Materi Training Angular

Prepared by:
Henrie Prawiro

Daftar Isi

1. What is Angular ?

Angular adalah framework open source untuk front end.
Angular ditulis dalam TypeScript.
Angular mempermudah pembuatan aplikasi untuk web (mobile maupun desktop).
Angular dikembangkan oleh Google.

Fitur kunci dari Angular :

- Components
  Fokus dari Angular adalah component.
  Component-component ini digunakan untuk membangun suatu aplikasi.
- TypeScript
  Angular dibuat berdasar pada TypeScript.
- Services
  Service adalah sekumpulan code yang dapat dipakai bersama-sama (shared) oleh component-component yang berbeda dalam sebuah aplikasi.
  Sebagai contoh, bila kita memiliki sebuah component yang mengambil data dari database, kita dapat mendefinisikannya sebagai shared service yang mana dapat digunakan di seluruh aplikasi.
- Angular memiliki kemampuan penanganan event yang baik, template-template yang bagus dan dukungan yang lebih baik bagi mobile device.

Angular adalah salah satu solusi pengembangan SPA (Single Page Application) yang paling banyak dikenal selain React dan Vue.js.
Versi pertama dari Angular adalah AngularJS, dilahirkan pada 2010.
Ada banyak kekurangan pada versi ini seperti misalnya ukuran bundle yang besar dan adanya masalah pada performance, hal ini memicu rewrite total dari framework ini pada tahun 2014-2015.
Karena perubahan yang begitu fundamental untuk mengubah image pamakai maka penyebutannya pun berubah menjadi Angular 2, Angular 4, Angular 5 dan Angular 6.
Angular versi 2 ke atas cukup disebut dengan istilah Angular saja.

2. Getting Started

2.1. Requirement

1. Node.js
   Untuk memulai angular, Node.js harus telah terinstall.
   Jalankan perintah berikut untuk memeriksa apakah Node.js telah terinstall :

   ```
   npm –v
   ```

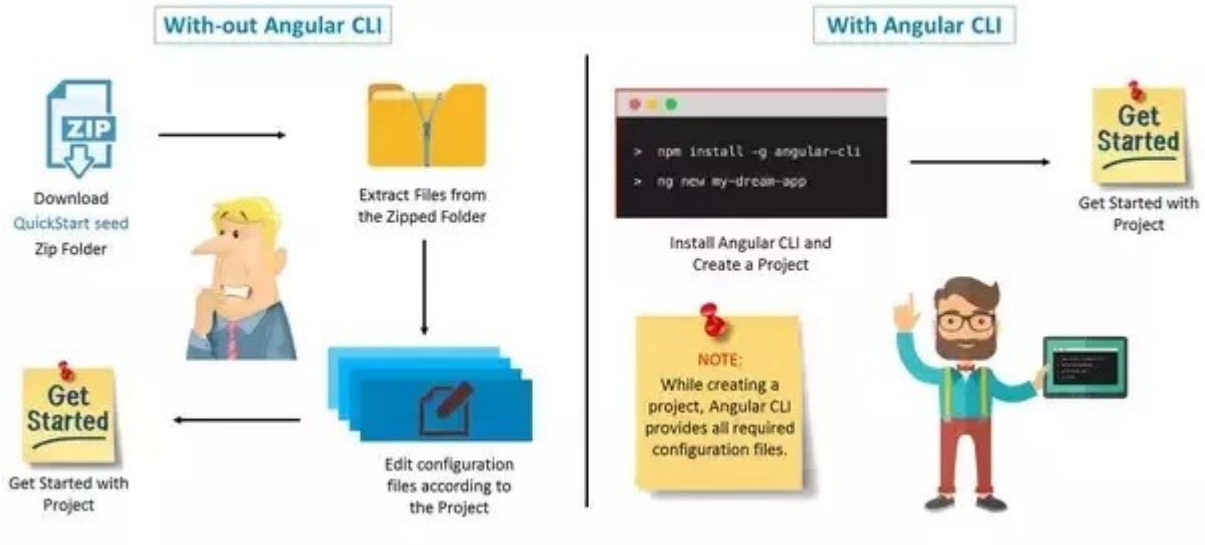
   Versi npm yang digunakan minimal 3.0.0

2. TypeScript
   Berikutnya yang harus ada adalah TypeScript.
   Untuk meng-install TypeScript dapat dilakukan dengan :

   ```
   npm install –g typescript
   ```

3. Browser
   Disarankan menggunakan Chrome

4. Angular CLI (Command Line Interface)



Untuk meng-install Angular CLI jalankan perintah berikut :

```
npm install -g @angular/cli@latest
```

Catatan : untuk versi yang hendak diinstal disarankan untuk mencari informasi dari internet.

2.2. Creating Project

Untuk menciptakan project Angular, pada folder dimana project tersebut hendak disimpan, jalankan perintah :

```
ng new angular-hello-world
```

Angular akan menciptakan banyak file untuk project baru ini.

Salah satu dari file yang diciptakan adalah : `index.html`, yang berisi :

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>AngularHelloWorld</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Apa yang ada di bagian `<head>` adalah hal yang sudah umum, yang perlu menjadi perhatian adalah pada :

```
<body>
  <app-root>Loading...</app-root>
</body>
```

Tag `<app-root>` adalah tempat dimana aplikasi yang dibuat akan di render.
Teks `'Loading...'` adalah placeholder yang akan tampil sebelum aplikasi diload ke layar. Teks ini dapat diganti dengan isi lain semisal sebuah gambar (dengan menggunakan tag `img`).
`<app-root>` adalah component yang didefinisikan oleh aplikasi Angular.

Dengan Angular dapat didefinisikan custom tag HTML dengan fungsionalitas sesuai kebutuhan pula.

Tag `<app-root>` adalah pintu masuk bagi keseluruhan aplikasi.

2.3. Running Application

Angular CLI memiliki built in HTTP server yang dapat digunakan untuk menjalankan aplikasi.

Untuk menggunakannya, masuk ke root folder dari project/aplikasi yang hendak dijalankan, lalu jalankan :

```
ng serve
```

Untuk men-set agar berjalan di port tertentu :

```
ng serve --port 9001
```

Setelah berjalan, buka aplikasi pada browser di URL : `http://localhost:9001`

3. Beginning Angular

3.1. Hello World

Making A Component
Ide besar di belakang Angular adalah component.

Pada aplikasi Angular, kita menulis tag HTML yang akan menjadi aplikasi yang interaktif, tetapi masalahnya browser hanya mengerti kumpulan tag yang terbatas.
Bagaimana bila diinginkan tag `<login>` yang menampilkan panel login ?

Ini adalah ide mendasar yang ada di belakang component, Angular akan memberitahu browser tag-tag baru dengan fungsionalitas tertentu yang menempel padanya.

Berikut ini contoh membuat component sederhana dengan tag `<app-hello-world>` yang nantinya dapat digunakan di HTML :

```
<app-hello-world></app-hello-world>
```

Untuk menciptakan component baru bernama `hello-world`, pada folder dimana project disimpan jalankan perintah :

```
ng generate component hello-world
```

Component dasar memiliki dua bagian :
1. Sebuah Component decorator
2. Sebuah class yang berisi definisi dari component

Setelah perintah di atas dijalankan akan terbentuk file TypeScript `hello-world.component.ts` yang berisi :

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {
```

```
  }

}
```

Penjelasan :
Statement `import` mendefinisikan modules yang hendak digunakan dalam code, dalam hal ini : `Component` dan `OnInit`.
`"@angular/core"` menunjukkan tempat dimana dependencies yang digunakan berada.
`OnInit` digunakan untuk menjalankan suatu coding ketika component diinisialisasi.

Component Decorators
Component Decorators dinyatakan dengan :

```
@Component({
  // ...
})
```

Decorators dapat dibayangkan sebagai metadata bagi component yang dibuat.
Ketika digunakan `@Component` maka ini artinya kita men-"dekorasi" HelloWorld sebagai sebuah Component.
Tugas utama dari `@Component` adalah mengkonfigurasi bagaimana "dunia luar" berinteraksi dengan component ini.
Kita hendak menggunakan component ini dengan tag `<app-hello-world>`, untuk itu kita mengkonfigurasi `@Component` dan menentukan *selector* yaitu `app-hello-world`.

Propety selector di sini menandakan elemen DOM yang digunakan oleh komponen ini, dalam hal ini setiap tag `<app-hello-world></app-hello-world>` akan dicompile menggunakan class `HelloWorldComponent` dan mendapatkan fungsionalitas yang ada di dalamnya.

Adding a template with templateUrl
Pada contoh di atas `templateUrl` diisi dengan `./hello-world.component.html`. Ini berarti template akan diambil dari file `hello-world.component.html` pada folder yang sama dengan component class.

Isi file `./hello-world.component.html` :

```
<p>
  hello-world works!
</p>
```

Ketika Angular me-load komponen ini, ia akan membaca dari file ini (`hello-world.component.html`) dan menggunakannya sebagai template bagi komponen yang dibuat.

Template dapat didefinisikan dalam dua cara, dengan cara di atas atau dengan menggunakan *inline template*, contoh :

```
@Component({
  selector: 'app-hello-world',
  template: `
     <p>
       hello-world works inline!
     </p>
  `
})
```

Perhatikan bahwa untuk string template digunakan backtick (`), ingat backtick pada JavaScript dan TypeScript (yang merupakan fitur dari ES6).

Adding CSS Styles with styleUrls

```
styleUrls: ['./hello-world.component.css']
```

Ini menandakan bahwa hendak digunakan file CSS `hello-world.component.css` (pada folder yang

sama dengan component class) sebagai style bagi komponen ini.
Angular menggunakan konsep yang disebut dengan "style-encapsulation" yang berarti bahwa style-style yang ditentukan bagi komponen tertentu hanya berlaku bagi komponen tersebut saja.

Loading Our Component
Berikut contoh penggunaan component `app-hello-world` di atas :

Buka file `app.component.html`, lalu isi dengan :

```
<h1>
   <app-hello-world></app-hello-world>
</h1>
```

3.2. Adding Data to the Component

Semisal hendak dibuat satu component yang menampilkan nama user, berikut hal yang harus dilakukan :

1. Menciptakan source component
   Jalankan perintah berikut untuk menciptakan source component :

   ```
   ng generate component user-item
   ```

2. Menambahkan tag app-user-item (nama component yang baru dibuat) ke `app.component.html`

   ```
   <h1>
      <app-hello-world></app-hello-world>

      <app-user-item></app-user-item>
   </h1>
   ```

3. Isi `user-item.component.ts`

   ```
   export class UserItemComponent implements OnInit {
      name : string; // <-- added name property

      constructor() {
         this.name = 'Felipe'; // set the name
      }

      ngOnInit() {
      }
   }
   ```

4. Ketika kita memiliki property dalam sebuah component (`name`), kita dapat menampilkan property tersebut pada template dengan menggunakan tanda `{}`
   Pada `user-item.component.hmtl` isi dengan :

   ```
   <p>
      Hello {{ name }}
   </p>
   ```

   Perhatikan bahwa pada template kita menambahkan syntax `{{ name }}`, ini disebut dengan *template tags* (disebut juga *mustache tags*).
   Apapun yang ada dalam *template tags* dapat dikembangkan menjadi sebuah expression.
   Karena template ini terhubung ke component, name akan menampilkan nilai dari `this.name`.

3.3. Working With Arrays

Pada Angular kita dapat melakukan iterasi terhadap list of object yang ada di template dengan menggunakan syntax `*ngFor`.
Berikut langkah untuk membuat component yang menampilkan daftar user :

1. Jalankan perintah untuk menciptakan component

```
ng generate component user-list
```

2. Ubah `app.component.html`

```
<h1>
   <app-hello-world></app-hello-world>

   <app-user-list></app-user-list>
</h1>
```

3. Buat isi dari component class `user-list.component.ts`

```
export class UserListComponent implements OnInit {
   names : string[];

   constructor() {
      this.names = ['Ari','Carlos','Felipe','Nate'];
   }

   ngOnInit() {
   }
}
```

4. Buat isi template `user-list.component.html`

```
<ul>
   <li *ngFor="let name of names">Hello {{ name }}</li>
</ul>
```

   Syntax `*ngFor` menandakan bahwa akan dilakukan looping.
   Ide dari `*ngFor` adalah kita akan menciptakan element DOM baru bagi setiap item yang ada di list tersebut.

   `"let name of names"`, `names` adalah array yang didefinisikan di `user-list.component.ts`.
   `let name` disebut dengan reference, `"let name of names"` memiliki arti : loop untuk tiap elemen pada `names` dan isikanlah masing-masing item ke variabel lokal yang bernama `name`.
   `ngFor` akan me-render satu tag `<li>` untuk tiap item yang ditemukan pada array `names` dan mendeklarasikan sebuah variabel lokal bernama : `name`.

3.4. Component of Component

Pada contoh sebelumnya, kita telah menciptakan `UserItemComponent`, jadi daripada me-render setiap item pada `UserListComponent` seharusnya digunakan `UserItemComponent` sebagai child component.

Untuk mewujudkan hal ini, perlu dilakukan tiga hal berikut :
- Konfigurasi `UserListComponent` untuk me-render menggunakan `UserItemComponent` (di template nya)
- Konfigurasi `UserItemComponent` untuk menerima variabel `name` sebagai input
- Konfigurasi template dari `UserListComponent` untuk mengirimkan `name` ke `UserItemComponent`

Langkah :
1. Tambahkan tag `app-user-item` ke dalam `user-list.component.html`

