

LARAVEL VIEW & TEMPLATING

KELAS PROGRAMMING FULL STACK DEVELOPER

MITRA PELATIHAN



Jabar Digital Academy

digitalacademy.jabarprov.go.id

2024

BAB III

LARAVEL VIEW & TEMPLATING

1. Tujuan

- a. Peserta didik dapat memahami *View* pada Laravel.
- b. Peserta didik dapat memahami cara *Passing Data* ke *views*.
- c. Peserta didik dapat memahami *Blade templating*.
- d. Peserta didik dapat memahami cara membuat *Layout*.
- e. Peserta didik dapat memahami penggunaan *Component* secara dinamis.
- f. Peserta didik dapat memahami cara menangani *Form* dalam *view*.

2. Perlengkapan

- a. Modul 3. Laravel View and Templating
- b. IDE atau Teks Editor (Visual Studio Code, Notepad++, Sublime)

3. Materi

Dalam sebuah aplikasi web, terdapat banyak halaman yang berbeda-beda, namun seringkali beberapa di antaranya memiliki komponen-komponen yang serupa, dengan perbedaan terletak pada tata letaknya. Contoh hal ini adalah pada navigasi, header, atau footer yang mungkin tetap konsisten di sebagian besar halaman, tetapi isi konten utama berubah. Dalam situasi seperti ini, penggunaan templating menjadi sangat penting. Dengan menggunakan templating, kita dapat membuat struktur dasar yang konsisten untuk semua halaman, dan hanya mengubah bagian-bagian tertentu yang berbeda antar halaman.

Dalam Laravel, konsep MVC memungkinkan pemisahan yang jelas antara logika aplikasi dan tampilan. Bagian "view" dari MVC adalah komponen yang bertanggung jawab menangani semua kebutuhan di sisi tampilan dan menampilkan data kepada pengguna. Melalui view, kita dapat melakukan layouting pada halaman dan mengelola komponen, sehingga memungkinkan untuk merancang dan mengatur tampilan dengan cara yang menarik dan mudah dinavigasi. Dengan demikian, penggunaan konsep MVC dalam Laravel memungkinkan pengembangan aplikasi

web yang terstruktur dan terorganisir dengan baik, serta memberikan pengalaman pengguna yang lebih baik.

1) Views

View adalah sebuah tempat untuk kita dapat mengatur tampilan pada halaman web kita tanpa perlu berfokus pada logika aplikasi kita. dengan view template biasanya ditulis menggunakan blade templating, untuk membuat blade templating cukup beri saja ekstensi file bernama *.blade* dengan begitu semua perintah yang ada didalam *blade templating*.

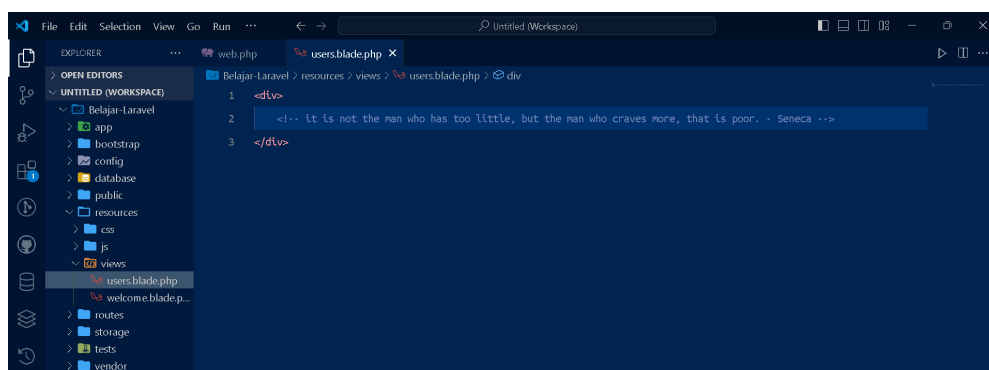
Untuk membuat sebuah *view* bisa dengan menjalankan perintah ini di dalam command prompt, pastikan posisi di dalam cmd berada di dalam project Laravel.

```
php artisan make:view (nama_file)
```

Kita akan coba membuat sebuah view yang akan menampilkan daftar user.

```
php artisan make:view users
```

Ketika sudah menjalankan perintah diatas maka secara otomatis dibuatkan file yang sesuai dengan nama yang sudah kita tentukan, kita bisa lihat di folder *resources/views*.

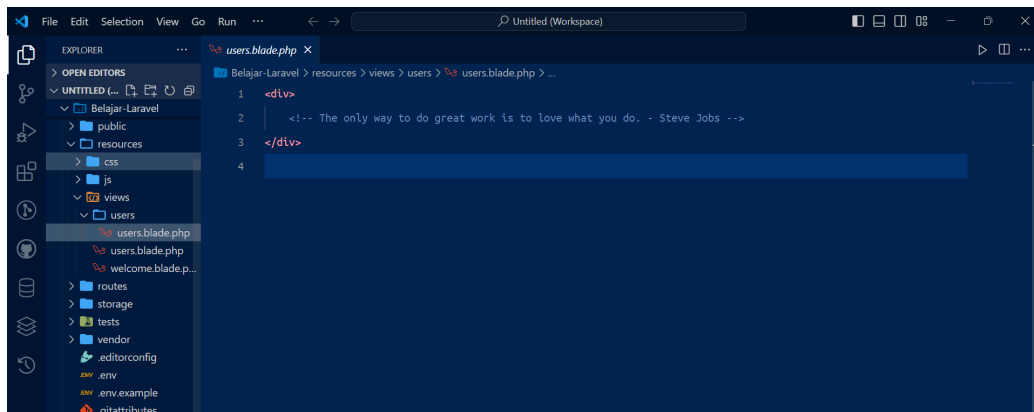


Contoh View Yang Sudah Dibuat.

Kita juga dapat membuat sebuah view yang akan ditempatkan di dalam folder yang ada di dalam folder *views*, dapat kita lakukan dengan menggunakan perintah ini.

php artisan make:view users/users

Bisa kita periksa didalam folder *views* maka secara otomatis dibuatkan folder *users* dan juga file *users*.



Folder User Di Dalam *Views* Ditambahkan.

Didalam folder *views* kita dapat membuat struktur layout sehingga kita hanya perlu membuat satu layout namun digunakan di banyak halaman, selain membuat templating kita juga dapat membuat sebuah komponen yang dinamis sehingga kita hanya perlu membuat satu komponen namun bisa kita atur nilai di dalam komponen tersebut sehingga komponen dinamis.

