

NOTA KELAS LARAVEL

TAHAP 1

Bersama Iszuddin Ismail



MAKLUMAN

Ini **bukanlah** rujukan lengkap untuk Laravel. Ini adalah nota rujukan kepada Kelas Laravel oleh Iszuddin Ismail di [KelasProgramming.com](https://kelasprogramming.com). Untuk rujukan paling sempurna, sila ke dokumentasi rasmi Laravel di <https://laravel.com/docs/>

Dapatkan Kombo Lengkap Video Pembelajaran PHP Dalam Bahasa Melayu Dari KelasProgramming.com

56 VIDEO, LEBIH 37 JAM PEMBELAJARAN



KelasProgramming.com menawarkan siri video pembelajaran PHP untuk membantu penuntut IPT, pegawai IT, programmer baru, mahupun mereka yang ingin beralih bidang atau membina aplikasi web sebagai hobi. Semua pembelajaran disampaikan dalam Bahasa Melayu.

Dapatkan di <https://kelasprogramming.com>

KANDUNGAN

Pengenalan	6
Laravel : MVC Framework	6
Persiapan VSCode	7
Pemasangan	9
Menggunakan Laragon	9
Pemasangan dengan Composer	12
Pemasangan dengan Laravel Installer	13
Aplikasi CLI Artisan	14
Struktur Fail dan Folder	14
Konfigurasi	16
Router	16
Redirect	17
View	17
Method / Verb	18
Anonymous Function	18
Parameter (Data dalam URL)	18
Controller	18
Model Binding	20
Nama Router	20
Grouping	21
Middleware	21
Resource (Complete RESTful Routes)	22
Menyemak Semua Router	23
Controller	23
Namespace	24
Nama Fail & Case Sensitive	24
Membina Controller Dengan Artisan	24
View & Blade	25
Memanggil View & Menghantar Data	25
Section & Layout	26
Fungsi & Logic	27
Borang (Forms)	29
Database Migration	30
Membina Fail Database Migration	30
Operasi Table	32
Operasi Column	32
Arahan Migrate	33

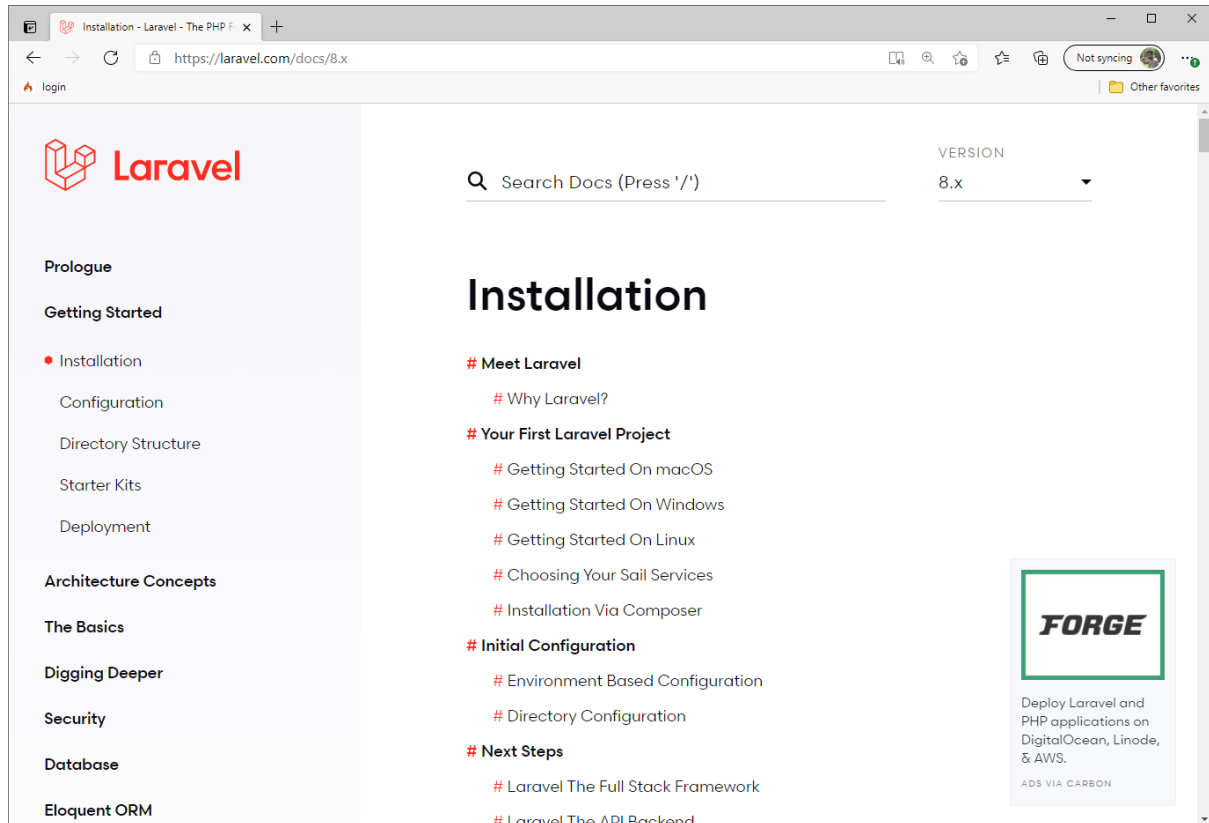
Model / Eloquent	33
Tetapan Database	33
Tetapan Model	34
Membina Model Dengan Artisan	35
Menggunakan Model	36
Mendapatkan Data	37
Eloquent & Query Builder	38
Menyimpan Rekod Baru	38
Mengemaskini Rekod	39
Membuang Data	39
One-to-Many Relationship	39
Many-to-Many Relationship	42
Eloquent Collection	44
Factory & Tinker	44
Factory	44
Tinker	45
Database & Query Builder	47
Melaksanakan Arahan SQL	47
Mengguna Nilai dari Variable	48
Pelbagai Database Connection	48
Memaparkan Rekod dengan Pagination	48
Menggunakan Pagination dengan Bootstrap CSS	50
Borang & Memproses Input Pengguna	51
Menerima Data Dari Borang	51
Validation	53
Session	56
Flash Data	58
Menghantar Email	58
Mailtrap.io	58
Tetapan	59
Pengenalan	59
Membina Mailable dengan Artisan	60
Membina View untuk Email	61
Menguji Email Dalam Browser	62
Menghantar Email	62
Menyertakan Data ke View	64
Authentication	66
Pakej Authentication Laravel 8	66
Laravel Starter Kit	67

Membina Authentication Ringkas	68
LANGKAH 1 : Membina table users	69
LANGKAH 2 : Pendaftaran	69
LANGKAH 3 : Login	72
LANGKAH 4 : Logout	75
LANGKAH 5 : Dashboard	75
LANGKAH 6 : Routes	76
Mencuba Fortify	77
Pemasangan	78
Konfigurasi	79
Membina Register View	82
Membina Login View	83
Operasi Lanjutan dengan Fortify	83
Membina Authentication dengan FortifyUI	84
Pemasangan	84
FortifyUI Preset : Hiasan CSS Pantas	86
Mencuba FortifyUIkit	87
Memasang Semua Ciri-Ciri Fortify	89
Operasi Authentication	91
Mendapatkan Maklumat Pengguna	91
Memeriksa Pengguna Telah Login	92
Menghantar Pengguna Belum Login Ke Laman Lain	92
Melindungi URL	93
Debugging	93
Fungsi dd()	93
Debugbar	94
Membaca Log	97
Rujukan Tambahan	98
Helper	98
Facade	98
Service Provider	98
Tarikh dan Masa dengan Carbon	98

Pengenalan

Rujukan utama untuk Laravel adalah di laman web rasminya:

- <https://laravel.com/docs>



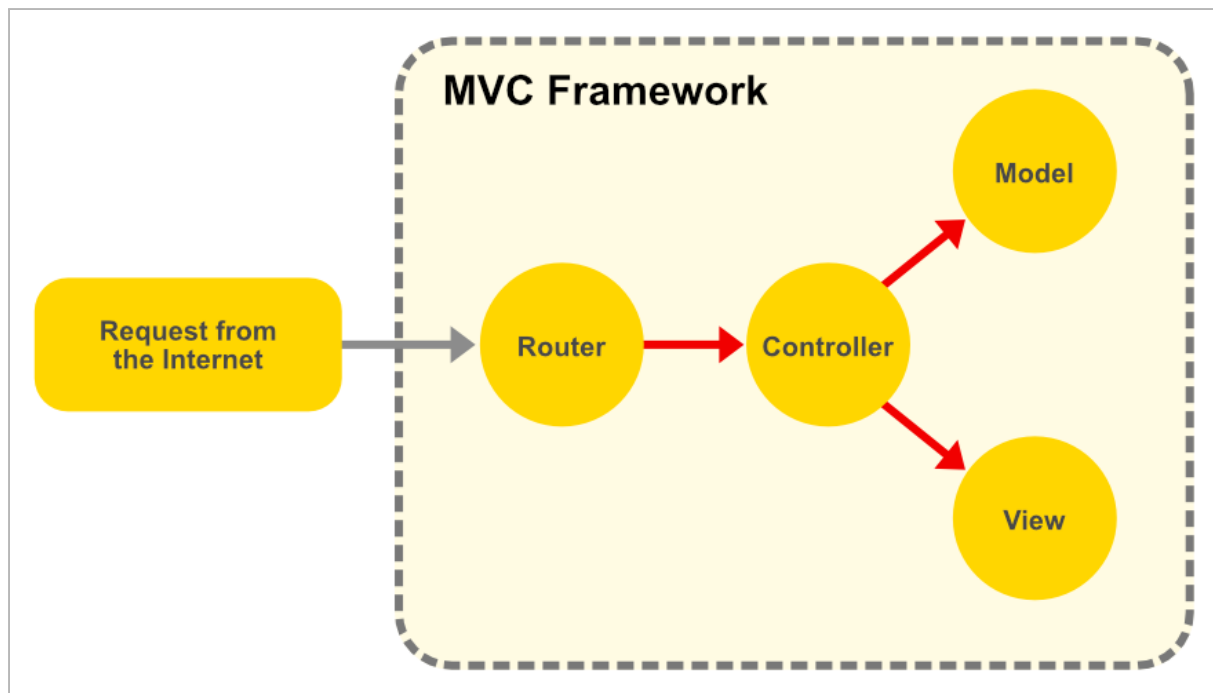
Laravel : MVC Framework

MVC bermaksud Model-View-Controller. Berikut adalah fungsi utama komponen ini :

- **Model** : Komunikasi dengan *database*.
- **View** : Menyimpan kod HTML dan antaramuka untuk *browser*.
- **Controller** : Menyimpan kod PHP untuk pelbagai operasi and logik.

