

# PANDUAN Framework PHP

# Laravel

The PHP Framework for Web Artisans

## Panduan Laravel PHP Framework

Oleh Tim AirPutih ([info@airputih.or.id](mailto:info@airputih.or.id))

### Hak Cipta

Hak Cipta (c) 2014 dipegang oleh tim penulis, dan dipublikasikan berdasarkan lisensi Creative Commons Atribusi Non-Commercial, Share Alike:

<http://creativecommons.org/licenses/by-nc-sa/2.5/>

<http://creativecommons.org>

Anda bebas menyalin, menyebarluaskan, dan mengadaptasi tulisan ini dengan ketentuan tulisan hasil adaptasi dari tulisan ini harus menyebutkan nama penulis ini dan disebarluaskan dengan lisensi yang sama atau mirip dengan lisensi ini.



Sumber :

1. <http://laravel.com>
2. <https://wikipedia.org>

## DAFTAR ISI

### **Laravel PHP Framework**

Materi dan Konsep.....	1
Pengertian.....	1
Konsep MVC.....	1
Composer.....	2
Install composer.....	2
Fitur Laravel.....	2
Basic Routing.....	3
Request & Input.....	5
Views & Responses.....	10
Controllers.....	15
Forms dan HTML.....	19

### **Lab. Tutorial Laravel 4.2**

Kebutuhan.....	22
Instalasi laravel via composer di Ubuntu 14.04.....	22
Struktur Folder dan File Laravel.....	24
Membuat blog sederhana.....	25

# Laravel PHP Framework

## Materi dan Konsep

### Pengertian

Laravel adalah framework PHP dengan kode terbuka (*open source*) dengan desain MVC (*Model-View-Controller*) yang digunakan untuk membangun aplikasi website. Framework ini pertama kali dibangun oleh Taylor Otwell pada tanggal 22 Februari 2012.

### Konsep MVC

Model-View-Controller atau MVC adalah sebuah metode untuk membuat sebuah aplikasi dengan memisahkan data (Model) dari tampilan (View) dan cara bagaimana memprosesnya (Controller).

1. Model, Model mewakili struktur data. Biasanya model berisi fungsi-fungsi yang membantu seseorang dalam pengelolaan basis data seperti memasukkan data ke basis data, pembaruan data dan lain-lain.
2. View, View adalah bagian yang mengatur tampilan ke pengguna. Bisa dikatakan berupa halaman web.
3. Controller, Controller merupakan bagian yang menjembatani model dan view.

4. Controller berisi perintah-perintah yang berfungsi untuk memproses suatu data dan mengirimkannya ke halaman web.

## Composer

Composer adalah dependensi manajer aplikasi level untuk bahasa pemrograman PHP yang menyediakan format standar untuk mengelola dependensi software PHP dan *library* yang diperlukan. Composer ini dikembangkan oleh Nils Adermann dan Jordi Boggiano, rilis pertama kali pada tanggal 1 Maret 2012. Composer ini terinspirasi dari “npm”-nya Node.js dan “bundler”-nya Ruby. Composer digunakan melalui perintah command line.

### Install composer

- Linux dan Mac OSX

```
$ curl -sS https://getcomposer.org/installer | php
```

atau jika belum mempunyai paket curl

```
$ php -r "readfile('https://getcomposer.org/installer');" | php
```

agar dapat diakses secara global ketik perintah berikut

```
$ mv composer.phar /usr/local/bin/composer
```

- Windows

Download installer dari <https://getcomposer.org/Composer-Setup.exe> dan lakukan instalasi.

## Fitur Laravel

- *Bundles*, Bundel atau ikatan menyediakan sistem kemasan modular. Dengan fitur ini kita dapat dengan mudah untuk melakukan penambahan paket aplikasi ke dalam project kita. Laravel versi 4.x menggunakan *composer* sebagai manajer aplikasi.
- *Eloquent ORM (Object-Relational Mapping)*, merupakan implementasi PHP lanjutan dari *active record* yang menyediakan metode tersendiri dalam mengatur

*relationship* antar obyek di database. Laravel *query builder* adalah salah satu fitur yang disupport *Eloquent*.

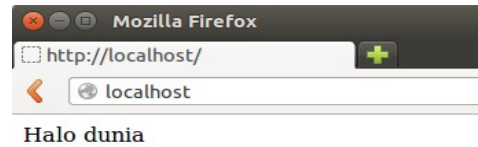
- *Application logic*, fitur pengembangan aplikasi secara umum, baik dengan *controller* atau pendeklarasian *route*
- *Reverse routing*, fitur yang mampu mendefinisikan hubungan antara *link* dan *route*, sehingga memungkinkan perubahan *link* dari *route* atau tanpa melakukan perubahan di *view*.
- *Restfull controllers*, merupakan cara opsional untuk memisahkan logika antara HTTP GET dan POST
- *Class auto loading*, menyediakan fitur untuk *load* PHP *class* tanpa perlu melakukan *include*, *On-demand loading* hanya akan me-load *class* yang diperlukan.
- *View composers*, merupakan kode *logic* yang dieksekusi ketika *view* di-load
- *IoC container*, memungkinkan obyek baru yang akan dihasilkan sesuai prinsip kontrol, dengan instansiasi opsional dan referensi dari obyek baru.
- *Migrations*, menyediakan sistem kontrol untuk skema database, sehingga memungkinkan untuk menghubungkan antara perubahan kode aplikasi dengan layout database, memudahkan *deploy* dan *update* aplikasi.
- *Unit testing*, menyediakan fitur testing untuk mendeteksi atau mencegah kode ganda atau berulang (regresi), unit test ini dapat dijalankan melalui perintah command line.
- *Automatic pagination*, fitur yang memungkinkan pembuatan halaman/*paging* secara otomatis dengan metode yang sudah diintegrasikan ke laravel.

## **Basic Routing**

Routing berfungsi untuk mengatur alur url dan mendefinisikan url tersebut akan memanggil controller mana atau mengeksekusi fungsi apa.

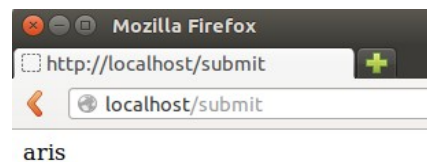
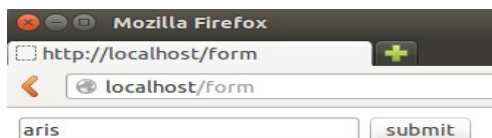
- **Basic GET route**

```
1 <?php
2
3 Route::get('/', function()
4 {
5     return "Halo dunia";
6 });
7
8
```



- **Basic POST route**

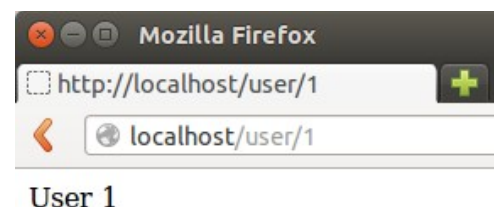
```
1 <?php
2
3 Route::get('form', function()
4 {
5     return "
6     <form action='submit' method='post'>
7         <input type='text' name='nama'>
8         <input type='submit' value='submit'>
9     </form>";
10 });
11
12 Route::post('submit', function()
13 {
14     return Input::get('nama');
15 });
16
17
```



Ketika button submit diklik maka halaman akan mengarah ke url /submit dan mengirimkan / post data yaitu nama.

- **Route Parameters**

```
1 <?php
2
3 Route::get('user/{id}', function($id)
4 {
5     return 'User '.$id;
6 });
7
8
```



Route '{id}' akan membaca karakter apapun yang terdapat di url setelah 'user/'

## Request & Input

- Basic input

```
1 <?php
2
3 Route::get('form', function()
4 {
5     return "
6     <form action='submit' method='post'>
7         <input type='text' name='nama'>
8         <input type='submit' value='submit'>
9     </form>";
10 });
11
12 Route::post('submit', function()
13 {
14     return Input::get('nama');
15 });
16
17
```

untuk mengakses semua input dari user, tanpa memerlukan HTTP verb seperti \$\_POST atau \$\_GET.

- Jika value kosong, bisa juga diset default value:

```
$nama = Input::get('nama', 'Aris');
```

artinya jika value nama kosong, variabel \$nama akan diisi 'Aris'.

- Untuk melakukan pengecekan terhadap input value dapat dilakukan seperti berikut:

```
if (Input::has('name'))
{
    //
}
```

artinya jika value name bernilai true atau tidak kosong, maka akan masuk ke kondisi if

- Mengambil semua input value

```
$input = Input::all();
```

akan menghasilkan variable \$input yang berbentuk array.

- Mengambil beberapa inputan saja



```
$input = Input::only('username', 'password');
```

```
$input = Input::except('password_confirmation');
```

kode yang pertama akan mengambil hanya value username dan password saja, sementara

kode yang kedua akan mengambil semua input value kecuali password\_confirmation.

- Mengambil value dari form dengan input berupa array

```
$input = Input::get('produk.nama');
```

akan mengambil input value dari form dengan name='produk[nama]'

- **Cookies**

Semua cookies di laravel dienkripsi dan ditandai dengan kode autentikasi, yang berarti cookie akan invalid jika diubah dari sisi client sehingga menjadi keamanan tersendiri bagi aplikasi web kita.

- Mengambil cookie value

```
$value = Cookie::get('name');
```

- Menambahkan cookie baru ke response

```
$response = Response::make('Halo Dunia');
```

```
$response->withCookie(Cookie::make('name', 'value',  
$minutes))
```

- Mengantri ke queue untuk response berikutnya

```
Cookie::queue($name, $value, $minutes)
```

- Membuat cookie tanpa expired

```
$cookie = Cookie::forever('name', 'value');
```

- **Old Input**

Digunakan menyimpan inputan dari halaman sebelumnya, misalnya untuk mengembalikan input value ketika validasi error.

- Menyimpan sementara (flash) input ke dalam session

```
//flash semua inputan
```

```
Input::flash();
```

```
//flash hanya username dan email
```

```
Input::flashOnly('username', 'email');
```

```
//flash semua inputan kecuali password
```

```
Input::flashExcept('password');
```

```
//membubuhkan langsung semua input ke Redirect
```

```
return Redirect::to('form')->withInput();
```

```
//membubuhkan langsung semua input kecuali password
```

```
return Redirect::to('form')
```

-

```
>withInput(Input::except('password'));
```

- Mengambil old data selain dari flash

```
Input::old('username')
```

- **Files**

Contoh kode untuk upload file

```
if(Input::hasFile('foto')) //mengecek value foto  
{
```

```
$file = Input::file('foto'); //semua berkas file
$filename = $file->getClientOriginalName(); //nama
file
$file->move(public_path().'/uploads/', $filename);
}
```

- Mengambil path file

```
$path = Input::file('foto')->getRealPath();
```

- Mengambil nama file asli

```
$filename = Input::file('foto')->
getClientOriginalName();
```

- Mengambil ekstensi atau format file

```
$extension = Input::file('foto')
->getClientOriginalExtension();
```

- Mengambil size file

```
$size = Input::file('foto')->getSize();
```

- Mengambil MIME type file

```
$mime = Input::file('foto')->getMimeType();
```

- **Request Information**

- Mengambil URI request

```
$uri = Request::path()
```

- Mengambil dan mengecek request method

```
//mengambil method request
```

```
$method = Request::method();
```

```
//mengecek method request  
  
if (Request::isMethod('post'))  
  
{  
  
    //  
  
}
```

- Menentukan apakah request sesuai pattern

```
if (Request::is('admin/*'))  
  
{  
  
    //  
  
}
```

- Mengambil request url dan URI Segment

```
//request url  
$url = Request::url();  
  
//uri segment  
$segment = Request::segment(1);
```

- Mengambil request header

```
$value = Request::header('Content-Type');
```

- Mengambil value dari \$\_SERVER

```
$value = Request::server('PATH_INFO')
```

- Menentukan apakah request berasal dari protokol HTTPS

```
$if (Request::secure())
```

```
{  
    //  
}
```

- Menentukan apakah request menggunakan AJAX

```
$if (Request::ajax())  
  
{  
  
//  
  
}
```

- Menentukan apakah request ber-type JSON

```
$if (Request::isJson())  
  
{  
  
//  
  
}
```

- Menentukan apakah request menginginkan type JSON

```
$if (Request::wantsJson())  
  
{  
  
//  
  
}
```

- Mengecek apakah request sesuai format yang diinginkan

```
$if (Request::format() == 'json')  
  
{  
  
//  
  
}
```

## Views & Responses

- **Basic Responses**

- Menghasilkan atau mengembalikan string dari routes

```
Route::get('/', function()  
{  
    return 'Halo dunia';  
});
```

- Membuat custom responses

```
$response = Response::make($contents, $statusCode);  
  
$response->header('Content-Type', $value);  
  
return $response;
```

- **Redirects**

Redirect digunakan untuk mengalihkan halaman. Redirect di laravel bermacam-macam jenisnya, bisa langsung ke route url, route name, dan juga langsung ke method dari sebuah controller

```
//redirect ke route url  
return Redirect::to('user/login');
```

```
//redirect ke route url dengan flash data  
return Redirect::to('user/login')->with('message',  
'Login Failed');
```

```
//redirect ke route name  
return Redirect::route('login');
```

```
//redirect ke route name dengan parameter
```

```
return Redirect::route('profile', array(1));

//redirect ke route name dengan nama parameter dan
value-nya
return Redirect::route('profile', array('user' =>
1));

//redirect ke method controller
return Redirect::action('HomeController@index');

//redirect ke method controller dengan parameter
return Redirect::action('UserController@profile',
array(1));

//redirect ke method controller dengan nama
parameter dan value-nya
return Redirect::action('UserController@profile',
array('user' => 1));
```

- **Views**

Views pada umumnya berisi kode-kode HTML, dan menyediakan cara yang aman untuk memisahkan antara controller dan tampilan. Views disimpan pada direktori app/views.