```
<ul>
   <li *ngFor="let name of names">
      <app-user-item></app-user-item>
   </li>
</ul>
```

2. Menambahkan input
   `name` pada `UserItemComponent` saat ini di set dengan `'Felipe'`, ia harus diubah agar ia dapat menerima input bagi property `name` tersebut.

Angular menyediakan decorator `@Input`, berikut perubahan yang harus dilakukan pada `UserItemComponent.ts` :

```
import {
   Component,
   OnInit,
   Input          // <--- added this
} from '@angular/core';

@Component({
   selector: 'app-user-item',
   templateUrl: './user-item.component.html',
   styleUrls: ['./user-item.component.css']
})

export class UserItemComponent implements OnInit {
   @Input() name: string;   // <--- added this

   constructor() {
      // removed setting name
   }

   ngOnInit(){
   }
}
```

Decorator `@Input` akan dibahas lebih detil nanti, saat ini yang perlu diketahui hanya sebatas bahwa decorator ini memungkinkan digunakan untuk mengirimkan nilai dari parent template.

Untuk dapat menggunakan `@Input`, harus pula ditambahkan `Input` pada blok `import`.

3. Mengirimkan nilai input
   Untuk mengirimkan nilai ke sebuah component digunakan syntax `[]` pada template.
   Berikut isi dari `user-list.component.html` :

```
<ul>
   <li *ngFor="let name of names">
      <app-user-item [name]="name"></app-user-item>
   </li>
</ul>
```

Pada Angular bila ditambahkan attribut dalam `[]` semisal `[foo]` ini artinya bahwa kita hendak mengirimkan nilai ke input yang bernama `foo` pada component tersebut.
Untuk memperjelas code di atas berikut sedikit perubahan penamaan dari reference :

```
<li *ngFor="let individualUserName of names">
   <app-user-item [name]="individualUserName"></app-user-item>
</li>
```

3.5. Bootstrapping

Istilah Bootstrapping di sini adalah tentang bagaimana program Angular dijalankan, beberapa instruksi inisial yang dilakukan yang memungkinkan jalannya program-program selanjutnya.

Aplikasi Angular dijalankan dengan perintah (dijalankan di folder tempat project berada) :

```
ng serve
```

`ng` kemudian akan menuju ke file `.angular-cli.json` untuk mencari pintu masuk (entry point) ke aplikasi yang dijalankan.
Secara umum akan terlihat seperti berikut :

- `.angular-cli.json` menetapkan sebuah file "main", yang mana dalam hal ini adalah `main.ts`

- `main.ts` adalah entry-point bagi aplikasi dan ia melakukan bootstrap bagi aplikasi kita
  Pada `main.ts`, baris inilah yang melakukan boot atas `AppModule` dan selanjutnya mem-boots aplikasi Angular.
  `platformBrowserDynamic().bootstrapModule(AppModule);`
- bootstrap melakukan booting atas sebuah modul Angular (module ini akan dijelaskan nanti)
- Kita menggunakan `AppModule` untuk melakukan bootstrap terhadap `app.AppModule` yang mana disebutkan pada `src/app/app.module.ts`
- `AppModule` menyebutkan component mana yang digunakan sebagai component tertinggi (top-level), dalam hal ini adalah `AppComponent`.
- `AppComponent` kemudian me-render seluruh aplikasi (`AppComponent` memiliki tag `<app-user-list>` pada templatenya dan ini me-render daftar user)

Saat ini yang perlu menjadi perhatian adalah Angular module system yaitu `ngModule`.

Ketika kita mem-boot suatu aplikasi Angular, kita tidak mem-boot secara langsung sebuah component melainkan menciptakan sebuah `ngModule` yang mana mengarah ke component yang hendak di load.

Perhatikan code app.module.ts berikut :

```
@NgModule({
   declarations: [
      AppComponent,
      HelloWorldComponent,
      UserItemComponent,
      UserListComponent
   ],
   imports: [
      BrowserModule,
      FormsModule,
      HttpModule
   ],
   providers: [],
   bootstrap: [AppComponent]
})
export class AppModule { }
```

Seperti halnya semua decorator, `@NgModule ( ... )` menambahkan metadata ke class yang mengikuti di bawahnya (dalam hal ini `AppModule`).

Decorator `@NgModule` memiliki tiga key :

1. `declarations`
   `declarations` menetapkan component-component yang didefinisikan di module ini.
   Berikut ide penting pada Angular :
   Component-component yang digunakan pada template harus dideklarasikan pada `NgModule` sebelum dapat menggunakannya pada template.

   `NgModule` dapat dibayangkan seperti sebuah "package" dan `declarations` menyebutkan apa saja component yang "dimiliki" oleh module ini.

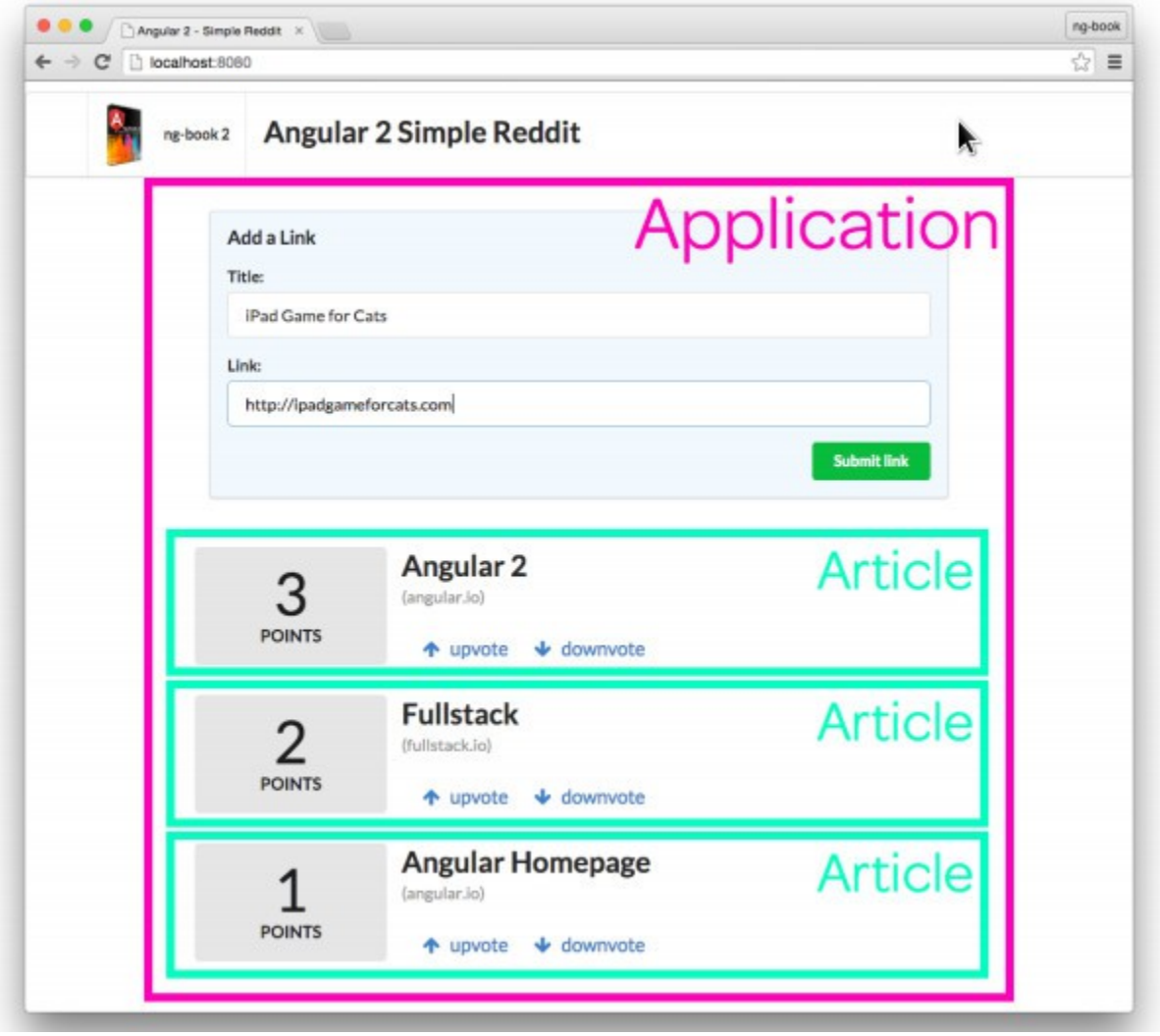
2. `imports`
   `imports` menggambarkan dependencies apa yang dimiliki oleh module ini. Kita menciptakan aplikasi bagi browser, jadi perlu diimport `BrowserModule`.

3. `providers`
   `providers` digunakan untuk melakukan dependency injection. Jadi untuk membuat suatu service tersedia dan dapat diinject di seluruh aplikasi, service tersebut harus ditambahkan di sini.

4. `bootstrap`.
   `bootstrap` memberi tahu Angular bahwa ketika module ini digunakan untuk mem-bootstrap sebuah aplikasi, `AppComponent` sebagai komponen tertinggi (top-level component) yang perlu diload.

4. Simple Reddit

Berikut ini secara step-by-step kita akan membuat aplikasi Reddit sederhana dimana bentuk dari aplikasi akan seperti ini :



Akan ada dua component bagi aplikasi ini :

1. Keseluruhan aplikasi yang mana berisi form yang digunakan untuk men-submit artikel baru (ditandai dengan warna magenta di gambar)
2. Tiap-tiap artikel (ditandai dengan warna hijau)

Program secara lengkap untuk Simple Reddit ini telah ada pada folder : `reddit`

Bagian ini akan menjelaskan step-by-step bagaimana aplikasi tersebut dibangun.

1. Creating project
   Untuk menciptakan aplikasi tersebut seperti biasa digunakan dengan perintah :

   ```
   ng new angular-reddit
   ```

   Masuk ke folder dimana project hendak diletakkan dan jalankan perintah di atas.

2. CSS
   Demo ini menggunakan Semantic UI (suatu framework CSS untuk UI), oleh sebab itu ada beberapa file CSS yang harus dicopy ke folder angular-reddit/src yang baru tercipta tadi.

   `index.html` ke `/angular-reddit/src`
   `style.css` ke `/angular-reddit/src`
   folder `app` ke `/angular-reddit/src/app`

folder `assets` ke `/angular-reddit/src/assets`

3. Menjalankan server Angular
   Masuk ke folder aplikasi tadi (angular-reddit), lalu jalankan :

   ```
   ng serve
   ```

   (jangan matikan `ng serve` yang sudah berjalan ini, dengan `ng serve` yang sedang berjalan, bila kita mengubah suatu code, untuk melihat perubahan yang terjadi hanya perlu melakukan refresh pada layar browser saja)

4. Membuka project Angular di VSCode
   Buka VSCode, pilih File - Open Folder dan pilih folder `angular-reddit` yang telah tercipta tadi

5. Menyesuaikan Application Component
   Sebagai langkah awal kita akan membentuk sebuah component baru yang mana akan :
   - menyimpan daftar artikel saat ini
   - berisi form untuk men-submit artikel baru

   Buka file `app.component.html` dan ubah isinya menjadi :

   ```
   <form class="ui large form segment">
      <h3 class="ui header">Add a Link</h3>

      <div class="field">
         <label for="title">Title:</label>
         <input name="title">
      </div>
      <div class="field">
         <label for="link">Link:</label>
         <input name="link">
      </div>
   </form>
   ```

   Amati hasilnya di browser.

6. Menambahkan interaksi
   Interaksi pada contoh aplikasi ini diwujudkan dengan menambahkan button submit.
   Ketika form di-submit, kita akan memanggil sebuah function untuk menciptakan dan menambah sebuah link artikel.
   Hal ini dilakukan dengan menambahkan event interaksi pada tag `<button>`.
   Angular akan merespon terhadap event yang didefinisikan dalam tanda `()`.

   Tambahkan code berikut pada `app.component.html` :

   ```
   <button (click)="addArticle()" class="ui positive right floated button">
      Submit link
   </button>
   ```

   Pada code di atas, ketika button ditekan, ia akan memanggil sebuah function bernama `addArticle()`, yang mana function ini harus didefinisikan pada class `AppComponent`.

   Tambahkan function `addArticle()` pada `app.component.ts` :

   ```
   export class AppComponent {
      addArticle(title: HTMLInputElement, link: HTMLInputElement): boolean {
         console.log(`Adding article title: ${title.value} and link: ${link.value}`);
         return false;
      }
   }
   ```

   NB: Perhatikan bahwa digunakan template string (`)

Function di atas akan dipanggil ketika button diklik.
Perhatikan bahwa function `addArticle` dapat menerima dua argument : `title` dan `link`.
Untuk ini template perlu diubah agar button mengirimkan argument tersebut.

Ubah pada file `app.component.html` dengan menambahkan `#newtitle` dan `#newlink`, perhatikan code di bawah ini :

```
<form class="ui large form segment">
   <h3 class="ui header">Add a Link</h3>

   <div class="field">
      <label for="title">Title:</label>
      <input name="title" #newtitle> <!-- changed -->
   </div>
   <div class="field">
      <label for="link">Link:</label>
      <input name="link" #newlink> <!-- changed -->
   </div>

   <!-- added this button -->
   <button (click)="addArticle(newtitle, newlink)"
              class="ui positive right floated button">
      Submit link
   </button>

</form>
```

Perhatikan bahwa pada input tag digunakan tanda `#` untuk memberitahu Angular untuk menempatkan tag tersebut pada variabel lokal yang mana nantinya dapat digunakan untuk dikirimkan ke function `addArticle()` pada button.