Lagi satu komponen penting yang sepatutnya juga ditekankan dalam MVC adalah Route.

- **Router** : Mengandungi konfigurasi URL dan pemprosesnya.

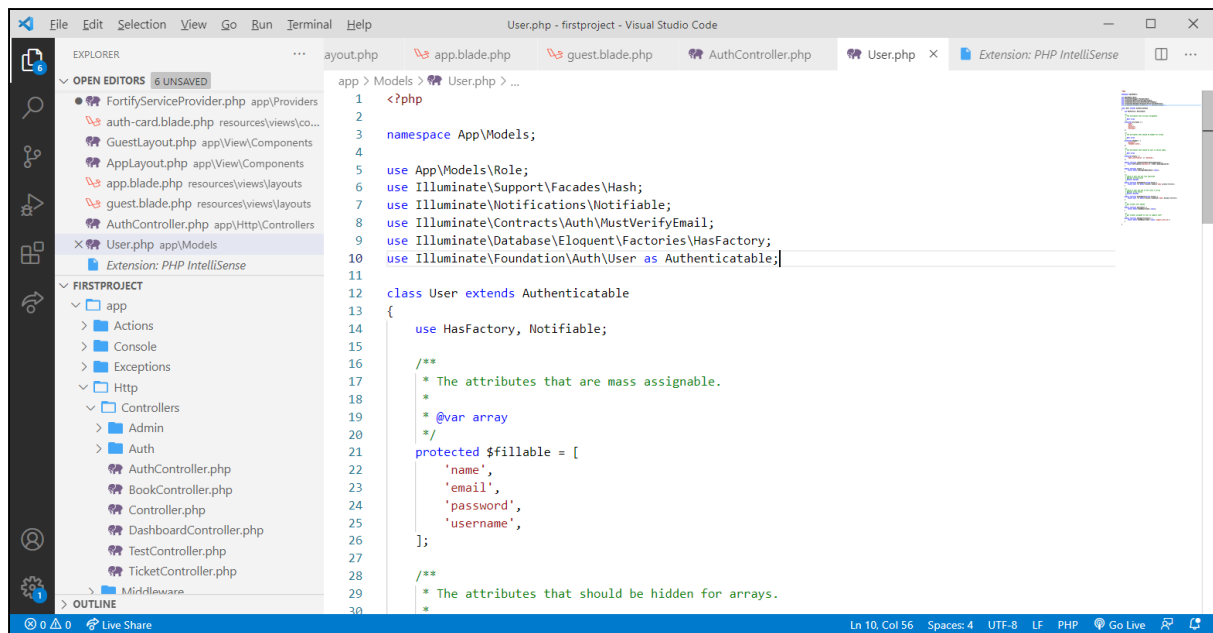


- Aplikasi bermula di Router.
- Router menentukan Controller mana yang akan digunakan.
- Controller akan menggunakan Model dan View.

Persiapan VSCode

Kami mencadangkan VSCode sebagai *code editor* ataupun IDE anda untuk kerja-kerja pembangunan web dengan Laravel.

Dapatkan VS Code di <https://code.visualstudio.com/Download>

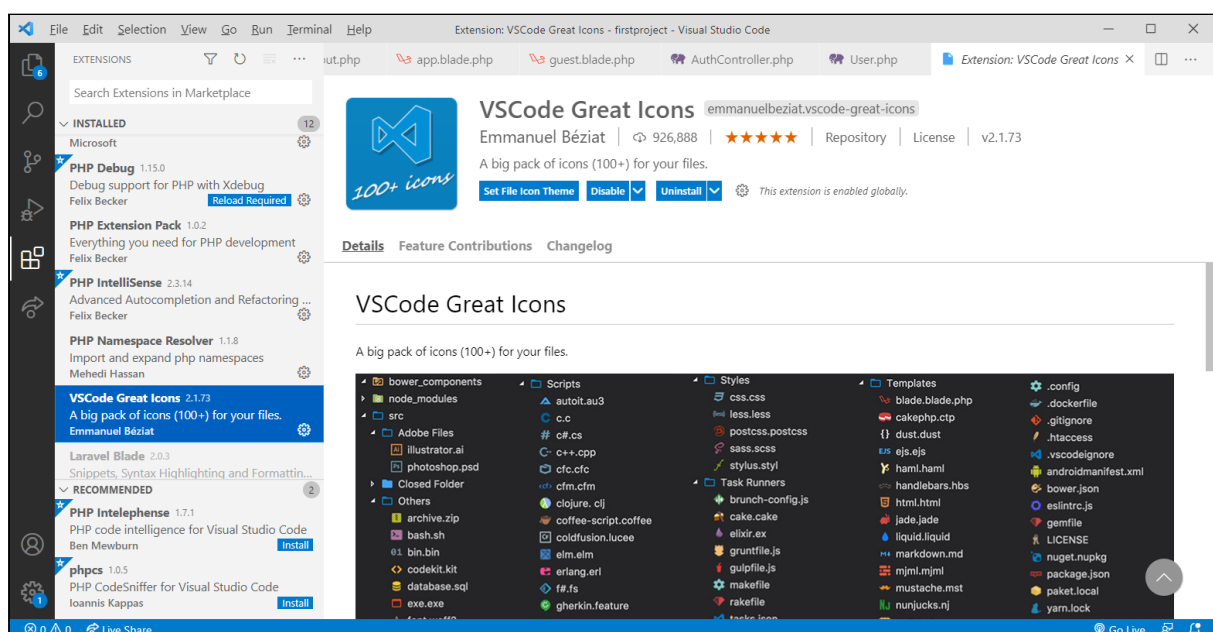


Dari ikon di sebelah kiri, pergi ke ikon bergambar kotak



Buat carian dan dapatkan **extensions** berikut.

1. **PHP Intellisense** - ianya membantu dengan autocomplete yang pelbagai untuk PHP.
2. **PHP Namespace Resolver** - ianya membantu untuk import library yang diperlukan.
3. **Laravel goto view** - lebih mudah untuk ke **View** dari **Controller**.
4. **VSCode Great Icon** - lebih mudah mengenalpasti jenis fail dengan ikon.

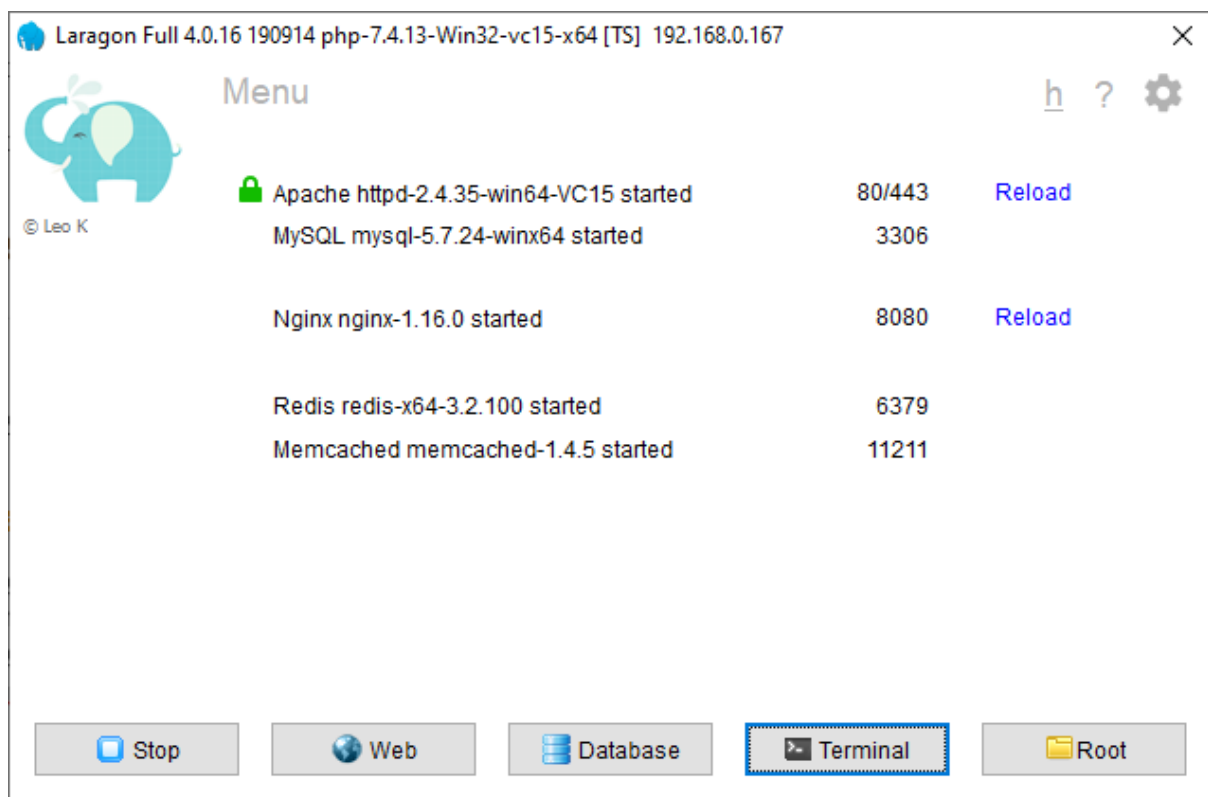


Pemasangan

Menggunakan Laragon

Laragon memudahkan kerja-kerja pembangunan PHP di sistem operasi Windows. Ianya hanya digunakan untuk pembangunan dan bukan sebagai webserver apabila aplikasi siap kelak. Laragon adalah sebuah pakej lengkap dengan webserver (Apache), PHP, *database* MySQL, Composer, NPM dan lain-lain perisian.