Didalam file *blade* ini kita dapat memberikan struktur HTML, memberikan CSS atau Javascript jika diperlukan. Kita juga bisa menyisipkan *code* php di dalam struktur HTML kita untuk keperluan seperti menampilkan data atau mengolah data.

Untuk melihat *view* yang sudah kita buat, kita perlu membuat sebuah *route* lalu di dalamnya *action* nya tambahkan nilai *return* dengan memanggil *function view*, lalu didalamnya berisikan nama dari *file* yang ada didalam folder *view*. Jika kita lihat di *route* pada halaman awal, nilai *return* didalamnya berisikan *view* yang akan menjalankan file *welcome* didalam folder *view*. Kita akan mencoba membuat *route* yang akan menjalankan file *users.blade.php* yang berada di luar folder *user*.

```
// routes/web.php

Route::get("/users", function(){
    return view("users");
});
```

Pembuatan Route Untuk Menjalankan View.

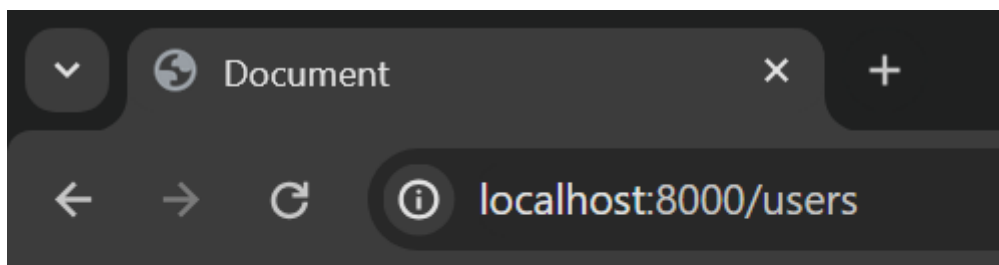
Lalu didalam file *users.blade.php* kita tambahkan struktur HTML sederhana seperti ini.

```
<!-- resources/views/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>Ini halaman list users</h1>
  </body>
</html>
```

Struktur HTML Dasar Di Dalam File Blade.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



Ini halaman list users

Tampilan Ketika Mengakses *Route Users*.

Sekarang kita akan coba menampilkan *view* yang ada didalam folder *user*. untuk menjalankan *view* yang ada di berada di dalam folder apapun yang ada di dalam *view* itu seperti ini.

```
// routes/web.php

Route::get("/users", function(){
    return view("users.users");
});
```

Pemanggilan View Yang Berada Didalam Folder *User*.

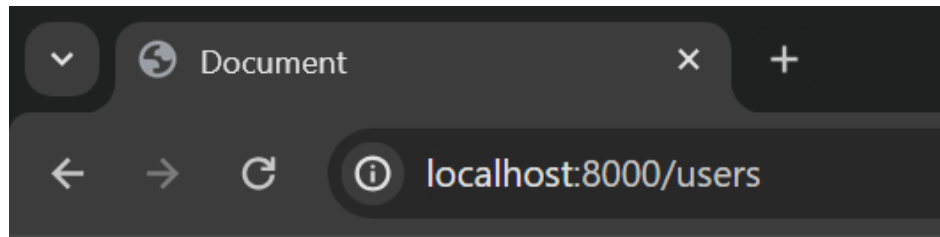
Untuk memisahkan antara nama folder dengan nama file dengan titik. Lalu didalam file *users.blade.php* yang berada di dalam folder *user*, kita buat struktur HTML di dalamnya seperti ini.

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>Ini halaman list users</h1>
    <h2>yang berada didalam folder user</h2>
  </body>
</html>
```

Struktur HTML Didalam *users*.

Ketika kita jalankan di browser maka akan tampil seperti ini.



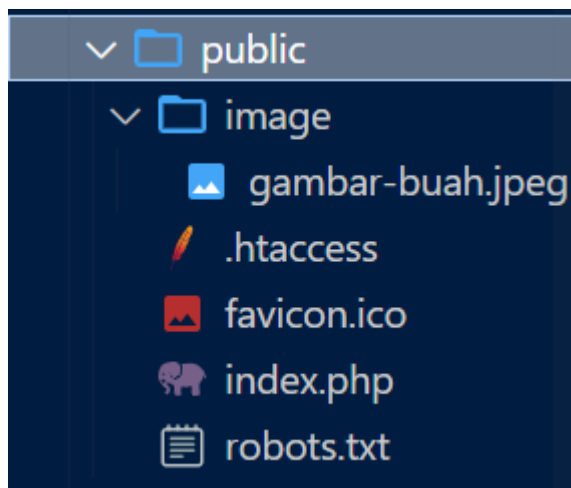
Ini halaman list users yang berada didalam folder user

Tampilan Dari View *users*.

Agar terlihat teratur dan tertata, kita hapus saja file *users.blade.php* yang berada di luar folder *user*.

2) Assets.

Jika assets yang akan kita pakai dalam aplikasi laravel kita itu berupa file yang diunduh, kita dapat menyimpan semua asetnya di dalam folder `public`, untuk menggunakan asset yang ada di dalam folder `public`, kita dapat menggunakan fungsi asset didalam source yang akan digunakan, contohnya saya memiliki gambar didalam folder `public` seperti ini.



Lalu untuk menggunakan gambar yang ada di dalam `public` menggunakan function asset seperti ini.

```

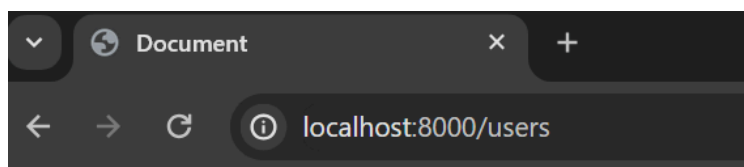
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>Ini halaman list user</h1>
    <h2>Dibawah ini gambar buah buahan</h2>
    
  </body>
</html>

```

Penggunaan Asset Yang Akan Ditampilkan Di Browser.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



Ini halaman list user

Dibawah ini gambar buah buahan



3) Passing Data.

Saat mengembangkan tampilan menggunakan *view*, kita perlu menampilkan data yang diperlukan, untuk itu kita perlu mengirim data dari *action* yang ada didalam *route* ke *view*. Caranya adalah dengan menambahkan argumen dalam *function view* berupa *array associative*, lalu tentukan *key* dan *valuenya*. Contohnya seperti ini.


