



Laravel Blade Template

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow





Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com



Sebelum Belajar

- Laravel Dasar



Agenda

- Pengenalan Blade Template
- Menampilkan View
- If Statement
- For Loop
- While Loop
- Include
- Template Inheritance
- Dan lain-lain

Pengenalan Blade Template



Blade Template

- Blade adalah library yang terdapat di Laravel sebagai templating engine
- Tidak seperti template engine lainnya, Blade tidak melarang kita untuk menggunakan kode PHP di file template, walaupun ini sebenarnya tidak disarankan
- Semua file template di Blade sebenarnya akan di compile menjadi kode PHP
- Blade menggunakan file `.blade.php` sebagai nama file template nya
- Dan biasanya disimpan di folder `resources/views`

Membuat Project



Membuat Project

- `composer create-project laravel/laravel=version belajar-laravel-blade-template`
- <https://packagist.org/packages/laravel/laravel>

```
LARAVEL
→ LARAVEL composer create-project laravel/laravel=9.3.8 belajar-laravel-blade-template
Creating a "laravel/laravel=9.3.8" project at "./belajar-laravel-blade-template"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v9.3.8)
- Downloading laravel/laravel (v9.3.8)
- Installing laravel/laravel (v9.3.8): Extracting archive
Created project in /Users/khannedy/Developments/BELAJAR/LARAVEL/belajar-laravel-blade-template
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
```

Menampilkan View



Blade


- Membuat response dari Route itu sangat mudah, tapi jika misal kita ingin membuat response yang kompleks seperti HTML, maka akan sulit jika kita lakukan di Route
- Blade adalah fitur di Laravel yang digunakan untuk mempermudah dalam pembuatan tampilan halaman web HTML
- Dengan Blade template, kita bisa membedakan lokasi logic aplikasi, dengan kode tampilan
- Semua template disimpan di folder resources/views



Blade Variable

- Salah satu keuntungan menggunakan template dibanding kode PHP langsung adalah, kita bisa memaksa programmer untuk memisahkan logic kode program dengan tampilan (di template)
- Di Blade, walaupun kita bisa membuat kode PHP, tapi tidak disarankan menggunakan itu
- Cara yang direkomendasikan adalah, kita hanya membuat variable di template blade, lalu mengirim variable tersebut dari luar ketika akan menampilkan template nya
- Untuk membuat menampilkan variable di blade template, kita bisa gunakan {{ \$nama }}, dimana nanti variable \$nama bisa diambil secara otomatis dari data yang kita kirim ketika menampilkan view blade nya

Kode : Hello View



The image shows a code editor window with a single file named 'hello.blade.php'. The code is a Blade template for an HTML document. It includes a title tag with a placeholder for a variable named '\$name'. The body contains a single heading tag, also with a placeholder for '\$name'. The editor shows line numbers from 1 to 9. The status bar at the bottom indicates 'Uncommitted changes'.

```
1 <html>
2 <head>
3     <title>{{ $name }}</title>
4 </head>
5 <body>
6     <h1>{{ $name }}</h1>
7 </body>
8 </html>
9 You, 3 minutes ago • Uncommitted changes
```



Rendering View

- Setelah kita membuat View, selanjutnya untuk me-render (menampilkan) View tersebut di dalam Router, kita bisa menggunakan function `Route::view(uri, template, array)` atau menggunakan `view(template, array)` di dalam closure function Route / Controller
- Dimana template adalah nama template, tanpa menggunakan blade.php, dan array berisikan data variable yang ingin kita gunakan



Kode : Rendering View

```
19  
20 Route::get('/hello', function () {  
21     return view('hello', ['name' => 'Laravel']);  
22 });  
23
```



Test Rendering View

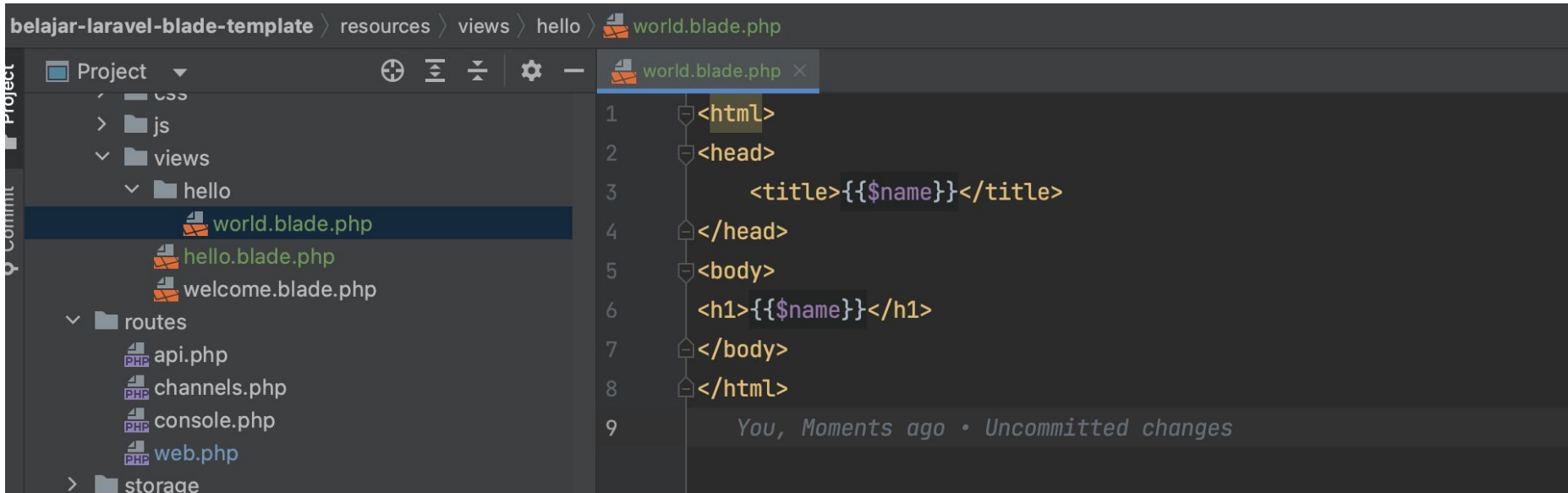
```
9 class HelloTest extends TestCase
10 {
11     public function testHello()
12     {
13         $this->get("/hello")
14         ->assertSeeText("Laravel");
15     }
16 }
```




Nested View Directory

- View juga bisa disimpan di dalam directory lagi di dalam directory views
- Hal ini baik ketika kita sudah banyak membuat views, dan ingin melakukan management file views
- Namun ketika kita ingin mengambil views nya, kita perlu ganti menjadi titik, tidak menggunakan / (slash)
- Misal jika kita buat views di folder admin/profile.blade.php, maka untuk mengaksesnya kita gunakan admin.profile

Kode : View



The screenshot shows an IDE interface with a project explorer on the left and a code editor on the right. The project explorer displays the file structure of a Laravel project, including folders for CSS, JS, and Views, and files for routes and storage. The code editor shows the content of the `world.blade.php` file, which is a Blade template. The template includes an HTML structure with a head section containing a title and a body section containing an h1 tag, both using Blade syntax for variable interpolation. A status bar at the bottom indicates that there are uncommitted changes.

```
belajar-laravel-blade-template > resources > views > hello > world.blade.php
```

Project Explorer:

- css
- js
- views
 - hello
 - world.blade.php
 - hello.blade.php
 - welcome.blade.php
- routes
 - api.php
 - channels.php
 - console.php
 - web.php
- storage

Code Editor:

```
1 <html>
2 <head>
3     <title>{{$name}}</title>
4 </head>
5 <body>
6     <h1>{{$name}}</h1>
7 </body>
8 </html>
9
```

Status Bar: You, Moments ago • Uncommitted changes



Kode : Route View

```
Route::get('/world', function () {  
    return view('hello.world', ["name" => "Laravel"]);  
});
```



```
public function testWorld()  
{  
    $this->get("/world")  
        ->assertSeeText("Laravel");  
}
```



Test View Tanpa Routing

- Kadang kita juga ingin membuat View tanpa routing, misal untuk mengirim email misalnya
- Pada kasus ini, kita bisa melakukan test view secara langsung, tanpa harus membuat Route terlebih dahulu



Kode : Test View Tanpa Route

```
public function testViewWithoutRouting()
{
    $this->view("hello", ["name" => "Laravel"])
        ->assertSeeText("Laravel");
}
```