Laragon boleh didapati dari <https://laragon.org/download/index.html>



Kemaskini Composer

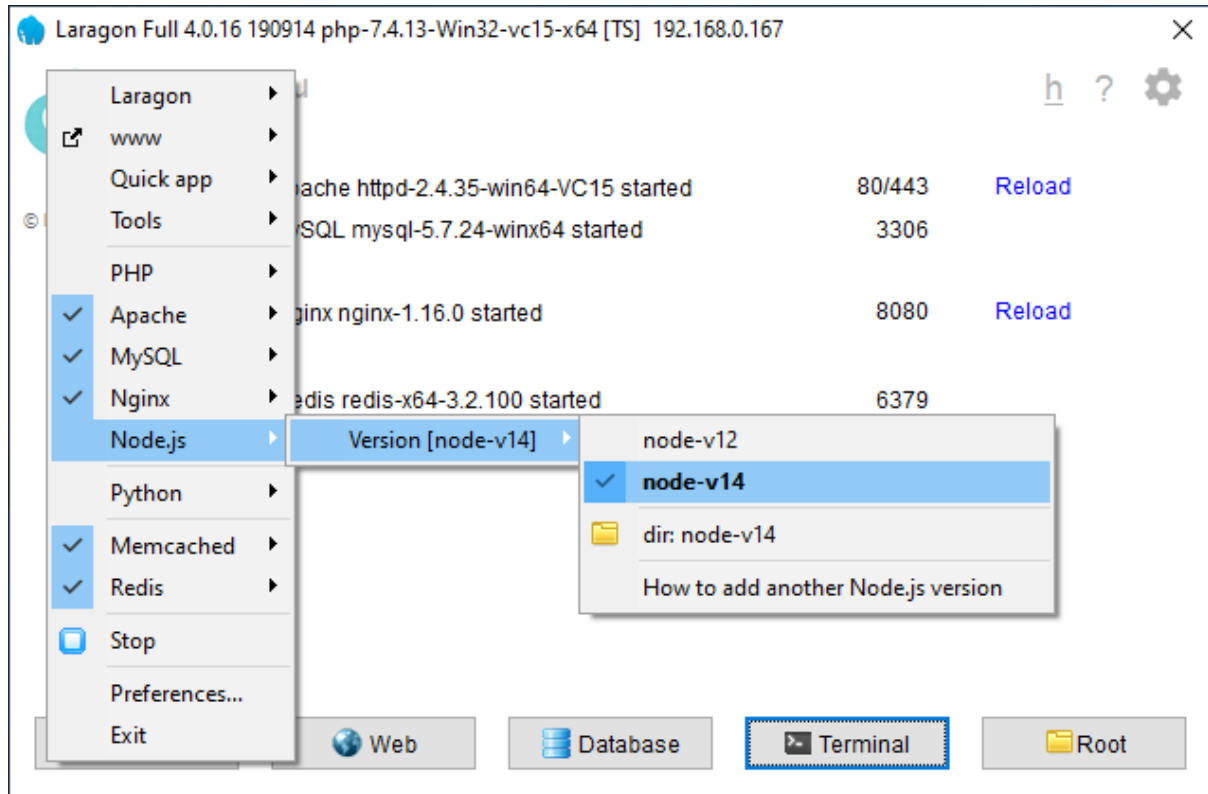
Sebelum pemasangan Laravel, sila kemaskini versi Composer. Buka Terminal daripada Laragon dan laksanakan arahan ini :

```
composer selfupdate
```

Kemaskini NPM / NodeJS

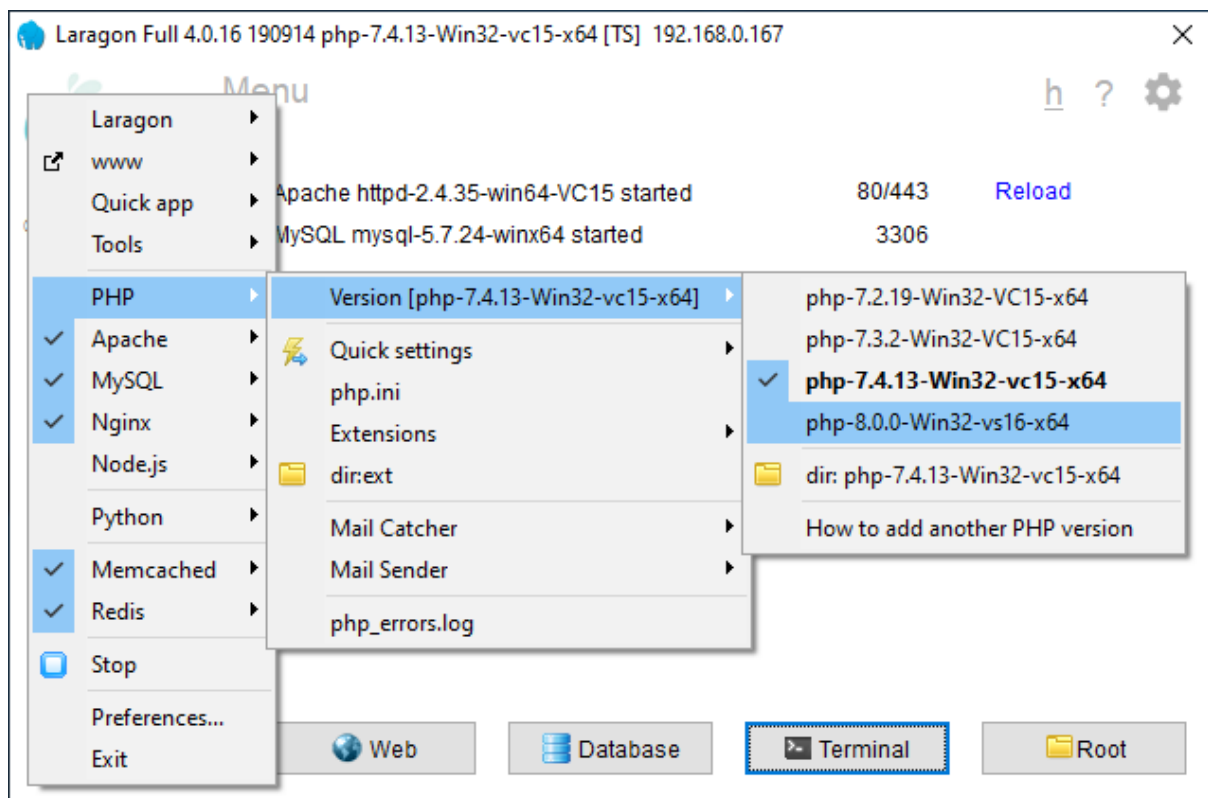
1. Dapatkan NodeJS terkini di <https://nodejs.org/en/download/>
2. Muat-turun versi LTS, Window Binary.

3. Extract fail ZIP ke *folder* C:\laragon\bin\nodejs (setiap versi di dalam *folder* sendiri)
4. Dari Laragon, *right-click* dan teruskan ke Node.js > Version.
5. Pilih versi Node.js yang telah dimuat-turun tadi.



Kemaskini Versi PHP

1. Dapat versi PHP yang terkini di <https://windows.php.net/download>
2. Muat-turun versi yang dikehendaki, namun dicadangkan dapatkan versi Thread Safe untuk kegunaan bersama Laragon.
3. Extract fail ZIP ke folder D:\laragon\bin\php (setiap versi di dalam *folder* sendiri)
4. Dari Laragon, *right-click* dan terus ke PHP > Version.
5. Pilih versi PHP yang dikehendaki.



Pemasangan Laravel dengan Laragon

Setelah mengemaskini PHP, Composer dan Node.JS, kita boleh memulakan projek Laravel yang baru. Dicadangkan, menggunakan versi PHP 7.4 atau lebih baru.

