

*Knowing Make Everything Easier !*



# ANGULARJS UNTUK PEMULA

Agung Setiawan



Agung Setiawan



# Table of Contents

Introduction	0
Bab 1 Dasar-Dasar	1
Bab 2 Directive	2
Bab 3 Service	3
Bab 4 Route	4
Bab 5 \$watch, \$digest dan \$apply	5
Bab 6 Contoh Aplikasi CRUD Menggunakan Web API	6

Buku yang ditujukan untuk pemula ini membahas `AngularJs`, framework JavaScript besutan Google yang digunakan untuk frontend development khususnya Single Page Application (SPA).

Materi yang disajikan bersifat praktik dan diharapkan pembaca bisa langsung *coding* untuk langsung merasakan dan mulai mengenal `AngularJs`. Diharapkan tidak *copy - paste* karena belajar ngoding memang harus benaran ngoding.

`AngularJs` dalam buku ini merupakan versi 1, untuk versi 2 harap menunggu penulis mempelajarinya terlebih dahulu :D

Ebook ini gratis dan boleh disebarluaskan dengan syarat harus link dari domain ini, tidak boleh *dihost* pada domain lain.

Source code contoh-contoh program akan segera ditaruh Github, untuk saat ini mohon maaf masih belum :)

Buku ini ditulis oleh [Agung Setiawan](#). Jika ada pertanyaan, kritik, dan saran silahkan untuk mengirim email ke `com.agungsetiawan@gmail.com`.



# Bab 1 Dasar-Dasar

Materi yang dipelajari pada bab ini :

- Memasang AngularJs
- Hello World
- Workflow
- Filter
- Basic Directive

## 1.1 Memasang AngularJs

Proses instalasi atau pemasangan AngularJs ke dalam aplikasi yang akan kita bangun sangatlah mudah. Kita hanya perlu meng-*include* file AngularJs di dalam file HTML menggunakan tag `<script>` seperti layaknya file JavaScript biasa.

```
<html>
  <head>
    </head>
  <body>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.6/angular.min.js"></script>
  </body>
</html>
```



Atau jika pembaca lebih memilih untuk mengunduh filenya terlebih dahulu untuk direferensikan melalui folder lokal maka menjadi seperti di bawah ini.

```
<html>
  <head>
    </head>
  <body>

    <script src="js/angular.min.js"></script>
  </body>
</html>
```

## 1.2 Hello World

Supaya belajarnya jadi semangat maka akan saya tunjukkan sedikit kesaktian dari AngularJs. Idenya sangat sederhana namun akan membuat kita paham bagaimana *flow* dari AngularJs.

1. Buat sebuah file dengan nama **index.html** dan pasang AngularJs seperti pada contoh di atas.
2. Buat sebuah folder baru bernama **js** sejajar dengan file **index.html**.
3. Buat file dengan nama **app.js** di dalam **folder js** tadi dan isikan kode berikut.

```
var app=angular.module('FirstApp', []);
```

4. *Include* file **app.js** di dalam file **index.html** menggunakan tag `<script>`, letakkan pada bagian atas sebelum tag `</body>`.
5. Buat folder baru dengan nama **controllers** di dalam **folder js** dan buat file **MainController.js** di dalam folder baru ini.
6. Isikan kode berikut di dalam file **MainController.js**.

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.title='Belajar AngularJs';  
}]);
```

7. *Include* file **MainController.js** di dalam file **index.html** menggunakan tag `<script>`, letakkan pada bagian atas sebelum tag `</body>`.
8. Edit file **index.html** menjadi seperti tampak pada kode berikut.

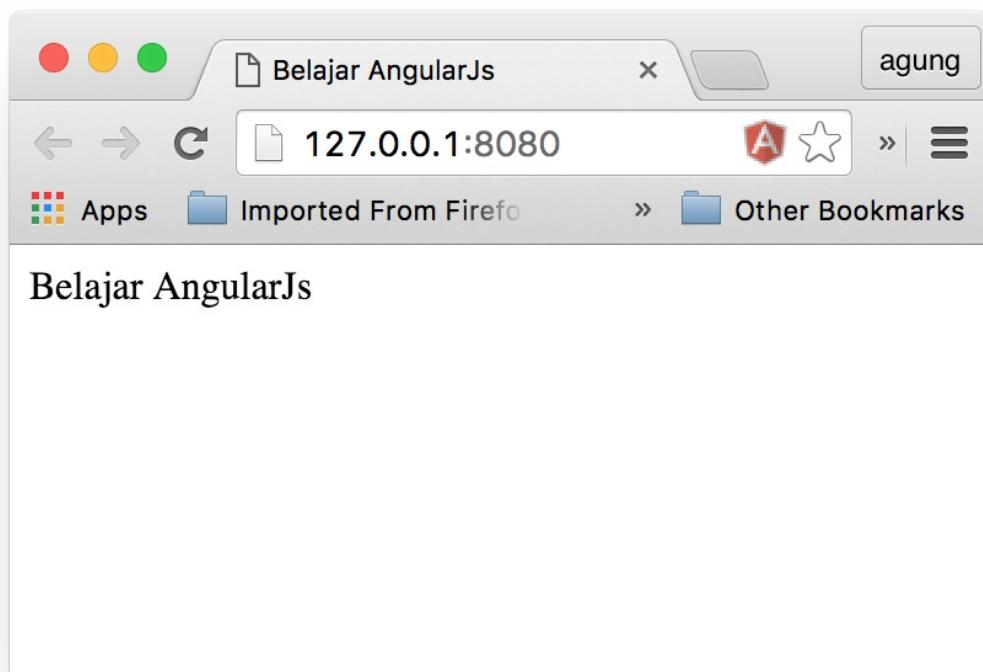
```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Belajar AngularJs</title>  
  </head>  
  <body ng-app="FirstApp">  
    <div ng-controller="MainController">  
  
    </div>  
  
    <script src="js/angular.min.js"></script>  
    <script src="js/app.js"></script>  
    <script src="js/controllers/MainController.js"></script>  
  </body>  
</html>
```

9. Jalankan **index.html** pada *browser*.

Aplikasi AngularJs yang paling sederhana (seperti di atas) terdiri dari 1 modul dan 1 controller. Apa yang ada pada file **app.js** adalah modul sedangkan apa yang ada pada file **MainController.js** adalah controller.

Supaya HTML tahu modul apa dan controller apa yang digunakan kita perlu memberitahunya dengan menggunakan directive `ng-app` untuk modul dan `ng-controller` untuk controller.

Kalau pembaca sudah benar melakukan langkah-langkah di atas maka hasil yang muncul pada *browser* seharusnya adalah seperti gambar berikut.



Sekarang coba ubah variabel **title** yang ada pada controller dengan kalimat sesuka pembaca dan kemudian *refresh browser* dan lihat perubahan yang terjadi.

Hal lain yang patut dicoba adalah menambahkan variabel baru dan kemudian coba dimunculkan pada *view*.

Pembahasan pada *controller*

```
app.controller('MainController',['$scope', function($scope){  
    $scope.title='Belajar AngularJs';  
    $scope.book={  
        title: 'Belajar AngularJs Bersama Agung Setiawan',  
        author : 'Agung Setiawan',  
        price  : 80000  
    };  
}]);
```

Perubahan pada *view*

```
<div ng-controller="MainController">  
    {{title}} <br>  
    {{book.title}} <br>  
    {{book.author}} <br>  
    {{book.price}}  
</div>
```



Sampai sini kita sudah belajar hal paling dasar untuk memahami AngularJs.

## 1.3 Workflow

Biasanya setiap *developer* memiliki *workflow* / urutan kerja yang berbeda dengan *developer* yang lain dalam mengerjakan aplikasi. Akan tetapi, seringnya ada suatu *workflow* yang bersifat umum yang diikuti oleh semua *developer*.

Dari langkah-langkah yang kita kerjakan pada sub bab 1.2 dapat disimpulkan bahwa *workflow* untuk membuat sebuah aplikasi AngularJs adalah :

1. Membuat module, *include* file modul di HTML kemudian memberi tahu HTML modul apa yang digunakan menggunakan `ng-app` .
2. Membuat `controller` , *include* file modul di HTML kemudian memberi tahu HTML `controller` apa yang digunakan menggunakan `ng-controller` .
3. Menambahkan data pada controller menggunakan `$scope` .
4. Menampilkan data di *view* menggunakan ekspresi ``.

## 1.4 Filter

Terdapat fitur pada AngularJs yang bernama *filter*, digunakan untuk memformat tampilan sebuah variabel di *view*. Jadi alih-alih kita melakukan *formatting* di controller, kita melakukannya di *view* dengan bantuan filter.

Pada contoh kode sebelumnya terlihat kalau harga yang kita ketikkan berupa angka biasa dan ketika ditampilkan di *view* maka hasilnya masih apa adanya, tanpa format mata uang.

Kita bisa menggunakan filter `currency` yang disediakan AngularJs untuk memformat harga sehingga enak dilihat.

```
<div ng-controller="MainController">
  {{title}} <br>
  {{book.title}} <br>
  {{book.author}} <br>
  {{book.price | currency}}
</div>
```

Penggunaan filter sangat menyenangkan seperti bisa dilihat pada listing kode di atas. Kita menambahkan sebuah tanda *pipeline* ( | ) dibelakang variabel yang ingin kita filter kemudian diikuti dengan filter yang kita gunakan. Setelah lakukan perubahan dengan menambahkan filter coba jalankan pada *browser*.

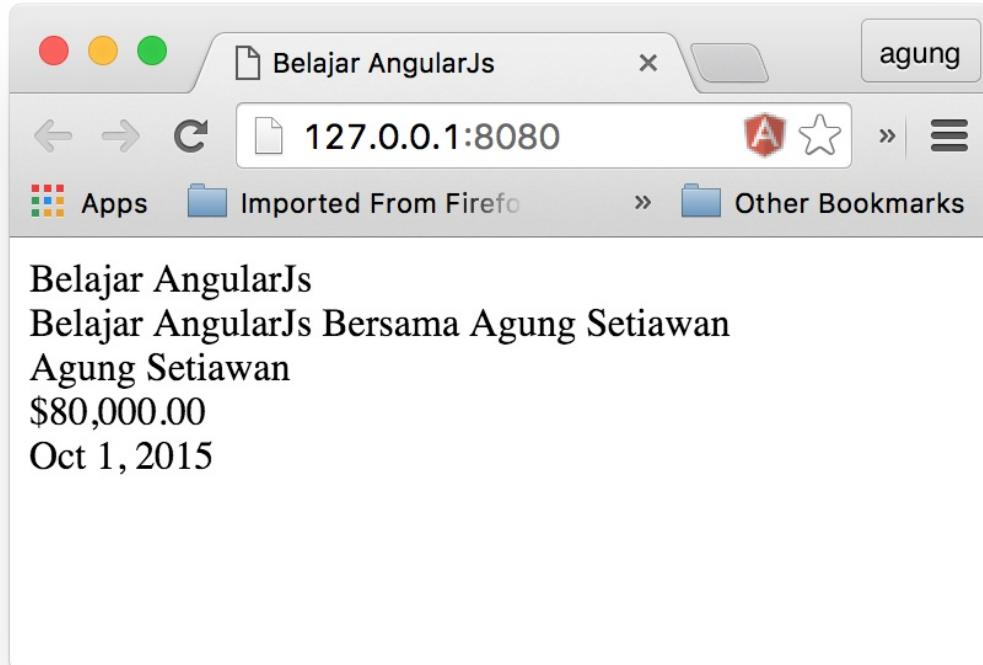


Ada juga filter lain yang bisa digunakan misalnya `date` untuk mengatur format tanggal. Sebelumnya tambahkan terlebih dahulu variabel baru di controller.

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.title='Belajar AngularJs';  
    $scope.book={  
        title: 'Belajar AngularJs Bersama Agung Setiawan',  
        author : 'Agung Setiawan',  
        price  : 80000,  
        pubdate : new Date('2015','09',01')  
    };  
}]);
```

Tampilkan variabel baru tersebut ke dalam view.

```
<div ng-controller="MainController">  
    {{title}} <br>  
    {{book.title}} <br>  
    {{book.author}} <br>  
    {{book.price | currency}} <br>  
    {{book.pubdate | date }}  
</div>
```



Ngomong-ngomong nih ada yang memperhatikan tidak kenapa bulan 09 kok jadinya Oktober yang seharusnya September?. Bertanya-tanya ya? Ini karena bulan di JavaScript dihitungnya mulai dari 0 bukan dari 1.

Untuk mengetahui filter apa saja yang telah disediakan oleh AngularJs, pembaca bisa merujuk ke dokumentasi yang ada di <https://docs.angularjs.org/api/ng/filter>.

Sudah pasti tidak semua filter yang tersedia mampu mengakomodasi kebutuhan kita seperti misalnya format mata uang dalam Rupiah. AngularJs memungkinkan bagi kita untuk membuat filter sendiri atau *custom filter*.

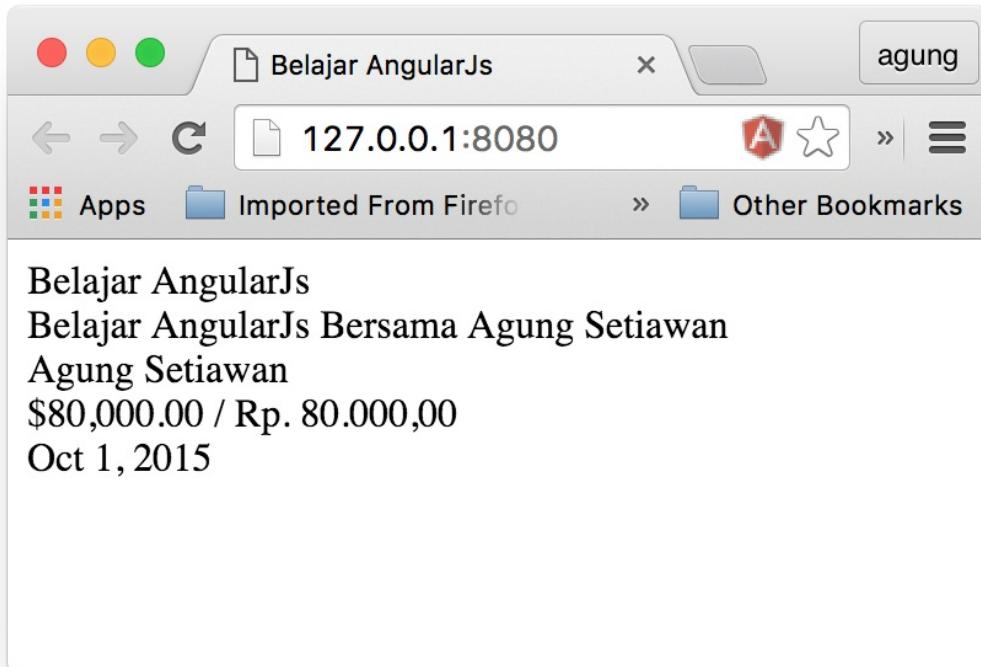
Misalkan kita ingin membuat filter untuk memformat angka menjadi mata uang Rupiah. Lakukan perubahan pada file **app.js**.

```
var app=angular.module('FirstApp',[]);
app.filter('rupiah', function(){
    return function toRp(angka){
        var rev = parseInt(angka, 10).toString().split('').reverse().join('');
        var rev2  = '';
        for(var i = 0; i < rev.length; i++){
            rev2 += rev[i];
            if((i + 1) % 3 === 0 && i !== (rev.length - 1)){
                rev2 += '.';
            }
        }
        return 'Rp. ' + rev2.split('').reverse().join('') + ',00';
    }
});
```

Jangan pusing dengan logika yang digunakan untuk memformat angka menjadi dalam bentuk mata uang rupiah, perhatikan saja bagaimana cara membuat filter yang sesuai dengan kebutuhan kita. Itu pun logika yang digunakan saya dapat dari *googling*. Setelah itu pakai filter untuk memformat angka harga di halaman *view*.

```
<div ng-controller="MainController">
    ...
    {{book.price | currency}} / {{book.price | rupiah}} <br>
    ...
</div>
```

Lihat perubahan pada browser dan silahkan berteriak "Wooooowwww!".