- Simple view

Lokasi file di app/views/halo.php

```
<html>
<body>
    <h1>Halo, <?php echo $nama; ?></h1>
</body>
</html>
```

Kemudian membuat route sebagai berikut:

```
Route::get('/', function()  
{  
    return View::make('halo', array('nama' =>  
        'Aris'));  
});
```

- Passing data ke views

// cara konvensional

```
$view = View::make('halo')->with('nama', 'Aris');
```

// cara lain

```
$view = View::make('halo')->withNama('Aris');
```

- Passing dengan parameter

```
$view = View::make('halo', $data);
```

- Passing data ke semua views

```
View::share('nama', 'Aris');
```

- Passing view dari sub view ke view

misalnya terdapat subview di app/views/template/footer.php, kita ingin memasukkan ke view halo, seperti berikut

```
$view = View::make('halo')  
    ->nest('footer', 'template.footer');  
  
//atau  
  
$view = View::make('halo')  
    ->nest('footer', 'template.footer',  
    $data);
```



kemudian di render di view halo seperti berikut

```
<html>
  <body>
    <h1>Halo, <?php echo $nama; ?></h1>
  </body>
  <?php echo $footer ?>
</html>
```

- Melihat keberadaan view

untuk mengecek keberadaan sebuah view bisa menggunakan method

View::exists

```
if (View::exists('template.content'))
{
    //
}
```

- **View Composers**

View composers digunakan untuk mengorganisasikan view-view yang sudah ada menjadi satu lokasi, fungsi ini bisa disebut seperti “view models” atau “presenters”

- Menentukan view composer

```
View::composer('profile', function($view)
{
    $view->with('count', User::count());
});
```

maka setiap kali view profile dirender, data count akan mengikuti view.

- Multi view composer

```
View::composer(array('profile', 'dashboard'),
```

```
function($view)
{
    $view->with('count', User::count());
});
```

- Class base composer

```
View::composer('profile', 'ProfileComposer');
```

dan class composer nya sebagai berikut:

```
class ProfileComposer {
    public function compose($view)
    {
        $view->with('count', User::count());
    }
}
```

- Mengatur multiple composer

```
View::composers(array(
    'AdminComposer' => array('admin.index',
    'admin.profile'),
    'UserComposer' => 'user',
    'ProductComposer@create' => 'product'
));
```

- **Special Responses**

- JSON Response

```
return Response::json(array('nama' => 'Aris',
    'Kota' => 'Jakarta'));
```

- JSONP Response

```
return Response::json(array('nama' => 'Aris',
```

```
'Kota' => 'Jakarta'))  
  
-  
  
>setCallback(Input::get('callback'))  
  
;
```

- File Download Response

```
return Response::download($pathToFile);  
  
//atau  
  
return Response::download($pathToFile, $name,  
$headers);
```

- **Response Macros**

```
Response::macro('caps', function($value)  
{  
    return Response::make(strtoupper($value));  
});
```

## Controllers

- **Basic Controllers**

Walaupun bisa juga mendefinisikan semua logika di route, controller juga penting digunakan untuk mendefinisikan logika berdasarkan kelas-kelas sehingga proses OOP dapat dilakukan. Controller biasanya disimpan di direktori `app/controllers`, direktori ini sudah terdaftar di dalam `classMap` file `composer.json` secara default. Jika ingin meletakkan file controller di direktori lain, maka harus dideklarasikan terlebih dahulu agar composer tahu letak file tersebut.

Contoh basic controller:

```
class UserController extends BaseController {  
  
    /**
```

```
* Show the profile for the given user.
*/
public function showProfile($id)
{
    $user = User::find($id);

    return View::make('user.profile',
array('user' => $user));
}
}
```

Semua kelas controller seharusnya meng-extend kelas BaseController. Kelas BaseController juga disimpan di direktori app/controllers, dan mungkin dapat digunakan untuk menuliskan kode logika yang bersifat shared/untuk semua controller yang meng-extend-nya. Setelah contoh basic controller diatas dibuat, sekarang kita bisa memanggil controller tersebut dari routes seperti berikut:

```
Route::get('user/{id}',
'UserController@showProfile');
```

- Memberi nama method controller di route

```
Route::get('user',
array('uses'=>'UserController@index', 'as' =>
'userindex'));
```

- **Controller Filter**

Filter pada controller ini memiliki fungsi seperti “regular” route, jika di route seperti:

```
Route::get('profile', array('before' => 'auth',
'uses' => 'UserController@showProfile'));
```

maka bisa juga dilakukan di controller seperti berikut:

```
class UserController extends BaseController {  
  
    /**  
        * Instantiate a new UserController instance.  
    */  
  
    public function __construct()  
    {  
        $this->beforeFilter('auth', array('except'  
            => 'getLogin'));  
        $this->beforeFilter('csrf', array('on' =>  
            'post'));  
        $this->afterFilter('log', array('only'  
            => array('fooAction',  
                'barAction')));  
    }  
}
```

- **Implicit Controllers**

Di laravel tidaklah harus membuat satu route untuk satu method controller, bisa juga satu route untuk satu controller dan semua method, seperti berikut:

```
Route::controller('users', 'UserController');
```

Dengan route di atas bisa dibuat controller dengan syarat, kata pertama pada method adalah HTTP verb (get, post, put, delete) dan any untuk mengijinkan semua jenis HTTP verb yang diikuti nama methodnya, contoh kelasnya sebagai berikut:

```
class UserController extends BaseController {  
  
    public function getIndex()  
    {
```

```
        //
    }

    public function postProfile()
    {
        //
    }

    public function anyLogin()
    {
        //
    }
}
```

- **RESTful Resource Controllers**

Resource controllers mempermudah dalam membuat RESTful controllers terhadap resource. Sebagai contoh ketika kita ingin memanajemen (create, read, update, delete) foto yang disimpan dalam aplikasi kita, bisa langsung menggunakan command line php artisan

```
php artisan controller:make PhotoController
```

kemudian daftarkan controller ke dalam route

```
Route::resource('foto', 'PhotoController');
```

satu route tersebut akan menghasilkan bermacam-macam route untuk menhandel berbagai macam RESTful action pada resource foto. Action yang di handel antara lain:

Verb	Path	Action	Route Name
------	------	--------	------------

GET	/foto	index	foto.index
GET	/foto/create	create	foto.create
POST	/foto	store	foto.store
GET	/foto/{resource}	show	foto.show
GET	/foto/{resource}/edit	edit	foto.edit
PUT/PATCH	/foto/{resource}	update	foto.update
DELETE	/foto/{resource}	destroy	foto.destroy

Bisa juga menentukan action apa yang dibutuhkan secara spesifik

```
Route::resource('photo', 'PhotoController',  
array('only' => array('index', 'show')));  
  
Route::resource('photo', 'PhotoController',  
array('except' => array('create', 'store',  
'update', 'destroy')));
```

## Forms dan HTML

- Pembuka Form

```
{{ Form::open(array('url' => 'foo/bar')) }}  
  
//  
  
{{ Form::close() }}
```

akan menghasilkan kode html seperti berikut:

```
<form action="http://domain/fo/bar">  
  
</form>
```

```
echo Form::open(array('url' => 'foo/bar', 'method'  
=> 'put'))
```

akan menghasilkan kode html seperti berikut:

```
<form action="http://domain/fo/bar">
```

```
<input type="hidden" name="_method" value="put">

</form>
```

Selain menggunakan url, form laravel juga bisa menggunakan route untuk nama route, action untuk nama method dari controller

```
echo Form::open(array('route' => 'route.name'))

echo Form::open(array('action' =>
    'Controller@method'))
```

Untuk menambahkan enctype multipart/form-data gunakan seperti berikut:

```
echo Form::open(array('url' => 'foo/bar', 'files'
    => true))
```

- **Proteksi CSRF**

CSRF (Cross-site request forgery) digunakan untuk melindungi form dari serangan cross-site request. Laravel akan generate token csrf ini secara otomatis ketika kita menggunakan Form::open dengan method POST, PUT atau DELETE. Jika ingin generate secara manual, dapat ditambahkan seperti berikut:

```
echo Form::token()
```

kemudian tambahkan CSRF filter ke dalam route untuk action form tersebut, misalnya:

```
Route::post('profile', array('before' => 'csrf',
    function()

        //

    )))
```

- **Form Model**

Form model biasanya digunakan pada form edit. Form model ini berfungsi untuk memasukkan data-data dari database ke form value sesuai dengan field



masing-masing

```
echo Form::model($user, array('route'

=> array('user.update', $user-
>id)));
```

- **Label**

Generate elemen label

```
echo Form::label('email', 'E-Mail Address');
```

Generate elemen label dengan atribut html

```
echo Form::label('email', 'E-Mail Address',
array('class'=>'awesome'));
```

- **Text, Text Area, Field Password dan Field Hidden**

Generate Input text

```
echo Form::text('username');
```

Generate Input text dengan default value

```
echo Form::text('email', 'aris@airputih.or.id');
```

Untuk Field hidden dan Text Area strukturnya sama dengan text input.

Generate Field Password

```
echo Form::password('password');
```

- **Input Number**

Generate Input number

```
echo Form::number('name', 'value');
```

- **Input File**

Generate Input file

```
echo Form::file('image');
```

- **Dropdown List**

Generate dropdown list

```
echo Form::select('size', array('L'=>'large',  
'S'=>'small'));
```

Generate dropdown list dengan default value

```
echo Form::select('size', array('L'=>'large',  
'S'=>'small'), 'S');
```

Generate group list

```
echo Form::select('animal', array(  
    'Cats' => array('leopard' => 'Leopard'),  
    'Dogs' => array('spaniel' => 'Spaniel'),  
));
```

Generate dropdown list dengan nilai antara

```
echo Form::selectRange('number', 10, 20);
```

Generate dropdown list dengan nama bulan

```
echo Form::selectMonth('month');
```

- **Button**

Generate button submit

```
echo Form::submit('Click Me!');
```

## Lab. Tutorial Laravel 4.2

### Kebutuhan

Untuk dapat menggunakan laravel diperlukan paket pendukung dengan spesifikasi minimal sebagai berikut:

- Webserver (Apache2)



- PHP 5.4 keatas
- Mcrypt php extension
- Database (MySQL)
- Koneksi internet

## **Instalasi laravel via composer di Ubuntu 14.04**

### **a. Webserver, composer**

Masuk menu terminal dan ketikkan perintah berikut:

```
$ sudo apt-get install apache2 mysql-server curl php5-curl  
php5-mcrypt phpmyadmin
```

```
$ sudo php5enmod mcrypt  
$ sudo service apache2 reload
```

```
$ sudo a2enmod rewrite  
$ sudo service apache2 restart
```

```
$ cd ~  
$ curl -sS https://getcomposer.org/installer | sudo php  
$ sudo mv composer.phar /usr/local/bin/composer
```

```
$ sudo chgrp www-data /var/www/html/  
$ sudo chmod 775 /var/www/html/  
$ sudo chmod g+s /var/www/html/
```

Asumsi username saya adalah: aris

```
$ sudo usermod -a -G www-data aris
```

## *Panduan Laravel PHP Framework*

```
$ sudo chown aris /var/www/html/
```

### **b. Instalasi laravel via composer**

```
$ cd /var/www/html/
```

Misalnya kita akan membuat project dengan nama: blog

```
$ composer create-project laravel/laravel blog --prefer-dist
```

```
$ sudo cp /etc/apache2/sites-available/000-default.conf  
/etc/apache2/sites-available/laravel.conf
```

```
$ sudo nano /etc/apache2/sites-available/laravel.conf
```

Lalu edit dan tambahkan seperti berikut:

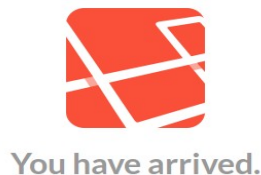
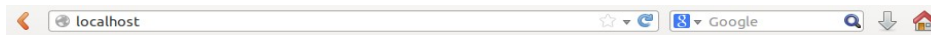
```
DocumentRoot /var/www/html/blog/public  
<Directory /var/www/html/blog/public>  
Options Indexes FollowSymlinks Multiviews  
AllowOverride All  
Order allow,deny  
allow from all  
</Directory>
```

```
$ sudo service apache2 reload  
$ sudo a2dissite 000-default.conf  
$ sudo a2ensite laravel.conf  
$ sudo service apache2 reload
```