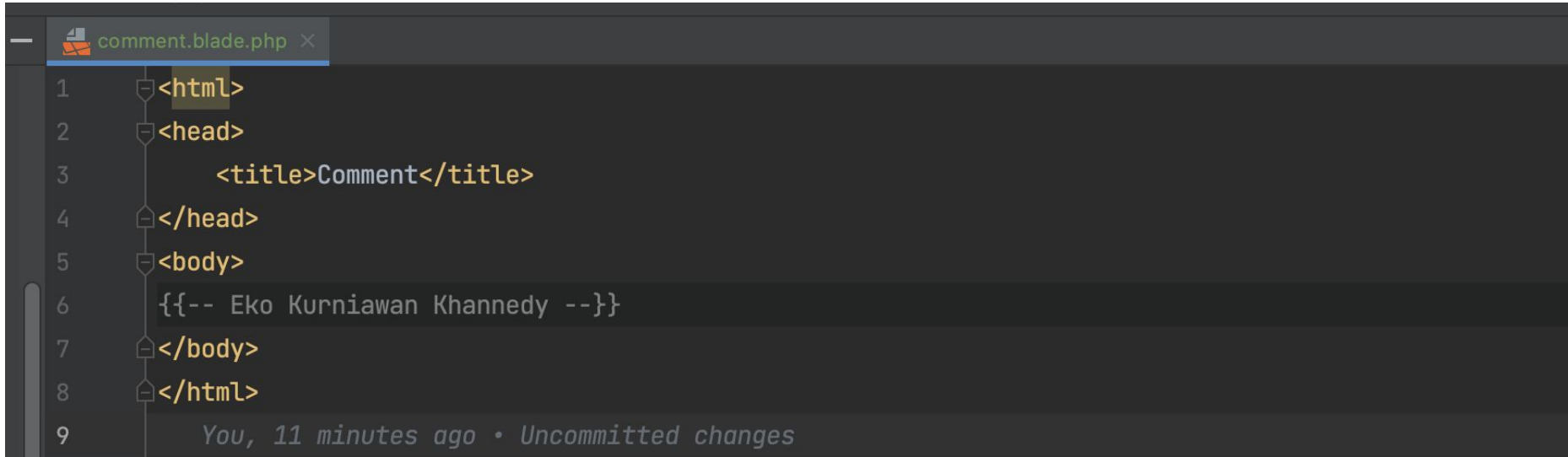
Comment



Comment

- Blade Template juga mendukung komentar, dengan menggunakan `{{-- isi komentar --}}`
- Isi komentar secara otomatis tidak akan dieksekusi, dan tidak akan ditampilkan juga di hasil HTML nya

Kode : Comment



```
comment.blade.php x
1 <html>
2 <head>
3     <title>Comment</title>
4 </head>
5 <body>
6     {{-- Eko Kurniawan Khannedy --}}
7 </body>
8 </html>
9 You, 11 minutes ago • Uncommitted changes
```




Kode : Unit Test Comment



```
public function testComment()
{
    $this->view("comment", [])
        ->assertSeeText("Comment")
        ->assertDontSeeText("Eko");
}
```

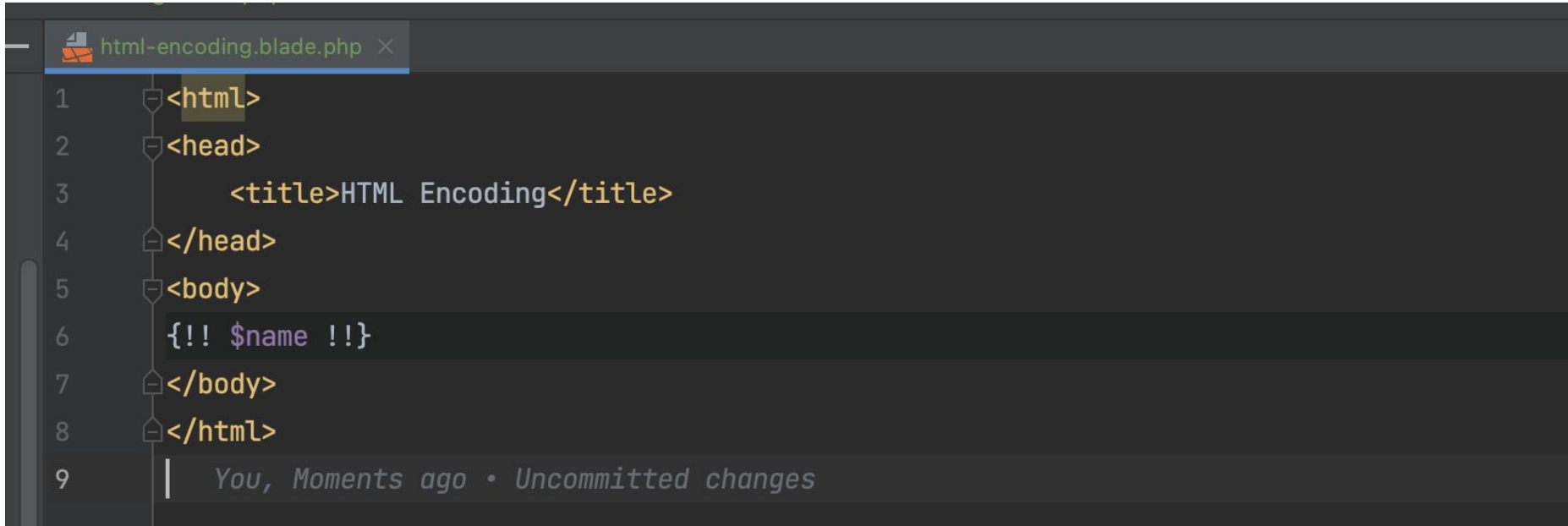
HTML Encoding



HTML Encoding

- Secara default, saat kita menampilkan data di Blade Template menggunakan `{{}}`, secara otomatis Blade akan memanggil function `htmlspecialchars(data)` untuk memastikan data aman dari tag HTML yang bisa menyebabkan XSS Attack (Cross Site Scripting)
- Namun jika kita ingin menampilkan tanpa di escape, maka kita bisa menggunakan `{!! $variable !!}`
- Namun harap hati-hati, karena jika tidak hati-hati, kita bisa terkena serangan XSS

Kode : HTML Encoding

A screenshot of a code editor window with a dark theme. The title bar shows a file named 'html-encoding.blade.php' with a close button. The code is written in PHP/Blade syntax and is as follows:

```
1 <html>
2 <head>
3     <title>HTML Encoding</title>
4 </head>
5 <body>
6     {!! $name !!}
7 </body>
8 </html>
9 | You, Moments ago • Uncommitted changes
```

The code is displayed on a dark background with syntax highlighting. Line numbers 1 through 9 are visible on the left. The code uses Blade's {!! !!} syntax for unescaped output. A status bar at the bottom indicates 'You, Moments ago • Uncommitted changes'.



Kode : Routing

```
Route::get('/html-encoding', function (\Illuminate\Http\Request $request) {  
    return view('html-encoding', ['name' => $request->input('name')]);  
});
```



Hasil HTML Encoding



← → ↻ 🏠 ⓘ localhost:8000/html-encoding?name=<h1>Eko</h1>

Eko

Disabled Blade



Disabled Blade

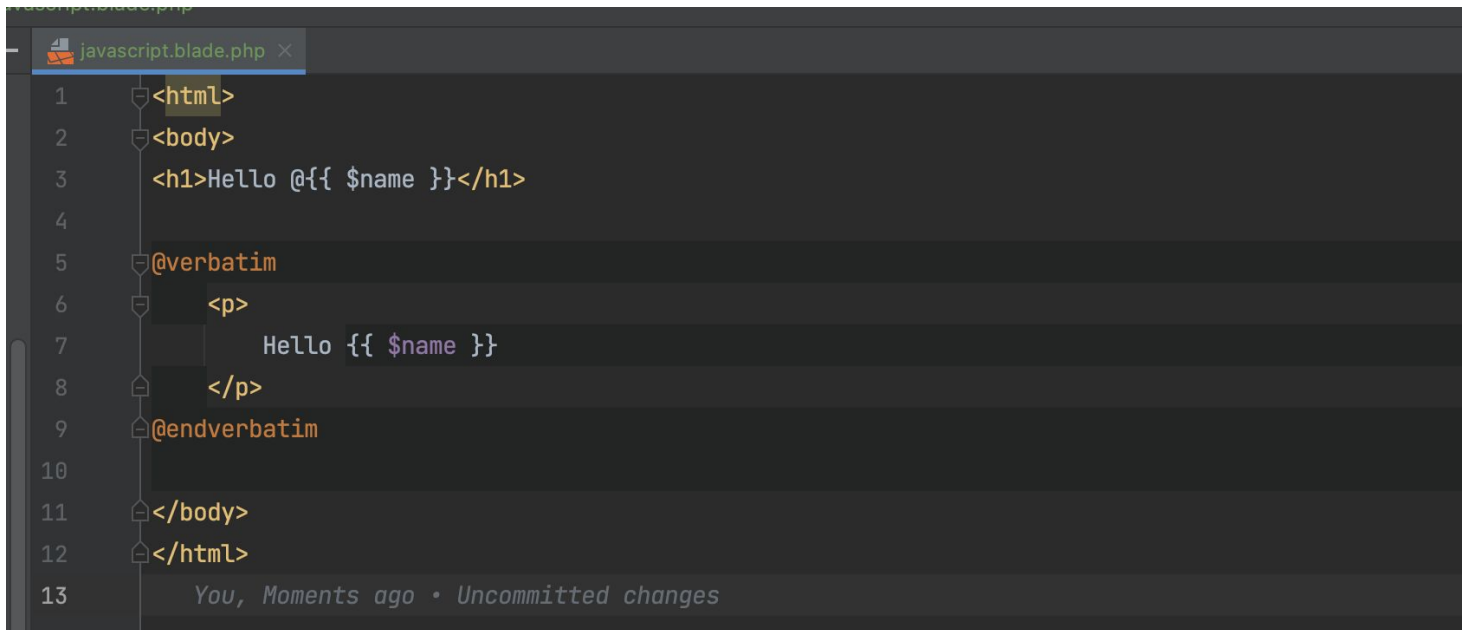
- Blade menggunakan tanda kurung kurawal untuk menampilkan variable
- Saat kita membuat web, kadang kita ingin menampilkan tulisan seperti perintah-perintah yang ada di Blade
- Hal ini akan menimbulkan perintah-perintah tersebut akan dieksekusi oleh Blade