## 1.5 Basic Directive

*Directive* adalah tag khusus yang digunakan oleh AngularJs. Sampai saat ini kita baru mengenal 2 buah directive yaitu `ng-app` dan `ng-controller`. Sub bab ini akan membahas beberapa *directive* dasar yang wajib dimengerti.

### 1.5.1 ng-repeat

Tidak bisa tidak, perulangan `for each` dibutuhkan untuk mendapatkan isi dari variabel yang terdapat ada suatu koleksi / array. Bagi pembaca yang sudah familiar dengan bahasa pemrograman seperti C#, Java maupun PHP pasti sudah mengetahui kegunaannya dan sangat sering digunakan.

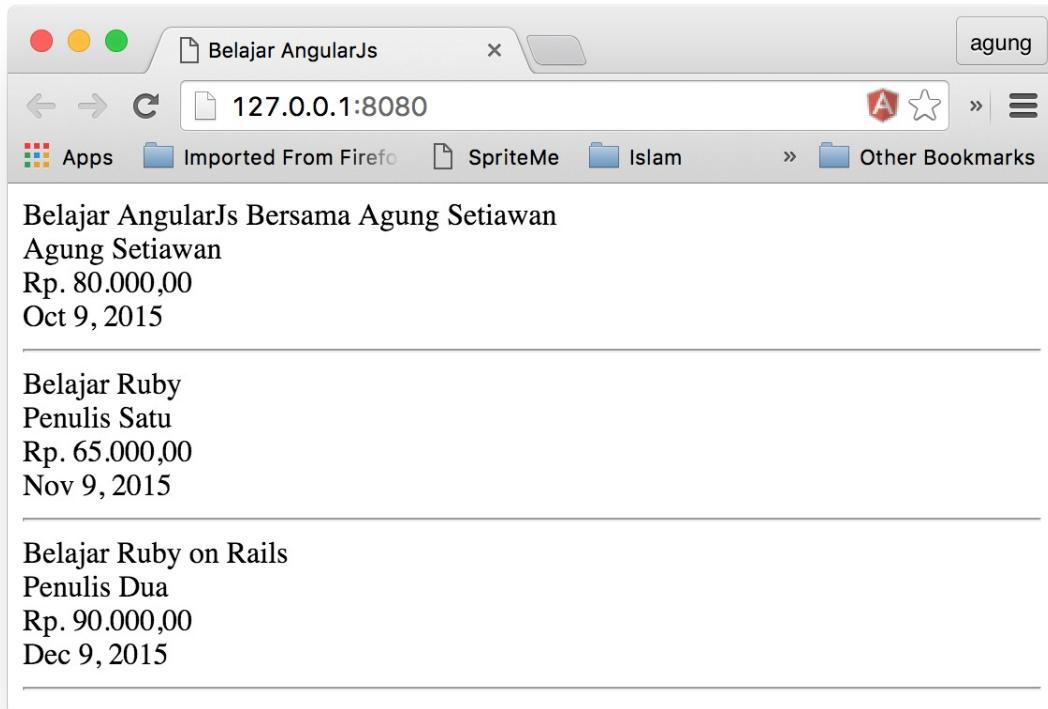
AngularJs menyediakan *directive* yang berguna untuk melakukan perulangan tersebut. Kita gunakan `ng-repeat` untuk melakukannya. Supaya lebih mudah dipahami maka saya tunjukkan langsung contoh kodennya. Masih menggunakan contoh yang sama maka pada file **MainController.js** hapus variabel `book` dan `title`. Sebagai gantinya buat variabel `books` yang berisi *array object*.

```
app.controller('MainController',['$scope', function($scope){
  $scope.books=[
    {
      title: 'Belajar AngularJs Bersama Agung Setiawan',
      author : 'Agung Setiawan',
      price  : 80000,
      pubdate : new Date('2015','09','09')
    },
    {
      title: 'Belajar Ruby',
      author : 'Penulis Satu',
      price  : 65000,
      pubdate : new Date('2015','10','09')
    },
    {
      title: 'Belajar Ruby on Rails',
      author : 'Penulis Dua',
      price  : 90000,
      pubdate : new Date('2015','11','09')
    }
  ];
}]);
```

Ingat *workflow*-nya, setelah menambahkan data di controller maka berikutnya adalah menampilkan data di *view*. Kali ini untuk menampilkan data buku yang ada pada variabel `books` digunakan `ng-repeat`.

```
<div ng-controller="MainController">
  <div ng-repeat="book in books">
    {{book.title}} <br>
    {{book.author}} <br>
    {{book.price | rupiah}} <br>
    {{book.pubdate | date}}
    <hr/>
  </div>
</div>
```

Jalankan di browser dan sekali lagi bolehlah berteriak "Wooooowwww!".



Beberapa variabel bawaan yang perlu diketahui berkaitan dengan ng-repeat.

1. \$index – Indeks elemen yang sedang diakses
2. \$first - Boolean yang menandakan elemen pertama atau bukan
3. \$middle - Boolean yang menandakan elemen bukan pertama dan bukan terakhir
4. \$last - Boolean yang menandakan elemen terakhir atau bukan
5. \$even - Boolean yang menandakan elemen berindeks genap
6. \$odd - Boolean yang menandakan elemen berindeks ganjil

## 1.5.2 ng-click

Directive berikutnya yang perlu dimengerti adalah `ng-click`. Sudah bisa diduga dari namanya ya directive ini pasti digunakan untuk menjalankan sesuatu ketika diklik.

Kembali saya kasih contoh penggunaannya.

```
app.controller('MainController',['$scope', function($scope){
    $scope.books=[  

        ...  

        ...  

    ];  
  

    $scope.logToConsole=function(index){  

        var book=$scope.books[index];  

        console.log(book);
    };
}]);
```

Di dalam file **MainController.js** kita tambahkan sebuah method baru dengan nama `logToConsole` yang memiliki sebuah parameter bernama indeks. Apa yang dilakukan method ini adalah mengambil satu data buku berdasarkan indeks array-nya dari variabel `books` kemudian dimunculkan melalui jendela `console`.

Method `logToConsole` ini yang akan kita gunakan di `view` bersama dengan `ng-click`. Berikut adalah kodennya.

```
<div ng-controller="MainController">  

    <div ng-repeat="book in books">  

        {{book.title}} <br>  

        {{book.author}} <br>  

        {{book.price | rupiah}} <br>  

        {{book.pubdate | date}} <br>  

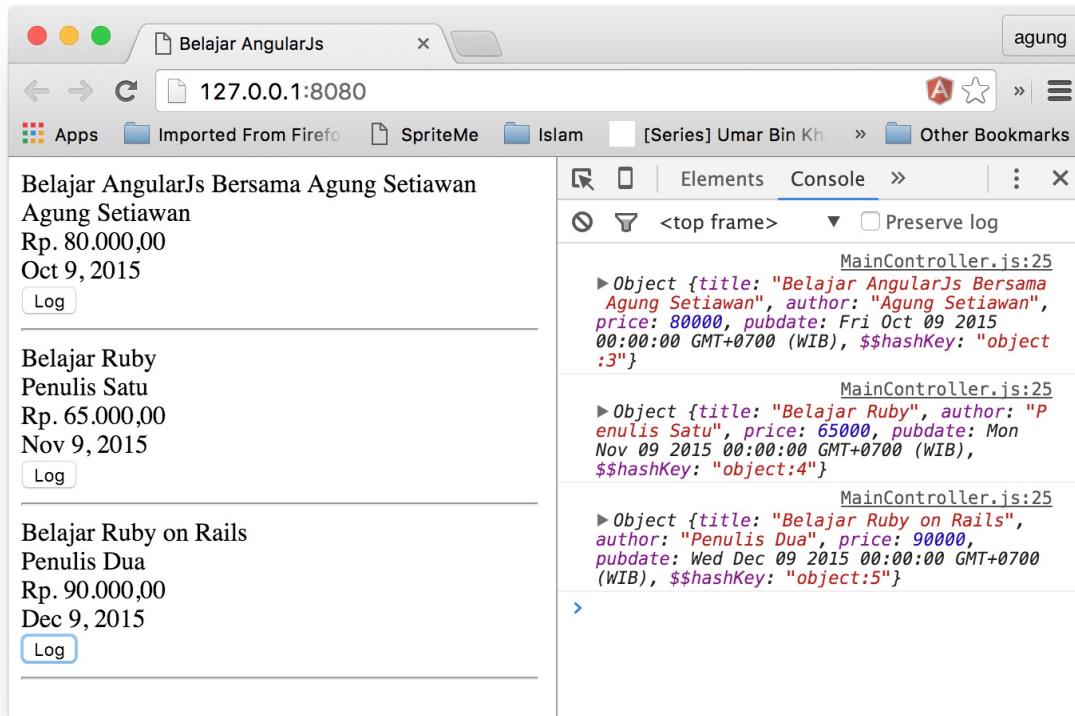
        <button ng-click="logToConsole($index)">Log</button>  

        <hr/>
    </div>
</div>
```

Jadi yang menjadi isi dari `directive` `ng-click` adalah nama method yang terdapat di controller, dalam hal ini adalah `logToConsole`.

Kode di atas sekaligus juga menunjukkan cara pengguna variabel `$index` yang sudah saya jelaskan pada sub bab sebelumnya yaitu tentang variabel-variabel yang ada di `ng-repeat`.

Ketika dijalankan pada `browser` cobalah untuk mengeklik button **Log** dan lihat hasil yang keluar di `console`.



Saya ingin pembaca benar-benar memahami materi ini. Oleh karena itu, mari kita buat sebuah contoh lagi yang menggunakan `ng-click`.

Sekarang, pada tiap array object pada variabel `books` tambahkan sebuah variabel baru dengan nama `likes` dan nilainya adalah 0.

```

$scope.books=[
{
  title: 'Belajar AngularJs Bersama Agung Setiawan',
  author : 'Agung Setiawan',
  price  : 80000,
  pubdate : new Date('2015','09','09'),
  likes : 0
},
{
  title: 'Belajar Ruby',
  author : 'Penulis Satu',
  price  : 65000,
  pubdate : new Date('2015','10','09'),
  likes : 0
},
{
  .....
}
];

```

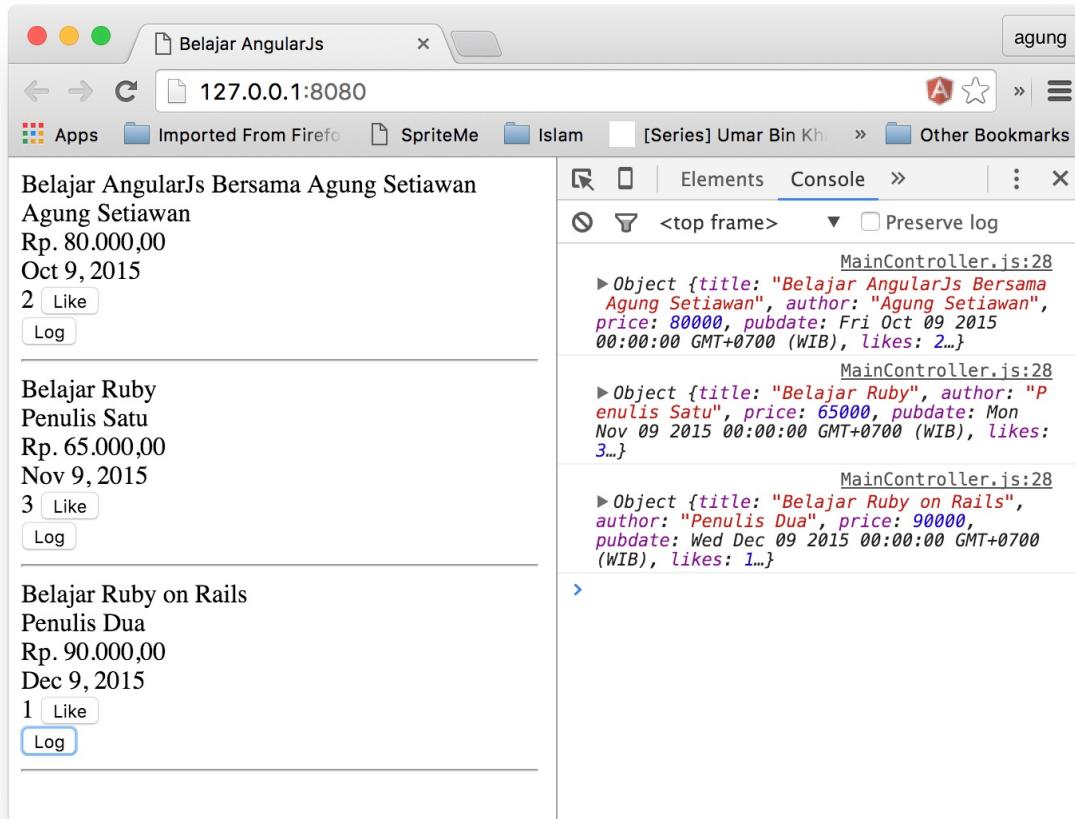
Kita akan menampilkan jumlah like pada *view* kemudian membuat sebuah tombol yang ketika diklik akan menambah jumlah likes sebanyak 1. Untuk keperluan itu kita butuh untuk menambah method baru.

Buka file **MainController.js** dan buat method baru seperti berikut

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.books=[  
        {  
            title: 'Belajar AngularJs Bersama Agung Setiawan',  
            author : 'Agung Setiawan',  
            price  : 80000,  
            pubdate : new Date('2015','09','09'),  
            likes : 0  
        },  
        {  
            ...  
            ...  
        }  
    ];  
  
    $scope.logToConsole=function(index){  
        var book=$scope.books[index];  
        console.log(book);  
    };  
  
    $scope.likes=function(index){  
        $scope.books[index].likes+=1;  
    };  
});
```

Sampai sini dapatkan logikanya bagaimana menambah nilai likes untuk masing-masing buku?. Seperti biasa, untuk mengetesnya perlu kita eksekusi di *browser* dan tekan tombol likes. Sebelumnya tentu buat terlebih dahulu bagian *view*-nya.