Kemudian buka browser, ketikkan: localhost. Jika muncul error "Error in exception handler." ketikkan:

```
$ sudo chmod -R 775 /var/www/html/blog/app/storage/
```

Jika berhasil akan tampil sebagai berikut:



## Struktur Folder dan File Laravel

Setiap framework mempunyai struktur, hirarki dan tata letak folder maupun file masing-masing. Di laravel strukturnya adalah sebagai berikut:

blog/

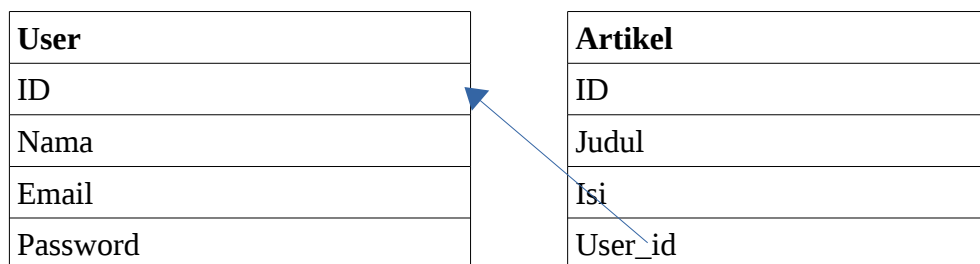
```
|— app
|   |— config
|   |   |— app.php
|   |   |— database.php (file konfigurasi koneksi ke database)
|   |   |— mail.php
|   |— controllers
|   |   |— BaseController.php
|   |   |— HomeController.php
```

## Panduan Laravel PHP Framework

```
| |— database
| | |— migrations
| | |— production.sqlite
| | |— seeds
| |— filters.php
| |— models
| | |— User.php
| |— routes.php (untuk mengatur alur dan struktur URL)
| |— views
| | |— emails
| | |— hello.php
|— artisan
|— public
| |— index.php
|— server.php
|— vendor
```

## Membuat blog sederhana

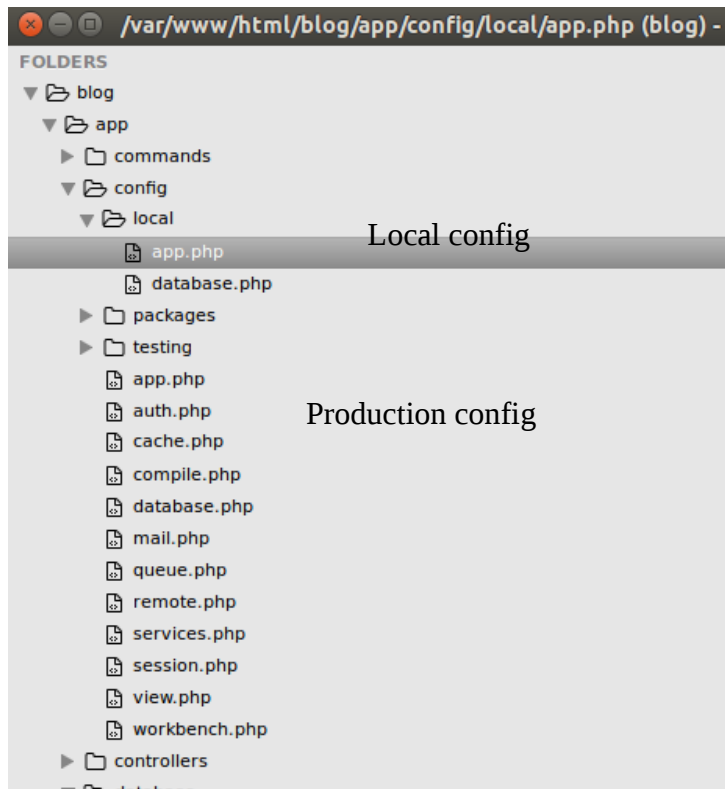
### a. Desain database



### b. Konfigurasi app laravel

Laravel mempunyai 2 bagian konfigurasi, yaitu production config dan local config. Production config adalah konfigurasi yang digunakan jika aplikasi web sudah published (public). Local config adalah konfigurasi yang digunakan pada saat development

(lokal).



Untuk mendeteksi secara otomatis apakah aplikasi web diakses dari lokal atau public, tentukan di environment-nya, yang berada pada file [blog/bootstrap/start.php](#). Pada variabel `$env` edit atau tambahkan kode seperti berikut:

```
$env = $app->detectEnvironment(array(
    'local' => array('localhost', gethostname()),
    'production' => array('*.*.com', '*.net',
'www.somedomain.com')
));
```

- Konfigurasi app local config, [blog/app/config/local/app.php](#)

```
'debug' => true
```

## Panduan Laravel PHP Framework

- Konfigurasi app production config [blog/app/config/app.php](#)

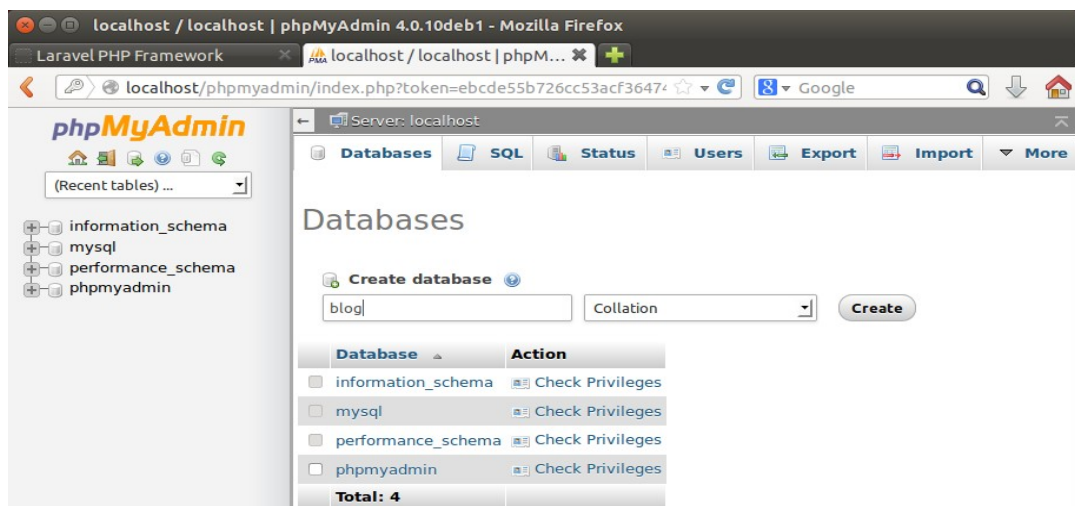
```
'debug' => true,  
'url' => '',  
'timezone' => 'Asia/Jakarta',
```

Dengan konfigurasi di atas, laravel akan mengaktifkan mode debug dimana jika terjadi kesalahan script atau kode yang anda tuliskan, laravel akan menampilkan error secara detail. Kemudian url di set '' (null) agar aplikasi bisa diakses tidak hanya menggunakan nama host, tetapi juga bisa menggunakan alamat ip. Dan timezone diset sesuai dengan aturan php timezone <http://php.net/manual/en/timezones.asia.php>

### c. Membuat database

Masuk ke browser ketik: <http://localhost/phpmyadmin>

- Login dengan user dan password phpmyadmin anda, klik menu “Databases” ketikkan nama database anda misalnya “blog” lalu klik tombol “Create”:



- Konfigurasi database local config, [blog/app/config/local/database.php](#)

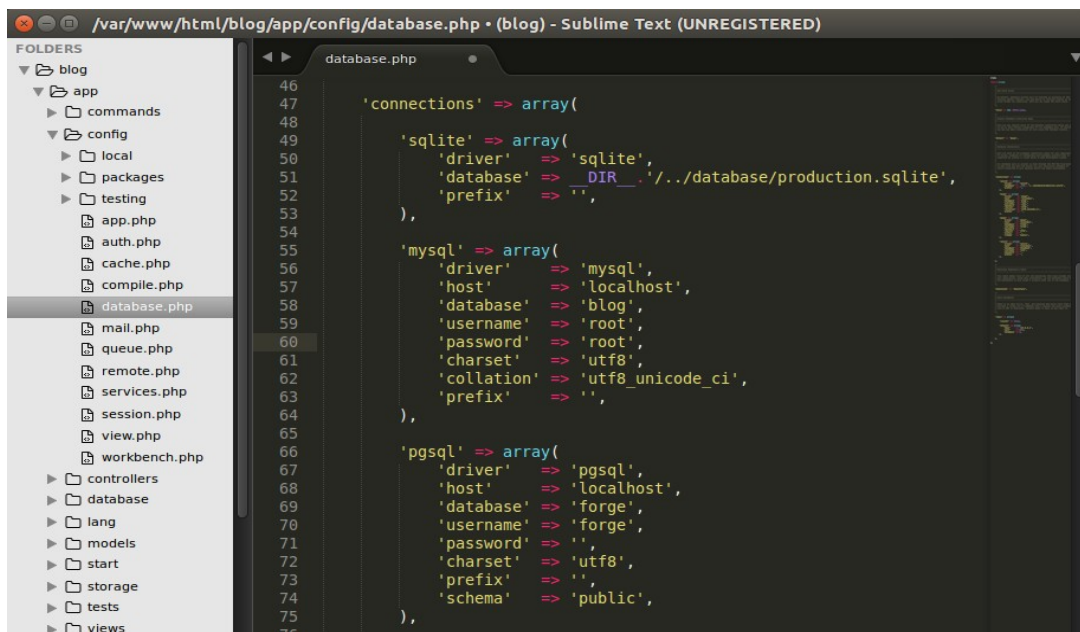
```
'mysql' => array(  
    'driver' => 'mysql',
```



```
'host'      => 'localhost',
'database'  => 'blog',
'username'  => 'root',
'password'  => 'root',
'charset'   => 'utf8',
'collation' => 'utf8_unicode_ci',
'prefix'    => '',
),
```

- Konfigurasi database production config blog/app/config/database.php

```
'mysql' => array(
    'driver'   => 'mysql',
    'host'     => 'localhost',
    'database' => 'blog',
    'username' => 'root',
    'password' => 'root',
    'charset'  => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'   => '',
),
```



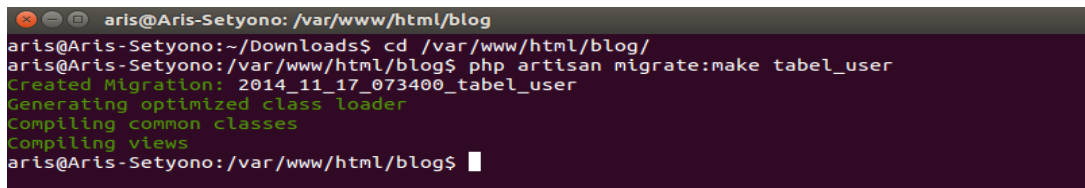
## *Panduan Laravel PHP Framework*

- Untuk membuat tabel di database, laravel menggunakan sistem migrate, yang memungkinkan anda membuat lebih dari satu jenis database dengan satu perintah yang sama. Untuk menjalankan sistem migrate ini dilakukan dengan cara artisan command line dari path root app, ikuti perintah berikut:

### **Membuat tabel user**

```
$ cd /var/www/html/blog
```

```
$ php artisan migrate:make tabel_user
```



```
aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono:~/Downloads$ cd /var/www/html/blog/
aris@Aris-Setyono:/var/www/html/blog$ php artisan migrate:make tabel_user
Created Migration: 2014_11_17_073400_tabel_user
Generating optimized class loader
Compiling common classes
Compiling views
aris@Aris-Setyono:/var/www/html/blog$
```

Laravel akan otomatis generate file “<tanggal>\_tabel\_user.php” di dalam folder blog/app/database/migrations. Buka file tersebut kemudian edit seperti berikut:

```
public function up()
{
    Schema::create('user', function (Blueprint $table)
    {
        $table->increments('id');
        $table->string('nama', 200);
        $table->string('email', 200);
        $table->string('password', 100);
        $table->string('remember_token', 100);
        $table->timestamps();
    });
}
```

```
public function down()
{

