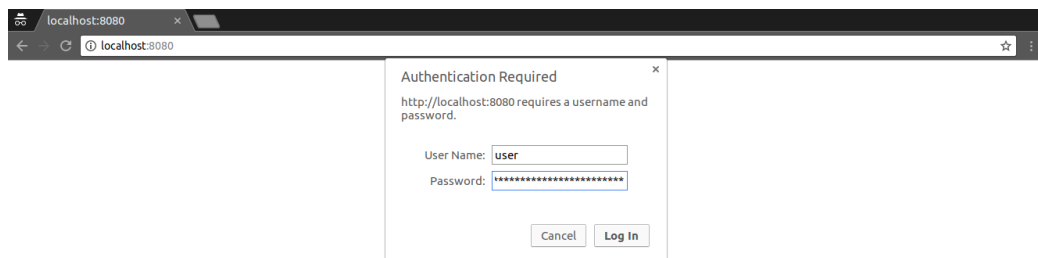
tamami@aikibo: ~/Documents/tamami-phb/oop-2/modul-oop2/resources/coba-
tResourceInformation)
2017-10-23 09:58:12.717 INFO 8141 --- [main] o.s.d.r.w.BasePathAware
HandlerMapping : Mapped "{[/profile],methods=[OPTIONS]}" onto public org.spr
ingframework.http.HttpEntity<?> org.springframework.data.rest.webmvc.ProfileContr
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 :

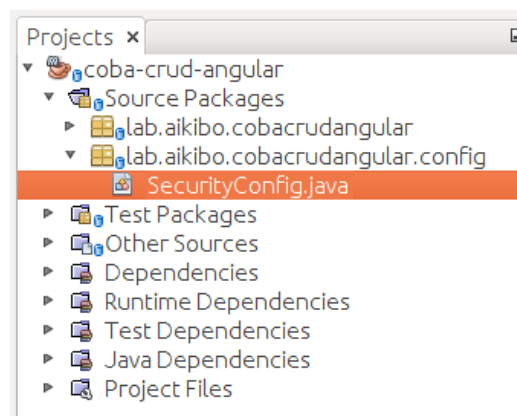


Gambar 2.4: *Default Page Setelah Login*

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

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*.

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	

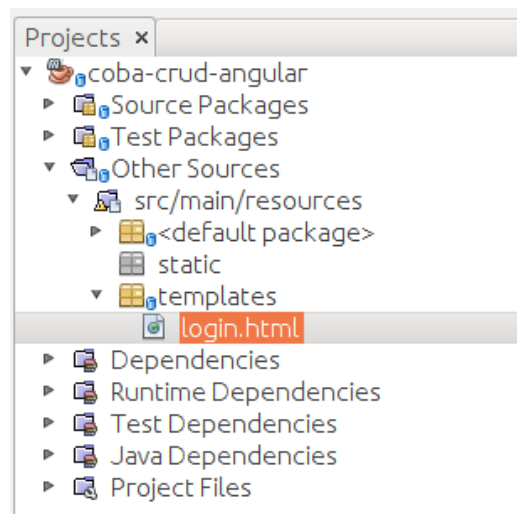
- b. Tabel *Role*. Nantinya daftar akses akan ditempatkan dalam tabel ini. Struktur tabelnya dengan nama tabel **roles** adalah sebagai berikut :

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-
PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rUpPvrszuE4W1povHYgTpBfshb
" 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)`