```
<div ng-controller="MainController">  
    <div ng-repeat="book in books">  
        {{book.title}} <br>  
        {{book.author}} <br>  
        {{book.price | rupiah}} <br>  
        {{book.pubdate | date}} <br>  
        {{book.likes}} <button ng-click="likes($index)">Like</button> <br>  
        <button ng-click="logToConsole($index)">Log</button>  
        <hr/>  
    </div>  
</div>
```



Jika kodingan pembaca belum berjalan sebagaimana mestinya coba untuk meneliti kode yang pembaca tulis dan baca lagi dari awal instruksi yang saya berikan dari awal secara lebih teliti.

## 1.5.2 ng-model

Di AngularJs terdapat sebuah jargon `two way data binding`. Maksudnya adalah kita bisa melakukan *data binding* dari kedua arah, yaitu dari controller ke *view* yang selama ini sudah lakukan dan yang satunya lagi yaitu arah kebalikan, dari *view* ke controller.

Untuk mengirim nilai dari *view* ke controller kita menggunakan *directive* `ng-model` yang berisi nama variabel yang terdapat pada controller.

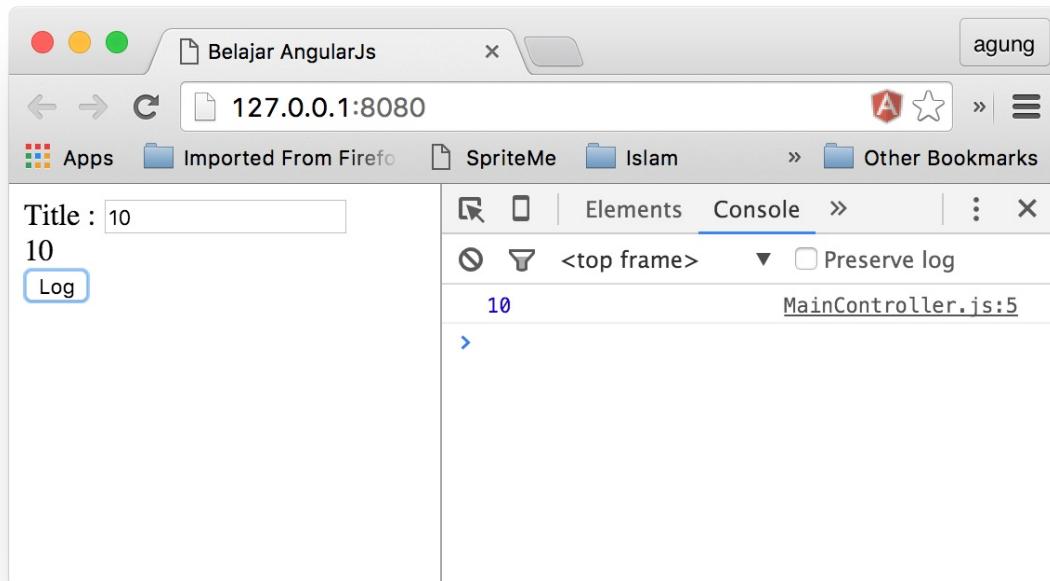
Pada controller siapkan sebuah variabel dengan nama `title` serta sebuah method untuk menge-log nilai dari variabel tersebut.

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.title=10;  
  
    $scope.log=function(){  
        console.log($scope.title);  
    };  
}]);
```

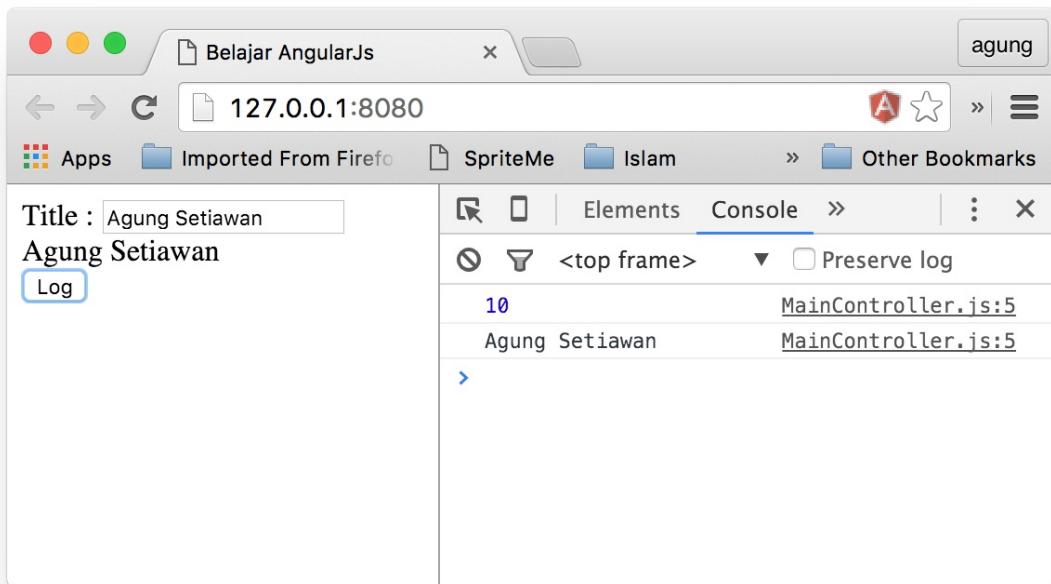
Buat view-nya yang berisi sebuah input dan sebuah ekspresi untuk menampilkan `title`. Pada input gunakan `ng-model`.

```
<div ng-controller="MainController">  
    Title : <input type="text" ng-model="title"/> <br>  
    {{title}} <br>  
    <button ng-click="log()">Log</button>  
</div>
```

Pada saat pertama kali dijalankan maka akan muncul nilai 10 baik pada ekspresi maupun pada bagian input. Hal ini dikarenakan nilai bawaan dari `title` **adalah** 10 seperti yang tertulis di controller. Coba tekan tombol Log dan hasil pada `console` juga 10.



Coba ubah nilai yang ada pada input maka nilai pada ekspresi pun akan ikut berubah seketika itu juga. Jangan lupa untuk menekan tombol Log maka nilai yang dimunculkan di `console` juga sesuai dengan apa yang ada kini pada input.



# Bab 2 Directive

Materi yang dipelajari pada bab ini :

- Membuat custom directive
- scope
- link function

## 2.1 Membuat Custom Directive

Setelah mempelajari beberapa *directive* dasar yang wajib diketahui untuk menunjang pekerjaan pembuatan aplikasi menggunakan AngularJs, sekarang saatnya belajar membuat *directive* sendiri yang sesuai dengan kebutuhan.

Secara penggunaan tag, *directive* di AngularJs terbagi menjadi 4, yaitu :

1. Element Directive ( `<my-directive></my-directive>` )
2. Attribute Directive ( `<div my-directive></div>` )
3. Class Directive ( `<div class="my-directive"></div>` )
4. Comment Directive ( `<!--directive:my-directive-->` )

Tujuan dari penulisan buku ini adalah saya ingin mengenalkan dasar-dasar dari AngularJS. Berkaitan dengan tujuan tersebut, karena materi *directive* ini sangat dalam dan rumit maka saya hanya menulis hal dasar yang sekiranya penting.

Sekarang mari kita lihat struktur paling dasar dari *custom directive* di AngularJs. Secara kodingan cara me-*register* *custom directive* hampir sama dengan me-*register* controller, beda di method yang digunakan dan nilai kembalian. Saat membuat *directive* kita diharuskan untuk mengembalikan objek *directive* yang memiliki beberapa property.

```
app.directive('myDirective',function(){
  return {
    restrict:'EA',
    template:'<h2>Hello {{dunia}}</h2>'
  };
});
```

Tulis kode di atas pada file **MyDirective.js** yang ada pada folder **js/directives**. Jangan lupa untuk membuat AngularJs modul terlebih dahulu dan pasang di *view workflow* yang biasa. Buat juga controller-nya sebagai berikut.

```
app.controller('MainController',['$scope', function($scope){  
    $scope.dunia="Dunia Indonesia";  
}]);
```

Untuk *view* bisa kita tampilkan *directive* yang kita buat tadi.

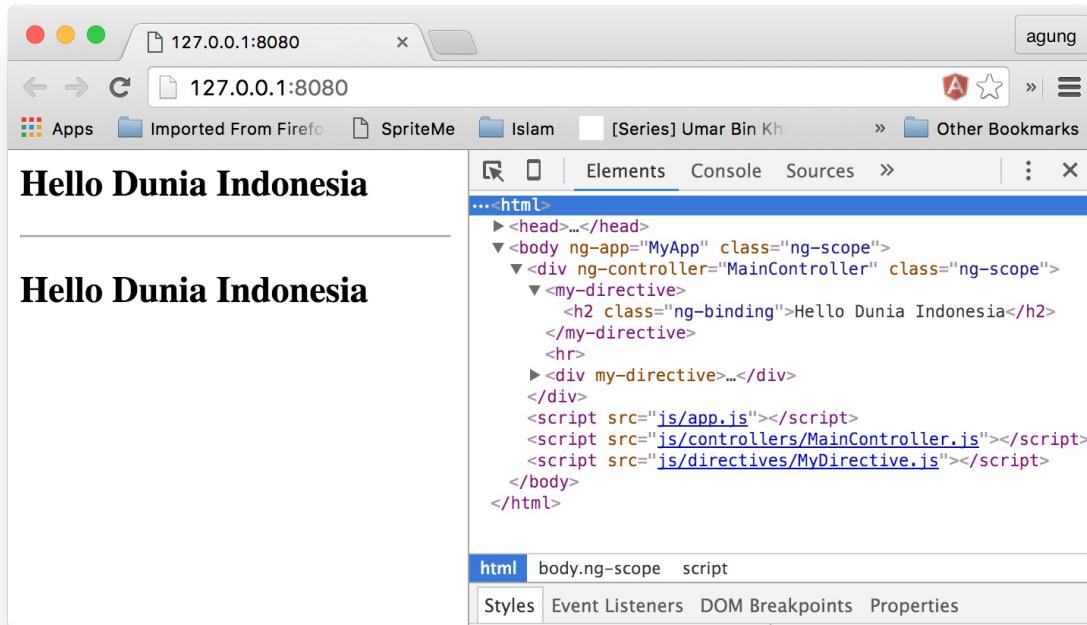
```
<html>  
  <head>  
  </head>  
  <body ng-app="MyApp">  
    <div ng-controller="MainController">  
      <my-directive></my-directive>  
      <hr/>  
      <div my-directive></div>  
    </div>  
  
    <script src="js/angular.min.js"></script>  
    <script src="js/app.js"></script>  
    <script src="js/controllers/MainController.js"></script>  
    <script src="js/directives/MyDirective.js"></script>  
  </body>  
<html>
```

Karena kita mengatur nilai property *restricted* sebagai `EA` yang artinya Element dan Attribute maka kita bisa menggunakan *directive* baik sebagai *element* maupun *attribute*. Untuk *class* dan *comment*, string yang digunakan berurutan adalah `c` dan `M`.

```
<my-directive></my-directive>  
<div my-directive></div>
```

Ngomong-ngomong kenapa *directive* kita menggunakan `my-directive` padahal kita di kodingan menuliskannya dengan `myDirective`? Itu adalah *convention* dari AngularJs.

Sekarang buka file *view* pada *browser* dan lihat hasil yang muncul.



Bisa dipahami?. Kalimat `Dunia Indonesia` muncul berasal dari variabel `dunia` yang ada pada controller kemudian dimunculkan melalui ekspresi `{{dunia}}` yang ada di dalam *directive*. Dari sini bisa diketahui bahwa semua variabel yang ada di scope controller juga bisa diakses dari *directive* yang berada di bawah controller tersebut.

Selanjutnya ketika dilihat melalui jendela *inspect element* terlihat bahwa kalimat `Hello Dunia Indonesia` yang menggunakan tag `h2` terletak di dalam tag *directive* yang kita buat.

```
<my-directive>
  <h2 class="ng-binding">Hello Dunia Indonesia</h2>
</my-directive>
```

Jika ingin posisi dari template yang kita gunakan tidak berada di dalam tag *directive* kita bisa mengurnya dengan menggunakan properti *replace* yang bernilai `true`. Ketika *replace* bernilai `true` maka *directive* akan digantikan / di-replace dengan apa yang ada pada properti *template*.

```
app.directive('myDirective', function(){
  return {
    restrict:'EA',
    template:'<h2>Hello {{dunia}}</h2>',
    replace:true
  };
});
```

Sampai sini saya minta pembaca untuk memahami dulu bagaimana tulisan `Hello Dunia Indonesia` bisa muncul. Amati baik-baik tiap langkah dan penjelasan yang saya berikan, dan sebisa mungkin untuk mencobanya sendiri dengan mengetikkan kode, bukan *copy – paste* hehe.

Meskipun *directive* yang kita buat masih sederhana namun ada hal-hal penting yang bisa kita pelajari. Kita sudah menggunakan tiga buah properti dari *directive*. Mari bahas satu per satu.

1. **restricted** – property ini untuk mengatur bagaimana *directive* kita akan digunakan. Ingat ada 4 jenis penggunaan *directive*. Ada 4 string untuk masing-masing yaitu `E, A, C, M`. Penggunaanya bisa digabung misal `EA`,
2. **template** – properti ini sebagai template yang akan ditampilkan ketika *directive* di-*render*. Jika ingin yang di-*render* adalah sebuah halaman HTML dari file lain maka yang digunakan adalah property `templateUrl` dengan nilai lokasi path file html yang bersangkutan.
3. **replace** – untuk mengatur apakah template yang ditampilkan me-replace tag *directive* atau tidak.

## 2.2 Scope

Pada sub bab sebelumnya saya sudah sempat menyinggung masalah *scope* yang ada didalam *directive* yang sebenarnya adalah *scope* yang berada di *parent*-nya. Lihat kembali kode terakhir dimana ekspresi mengambil variabel `dunia` yang ada di *controller*.

Bisa jadi ada kasus di mana kita membuat sebuah *directive* yang menggunakan *scope* yang bukan berasal dari *parent*-nya. Ada 2 cara untuk melakukannya yaitu :

1. Child Scope
2. Isolated Scope

Sebelum kita lari ke dua jenis *scope* tersebut akan lebih baik saya menjelaskan terlebih dahulu kenapa kita ingin menggunakan baik *child scope* maupun *isolated scope*.

*Scope* yang ada di *directive* itu secara *default* berasal dari *scope* yang dimiliki oleh *parent*-nya seperti yang sudah saya katakan sebelumnya. Pahami benar-benar kalimat saya yang itu. Karena *scope* tersebut merujuk ke *scope* yang sama maka ketika *scope* yang ada pada *directive* berubah, secara automatis *scope* yang ada di *parent* juga berubah, *make sense* kan?

Berikut ini contoh dari penjelasan di atas. Kita masih menggunakan *controller* yang sama dengan contoh sebelumnya.