```

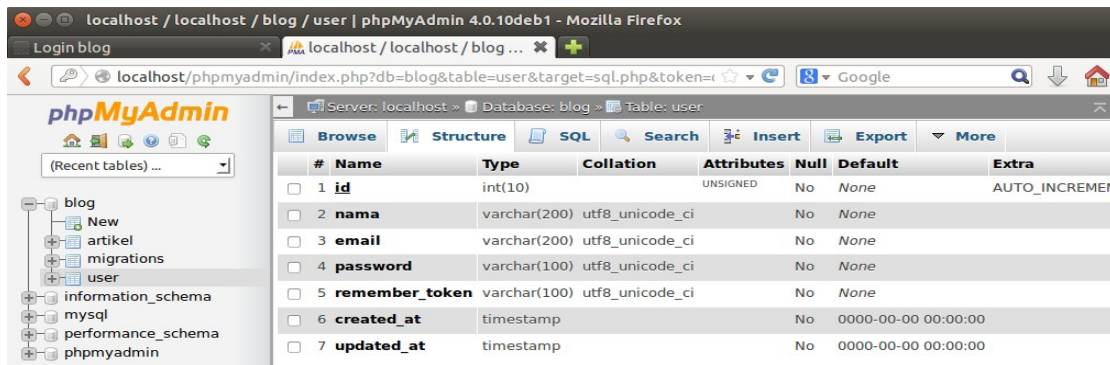
```
Schema::drop('user');
}
```

Kemudian jalankan perintah artisan dari terminal:

```
$ php artisan migrate
```

```
aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono:/var/www/html/blog$ php artisan migrate
Migrated: 2014_11_17_073400_tabel_user
aris@Aris-Setyono:/var/www/html/blog$
```

Secara otomatis tabel user di database blog akan digenerate, untuk melihat hasilnya cek di phpmyadmin seperti berikut:



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(10)		UNSIGNED	No	None	AUTO_INCREMENT
2	nama	varchar(200)	utf8_unicode_ci		No	None	
3	email	varchar(200)	utf8_unicode_ci		No	None	
4	password	varchar(100)	utf8_unicode_ci		No	None	
5	remember_token	varchar(100)	utf8_unicode_ci		No	None	
6	created_at	timestamp			No	0000-00-00 00:00:00	
7	updated_at	timestamp			No	0000-00-00 00:00:00	

## Membuat tabel artikel

```
$ php artisan migrate:make tabel_artikel
```

```
aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono:/var/www/html/blog$ php artisan migrate:make tabel_artikel
Created Migration: 2014_11_20_111740_tabel_artikel
Generating optimized class loader
aris@Aris-Setyono:/var/www/html/blog$
```

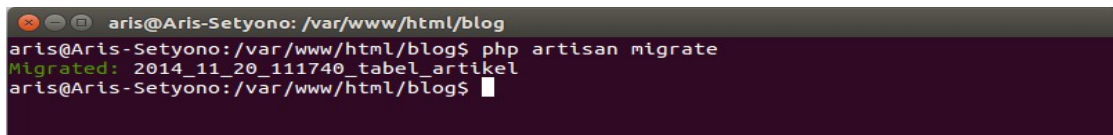
Edit file `blog/app/database/migrations/<tanggal>_tabel_artikel.php`

```
public function up()
{
    Schema::create('artikel', function (Blueprint $table)
```

```
        {  
            $table->increments('id');  
            $table->string('judul', 200);  
            $table->string('isi', 1000);  
            $table->integer('user_id');  
            $table->timestamps();  
        });  
    }  
  
    public function down()  
    {  
        Schema::drop('artikel');  
    }  
}
```

Kemudian jalankan perintah:

***\$ php artisan migrate***



A terminal window screenshot showing the command `$ php artisan migrate` being executed. The output shows the migration of the `artikel` table, with a timestamp of `2014_11_20_111740_tabel_artikel`. The terminal prompt is `aris@Aris-Setyono: /var/www/html/blog`.

Tabel user dan tabel artikel sudah terbuat langkah berikutnya adalah insert 1 data dummy ke database user.

### **Edit file blog/app/models/User.php**

Karena default laravel nama tabel untuk user adalah “users” sedangkan pada percobaan ini tabel user adalah “user” maka ubah model User table menjadi “user”

```
protected $table = 'user';
```

**Buat file/class baru di `blog/app/database/seeds` misalnya `InsertUser.php`**

`blog/app/database/seeds/InsertUser.php`

```
<?php
class InsertUser extends Seeder
{
    public function run()
    {
        User::create(
            array(
                'nama'      => 'Aris Setyono',
                'email'     => 'aris@airputih.or.id',
                'password' => Hash::make('123aris'),
            )
        );
    }
}
```

**Edit file `blog/app/database/seeds/DatabaseSeeder.php`**

pada function run ubah seperti berikut:

```
public function run()
{
    Eloquent::unguard();

    $this->call('InsertUser');
}
```

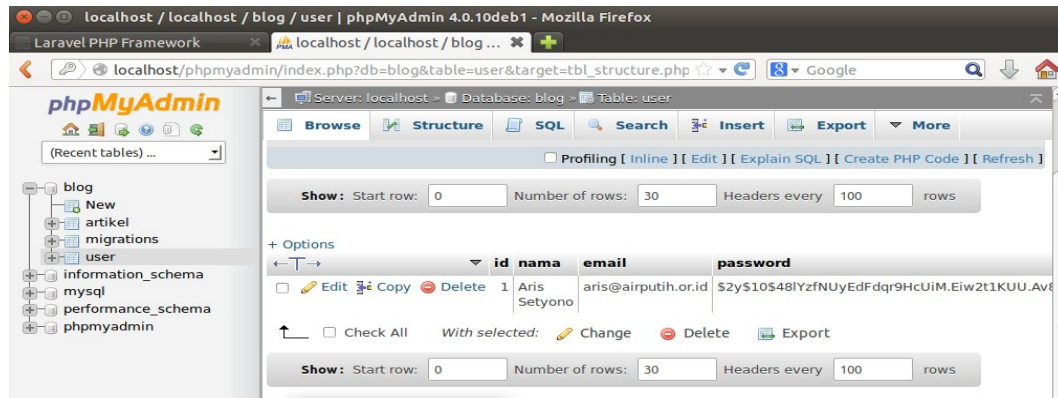
Kemudian jalankan perintah

```
$ php artisan db:seed
```

## Panduan Laravel PHP Framework

```
aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono: /var/www/html/blog$ php artisan db:seed
Seed: InsertUser
aris@Aris-Setyono: /var/www/html/blog$
```

Cek tabel user di database



Database sudah terbuat, kemudian dilanjutkan dengan pembuatan mekanisme login dan logout. Karena model user sudah ada maka tinggal membuat controller dan view saja.

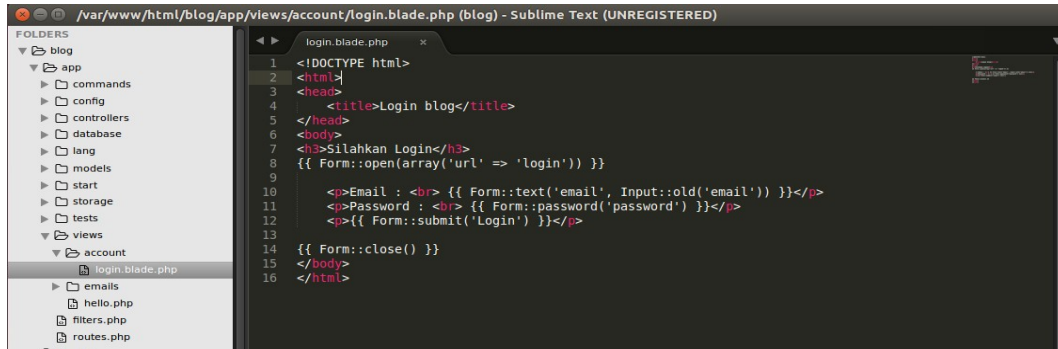
Membuat folder account dan form login di view memanfaatkan blade programming dengan eksistensi `.blade.php`.

`blog/app/views/account/login.blade.php`

```
<!DOCTYPE html>
<html>
<head>
    <title>Login blog</title>
</head>
<body>
<h3>Silahkan Login</h3>
{{ Form::open(array('url' => 'login')) }}

    <p>Email : <br> {{ Form::text('email',
Input::old('email')) }}</p>
    <p>Password : <br> {{ Form::password('password') }}</p>
    <p>{{ Form::submit('Login') }}</p>
```

```
{{ Form::close() }}  
</body>  
</html>
```

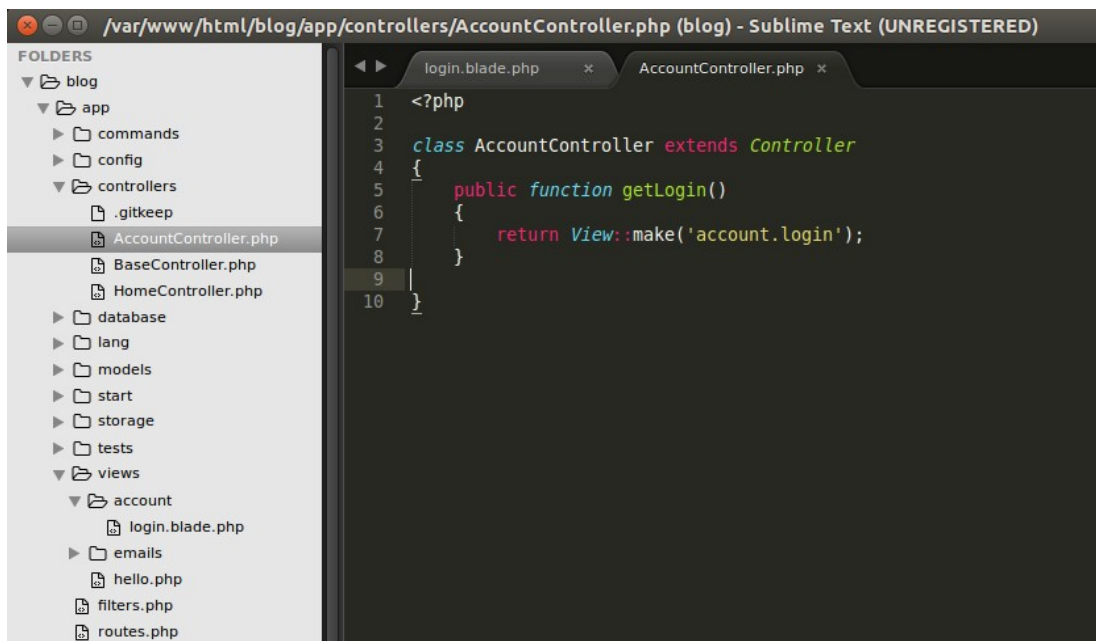


**Membuat file/class di controller misalnya dengan nama “AccountController”**

blog/app/controllers/AccountController.php

```
<?php  
  
class AccountController extends Controller  
{  
  
    public function getLogin()  
    {  
  
        return View::make('account.login');  
  
    }  
  
}
```

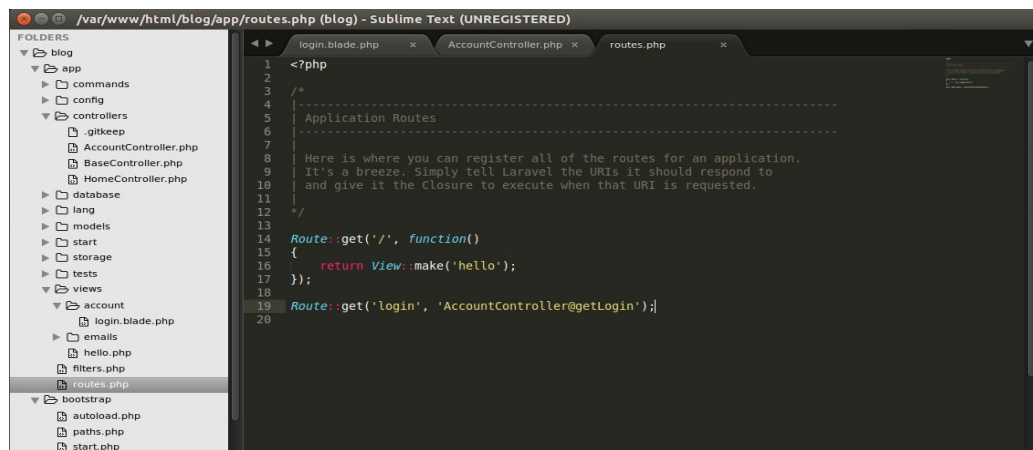
## Panduan Laravel PHP Framework



Edit file routes.php untuk mengatur URL yang digunakan untuk login.