```
<input name="title" #newtitle>
```

Markup di atas memberitahu Angular untuk menghubungkan `<input>` ke variabel `newtitle`.
Syntax `#newtitle` ini disebut dengan : *resolve*.
`newtitle` sekarang adalah sebuah object yang mewakili elemen DOM yaitu input (khususnya tipe dari `newtitle` adalah `HTMLInputElement`)
Karena `newtitle` adalah sebuah object, maka ini artinya kita mengambil nilai dari object ini dengan menggunakan `newtitle.value` (lihat pada `app.component.ts`)

Amati hasilnya di browser.

7. Menciptakan source untuk component artikel
   Artikel yang baru disubmit harus ditampilkan ke layar.
   Karena setiap artikel yang disubmit akan ditampilkan berupa list, maka selayaknya dibuatkan component untuk tiap artikel.

   Masuk ke folder `angular-reddit` dan jalankan perintah :

   ```
   ng generate component article
   ```

   akan tercipta file :
   `src/app/article/article.component.css`
   `src/app/article/article.component.html`
   `src/app/article/article.component.spec.ts`
   `src/app/article/article.component.ts`

   dan akan diupdate file :
   `src/app/app.module.ts`
   (terkait deklarasi dan importnya)

Ada tiga bagian yang harus didefinisikan pada component baru ini :

- View untuk ArticleComponent di template
- Property bagi ArticleComponent dengan memberikan tanda `@Component` pada class tersebut
- Mendefinisikan logika atas komponent tersebut (component-definition class)

Ketiga bagian itu akan dibuat dalam langkah berikut ini.

8. Membuat template bagi ArticleComponent
   Buka file `article.component.html` dan isi dengan :

```
<div class="four wide column center aligned votes">
   <div class="ui statistic">
      <div class="value">
         {{ votes }}
      </div>
      <div class="label">
         Points
      </div>
   </div>
</div>
<div class="twelve wide column">
   <a class="ui large header" href="{{ link }}">
      {{ title }}
   </a>
   <ul class="ui big horizontal list voters">
      <li class="item">
         <a href (click)="voteUp()">
            <i class="arrow up icon"></i>
              upvote
            </a>
      </li>
      <li class="item">
         <a href (click)="voteDown()">
            <i class="arrow down icon"></i>
            downvote
         </a>
       </li>
    </ul>
</div>
```

   Penjelasan :
   Template di atas menggambarkan bahwa kita memiliki dua kolom :

   - Jumlah votes di sebelah kiri
   - Informasi tentang artikel di sebelah kanan

   Kolom-kolom ini ditetapkan dengan menggunakan class CSS : `four wide column` dan `twelve wide column`

   `votes` dan `title` ditampilkan dengan menggunakan template tag. Nilai untuk `votes` dan `title` merupakan property dari class `ArticleComponent` (yang mana akan kita definisikan sebentar lagi)
   Sebagai catatan, kita dapat menggunakan template string sebagai nilai, seperti pada tag : `href="{{ link }}"`
   Pada link upvote dan downvote dikenai sebuah action. Digunakan `(click)` untuk menghubungkan ke function `voteUp()`/`voteDown()`.

9. Membuat komponen (class) ArticleComponent
   Pertama-tama definisikan component dengan menggunakan decorator `@Component`.
   Berikut isi decorator `@Component` di file `article.component.ts` :

```
@Component({
   selector: 'app-article',
   templateUrl: './article.component.html',
   styleUrls: ['./article.component.css'],
})
```

`selector` pada decorator `@Component` menyatakan bahwa penggunaan component ini pada template adalah dengan menggunakan tag `<app-article>`, contoh :

```
<app-article></app-article>
```

Selanjutnya buat isi dari class `ArticleComponent` (pada file `article.component.ts`) :

```
export class ArticleComponent implements OnInit {
    @HostBinding('attr.class') cssClass = 'row';
    votes: number;
    title: string;
    link: string;

    constructor() {
      this.title = 'Angular 2';
      this.link = 'http://angular.io';
      this.votes = 10;
    }

    voteUp() {
      this.votes += 1;
    }

    voteDown() {
      this.votes -= 1;
    }

    ngOnInit() {
    }

}
```

Di sini kita menciptakan empat property :
1. `cssClass` : class CSS yang akan kita terapkan pada "host" dari component ini
2. `votes` : angka yang mewakili total dari semua upvotes dikurangi downvotes
3. `title` : sebuah string yang menampung judul dari artikel
4. `link` : sebuah string yang menampung URL dari artikel.

Kita menginginkan tiap `app-article` berada pada satu row. Dalam hal ini kita menggunakan Semantic UI dan Semantic menyediakan sebuah class CSS untuk rows yang disebut dengan `row`.
(Semantic UI adalah suatu framework untuk komponen-komponen UI yang dapat digunakan untuk menerapkan theme pada website).

Pada Angular, host dari sebuah component adalah elemen dimana component ini ditempelkan/digunakan.
Kita dapat men-set property dari host element dengan menggunakan decorator `@HostBinding()`.
Dalam hal ini kita meminta Angular untuk menjaga agar nilai dari class di host element sinkron dengan property`cssClass`.
Dengan menggunakan `@HostBinding()`, host element di-set agar memiliki class "`row`".
`constructor`, `voteUp` dan `voteDown` telah jelas.

10. Penggunaan component `app-article`
    Untuk menggunakan komponen `app-article` di atas, tambahkan tag `<app-article></app-article>` pada `AppComponent` (`app.component.html`) :

```
...
   <button (click)="addArticle(newtitle, newlink)"
              class="ui positive right floated button">
      Submit link
   </button>
 </form>

<div class="ui grid posts">
```

```
   <app-article>
   </app-article>
</div>
```

Bila kita menciptakan `ArticleComponent` menggunakan Angular CLI (melalui `ng generate component`), secara default ia akan "memberitahu" Angular tentang tag `app-article`.
Namun bila kita menciptakan component ini secara manual lalu me-reload browser, kita mungkin akan melihat bahwa tag `<app-article>` tidak tercompile.
Hal ini terjadi karena `AppComponent` belum mengetahui adanya komponen `ArticleComponent`.
Agar `AppComponent` mengenal `ArticleComponent`, tambahkan `ArticleComponent` ke dalam daftar deklarasi di `NgModule`.

File `app.module.ts` :

```
import { AppComponent } from './app.component';
import { ArticleComponent } from './article/article.component';  // <-- added this

@NgModule({
    declarations: [
       AppComponent,
       ArticleComponent // <-- added this
    ],
   ...
```

Di sini kita melakukan :
1. import `ArticleComponent`
2. menambahkan `ArticleComponent` ke daftar deklarasi

Tetapi, dengan code di atas, klik pada link vote up atau vote down akan menyebabkan page untuk reload bukannya mengupdate daftar artikel.
Secara default JavaScript menyebarkan event click ke semua komponen parent. Karena event click disebarkan ke parent, browser berusaha untuk mengikuti link kosong ini (link vote up atau vote down) yang mana ini berujung pada di-reload nya page ini.

Untuk memperbaikinya, kita harus membuat event handler untuk click tersebut agar mengembalikan nilai false. Hal ini untuk memastikan agar browser tidak mencoba untuk me-refresh page (karena dengan ini kita memberitahu browser agar tidak menyebarkan event ke parentnya).

```
voteDown(): boolean {
   this.votes -= 1;
   return false;
}
// and similarly with `voteUp()`
```

11. Merender beberapa Row
    Saat ini kita hanya memiliki satu artikel pada page.
    Artikel lain mungkin saja ditambahkan dengan menambah tag `<app-article>` tetapi bila ini dilakukan isinya pasti akan sama satu sama lain.

    Cara yang baik adalah dengan memisahkan struktur data yang kita gunakan dari coding component nya.
    Untuk melakukan ini, ciptakan sebuah struktur data yang mewakili satu artikel.

    Menciptakan class Article
    Tambahkan satu file `article.model.ts` untuk mendefinisikan sebuah class `Article`.

    Klik kanan pada `src-app-article` (pada project explorer di VSCode), lalu pilih menu New file dan masukkan nama file : `article.model.ts`
    Lalu isi dengan :

```
export class Article {
    title: string;
    link: string;
```

```
    votes: number;

    constructor(title: string, link: string, votes?: number) {
       this.title = title;
       this.link = link;
       this.votes = votes || 0;
    }
}
```

Dengan ini kita menciptakan sebuah class baru yang mewakili sebuah `Article`. Perhatikan bahwa ini adalah class polos (plain) dan bukan komponen Angular.
Pada pola Model-View-Controller, ini bisa disebut dengan *Model*.

NB: ingat pada TypeScript, parameter `votes` adalah opsional dan memiliki default 0.

Sekarang update `ArticleComponent` untuk menggunakan class `Article` tersebut.

File `article.component.ts` :

```
import { Article } from './article.model';

export class ArticleComponent implements OnInit {
   @HostBinding('attr.class') cssClass = 'row';
   article: Article;

   constructor() {
      this.article = new Article(
         'Angular 2',
         'http://angular.io',
         10);
   }

   voteUp(): boolean {
      this.article.votes += 1;
      return false;
   }

   voteDown(): boolean {
      this.article.votes -= 1;
      return false;
   }

   ngOnInit() {
   }

}
```

Perubahan ini berdampak pula pada view (template) dalam hal cara mendapatkan nilai dari artikel.
Sebelumnya digunakan `{{ votes }}`, ini harus diganti menjadi `{{ article.votes }}`, demikian pula dengan title dan link.

File `article.component.html` :

```
<div class="four wide column center aligned votes">
   <div class="ui statistic">
       <div class="value">
           {{ article.votes }}
       </div>
       <div class="label">
           Points
       </div>
   </div>
</div>
```

```
<div class="twelve wide column">
   <a class="ui large header" href="{{ article.link }}">
       {{ article.title }}
   </a>
   <ul class="ui big horizontal list voters">
       <li class="item">
           <a href (click)="voteUp()">
               <i class="arrow up icon"></i>
                  upvote
               </a>
       </li>
       <li class="item">
           <a href (click)="voteDown()">
               <i class="arrow down icon"></i>
               downvote
           </a>
       </li>
   </ul>
</div>
```

Sekarang aplikasi menjadi lebih baik, tetapi ada sesuatu yang kurang yaitu method voteUp dan voteDown melanggar prinsip encapsulation, yang mana method tersebut mengubah property internal dari artikel secara langsung. ArticleComponent *"knows too much"* tentang class `Article`.
Oleh karena itu class `Article` harus diperbaiki.

```
export class Article {
   title: string;
   link: string;
   votes: number;

   constructor(title: string, link: string, votes?: number) {
      this.title = title;
      this.link = link;
      this.votes = votes || 0;
   }

   voteUp(): void {
      this.votes += 1;
   }

   voteDown(): void {
      this.votes -= 1;
   }

   // domain() is a utility function that extracts
   // the domain from a URL, which we'll explain shortly
   domain(): string {
      try {
         // e.g. http://foo.com/path/to/bar
         const domainAndPath: string = this.link.split('//')[1];
         // e.g. foo.com/path/to/bar
         return domainAndPath.split('/')[0];
      } catch (err) {
         return null;
      }
   }
}
```

Penggunaannya di `ArticleComponent` harus diubah pula.

File `article.component.ts` :

```
export class ArticleComponent implements OnInit {
```

```
   @HostBinding('attr.class') cssClass = 'row';
   article: Article;

   constructor() {
      this.article = new Article(
         'Angular 2',
         'http://angular.io',
         10);
   }

   voteUp(): boolean {
      this.article.voteUp();
      return false;
   }

   voteDown(): boolean {
      this.article.voteDown();
      return false;
   }

   ngOnInit() {
   }

}
```

12. Menyimpan lebih dari satu artikel
    Untuk menyimpan lebih dari satu artikel, `AppComponent` harus diubah agar dapat menyimpan sekumpulan artikel.

    File `app.component.ts` :

```
import { Component } from '@angular/core';
import { Article } from './article/article.model'; // <-- import this

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})

export class AppComponent {
   articles: Article[]; // <-- component property

   constructor() {
       this.articles = [
           new Article('Angular 2', 'http://angular.io', 3),
           new Article('Fullstack', 'http://fullstack.io', 2),
           new Article('Angular Homepage', 'http://angular.io', 1),
       ];
}
```

13. Setting input pada ArticleComponent
    Setelah kita memiliki daftar dari model `Article`, kita harus mengirimkannya ke `ArticleComponent` menggunakan Input.
    Pada `article.component.ts` yang lalu dapat dilihat bahwa pada constructor ada hard code untuk Article tertentu. Inti dari membuat component bukan hanya encapsulation tetapi juga reusability. Kita ingin agar dapat me-reuse komponen `app-article` dengan mengirimkan Article sebagai "parameter" ke component, seperti ini :