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.dunia="Dunia Indonesia";  
}]);
```

Untuk *directive* dan *view* ada sedikit perubahan. Berikut *directive* yang digunakan, ada tambahan properti *link* di dalamnya (materi *link* akan saya jelaskan pada sub bab berikutnya).

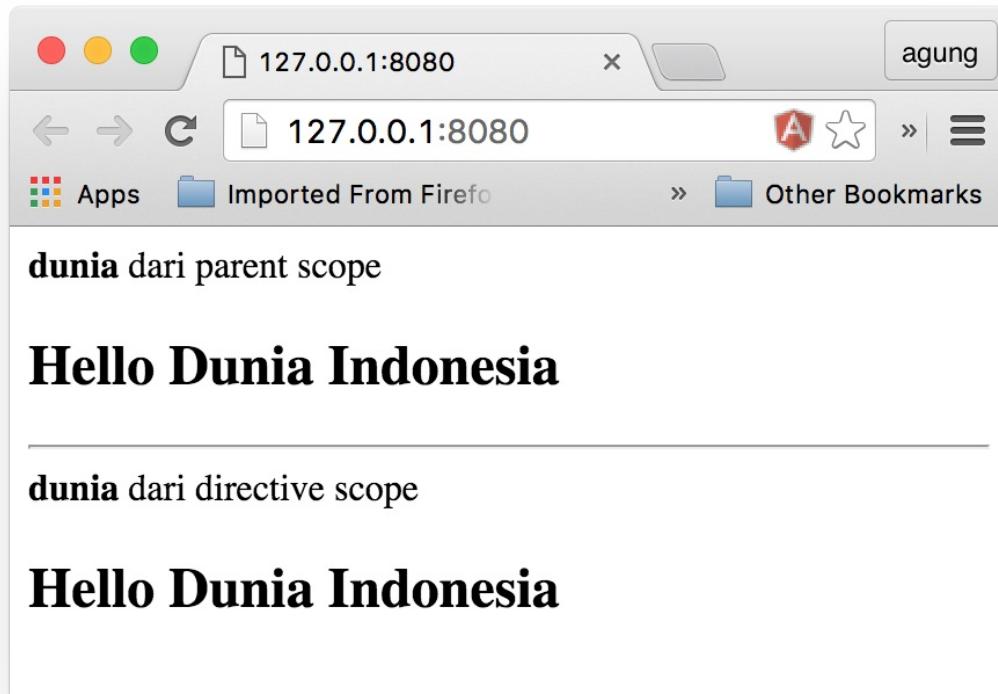
```
app.directive('myDirective',function(){  
    return {  
        restrict : 'EA',  
        template : '<h2>Hello {{dunia}}</h2>',  
        replace : true,  
        link : function(scope,elem,attrs){  
            elem.bind('click',function(){  
                scope.dunia='Saya diklik';  
                scope.$digest();  
            });  
        }  
    };  
});
```

Maksud dari `link` di atas adalah ketika *directive* diklik maka variabel dunia yang ada pada *scope* akan diganti nilainya menjadi `Saya diklik`.

*View* juga mengalami sedikit perubahan. Bagian atas digunakan untuk menampilkan variabel `dunia` yang ada di *controller (parent)* sedangkan yang bawah untuk variabel yang ada di *directive*.

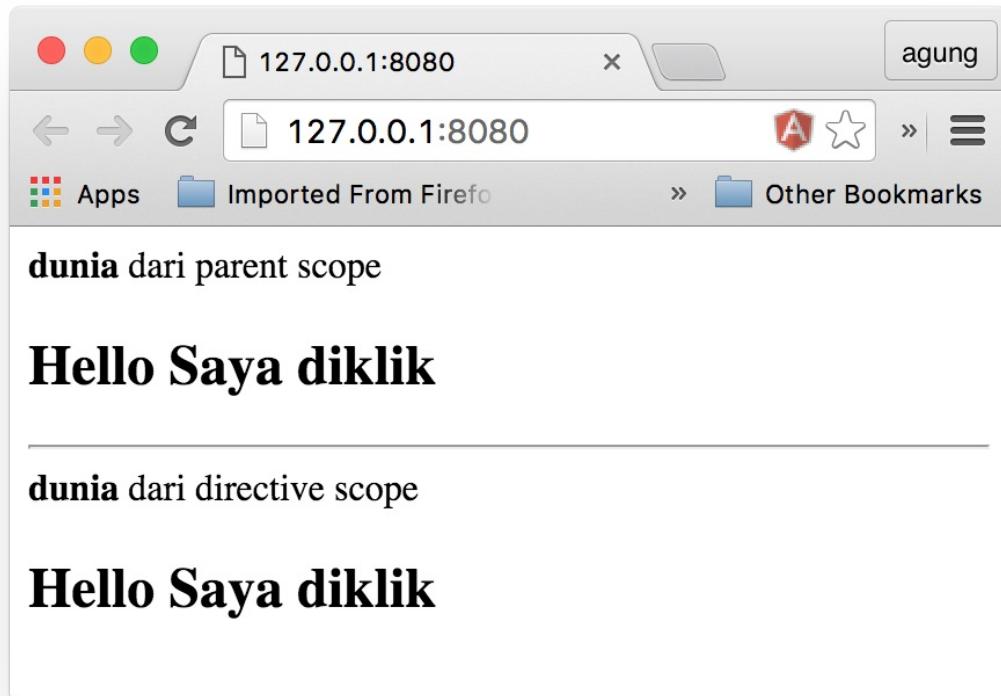
```
<div ng-controller="MainController">  
    <b>dunia</b> dari parent scope <br>  
    <h2>Hello {{dunia}} </h2>  
    <hr/>  
    <b>dunia</b> dari directive scope <br>  
  
    <my-directive></my-directive>  
</div>
```

Ketika dijalankan maka yang muncul adalah seperti pada gambar di bawah.



Sekarang kita coba klik tulisan `Hello Dunia Indonesia` yang berada pada bagian bawah yaitu pada bagian *directive* yang sudah kita tambahai fungsi untuk mengubah nilai variabel dunia ketika diklik. Maka yang terjadi adalah bukan hanya tulisan yang bawah yang berubah tetapi juga tulisan bagian atas. Hal ini sesuai penjelasan saya di atas, karena variabel yang diacu adalah variabel yang sama maka ketika berubah semuanya ikut berubah.

Hal – hal seperti inilah yang ingin dihindari karena bisa menyebabkan kekacauan jika variabel yang ada pada *directive* mengacu ke variabel yang sama pada *parent*-nya. Sebenarnya kita ingin mengubah yang ada pada *directive* saja, eh namun yang ada pada *parent* ikut berubah juga.



Gambar di atas menunjukkan perubahan yang terjadi ketika *directive* diklik. Tampak jelas bagian *parent* juga ikut berubah.

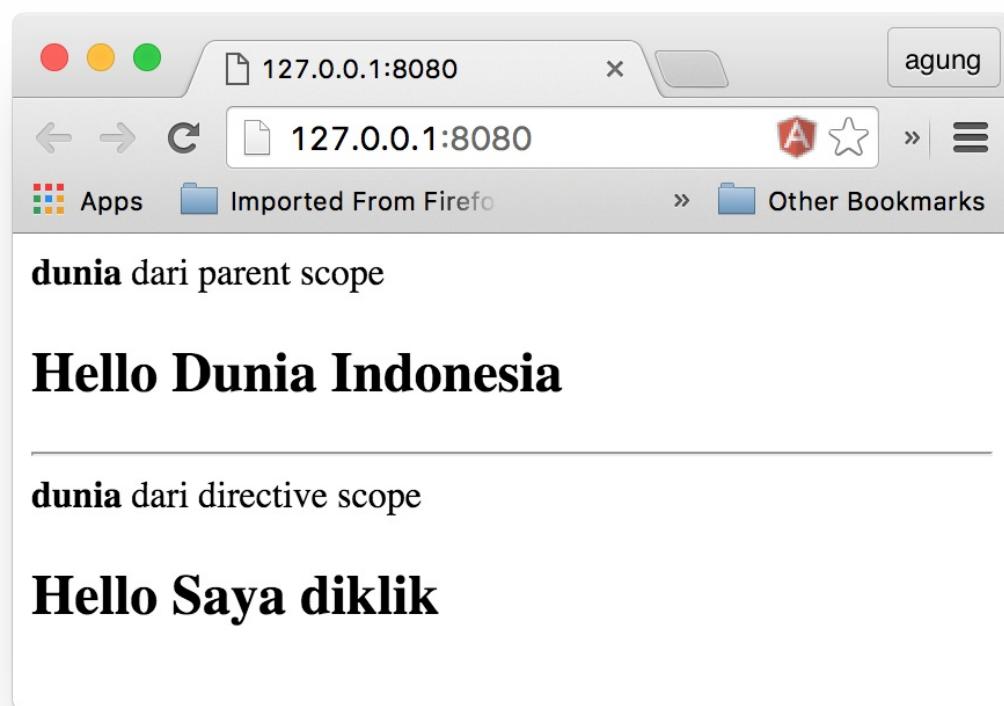
## 2.2.1 Child Scope

*Child scope* adalah *scope* yang merupakan *prototype* dari *parent scope*. Artinya bentuknya sama dengan punya *parent* cuma dia itu tiruannya, alias semacam nge-copy dari *parent*. Karena nge-copy berarti dia merupakan *scope* yang berbeda dengan yang dimiliki oleh *parent*. Hal ini menyebabkan perubahan *scope* di *directive* tidak akan berpengaruh di *parent*.

Untuk menggunakan *child scope* kita menggunakan properti *scope* dengan nilai `true` pada objek *directive*.

```
app.directive('myDirective', function(){
  return {
    scope : true,
    restrict : 'EA',
    template : '<h2>Hello {{dunia}}</h2>',
    replace : true,
    link : function(scope, elem, attrs){
      elem.bind('click', function(){
        scope.dunia='Saya diklik';
        scope.$digest();
      });
    }
  };
});
```

Sekarang coba jalankan kembali dan klik sekali lagi pada *directive*. Kali ini perubahan hanya terjadi pada bagian *directive* (bagian sebelah bawah).



### 2.2.2 Isolated Scope

Sesuai dengan namanya, *isolated scope* berarti *scope* yang sifatnya terisolasi, berlaku hanya di area *directive*, tidak merembet sampai *parent*. Kalau *child scope* tadi tidak mempengaruhi *parent* juga tetapi dia bentuknya merupakan kopian dari *parent*, *isolated scope* harus kita definisikan tersendiri untuk bisa digunakan.

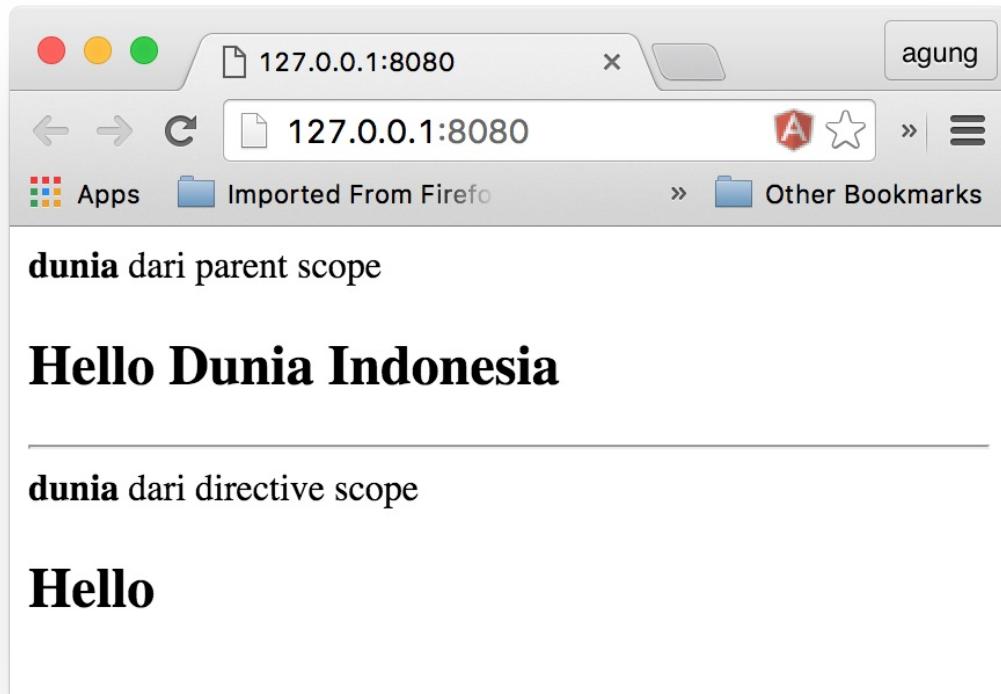
Mari kita ubah kode *directive* sebelumnya dengan mengubah *scope* yang semula *child scope* menjadi *isolated scope*. Untuk mendefinisikan *isolated scope* kita menggunakan properti *scope* yang berupa objek.

```
app.directive('myDirective', function(){
  return {
    scope : {

    },
    restrict : 'EA',
    template : '<h2>Hello {{dunia}}</h2>',
    replace : true,
    link : function(scope, elem, attrs){
      elem.bind('click', function(){
        scope.dunia='Saya diklik';
        scope.$digest();
      });
    }
  };
});
```

Di atas kita membuat *isolated scope* yang bernilai kosong, nanti akan kita isi dan bagaimana contoh penggunaanya.

Sekarang coba jalankan lagi contoh aplikasi kita. Kali ini bagian *directive* tidak bisa membaca variabel `dunia` yang ada pada *scope*. Kejadian ini disebabkan karena kita menggunakan *isolated scope*, jadi *directive* tidak mengenali *scope* selain yang ada di *isolated scope*.



Sekarang pertanyaannya bagaimana *directive* bisa menggunakan *isolated scope*? Mungkin pembaca akan mengira-ngira dengan seperti ini definisikan variabel di dalam objek scope dan kasih nilainya , seperti berikut.

```
app.directive('myDirective', function(){
  return {
    scope : {
      dunia : 'Halo Saya Isolated Scope'
    },
    ...
    ...
    ...
  };
});
```

Namun setelah kode diatas ditulis dan dieksekusi maka tidak berjalan seperti yang kita inginkan haha. Ya karena memang bukan seperti itu cara menggunakan *isolated scope*. Pun misal cara di atas bisa maka data variabel akan menjadi *hard code*.

Sub bab selanjutnya saya tulis untuk membahas masalah cara melakukan *binding* antara *isolated scope* dengan *parent scope* sehingga kita bisa menaruh sebuah nilai pada *isolated scope*.

## 2.2.3 Binding Antara Parent Scope dengan Isolated Scope

Seperti sudah saya singgung pada sub bab berikutnya, kita tidak bisa mendefinisikan *isolated scope* dengan serta-merta langsung diisi nilanya. Akan tetapi, kita bisa mengisi nilai yang ada pada variabel di *isolated scope* dengan nilai yang berasal dari variabel di luarnya (*parent scope*).

Ada 2 cara untuk memberi nilai pada isolated scope yaitu menggunakan :

1. `@` untuk *one way text binding*
2. `=` untuk *two way binding*

Mari kita bahas satu demi satu 2 cara di atas.

### **@ untuk one way text binding**

Diberi nama seperti itu karena dengan cara ini nilai yang dikirim ke *isolated scope* berasal dari *parent scope* tetapi tidak bisa berlaku kebalikan (nilai di-*isolated scope* tidak akan mempengaruhi *parent scope*).

Perhatikan contoh *directive* di bawah ini

```
app.directive('oneWay', function(){
  return {
    restrict:'E',
    scope:{
      title:'@'
    },
    template:'<h2>Title Inside : {{title}}</h2>',
    link : function(scope,elem,attrs){
      elem.bind('click',function(){
        scope.$apply(function() {
          scope.title = "JavaScript";
        });
      });
    }
  };
});
```

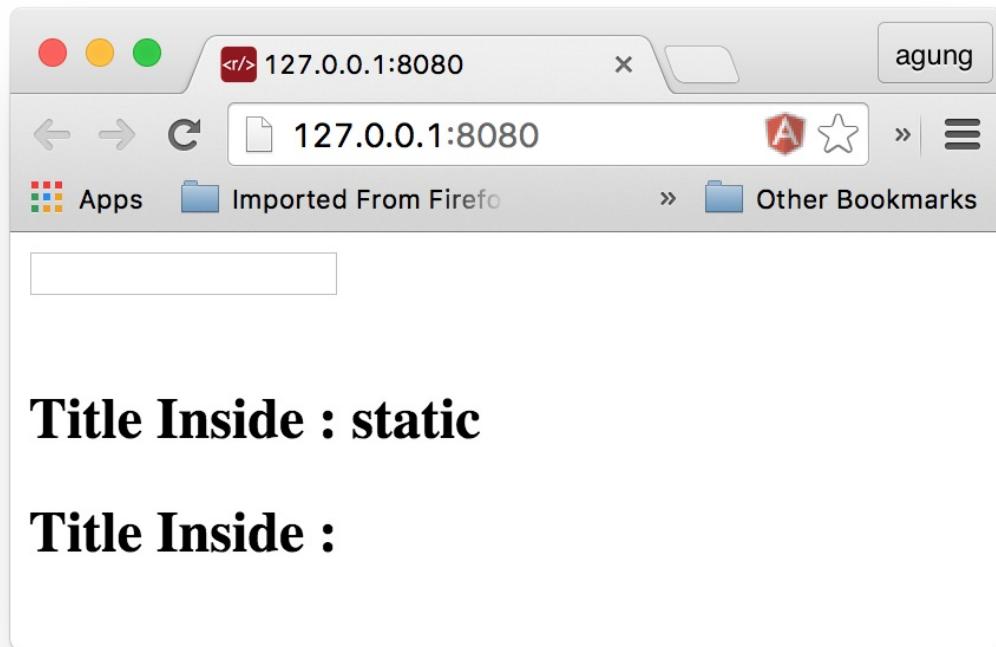
Nantinya kode di atas akan menampilkan pesan berupa `Title Inside : {{inside}}` dengan nilai variabel `inside` berasal dari luar / *parent scope*.

Kode html dan AngularJs di bawah ini menunjukkan bagaimana cara mengisi nilai variabel `title` yang ada pada *isolated scope* yang berasal dari *parent scope*.