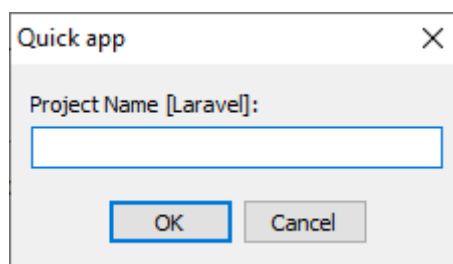
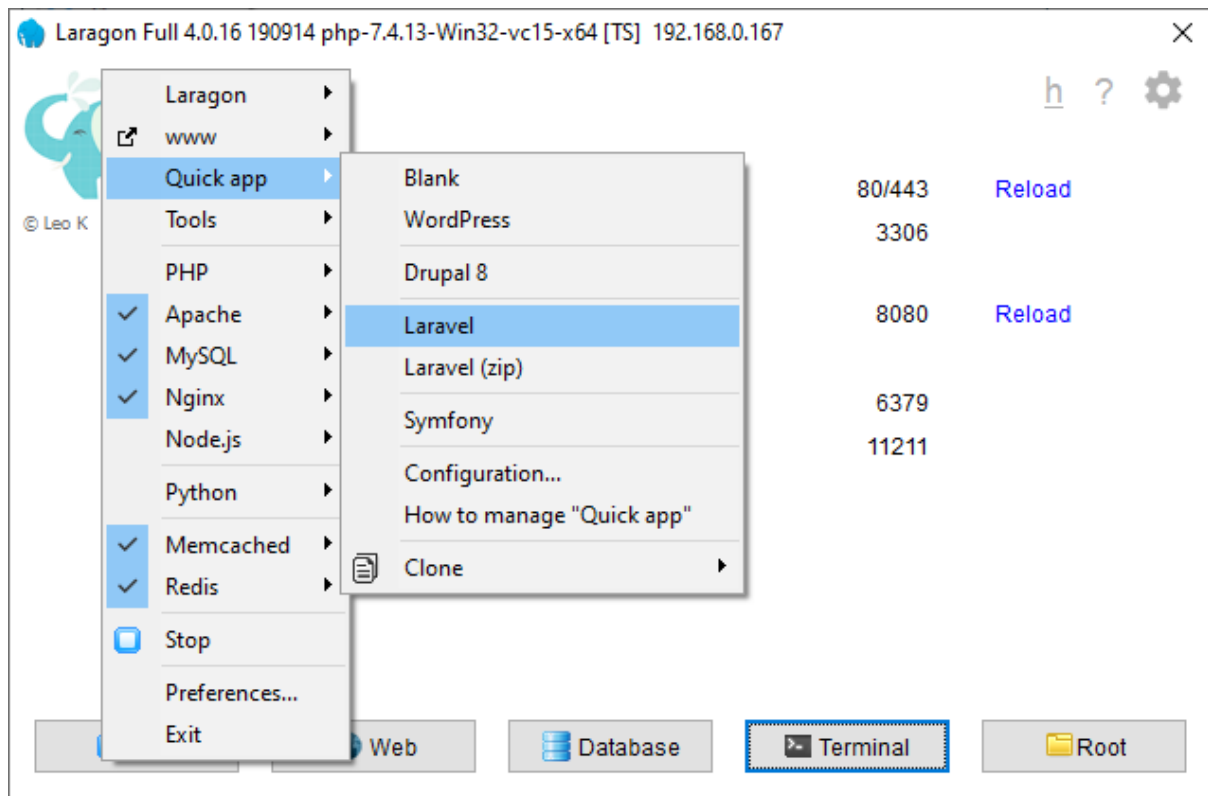
1. Dari Laragon, *right-click*.
2. Pilih Quick app > Laravel
3. Masukkan nama projek, seperti ***aplikasi***
4. Klik butang OK.

Apabila terdapat notifikasi untuk akses Administrator, klik Yes.

Apabila Laragon selesai membuat pemasangan, projek boleh diakses dengan *browser* melalui alamat berikut

- <http://aplikasi.test>

Laragon juga telah membina satu database MySQL dengan nama yang sama seperti projek, *aplikasi*.



Pemasangan dengan Composer

Pemasangan Composer tidak diperlukan sekiranya anda menggunakan Laragon. Tetapi pasti Composer telah dipasang dengan versi yang terkini.

Composer adalah aplikasi *command line* untuk pengurusan pakej PHP. Dapatkan Composer sekiranya belum tersedia di komputer.

- <https://getcomposer.org/>

Setelah mempunyai Composer, laksanakan berikut arahan satu persatu di *Terminal* atau *Command Prompt*.

```
composer create-project laravel/laravel aplikasi
cd aplikasi
```

```
php artisan serve
```

Araha ini akan membina satu *folder* baru bernama **aplikasi**. Arahan php artisan serve akan memulakan *webserver* untuk pembangunan.

Pemasangan dengan Laravel Installer

Pertama, Composer diperlukan untuk pemasangan Laravel Installer seperti berikut:

```
composer global require laravel/installer
```

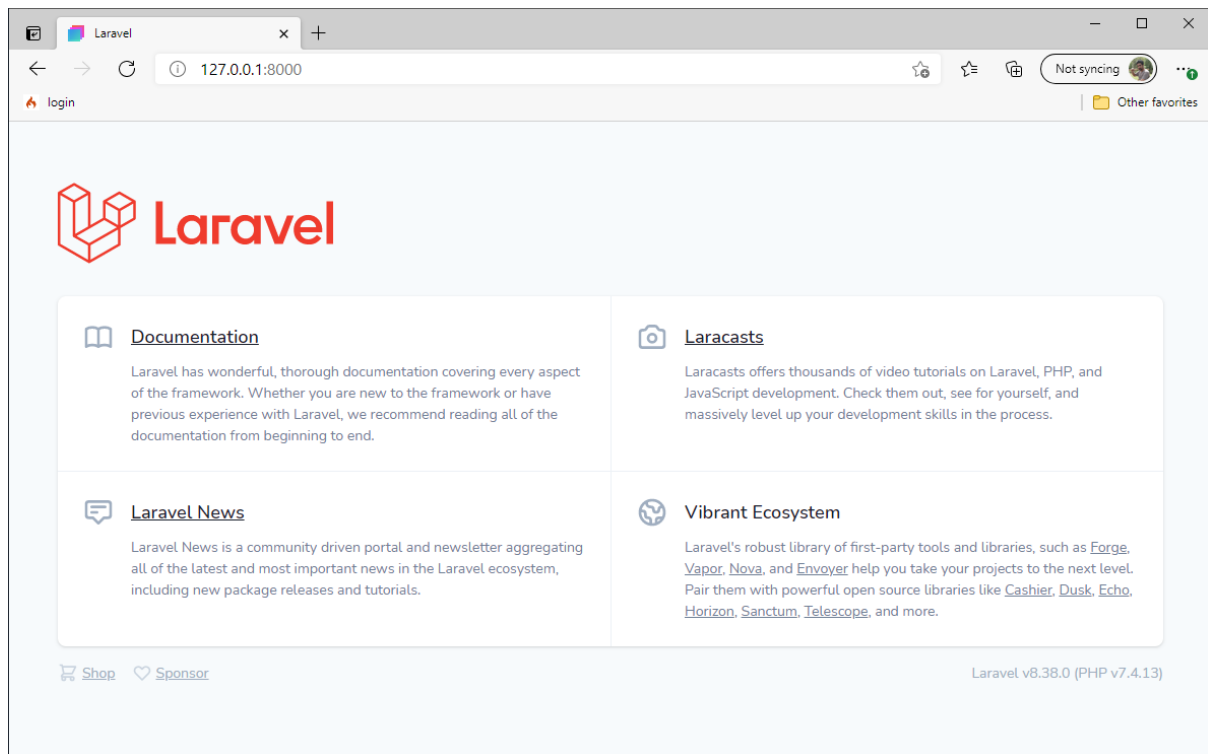
Kemudian laksanakan arahan berikut arahan satu persatu untuk pemasangan Laravel.

```
laravel new aplikasi  
cd aplikasi  
php artisan serve
```

Arahan php artisan serve akan memulakan webserver untuk pembangunan. ***Ini tidak diperlukan sekiranya anda menggunakan Laragon.***

Aplikasi Laravel boleh mula diuji di *browser* melalui alamat berikut :

- <http://127.0.0.1:8000>



Aplikasi CLI Artisan

Laravel didatangkan dengan aplikasi CLI bernama Artisan. Aplikasi ini memudahkan *programmer* menulis kod seperti *Controller*, *Model*, dan lain-lain. Ia juga mempunyai fungsi-fungsi lain.

Arahan untuk Artisan mesti dilaksanakan dalam folder paling atas untuk projek.

Arahan berikut akan menyenaraikan semua fungsi-fungsi Artisan.

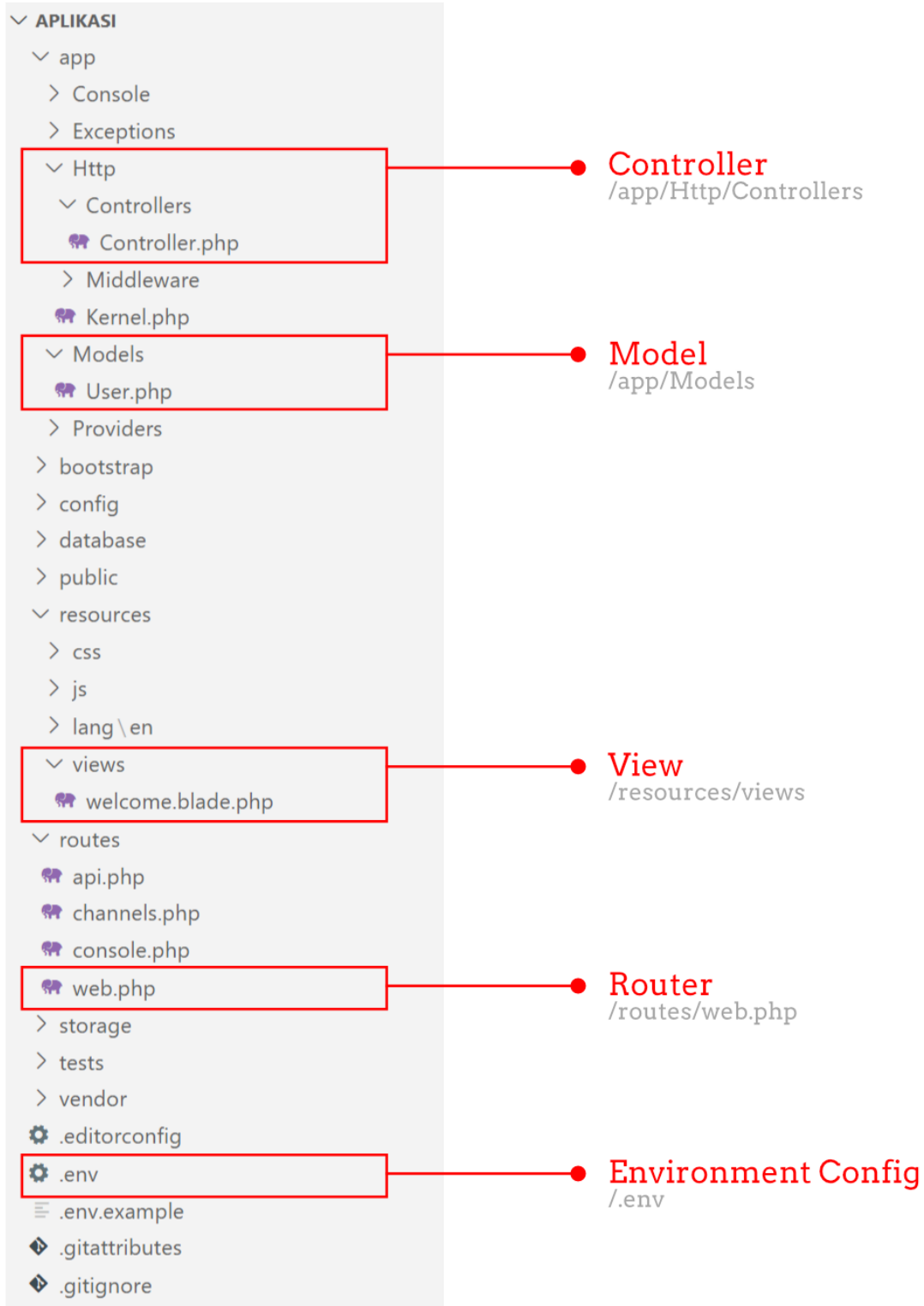
Arahan PHP Artisan

```
$ php artisan
```