```
  <app-article [article]="article1"></app-article>
  <app-article [article]="article2"></app-article>
```

    Angular memungkinkan hal itu dengan menggunakan decorator Input pada property dari component :

```
class ArticleComponent {
   @Input() article: Article;
   // ...
```

Berikut isi dari `article.component.ts` setelah menggunakan `@Input` :

```
import {
    Component,
    OnInit,
    Input,            // <-- added,
    HostBinding
} from '@angular/core';
import { Article } from './article.model'; // <-- added

@Component({
    selector: 'app-article',
    templateUrl: './article.component.html',
    styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
    @HostBinding('attr.class') cssClass = 'row';
    @Input() article: Article;

   constructor() {
      // article is populated by the Input now,
      // so we don't need anything here
   }

   voteUp(): boolean {
      this.article.voteUp();
      return false;
   }

   voteDown(): boolean {
      this.article.voteDown();
      return false;
   }

   ngOnInit() {
   }

}
```

Catatan :

Bila diinginkan penggunaan input dengan nama yang berbeda pada template seperti misalnya :

```
<my-component [shortName]="myName" [oldAge]="myAge"></my-component>
```

Maka pada component dapat dituliskan :

```
  @Input('shortName') name: string;
  @Input('oldAge') age: number;
```

Merender daftar Article

Sebelumnya kita mengkonfigurasi `AppComponent` untuk menyimpan sebuah array yang berisi artikel-artikel. Sekarang mari kita ubah `AppComponent` agar me-render semua artikel tersebut.
Untuk itu, kita tidak dapat menggunakan tag `<app-article>` saja, kita akan menggunakan `NgFor` untuk meng-iterasi daftar artikel dan me-render tiap `app-article`.

Pada template dari `AppComponent` (`app.component.html`) tepat di bawah `</form>` tambahkan :

```
...
     Submit link
  </button>
</form>
```

```
<!-- start adding here -->
<div class="ui grid posts">
  <app-article
     *ngFor="let article of articles"
     [article]="article">
  </app-article>
</div>
<!-- end adding here -->
```

`*ngFor="let article of articles"` akan meng-iterasi daftar artikel dan menciptakan variabel lokal `article` untuk tiap item di list.
Untuk menetapkan input artikel pada komponen, digunakan ekspresi `[inputName]="inputValue"` dalam hal ini `[article]="article"`

Reload browser untuk melihat perubahannya.

14. Menambahkan artikel yang baru diinput
    Sekarang kita harus mengubah function `addArticle` untuk menambahkan artikel baru ketika button submit ditekan.

    File app.component.ts :

```
addArticle(title: HTMLInputElement, link: HTMLInputElement): boolean {
   console.log(`Adding article title: ${title.value} and link: ${link.value}`);
   this.articles.push(new Article(title.value, link.value, 0));
   title.value = '';
   link.value = '';
   return false;
}
```

    Perubahan tersebut akan :
    1. menciptakan sebuah object `Article` baru dengan `title` dan URL (`link`) yang di-submit
    2. menambahkan object `Article` baru tersebut ke array `articles`
    3. membersihkan inputan untuk `title` dan `link` di layar (menjadi kosong lagi).

    Reload browser untuk melihat perubahannya.

15. Finishing Touches
    Sebagai sentuhan akhir, kita akan menambahkan sebuah hint di samping link yang menunjukkan domain dimana user akan dialihkan ketika link diklik.

    File article.model.ts :

```
   domain(): string {
      try {
         // e.g. http://foo.com/path/to/bar
         const domainAndPath: string = this.link.split('//')[1];
         // e.g. foo.com/path/to/bar
         return domainAndPath.split('/')[0];
      } catch (err) {
         return null;
      }
   }
```

    Panggil function ini pada template milik `ArticleComponent` :

    File `article.component.html` :

```
  <div class="twelve wide column">
    <a class="ui large header" href="{{ article.link }}">
       {{ article.title }}
    </a>
```

```
<!-- right here -->
<div class="meta">({{ article.domain() }})</div>
<ul class="ui big horizontal list voters">
   <li class="item">
      <a href (click)="voteUp()">
```

Reload browser untuk melihat perubahannya.

16. Mengurutkan (sort) artikel berdasarkan score
    Sekarang kita akan mengurutkan artikel-artikel yang tampil berdasarkan score-nya.
    Kita menyimpan artikel-artikel dalam sebuah array di class `AppComponent`, tetapi array tersebut belum diurutkan.
    Cara paling mudah untuk menangani hal ini adalah dengan menciptakan sebuah method baru `sortedArticles` pada `AppComponent`.

    File `app.component.ts` :

    ```
    sortedArticles(): Article[] {
       return this.articles.sort((a: Article, b: Article) => b.votes - a.votes);
    }
    ```

    Pada file `app.component.html`, ubah agar `ngFor` melakukan iterasi dari `sortedArticles()` :

    ```
    <div class="ui grid posts">
       <app-article
          *ngFor="let article of sortedArticles()"
          [article]="article">
       </app-article>
    </div>
    ```

17. Deployment
    Setelah kita memiliki aplikasi yang dapat sepenuhnya berjalan, langkah selanjutnya adalah meletakkannya agar dapat berjalan live di internet.
    Deploying adalah tindakan meletakkan code ke sebuah server sehingga dapat diakses oleh orang lain.
    Untuk itu kita akan melakukan :
    - compile semua code TypeScript agar menjadi JavaScript
    - membungkus (bundle) semua file JavaScript ke dalam satu atau dua file
    - upload JavaScript, HTML, CSS dan gambar-gambar ke sebuah server.

    Pada akhirnya, aplikasi Angular adalah sebuah file HTML yang me-load code JavaScript, jadi kita harus meng-upload code yang kita buat ke sebuah computer di suatu tempat di internet.

    Building Our App for Production
    Angular CLI dapat digunakan untuk melakukan build untuk production.
    Hal ini dapat dilakukan dengan menjalankan perintah :

    ```
    ng build --target=production --base-href '/'
    ```

    atau

    ```
    ng build --prod --base-href '/'
    ```

    Perintah ini memberitahu tool `ng` agar mem-build aplikasi kita yang diperuntukkan bagi production environment.
    `base-href` menetapkan apa 'root' URL dari aplikasi ini.
    Sebagai contoh, jika kita ingin men-deploy aplikasi ke sebuah sub folder pada server di bawah `/ng-book-demo/`, dapat dilakukan dengan : `--base-href '/ng-book-demo/'`
    Setelah proses build selesai, kita akan memiliki folder `dist` yang berisi hasil dari build tersebut.

    Untuk mem-build aplikasi kita di atas, masuk ke folder `angular-reddit`, dan jalankan :

    ```
    ng build --target=production --base-href '/'
    ```

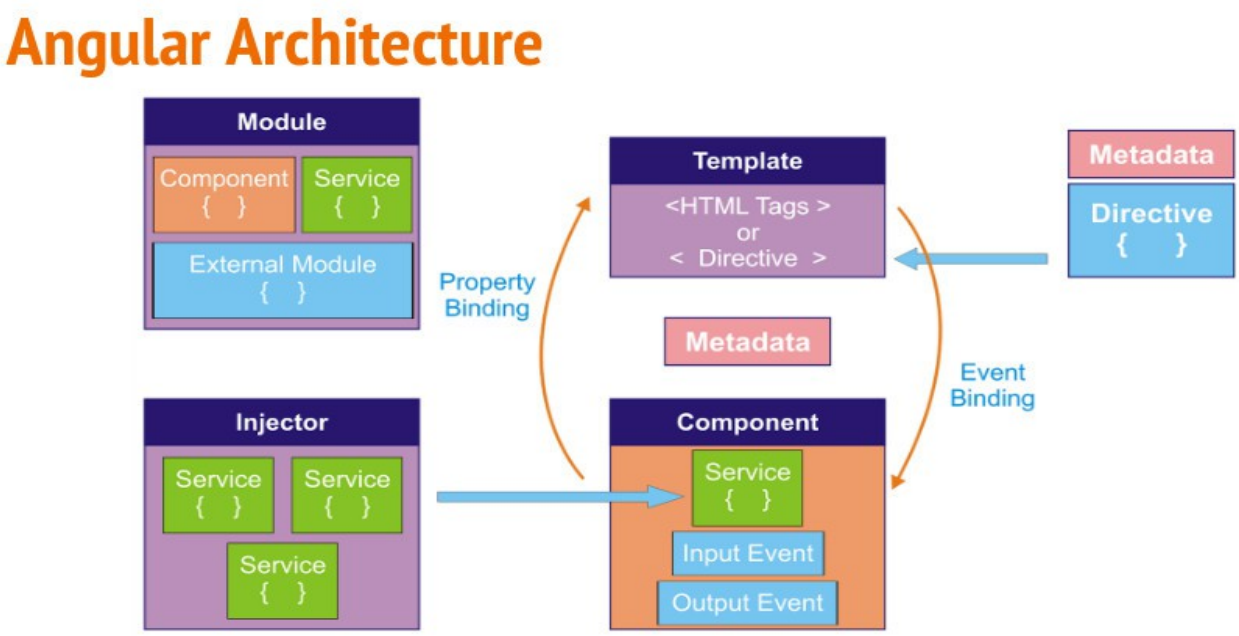
File-file yang ada di folder `angular-reddit/dist` tersebut adalah hasil compile keseluruhan dari aplikasi kita.

18. Uploading ke Server
    Langkah terakhir adalah melakukan upload ke server di internet. Untuk ini akan ada banyak cara yang dapat digunakan.

5. How Angular Works

Ide besar utama dari Angular adalah bahwa sebuah aplikasi Angular terbentuk dari component-component.

5.1. Angular Architecture



Dasar dari aplikasi Angular adalah NgModules, NgModules menyediakan konteks untuk keperluan compile bagi component-component yang digunakan oleh aplikasi ini.
Aplikasi Angular dibentuk dari sekumpulan NgModules.
Sebuah aplikasi memiliki paling tidak sebuah root module yang digunakan untuk bootstrapping dan biasanya memiliki banyak module-module lain.
NgModule mengidentifikasi module-module, component-component, directives, dan lain-lain yang digunakan dalam aplikasi.

Component mendefinisikan view, yang mana terdiri dari sekumpulan screen element yang dapat digunakan dan dimodifikasi oleh Angular berdasarkan data dan logika program yang kita buat.

Component menggunakan services, yang mana menyediakan fungsionalitas tertentu yang tidak berhubungan langsung dengan view. Service dapat diinjeksi ke dalam component sebagai dependency, yang mana ini membuat code menjadi modular, reusable dan efisien.

Baik component maupun service adalah class, dengan decorator sebagai penanda atas tipe mereka dan menyediakan metadata untuk memberitahu Angular bagaimana cara menggunakan component/service tersebut.

Metadata bagi sebuah class component menghubungkan component dengan sebuah template yang mendefinisikan sebuah view.
Template adalah kombinasi dari HTML sederhana dengan directive-directive Angular (misal: ngFor) dan binding markup (misal: mustaches tag) yang memungkinkan Angular untuk memodifikasi HTML sebelum direnderer untuk ditampilkan.

Metadata bagi sebuah class service menyediakan informasi yang dibutuhkan Angular untuk membuat service tersebut tersedia bagi component melalui dependency injection (DI).

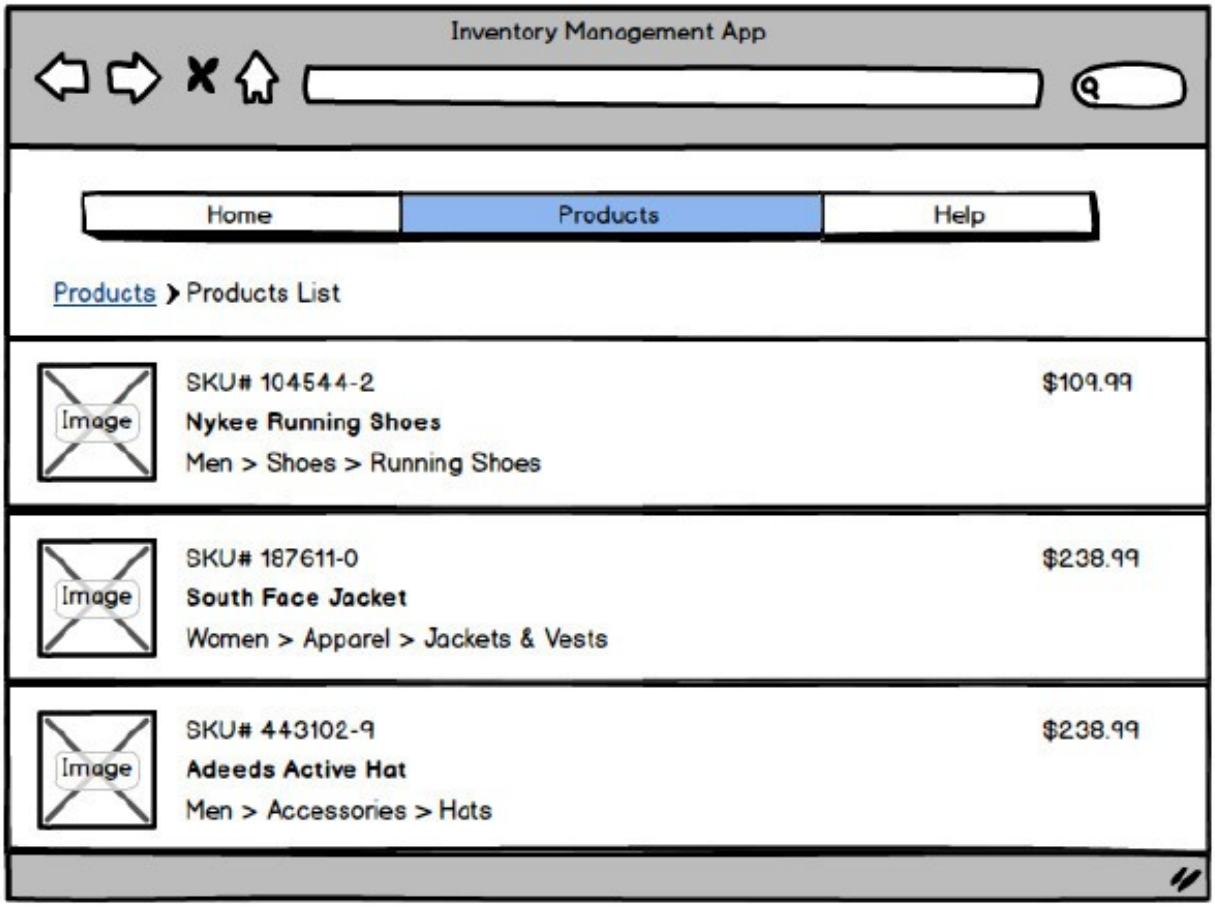
Component-component dari sebuah aplikasi biasanya mendefinisikan banyak view yang disusun berdasar hirarki.
Angular menyediakan layanan Router untuk membantu mendefinisikan path navigasi ke view-view yang ada.
Router menyediakan kemampuan navigasi canggih secara in-browser.

5.2. Designing Application

Aplikasi Angular tidak lebih dari sebuah pohon Component.
Di ujung pangkal dari pohon tersebut (root) top level component adalah aplikasi itu sendiri dan inilah yang akan dirender oleh browser ketika melakukan "booting" (red: bootstrapping) aplikasi.

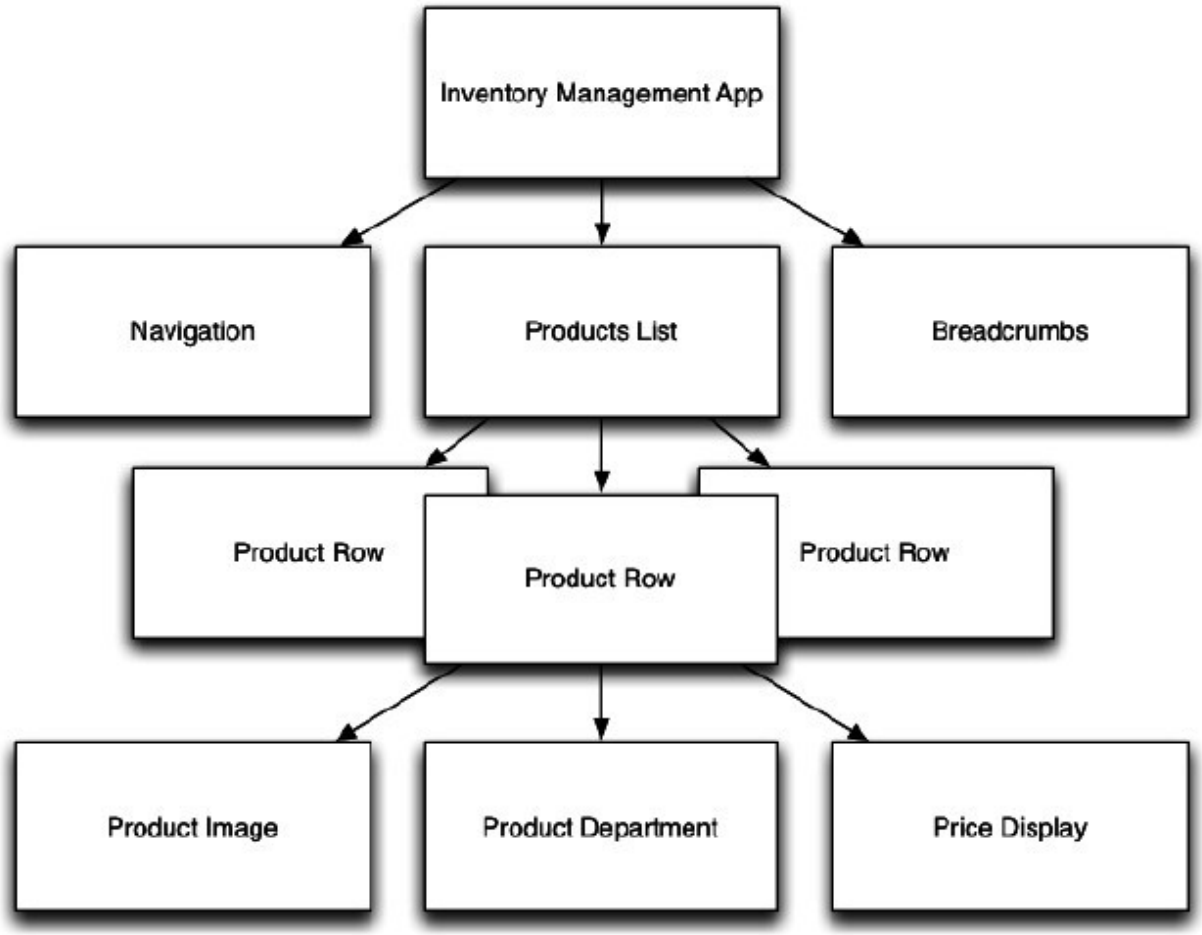
Satu hal yang luar biasa tentang component adalah bahwa mereka dapat disusun ulang (composable).
Ini berarti bahwa kita dapat membangun component yang lebih besar dari component yang lebih kecil.
Application adalah component yang me-render component lain.
Karena component-component terstruktur dalam bentuk parent/child tree, tiap kali component dirender, ia secara recursive me-render component-component anaknya.

Perhatikan gambar berikut :



Dari gambar di atas kita dapat mengelompokkan page tersebut dalam tiga component utama :
1. Komponen untuk Navigation
2. Komponen Breadcrumbs
3. Komponen untuk Product Info / Product List
   Product List dibentuk dari sekumpulan produk-produk.
   Komponen ini dapat dipecah menjadi komponen yang lebih kecil dimana dapat dikatakan bahwa Product List terdiri dari beberapa Product Rows.
   Dan tentu Product Rows dapat dipecah lagi menjadi komponen yang lebih kecil, yaitu :
   - Product Image
   - Product Department
   - Price Display

Pada akhirnya, bila semua komponen tersebut dirangkai dalam suatu pohon, akan menjadi diagram berikut :



Di paling atas (root) adalah aplikasi ini sendiri yaitu Inventory Management App, diikuti dengan komponen Navigation, Breadcrumb, Product List dan seterusnya.

Langkah-langkah mengembangkan (coding) aplikasi untuk inventory di atas pada dasarnya sama dengan yang telah kita lakukan sebelumnya, oleh sebab itu di sub bab ini tidak lagi dituliskan step-by-step untuk pengembangannya.
Bagian ini akan menjelaskan beberapa hal yang belum disinggung pada pokok bahasan sebelumnya.
Tetapi untuk kejelaan telah disediakan satu project yang telah jadi untuk dapat dipelajari yaitu :
`inventory-app`

5.3. Custom Event

Pada aplikasi ini, suatu produk dapat dipilih (diklik) untuk kemudian dilakukan suatu tindakan atas pilihan tersebut.

Di Angular kita mengirimkan data ke component menggunakan input, contoh :