```
// routes/web.php

Route::get("/users", function(){
    return view("users.users", [
        "id" => 1,
        "nama" => "juna",
        "age" => 18
    ]);
});
```

Penambahan Argumen Di Dalam Function View.

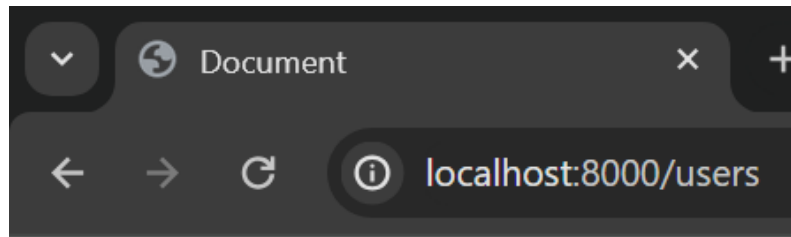
Cara mengolah data yang dikirim dalam *route* yaitu dengan menggunakan variabel yang sesuai dengan *key* yang dikirim dari *route*. Contohnya seperti ini.

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1><?php echo $id ?></h1>
    <h1><?php echo $name ?></h1>
    <h1><?php echo $age ?></h1>
  </body>
</html>
```

Contoh Penggunaan Data Yang Dikirim Dari *Route*.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



1

juna

18

Tampilan Ketika Dijalankan Di Browser.

4) Blade Templates.

Di dalam laravel terdapat *templating* yang sudah disediakan, kita dapat menggunakannya sesuai dengan kebutuhan kita. *Blade Templates* adalah fitur dalam Laravel yang memungkinkan kita untuk membuat tampilan HTML yang dinamis dan mudah dikelola. *Blade* memungkinkan penggunaan sintaks sederhana dan ekspresif untuk menghasilkan tampilan yang konsisten dan mudah dipelihara. Kita dapat menggunakan code php baik yang murni maupun template yang sudah disediakan oleh *blade*. Semua *code blade template* akan di compile menjadi *code php* biasa, semua hasil compile bisa kita lihat di dalam folder *storage/framework/views*. Berikut perbandingan penggunaan PHP murni dengan menggunakan blade template.

Jika sebelumnya kita untuk menampilkan data yang dikirim dari route menggunakan PHP murni seperti ini.

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1><?php echo $id ?></h1>
    <h1><?php echo $name ?></h1>
    <h1><?php echo $age ?></h1>
  </body>
</html>
```

Contoh Code Menggunakan PHP Murni.

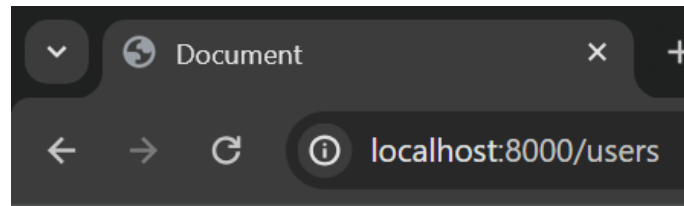
Jika kita menggunakan Blade Templating menjadi seperti ini.

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>{{ $id }}</h1>
    <h1>{{ $name }}</h1>
    <h1>{{ $age }}</h1>
  </body>
</html>
```

Contoh Code Menggunakan Blade Template.

Ketika kita jalankan maka tampilannya akan sama.



1

juna

18

Tampilan Ketika Dijalankan Di Browser.

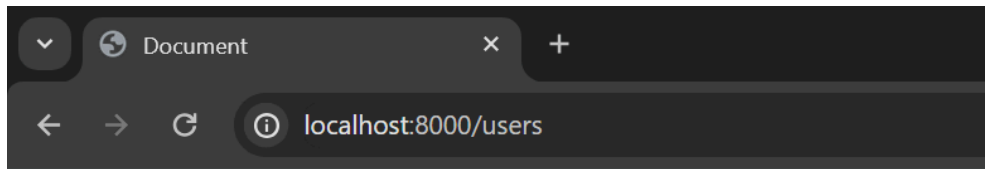
Perlu diketahui juga, bahwa Anda tidak dibatasi untuk menampilkan konten variabel yang diteruskan ke view. Anda juga dapat menampilkan hasil fungsi PHP apapun. Jika kita tambahkan elemen HTML di dalam route seperti ini.

```
// routes/web.php

Route::get("/users", function(){
    return view("users.users", [
        "id" => 1,
        "nama" => "<h4 style='color:#ff0000'>juna</h4>",
        "age" => 18
    ]);
});
```

Code Menambahkan Elemen HTML Di Dalam

Secara default, pernyataan Blade `{{ }}` secara otomatis dikirim melalui fungsi PHP `htmlspecialchars` untuk mencegah serangan XSS. Ketika kita jalankan maka elemen HTML tidak akan dijalankan dan ditampilkan dalam bentuk teks.



1

<h4 style='color:#ff0000;'>juna</h4>

18

Tampilan Ketika Dijalankan Di Browser.

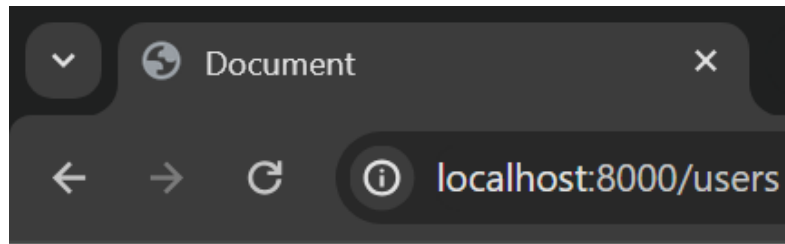
Jika kita tidak ingin data yang di-escape, kita dapat menambahkan !! sebelum dan sesudah data yang akan ditampilkan, contohnya seperti ini:

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>{!! $id !!</h1>
    <h1>{!! $name !!</h1>
    <h1>{!! $age !!</h1>
  </body>
</html>
```

Contoh Code Untuk Mengabaikan Serangan XSS.

Ketika kita jalankan di browser maka akan menampilkan seperti ini.



1

juna

18

Tampilan Ketika Dijalankan Di Browser.

Untuk membuat komentar menggunakan *blade template* yaitu kita tambahkan saja `{{-- pesan komentar --}}`, komentar tidak akan dieksekusi, contoh penggunaannya seperti ini.

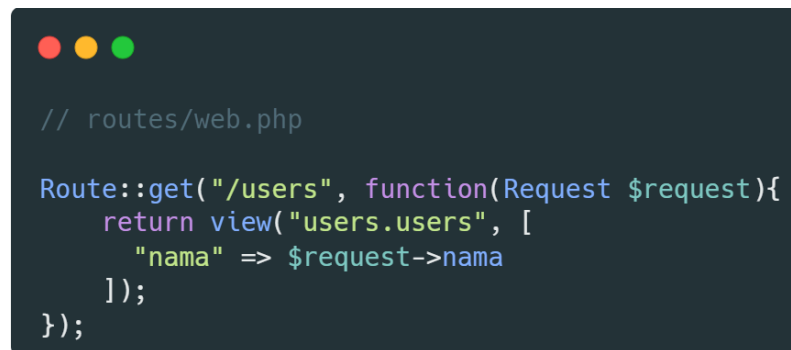
```
<!-- resources/views/users/users.blade.php -->