Struktur Fail dan Folder

Aplikasi Laravel yang dipasang menyediakan fail dan *folder* seperti berikut.

Dalam mempelajari Laravel sebagai sebuah *framework* MVC, perhatikan lokasi di mana Model, View, Controller and juga Router disimpan.



Konfigurasi

Laravel perlu ditetapkan beberapa tetapan sebelum kita boleh bermula. Ada dua tempat di mana Konfigurasi boleh dilaksanakan.

1. Fail .env di folder /aplikasi atau paling tinggi dalam projek
2. Fail-fail dalam folder /config

Tetapan di fail .env akan sentiasa diutamakan berbanding tetapan di dalam folder /config. Sekiranya tiada fail .env, gunakan fail .env.example dan tukarkan namanya kepada .env.

Berikut adalah beberapa tetapan utama :

```
APP_NAME=Aplikasi
APP_ENV=local
APP_KEY=base64:fdMu/5IeTVePCBJo6PN6y1JIjWMd9GP8dxajBkqTBM8=
APP_DEBUG=true
APP_URL=http://aplikasi.test

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=aplikasi
DB_USERNAME=root
DB_PASSWORD=
```

Router

Dokumentasi penuh di <https://laravel.com/docs/8.x/routing>.

Router menentukan URL aplikasi dan bagaimana ianya diproses. Fail untuk Router boleh didapati di folder /routes.

routes	
api.php	api.php - untuk pembangunan API
channels.php	channels.php - untuk Websocket
console.php	console.php - untuk aplikasi CLI /Terminal
web.php	web.php - untuk aplikasi web

Untuk pembangunan web, kita hanya gunakan web.php.

Berikut adalah beberapa ciri-ciri dan struktur Router :

1. Redirect
2. View
3. Method / Verb
4. Anonymous function
5. Parameter
6. Controller
7. Model Binding
8. Nama Router
9. Grouping (pengasingan)

Redirect

Format

```
Route::redirect('<url>', '<destinasi>', <kod status>);
```

Contoh

```
Route::redirect('/here', '/there', 301);
```

Kod status adalah tidak wajib. Sekiranya tidak dinyatakan, ia akan menggunakan kod 302.

View

Format

```
Route::view('<url>', '<view>', <data>);
```

Contoh

```
Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

View **welcome** merujuk kepada fail di /resources/views/welcome.blade.php.

Contoh

```
Route::view('/welcome', 'page.welcome', ['name' => 'Taylor']);
```

View **page.welcome** merujuk kepada fail di /resources/views/page/welcome.blade.php.

Method / Verb

Ini merujuk kepada HTTP Protocol. Anda boleh menetapkan Router khusus untuk HTTP Protocol tertentu seperti GET, POST, PUT, PATCH, DELETE, OPTIONS.

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Anonymous Function

Untuk pemrosesan ringkas, kita boleh gunakan *Anonymous Function*.

```
Route::get('/user', function () {
    return 'Hello';
});
```

Parameter (Data dalam URL)

```
Route::get('/user/{nama}', function ($nama) {
    echo 'Hello '.$nama;
});
```

Dengan format di bawah, parameter nama menjadi tidak wajib.

```
Route::get('/user/{nama?}', function ($nama = 'saudara') {
    echo 'Hello '.$nama;
});
```

Controller

Router boleh menetapkan agar Controller yang membuat pemrosesan untuk UR.

Format

```
Route::get('<url>', [<NamaController>::class, '<nama_fungsi>']);
```

Contoh: dalam /routes/web.php

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Pastikan menggunakan **use** untuk menggunakan Controller.

Dalam `/app/Http/Controllers/UserController.php`

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * public function index()
     * {
     *     return 'Hello';
     * }
}
```

Dari contoh, perhatikan **index** dari `web.php` dan juga nama fungsi di function **index()**.

Contoh: dalam `/routes/web.php`

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Dalam `/app/Http/Controllers/UserController.php`

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * public function index()
     * {
     *     return 'Hello';
     * }
}
```

Berikut adalah bagaimana menggunakan Router dengan Controller dan menghantar data parameter.

Contoh: dalam /routes/web.php

```
use App\Http\Controllers\UserController;

Route::get('/user/{nama}', [UserController::class, 'index']);
```

Dalam /app/Http/Controllers/UserController.php

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * public function index($nama)
     * {
     *     return 'Hello '.$nama;
     * }
}
```

Model Binding

Model merujuk kepada data dari *database*. Model boleh disertakan sebagai parameter tanpa kod tambahan.

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
    return $user->email;
});
```

Nama Router

Memberikan nama kepada router memudahkan pengaturcaraan kelak, di dalam Controller mahupun di dalam View.

Contoh: dalam /routes/web.php

```
use App\Http\Controllers\UserController;

Route::get('/', [UserController::class, 'index'])->name('home');
```

Contoh: dalam /resources/views/template.php

```
<a href="{ route('home') }">Utama</a>
```

Perhatikan **home** di *web.php* dan **home** di *template.php*.

Grouping

Beberapa definisi router boleh dikumpulkan bersama dan diasingkan daripada yang lain.

Contoh: Grouping dengan nama

```
Route::name('admin.')->group(function () {
    Route::get('/users', function () {
        // Route assigned name "admin.users"...
    }->name('users'));
});
```

Dengan contoh di atas, semua router dalam kumpulan ini akan dinamakan dengan “admin.” di hadapannya.

Contoh: Grouping dengan prefix

```
Route::prefix('admin')->name('admin.')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    }->name('users'));
});
```

Dengan contoh di atas, Route di dalam boleh diakses dengan URL **/admin/users**.

Middleware

Middleware adalah kod ataupun *library* yang dilaksanakan sebelum proses tiba ke Controller. Middleware yang terbina dalam Laravel adalah seperti auth, yang membantu menyemak sama ada pengguna telah login ataupun tidak.

```
Route::middleware('auth')->group(function () {
    Route::get('/admin', function () {
        //...
    });

    Route::get('/admin/user', function () {
        //...
    });
});
```

Dalam contoh di atas, *middleware* **auth** akan dilaksana untuk kedua-dua Router dalam kumpulan.

Resource (Complete RESTful Routes)

Resource Routes membantu membina beberapa syarat URL yang menepati konsep RESTful.

Resource Routes

```
Route::resource('photos', PhotoController::class);
```

Secara automatik ini juga adalah ringkasan kepada tujuh definisi router berikut

```
1 Route::get('/photos', [PhotoController::class,
  'index'])->name('photo.index');
2 Route::get('/photos/create', [PhotoController::class,
  'create'])->name('photo.create');
3 Route::post('/photos', [PhotoController::class,
  'store'])->name('photo.store');
4 Route::get('/photos/{photo}', [PhotoController::class,
  'show'])->name('photo.show');
5 Route::get('/photos/{photo}/edit', [PhotoController::class,
  'edit'])->name('photo.edit');
6 Route::put('/photos/{photo}', [PhotoController::class,
  'update'])->name('photo.update');
7 Route::delete('/photos/{photo}', [PhotoController::class,
  'destroy'])->name('photo.destroy');
```

Membina Controller untuk definisi Resource Router dipermudahkan dengan arahan PHP Artisan. Laksana arahan berikut di Terminal.

Arahan PHP Artisan : Membina Resource Controller

```
$ php artisan make:controller PhotoController --resource
```

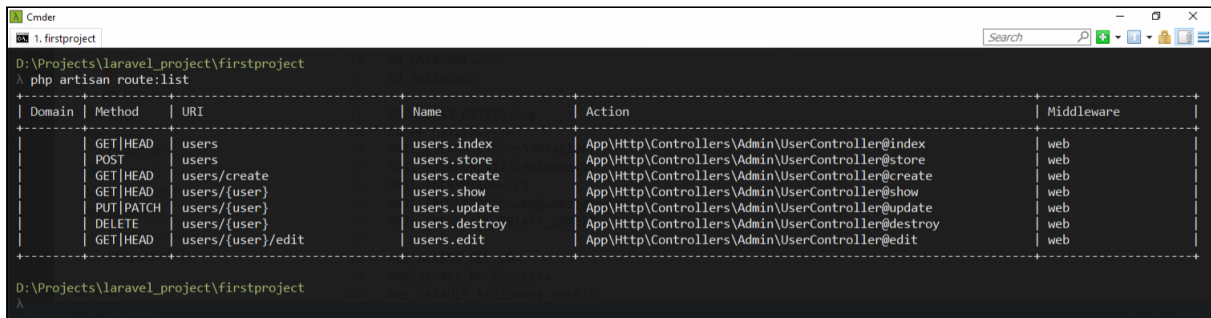
Arahan ini akan membina file *Controller /app/Http/Controllers/PhotoController.php*.

Fail *Controller* baru ini disediakan siap dengan fungsi-fungsi yang menepati definisi Resource Router. Ianya boleh terus diedit dan dikemaskini mengikut kesesuaian.

Menyemak Semua Router

Kita boleh menyemak semua router yang telah dibina dengan arahan Artisan berikut :