```
<products-list
   [productList]="products"
   ...
```

Ini berarti bahwa kita mengirimkan isi dari `products` milik component ini ke property `productList` milik tag component yang digunakan (dalam hal ini `products-list`)

Sedangkan untuk mengirimkan data keluar dari component dilakukan menggunakan outputs (ditandai dengan tanda `()`, disebut dengan *output binding*), contoh (pada `app.component.html`) :

```
<products-list
   ...
   (onProductSelected)="productWasSelected($event)">
```

Code di atas berarti bahwa kita menunggu (listen) event `onProductSelected` (output) dari komponen `ProductsList`.
`productWasSelected` adalah action/function yang akan dipanggil/dijalankan sewaktu event tersebut muncul.
`$event` adalah variabel khusus yang mewakili sesuatu yang dikirimkan (emitted) oleh output.

Function yang dipanggil sewaktu event tersebut muncul dapat dilihat pada `app.component.ts` :

```
productWasSelected(product: Product): void {
   console.log('Product clicked: ', product);
}
```

Karena event tersebut melekat pada `product-list` (komponen `ProductList`), maka output harus didefinisikan pada komponen ini (`products-list.component.ts`) :

```
@Output() onProductSelected: EventEmitter<Product>;
```

Ada banyak built-in events seperti misalnya : `mousedown`, `mousemove`, `dbl-click`, dll yang mana sudah ter-publish.
Bila kita hendak membuat custom event, maka kita juga harus mem-publikasi-kan custom event tersebut.

Untuk menciptakan custom output event, kita harus melakukan tiga hal :
1. mendefinisikan outputs pada konfigurasi `@Component`
2. menempelkan `EventEmitter` pada property output
3. emit (menyiarkan) an event (sesuai dengan yang diinginkan)

Pada file `products-list.component.ts` :

```
@Output() onProductSelected: EventEmitter<Product>;

   constructor() {
       this.onProductSelected = new EventEmitter();
   }
```

Pada template, yaitu di file `products-list.component.html` :

```
<div class="ui items">
    <product-row
       *ngFor="let myProduct of productList"
       [product]="myProduct"
       (click)='clicked(myProduct)'
       [class.selected]="isSelected(myProduct)">
    </product-row>
</div>
```

Perhatikan bahwa pada tag `product-row`, dilakukan suatu action bila suatu product diklik.
NB:

Baris `[class.selected]="isSelected(myProduct)"` digunakan untuk men-set class pada elemen ini sesuai kondisi yang diinginkan (kondisi dihasilkan oleh `isSelected(myProduct)`).
Syntax tersebut menyatakan : "tambahkan class CSS selected jika `isSelected(myProduct)` mengembalikan nilai true.

Pada file `products-list.component.ts`, definisikan action untuk klik tadi :

```
clicked(product: Product): void {
   this.currentProduct = product;
   this.onProductSelected.emit(product);
```

```
    }
```

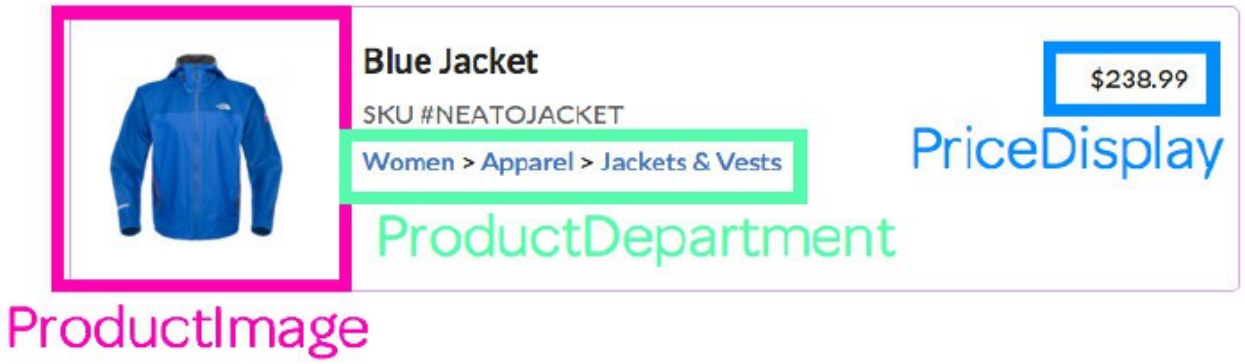
Function di atas melakukan dua hal :

1. men-set this.currentProduct dengan product yang diklik
2. menyiarkan (emit) product yang diklik pada output yang telah didefisikan (dalam hal ini `onProductSelected`) sehingga custom event `onProductSelected` tertrigger.

5.4. Detail Component of Component

Component ketiga dari aplikasi di atas adalah component untuk masing-masing baris product yaitu `ProductRowComponent`, component ini terdiri dari component-component lain.

Berikut gambaran visual dari tiga component yang akan digunakan oleh `ProductRowComponent`.



Konfigurasi dari `ProductRowComponent` (file `product-row.components.ts`) :

```
import {
    Component,
    Input,
    HostBinding
} from '@angular/core';
import { Product } from '../product.model';