{{-- ini komentar --}}
<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>{!! $id !!</h1>
    <h1>{!! $name !!</h1>
    <h1>{!! $age !!</h1>
  </body>
</html>
```

Komentar Di dalam Blade Template.

A. If Statements. (Ternary Operator)

Di dalam *blade template* terdapat *if statement* seperti *if*, *else*, *else if*. Fungsi dari *if statements* ini sama seperti pengkondisian pada umumnya. Saat menggunakan *If Statements* dalam *blade template*, kita dapat memberikan tanda akhir dari *block code* yang akan dieksekusi dengan menambahkan kata kunci `@end` diikuti dengan jenis struktur kontrol yang sedang digunakan, misalnya `@endif` untuk *if statement*. Ini membantu dalam mengidentifikasi akhir dari *block code* yang sedang dievaluasi sehingga memudahkan pembacaan dan pemahaman struktur kode. Kita akan langsung praktekkan, pertama-tama kita ambil data dari query URL di dalam *route*, seperti ini.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in PHP and is a route definition for a web application. It shows a GET route for '/users' that returns a view named 'users.users' with a data array containing 'nama' mapped to '\$request->nama'.

```
// routes/web.php

Route::get("/users", function(Request $request){
    return view("users.users", [
        "nama" => $request->nama
    ]);
});
```

Mengirim Data Yang Diterima Dari URL Ke *View*.

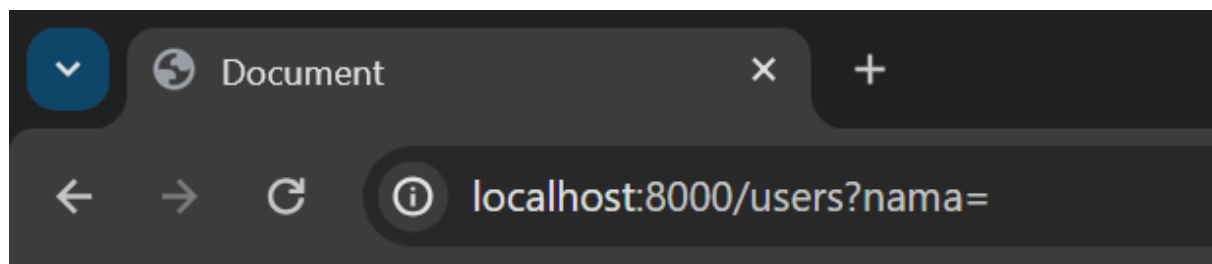
Lalu kita olah data yang dikirim dari route menggunakan pengkondisian, jadi kita akan tampilkan *hello* lalu diikuti dengan nama yang diberikan, jika nilai dari parameter di url kosong, maka akan tampil data tidak ditemukan. Contoh penggunaan *if* didalam *blade template*.

```
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    @if($nama != null)
      <h1>Hellooo, {{ $nama }}</h1>
    @else
      <h1>user tidak ditemukan</h1>
    @endif
  </body>
</html>
```

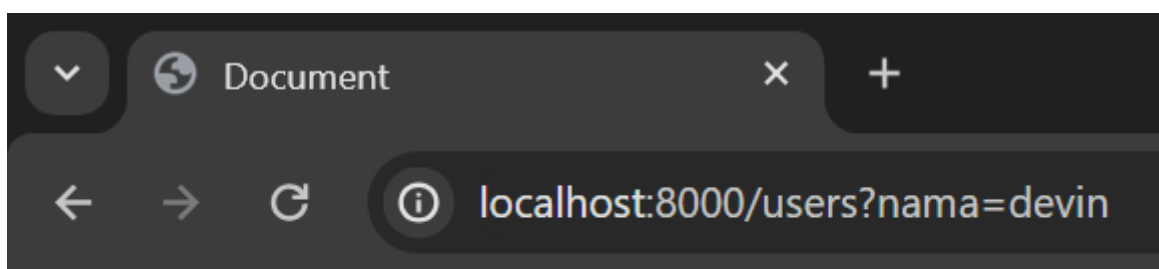
Penggunaan *If* Di Dalam *Blade Template*.

Jika kita jalankan di browser maka akan terlihat seperti ini.



user tidak ditemukan

Tampilan Ketika *Query* Di URL Kosong.



Hellooo, devin

Tampilan Ketika *Query* Di URL Tidak Kosong.

Kita juga dapat menggunakan *ternary operator* untuk melakukan pengkondisian. *ternary operator* adalah sebuah operator yang digunakan untuk mengevaluasi dari kedua data yang nantinya akan bernilai true atau false tergantung dari kondisi yang diberikan. penggunaan *ternary operator* seperti ini.

Condition ? TrueExpression : FalseExpression;

Kita juga bisa membuat *Ternary operator* bersarang atau *nested ternary operator* seperti ini.

**Condition1 ? TrueExpression :
Condition2 ? TrueExpression : FalseExpression;**

Penggunaan Ternary operator bersarang sangat tidak disarankan jika memiliki kondisi yang banyak karena *code* yang dibuat akan sulit dipelihara. Contoh penggunaan Ternary operator

```
// resources/views/user/ListUser.blade.php

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    @if($nama != null)
      <h1>Hellooo, {{ $nama }}</h1>
    @else
      <h1>user tidak ditemukan</h1>
    @endif
  </body>
</html>
```

Contoh Pengkondisian Menggunakan *If Statement*.