```
$ php artisan route:list
```



Domain	Method	URI	Name	Action	Middleware
	GET HEAD	users	users.index	App\Http\Controllers\Admin\UserController@index	web
	POST	users	users.store	App\Http\Controllers\Admin\UserController@store	web
	GET HEAD	users/create	users.create	App\Http\Controllers\Admin\UserController@create	web
	GET HEAD	users/{user}	users.show	App\Http\Controllers\Admin\UserController@show	web
	PUT PATCH	users/{user}	users.update	App\Http\Controllers\Admin\UserController@update	web
	DELETE	users/{user}	users.destroy	App\Http\Controllers\Admin\UserController@destroy	web
	GET HEAD	users/{user}/edit	users.edit	App\Http\Controllers\Admin\UserController@edit	web

Controller

Dokumentasi penuh di <https://laravel.com/docs/8.x/controllers>.

Controller adalah di mana pemprosesan aplikasi berlaku. File Controller disimpan dalam folder `/app/Http/Controllers`.

Contoh : Controller, `/app/Http/Controllers/UserController.php`

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class UserController extends Controller
{
    public function show($id)
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

Ini mungkin digabungkan dengan router seperti berikut :

Contoh : Router dalam `/resources/web.php`

```
use App\Http\Controllers\UserController;

Route::get('/user/{id}', [UserController::class, 'show']);
```

Namespace

Laravel menggunakan konsep *Object Oriented Programming* dan *Namespace*. *Namespace* mengganti kaedah `include()` dan `require()` untuk menggunakan kod dari fail lain dalam aplikasi PHP. Untuk memaklumkan kepada bahagian mengenali kod yang ditulis, ianya mesti disertakan dengan *Namespace*.

- **Namespace untuk Controller** : `App\Http\Controllers;`
- **Lokasi Folder** : `/app/Http/Controllers`

Sekiranya ingin menyimpan Controller di dalam folder lain, kita boleh gunakan seperti contoh berikut.

- **Namespace untuk Controller** : `App\Http\Controllers\Admin;`
- **Lokasi Folder** : `/app/Http/Controllers/Admin`

Nama Fail & Case Sensitive

Berhati-hati dengan nama yang diberikan untuk fail dan folder. Nama fail merujuk kepada nama Controller yang digunakan. Sistem operasi Windows mempunyai sifat tidak *case-sensitive*. Tetapi sekiranya kita tak beri perhatian kepada huruf besar dan kecil, ini mungkin memberi masalah apabila aplikasi dilancarkan di server dengan sistem operasi Linux.

Keadah yang digunakan untuk menamakan *Controller* adalah seperti berikut :

- Menggunakan camel-case, seperti `NamaController`
- Nama fail mesti sama dengan nama class Controller, seperti `NamaClass.php`
- Bersifat single (bukan plural), seperti `UserController`, bukan `UsersController`

Membina Controller Dengan Artisan

Gunakan arahan berikut di terminal untuk membina rangka Controller dengan Artisan.

Arahan PHP Artisan : Membina Controller

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

Arahan PHP Artisan : Membina Resource Controller


```
$ php artisan make:controller PhotoController -r
```

Arahan Resource Controller boleh dipadankan terus dengan Resource Route sebelum ini. Ia akan terus membina Controller bersama fungsi-fungsi yang sepadan dengan Route yang dibina.

View & Blade

Dokumentasi penuh di <https://laravel.com/docs/8.x/blade>.

View adalah seperti sistem template. Dalam Laravel, ianya hadir dengan sistem Blade. View boleh digunakan dari Controller dan View.

- View disimpan di *folder /resources/views*
- Nama fail untuk view diakhiri dengan “.blade.php” seperti “user.blade.php”.
- View boleh disimpan di dalam *folder* lebih dalam seperti */resources/views/templates/app.php*
- View dipanggil dengan fungsi **view()**
- Untuk view di dalam folder yang lebih dalam, tanda titik (.) digunakan bersama fungsi view() seperti view('templates.app')

Memanggil View & Menghantar Data

Contoh : Controller, */app/Http/Controllers/UserController.php*

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class UserController extends Controller
{
    public function show($nama)
    {
        return view('user.profile',
            // data dihantar melalui associative array
            [
                'nama' => $nama
            ]
        );
    }
}
```

Contoh, View, */resources/views/user/profile.blade.php*

```
<?php
```

```
Hello, {{ $nama }}.
```

Perhatikan view yang dipanggil **“user.profile”**. Dengan ini, fail view adalah di **“/resources/views/user/profile.blade.php”**

Perhatikan juga nama fail untuk view ada **“profile.blade.php”**

Akhir sekali, di dalam fail view, *variable* dipanggil dalam kurungan **{{ ... }}**.

Section & Layout

Kita boleh membina *layout* utama untuk aplikasi dengan menggunakan **@yield()**.

Contoh : View di /resources/views/layouts/app.blade.php

```
1 <html>
2   <head>
3     <title>Aplikasi</title>
4     <link rel="stylesheet" href="/css/style.css">
5   </head>
6   <body>
7     <h1>Aplikasi</h1>
8     <hr>
9     <div class="container">
10      <div class="row">
11        <div class="col-12">
12          @yield('content')
13        </div>
14      </div>
15    </div>
16  </body>
17 </html>
```

Sekarang kita boleh menggunakan view in dari view yang lain. Di sini kita akan gunakan **@extends**, **@section** dan **@endsection**.

Contoh : View di /resources/views/user/profile.blade.php

```
1 @extends('layouts.app')
2
3 @section('content')
4 <h3>User Profile</h3>
5 <p>{{ $nama }}</p>
6 @endsection
```

Perhatikan `@extends('layouts.app')` merujuk kepada fail view di `/resources/views/layouts/app.blade.php`

Apa yang terkandung di antara `@section` dan `@endsection` akan menggantikan `@yield`.

Sekarang view boleh digunakan bersama Controller seperti contoh di bawah.

Contoh : Controller, `/app/Http/Controllers/UserController.php`

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class UserController extends Controller
{
    public function show($nama)
    {
        return view('user.profile',
            // data dihantar melalui associative array
            [
                'nama' => $nama
            ]);
    }
}
```

Fungsi & Logic

Logic If

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
```

```
I don't have any records!  
@endif
```

Isset dan Empty

```
@isset($records)  
    // $records is defined and is not null...  
@endisset
```

```
@empty($records)  
    // $records is "empty"...  
@endempty
```

Memeriksa Status Login

```
@auth  
    // The user is authenticated...  
@endauth
```

```
@guest  
    // The user is not authenticated...  
@endguest
```

Gelung (Loop)

```
@for ($i = 0; $i < 10; $i++)  
    The current value is {{ $i }}  
@endfor
```

```
@foreach ($users as $user)  
    <p>This is user {{ $user->id }}</p>  
@endforeach
```

```
@forelse ($users as $user)  
    <li>{{ $user->name }}</li>  
@empty  
    <p>No users</p>  
@endforelse
```

```
@while (true)  
    <p>I'm looping forever.</p>  
@endwhile
```

@include membenarkan kita menarik view lain ke dalam view semasa

```
<div class="pull-right">  
    @include('layouts.sidebar')  
</div>
```

Memaparkan kandungan HTML

```
Kandungan buku <br> {!! $halaman1 !!}.
```

Memaparkan URL daripada router dengan *route()*

```
<a href="{{ route('home') }}">UTAMA</a>
```

Membuat komen dalam Blade

```
{{!-- This comment will not be present in the rendered HTML --}}
```

Borang (Forms)

Dalam Laravel, data dari *form* dilindungi dengan CSRF (cross-site-request-forgery). Ini adalah ciri-ciri keselamatan yang terbina. Oleh demikian ada beberapa perkara yang perlu diberi perhatian bila menggunakan form.

@csrf

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

Borang dalam Laravel mesti disertai dengan @csrf yang akan menjadi token keselamatan.

@method

```
<form action="/foo/bar" method="POST">
    @method('PUT')
    ...
</form>
```

Secara am, dari *browser*, borang hanya boleh menghantar protokol POST dan GET. Dengan Laravel, kita boleh menggunakan juga protokol lain seperti PUT, PATCH, DELETE juga.

@error()

```
<input type="text" name="title">
    @error('title')
        <div class="alert alert-danger">{{ $message }}</div>
    @enderror
```

`@error()` digunakan bersama validation di dalam Controller. Ianya membantu mengenalpasti ralat dalam input pengguna dan memaparkan mesej ralat tersebut.

Database Migration

Dokumentasi rasmi di <https://laravel.com/docs/8.x/migrations>

Mungkin kita pernah melakukan database migration semasa membina *Authentication*. Database migration berfungsi untuk berikut :

- Merekodkan perubahan database, seperti *table* baru, *column* baru, dll.
- Memudahkan programmer dalam pasukan mendapatkan struktur database.
- Membantu mengembalikan database ke keadaan asal sebelum berlaku perubahan.

Berikut adalah arahan database migration :

```
$ php artisan migrate
```

Membina Fail Database Migration

Arahan berikut arahan untuk membina fail database migration.

```
$ php artisan make:migration <name_fail_migration>
```

Nama fail migration biasanya merujuk kepada operasi yang ingin dilaksanakan. Contoh, `create_table_books`.

```
$ php artisan make:migration create_books_table
```

Ini akan menghasilkan fail dalam folder **/database/migrations**. Berikut adalah contoh fail yang dihasilkan oleh Artisan.