/**
   * @ProductRow: A component for the view of single Product
  */
@Component({
    selector: 'product-row',
    templateUrl: './product-row.component.html',
})
export class ProductRowComponent {
    @Input() product: Product;
    @HostBinding('attr.class') cssClass = 'item';
}
```

(tidak ada hal khusus dalam konfigurasi di atas)

Template untuk konfigurasi di atas (file `product-row.component.html`) :

```
<product-image [product]="product"></product-image>
<div class="content">
    <div class="header">{{ product.name }}</div>
    <div class="meta">
       <div class="product-sku">SKU #{{ product.sku }}</div>
    </div>
    <div class="description">
       <product-department [product]="product"></product-department>
    </div>
```

```
</div>
<price-display [price]="product.price"></price-display>
```

Secara konsep tidak ada hal baru pada template di atas, tetapi terlihat bahwa template untuk `product-row` tersebut menggunakan tiga component lain yaitu :
`product-image`, `product-department` dan `price-display`.

1. ProductImageComponent

   Template dari `ProductImageComponent` terdiri dari hanya satu baris saja, sehingga tidak perlu dipisah dalam file HTML tersendiri.

   Pada file `product-image.component.ts` :

   ```
   /**
      * @ProductImage: A component to show a single Product's image
      */
   @Component({
       selector: 'product-image',
       template: `
       <img class="product-image" [src]="product.imageUrl">
       `
   })
   export class ProductImageComponent {
       @Input() product: Product;
       @HostBinding('attr.class') cssClass = 'ui small image';
   }
   ```

   Yang perlu menjadi perhatian adalah, di sini kita menggunakan input `[src]` pada tag `<img>`, dengan ini kita memberitahu Angular bahwa kita menggunakan `src` sebagai input pada tag `<img>` ini, sehingga nantinya Angular akan me-replace nilai dari atribut `src` ketika sudah ada expression yang didapat.
   Sebenarnya bisa juga digunakan cara berikut :

   ```
   <img src="{{ product.imageUrl }}">
   ```

   Kedua cara tersebut pada dasarnya melakukan hal yang sama, jadi silakan dipilih mana yang paling cocok untuk tim anda.

2. PriceDisplayComponent

   Pada file `price-display.component.ts` :

   ```
   import {
       Component,
       Input
   } from '@angular/core';

   /**
      * @PriceDisplay: A component to show the price of a
      * Product
      */
   @Component({
       selector: 'price-display',
       template: `
       <div class="price-display">\${{ price }}</div>
       `
   })
   export class PriceDisplayComponent {
       @Input() price: number;
   }
   ```

Catatan hanya ada di tanda `$`, kita melakukan escaping pada tanda dollar (`\$`) karena ini adalah backtick string dan tanda `$` digunakan untuk menandai *template variable*.
(Contoh di JavaScript :
`var message = `Hello **${customer.name}**, want to buy **${card.amount} ${card.product}** for a total of ${card.amount * card.unitprice} bucks?`)

3. ProductDepartmentComponent

Pada file `product-department.component.ts` :

```
import {
    Component,
    Input
} from '@angular/core';
import { Product } from '../product.model';

/**
   * @ProductDepartment: A component to show the breadcrumbs to a
   * Product's department
   */
@Component({
    selector: 'product-department',
    templateUrl: './product-department.component.html'
})
export class ProductDepartmentComponent {
    @Input() product: Product;
}
```

Pada file `product-department.component.html` :

```
 <div class="product-department">
     <span *ngFor="let name of product.department; let i=index">
       <a href="#">{{ name }}</a>
       <span>{{i < (product.department.length-1) ? '>' : ''}}</span>
     </span>
 </div>
```

Hal yang perlu diperhatikan pada komponen `ProductDepartmentComponent` adalah `ngFor` dan tag `<span>`.
`ngFor` melakukan looping terhadap `product.department` dan meletakkan tiap department ke variabel `name`.
Bagian yang baru adalah `let i = index`, ini digunakan untuk mendapatkan nomor iterasi dari `ngFor`.
Pada tag `<span>`, kita menggunakan variabel `i` untuk menentukan kapan ditampilkan tanda "`>`".
Hasil yang hendak dicapai adalah string seperti ini :

```
Women > Apparel > Jackets & Vests
```

Expression `{{i < (product.department.length-1) ? '>' : ''}}` menyatakan bahwa "`>`" dituliskan bila ini bukan department yang terakhir

Langkah terakhir dari pembuatan aplikasi ini adalah memastikan bahwa telah ada `NgModule` bagi aplikasi tersebut (lihat di `app.module.ts`) dan melakukan booting.

6. Built-in Directives

6.1. NgIf

`ngIf` digunakan untuk menampilkan atau menyembunyikan sebuah elemen berdasarkan suatu kondisi.
Jika hasil dari expresi mengembalikan nilai false, elemen akan dihilangkan dari DOM.

Contoh :

```
<div   *ngIf="false"></div>         <!--    never displayed -->
<div   *ngIf="a > b"></div>         <!--    displayed if a is more than b -->
<div   *ngIf="str == 'yes'"></div>  <!--    displayed if str is the string "yes" -->
<div   *ngIf="myFunc()"></div>      <!--    displayed if myFunc returns truthy →
```

6.2. NgSwitch

Contoh :

```
<div class="container" [ngSwitch]="myVar">
   <div *ngSwitchCase="'A'">Var is A</div>
   <div *ngSwitchCase="'B'">Var is B</div>
   <div *ngSwitchDefault>Var is something else</div>
</div>
```

Contoh lain :
(dalam contoh ini bila choice diisi = 2, maka akan tampil dua item)

```
<h4 class="ui horizontal divider header">
   Current choice is {{ choice }}
</h4>

<div class="ui raised segment">
   <ul [ngSwitch]="choice">
      <li *ngSwitchCase="1">First choice</li>
      <li *ngSwitchCase="2">Second choice</li>
      <li *ngSwitchCase="3">Third choice</li>
      <li *ngSwitchCase="4">Fourth choice</li>
      <li *ngSwitchCase="2">Second choice, again</li>
      <li *ngSwitchDefault>Default choice</li>
   </ul>
</div>
```

6.3. NgStyle

Dengan `NgStyle`, kita dapat menset CSS property dari suatu elemen DOM dari suatu ekspresi Angular.

Penggunaan paling mudah dari instruksi ini adalah dengan melakukan `[style.<cssproperty>]="value"`, contoh :

```
  <div [style.background-color]="'yellow'">
     Uses fixed yellow background
  </div>
```

Cara lain adalah dengan menggunakan `ngStyle` yang menggunakan bentuk pasangan key-value untuk tiap property yang ingin di set, contoh :

```
<div [ngStyle]="{color: 'white', 'background-color': 'blue'}">
  Uses fixed white text on blue background
</div>
```

Kekuatan sesungguhnya dari `NgStyle` adalah dalam penggunaannya dengan nilai dinamis.
Contoh :

```
<div class="ui input">
  <input type="text" name="color" value="{{color}}" #colorinput>
</div>

<div class="ui input">
  <input type="text" name="fontSize" value="{{fontSize}}" #fontinput>
</div>
```

```
<button class="ui primary button" (click)="apply(colorinput.value, fontinput.value)">
  Apply settings
</button>
```

Function apply melakukan :

```
   apply(color: string, fontSize: number): void {
      this.color = color;
      this.fontSize = fontSize;
   }
```

Penggunaan `fontSize` :

```
<div>
   <span [ngStyle]="{color: 'red'}" [style.font-size.px]="fontSize">
      red text
   </span>
</div>
```

Penting untuk diingat bahwa kita harus menyebutkan unit yang diinginkan, misal `font-size = 12` bukanlah nilai yang valid bagi CSS, kita harus menyebutkan unitnya misal : `12px` atau `1.2em`.
Angular menyediakan syntax yang membantu dalam menetapkan unit, yaitu dengan notasi `[style.font-size.px]`.
Suffix `.px` menandakan bahwa kita melakukan set pada property `font-size` dengan nilai dalam pixel.
Dapat juga diterapkan untuk `[style.font-size.em]` maupun `[style.font-size.%]`.

Penggunaan `color` :

```
<h4 class="ui horizontal divider header">
   ngStyle with object property from variable
</h4>

<div>
   <span [ngStyle]="{color: color}">
      {{ color }} text
   </span>
</div>

<h4 class="ui horizontal divider header">
   style from variable
</h4>

<div [style.background-color]="color"
       style="color: white;">
   {{ color }} background
</div>
```

6.4. NgClass

Digunakan untuk secara dinamis men-set dan mengubah class CSS milik elemen DOM tertentu.
Directive ini digunakan dengan mengirimkan sebuah object literal (red: object dalam notasi *json*).
Object ini diharapkan memiliki key yaitu nama class dan nilai true/false untuk menunjukkan apakah class tersebut hendak diterapkan atau tidak.

Misal dimiliki class CSS :

```
.bordered {
   border: 1px dashed black;
   background-color: #eee; }
```

Berikut contoh penggunaan `ngClass` pada dua `div` dimana yang pertama selalu tidak memiliki border sedang `div` kedua selalu memiliki border :

```
     <div [ngClass]="{bordered: false}">This is never bordered</div>
     <div [ngClass]="{bordered: true}">This is always bordered</div>
```

Tentu saja akan lebih bermanfaat untuk menggunakan `NgClass` dengan expression agar penggunaan class lebih dinamis :

```
    <div [ngClass]="{bordered: isBordered}">
      Using object literal. Border {{ isBordered ? "ON" : "OFF" }}
    </div>
```

NB : `isBordered` adalah suatu variabel dari component ini.

Alternatif lain adalah dengan menggunakan object :

```
@Component({
   selector: 'app-ng-class-example',
   templateUrl: './ng-class-example.component.html'
})
export class NgClassExampleComponent implements OnInit {
   isBordered: boolean;
   classesObj: Object;
   classList: string[];

   constructor() {
   }

   ngOnInit() {
       this.isBordered = true;
       this.classList = ['blue', 'round'];
       this.toggleBorder();
   }

   toggleBorder(): void {
       this.isBordered = !this.isBordered;
       this.classesObj = {
          bordered: this.isBordered
       };
   }
...
```

Penggunaan `classesObj` :

```
   <div [ngClass]="classesObj">
       Using object var. Border {{ classesObj.bordered ? "ON" : "OFF" }}
   </div>
```

Kita juga dapat menggunakan daftar nama class yang mana nama class ini hendak ditambahkan ke elemen.
Ini dapat dilakukan dengan mengirimkan array literal :

```
   <div class="base" [ngClass]="['blue', 'round']">
       This will always have a blue background and
       round corners
   </div>
```

Maupun dengan melalui property di component :

```
  this.classList = ['blue', 'round'];
```

dan kirimkan ke NgClass :

```
   <div class="base" [ngClass]="classList">
```

```
      This is {{ classList.indexOf('blue') > -1 ? "" : "NOT" }} blue
      and {{ classList.indexOf('round') > -1 ? "" : "NOT" }} round
  </div>
```

Pada contoh ini daftar nama class di `classList` akan digabung dengan class yang telah ada di element tersebut (yaitu `base`), sehingga elemen tersebut akan memiliki tiga class CSS yang digunakan : `base`, `blue` dan `round` seperti ini :

```
<div class="base blue round"> ...
```

6.5. NgFor

`NgFor` digunakan untuk mengulang elemen DOM dan mengirimkan satu elemen dari array pada tiap iterasinya.

Syntax : `*ngFor="let item of items"`

Contoh :
property pada component :

```
this.cities = ['Miami', 'Sao Paulo', 'New York'];
```

penggunaan :

```
<div class="ui list" *ngFor="let c of cities">
  <div class="item">{{ c }}</div>
</div>
```

Contoh lain :
property pada component :

```
this.people = [
          { name: 'Anderson', age: 35, city: 'Sao Paulo' },
          { name: 'John', age: 12, city: 'Miami' },
          { name: 'Peter', age: 22, city: 'New York' }
];
```

penggunaan :

```
<table class="ui celled table">
  <thead>
      <tr>
         <th>Name</th>
         <th>Age</th>
         <th>City</th>
      </tr>
  </thead>
  <tr *ngFor="let p of people">
      <td>{{ p.name }}</td>
      <td>{{ p.age }}</td>
      <td>{{ p.city }}</td>
  </tr>
</table>
```

Contoh nested `NgFor` :

```
<div *ngFor="let item of peopleByCity">
  <h2 class="ui header">{{ item.city }}</h2>

  <table class="ui celled table">
      <thead>
         <tr>
            <th>Name</th>
```

```
            <th>Age</th>
          </tr>
        </thead>
        <tr *ngFor="let p of item.people">
          <td>{{ p.name }}</td>
          <td>{{ p.age }}</td>
        </tr>
    </table>
</div>
```

Getting an index
Ada kalanya ingin diketahui berapa nomor index untuk tiap item dari array yang diiterasi.
Hal ini dapat dilakukan dengan cara :

```
<div class="ui list" *ngFor="let c of cities; let num = index">
  <div class="item">{{ num+1 }} - {{ c }}</div>
</div>
```

6.6. NgNonBindable

`NgNonBindable` digunakan untuk memberitahu Angular agar tidak meng-compile atau menghubungkan section tertentu dari page.
Sebagai contoh kita hendak me-render literal text `{{ content }}` pada template kita.
Biasanya text tersebut akan terhubung ke variabel karena digunakan syntax `{{ }}`.
Bagaimana caranya bila kita memang benar-benar ingin menampilkan teks `{{ content }}` pada layar ?
Untuk ini digunakan `NgNonBindable`.

Contoh :