```
<div ng-controller="MainController">
  <input type="text" ng-model="title"/> <br> <br>

  <one-way title="static"></one-way>
  <one-way title="{{title}}></one-way>
</div>
```

Tampilan awal ketika dijalankan sudah bisa pembaca tebak?, coba cocokkan perkiraan pembaca dengan *screen shot* berikut ini.

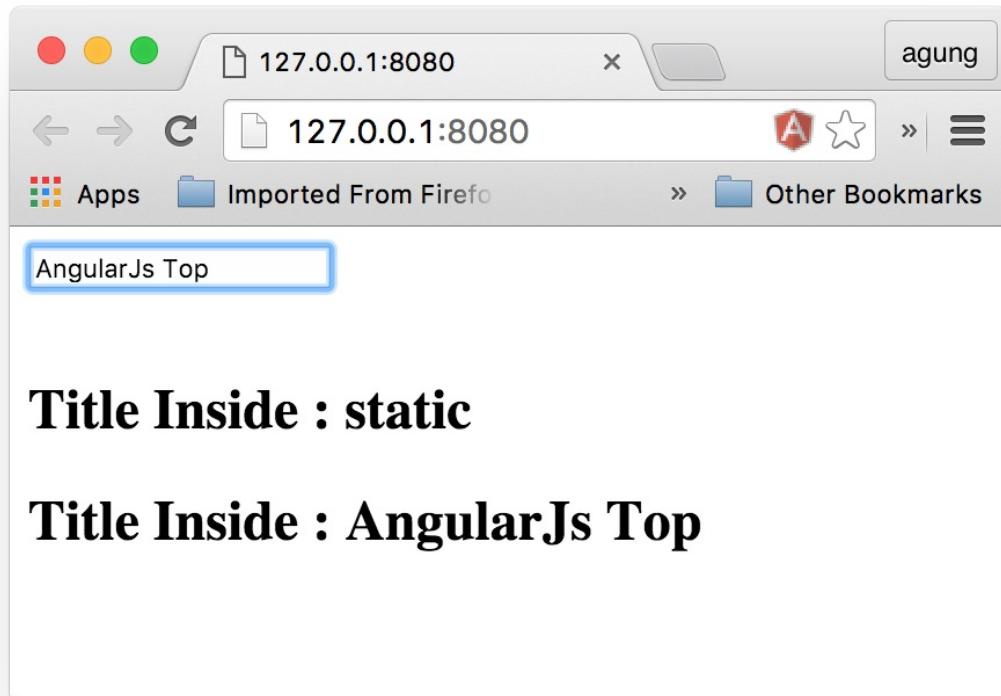


*Directive* pertama akan menghasilkan teks `Title Inside : static` karena pada *directive* ini kita mem-*passing* nilai ke variabel *inside* dengan nilai `static`.

```
<one-way title="static"></one-way>
```

Sedangkan *directive* kedua akan kosong karena nilai yang kita *passing* adalah berupa teks yang didapat dari variabel `title` yang pada saat awal tidak bernilai.

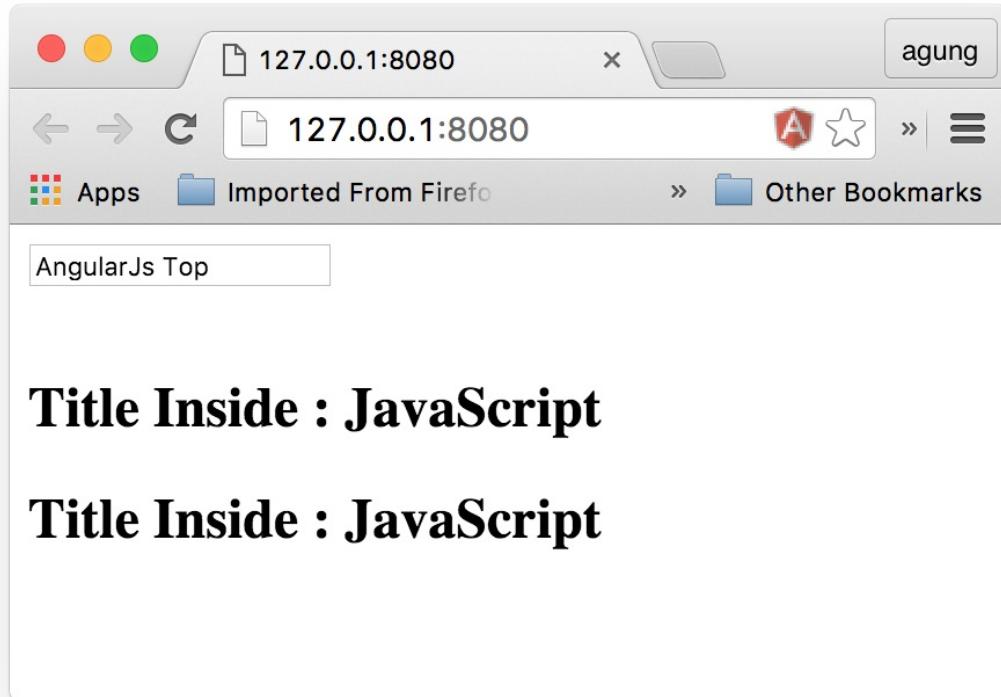
Sekarang coba isi input teks yang ada dengan kata apapun, maka *directive* akan ikut berubah.



Ngomong-ngomong, jika menggunakan `@` untuk mem-passing variabel maka yang bisa di-passing hanyalah string, itu mengapa kita menggunakan ekspresi `{{title}}` saat mem-passing variabel title.

```
<one-way title="{{title}}"></one-way>
```

Terus masalah *one way*-nya sudah saya jelaskan pada bagian *parent scope* dan *isolated scoped* ya. Untuk mencobanya silahkan klik pada kedua *directive* maka `title` akan berubah menjadi `JavaScript` tetapi variabel pada `title` yang ada pada inputan teks tidak berubah.



## = untuk two way binding

Perbedaan dengan menggunakan @ adalah

1. Yang di-passing berupa object bukan string
2. Perubahan scope di parent berpengaruh ke child dan berlaku sebaliknya

Dengan menggunakan contoh koding yang masih sama kita hanya perlu mengganti tanda dari @ menjadi =

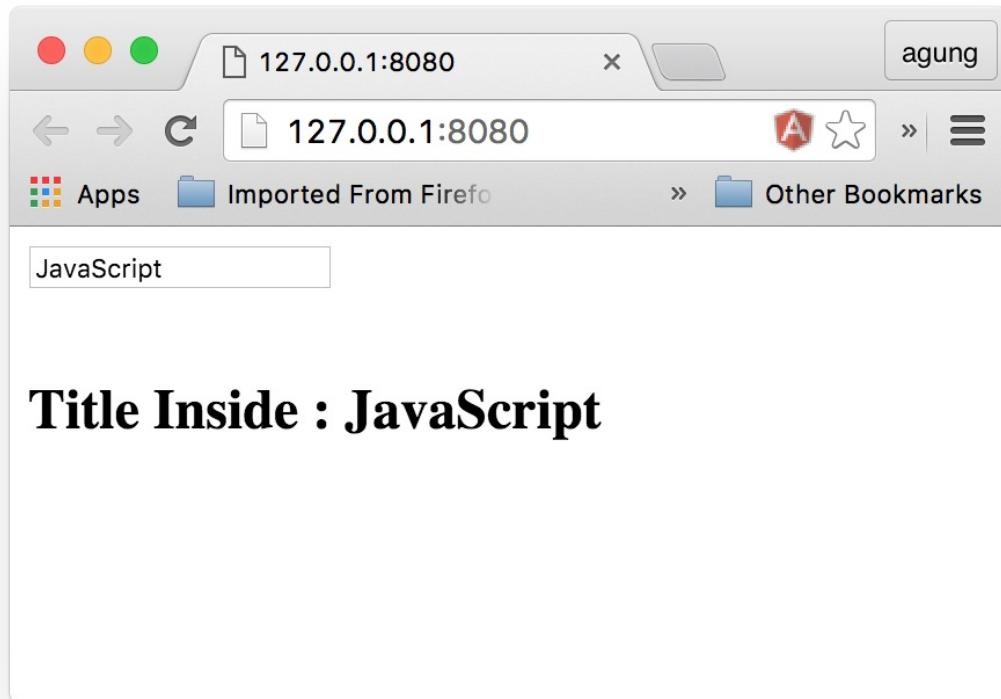
```
app.directive('oneWay', function(){
  return {
    restrict:'E',
    scope:{  
      title:'='  
    },
    template:'<h2>Title Inside : {{title}}</h2>',
    link : function(scope,elem,attrs){
      elem.bind('click',function(){
        scope.$apply(function() {
          scope.title = "JavaScript";
        });
      });
    }
  };
});
```

Selain itu kita tidak lagi menggunakan ekspresi (  `{{var}}`  ) saat mem-*passing* sebuah nilai. Perhatikan contoh di bawah ini pada HTML-nya.

```
<div ng-controller="MainController">
  <input type="text" ng-model="title"/> <br> <br>

  <one-way title="title"></one-way>
</div>
```

Ketika diklik *parent* akan ikut berubah



## 2.3 link function

Pembaca pasti sudah berkali-kali melihat dari beberapa contoh di atas terdapat properti `link` pada `directive` yang berupa fungsi. Saya juga yakin pembaca sudah mengerti apa kegunaannya.

`link` digunakan untuk melakukan utilisasi terhadap `scope`. Bahasa gampangnya, kalau mau ngapa-ngapain `scope` misal mengubah nilainya ya gunakan `link`.

`link` memiliki 3 buah parameter yaitu :

1. **scope** – `scope` yang ada pada `directive`
2. **elem** – elemen dari `directive`, digunakan untuk manipulasi DOM
3. **attrs** – `attribute` dari `directive`

Jika mau mengubah nilai variabel pada `scope` maka gunakan parameter `scope` untuk mengubahnya, misal ada variabel dengan nama `title` yang ingin diubah maka kode berikut diperlukan.

```
scope.title = "Nilai baru";
```

Nilai `scope` tidak akan berubah kalau tidak ada apa-apa. Masa iya tiba-tiba nilai bisa berubah? Pasti tidak kan?. Paling sering, suatu nilai berubah pada saat ada `event klik`. Untuk mengakomodasi hal seperti itu digunakanlah parameter `elem`.

Misal kita ingin memberi nilai baru pada `title` ketika ada klik pada `directive`, maka gunakan `elem` supaya bereaksi ketika diklik. `elem` adalah `jQLite` (subset dari `jQuery`) jadi cara penggunaanya sama saja dengan `jQuery` untuk melakukan manipulasi DOM.

```
link : function(scope, elem, attrs){  
    elem.bind('click', function(){  
        scope.$apply(function() {  
            scope.title = "Nilai baru";  
        });  
    });  
}
```

`Click` adalah hanya sebagian dari `event` yang bisa digunakan di sini, untuk `event` apa saja yang ada silahkan merujuk ke dokumentasi resmi dari `jQuery` atau `googling`.

Diperhatikan secara seksama maka pembaca akan menemukan kode yang sedikit aneh yaitu adanya `scope.$apply`, apa kegunaannya? Sabar dulu ya nanti pada bab mendatang akan saya bahas juga. Pokoknya untuk sekarang cukup tahu kegunaan dari `link` dan cara menggunakan parameter-parameter yang dimilikinya :)

## 2.4 Demo Directive

Setelah melewati perjalanan yang cukup menyita energi untuk belajar `directive` melalui subbab – subbab sebelumnya, yok sekarang mari kita coba untuk mengimplementasikan apa yang telah kita pelajari dengan membuat sebuah `directive` yang pantas digunakan di *real world project*.

Tujuan utama dari `directive` yang akan kita buat adalah untuk menampilkan data tertentu dalam bentuk tabel yang bisa digunakan berulang kali. Di ASP.NET hal seperti ini disebut *partial view*. Di Ruby on Rails disebut *partial render*.

Langkah pertama seperti biasa buat module dan pasang di HTML-nya dan kemudian buat controller dan juga pasang di HTML. Jangan lupa untuk menyertakan file `.js`.

```
var app=angular.module('MyApp', []);
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Demo Latihan Directive</title>
</head>
<body ng-app="MyApp">

    <div ng-controller="MainController">
        {{judul}}
        <hr/>

    </div>
    <script src="js/angular.min.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers/MainController.js"></script>
    <script src="js/directives/AgsTable.js"></script>
</body>
</html>
```

Berikut adalah kode awal yang ada di controller.

```
app.controller('MainController', ['$scope', function($scope){
    $scope.judul='Demo Penggunaan AngularJs Directive';
}]);
```

Jalankan pada *browser* dan pastikan tulisan `Demo Penggunaan AngularJs Directive` muncul.

Langkah selanjutnya adalah membuat *directive* yang menggunakan template berupa file HTML.

**PERHATIAN**, untuk bisa menggunakan template yang berupa halaman HTML, kita harus menjalankan aplikasi AngularJs melalui *web server*, semisal Apache.

Kode di bawah ini merupakan *directive* kita.

```
app.directive('agsTable',function(){
    return {
        restrict:'E',
        scope:{
            books:'='
        },
        templateUrl:'js/directives/AgsTable.html'
    };
});
```

Adapun file HTML yang kita jadikan sebagai template yang bisa digunakan berulang kali bisa dilihat di bawah ini.