Tanda @

- Pada bagian yang kita ingin Blade Template tidak dieksekusi, kita bisa menambahkan @ diawal perintah Blade Template nya
- Misalnya @{{ \$eko }}, maka akan ditampilkan apa adanya tanpa @
- Misalnya @@if, maka hasilnya akan ditampilkan apa adanya tanpa @ pertama
- Namun jika misal terdapat banyak baris yang kita harapkan tidak menggunakan Blade Template, kita bisa menggunakan perintah @verbatim dan diakhiri dengan @endverbatim

Kode : Disabled Blade



```
1 <html>
2 <body>
3   <h1>Hello @{{ $name }}</h1>
4
5   @verbatim
6     <p>
7       Hello {{ $name }}
8     </p>
9   @endverbatim
10
11 </body>
12 </html>
13 You, Moments ago • Uncommitted changes
```



Kode : Test Disabled Blade

```
public function testDisabledBlade()
{
    $this->view("disabled", ["name" => "Eko"])
        ->assertDontSeeText("Eko")
        ->assertSeeText('Hello {{ $name }}');
}
```

If Statement



If Statement

- Blade Template mendukung percabangan if menggunakan perintah/directive @if, @elseif, @else dan @endif
- Perintah ini hampir mirip dengan kode PHP

Kode : If Statement

```
if.blade.php x
1 <html>
2 <body>
3 <p>
4     @if (count($hobbies) == 1)
5         I have one hobby!
6     @elseif (count($hobbies) > 1)
7         I have multiple hobbies!
8     @else
9         I don't have any hobbies!
10    @endif
11 </p>
12 </body>
13 </html>
```

Kode : Test If Statement

```
public function testIfStatement()
{
    $this->view("if", ["hobbies" => []])
        ->assertSeeText("I don't have any hobbies!", false);

    $this->view("if", ["hobbies" => ["Coding"]])
        ->assertSeeText("I have one hobby!");

    $this->view("if", ["hobbies" => ["Coding", "Gaming"]])
        ->assertSeeText("I have multiple hobbies!");
}
```

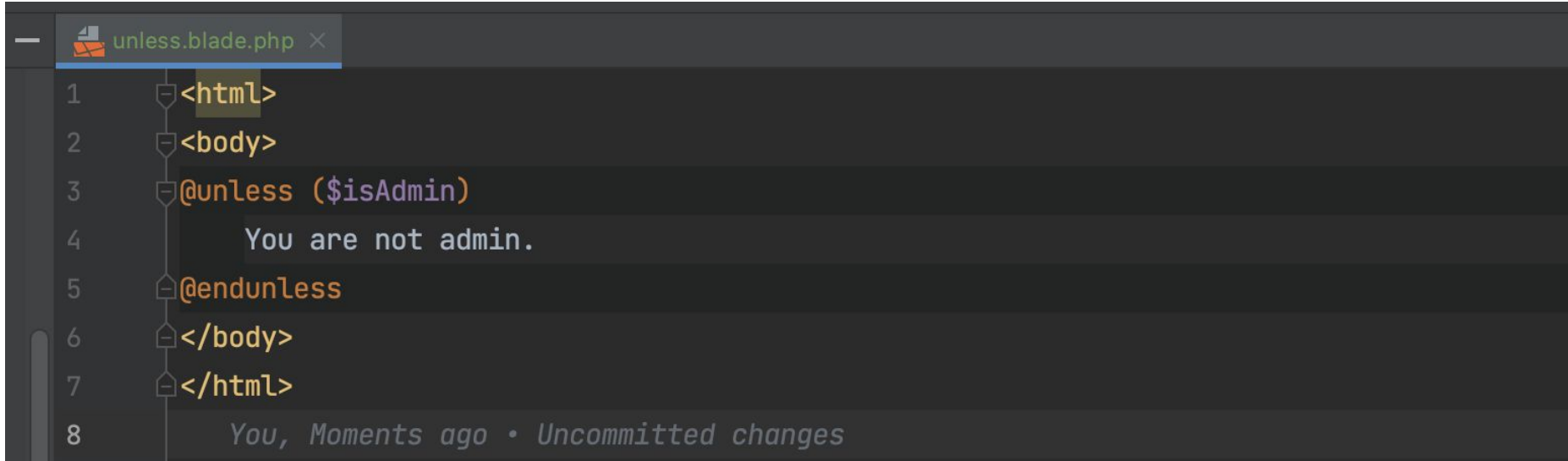
Unless Statement



Unless Statement

- Blade Template mendukung directive @unless dan @endunless
- Directive ini digunakan kebalikan dari if statement, dimana jika nilainya false, maka isi body akan eksekusi

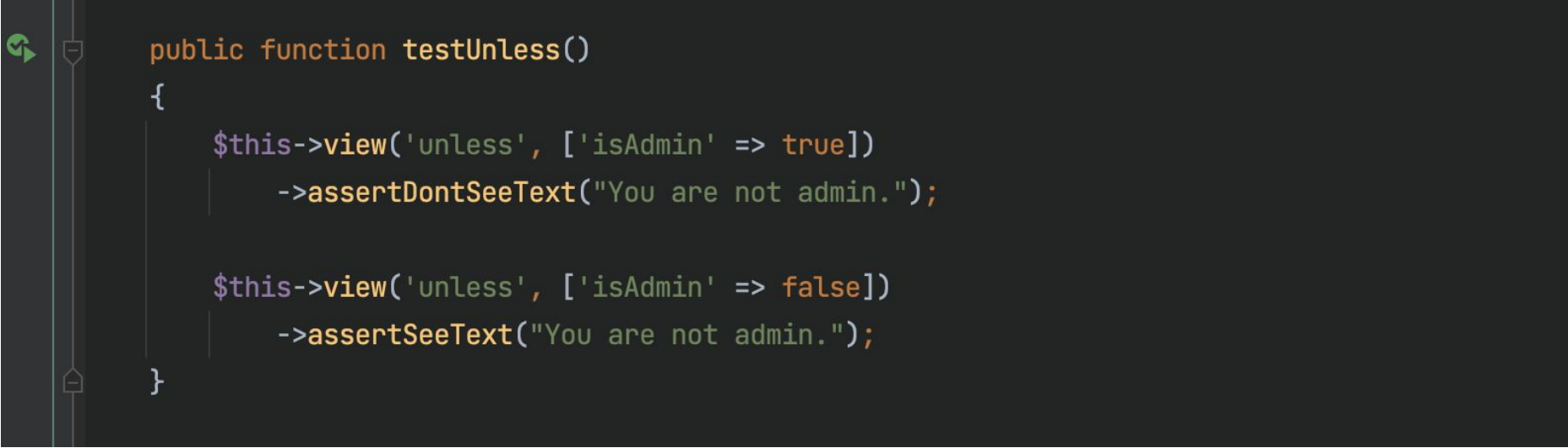
Kode : Unless



```
unless.blade.php x
1 <html>
2 <body>
3 @unless ($isAdmin)
4     You are not admin.
5 @endunless
6 </body>
7 </html>
8 You, Moments ago • Uncommitted changes
```



Kode : Test Unless



```
public function testUnless()
{
    $this->view('unless', ['isAdmin' => true])
        ->assertDontSeeText("You are not admin.");

    $this->view('unless', ['isAdmin' => false])
        ->assertSeeText("You are not admin.");
}
```

Isset dan Empty



Isset dan Empty

- Blade template juga memiliki directive @isset dan @empty
- @isset digunakan untuk mengecek apakah sebuah variable ada dan tidak bernilai null
- @empty digunakan untuk mengecek apakah sebuah variable merupakan array kosong

Kode : Isset dan Empty

```
issetempty.blade.php x
1 <html>
2 <body>
3 <p>
4     @isset($name)
5         Hello, my name is {{$name}}
6     @endisset
7 </p>
8 <p>
9     @empty($hobbies)
10        I don't have hobbies.
11    @endempty
12 </p>
13 </body>
14 </html>
```

Kode : Test Isset dan Empty

```
public function testIssetAndEmpty()
{
    $this->view("issetempty", [])
        ->assertDontSeeText("Hello");

    $this->view("issetempty", ["name" => "Eko"])
        ->assertSeeText("Hello, my name is Eko", false)
        ->assertSeeText("I don't have hobbies.", false);

    $this->view("issetempty", ["name" => "Eko", "hobbies" => ["Coding"]])
        ->assertSeeText("Hello, my name is Eko", false)
        ->assertDontSeeText("I don't have hobbies.", false);
}
```