```

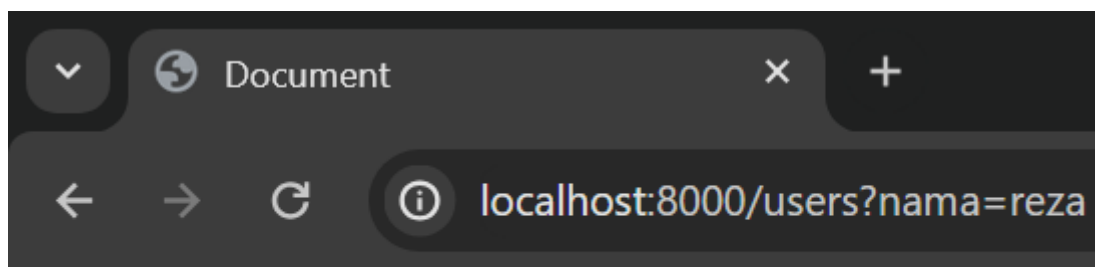
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>{{ $nama != null ? "hellooo ".$nama : "User tidak ditemukan." }}</h1>
  </body>
</html>

```

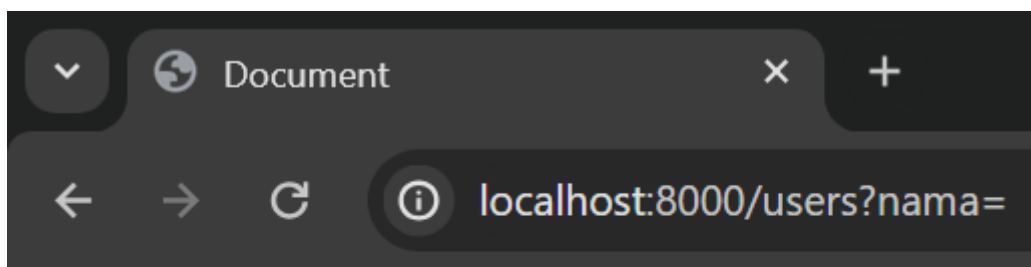
Contoh Penggunaan Ternary Operator.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



haloo reza

Tampilan Di Browser Ketika Memiliki Nilai.



User tidak ditemukan.

Tampilan Di Browser Ketika Tidak Memiliki Nilai.

B. Looping.

Didalam *Blade Template* kita dapat menggunakan *looping* didalam *view* kita. Kita dapat menggunakan looping yang ada di dalam PHP seperti *for*, *foreach*, *while*. Jenis *looping* yang sering digunakan yaitu *foreach* untuk menampilkan data berupa

list atau *array*. Penulisan *looping* sama seperti pada umumnya namun kita dapat menyisipkan elemen HTML di dalamnya, lalu diakhiri dengan *@end* lalu diikuti dengan jenis struktur kontrol yang digunakan.

Kita akan coba praktekkan namun kita buat dulu data *user* di dalam *route*, lalu kita kirim ke *view* agar kita dapat olah datanya di dalam *view*.

```
// routes/web.php

Route::get('/users', function(){
    $users = [
        [
            "id" => 1,
            "nama" => "ronald Alberty",
            "umur" => 20,
        ],
        [
            "id" => 2,
            "nama" => "kevin Leonardo",
            "umur" => 24,
        ],
        [
            "id" => 3,
            "nama" => "Stevano Michael",
            "umur" => 20,
        ]
    ];

    return view("users.users", [
        "users" => $users
    ]);
});
```

Code Membuat Data *User* Lalu Dikirim Ke *View*.

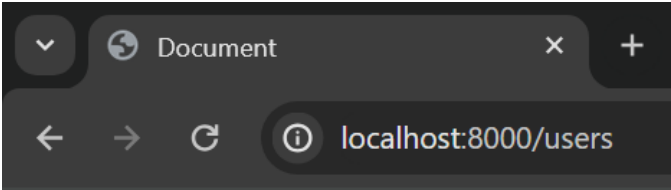
Setelah kita buat data yang akan kita kirim ke *view*, kita akan buat tabel yang di dalam data user yang sudah didapatkan dari route, kita akan menggunakan for loop untuk menampilkan data, lalu didalamnya kita tambahkan elemen untuk menampilkan baris dalam tabel, seperti ini.

```
// resources/views/users/users.blade.php

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @for($i = 0; $i < count($users); $i++)
          <tr>
            <td>{{ $users[$i]["id"] }}</td>
            <td>{{ $users[$i]["nama"] }}</td>
            <td>{{ $users[$i]["umur"] }}</td>
          </tr>
        @endfor
      </tbody>
    </table>
  </body>
</html>
```

Penggunaan Perulangan Dengan *For* Dalam *Blade Template*.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



The screenshot shows a web browser window with the title 'Document'. The address bar displays 'localhost:8000/users'. Below the browser window, a table is rendered with three columns: 'no', 'nama', and 'umur'. The table contains three rows of data.

no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Hasil Dari Perulangan Dengan *For* Di dalam *View*.

Ketika menggunakan *looping For* di *Blade template*, kita harus menentukan *indeks array* dan kunci secara eksplisit, yang bisa sedikit rumit. Sebagai alternatif, kita bisa menggunakan *foreach*, yang secara otomatis menangani perulangan data dalam array tanpa perlu menentukan indeks dan kunci secara manual. Hal ini membuat kode lebih sederhana dan mudah dipahami.

```

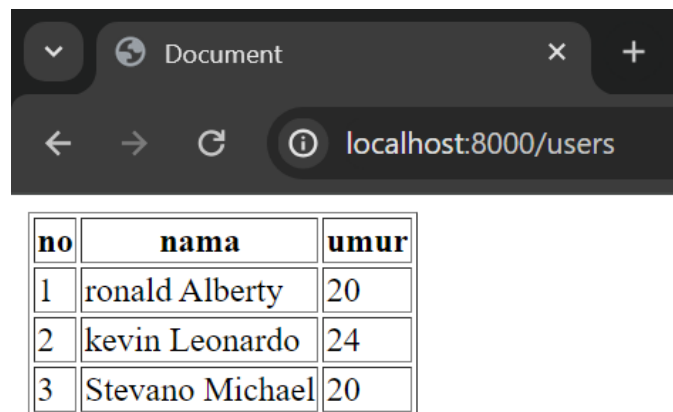
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @foreach($users as $user)
          <tr>
            <td>{{ $user["id"] }}</td>
            <td>{{ $user["nama"] }}</td>
            <td>{{ $user["umur"] }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </body>
</html>

```

Contoh Penggunaan *Foreach* Didalam *Blade Template*.

Ketika kita jalankan maka tampilannya akan sama.



no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Hasil Dari Perulangan *Foreach* Di dalam *View*.

Jika kita memerlukan index seberapa perulangan yang dilakukan, kita cukup tambahkan variabel baru lalu beri => diikuti dengan variabel yang akan digunakan untuk menampilkan data yang ada di dalam array, contohnya seperti ini.