Perhatikan fungsi `up()` dan `down()`. Fungsi **`up()`** digunakan apabila arahan **`php artisan migrate`** dilakukan. Fungsi `down()` digunakan apabila arahan **`php artisan migrate:rollback`** dilakukan.

Di dalam setiap fungsi ini, adalah arahan yang akan dilaksanakan.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateBooks extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table)
        {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('books');
    }
}
```

Operasi Table

Untuk membina table baru atau mengubahsuai, facade Schema digunakan. Pastikan kedua *namespace* ini telah diimport dengan **use**.

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

Membina *table* baru

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->timestamps();
});
```

Mengubahsuai *table*

```
Schema::table('users', function (Blueprint $table) {
    $table->integer('votes');
});
```

Menukar nama *table*

```
Schema::rename($from, $to);
```

Membuang *table*

```
Schema::drop('users');

Schema::dropIfExists('users');
```

Operasi Column

Perubahan kepada column berlaku di dalam *closure* apabila Schema dipanggil. Kita akan menggunakan objek **\$table** yang diberikan.

Menambah column baru

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->decimal('price', 8, 2);
    $table->timestamps();
});
```



```
});
```

Terdapat pelbagai fungsi untuk membina jenis-jenis *column* yang berbeza. Rujuk dokumentasi rasmi untuk senarai lengkap

<https://laravel.com/docs/8.x/migrations#available-column-types>

Arahan Migrate

Arahan untuk up()

```
$ php artisan migrate
```

Arahan untuk down()

```
$ php artisan migrate:rollback
```

Model / Eloquent

Dokumentasi penuh di <https://laravel.com/docs/8.x/eloquent>.

Seperti View dengan Blade, Model dalam Laravel menggunakan enjinnya sendiri yang dipanggil **Eloquent**. Model merujuk kepada sesuatu *table* dalam *database*.

Berikut adalah syarat-syarat dan keadah yang biasa digunakan dalam aplikasi Laravel.

- *Table* dalam database mestilah menggunakan nama plural, seperti **users**, bukan **user**.
- Nama model mestilah menggunakan nama *singular*. Seperti nama *class*, ianya menggunakan camel-case. Contoh : User.php.
- Model disimpan di dalam folder **/app/Models**
- Table mempunyai kolum bernama **id** dengan ciri *primary key* dan *autoincrement*.
- Tetapan untuk nama table dan juga nama kolum id boleh diubah sekiranya perlu.

Tetapan Database

Perkara pertama yang perlu dilaksanakan sebelum boleh menggunakan Model adalah memastikan sambungan ke *database* berjaya dilaksanakan.

Kita boleh menyediakan lebih daripada satu tetapan untuk sambungan ke *database* di dalam fail **.env**. Dan sambungan *database* utama yang akan digunakan boleh dibuat di dalam fail **/config/database.php**.

Contoh : tetapan sambungan database dalam fail **.env**

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel-app
DB_USERNAME=root
DB_PASSWORD=
```

Contoh : tetapan asal dalam fail /config/database.php

```
<?php
use Illuminate\Support\Str;

return [
    'default' => env('DB_CONNECTION', 'mysql'),
    ...
];
```

Semak semula bahagian **Konfigurasi** untuk tetapan sambungan ke *database*.

Tetapan Model

Sekiranya tidak dapat memenuhi syarat tetapan untuk Model mengikut kaedah Laravel seperti berkenaan nama *table* dan kolum *id*, ianya boleh diubah mengikut keperluan.

Contoh : Menyatakan tetapan Model secara manual

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Flight extends Model
{
    // menyatakan nama table
    protected $table = 'my_flights';

    // menyatakan kolum id, sekiranya bukan id
    protected $primaryKey = 'flight_id';

    // sifat autoincrement kolum id
    public $incrementing = false;

    // jenis kolum id, sekiranya bukan integer
    protected $keyType = 'string';
}
```

```
// mempunyai column created_at dan updated_at
public $timestamps = true;

// nama kolum created_at sekiranya bukan created_at
const CREATED_AT = 'creation_date';

// nama kolum updated_at sekiranya bukan created_at
const UPDATED_AT = 'updated_date';

// menyatakan database connection untuk digunakan
protected $connection = 'sqlite';

// mempunyai ciri soft delete, perlukan kolum deleted_at
// perlukan namespace untuk class SoftDelete di atas
use SoftDeletes;
}
```

Beri perhatian di mana kolum **deleted_at**, **created_at**, **updated_at** yang berjenis **DATETIME** diperlukan sekiranya digunakan dalam Model.

Membina Model Dengan Artisan

Cara terpantas untuk mula membina Model adalah dengan menggunakan Artisan.

Arahan PHP Artisan : Membina Model

```
php artisan make:model Book
```

Rujuk syarat di atas, Laravel akan mengandaikan table yang akan digunakan adalah *books*. Namun definisi untuk Model boleh diubah sekiranya perlu. Dengan arahan yang baru ini tadi, Laravel akan menjana fail **/app/Models/Book.php**.

Rangka Model : Book.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;
}
```

}

Menggunakan Model

Andaikan table *books* adalah seperti berikut :

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default
1	id	BIGINT	20	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT
2	title	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
3	author	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
4	price	DECIMAL	10,2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL

Berikut adalah contoh Route, Controller, dan View.

Route

```
use App\Http\Controllers\BookController;

Route::get('/book/{$id}', [BookController::class, 'show']);
```

Pastikan menggunakan **use** dengan Namespace yang betul untuk menggunakan Controller.

Controller : /app/Http/Controllers/BookController.php

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Models\Book;

class BookController extends Controller
{
    public function show($id) {
        $book = Book::find($id);
        return view('book',
            [
                'book' => $book
            ]
        );
    }
}
```

```
}
```

Pastikan menggunakan **use** dengan Namespace yang betul untuk menggunakan Model.

View : /resources/views/book.blade.php

```
<p><strong>Title: </strong>{{ $book->title }}</p>
<p><strong>Author: </strong>{{ $book->author }}</p>
```

Dengan semua kod seperti ini, lama ini boleh diakses dengan URL berikut:

- <http://aplikasi.test/books/1>

Mendapatkan Data

Mendapatkan semua rekod

```
$books = Book::all();
```

Mendapatkan semua rekod dengan ID.

```
$book = Book::find($id);
```

Mendapatkan rekod dengan where().

```
$books = Book::where('author', 'Usman Awang')->get();
```

```
$books = Book::where('price', '>', 10.00)->get();
```

Mendapatkan semua dengan susunan khusus.

```
$books = Book::orderBy('author')->get();
```

```
$books = Book::orderByDesc('author,')->get();
```

Mendapatkan semua rekod pertama sahaja.

```
$book = Book::where('author', 'Usman Awang')->first();
```

Eloquent & Query Builder

Dengan Eloquent Model, kita juga boleh menggunakan pelbagai lagi fungsi yang terdapat dari Query Builder seperti **count()**, **avg()**, **sum()**, **orderBy()**, **whereIn()**, **whereBetween()** dan lain-lain lagi.

Rujuk dokumentasi penuh tentang Query Builder di <https://laravel.com/docs/8.x/queries#basic-where-clauses>.

Menyimpan Rekod Baru

Sebelum boleh menyimpan atau mengemaskini rekod, pastikan telah menetapkan kolum yang boleh diisi dan dikemaskini dengan property **fillable**.

Fillable

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;
    protected $fillable = ['title', 'author', 'price'];
}
```

Ada dua kaedah menyimpan rekod baru.

Contoh : Menyimpan rekod dengan objek Model dan fungsi save().

```
use App\Models\Book;

$book = new Book;

$book->title = 'Salina';
$book->author = 'A. Samad Said';
$book->price = 49.95;

$book->save();
```

```
echo $book->id;
```

Contoh : Menyimpan rekod Model::create.

```
use App\Models\Book;

$book = Book::create([
    'title' => 'Srengenge',
    'author' => 'Shahnon Ahmad',
    'price' => 55.45,
]);

echo $book->id;
```

Mengemaskini Rekod

Sebelum boleh menyimpan atau mengemaskini rekod, pastikan telah menetapkan kolum yang boleh diisi dan dikemaskini dengan *property fillable*. **Rujuk bahagian Menyimpan Rekod Baru.**

Contoh : mengemaskini rekod

```
use App\Models\Book;

$book = Book::find(1); // mendapatkan rekod dengan id
$book->author = 'Arena Wati';
$book->save();
```

Membuang Data

Contoh : Membuang rekod

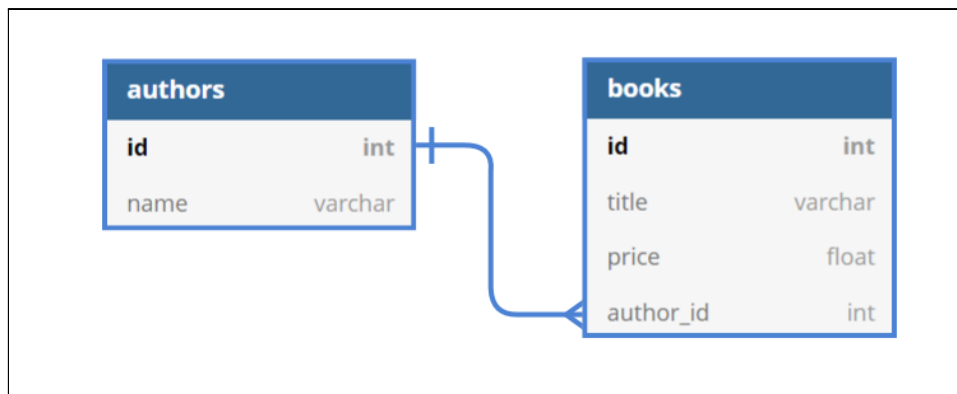
```
use App\Models\Book;

$book = Book::find(1);
$book->delete();
```

Sekiranya menggunakan tetapan Soft Delete (*rujuk bahagian Tetapan Model*), kolum ***deleted_at*** akan diisi dengan nilai tarikh dan masa ***delete*** dilaksanakan. Apabila data belum dibuang, ianya mempunyai nilai NULL.