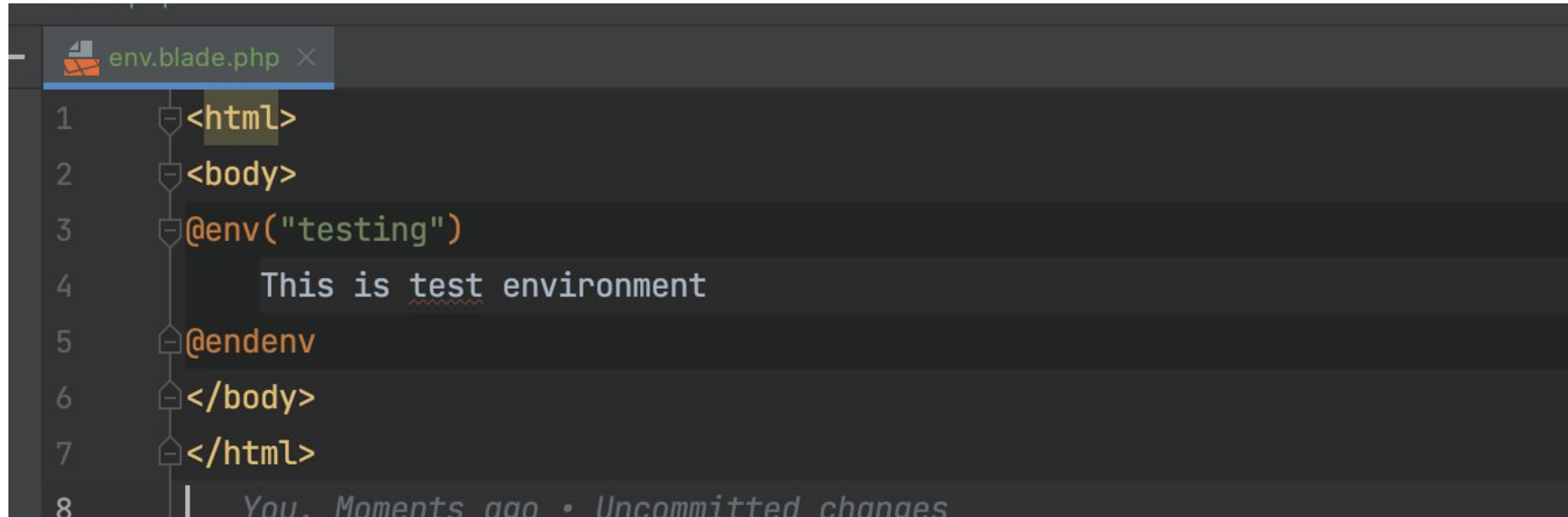
Env



Env

- Laravel mendukung multi environment, kita bisa menggunakan Facade Env untuk mendapatkan environment yang digunakan saat ini
- Dalam Blade Template, kita bisa menggunakan directive `@env(name)` atau `@env([name1, name2])` untuk mengecek apakah sedang menggunakan environment tersebut atau tidak

Kode : Env



```
1 <html>
2 <body>
3 @env("testing")
4     This is test environment
5 @endenv
6 </body>
7 </html>
```

8 You... Moments ago • Uncommitted changes



Kode : Unit Test Env

```
public function testEnv()  
{  
    $this->view("env", [])  
        ->assertSeeText("This is test environment");  
}
```

Switch Statement



Switch Statement

- Selain menggunakan `@if`, percabangan di Blade Template juga bisa menggunakan directive `@switch`
- Seperti di kode PHP, dalam Blade Template, kita perlu kombinasikan dengan `@case` dan juga `@break` dan `@default`

Kode : Switch

```
switch.blade.php x
1 <html>
2 <body>
3 @switch($value)
4     @case('A')
5         Memuaskan
6     @break
7     @case('B')
8         Bagus
9     @break
10    @case('C')
11        Cukup
12    @break
13    @default
14        Tidak Lulus
15 @endswitch
16 </body>
17 </html>
```

Kode : Test Switch

```
public function testSwitch()
{
    $this->view("switch", ["value" => "A"])
        ->assertSeeText("Memuaskan");

    $this->view("switch", ["value" => "B"])
        ->assertSeeText("Bagus");

    $this->view("switch", ["value" => "C"])
        ->assertSeeText("Cukup");

    $this->view("switch", ["value" => "D"])
        ->assertSeeText("Tidak Lolos");
}
```

For Loop



For Loop

- Blade juga mendukung perintah untuk melakukan perulangan for
- For loop mirip seperti di PHP, kita bisa menggunakan increment

Kode : For Loop

```
for.blade.php x
1  <html>
2  <body>
3  <ul>
4      @for($i = 0; $i < $limit; $i++)
5          <li>{{ $i }}</li>
6      @endfor
7  </ul>
8  </body>
9  </html>
10 | You, Moments ago • Uncommitted changes
```



Kode : Test For Loop

```
public function testFor()
{
    $this->view("for", ["limit" => 10])
        ->assertSeeText("0")
        ->assertSeeText("5")
        ->assertSeeText("9");
}
```



Foreach Loop

- Kadang ada kalanya kita ingin melakukan iterasi terhadap semua data yang terdapat di array
- Di Blade Template, kita bisa lebih mudah menggunakan directive `@foreach` dibandingkan menggunakan `@for`

Kode : Foreach Loop

```
foreach.blade.php x
1  <html>
2  <body>
3  <ul>
4      @foreach($hobbies as $hobby)
5          <li>{{ $hobby }}</li>
6      @endforeach
7  </ul>
8  </body>
9  </html>
10 You, Moments ago • Uncommitted changes
```



Kode : Test Foreach Loop



```
public function testForeach()
{
    $this->view("foreach", ["hobbies" => ["Coding", "Gaming"]])
        ->assertSeeText("Coding")
        ->assertSeeText("Gaming");
}
```



Forelse Loop

- Apa yang terjadi ketika misal kita melakukan iterasi menggunakan `@foreach`, namun ternyata datanya tidak ada?
- Secara otomatis tidak akan menampilkan data apapun
- Kadang kita ingin menampilkan sesuatu ketika tidak ada datanya
- Pada kasus ini, kita bisa menggunakan `@forelse`, dimana kita bisa menambahkan directive `@empty` ketika data array tidak ada isinya

Kode : Forelse Loop

```
forelse.blade.php
1  <html>
2  <body>
3  <ul>
4      @forelse($hobbies as $hobby)
5          <li>{{ $hobby }}</li>
6      @empty
7          <li>Tidak Punya Hobby</li>
8      @endforelse
9  </ul>
10 </body>
11 </html>
12 | You, Moments ago • Uncommitted changes
```


Kode : Test Forelse Loop

```
public function testForelse()  
{  
    $this->view("forelse", ["hobbies" => []])  
        ->assertSeeText("Tidak Punya Hobby");  
  
    $this->view("forelse", ["hobbies" => ["Coding", "Gaming"]])  
        ->assertSeeText("Coding")  
        ->assertSeeText("Gaming");  
}
```

Raw PHP



Raw PHP

- Kadang ada kasus kita ingin menggunakan kode PHP langsung di Blade Template
- Walaupun ini tidak direkomendasikan, tapi bisa kita lakukan
- Kita bisa gunakan directive @php untuk memasukkan kode php

Kode : Raw PHP

```
1 <html>
2 <body>
3 @php
4     class Person {
5         public string $name;
6         public string $address;
7     }
8
9     $person = new Person();
10    $person->name = "Eko";
11    $person->address = "Indonesia";
12 @endphp
13
14 <p>{{ $person->name }}</p>
15 <p>{{ $person->address }}</p>
16
17 </body>
```



Kode : Tes Raw PHP

```
public function testRawPHP()
{
    $this->view("php", [])
        ->assertSeeText("Eko")
        ->assertSeeText("Indonesia");
}
```

While Loop



While Loop

- Selain perulangan for, di Blade Template juga mendukung perulangan @while
- Dengan perulangan @while, kita bisa melakukan perulangan selama kondisi @while bernilai true

Kode : While Loop

```
while.blade.php x
1 <html>
2 <body>
3 @while($i < 10)
4     The current value is {{ $i }}
5     @php
6         $i++;
7     @endphp
8 @endwhile
9 </body>
10 </html>
11 You, 5 minutes ago • Uncommitted changes
```




Kode : Test While Loop

```
30 public function testWhile()  
31 {  
32     $this->view("while", ["i" => 0])  
33     ->assertSeeText("The current value is 0")  
34     ->assertSeeText("The current value is 5")  
35     ->assertSeeText("The current value is 9");  
36 }  
37  
38
```

Loop Variable



Loop Variable

- Saat kita menggunakan perulangan @foreach, terdapat variable \$loop yang tersedia di dalam perulangannya
- Variable \$loop berisikan informasi yang bermanfaat seperti index iterasi, apakah ini iterasi pertama atau terakhir, dan lain-lain



Loop Property

Property	Description
<code>\$loop->index</code>	The index of the current loop iteration (starts at 0).
<code>\$loop->iteration</code>	The current loop iteration (starts at 1).
<code>\$loop->remaining</code>	The iterations remaining in the loop.
<code>\$loop->count</code>	The total number of items in the array being iterated.
<code>\$loop->first</code>	Whether this is the first iteration through the loop.
<code>\$loop->last</code>	Whether this is the last iteration through the loop.
<code>\$loop->even</code>	Whether this is an even iteration through the loop.
<code>\$loop->odd</code>	Whether this is an odd iteration through the loop.
<code>\$loop->depth</code>	The nesting level of the current loop.
<code>\$loop->parent</code>	When in a nested loop, the parent's loop variable.