```

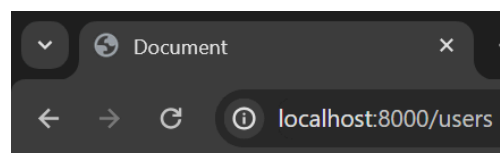
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @foreach($users as $index => $user)
          <tr>
            <h1>Perulangan yang ke {{ $index }}</h1>
            <td>{{ $user["id"] }}</td>
            <td>{{ $user["nama"] }}</td>
            <td>{{ $user["umur"] }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </body>
</html>

```

Menambahkan Index Di Dalam Foreach.

Ketika kita jalankan maka terlihat seperti ini.



Perulangan yang ke 0

Perulangan yang ke 1

Perulangan yang ke 2

no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Tampilan Ketika Dijalankan Di Browser.

C. Loop Variable.

Ketika kita menggunakan *foreach* Didalamnya terdapat *variable* yang sudah disediakan oleh *blade template* yang dapat digunakan untuk keperluan kita, Berikut beberapa variabel beserta fungsi yang sering digunakan didalam *foreach*.

Fungsi	Deskripsi
<code>\$loop->index</code>	Indeks iterasi loop saat ini (dimulai dari 0).
<code>\$loop->iteration</code>	Iterasi loop saat ini (dimulai dari 1).
<code>\$loop->first</code>	Mengambil index pertama dari data array.
<code>\$loop->last</code>	Mengambil index terakhir dari data array.
<code>\$loop->count</code>	Mendapatkan jumlah data yang ada didalam array.
<code>\$loop->parent</code>	Ketika melakukan looping bersarang maka parentnya yang akan dijadikan sebagai referensi

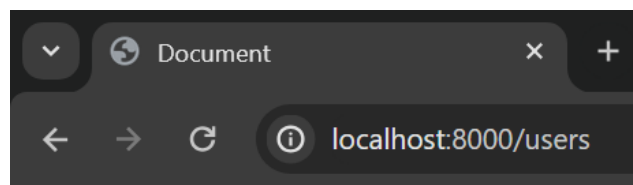
Contoh penggunaannya seperti ini.

```
// resources/views/user/ListUser.blade.php

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @foreach($users as $user)
          <tr @if($loop->first)
            style="background-color: salmon"
          @endif >
            @foreach ($user as $a => $s)
              <td @if($loop->parent->index % 2 == 0)
                style="font-weight:800"
              @endif >{{ $s }}</td>
            @endforeach
          </tr>
        @endforeach
      </tbody>
    </table>
  </body>
</html>
```

Penggunaan *Loop Variabel* Didalam *Foreach*.

Pada kondisi if pertama, jika index dari perulangan yang sedang dijalankan adalah satu maka warna background dari barisnya akan berbeda dengan baris yang lainnya. Lalu kita tambahkan *foreach* berdasarkan data dari variabel *user* agar kita tidak perlu mendefinisikan elemen *td* secara berulang dan di dalamnya tambahkan kondisi yang didalamnya memanggil variabel *loop* lalu kita panggil juga *parent* agar index yang dipakai adalah dari perulangan yang *parent*. Jika kita jalankan di browser maka akan terlihat seperti ini.



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/users'. Below the browser window, a table is displayed with three columns: 'no', 'nama', and 'umur'. The first row is highlighted with a red background.

no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Tampilan Ketika Dijalankan Di Browser.

D. Raw PHP.

Terkadang, *Blade template* bawaan Laravel tidak cukup untuk kebutuhan tampilan kita. Untuk menangani hal tersebut, kita bisa menambahkan kode PHP langsung di dalam *view* menggunakan *@php directive*. Ini memberi kita fleksibilitas untuk menambahkan logika kustom langsung di dalam tampilan *Blade*.


```

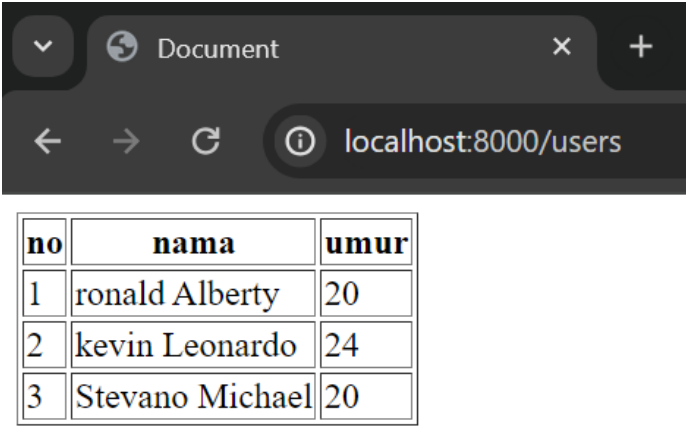
<!-- resources/views/users/users.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @php
          $i = 1;
        @endphp
        @foreach($users as $user)
          <tr>
            <td>{{ $i++ }}</td>
            <td>{{ $user["nama"] }}</td>
            <td>{{ $user["umur"] }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </body>
</html>

```

Contoh Penggunaan *@php* Di Dalam View.

Pada kolom nomor akan diisi dengan penjumlahan dari variabel *\$i* tergantung berapa banyak perulangan dilakukan, Ketika dijalankan di browser maka akan terlihat seperti ini.



no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Tampilan Ketika Dijalankan Di Browser.

E. Include

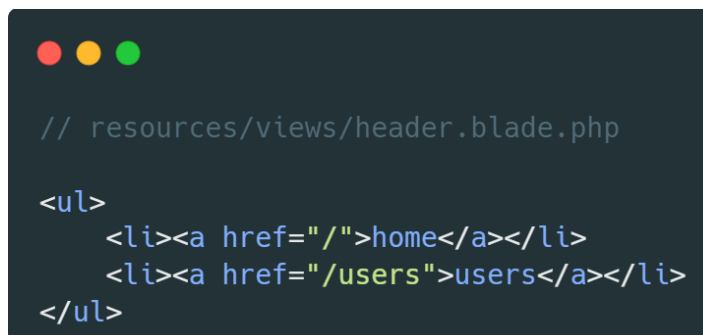
Include dalam Laravel memungkinkan kita untuk menyertakan tampilan (view) lain ke dalam tampilan utama Anda. Dengan menggunakan *include*, kita dapat membuat sebuah tampilan yang bersifat modular, yang dapat digunakan kembali di berbagai bagian aplikasi Anda. Hal ini memungkinkan kita untuk menciptakan kode yang lebih bersih, terstruktur, dan mudah dipelihara. Contoh penggunaan *include* seperti ini.

Pertama-tama kita buat view bernama *header* dan *detail* di dalam folder user.