```
<head>
  <style type="text/css">
    .table{
      border-collapse: collapse;
    }

    tr,td,th{
      border: 1px solid #000;
      padding: 5px;
    }

    td{
      width: 300px;
    }

    .number{
      text-align: right;
    }
  </style>
</head>

<table class="table">
  <tr>
    <th>Judul</th>
    <th>Penulis</th>
    <th>Rating</th>
  </tr>

  <tr ng-repeat="b in books">
    <td>{{b.judul}}</td>
    <td>{{b.penulis}}</td>
    <td class="number">{{b.rating}}</td>
  </tr>
</table>
```

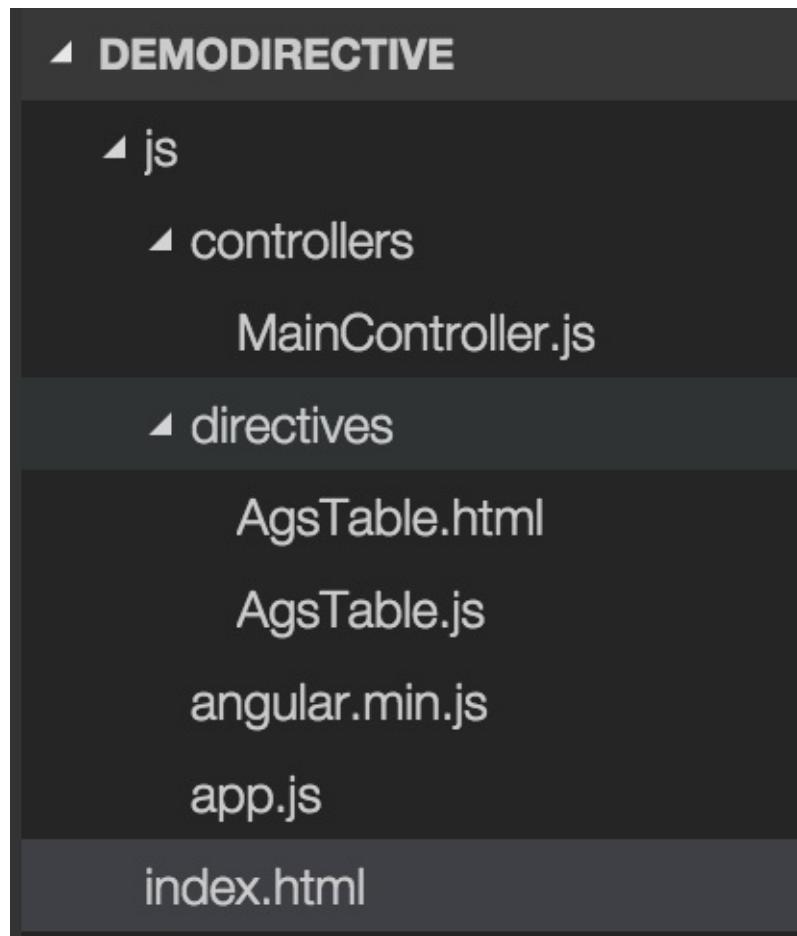
Kembali lagi ke *controller*. Tambahkan sebuah variabel bernama `books` yang menampung data beberapa buku, usahakan minimal 3 ya.

```
app.controller('MainController', ['$scope', function($scope){  
    $scope.judul='Demo Penggunaan AngularJs Directive';  
  
    $scope.books=[  
        {  
            'judul':'Ayah',  
            'penulis':'Andrea Hirata',  
            'rating':4  
        },  
        {  
            'judul':'Eragon',  
            'penulis':'Christoper Paolini',  
            'rating':3  
        },  
        {  
            'judul':'Blink',  
            'penulis':'Malcolm Gladwell',  
            'rating':3  
        },  
    ];  
});
```

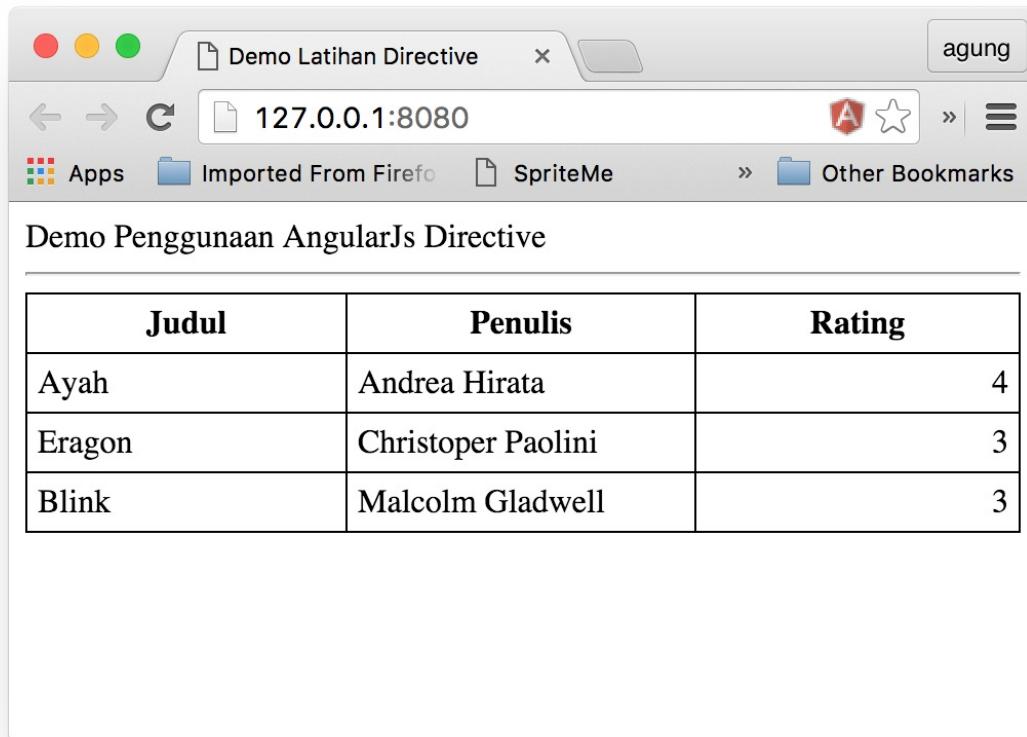
Sip kalau bisa mengikuti sampai sini. Langkah terakhir adalah memasukkan data `books` yang ada di `controller` ke dalam `directive`. Pasti bisa dong?, kalau belum bisa coba pahami lagi pada subbab sebelumnya. Kalau sudah yakin bisa coba cocokkan dengan kode berikut ini.

```
<div ng-controller="MainController">  
    {{judul}}  
    <hr/>  
    <ags-table books="books"></ags-table>  
</div>
```

Jika pembaca bingung mengikuti bagaimana struktur foldernya, di bawah ini saya sertakan struktur folder dari pekerjaan di atas.



Selesai sudah utak-atik kita untuk membuat *directive*. Buka file **index.html** di *browser* maka seharusnya yang pembaca lihat adalah seperti pada gambar di bawah.



Manfaat dari penggunaan *directive* ini adalah jika pada bagian lain dari aplikasi kita ingin menampilkan data buku yang memiliki format yang sama dengan isi yang berbeda maka kita tinggal mengganti `scope books` dengan data yang lain. Misalnya gini, di halaman depan dimunculkan buku-buku yang baru terbit, sedangkan di halaman lain dimunculkan buku-buku yang menduduki rangking 1 – 10.

The screenshot shows a web browser window titled "Demo Latihan Directive". The address bar displays "127.0.0.1:8080". The page content is titled "Demo Penggunaan AngularJs Directive" and contains two tables.

**Table 1:**

Judul	Penulis	Rating
Ayah	Andrea Hirata	4
Eragon	Christoper Paolini	3
Blink	Malcolm Gladwell	3

**Table 2:**

Judul	Penulis	Rating
Robohnya Surau Kami	AA Navis	4
Arok Dedes	Pramoedya Ananta Toer	4
Harimau-Harimau	Mochtar Lubis	4

Sampai sini maka berakhir sudah pelajaran kita mengenai *directive* pada AngularJs. Saya yakin pasti masih banyak hal yang harus dipelajari lebih lanjut di *directive*. Mari sama-sama terus belajar :)

# Bab 3 Service

Materi yang dipelajari pada bab ini :

- Value
- Factory
- Service
- Provider
- Constant

Ketika aplikasi yang kita kembangkan merupakan aplikasi yang besar, maka akan lebih baik jika kita memecah-mecahnya menjadi bagian yang terpisah yang memiliki fungsi masing-masing. Tujuan pemecahan ini supaya kode menjadi rapi, tidak tumpuk blek menjadi satu di *controller* yang sangat membingungkan. Untuk keperluan ini AngularJs menyediakan 2 buah senjata : *dependency injection* dan *service*.

## 3.1 Value

*Value* di AngularJs digunakan sebagai semacam *dictionary* yang berisi *key/kunci* dan *value/nilai*. Nilai yang digunakan bisa berupa tipe data apa saja, string, angka maupun object. Value ini nantinya di-*inject* baik ke *controller*, *factory*, maupun *service* tergantung kebutuhan.

Sebuah *Value* merupakan bagian dari module, jadi untuk membuatnya kita menggunakan method *value* pada module.

```
var app=angular.module('MyApp',[]);
app.value('string','Belajar AngularJs Service');
app.value('number',100);
app.value('object',{'title':'AngularJs Service','desc':'Belajar'})
```

Pada kode di atas, parameter pertama adalah nama dari *value* dan parameter kedua adalah isi nya. Setelah mendefiniskan *value*, kita bisa meng-*inject*-nya di *controller*.

```
app.controller('MainController',['$scope','string',function($scope, string){
  console.log(string);
}]);
```

Cek hasilnya pada *console* di *browser*.

Variabel string yang di-*inject* pada kode di atas adalah berasal dari value yang juga bernama `string` yang berisi `Belajar AngularJs Service`. Nama keduanya harus persis sama ketika digunakan.

## 3.2 Factory

Jika *Value* terlihat `kok gitu doang ya?` maka pembaca pasti akan merasa senang jika sudah berkenalan dengan *factory*.

Berbeda dengan *value* yang nilainya di-*hard code* sehingga tidak bersifat dinamis maka *factory* bisa memberikan nilai yang dinamis. **Yang perlu diingat dari *factory* adalah**, saat di-*inject* nanti, yang di-*inject* bukanlah *factory* itu sendiri tetapi nilai yang dikembalikan olehnya.

*Factory* juga dibuat di module, kali ini menggunakan method yang bernama *factory*.

```
var app=angular.module('MyApp', []);

app.factory('myFactory', function(){
  return 'Berasal dari dalam Factory';
});
```

Cara meng-*inject*-nya sama persis dengan *value*, perhatikan kode di bawah ini sebagai referensi.

```
app.controller('MainController',['$scope','myFactory',function($scope, myFactory){
  console.log(myFactory);
}]);
```

Sama dengan *Value*, nama yang di-*inject* juga sama dengan nama *factory* yang kita definisikan saat pembuatannya. Cek hasilnya pada *console browser*.

Kurang aplikatif ya contohnya?. Oke yang agak mendingan saya berikan contoh *factory* untuk aplikasi kalkulator.

Kalkulator memiliki 4 operasi dasar : tambah, kurang, kali, bagi. Tentu sudah terpikirkan bahwa nilai inputan harus bersifat dinamis yang artinya berarti kita memerlukan parameter untuk menangani hal yang bersifat dinamis tersebut. Pertanyaanya, bisakah *factory* menerima parameter?.

Jawaban dari pertanyaan di atas adalah bisa, dengan memahami bahwa yang di-*inject* adalah nilai kembalian dari *factory*, bukan *factory* itu sendiri maka kita bisa membuat sebuah *factory* seperti di bawah ini.

```
var app=angular.module('MyApp', []);  
  
app.factory('KalkulatorFactory', function(){  
    var kalkulator={};  
  
    kalkulator.tambah=function(angkaA, angkaB){  
        return parseInt(angkaA)+parseInt(angkaB);  
    };  
  
    kalkulator.kurang=function(angkaA, angkaB){  
        return angkaA-angkaB;  
    };  
  
    kalkulator.kali=function(angkaA, angkaB){  
        return angkaA*angkaB;  
    };  
  
    kalkulator.bagi=function(angkaA, angkaB){  
        return angkaA/angkaB;  
    };  
  
    return kalkulator;  
});
```

Pada bagian paling awal dari *factory* terdapat kode `var kalkulator={};`. Kode ini digunakan untuk membuat sebuah objek yang bernama `kalkulator`. Kode berikutnya berturut-turut adalah ke-4 operasi dasar kalkulator yang berupa fungsi yang dimiliki oleh objek kalkulator tadi. Pada bagian akhir objek kalkulator dijadikan nilai kembalian oleh *factory*. Dengan objek kalkulator dijadikan sebagai nilai kembalian maka nanti kita bisa menggunakananya untuk melakukan ke-4 operasi dasar. Itulah mengapa saya tekankan dari awal untuk memahami bahwa pada *factory* yang *di-inject* adalah nilai kembalinya .

Halaman *view* kita buat sebagai jalan bagi pengguna untuk memasukkan angka dan memilih operasi matematika apa yang akan digunakan.

```
<div ng-controller="MainController">
  {{title}}
  <hr/>

  Angka 1 :<input type="text" ng-model="angkaA"> <br>
  Angka 2 :<input type="text" ng-model="angkaB"> <br>
  Hasil : {{hasil}}
  </br>
  </br>
  <button ng-click="penambahan()">+</button>
  <button ng-click="pengurangan()">-</button>
  <button ng-click="perkalian()">*</button>
  <button ng-click="pembagian()">/</button>
</div>
```

Dan terakhir, bagian yang penting adalah *di-controller*, bagaimana cara kita menggunakan *factory* untuk melakukan operasi matematika.

```
app.controller('MainController', ['$scope', 'KalkulatorFactory', function($scope, KalkulatorF
  $scope.title='Kalkulator Factory';

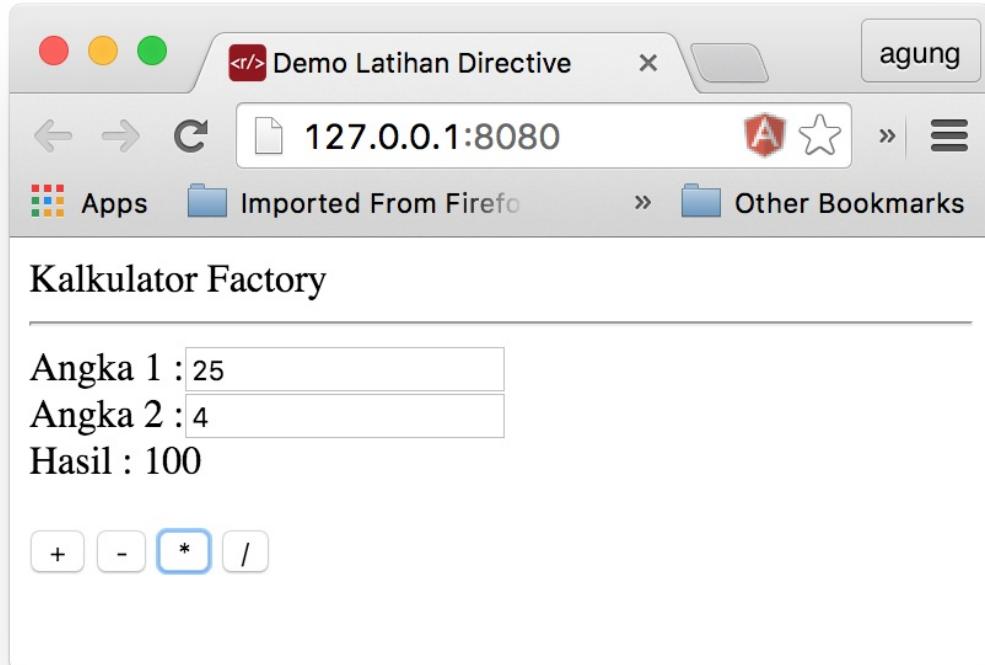
  $scope.penambahan=function(){
    $scope.hasil=KalkulatorFactory.tambah($scope.angkaA,$scope.angkaB);
  }

  $scope.pengurangan=function(){
    $scope.hasil=KalkulatorFactory.kurang($scope.angkaA,$scope.angkaB);
  }

  $scope.perkalian=function(){
    $scope.hasil=KalkulatorFactory.kali($scope.angkaA,$scope.angkaB);
  }
  $scope.pembagian=function(){
    $scope.hasil=KalkulatorFactory.bagi($scope.angkaA,$scope.angkaB);
  }
}]);
```

Seperti sudah selazimnya, kita meng-*inject factory* ke dalam *controller* dengan cara mem-*passing* nama *factory* di dalam parameter fungsi. Selanjutnya, karena nilai kembalian dari *factory* adalah objek kalkulator yang memiliki 4 buah method, maka kita bisa memanggil 4 buah method tersebut di dalam *controller*.

Coba jalankan di *browser*, masukkan angka pada *text box* dan tekan salah satu tombol operasi. Pastikan kode pembaca sudah berjalan dengan benar.



### 3.3 Service

Pendekar ketiga kita adalah *service*. Kalau saya bilang, *service* ini mirip dengan *factory*. Perbedaan antara keduanya adalah, jika pada *factory* yang di-*inject* adalah nilai kembalian dari *factory* maka pada *service* yang di-*inject* adalah *service* itu sendiri.

*Service* bekerja pada module dengan menggunakan method *service*.