Tambahkan route login di blog/app/routes.php

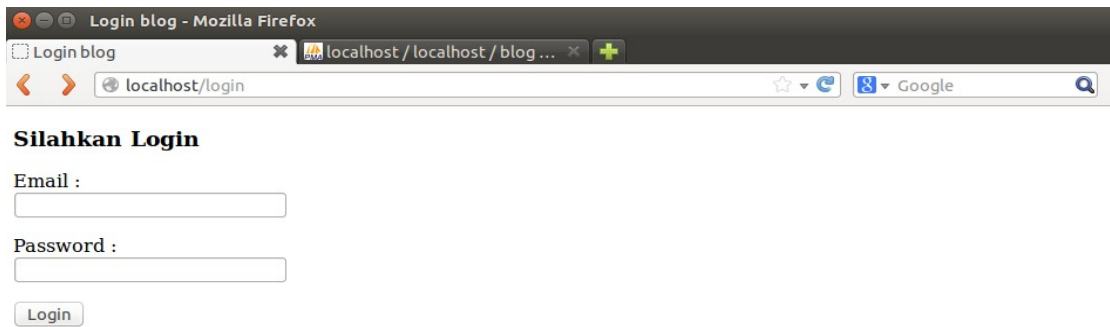
```
Route::get('login', 'AccountController@getLogin');
```



Kemudian untuk melihat hasilnya buka menggunakan web browser

<http://localhost/login>





Tambahkan function `postLogin` untuk menangkap dan melakukan pengecekan email dan password

<blog/app/controllers/AccountController.php>

```
public function postLogin()
{
    $rules = array(
        'email'      => 'required/email',
        'password'   => 'required',
    );

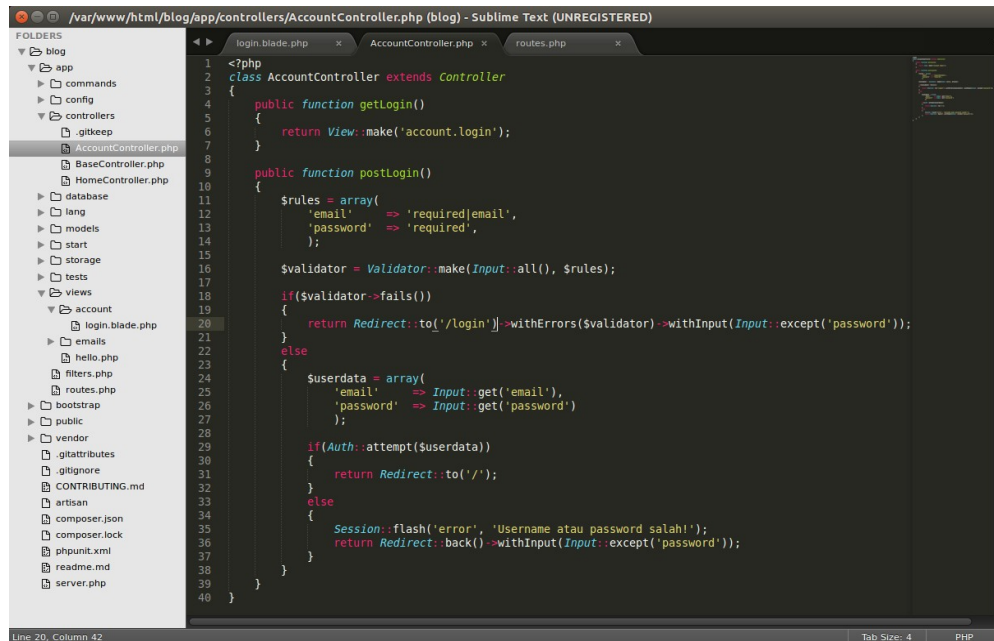
    $validator = Validator::make(Input::all(), $rules);

    if($validator->fails())
    {
        return Redirect::to('/login')
            ->withErrors($validator)
            ->withInput(Input::except('password'));
    }
    else
    {
        $userdata = array(
            'email'      => Input::get('email'),
            'password'   => Input::get('password')
        );
    }
}
```

## Panduan Laravel PHP Framework

```
        if(Auth::attempt($userdata))
        {
            return Redirect::to('/');
        }
        else
        {
            Session::flash('error', 'Username atau password
salah!');

            return Redirect::back()
                ->withInput(Input::except('password'));
        }
    }
}
```



**Tambahkan function logout untuk melakukan logout authentication**

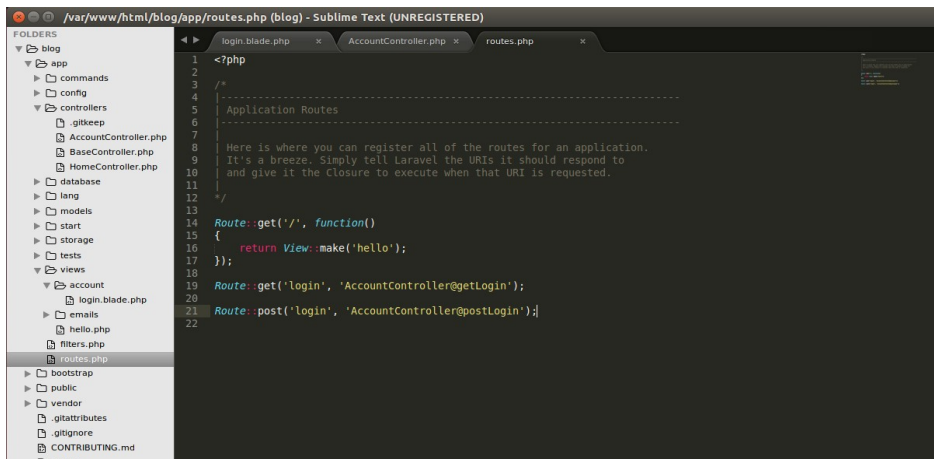
[blog/app/controllers/AccountController.php](#)

```
public function doLogout ()
{
    Auth::logout ();
    return Redirect::to('/login');
```

```
}
```

Tambahkan juga route di routes.php untuk method post

```
Route::post('login', 'AccountController@postLogin');  
Route::get('logout', 'AccountController@doLogout');
```

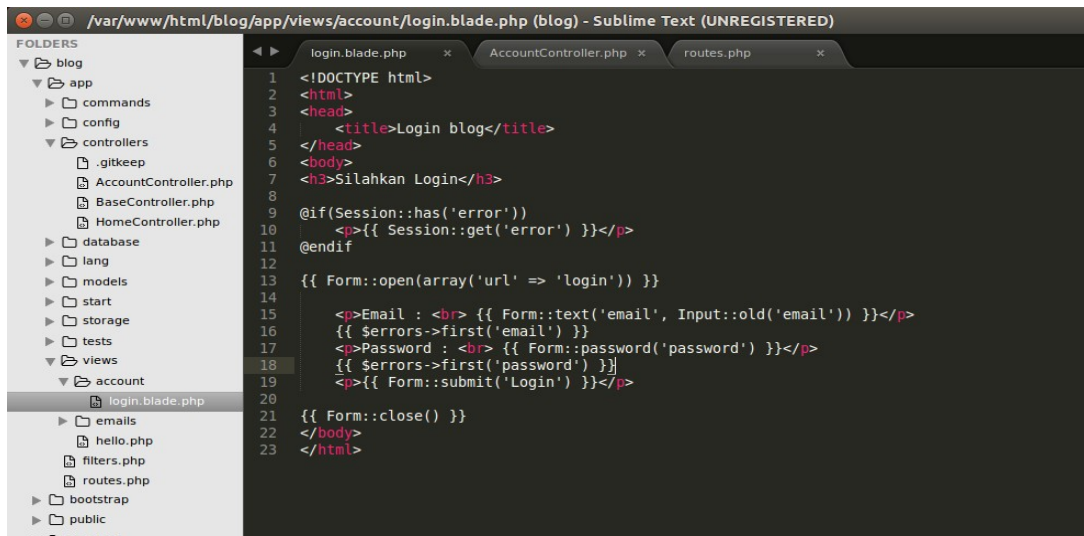


Tambahkan juga di view untuk menampilkan error authentication dan error validation

<blog/app/views/account/login.blade.php>

```
@if(Session::has('error'))  
    <p>{{ Session::get('error') }}</p>  
@endif  
  
{{ $errors->first('email') }}  
{{ $errors->first('password') }}
```

## Panduan Laravel PHP Framework



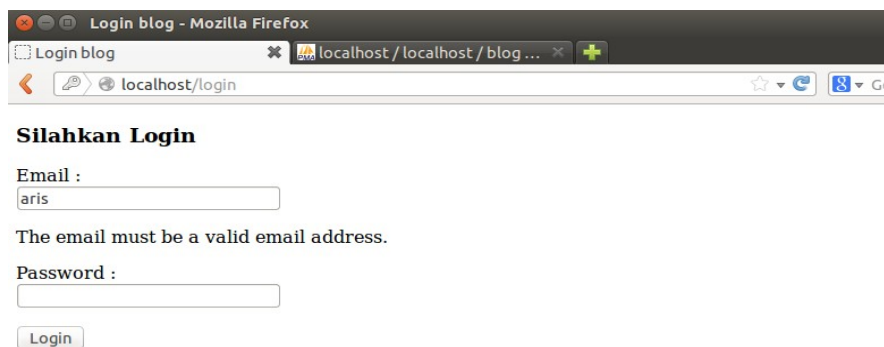
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Login blog</title>
5 </head>
6 <body>
7 <h3>Silahkan Login</h3>
8
9 @if(Session::has('error'))
10 <p>{{ Session::get('error') }}</p>
11 @endif
12
13 {{ Form::open(array('url' => 'login')) }}
14
15 <p>Email : <br> {{ Form::text('email', Input::old('email')) }}</p>
16 {{ $errors->first('email') }}
17 <p>Password : <br> {{ Form::password('password') }}</p>
18 {{ $errors->first('password') }}
19 <p>{{ Form::submit('Login') }}</p>
20
21 {{ Form::close() }}
22 </body>
23 </html>
```

Test form login apakah authentication dan validation sudah berhasil.

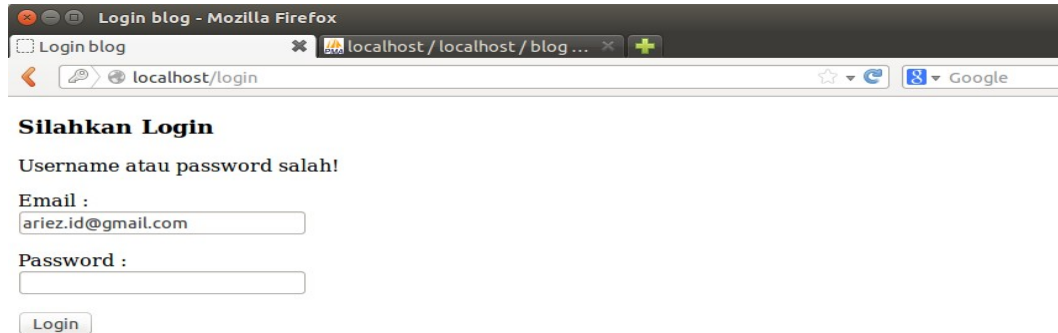
- Form tanpa diisi langsung submit



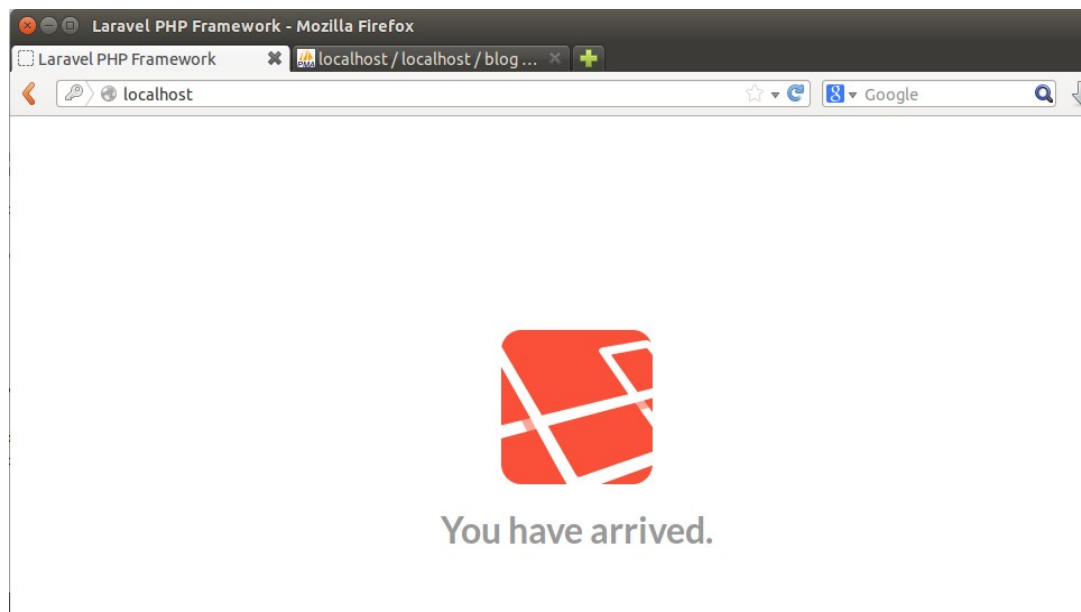
- Form diisi email yang tidak sesuai format dan password



- Form diisi email dan password yang belum terdaftar atau belum ada di database



- Form diisi email dan password yang sudah terdaftar, (aris@airputih.or.id:123aris)



### **Membuat controller untuk manajemen artikel**

Membuat file/class controller baru yang resourceful (index, create, store, show, edit, update, destroy) bisa melalui command line seperti berikut:

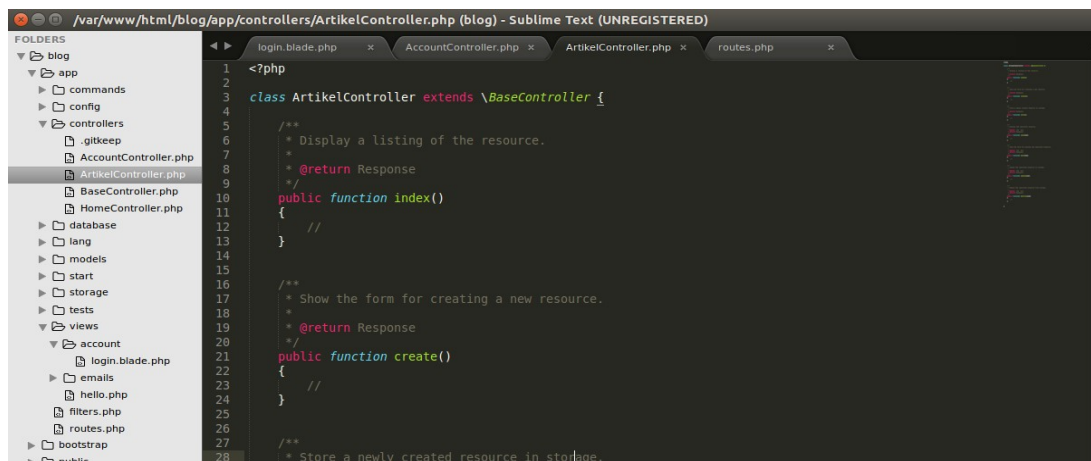
```
$ php artisan controller:make ArtikelController
```

## Panduan Laravel PHP Framework