Kode : Loop Variable

```
loopvariable.blade.php x
1 <html>
2 <body>
3 <ul>
4     @foreach($hobbies as $hobby)
5         <li>{{ $loop->iteration }}. {{ $hobby }}</li>
6     @endforeach
7 </ul>
8 </body>
9 </html>
10 You, A minute ago • Uncommitted changes
```



Kode : Test Loop Variable

```
7  
8  public function testLoopVariable(){  
9      $this->view("loopvariable", ["hobbies" => ["Coding", "Gaming"]])  
10         ->assertSeeText("1. Coding")  
11         ->assertSeeText("2. Gaming");  
12     }  
13
```

CSS Class



CSS Class

- Saat membuat web, kadang kita ingin mengubah-ubah CSS class berdasarkan response backend
- Blade Template menyediakan @class directive untuk mempermudah kita melakukannya

Kode : CSS Class

```
class.blade.php x
1 <html>
2 <head>
3   <style>
4     .red {
5       color: red;
6     }
7
8     .bold {
9       font-weight: bold;
10    }
11  </style>
12 </head>
13 <ul>
14   @foreach($hobbies as $hobby)
15     <li @class(['red', 'bold' => $hobby['love']])>{{ $hobby['name'] }}</li>
16   @endforeach
17 </ul>
18 </html>
```

Kode : Test CSS Class

```
public function testClass()
{
    $this->view("class", ["hobbies" => [
        [
            "name" => "Coding",
            "love" => true
        ],
        [
            "name" => "Gaming",
            "love" => false
        ]
    ])

    ->assertSee('<li class="red bold">Coding</li>', false)
    ->assertSee('<li class="red">Gaming</li>', false);
}
```

Include



Include

- Saat kita membuat halaman web, kadang ada beberapa bagian yang sama di beberapa halaman web
- Pada kasus ini, ada baiknya kita membuat kode web tersebut di file template yang berbeda
- Selanjutnya kita bisa menggunakan directive `@include` untuk mengambil file template tersebut

Kode : Header Template

```
header.blade.php x
1 @isset($title)
2     <h1>{{$title}}</h1>
3 @else
4     <h1>Programmer Zaman Now</h1>
5 @endisset
6 | You, Moments ago • Uncommitted changes
```

Kode : Include



```
1 <html>
2 <body>
3     @include('header')
4     <p>Selamat Datang di Web</p>
5 </body>
6 </html>
```

Kode : Test Include

```
public function testLayout()
{
    $this->view("include", [])
        ->assertSeeText("Programmer Zaman Now")
        ->assertSeeText("Selamat Datang di Web");

    $this->view("include", ["title" => "Eko"])
        ->assertSeeText("Eko")
        ->assertSeeText("Selamat Datang di Web");
}
```



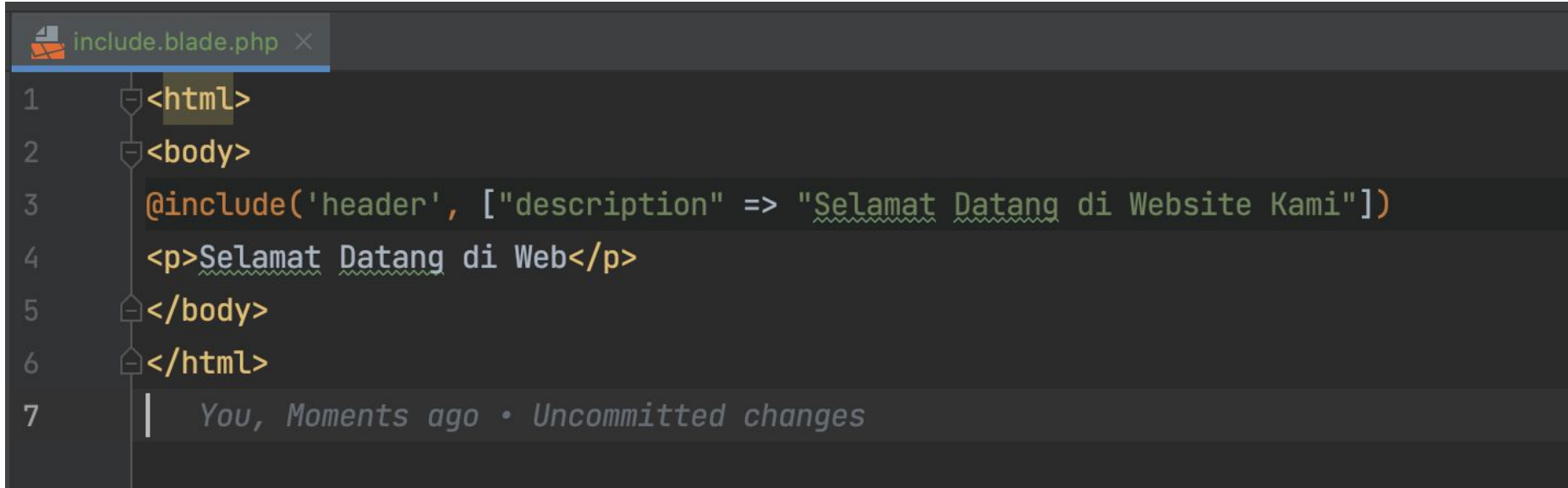
Include Parameter

- Secara default, semua data yang dikirim ke template utama akan dikirim ke template yang kita @include
- Namun, jika kita ingin menambah parameter tambahan, kita juga bisa menambahnya ketika menggunakan directive @include(template, data)

Kode : Header Template

```
header.blade.php x
1  @isset($title)
2      <h1>{{$title}}</h1>
3  @else
4      <h1>Programmer Zaman Now</h1>
5  @endisset
6
7  @isset($description)
8      <p>{{$description}}</p>
9  @endisset
10  You, Moments ago • Uncommitted changes
```

Kode : Include



```
1 <html>
2 <body>
3     @include('header', ["description" => "Selamat Datang di Website Kami"])
4     <p>Selamat Datang di Web</p>
5 </body>
6 </html>
7 | You, Moments ago • Uncommitted changes
```

Kode : Test Include

```
public function testLayout()
{
    $this->view("include", [])
        ->assertSeeText("Programmer Zaman Now")
        ->assertSeeText("Selamat Datang di Website Kami")
        ->assertSeeText("Selamat Datang di Web");

    $this->view("include", ["title" => "Eko"])
        ->assertSeeText("Eko")
        ->assertSeeText("Selamat Datang di Website Kami")
        ->assertSeeText("Selamat Datang di Web");
}
```

Include Condition



Include Condition

- Kadang terdapat kondisi kita ingin melakukan @include pada kondisi tertentu
- Maka biasanya kita akan menggunakan directive @if untuk melakukan pengecekan kondisi tersebut
- Blade Template memiliki directive lain untuk mempermudah kita ketika membutuhkan kondisi tertentu untuk melakukan @include



Include Condition Directive

Directive	Keterangan
@includeWhen(kondisi, template, data)	Include dilakukan ketika kondisi true
@includeUnless(kondisi, template, data)	Include dilakukan ketika kondisi false

Kode : Layout

header-admin.blade.php



header-admin.blade.php x

```
1 <h1>Selamat Datang Owner</h1>
2 <p>Untuk melihat laporan penjualan, silahkan buka menu laporan</p>
3 You, Moments ago • Uncommitted changes
```

Kode : Include Condition

```
include-condition.blade.php x
1  <html>
2  <body>
3      @includeWhen($user['owner'], 'header-admin')
4      <p>Selamat Datang {{ $user['name'] }}</p>
5  </body>
6  </html>
7  | You, Moments ago • Uncommitted changes
```




Kode : Test Include Condition

```
public function testIncludeCondition()
{
    $this->view("include-condition", ["user" => [
        "name" => "Eko",
        "owner" => true
    ]])
    ->assertSeeText("Selamat Datang Owner")
    ->assertSeeText("Selamat Datang Eko");

    $this->view("include-condition", ["user" => [
        "name" => "Eko",
        "owner" => false
    ]])
    ->assertDontSeeText("Selamat Datang Owner")
    ->assertSeeText("Selamat Datang Eko");
}
```

Each dan Once



Each dan Once

- Pada kasus tertentu, kadang kita ingin menampilkan layout sesuai dengan jumlah data
- Lalu misal kita perlu menambahkan sebuah kode yang cukup sekali saja ditampilkan, walaupun layout ditampilkan lebih dari sekali, misal kode CSS atau JavaScript
- Kita bisa menggunakan directive `@each` dan `@once`