```
<div class='ngNonBindableDemo'>
   <span class="bordered">{{ content }}</span>
   <span class="pre" ngNonBindable>
      &larr; This is what {{ content }} rendered
   </span>
</div>
```

7. Forms

Form dapat menjadi sangat kompleks karena :
- Input pada form ditujukan untuk memodifikasi data, baik di page maupun di server
- Perubahan seringkali harus direfleksikan/ditampilkan di suatu tempat di page
- User memiliki banyak kelonggaran atas apa yang mereka masukkan, jadi kita harus melakukan validasi atas nilai tersebut
- UI perlu secara jelas menyebutkan apa yang diharapkan dan error yang terjadi bila ada
- Field dapat memiliki logika yang kompleks
- Diperlukan kemampuan untuk men-test form tanpa bergantung pada DOM selector

Angular menyediakan tools untuk mengatasi kompleksitas di atas :
- FormControls
  Untuk membungkus (encapsulate) input di form dan memberi kita object untuk digunakan dengannya.
- Validators
  Memberikan kemampuan untuk mem-validasi input sesuai dengan kebutuhan kita
- Observers
  Memungkinkan kita memantau perubahan yang terjadi pada form dan bereaksi sesuai dengan kebutuhan.

Dua object mendasar di form Angular adalah `FormControl` dan `FormGroup`.

7.1. FormControls

Sebuah `FormControl` mewakili sebuah input field, ini adalah unit terkecil dari form Angular.

`FormControl` membungkus nilai suatu field dan keadaannya seperti misalnya : valid, dirty (changed) atau memiliki error.

Berikut contoh penggunaan `FormControl` di TypeScript :

```
// create a new FormControl with the value "Nate"
let nameControl = new FormControl("Nate");

let name = nameControl.value; // -> Nate

// now we can query this                    control for certain values:
nameControl.errors // ->                     StringMap<string, any> of errors
nameControl.dirty // ->                      false
nameControl.valid // ->                      true
// etc.
```

Berikut sepotong contoh `FormControl` dari suatu template :

```
<!-- part of some bigger form -->
<input type="text" [formControl]="name" />
```

Dengan menuliskan hal di atas, kita menciptakan suatu object `FormControl`.

7.2. FormGroup

Kebanyakan form memiliki lebih dari satu field, jadi kita memerlukan sebuah cara untuk mengelola beberapa `FormControl`.
Berikut cara menciptakan sebuah `FormGroup` :

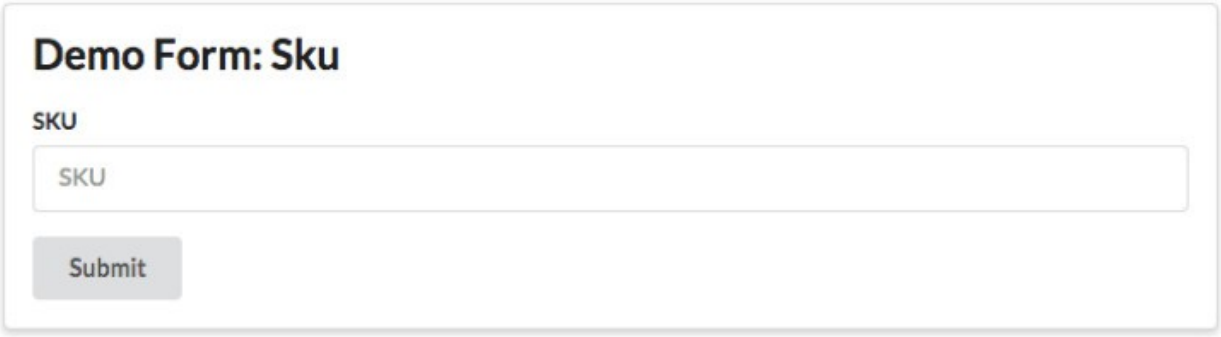
```
let personInfo = new FormGroup({
      firstName: new FormControl("Nate"),
      lastName: new FormControl("Murray"),
      zip: new FormControl("90210")
})
```

Untuk memeriksa status atau nilai dari `personInfo` dapat dilakukan dengan mudah :

```
personInfo.value;
// -> {
//     firstName: "Nate",
//     lastName: "Murray",
//     zip: "90210"
// }

// now we can query this control group for certain values, which have sensible
// values depending on the children FormControl's values:
personInfo.errors // -> StringMap<string, any> of errors
personInfo.dirty // -> false
personInfo.valid // -> true
// etc.
```

Perlu diingat bahwa ketika kita mengambil nilai dari `FormGroup` kita menerima sebuah object yang berisi pasangan key-value.

Untuk memperjelas mengenai form, berikut bentuk form yang akan kita buat :



NB:

SKU = stock keeping unit, adalah istilah bagi id unique suatu produk yang digunakan sebagai tracking pada inventory.

7.3. Loading the FormsModule

Untuk menggunakan library form, kita harus meng-import library untuk form di `NgModule`.

Ada dua cara menggunakan form di Angular yaitu dengan menggunakan `FormsModule` atau menggunakan `ReactiveFormsModule` dan karena pada contoh ini kita akan menggunakan keduanya, maka kita akan import keduanya di module.

```
import {
   FormsModule,
   ReactiveFormsModule
} from '@angular/forms';

// farther down...

@NgModule({
   declarations: [
      FormsDemoApp,
      DemoFormSkuComponent,
      // ... our declarations         here
   ],
   imports: [
      BrowserModule,
      FormsModule,                //  <-- add this
      ReactiveFormsModule //          <-- and this
   ],
   bootstrap: [ FormsDemoApp          ]
})
class FormsDemoAppModule {}
```

FormsModule memungkinkan kita menggunakan directive seperti :

- ngModel
- NgForm

Sedangkan ReactiveFormsModule memungkinkan kita menggunakan directive :

- formControl
- ngFormGroup

Ciptakan component untuk form ini :

```
import { Component, OnInit } from '@angular/core';

@Component({
    selector: 'app-demo-form-sku',
    templateUrl: './demo-form-sku.component.html',
```

Pada file demo-form-sku.component.html :

```
<div class="ui raised segment">
   <h2 class="ui header">Demo Form: Sku</h2>
   <form #f="ngForm"
           (ngSubmit)="onSubmit(f.value)"
           class="ui form">

       <div class="field">
         <label for="skuInput">SKU</label>
         <input type="text"
                  id="skuInput"
                  placeholder="SKU"
                  name="sku" ngModel>
       </div>

       <button type="submit" class="ui button">Submit</button>
   </form>
</div>
```

NB :

`class = "ui form"` dan `class = "field"` ini benar-benar opsional, ia ada karena pada contoh ini kita menggunakan Semantic UI

form & NgForm

Karena kita telah mengimport `FormsModule`, ini membuat `NgForm` tersedia untuk dipakai pada view.
`NgForm` secara otomatis akan menempel pada tag `<form>` yang ada.

Ada dua fungsionalitas penting yang diberikan oleh `NgForm` :

1. Sebuah FormGroup bernama `ngForm`
2. Sebuah output (`ngSubmit`)

Tanda `#` menandakan bahwa kita ingin menciptakan sebuah variabel local `f` bagi view ini.
Object `ngForm` bertipe `FormGroup`, ini berarti kita dapat menggunakan `f` sebagai sebuah `FormGroup` di view tersebut, dan inilah yang kita lakukan pada output (`ngSubmit`).

Kita menghubungkan action `ngSubmit` dengan syntax : `(ngSubmit)="onSubmit(f.value)"`

- `(ngSubmit)` berasal dari `NgForm`
- `onSubmit()` adalah apa yang hendak dilakukan saat submit
- `f.value`, `f` adalah `FormGroup` dan `.value` akan mengembalikan pasangan key-value dari `FormGroup` ini.

input & ngModel

Directive `ngModel` menetapkan selector dari `ngModel`, Ini berarti kita dapat menempelkannya ke tag `<input>` dengan atribut : `ngModel="whatever"`. Dalam contoh ini kita mendefiniskan `ngModel` tanpa nilai atribut.
Dengan mendefinisikan `ngModel` tanpa atribut ini artinya :

1. ini adalah one-way data binding
2. kita hendak menciptakan sebuah `FormControl` pada form ini dengan nama `sku` (isi dari atribut `name` dari tag `<input>`)

`NgModel` menciptakan sebuah `FormControl` baru yang secara otomatis ditambahkan ke `FormGroup` (dalam hal ini adalah pada form) dan kemudian menghubungkan sebuah elemen DOM ke `FormControl` baru tersebut. Dengan ini, ia men-set up hubungan antara input tag dengan `FormControl` dan hubungan tersebut berdasarkan sebuah nama : `sku`.

Pada definisi class untuk component tersebut (`demo-form-sku.component.ts`) :

```
export class DemoFormSkuComponent implements OnInit {

   constructor() { }
```

```
   ngOnInit() {
   }

   onSubmit(form: any): void {
      console.log('you submitted value:', form);
   }

}
```

Bila aplikasi di atas dijalankan ia akan menghasilkan suatu form dengan isi : `{sku: "ABC"}` (dengan asumsi yang diisikan ke layar adalah "ABC")

7.4. FormBuilder (Reactive Form)

Cara lain dalam membangun form adalah dengan menggunakan `FormBuilder`.
`FormBuilder` membantu kita membuat `FormControl` dan `FormGroup`.

Pertama-tama kita harus melakukan import atas `FormBuilder` :

Pada file `demo-form-sku-with-builder.component.ts` :

```
import { Component, OnInit } from '@angular/core';
import {
    FormBuilder,
    FormGroup
} from '@angular/forms';
```

Penggunaan `FormBuilder`
Kita melakukan inject `FormBuilder` ke dalam class component dengan menciptakan argument pada constructor class component :

```
import { Component, OnInit } from '@angular/core';
import {
   FormBuilder,
   FormGroup
} from '@angular/forms';

@Component({
   selector: 'app-demo-form-sku-with-builder',
   templateUrl: './demo-form-sku-with-builder.component.html',
   styles: []
})
export class DemoFormSkuWithBuilderComponent implements OnInit {
   myForm: FormGroup;

   constructor(fb: FormBuilder) {
      this.myForm = fb.group({
         'sku': ['ABC123']
       });
    }

    ngOnInit() {
    }

    onSubmit(value: string): void {
       console.log('you submitted value: ', value);
    }

}
```

Ada dua fungsi utama yang akan kita gunakan dari `FormBuilder` yaitu :

- control (untuk menciptakan `FormControl` baru)
- group (untuk menciptakan `FormGroup` baru)

Penggunaan `myForm`
Sebelumnya dijelaskan bahwa `ngForm` ditempelkan secara otomatis sewaktu kita menggunakan `FormsModule` dan juga `ngForm` menciptakan `FormGroup`nya sendiri.
Saat ini kita harus mengubah itu agar kita dapat menggunakan variabel `myForm` yang telah kita ciptakan menggunakan `FormBuilder`.

Pada file `demo-form-sku-with-builder.component.html` :

```
<h2 class="ui header">Demo Form: Sku with Builder</h2>
<form [formGroup]="myForm"
      (ngSubmit)="onSubmit(myForm.value)"
      class="ui form">
```

Di sini kita memberitahu Angular bahwa kita ingin menggunakan `myForm` sebagai `FormGroup` dari form ini.

Kita juga perlu mengubah onSubmit untuk menggunakan `myForm` bukan `f`.
Dan akhirnya kita perlu menghubungkan `FormControl` kita ke input tag.

Pada file `demo-form-sku-with-builder.component.html` :