```
php artisan make:view header
```

```
php artisan make:view users/detail
```

Lalu kita buat *route* baru yang akan menampilkan detail user, setiap halaman kita tambahkan *@include* di dalamnya kita buat baris link yang mengarah ke halaman user dan halaman awal, seperti ini.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a light blue font and represents a Blade template for a header view. It starts with a comment line, followed by an opening tag, two elements containing links to the home and users pages, and ends with a closing tag.

```
// resources/views/header.blade.php

<ul>
  <li><a href="/">home</a></li>
  <li><a href="/users">users</a></li>
</ul>
```

Code Didalam View Header.

Lalu kita buat route baru yang memiliki URL dinamis yang akan menampilkan data user berdasarkan ID yang dikirim dari URL, dan tampilkan view *detail* yang sudah kita, dan kita beri nama users.detail Seperti di bawah ini.

```
// routes/web.php

Route::get('/users/{id}', function(int $id){
    $users = [
        [
            "id" => 1,
            "nama" => "ronald Alberty",
            "umur" => 20,
        ],
        [
            "id" => 2,
            "nama" => "kevin Leonardo",
            "umur" => 24,
        ],
        [
            "id" => 3,
            "nama" => "Stevano Michael",
            "umur" => 20,
        ]
    ];

    $result = null;
    foreach($users as $user){
        if($user["id"] == $id){
            $result = $user;
        }
    }

    return view("user.detail", [
        "user" => $result
    ]);
})->name("users.detail");
```

Route Baru Untuk Menampilkan *detail*.

Lalu didalam *view detail* kita akan tampilkan data yang sudah dikirim dari *route* dan kita tambahkan include untuk menampilkan *header*, seperti ini.

```
<!-- resources/views/users/detail.blade.php -->

<html>
    <head>
        <title>
            {{ $user["nama"] }}
        </title>
    </head>
    <body>
        @include("header")
        <h1>Hello, {{ $user["nama"] }}</h1>
        <h2>Umurku {{ $user["umur"] }}</h2>
    </body>
</html>
```

Code Didalam *View detail user*.

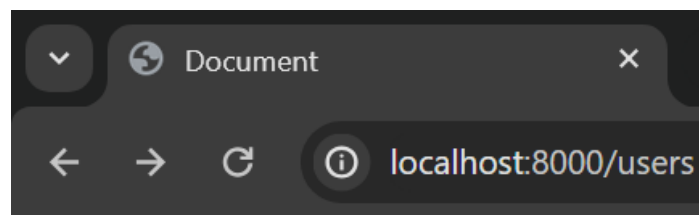
Lalu didalam view user di atas tabel kita tambahkan *include* yang berisikan nama file header, dan kita tambahkan elemen *hyperlink* di kolom nama dan arahkan *link* ketika di klik ke *route* yang sudah kita buat sebelumnya, nilai attribute *href* kita tidak perlu ketik *full* url nya, kita bisa menggunakan *function route* untuk mendapatkan url sesuai dengan nama *route* nya. Contohnya seperti ini.

```
<!-- resources/views/users/detail.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    @include("header")
    <table border="1">
      <thead>
        <tr>
          <th>no</th>
          <th>nama</th>
          <th>umur</th>
        </tr>
      </thead>
      <tbody>
        @foreach($users as $user)
          <tr>
            <td>{{ $user["id"] }}</td>
            <td>
              <a href="{{ route('users.detail', $user['id']) }}">{{ $user["nama"] }}</a>
            </td>
            <td>{{ $user["umur"] }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </body>
</html>
```

Penggunaan *Include* Di Dalam *View User*.

Ketika kita jalankan di browser maka akan terlihat seperti ini.

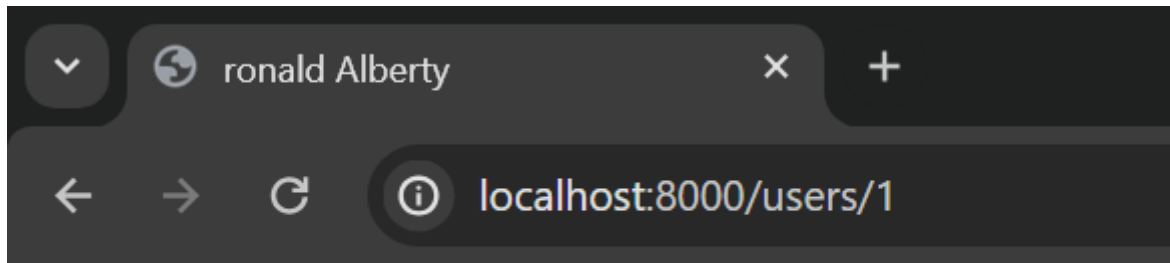


- [home](#)
- [users](#)

no	nama	umur
1	ronald Alberty	20
2	kevin Leonardo	24
3	Stevano Michael	20

Tampilan Ketika Dijalankan Di Browser.

Ketika kita *klik* nama di dalam tabel maka akan mengarahkan ke halaman *route* yang sudah kita.



- [home](#)
- [users](#)

Helloo, saya ronald Alberty

Umurku 20

Tampilan Halaman *Detail User*.

Kedua halaman ini memiliki dua *hyperlink* yang mengarah ke halaman "home" dan "users". *Hyperlink* ini berasal dari *view* yang kita buat bernama "header", dan kita menggunakan *include* untuk menampilkan *view* "header" di kedua halaman tersebut.

Kita juga dapat melakukan pengiriman data ke dalam *include* dengan cara menambahkan *array* sebagai argumen kedua di dalam perintah `@include`. Sebagai contoh, jika kita ingin menampilkan pesan saat mengakses route user detail, kita dapat menampilkan pesan "Data telah ditemukan" jika data tersedia, atau "Data tidak ditemukan" jika data tidak tersedia.

```

<!-- resources/views/users/detail.blade.php -->

<html>
  <head>
    <title>
      @if($user != null)
        {{ $user["nama"] }}
      @endif
    </title>
  </head>
  <body>
    @php
      $message = "User telah ditemukan";
      if($user == null){
        $message = "User tidak ditemukan";
      }
    @endphp

    @include("header", [
      "message" => $message
    ])

    @if($user != null)
      <h1>Hello, saya {{ $user["nama"] }}</h1>
      <h2>Umurku {{ $user["umur"] }}</h2>
    @endif
  </body>
</html>

```

Mengirim Data Ke Dalam *View Header* Dengan *Include*.

Kita menggunakan perintah *isset* dalam *blade* untuk memastikan apakah variable *message* pernah didefinisikan sebelumnya atau belum, jika sudah maka *code* didalamnya akan dijalankan, jika belum tidak akan menyebabkan *error* sehingga memberikan data ke dalam *view* ini bersifat tidak wajib.