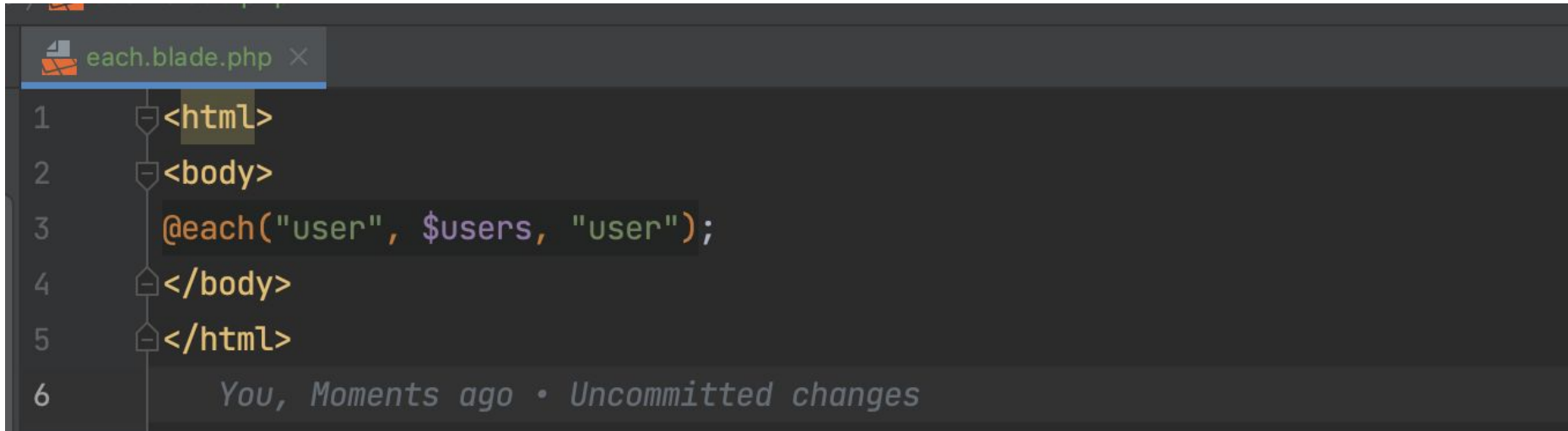
Each dan Once Directive

Directive	Keterangan
@each(template, array, variable)	Include template berkali-kali sesuai iterasi jumlah array, dimana data tiap array bisa diakses dengan variable di template nya
@once	Disimpan di layout, secara otomatis jika layout tersebut ditampilkan berkali-kali, bagian @once hanya ditampilkan sekali saja

Kode : Layout

```
user.blade.php x
1  @once
2      <style>
3          .red {
4              color: red;
5          }
6      </style>
7  @endonce
8
9  <h1>{{ $user['name'] }}</h1>
10 <ul>
11     @foreach($user['hobbies'] as $hobby)
12         <li class="red">{{ $hobby }}</li>
13     @endforeach
14 </ul>
15 You Moments ago • Uncommitted changes
```

Kode : Each



The screenshot shows a code editor with a dark theme. The active file is named 'each.blade.php'. The code is as follows:

```
1 <html>
2 <body>
3     @each("user", $users, "user");
4 </body>
5 </html>
```

Line numbers 1 through 6 are visible on the left. At the bottom of the editor, a status bar indicates 'You, Moments ago • Uncommitted changes'.

Kode : Test Each

```
public function testEach()
{
    $this->view("each", ["users" => [
        [
            "name" => "Eko",
            "hobbies" => ["Coding", "Gaming"]
        ],
        [
            "name" => "Kurniawan",
            "hobbies" => ["Coding", "Gaming"]
        ]
    ]])
    ->assertSeeInOrder([".red", "Eko", "Coding", "Gaming", "Kurniawan", "Coding", "Gaming"]);
}
```

Form



Form Directive

- Blade Template memiliki beberapa directive untuk membantu kita mempermudah ketika kita membuat form
- Ada `@checked(kondisi)` `@selected(kondisi)` `@disabled(kondisi)` `@readonly(kondisi)` dan `@required(kondisi)`
- Kondisi pada form directive merupakan boolean, jika true, maka secara otomatis directive tersebut akan dijalankan
- Penggunaan form directive ini lebih mudah dibanding kita menggunakan directive `@if` secara manual



Detail Form Directive

Directive	Jika kondisi true
@checked	Maka input checkbox akan memiliki attribute checked
@selected	Maka input option akan memiliki attribute selected
@disabled	Maka input akan memiliki attribute disabled
@readonly	Maka input akan memiliki attribute readonly
@required	Maka input akan memiliki attributed required

Kode : Form Directive

```
form.blade.php x
1 <html>
2 <body>
3 <form>
4     <input type="checkbox" @checked($user['premium']) value="Premium"/> <br/>
5     <input type="text" value="{{ $user['name'] }}" @readonly(! $user['admin'])/> <br>
6 </form>
7 </body>
8 </html>
9 | You, Moments ago • Uncommitted changes
```

Kode : Test Form Directive

```
public function testForm()
{
    $this->view("form", ["user" => [
        "premium" => true,
        "name" => "Eko",
        "admin" => true
    ]])
    ->assertSee("checked")
    ->assertSee("Eko")
    ->assertDontSee("readonly");
}
```

CSRF



CSRF

- Materi CSRF (Cross Site Request Forgery) sudah kita bahas di kelas Laravel Dasar
- Secara default, saat kita mengirim HTTP Post ke aplikasi Laravel kita, Laravel akan mengecek token CSRF, untuk memastikan bahwa request tersebut benar berasal dari web kita
- Pengecekan ini dilakukan oleh VerifyCsrfToken Middleware
- Blade Template memiliki directive @csrf yang bisa digunakan untuk mempermudah kita ketika ingin menambahkan token CSRF di form kita

Kode : CSRF

```
csrf.blade.php x
1 <html>
2 <body>
3 <form action="/test" method="post">
4     @csrf
5     <input type="text" name="name">
6     <input type="submit" value="Send">
7 </form>
8 </body>
9 </html>
10 You, Moments ago • Uncommitted changes
```



Kode : Test CSRF

```
public function testCSRF()
{
    $this->view("csrf", [])
        ->assertSee("_token");
}
```


—

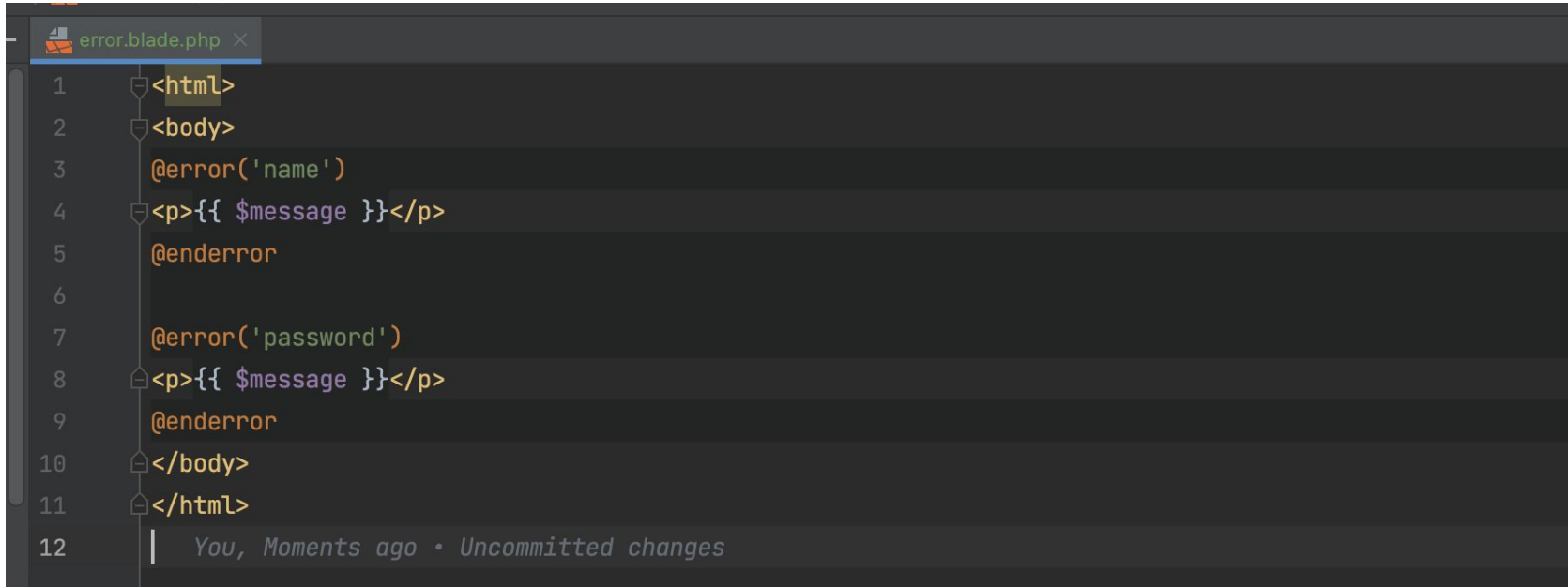
Error



Error

- Laravel memiliki fitur yang bernama Validation, dimana kita bisa dengan mudah melakukan validasi
- Laravel Validation akan dibahas di materi tersendiri
- Ketika terjadi validation error, kita bisa menangkap error dengan directive `@error(field)` dan menangkap message nya dengan variable `$message` di dalam directive `@error` nya

Kode : Error



```
error.blade.php x
1 <html>
2 <body>
3     @error('name')
4     <p>{{ $message }}</p>
5     @enderror
6
7     @error('password')
8     <p>{{ $message }}</p>
9     @enderror
10 </body>
11 </html>
12 | You, Moments ago • Uncommitted changes
```



Kode : Test Error

```
public function testValidationErrors()
{
    $errors = [
        "name" => "Name is required",
        "password" => "Password is required"
    ];

    $this->withErrors($errors)
        ->view("error", [])
        ->assertSeeText("Name is required")
        ->assertSeeText("Password is required");
}
```