```
<label for="skuInput">SKU</label>
<input type="text"
       id="skuInput"
       placeholder="SKU"
       [formControl]="myForm.controls['sku']">
```

Di sini kita menginstruksikan `formControl` untuk melihat ke `myForm.controls` dan menggunakan `FormControl sku` sebagai input.

7.5. Validation

7.5.1 Adding Validation

User tidak selalu memasukkan data dengan format yang benar. Bila format data yang dimasukkan salah, kita akan memberi feedback dan tidak mengijinkan form disubmit.
Untuk ini kita menggunakan validators.
Validators disediakan oleh module Validators dan validator paling sederhana adalah `Validators.required` yang mana menandakan bahwa field tersebut dibutuhkan (required) bila tidak maka `FormControl` akan dinyatakan tidak valid.

Untuk menggunakan validators kita perlu :
1. menempelkan validator ke object `FormControl`
2. memeriksa status dari validator di view (template) dan melakukan aksi yang sesuai.

Untuk menempelkan validator ke object `FormControl` dilakukan dengan memberikan argument ke dua di constructor `FormControl` :

Pada file `demo-form-with-validations-explicit.component.ts` :

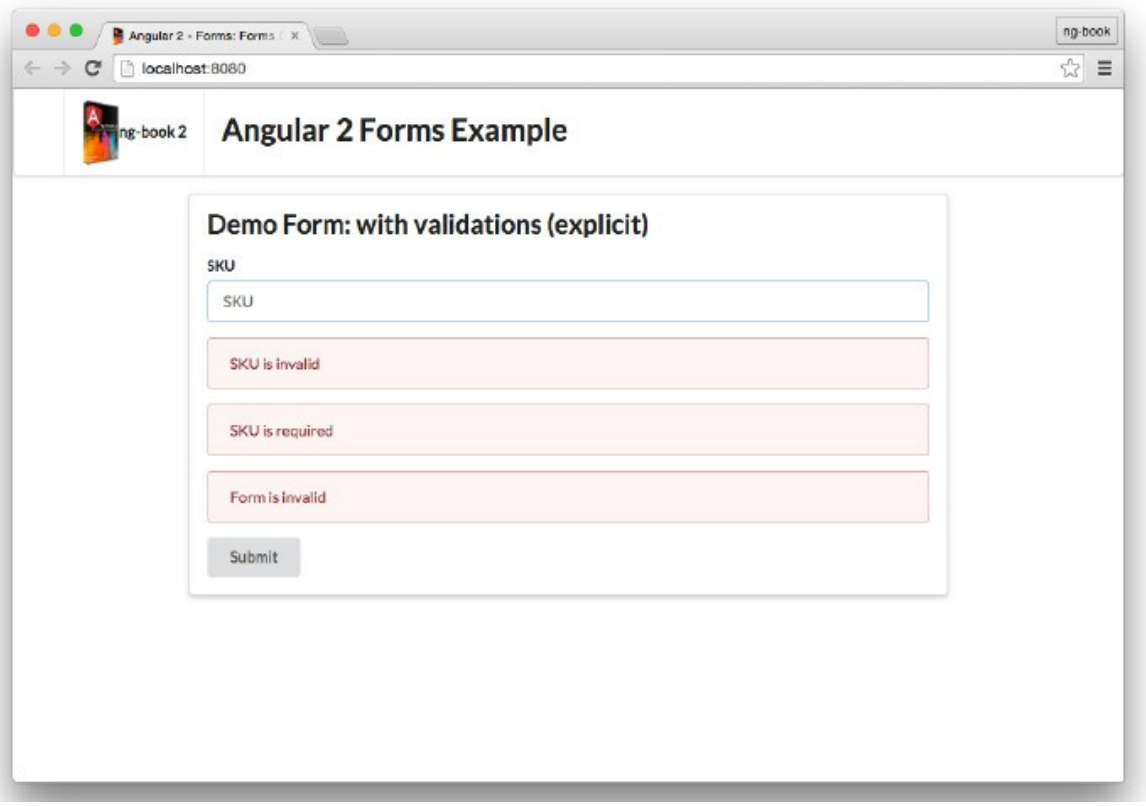
```
constructor(fb: FormBuilder) {
   this.myForm = fb.group({
      'sku': ['', Validators.required]
   });

   this.sku = this.myForm.controls['sku'];
}
```

Sekarang kita perlu menggunakan validasi ini pada view (template). Ada dua cara untuk meng-akses nilai validasi pada view :
1. Kita dapat secara eksplisit memasukkan FormControl `sku` ke sebuah instance variabel dari

class
2. Kita dapat mencari FormControl `sku` dari `myForm` pada view.
   Ini memberikan pekerjaan yang lebih sedikit pada component class.

Untuk memperjelas hal di atas, berikut contoh bagi kedua cara tersebut :

Tampilan yang diharapkan :



Cara paling fleksibel untuk berurusan dengan individual `FormControl` di view adalah dengan men-set tiap `FormControl` sebagai sebuah variabel pada component class.

Pada `demo-form-with-validations-explicit.component.ts` :

```
export class DemoFormWithValidationsExplicitComponent {
    myForm: FormGroup;
    sku: AbstractControl;

    constructor(fb: FormBuilder) {
      this.myForm = fb.group({
         'sku': ['', Validators.required]
      });

      this.sku = this.myForm.controls['sku'];
    }

    onSubmit(value: string): void {
       console.log('you submitted value: ', value);
    }

}
```

Perhatikan bahwa :
1. Kita men-setup `sku: AbstractControl`
2. Kita mengisi `this.sku` setelah menciptakan `myForm` dengan `FormBuilder`

Ini berarti bahwa kita dapat mengakses `sku` dimanapun pada view.
Kekurangannya adalah kita harus men-setup variabel untuk setiap field yang ada di form. Untuk form yang besar ini tentu akan bertele-tele.
Nah setelah kita mendapatkan hasil validasi dari sku, opsi untuk penampilannya di layar adalah :

1. Memeriksa validitas dari keseluruhan form dan menampilkan pesan
   Kita dapat memeriksa validitas dari keseleuruhan form dengan melihat pada `myForm.valid`.

   Pada file `demo-form-with-validations-explicit.component.html` :

   ```
   <div *ngIf="!myForm.valid"
   ```

   Ingat bahwa `myForm` adalah sebuah `FormGroup` dan `FormGroup` valid bila semua `FormControl` di dalamnya juga valid.

2. Memeriksa validitas dari individual field dan menampilkan pesan
   Kita juga dapat menampilkan pesan untuk field tertentu bila `FormControl` bagi field tersebut tidak valid

   Pada file `demo-form-with-validations-explicit/demo-form-with-validations-explicit.component.html` :

   ```
         [formControl]="sku">
             <div *ngIf="!sku.valid"
               class="ui error message">SKU is invalid</div>
             <div *ngIf="sku.hasError('required')"
   ```

3. Memeriksa validitas dari individual field dan mewarnai field tersebut dengan merah bila tidak valid
   Di sini digunakan Semantic UI class `.error`, ini artinya jika class `.error` ditambahkan ke `<div class = "field">` ia akan menampilkan tag `<input>` dengan border merah.
   Untuk melakukan ini, kita dapat menggunakan conditional pada property class :

   Pada file `demo-form-with-validations-explicit.component.html :`

   ```
   <div class="field"
        [class.error]="!sku.valid && sku.touched">
   ```

   Ide dari visualisasi di atas adalah : kita hanya menampilkan keadaan error jika user mencoba meng-edit form ("menyentuhnya") dan sekarang dia tidak valid.

4. Memeriksa validitas dari individual field pada kebutuhan tertentu dan menampilkan pesan atas itu.
   Input field dapat tidak valid untuk berbagai alasan, seringkali kita menginginkan untuk menampilkan pesan yang berbeda-beda tergantung dari penyebabnya.
   Untuk ini dapat digunakan method hasError :

   Pada file `demo-form-with-validations-explicit.component.html` :

   ```
   <div *ngIf="sku.hasError('required')"
     class="ui error message">SKU is required</div>
   ```

5. Perlu diingat bahwa `hasError` didefinisikan baik di `FormControl` maupun `FormGroup`, ini artinya kita dapat mengirimkan argument kedua untuk mencari field tertentu dari `FormGroup`.
   Contoh sebelumnya dapat juga dituliskan sebagai berikut :

   ```
   <div *ngIf="myForm.hasError('required', 'sku')"
      class="error">SKU is required</div>
   ```

Removing the sku instance variable

Pada contoh di atas, kita men-set `sku: AbstractControl.`
Seringkali kita tidak ingin menciptakan variabel untuk tiap `AbstractControl`, untuk itu kita dapat menggunakan property `myForm.controls` seperti ditunjukkan berikut ini :

Pada file `demo-form-with-validations-short-hand.component.html` :

```
      <label for="skuInput">SKU</label>
      <input type="text"
               id="skuInput"
               placeholder="SKU"
               [formControl]="myForm.controls['sku']">
       <div *ngIf="!myForm.controls['sku'].valid"
         class="ui error message">SKU is invalid</div>
       <div *ngIf="myForm.controls['sku'].hasError('required')"
```

Dengan cara ini kita dapat meng-akses `sku` control tanpa dipaksa untuk secara eksplisit menambahkannya sebagai sebuah variabel di component class.

7.5.2 Custom Validation

Untuk membuat custom validation mari kita amati source `Validators.required` dari Angular core :

```
export class Validators {
  static required(c: FormControl): StringMap<string, boolean> {
     return isBlank(c.value) || c.value == "" ? {"required": true} : null;
  }
```

Sebuat validator mengambil `FormControl` sebagai input dan mengembalikan sebuah `StringMap<string, boolean>` diamana key nya adalah "error code" dan nilainya true bila ia gagal.

Melanjutkan contoh kita di atas, misal `sku` harus dimulai dengan 123, maka kita dapat menulis validator seperti berikut :

Pada file `demo-form-with-custom-validation.component.ts` :

```
 function skuValidator(control: FormControl): { [s: string]: boolean } {
    if (!control.value.match(/^123/)) {
      return {invalidSku: true};
    }
 }
```

Validator ini akan mengembalikan kode error `invalidSku` jika nilai input tidak dimulai dengan 123.
(NB: `/^123/` adalah format *regex* (method `.match` biasanya menerima inputan *regex*))

Sekarang kita perlu menambahkan validator tersebut ke `FormControl`.
Namun ada sedikit masalah, kita telah memiliki validator pada `sku`.
Bagaimana caranya menambahkan beberapa validator pada suatu field ?
Untuk ini kita harus menggunakan `Validators.compose`

Pada `file demo-form-with-custom-validation.component.ts` :

```
    constructor(fb: FormBuilder) {
      this.myForm = fb.group({
         'sku': ['', Validators.compose([
           Validators.required, skuValidator])]
      });
```

`Validators.compose` membungkus dua validator dan menempelkan keduanya ke `FormControl`. `FormControl` tidak valid kecuali kedua validasi tersebut valid.

Penggunaan pada view :

Pada file `demo-form-with-custom-validation.component.html` :

```
<div *ngIf="sku.hasError('invalidSku')"
  class="ui error message">SKU must begin with <span>123</span></div>
```

7.6. Watching For Changes

Sejauh ini kita hanya mengambil nilai dari form saat form disubmit (via `onSubmit`).
Ada kalanya kita ingin memantau tiap perubahan nilai yang terjadi di sebuah control.

Baik `FormGroup` maupun `FormControl` memiliki sebuah `EventEmitter` yang dapat digunakan untuk memantau perubahan.
Untuk mengamati perubahan pada sebuah kontrol kita harus :

1. mendapat akses ke `EventEmitter` dengan memanggil `control.valueChanges`, lalu
2. menambahkan sebuah observer dengan menggunakan method `.subscribe`

Catatan :

`EventEmitter` adalah sebuah Observable.
Observable membuka suatu channel komunikasi secara terus menerus yang mana beberapa data dapat dikeluarkan dari waktu ke waktu.
`.subscribe` adalah suatu cara untuk "berlangganan" suatu channel komunikasi atas observable object, yang mana memungkinkan kita untuk "mendengarkan" tiap data yang muncul.

Contoh :

Pada file `demo-form-with-events/demo-form-with-events.component.ts` :

```
constructor(fb: FormBuilder) {
    this.myForm = fb.group({
        'sku': ['', Validators.required]
    });

    this.sku = this.myForm.controls['sku'];

    this.sku.valueChanges.subscribe(
        (value: string) => {
            console.log('sku changed to:', value);
        }
    );

    this.myForm.valueChanges.subscribe(
        (form: any) => {
            console.log('form changed to:', form);
        }
    );

}
```

Pada contoh di atas kita mengamati dua event yang berbeda : perubahan pada field sku dan perubahan pada form secara keseluruhan.
Coba jalankan program di atas untuk mengamati apa yang terjadi.

7.7. ngModel

`ngModel` digunakan untuk menghubungkan sebuah model ke sebuah form.
`ngModel` adalah special directive karena ia mengimplementasi *two-way data binding*.
Angular dibangun untuk secara umum untuk melakukan *one-way data flow*: top to down.
Namun bila menyangkut form, adakalanya dibutuhkan hubungan dua arah.

Mari kita coba unutk menambahkan input `productName` dan menggunakan `ngModel` untuk meng-sinkron-kan nilai variabel di component class dengan view (template).

Pada file `demo-form-ng-model.component.ts` :

```
export class DemoFormNgModelComponent {
    myForm: FormGroup;
    productName: string;

    constructor(fb: FormBuilder) {
      this.myForm = fb.group({
         'productName': ['', Validators.required]
      });
    }

    onSubmit(value: string): void {
      console.log('you submitted value: ', value);
    }
}
```

Pada file `demo-form-ng-model.component.html` :

```
   <label for="productNameInput">Product Name</label>
   <input type="text"
          id="productNameInput"
          placeholder="Product Name"
          [formControl]="myForm.get('productName')"
          [(ngModel)]="productName">
```

Perhatikan bahwa syntax dari `ngModel` terlihat unik karena menggunakan baik kurung siku ("`[`") maupun kurung biasa ("`(`"), ini adalah suatu tanda dari hubungan dua arah (two-way).

Hal lain adalah, di sini kita masih menggunakan `formControl` untuk menentukan bahwa input ini harus terhubung ke `FormControl` di form kita. Ini kita lakukan karena `ngModel` hanya menghubungkan input ke variabel, `FormControl` benar-benar adalah hal yang berbeda.

Berikutnya, tambahkan tampilan untuk nilai `productName` yang diinput tersebut pada view :

Pada file `demo-form-ng-model.component.html` :

```
   <div class="ui info message">
      The product name is: {{productName}}
   </div>
```