```
var app=angular.module('MyApp', []);  
  
app.service('myService', function(){  
    this.methodA=function(){  
        ...  
    }  
  
    this.methodB=function(){  
        ...  
    }  
});
```

Dengan masih menggunakan aplikasi kalkulator sebagai contoh, maka kita ubah operasi matematika yang tadinya menggunakan factory menjadi menggunakan service.

```
var app=angular.module('MyApp', []);

app.service('KalkulatorService', function(){

    this.tambah=function(angkaA, angkaB){
        return parseInt(angkaA)+parseInt(angkaB);
    };

    this.kurang=function(angkaA, angkaB){
        return angkaA-angkaB;
    };

    this.kali=function(angkaA, angkaB){
        return angkaA*angkaB;
    };

    this.bagi=function(angkaA, angkaB){
        return angkaA/angkaB;
    };
});
```

Pada bagian *view* sama sekali tidak ada perubahan. Pada *controller* tinggal ubah yang tadinya nama *factory* kita menjadi nama *service* kita, sebagai berikut kodennya.

```
app.controller('MainController',['$scope', 'KalkulatorService', function($scope,Kalkulators

    $scope.title='Kalkulator Service';
    $scope.penambahan=function(){
        $scope.hasil=KalkulatorService.tambah($scope.angkaA,$scope.angkaB);
    }

    $scope.pengurangan=function(){
        $scope.hasil=KalkulatorService.kurang($scope.angkaA,$scope.angkaB);
    }

    $scope.perkalian=function(){
        $scope.hasil=KalkulatorService.kali($scope.angkaA,$scope.angkaB);
    }

    $scope.pembagian=function(){
        $scope.hasil=KalkulatorService.bagi($scope.angkaA,$scope.angkaB);
    }
}]);
```

Pada kode di atas kita yang pertama kita lakukan adalah meng-*inject* **KalkulatorService**. Karena service kita ini memang sudah berupa objek dari sananya maka kita bisa menggunakanya untuk memanggil method-method yang dimiliki.

Pada bagian belakang, yang dilakukan oleh AngularJs adalah melakukan instantiasi dari **KalkulatorService**.

```
var theService = new KalkulatorService();
```

Jalankan pada *browser* dan pastikan kalkulator berjalan dengan benar, tidak ada *error* yang terjadi.

## 3.4 Contoh Service Http

Bagian ini memuat contoh penggunaan *service* yang memanfaatkan komponen `$http` pada AngularJs untuk mengambil data json yang berasal dari Web API. Nilai balikan json tersebut akan kita tampilkan melalui AngularJs.

Kalau rasanya dari tadi contoh *service* yang saya berikan terkesan kurang aplikatif maka contoh kali ini saya jamin pasti digunakan di dunia nyata. Penggunaan `$http` untuk berkomunikasi dengan Web API digunakan jika kita membangun aplikasi yang berhubungan dengan data.

Langkah pertama pasti sudah hafal ya, membuat modul dan *controller* kemudian pasang pada file *view*. Jangan lupa untuk memasukkan file `.js` pada halaman Html.

```
var app=angular.module('MyApp', []);

app.controller('MainController',['$scope',function($scope){
  $scope.title="AngularJs Http Service";
}]);
```

```
<body ng-app="MyApp">
  <div ng-controller="MainController">
    {{title}}
    <hr/>

  </div>
</body>
```

Selanjutnya kita buat *service* yang menggunakan `$http`. Ketikkan kode di bawah ini.

```
app.service('UsersService', ['$http',function($http){  
  
    this.index=function(){  
        return $http.get('http://jsonplaceholder.typicode.com/users');  
    }  
  
}]);
```

Pada *service* di atas, kita melakukan injeksi komponen `$http` sehingga bisa digunakan di dalam *service*. Salah satu method yang dimiliki oleh `$http` adalah `get`, digunakan untuk mengambil data yang terletak pada *end point* tertentu, dalam contoh di atas adalah

`http://jsonplaceholder.typicode.com/users`.

Kembali ke bagian *controller*. Ubah *controller* supaya *service* di atas bisa digunakan dan panggil method `index` untuk mengambil data yang berada pada *resource server*.

```
app.controller('MainController',['$scope','UsersService', function($scope,UsersService){  
  
    $scope.title="AngularJs Http Service";  
  
    UsersService.index().success(function(data){  
        $scope.users=data;  
    });  
  
}]);
```

Apa yang terjadi di atas adalah kita melakukan injeksi *service* ke dalam *controller*. Selanjutnya *service* memanggil method yang dimiliki yaitu `index` untuk mengambil data dari Web API. Ketika berhasil / `success`, data yang diperoleh dimasukkan ke dalam variabel yang bernama `users`.

Kalau sudah sampai sini sih saya rasa pembaca sudah tahu bagaimana cara menampilkan data dari Web API tadi ke `view`-nya.

```

<div ng-controller="MainController">
  {{title}}
  <hr/>

  <table>
    <tr>
      <th>Name</th>
      <th>Username</th>
      <th>Email</th>
      <th>Website</th>
      <th>Company</th>
    </tr>

    <tr ng-repeat="u in users">
      <td>{{u.name}}</td>
      <td>{{u.username}}</td>
      <td>{{u.email}}</td>
      <td>{{u.website}}</td>
      <td>{{u.company.name}}</td>
    </tr>
  </table>

</div>

```

Name	Username	Email	Website	Company
Leanne Graham	Bret	Sincere@april.biz	hildegard.org	Romaguera-Crona
Ervin Howell	Antonette	Shanna@melissa.tv	anastasia.net	Deckow-Crist
Clementine Bauch	Samantha	Nathan@yesenia.net	ramiro.info	Romaguera-Jacobson
Patricia Lebsack	Karianne	Julianne.OConner@kory.org	kale.biz	Robel-Corkery
Chelsey Dietrich	Kamren	Lucio_Hettinger@annie.ca	demarco.info	Keebler LLC
Mrs. Dennis Schulist	Leopoldo_Corkery	Karley_Dach@jasper.info	ola.org	Considine-Lockman
Kurtis Weissnat	Elwyn.Skiles	Telly.Hoeger@billy.biz	elvis.io	Johns Group
Nicholas Runolfsdottir V	Maxime_Nienow	Sherwood@rosamond.me	jacynthe.com	Abernathy Group
Glenna Reichert	Delphine	Chaim_McDermott@dana.io	conrad.com	Yost and Sons
Clementina DuBuque	Moriah.Stanton	Rey.Padberg@karina.biz	ambrose.net	Hoeger LLC

Untuk Http method yang lain seperti `POST`, `DELETE` dan `PUT` penggunaannya hampir sama, nanti akan saya perlihatkan contohnya pada bab yang membahas studi kasus.

# Bab 4 Route

Materi yang dipelajari pada bab ini :

- Route Provider
- Route Param

Suatu aplikasi yang berguna sangat jarang atau bahkan tidak ada yang hanya memiliki satu halaman tanpa adanya halaman yang lain. Khusus untuk aplikasi web, halaman – halaman yang satu berkolaborasi dengan halaman melalui perantara sebuah link.

Berbeda dengan aplikasi web pada umumnya dimana ketika sebuah link diklik dan berpindah ke halaman lain yang terjadi adalah adanya proses *reload*, maka jika menggunakan AngularJs, transisi akan terasa *smooth* tanpa adanya proses *reload*.

## 4.1 Route Provider

*Route provider* adalah komponen pada AngularJs yang digunakan untuk mengkonfigurasi *route* yang ada pada aplikasi kita. Apa saja yang dikonfigurasi? Sebelum saya menjawabnya saya perlu memperlihatkan kepada pembaca contoh url dari aplikasi AngularJs yang memiliki *route*.

```
http://sample.com/index.html#books  
http://sample.com/index.html#authors  
http://sample.com/index.html#books/123
```

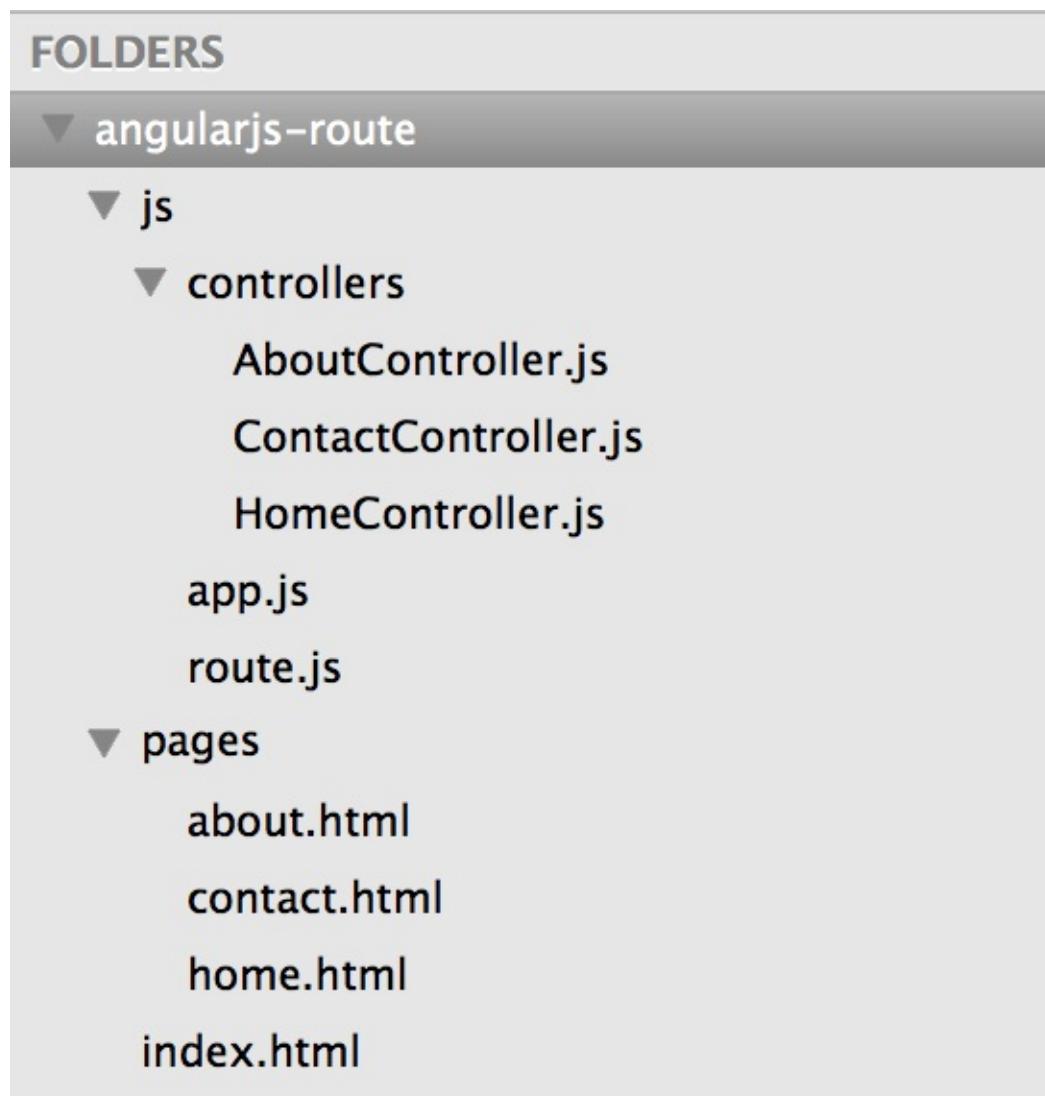
Url – url di atas mengakses aplikasi yang sama tetapi memiliki route yang berbeda. Route ditandai dengan adanya tanda `#`. Di belakang tanda `#` tersebut adalah *route*-nya.

Balik ke pertanyaan tadi, apa saja yang dikonfigurasi menggunakan *route provider*? Ada 2 hal yang dikonfigurasi, yaitu :

1. Template yang akan ditampilkan
2. *Controller* yang bekerja pada *template* yang ditampilkan

Langsung ke contoh saja ya biar paham maksudnya bagaimana dan cara menggunakannya seperti apa. Kasusnya misal kita ingin membuat 3 buah halaman yaitu `home` , `about` dan `contact` . Masing – masing halaman memiliki konten yang berbeda satu sama lain. Mari langsung kita terjun ke koding.

Di bawah ini adalah struktur akhir dari folder dan file dari contoh yang akan kita kerjakan. Jika di tengah jalan nanti pembaca merasa bingung dengan peletakan folder serta file maka bisa merujuk ke gambar ini.



Buat sebuah modul di dalam file `app.js` seperti yang sudah-sudah. Akan tetapi, kali ini ada sedikit perbedaan, coba lihat perbedaannya di mana.

```
var app=angular.module('AngularRouteApp',['ngRoute']);
```

Perbedaannya terletak pada apa yang ada di dalam tanda kurung kurawal, yaitu ada nya `ngRoute` yang merupakan modul untuk `route` yang memang dibuat terpisah dari modul utama AngularJs. Untuk bisa menggunakan `route` kita harus memuatnya ke dalam modul yang kita buat.

Selanjutnya buat 3 buah controller yang terletak di dalam folder `controllers`, masing-masing adalah `HomeController` , `AboutController` , dan `ContactController` .

```
app.controller('HomeController',['$scope', function($scope){  
    $scope.title = 'Home';  
    $scope.message = 'My name is Agung Setiawan and I am a Software Engineer';  
}]);
```

```
app.controller('AboutController',['$scope', function($scope){  
    $scope.title = 'About';  
    $scope.message = 'MI have various kind of experience developing software';  
    $scope.technologies = ['C#','ASP.NET MVC','LINQ','Entity Framework','SQL','GIT','Ruby'];  
}]);
```

```
app.controller('ContactController',['$scope', function($scope){  
    $scope.title = 'Contact';  
    $scope.contacts = ['com.agungsetiawan@gmail.com', '@agungsetiawanmu', 'http://agung-setia'];  
}]);
```

Dari masing-masing *controller* di atas kita buatkan halaman html-nya yang terletak di dalam folder pages. Pertama adalah halaman atau template untuk `HomeController`.

```
<h3>{{title}}</h3>  
{{message}}
```

Kemudian template untuk `AboutController`

```
<h3>{{title}}</h3>  
{{message}}  
  
<ul ng-repeat='t in technologies'>  
    <li>{{t}}</li>  
<ul>
```

Dan terakhir adalah template untuk `ContactController`.