```
aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono:/var/www/html/blog$ php artisan controller:make ArtikelController

Controller created successfully!
aris@Aris-Setyono:/var/www/html/blog$
aris@Aris-Setyono:/var/www/html/blog$
```

maka akan otomatis generate file ArtikelController.php di controllers



```
login.blade.php x AccountController.php x ArtikelController.php x routes.php x
1 <?php
2
3 class ArtikelController extends BaseController {
4
5     /**
6      * Display a listing of the resource.
7      *
8      * @return Response
9      */
10    public function index()
11    {
12        //
13    }
14
15    /**
16     * Show the form for creating a new resource.
17     *
18     * @return Response
19     */
20    public function create()
21    {
22        //
23    }
24
25    /**
26     * Store a newly created resource in storage.
27     */
28 }
```

## Tambahkan route resource di file routes.php

```
Route::group(array('before'=>'auth'), function()
{
    Route::resource('artikel', 'ArtikelController');
});
```

```

1 <?php
2
3 /*
4  * Application Routes
5  *
6  * Here is where you can register all of the routes for an application.
7  * It's a breeze. Simply tell Laravel the URIs it should respond to
8  * and give it the Closure to execute when that URI is requested.
9  */
10
11
12
13
14 Route::get('/', function()
15 {
16     return View::make('hello');
17 });
18
19 Route::get('login', 'AccountController@getLogin');
20
21 Route::post('login', 'AccountController@postLogin');
22 Route::get('logout', 'AccountController@doLogout');
23
24 Route::group(array('before'=>'auth'), function()
25 {
26     Route::resource('artikel', 'ArtikelController');
27 });
28

```

Untuk melihat route yang aktif jalankan perintah

***\$ php artisan routes***

```

aris@Aris-Setyono: /var/www/html/blog
aris@Aris-Setyono: /var/www/html/blog$ php artisan routes

```

Domain	URI	Name	Action	Before Filters	After Filters
	GET HEAD /		Closure		
	GET HEAD login		AccountController@getLogin		
	POST login		AccountController@postLogin		
	GET HEAD artikel	artikel.index	ArtikelController@index		
	GET HEAD artikel/create	artikel.create	ArtikelController@create		
	POST artikel	artikel.store	ArtikelController@store		
	GET HEAD artikel/{artikel}	artikel.show	ArtikelController@show		
	GET HEAD artikel/{artikel}/edit	artikel.edit	ArtikelController@edit		
	PUT artikel/{artikel}	artikel.update	ArtikelController@update		
	PATCH artikel/{artikel}		ArtikelController@update		
	DELETE artikel/{artikel}	artikel.destroy	ArtikelController@destroy		

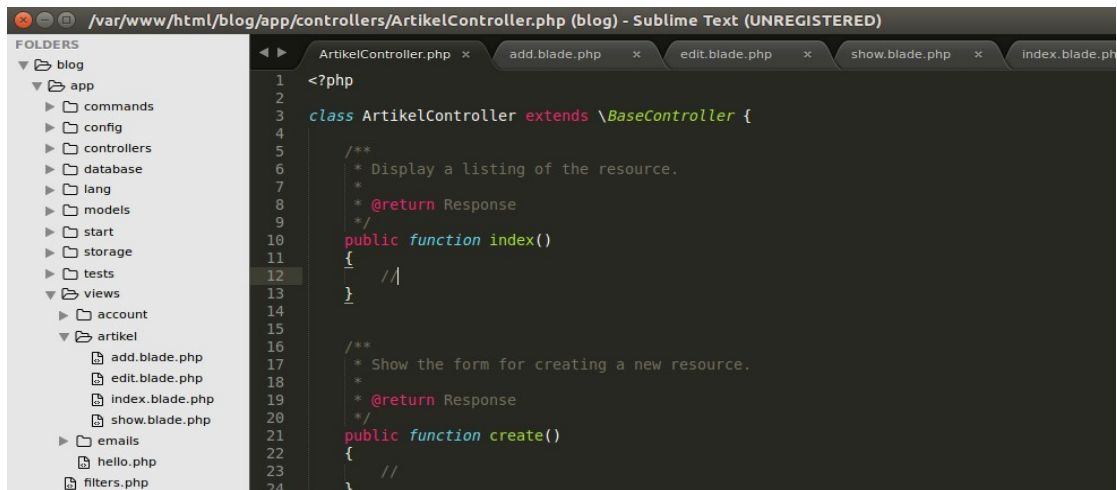
```

aris@Aris-Setyono: /var/www/html/blog$

```

Kemudian buat form add, edit, index, show di view-nya

## Panduan Laravel PHP Framework



Form add tambahkan kode berikut:

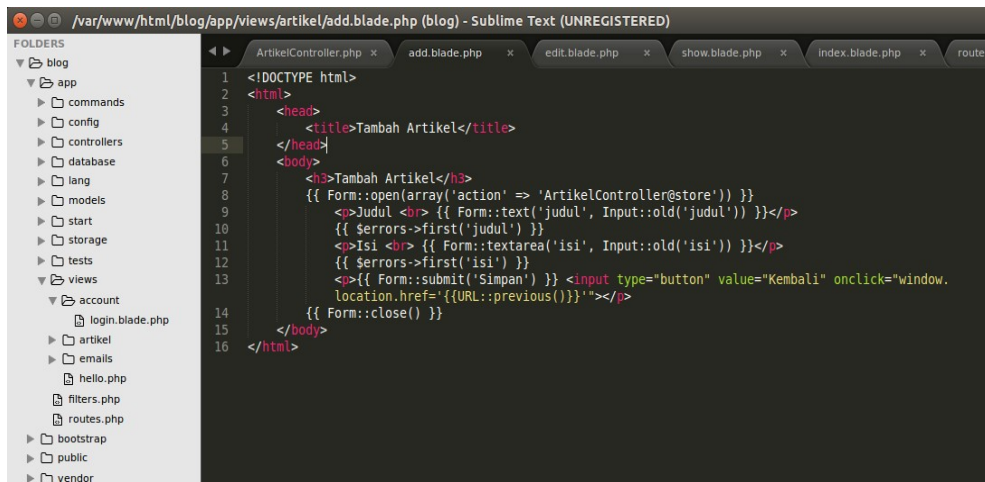
<blog/app/views/artikel/add.blade.php>

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tambah Artikel</title>
    </head>
    <body>
        <h3>Tambah Artikel</h3>
        {{ Form::open(array('action' => 'ArtikelController@store')) }}

        <p>Judul <br> {{ Form::text('judul',
Input::old('judul')) }}</p>
        {{ $errors->first('judul') }}
        <p>Isi <br> {{ Form::textarea('isi',
Input::old('isi')) }}</p>
        {{ $errors->first('isi') }}
        <p>{{ Form::submit('Simpan') }} <input type="button"
value="Kembali"
onclick="window.location.href='{{URL::previous()}}'"></p>
        {{ Form::close() }}
    </body>
```



</html>

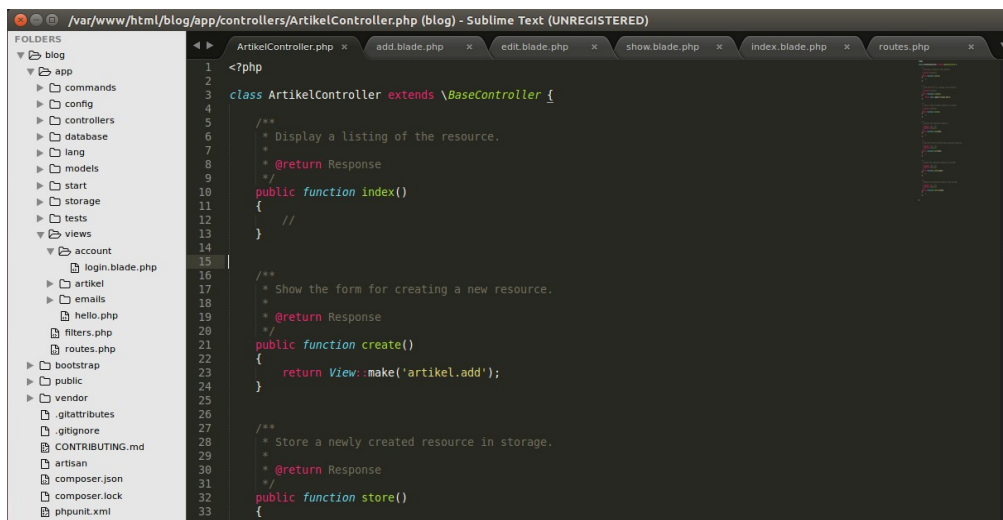


```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Tambah Artikel</title>
5   </head>
6   <body>
7     <h3>Tambah Artikel</h3>
8     {{ Form::open(array('action' => 'ArtikelController@store')) }}
9     <p>Judul <br> {{ Form::text('judul', Input::old('judul')) }}</p>
10    {{ $errors->first('judul') }}
11    <p>Isi <br> {{ Form::textarea('isi', Input::old('isi')) }}</p>
12    {{ $errors->first('isi') }}
13    <p>{{ Form::submit('Simpan') }} <input type="button" value="Kembali" onclick="window.
14      location.href='{{URL::previous()}}' "></p>
15    {{ Form::close() }}
16  </body>
17 </html>
```

Edit ArtikelController.php pada function create

blog/app/controllers/ArtikelController.php

```
public function create()
{
    return View::make('artikel.add');
}
```



```
1 <?php
2
3 class ArtikelController extends BaseController {
4
5     /**
6      * Display a listing of the resource.
7      *
8      * @return Response
9      */
10    public function index()
11    {
12        //
13    }
14
15    /**
16     * Show the form for creating a new resource.
17     *
18     * @return Response
19     */
20    public function create()
21    {
22        return View::make('artikel.add');
23    }
24
25    /**
26     * Store a newly created resource in storage.
27     *
28     * @return Response
29     */
30    public function store()
31    {
32        //
33    }
34}
```

Membuat file/class baru yaitu model Artikel. Karena ada relation dengan user maka buat function baru seperti berikut

[blog/app/models/Artikel.php](#)

```
<?php
class Artikel extends Eloquent
{
    protected $table = 'artikel';
    public function user()
    {
        return $this->hasOne('User', 'id', 'user_id');
    }
}
```