—

Stack



Stack

- Blade memiliki kemampuan mirip struktur data Stack / Tumpukan
- Dimana kita bisa mengirim data ke stack tersebut menggunakan directive `@push(nama)` atau `@pushIf(kondisi, nama)`
- Ketika kita ingin menampilkan semua data yang terdapat di Stack, kita bisa gunakan directive `@stack(nama)`
- Secara default `@push()` akan mengirim data ke posisi paling belakang, jika kita ingin mengirim data ke urutan paling awal, kita bisa gunakan directive `@prepend(name)`

Kode : Stack

```
stack.blade.php x
1 <html>
2 <body>
3 @push("script")
4     <script src="first.js"/>
5 @endpush
6 @push("script")
7     <script src="second.js"/>
8 @endpush
9 @prepend("script")
10    <script src="third.js"/>
11 @endprepend
12
13 @stack("script")
14 </body>
15 </html>
16 | You, Moments ago • Uncommitted changes
```



Kode : Test Stack

```
public function testStack()
{
    $this->view("stack", [])
        ->assertSeeInOrder(["third.js", "first.js", "second.js"]);
}
```

Template Inheritance



Template Inheritance

- Sebelumnya, kita menggunakan @include untuk menggunakan template lain
- Blade mendukung Template Inheritance, caranya terbalik dengan @include
- Dalam @include, parent akan melakukan include ke template child nya
- Dalam Template Inheritance, child yang akan melakukan extends ke parent template, dan child yang akan menentukan isi dari parent nya



Parent Layout

- Di parent layout, kita bisa menggunakan `@yield(name, default)` untuk lokasi template yang harus dibuat nanti di child layout

Kode : Parent Layout

```
parent.blade.php x
1 <html>
2 <head>
3     <title>Nama Aplikasi - @yield('title')</title>
4 </head>
5     @yield("header")
6     @yield("content")
7 </html>
8 You, Moments ago • Uncommitted changes
```



Child Layout

- Untuk membuat child layout, kita bisa menggunakan directive `@extends(parentlayout)`
- Di dalam child layout, kita bisa membuat `@section(name)` yang nanti akan dieksekusi di posisi `@yield(name)` di parent nya

Kode : Child Layout

```
child.blade.php x
1  @extends("parent")
2
3  @section("title", "Halaman Utama")
4
5  @section("header")
6      <p>Deskripsi Header</p>
7  @endsection
8
9  @section("content")
10     <p>Ini adalah halaman utama</p>
11 @endsection
12 You, A minute ago • Uncommitted changes
```

Kode : Test Template Inheritance

```
public function testParent()
{
    $this->view("child", [])
        ->assertSeeText("Nama Aplikasi - Halaman Utama")
        ->assertSeeText("Deskripsi Header")
        ->assertSeeText("Ini adalah halaman utama");
}
```



Show Directive

- Pada beberapa kasus, kadang kita ingin membuat default `@section(name)` di parent layout, namun tetap bisa di override di child layout
- Kita bisa menggunakan `@section(name)` di parent layout, namun ditutup dengan `@show`, bukan `@endsection`
- Jika di child layout kita membuat `@section(name)`, secara otomatis `@section(name)` di parent akan di override, namun jika kita tetap ingin memanggil `@section(name)` di parent, kita bisa gunakan directive `@parent` di child layout nya

Kode : Parent Layout

```
parent.blade.php x
1 <html>
2 <head>
3     <title>Nama Aplikasi - @yield('title')</title>
4 </head>
5 @section('header')
6     <h1>Default Header</h1>
7 @show
8 @section('content')
9     <p>Default Content</p>
10 @show
11 </html>
12 | You, Moments ago • Uncommitted changes
```

Kode : Child Layout

```
child.blade.php x
1  @extends("parent")
2
3  @section("title", "Halaman Utama")
4
5  @section("header")
6      @parent
7      <p>Deskripsi Header</p>
8  @endsection
9
10 @section("content")
11     <p>Ini adalah halaman utama</p>
12 @endsection
13  You, Moments ago • Uncommitted changes
```

Kode : Test Template Inheritance

```
public function testParent()
{
    $this->view("child", [])
        ->assertSeeText("Nama Aplikasi - Halaman Utama")
        ->assertSeeText("Default Header")
        ->assertSeeText("Deskripsi Header")
        ->assertSeeText("Ini adalah halaman utama")
        ->assertDontSeeText("Default Content");
}
```

Service Injection



Service Injection

- Blade Template juga mendukung directive `@inject(variable, service)` untuk mengambil object dari Service Container
- Secara otomatis data object akan di inject ke variable yang disebutkan di directive `@inject`



Kode : Membuat Service

```
namespace App\Services;

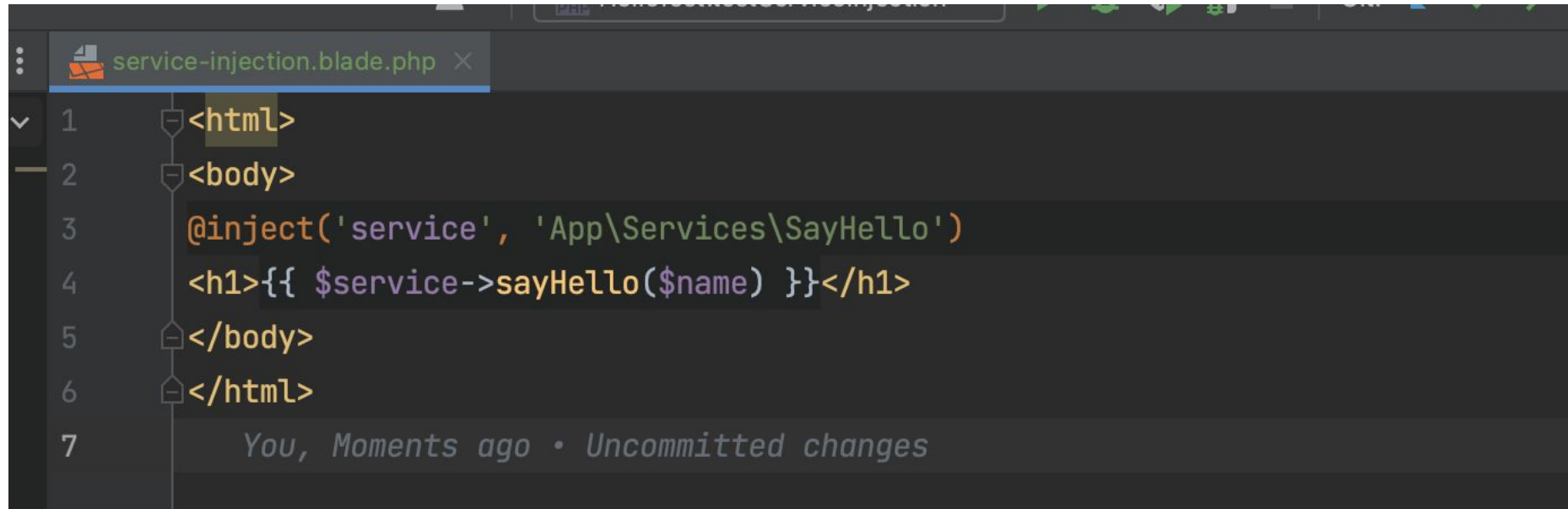
class SayHello
{
    function sayHello(string $name): string
    {
        return "Hello ${name}";
    }
}
```

You, Moments ago • Uncommitted changes

Kode : Registrasi ke AppServiceProvider

```
class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton(SayHello::class, function () {
            return new SayHello();
        });
    }
}
```

Kode : Service Injection



```
1 <html>
2 <body>
3     @inject('service', 'App\Services\SayHello')
4     <h1>{{ $service->sayHello($name) }}</h1>
5 </body>
6 </html>
7 You, Moments ago • Uncommitted changes
```

The image shows a code editor window with a file named 'service-injection.blade.php'. The code is a Blade template. Line 1: <html>. Line 2: <body>. Line 3: @inject('service', 'App\Services\SayHello'). Line 4: <h1>{{ \$service->sayHello(\$name) }}</h1>. Line 5: </body>. Line 6: </html>. Line 7: A status bar message 'You, Moments ago • Uncommitted changes'.