```
<h3>{{title}}</h3>  
  
<ul ng-repeat='c in contacts'>  
    <li>{{c}}</li>  
<ul>
```

Nah karena *controller* sudah siap dan template juga sudah siap maka sekarang saatnya kita melakukan konfigurasi untuk *route*. Seperti yang sudah saya katakan di awal bab bahwa yang perlu dikonfigurasi adalah *controller* yang mana dan template yang mana yang akan dipilih ketika *route* yang sedang diakses adalah yang ini.

Karena kita sudah memuat modul `ngRoute` ke dalam modul dari aplikasi kita maka kita memiliki akses untuk menggunakan `$routeProvider`. Komponen inilah yang kita gunakan untuk melakukan konfigurasi.

Pada file `route.js` ketikkan kode di bawah ini yang berperan mengatur *route* pada aplikasi Angular kita.

```
app.config(function($routeProvider){
  $routeProvider.when('/',{
    templateUrl : 'pages/home.html',
    controller : 'HomeController'
  })
  .when('/about',{
    templateUrl : 'pages/about.html',
    controller : 'AboutController'
  })
  .when('/contact',{
    templateUrl : 'pages/contact.html',
    controller : 'ContactController'
  })
  .otherwise({
    redirectTo : '/'
  })
});
```

Tanpa membutuhkan penjelasan yang canggih, kode di atas sudah menjelaskan dirinya sendiri, istilah kerennya *self explanatory*.

Method `when` digunakan untuk mengatur ketika *route* yang ada pada url bernilai **x** maka *controller* dan template yang digunakan adalah **y** dan **z**. Sudah hanya sesederhana itu. Pada kode di atas berarti ketika url yang diakses adalah `http://xxx/#/about` maka *controller* yang bekerja adalah `AboutController` dan template yang digunakan adalah `about.html`.

Selain `when` ada satu lagi pada kode di atas yaitu `otherwise` yang digunakan untuk menangani *route-route* yang tidak dikenal. Jika ada *route* yang tidak dikenal maka method ini yang digunakan. Pada kode di atas jika ada *route* tidak dikenal maka akan dialihkan ke *root url*.

Bagian terakhir adalah membuat halaman utama yang akan memuat semua *controller* serta template ketika *route* bernilai sesuai. Pada file `index.html` ketikkan kode html berikut ini.

```
<!DOCTYPE html>
<html>
<head>
    <title>AngularJs Route</title>
    <script src='https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.min.js'></sc
<script src='https://code.angularjs.org/1.4.7/angular-route.min.js'></script>
</head>

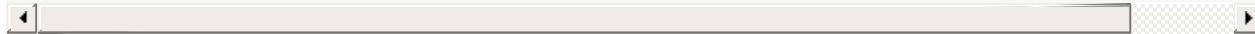
<body ng-app='AngularRouteApp'>

    <a href='#/'>Home</a>
    <a href='#/about'>About</a>
    <a href='#/contact'>Contact</a>

    <div>
        <div ng-view></div>
    </div>

    <script src='js/app.js'></script>
    <script src='js/route.js'></script>
    <script src='js/controllers/HomeController.js'></script>
    <script src='js/controllers/AboutController.js'></script>
    <script src='js/controllers/ContactController.js'></script>

</body>
</html>
```

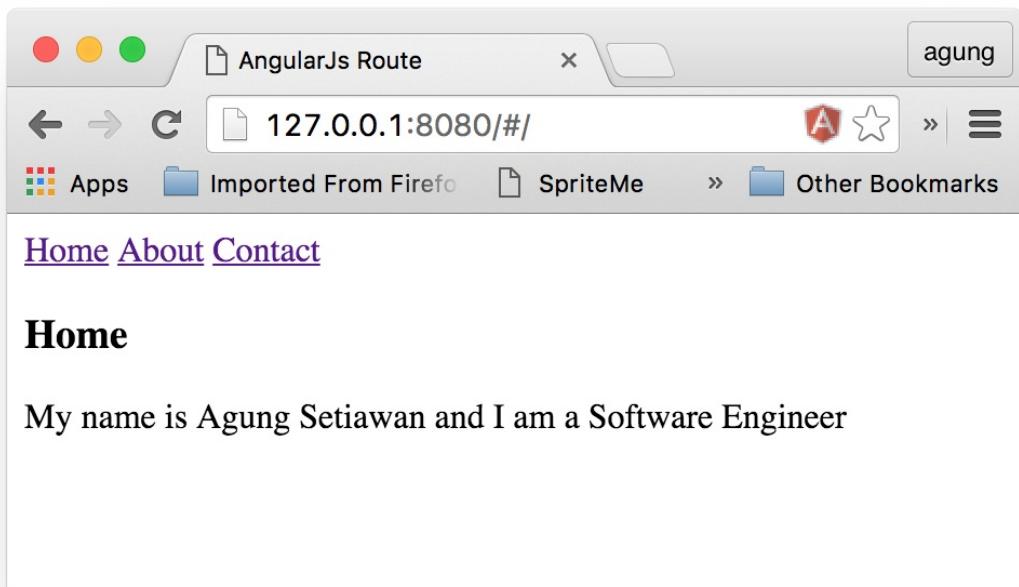


Bentuk html-nya masih mirip dengan yang sebelum – sebelumnya, hanya ada 2 tambahan yang baru. Tambahan pertama adalah kita memuat file `angular-route.min.js`. File inilah yang di dalamnya terdapat modul `ngRoute`. Tambahan kedua adalah penggunaan *directive* `ng-view` yang digunakan sebagai *place holder* untuk template yang akan dimuat sesuai dengan *route*.

**Penting untuk diperhatikan.** Pembaca harus menjalankan kode-kode diatas pada sebuah *web server* seperti sebelumnya pada bab yang membahas *directive*. Jika tidak, maka akan terjadi *error*.

Setelah bisa dijalankan silahkan nikmati perpindahan antar halaman yang terasa sangat *smooth*.

Berikut di bawah adalah *screenshot*-nya.

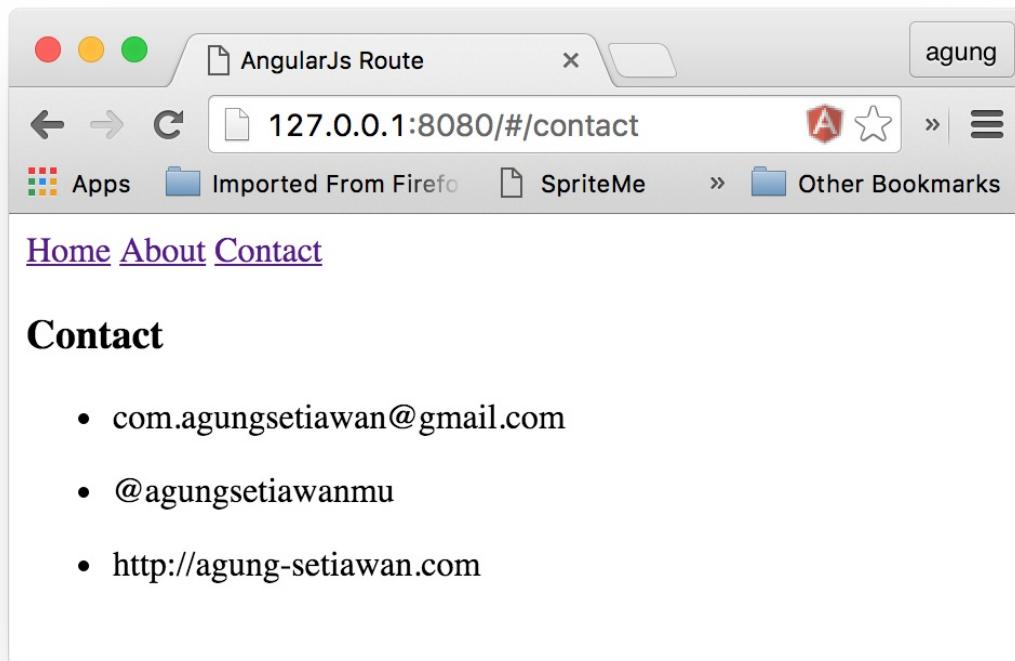


The screenshot shows a web browser window titled "AngularJs Route". The address bar displays "127.0.0.1:8080/#/about". The page content includes a navigation menu with links to "Home", "About", and "Contact". Below the menu, the word "About" is highlighted in bold. A list of technologies is provided under the heading "MI have various kind of experience developing software".

**About**

MI have various kind of experience developing software

- C#
- ASP.NET MVC
- LINQ
- Entity Framework
- SQL
- GIT
- Ruby
- Ruby on Rails



## 4.2 Route Params

Komponen yang satu ini digunakan untuk menangkap parameter yang ada di url. Kalau pembaca sudah pernah menggunakan bahasa pemrograman yang bekerja di sisi *server* pasti tidak asing dengan yang dimaksud url parameter.

Pada contoh url-url yang memiliki *route* pada sub bab 4.1 sebelumnya saya menyertakan satu yang memiliki parameter yaitu

```
http://sample.com/index.html#books/123
```

Angka 123 adalah parameter. Dengan adanya parameter ini kita bisa mengambil nilai 123 untuk kemudian menggunakannya untuk kepentingan aplikasi kita, misalnya mengambil data buku yang memiliki id sama dengan 123. Sekali lagi, jika pembaca sudah pernah ngoding aplikasi web pasti sangat paham dengan konsep ini.

Kembali kita terjun ke koding supaya paham cara mengambil nilai parameter yang ada pada url. Contoh kode yang akan dikerjakan lanjutan dari kode sebelumnya saja ya.

Buat sebuah *controller* baru dengan nama `ParamController` dan di bawah ini adalah kodennya.

```
app.controller('ParamController',['$scope','$routeParams', function($scope,$routeParams){  
    $scope.title = 'Parametes';  
    $scope.param = $routeParams.theparam;  
}]);
```

Di bagian *controller* ini ada sebuah komponen baru yang di-*inject* yaitu `$routeParams` yang digunakan untuk mengambil nilai parameter yang ada di url. Pada bagian isi ada kode seperti ini

```
$scope.param = $routeParams.theparam;
```

Dari mana bagian `theparam` didapatkan? Jawabannya adalah ada pada bagian konfigurasi *route*. Pada konfigurasi yang sudah ada tambahkan *route* baru sebagai berikut.

```
when('/param/:theparam', {  
    templateUrl : 'pages/param.html',  
    controller : 'ParamController'  
})
```

Pada konfigurasi *route* kita mendefinisikan nama parameter yang akan diambil melalui *controller*, pada kode di atas nama parameternya adalah `theparam`. Pada kedua bagian itu nama harus sama, jika di konfigurasi namanya adalah `paramsay a` maka pada *controller* juga **harus** `paramsaya`. Sudah jelas bukan sekarang?

Selanjutnya membuat template yang akan ditampilkan.

```
<h3>{{title}}</h3>  
  
The parameter is {{param}}
```

Dan langkah terakhir adalah menambahkan link untuk menuju halaman parameter dan menambahkan file *controller* baru pada file `index.html`.

```
...  
<a href='#/param/agung'>Param</a>  
...  
...  
...  
<script src='js/controllers/ParamController.js'></script>  
...
```

Coba akses link baru tersebut dan di bawah ini adalah hasilnya.



# Bab 5 \$watch, \$digest dan \$apply

Sudah kita pelajari bersama pada bab – bab awal bahwa ketika ada perubahan nilai pada *view* maka perubahan juga akan terjadi pada *scope* yang ada di *controller* dan begitu juga sebaliknya ketika nilai scope di *controller* berubah maka perubahan pada *view* pun akan terjadi juga.

Kemampuan Angular untuk melakukan hal – hal tersebut berhubungan dengan method – method yang dimiliknya yaitu `$watch` , `$digest` dan `$apply` . Secara umum, `$watch` dan `$digest` akan dijalankan secara automatis oleh Angular untuk memperbarui nilai tetapi ada kondisi dimana kita harus menggunakannya secara manual untuk mengupdate nilai.

## 5.1 \$watch

`$watch` berfungsi untuk mengawasi perubahan nilai yang terjadi pada sebuah variabel di *scope*. Parameter pertama dari method ini adalah string dari nama variabel yang akan diawasi kemudian parameter kedua adalah berupa fungsi yang akan dijalankan (*listener*) ketika nilai yang diawasi berubah.

Saya sertakan potongan kode contoh di bawah ini untuk memahami cara kerja `$watch` .

```
<body ng-app='AngularWatchApp' ng-controller='MainController'>
  <input type='text' ng-model='book'/>

  <div>
    {{book}}
  </div>

  <div>
    Book has been changed {{count}} times.
  </div>

  <!-- include .js here --!>
</body>
```

Halaman di atas akan memunculkan teks masukan yang nilainya langsung dimunculkan di bawahnya. Kita akan memunculkan angka berapa kali nilai sudah diubah dengan menggunakan `$watch` . Pada *controller* berikut adalah kodenya.

```
app.controller('MainController', function($scope){
    $scope.book = 'Zero to One';
    $scope.count = 0;

    $scope.$watch('book', function(newValue, oldValue){
        $scope.count = $scope.count + 1;
    })
});
```

Coba jalankan dan ubah nilai yang ada pada teks masukan maka setiap perubahan yang terjadi akan membuat nilai dari counter bertambah.

## 5.2 \$digest

Method ini bekerja dengan cara berkeliling ke semua *watch* yang ada. Ketika berkeliling nyamperin tiap *watch*, dia ngecek apakah nilai yang diawasi si *watch* ini berubah atau tidak, jika berubah maka dia panggil fungsi yang jadi parameter kedua dari si *watch* (fungsi *listener*).

Ada suatu kasus dimana kita harus memanggil `$digest` supaya nilai dari variabel bisa terupdate. Pada kasus tertentu saat melakukan proses *data binding* ada data terbaru yang tidak terupdate maka kondisi seperti ini biasanya karena `$digest` tidak jalan secara automatis dan kita harus menjalankannya secara manual.

```
link : function(scope, elem, attrs){
    elem.bind('click', function(){
        scope.dunia='Saya diklik';
        scope.$digest();
    });
}
```

Contoh kode di atas saya ambil dari kode saat kita belajar bersama materi *directive*. Coba jalankan kode yang ada pada materi tersebut tetapi dengan menghilangkan baris

`scope.$digest()` maka variabel `dunia` yang berisi `saya diklik` tidak akan muncul di-view. Hal ini terjadi karena `$digest` tidak jalan karena kode ini berada diluar jangkauan langsung dari `$scope` yang ada di *controller*.

## 5.3 \$apply

Kalau saya bilang method ini merupakan penyederhanaan dari penggunaan `$digest`. `$apply` menjalankan fungsi yang menjadi parameternya kemudian secara automatis dia akan menjalankan `$digest`. Dengan pengertian ini maka kode sebelumnya bisa diubah menjadi seperti ini.

```
link : function(scope,elem,attrs){
  elem.bind('click',function(){
    scope.$apply(function(){
      scope.dunia='Saya diklik';
    });
  });
}
```

# Bab 6 Contoh Aplikasi CRUD Menggunakan Web API

Coming soon ya, stay tuned ;)

Aplikasi contohnya sudah jadi, web api-nya pun sudah jadi. Masih perlu waktu untuk menuliskannya di buku ini jadi kalimat yang enak dipahami.

Sebagai iming-iming maka saya sertakan beberapa *screenshot* contoh aplikasinya.