**Edit ArtikelController.php pada function store**

[blog/app/controllers/ArtikelController.php](#)

```
public function store()
{
    $rules = array(
        'judul'    => 'required',
        'isi'       => 'required'
    );

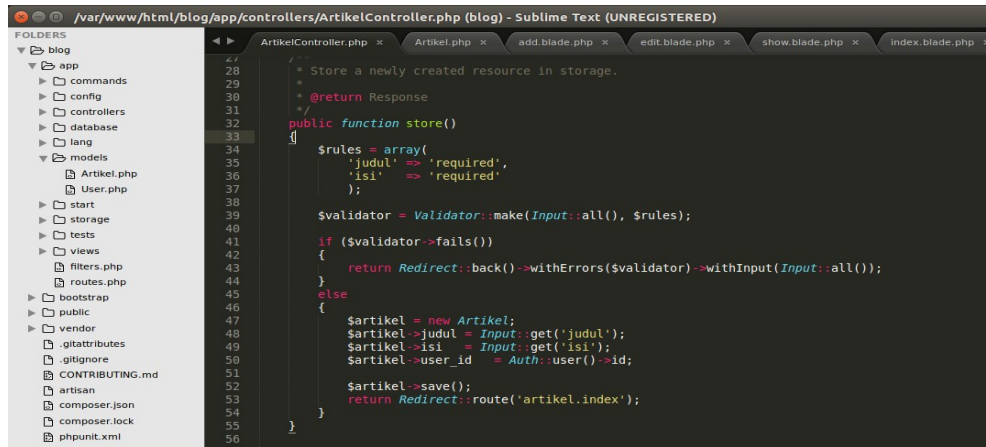
    $validator = Validator::make(Input::all(), $rules);
```

```

        if ($validator->fails())
        {
            return Redirect::back()->withErrors($validator)-
>withInput(Input::all());
        }
    else
    {
        $artikel = new Artikel;
        $artikel->judul    = Input::get('judul');
        $artikel->isi      = Input::get('isi');
        $artikel->user_id = Auth::user()->id;

        $artikel->save();
        return Redirect::route('artikel.index');
    }
}

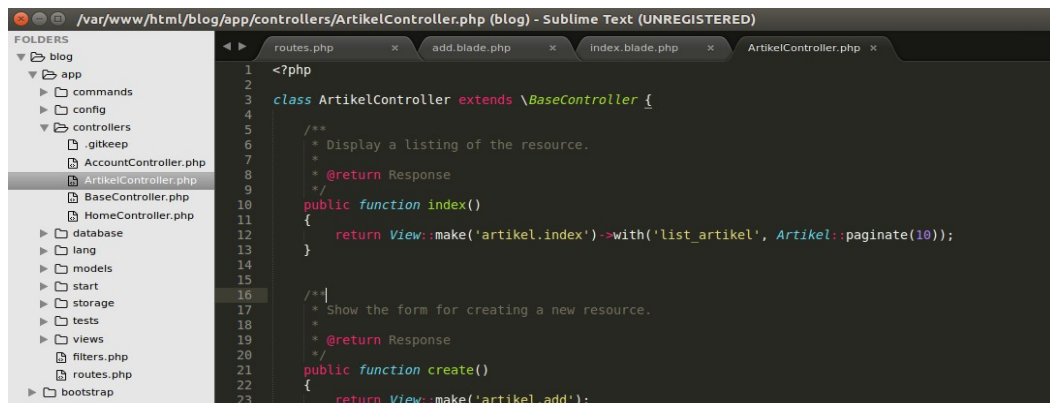
```



Membuat listing artikel, tambahkan kode berikut pada function index di controller artikel

[blog/app/controllers/ArtikelController.php](#)

```
public function index()
{
    return View::make('artikel.index')
        ->with('list_artikel', Artikel::paginate(10));
}
```



Membuat view untuk menampilkan list artikel,

[blog/app/views/artikel/index.blade.php](#)

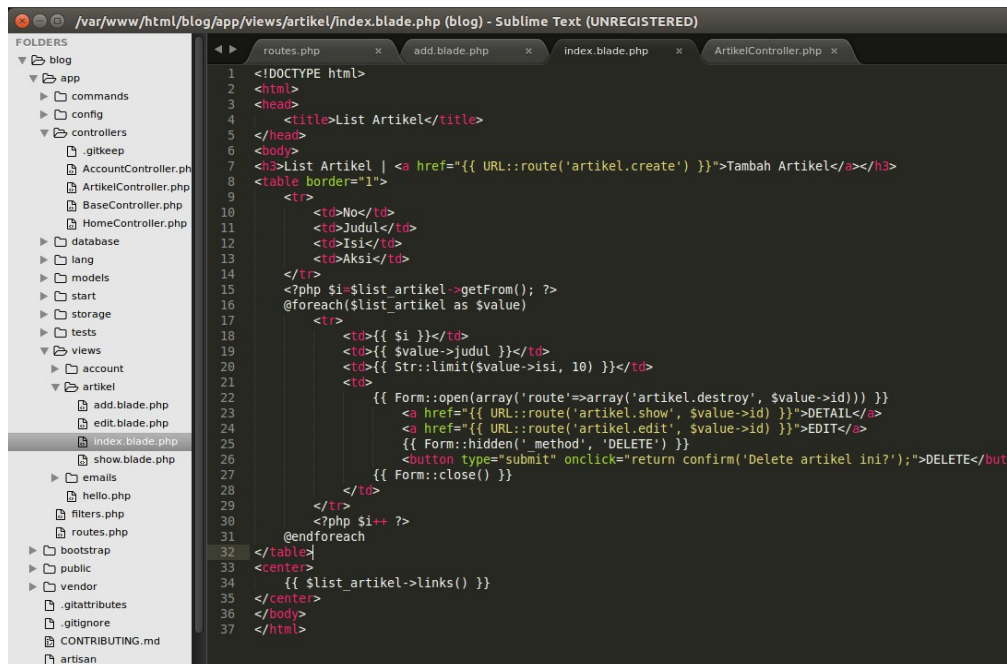
```
<!DOCTYPE html>
<html>
<head>
    <title>List Artikel</title>
</head>
<body>
    <h3>List Artikel | <a href="{{ URL::route('artikel.create') }}">Tambah
    Artikel</a></h3>
    <table border="1">
        <tr>
            <td>No</td>
```

```

        <td>Judul</td>
        <td>Isi</td>
        <td>Aksi</td>
    </tr>
    <?php $i=$list_artikel->getFrom(); ?>
    @foreach($list_artikel as $value)
        <tr>
            <td>{{ $i }}</td>
            <td>{{ $value->judul }}</td>
            <td>{{ Str::limit($value->isi, 10) }}</td>
            <td>
                {{ Form::open(array('route'=>array('artikel.destroy', $value->id))) }}
                <a href="{{ URL::route('artikel.show', $value->id) }}">DETAIL</a>
                <a href="{{ URL::route('artikel.edit', $value->id) }}">EDIT</a>
                {{ Form::hidden('_method', 'DELETE') }}
                <button type="submit" onclick="return confirm('Delete artikel ini?');">DELETE</button>
                {{ Form::close() }}
            </td>
        </tr>
    <?php $i++ ?>
    @endforeach
</table>
<center>
    {{ $list_artikel->links() }}
</center>
</body>
</html>

```

## Panduan Laravel PHP Framework

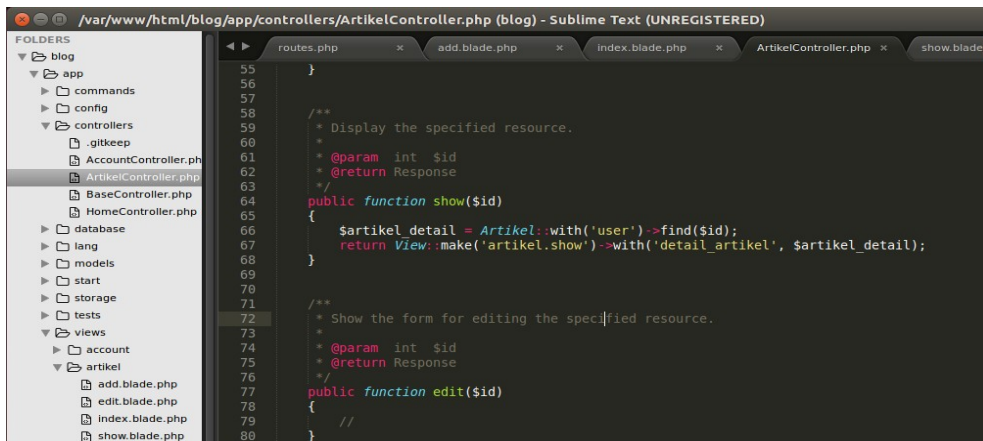


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>List Artikel</title>
5 </head>
6 <body>
7 <h3>List Artikel | <a href="{{ URL::route('artikel.create') }}">Tambah Artikel</a></h3>
8 <table border="1">
9 <tr>
10 <td>No</td>
11 <td>Judul</td>
12 <td>Isi</td>
13 <td>Aksi</td>
14 </tr>
15 <tr>
16 <td>{{ $i }}</td>
17 <td>{{ $value->judul }}</td>
18 <td>{{ Str::limit($value->isi, 10) }}</td>
19 <td>
20 {{ Form::open(array('route'=>array('artikel.destroy', $value->id))) }}
21 <a href="{{ URL::route('artikel.show', $value->id) }}">DETAIL</a>
22 <a href="{{ URL::route('artikel.edit', $value->id) }}">EDIT</a>
23 {{ Form::hidden('method', 'DELETE') }}
24 <button type="submit" onclick="return confirm('Delete artikel ini?');">DELETE</button>
25 {{ Form::close() }}
26 </td>
27 </tr>
28 <tr>
29 <td>{{ $i++ }}</td>
30 <td>{{ $value->judul }}</td>
31 <td>{{ Str::limit($value->isi, 10) }}</td>
32 <td>
33 {{ Form::open(array('route'=>array('artikel.destroy', $value->id))) }}
34 <a href="{{ URL::route('artikel.show', $value->id) }}">DETAIL</a>
35 <a href="{{ URL::route('artikel.edit', $value->id) }}">EDIT</a>
36 {{ Form::hidden('method', 'DELETE') }}
37 <button type="submit" onclick="return confirm('Delete artikel ini?');">DELETE</button>
38 {{ Form::close() }}
39 </td>
40 </tr>
41 </table>
42 <center>
43 {{ $list_artikel->links() }}
44 </center>
45 </body>
46 </html>
```

### Membuat detail artikel, function show pada ControllerArtikel.php

blog/app/controllers/ArtikelController.php

```
public function show($id)
{
    $artikel_detail = Artikel::with('user')->find($id);
    return View::make('artikel.show')
        ->with('detail_artikel',
            $artikel_detail);
}
```

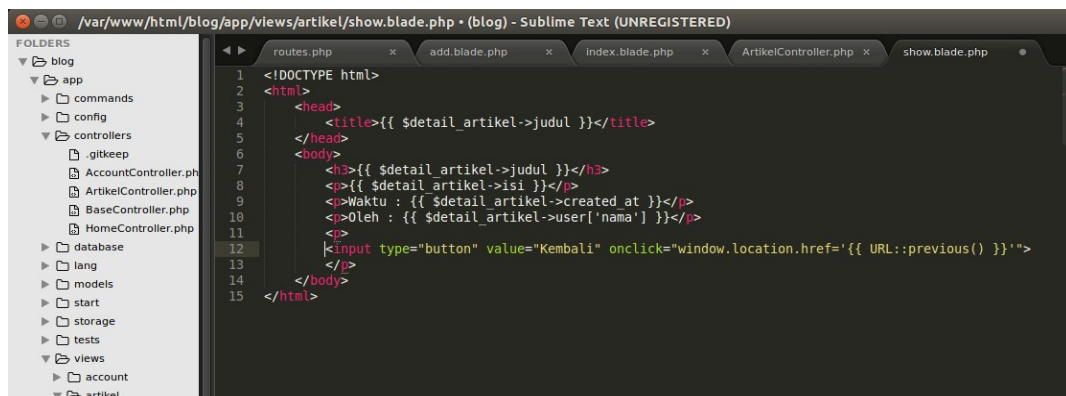


## Membuat form view untuk detail artikel

<blog/app/views/artikel/show.blade.php>

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ $detail_artikel->judul }}</title>
    </head>
    <body>
        <h3>{{ $detail_artikel->judul }}</h3>
        <p>{{ $detail_artikel->isi }}</p>
        <p>Waktu : {{ $detail_artikel->created_at }}</p>
        <p>Oleh : {{ $detail_artikel->user['nama'] }}</p>
        <p><input type="button" value="Kembali"
onclick="window.location.href='{{ URL::previous() }}'"></p>
    </body>
</html>
```

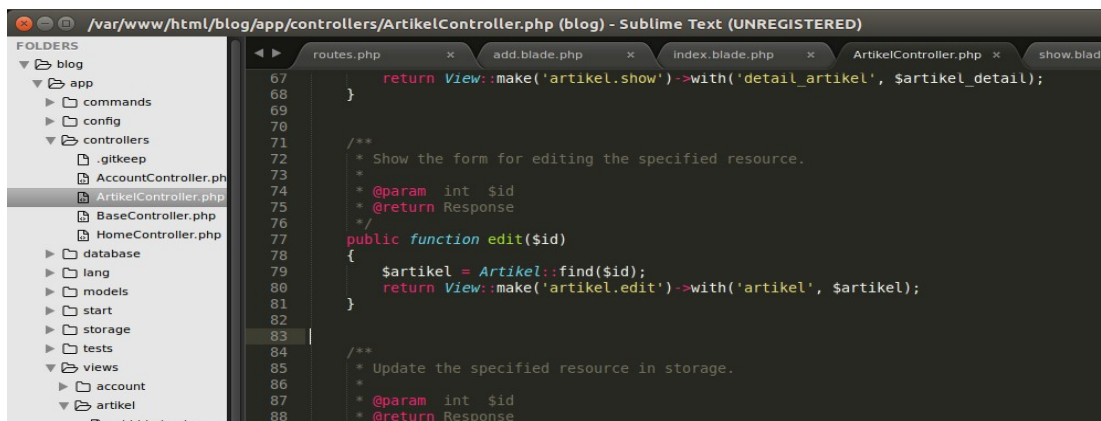
## Panduan Laravel PHP Framework



### Membuat edit artikel, function edit pada ArtikelController.php

<blog/app/controllers/ArtikelController.php>

```
public function edit($id)
{
    $artikel = Artikel::find($id);
    return View::make('artikel.edit')->with('artikel',
    $artikel);
}
```