```

// resources/views/user/userDetail.blade.php

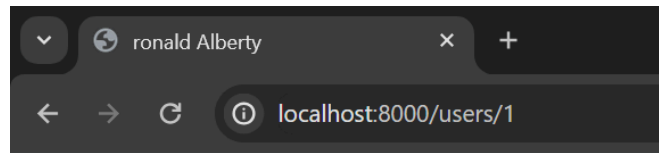
<ul>
  <li><a href="/">home</a></li>
  <li><a href="/users">users</a></li>
</ul>

@isset($message)
  <h2>{{ $message }}</h2>
@endisset

```

Penggunaan Data Yang Dikirim Melalui *Include*.

Ketika kita jalankan di browser maka akan terlihat seperti ini.



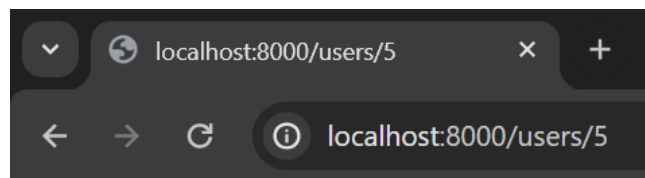
- [home](#)
- [users](#)

User telah ditemukan

Helloo, saya ronald Alberty

Umurku 20

Tampilan Ketika User Ditemukan.



- [home](#)
- [users](#)

User tidak ditemukan

Tampilan Ketika User Tidak Ditemukan.

5) Layouts.

Beberapa aplikasi terkadang memiliki tampilan yang sama di beberapa halamannya, yang membedakan adalah isi kontennya, kita akan kesulitan untuk mengelola jika kita menulis ulang tata letak dari satu halaman ke halaman lainnya. Dengan *blade template* kita dapat membuat satu struktur yang dapat digunakan disemua halaman. Ada beberapa cara untuk membuat *layout* di dalam *view*.

A. Template Inheritance.

membuat layouts dengan template yang disediakan oleh blade adalah cara yang paling mudah dan umum digunakan, ada beberapa perintah yang dapat digunakan untuk memakai atau membuat layout :

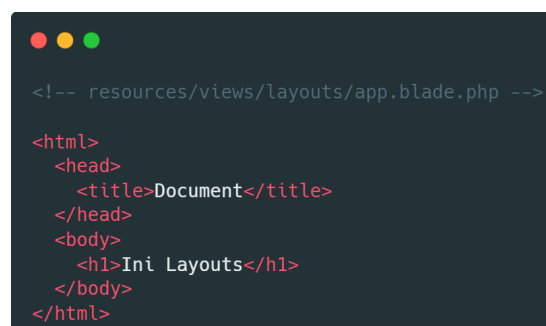
- `@extends` = Digunakan untuk menunjukkan bahwa sebuah halaman Blade akan meng-extend (mewarisi) layout utama yang sudah ditentukan sebelumnya.
- `@yield` = Digunakan dalam layout utama untuk menentukan tempat-tempat di mana konten dari halaman spesifik akan dimasukkan
- `@section` = Digunakan dalam halaman-halaman spesifik untuk menentukan konten spesifik yang akan dimasukkan ke dalam layout utama

Berikut adalah cara membuat layout dan cara menghubungkan dengan view yang lain.

Cara membuat layout dengan template Inheritance yaitu buat view dengan perintah artisan, dan perlu membuat view baru untuk dan akan mencoba menghubungkan setiap view dengan layout.

php artisan make:view layouts/app

view *app* yang ada di dalam layouts akan digunakan sebagai *layout* utama kita, jadi semua *view* akan mewarisi *layout* yang ada di *app*. kita buat struktur HTML di dalam *layout app* seperti ini.



```
<!-- resources/views/layouts/app.blade.php -->

<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>Ini Layouts</h1>
  </body>
</html>
```

Struktur HTML Di Dalam *Layouts*.

Lalu kita hubungkan *view index* didalam folder *landing* dengan *layout* yang sudah kita buat dengan perintah *extends* lalu didalamnya diisi dengan nama, seperti ini.


```
<!-- resources/view/index.blade.php -->

@extends("layouts.app")
```

Penggunaan *Extends* Di Dalam View *Index*.

Kita tambahkan *yield* di dalam *body* dan *title* dan di dalam *layout app* kita, fungsinya adalah agar konten setiap halaman bisa dinamis. Penambahan *yield* didalam *layout app* seperti ini.

```
<!-- resources/views/layouts/app.blade.php -->

<html>
  <head>
    <title>@yield("title")</title>
  </head>
  <body>
    <h1>header</h1>
    @yield("content")
    <h1>footer</h1>
  </body>
</html>
```

Penambahan *Yield* Di Dalam *Layout*.

Diatas *yield content* kita beri teks ini *header* dan dibawah *yield content* juga diberi teks ini *footer*. Lalu di dalam *view index* dan *users*, kita tambahkan dua *section* title dan content, lalu diisi dengan content yang akan ditampilkan seperti ini.

```
<!-- resources/views/index.blade.php -->

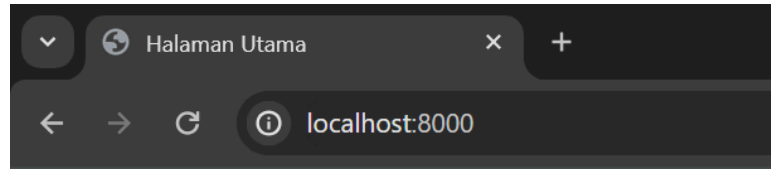
@extends("layouts.app")

@section("title", "Halaman Utama")

@section("content")
  <h1>Selamat datang di web kami</h1>
@endsection
```

Code Didalam View Index.

section digunakan untuk menyisipkan content yang akan ditampilkan di bagian section tertentu, contohnya pada section title kita gunakan untuk menentukan Ketika kita jalankan di browser maka akan terlihat seperti ini.



header

Selamat datang di web kami

footer

Ketika Akses *Route Index*.

6) Referensi

- <https://laravel.com/docs/master/blade>
- <https://laravel.com/docs/11.x/views>