Kode : Test Service Injection

```
public function testServiceInjection()
{
    $this->view("service-injection", ["name" => "Eko"])
        ->assertSeeText("Hello Eko");
}
```

Inline Blade Template



Blade Facade

- Blade juga memiliki Facade yang bisa kita gunakan untuk menggunakan Blade Template
- Ini akan mempermudah kita ketika kita ingin mengakses Blade Template dari Class seperti contohnya dari Controller
- Ada banyak sekali fitur yang terdapat di Blade Facade yang bisa kita gunakan
- <https://laravel.com/api/9.x/Illuminate/Support/Facades/Blade.html>



Inline Blade Template

- Salah satu fitur yang bisa kita gunakan dengan Blade Facade adalah, Inline Template
- Inline Template adalah kemampuan dimana kita bisa me-render template tanpa harus membuat file template, cukup menggunakan string saja
- Kita bisa menggunakan method `Blade::render(template, data)`



Kode : Test Inline Blade Template

```
public function testInlineBladeTemplate()
{
    $response = Blade::render('Hello {{$name}}', ['name' => 'Eko']);
    self::assertEquals("Hello Eko", $response);
}
```

Extending Blade



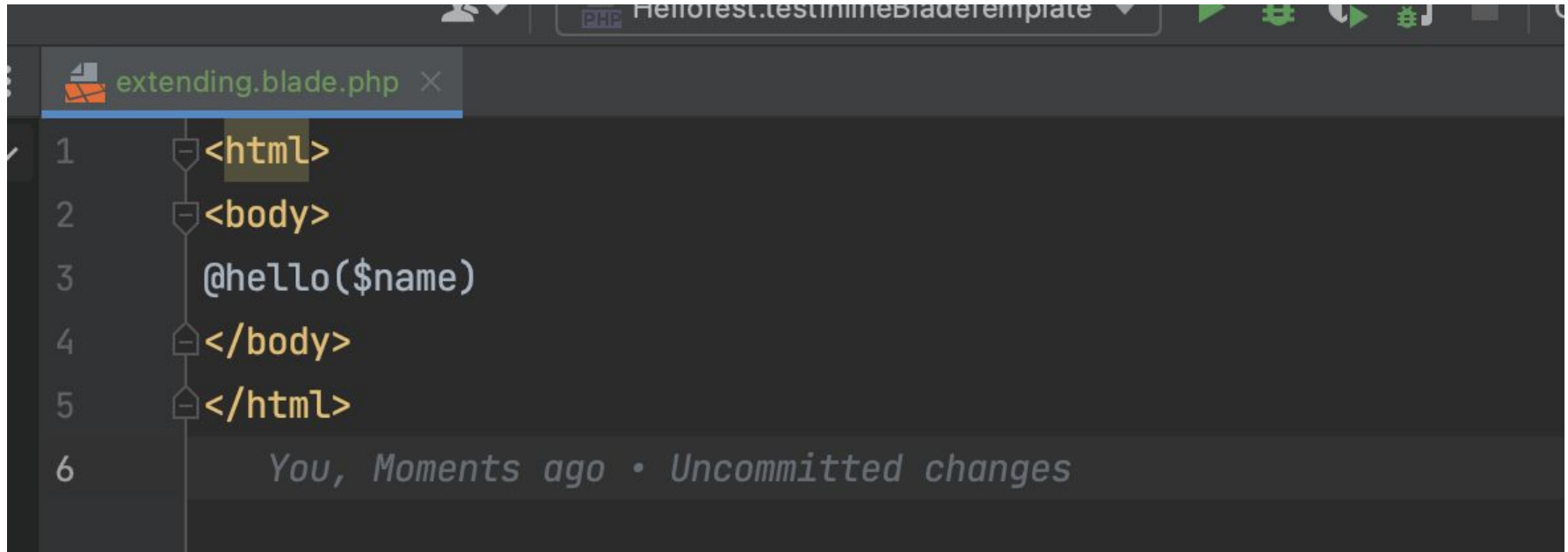
Extending Blade

- Blade Facade juga bisa kita gunakan untuk menambahkan custom directive
- Sehingga kita bisa membuat directive sendiri, dan bisa digunakan di Blade Template menggunakan method `Blade::directive(nama, function)`
- Agar aman, kita perlu meregistrasi directive di Service Provider

Kode : Extending Blade

```
23  /**
24  * Bootstrap any application services.
25  *
26  * @return void
27  */
28  public function boot()
29  {
30      Blade::directive('hello', function ($expression) {
31          return "<?php echo 'Hello ' . $expression; ?>";
32      });
33  }
34 }
```


Kode : Layout Extending Blade



The screenshot shows a code editor with a dark theme. The active file is 'extending.blade.php', which is a Blade template. The code is as follows:

```
1 <html>
2 <body>
3     @hello($name)
4 </body>
5 </html>
```

Line 6 of the editor shows a status bar message: 'You, Moments ago • Uncommitted changes'. The editor's interface includes a tab bar at the top with the file name and a close button, and a toolbar with various icons for editing and running code.



Kode : Test Extending Blade

```
public function testExtendingBlade()
{
    $this->view("extending", ["name" => "Eko"])
        ->assertSeeText("Hello Eko");
}
```

Custom Echo Handler



Custom Echo Handler

- Saat kita menggunakan `{{ $variable }}`, secara otomatis Blade Template akan mengkonversi menjadi `echo ($variable)-->__toString()`
- Pada kasus tertentu, mungkin kita ingin mengubah nya, misal ketika datanya berubah class tertentu, kita ingin mengubah hasil String nya
- Kita bisa menggunakan Facade Blade untuk mengubahnya, dengan cara menggunakan method `Blade::stringable(class, function)`



Kode : Class Model

```
namespace App\Models;
```

```
class Person
```

```
{
```

```
    public string $name;
```

```
    public string $address;
```

```
}
```

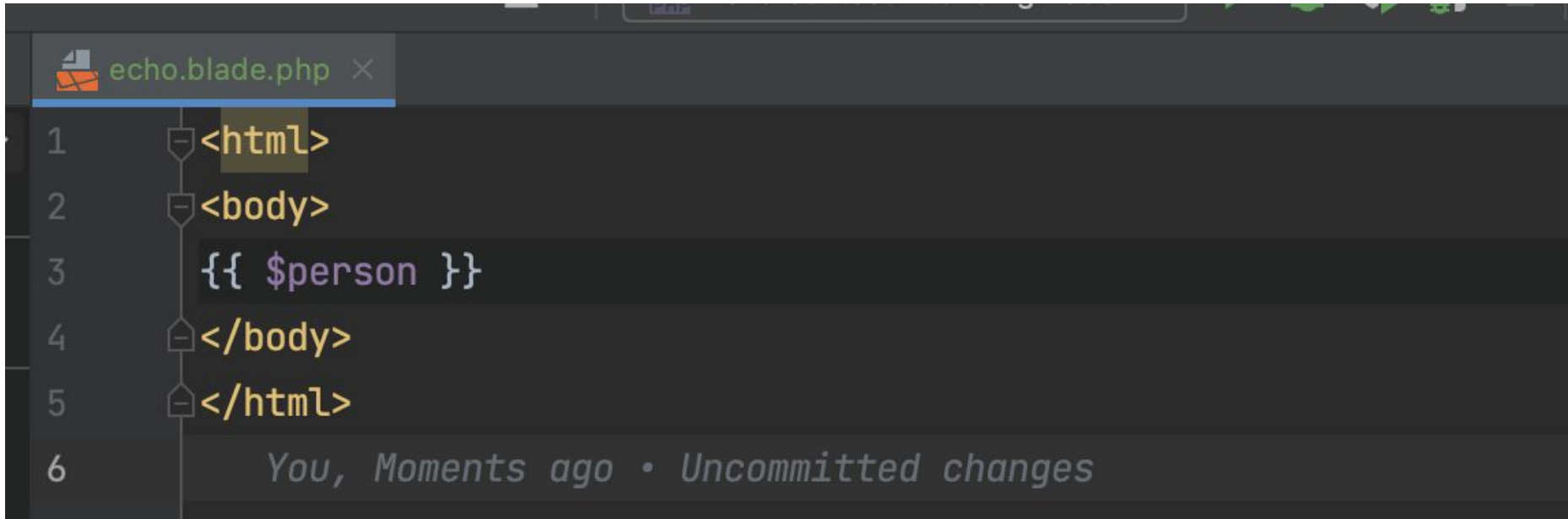
You, Moments ago • Uncommitted changes



Kode : Custom Echo Handler

```
/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    Blade::stringable(Person::class, function (Person $person) {
        return "$person->name : $person->address";
    });
}
```

Kode : Layout



```
1 <html>
2 <body>
3     {{ $person }}
4 </body>
5 </html>
6 You, Moments ago • Uncommitted changes
```

Kode : Test Custom Echo Handler

```
public function testEcho()
{
    $person = new Person();
    $person->name = "Eko";
    $person->address = "Indonesia";

    $this->view("echo", ["person" => $person])
        ->assertSeeText("Eko : Indonesia");
}
```

Optimize Template



Optimizing Template

- Secara default, Blade Template di compile menjadi kode PHP ketika ada request, Laravel akan mengecek apakah hasil compile Blade Template ada atau tidak, jika ada maka akan menggunakannya, jika tidak ada maka akan coba melakukan compile.
- Termasuk Laravel juga akan mendeteksi ketika ada perubahan Blade Template.
- Kompilasi ketika request masuk akan ada efek buruknya, yaitu performanya jadi lambat karena harus melakukan kompilasi. Oleh karena itu ketika nanti menjalankan aplikasi Laravel di production, ada baiknya melakukan proses kompilasi seluruh blade template terlebih dahulu, agar tidak perlu melakukan kompilasi lagi ketika request masuk



Compiling Template

- Untuk melakukan compile view atau blade template, kita bisa gunakan perintah :
php artisan view:cache
- Semua hasil compile view akan disimpan di folder storage/framework/views
- Jika kita ingin menghapus seluruh hasil compile, kita bisa gunakan perintah
php artisan view:clear

Materi Selanjutnya



Materi Selanjutnya

- Laravel Component
- Laravel Database / Eloquent
- Laravel Validation
- Laravel Command
- Laravel HTTP Client
- Dan lain-lain