### Membuat form view untuk edit artikel

<blog/app/views/artikel/edit.blade.php>

```
<!DOCTYPE html>
<html>
```



```

<head>

    <title>Edit Artikel</title>

</head>

<body>

    <h3>Edit Artikel</h3>

    {{ Form::model($artikel, array('action' =>
array('ArtikelController@update', $artikel->id), 'method'=>'PUT')) }}

        <p>Judul <br> {{ Form::text('judul',
Input::old('judul')) }}</p>

        {{ $errors->first('judul') }}

        <p>Isi <br> {{ Form::textarea('isi',
Input::old('isi')) }}</p>

        {{ $errors->first('isi') }}

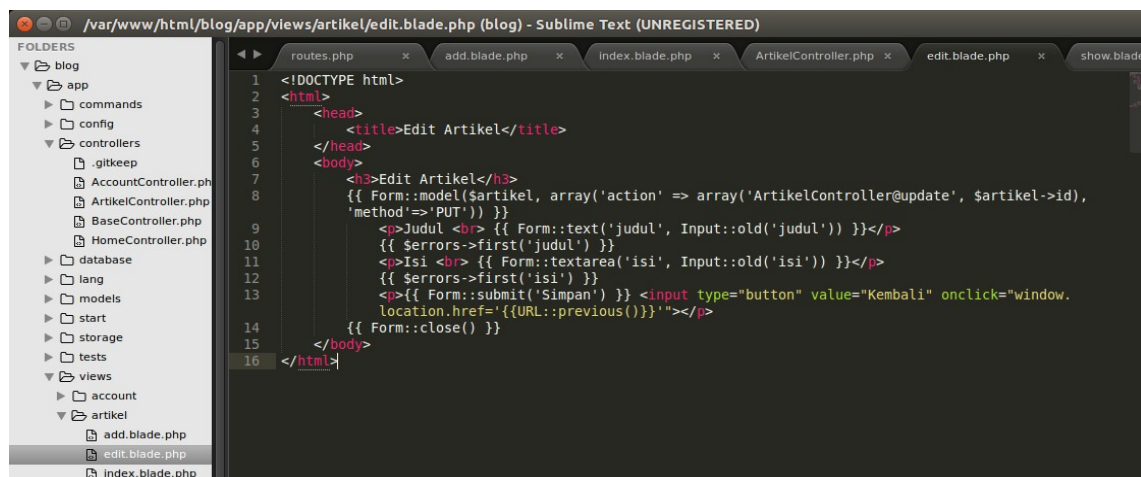
        <p>{{ Form::submit('Simpan') }} <input type="button"
value="Kembali"
onclick="window.location.href='{{URL::previous()}}'"></p>

        {{ Form::close() }}

    </body>

</html>

```



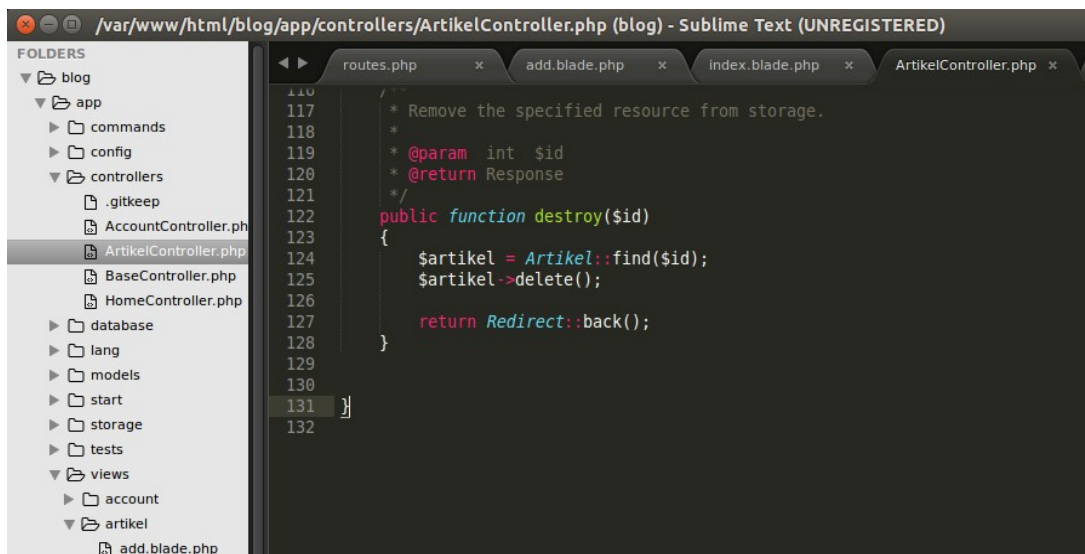
Membuat fungsi destroy artikel untuk menghapus artikel, function destroy pada ArtikelController.php

<blog/app/controllers/ArtikelController.php>

## Panduan Laravel PHP Framework

```
public function destroy($id)
{
    $artikel = Artikel::find($id);
    $artikel->delete();

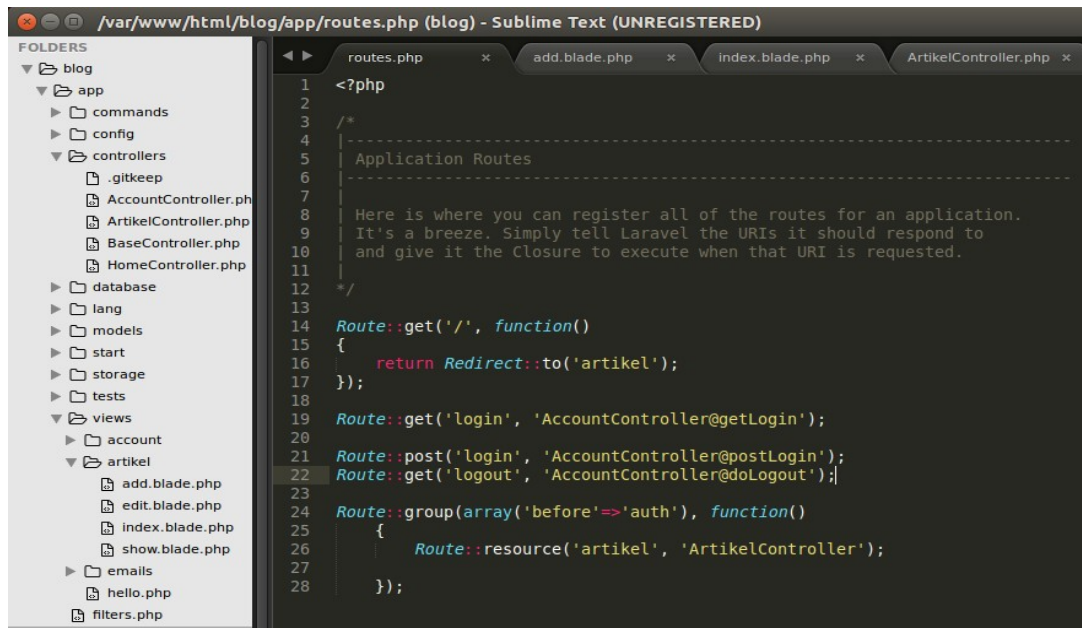
    return Redirect::back();
}
```



**Redirect base url ke artikel agar setelah login langsung masuk ke artikel**

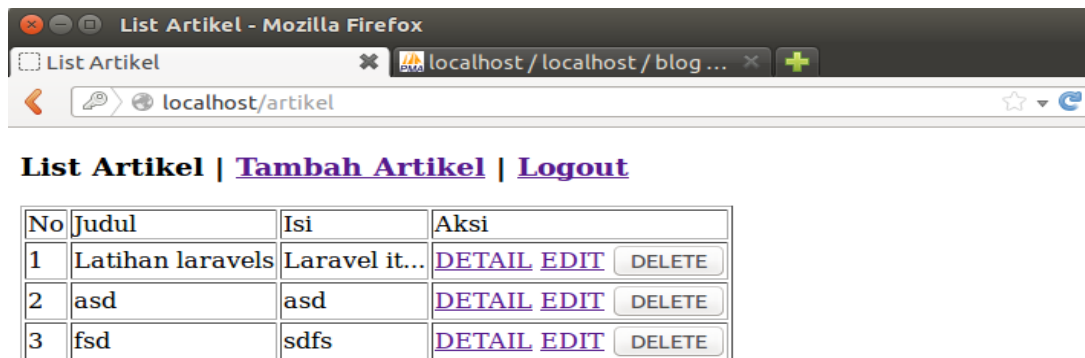
<blog/app/routes.php>

```
Route::get('/', function()
{
    return Redirect::to('artikel');
});
```



```
1 <?php
2
3 /*
4  * Application Routes
5  *
6  * Here is where you can register all of the routes for an application.
7  * It's a breeze. Simply tell Laravel the URIs it should respond to
8  * and give it the Closure to execute when that URI is requested.
9  */
10
11
12
13
14 Route::get('/', function()
15 {
16     return Redirect::to('artikel');
17 });
18
19 Route::get('login', 'AccountController@login');
20
21 Route::post('login', 'AccountController@postLogin');
22 Route::get('logout', 'AccountController@doLogout');
23
24 Route::group(array('before'=>'auth'), function()
25 {
26     Route::resource('artikel', 'ArtikelController');
27 });
28
```

### Tampilan list artikel



List Artikel | [Tambah Artikel](#) | [Logout](#)

No	Judul	Isi	Aksi
1	Latihan laravels	Laravel it...	<a href="#">DETAIL</a> <a href="#">EDIT</a> <a href="#">DELETE</a>
2	asd	asd	<a href="#">DETAIL</a> <a href="#">EDIT</a> <a href="#">DELETE</a>
3	fsd	sdfs	<a href="#">DETAIL</a> <a href="#">EDIT</a> <a href="#">DELETE</a>

### Tampilan tambah artikel



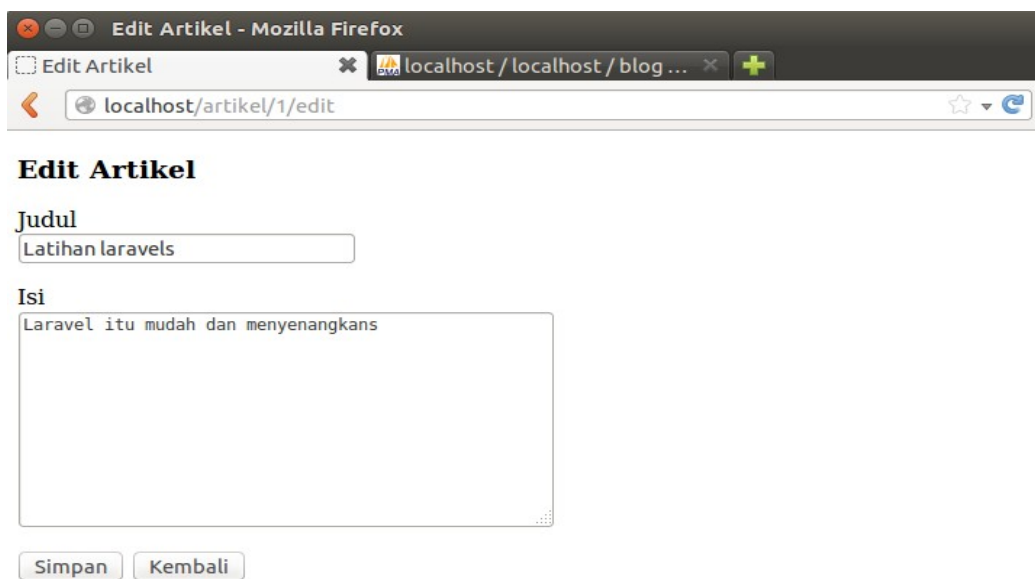
**Tambah Artikel** | [List Artikel](#) | [Logout](#)

Judul

Isi

Simpan Kembali

### Tampilan edit artikel



**Edit Artikel**

Judul

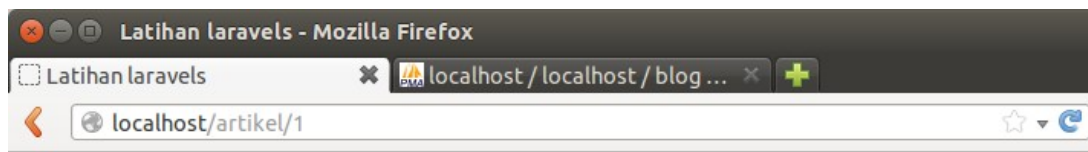
Latihan laravels

Isi

Laravel itu mudah dan menyenangkan

Simpan Kembali

### Tampilan detail artikel



## **Latihan laravels**

Laravel itu mudah dan menyenangkan

Waktu : 2014-12-03 13:44:55

Oleh : Aris Setyono

[Kembali](#)