One-to-Many Relationship

Andaikan *table* seperti berikut. Authors mempunyai banyak Books.



Kita boleh membina model-model berikut dan menghubungkannya.

Contoh : Model Author

```
<?php
namespace App\Models;

use App\Models\Book;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Author extends Model
{
    use HasFactory;
    protected $fillable = ['name'];

    public function books() {
        return $this->hasMany(Book::class);
    }
}
```

Contoh : Model Book

```
<?php
namespace App\Models;

use App\Models\Author;
use Illuminate\Database\Eloquent\Model;
```



```
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Book extends Model
{
    use HasFactory;
    protected $fillable = ['title','price'];

    public function author() {
        return $this->belongsTo(Author::class);
    }
}
```

Contoh : Penggunaan Model dengan Relationship di dalam Controller

```
<?php
namespace App\Http\Controllers;

use App\Models\Book;
use App\Models\Author;
use App\Http\Controllers\Controller;

class BookController extends Controller
{
    public function list_books() {
        $books = Book::all();
        foreach($books as $book) {
            echo '<h5>'.$book->title.'</h5><br>';
            echo 'Author : '.$book->author->name.'<br>';
        }
    }

    public function list_authors() {
        $authors = Author::all();
        foreach($authors as $author) {
            echo '<h5>Books by '.$author->name.'</h5>';
            foreach($author->books as $book) {
```

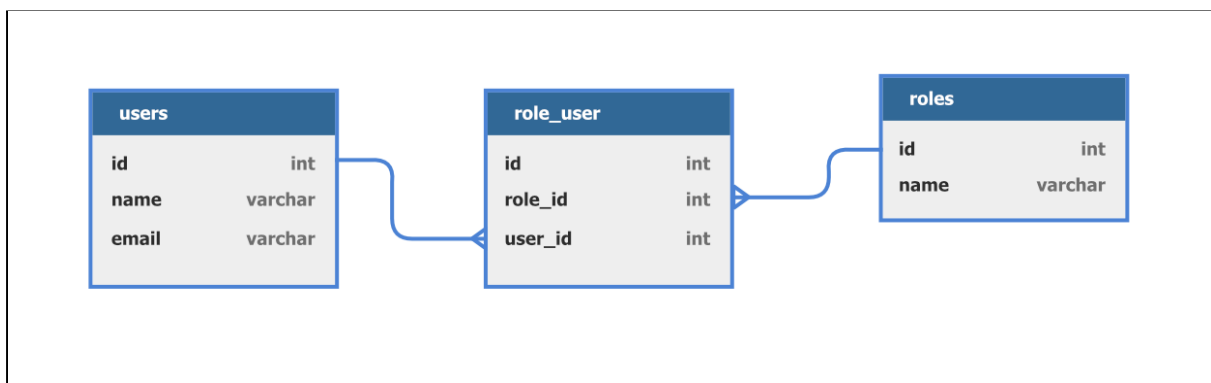
```

        echo $book->title.'<br>';
    }
}
}
}

```

Terdapat beberapa cara lain untuk menyatakan Relationship dengan Eloquent. Rujuk dokumentasi rasmi di <https://laravel.com/docs/8.x/eloquent-relationships>.

Many-to-Many Relationship



Untuk hubungan many-to-many seperti di atas, kita perlukan *table* penghubung.

Sekiranya kita menggunakan kemudahan dalam Laravel, *table* penghubung akan dinamakan dengan kedua-dua *table* yang terlibat dengan syarat berikut :

- Turutan nama table yang terlibat disusun mengikut alphabetical. (role_user)
- Nama *table* biasanya bersifat plural (users, roles), tetapi table penghubung bersifat singular (role_user).

Contoh : Model User

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The roles that belong to the user.
     */
}

```

```
public function roles()
{
    return $this->belongsToMany(Role::class);
}
```

Contoh : Model Role

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany(User::class);
    }
}
```

Table role_user tidak memerlukan Model.

Berikut adalah contoh kegunaan dalam *controller*.

Contoh : memaparkan roles dimiliki oleh pengguna.

```
use App\Models\User;

$user = User::find(1);

foreach ($user->roles as $role) {
    //
}
```

Untuk menyimpan dan mengemaskini roles oleh pengguna, kita boleh gunakan fungsi `sync()`.

Contoh : memberikan roles kepada user.

```
$user = User::findOrFail(1);
$user->roles()->sync([1,2]); // array of roles ID
```

Maklumat lanjut di dokumentasi rasmi

<https://laravel.com/docs/8.x/eloquent-relationships#many-to-many>

Eloquent Collection

Ada pelbagai lagi keupayaan Eloquent Model dalam Laravel dengan **Collection**. Data yang dipulangkan dengan lebih daripada satu rekod, dipulangkan sebagai **Eloquent Collection**.

Eloquent Collection dibina daripada class *Collection*. Jadi di sini, kita boleh menggunakan keupayaan kedua-dua Eloquent Collection dan *Collection* apabila menggunakan Eloquent Model.

Dokumentasi Eloquent Collection: <https://laravel.com/docs/8.x/eloquent-collections>.

Dokumentasi Collection: <https://laravel.com/docs/8.x/collections>.

Factory & Tinker

Factory

Factory membantu kita membina data contoh untuk pembangunan, berasaskan kepada Model yang telah dibina. Arahan berikut akan membina Factory.

```
$ php artisan make:factory BookFactory
```

Ini akan membina fail **/database/factories/BookFactory.php**. Dengan format **<Nama model>Factory.php**, factory yang dibina adalah khusus untuk *model* Book. Kita boleh menyatakan apakah model untuk factory semasa arahan php artisan seperti berikut.

```
$ php artisan make:factory BookFactory --model=Book
```

Kita perlu menetapkan definisi bagi data rekod untuk *table* books. Buka **/database/factories/BookFactory.php**.

/database/factories/BookFactory.php

```
<?php

namespace Database\Factories;

use App\Models\Book;
use Illuminate\Database\Eloquent\Factories\Factory;
```

```
class BookFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Book::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'title' => $this->faker->sentence(),
            'price' => $this->faker->randomFloat(2, 10, 50)
        ];
    }
}
```

Perhatikan teks yang dikuningkan. Ianya menetapkan data rawak yang akan dijanakan untuk **title** dan **price**. Ianya menggunakan *library* **Faker**. Pelbagai lagi data rawak boleh dijana untuk Orang, Alamat, Nombor, Teks dan lain-lain.

Rujuk dokumentasi Faker di sini <https://fakerphp.github.io/>

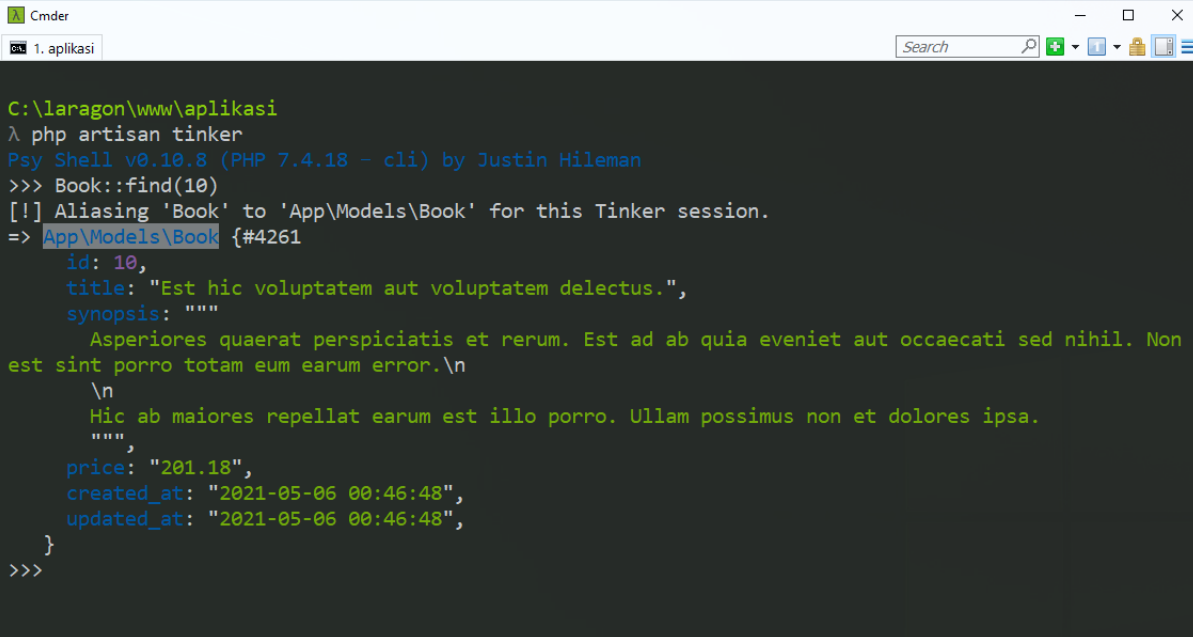
Tinker

Tinker adalah aplikasi untuk kita bermain, menguji dan memanipulasi aplikasi Laravel yang kita bangun secara *command-line*. Berikut adalah arahan untuk Tinker.

```
$ php artisan tinker
```

Di dalam Tinker, kita boleh melakukan pelbagai kod PHP dan juga mengakses **Model** yang telah dibina.

Kita menggunakan tinker sebagai satu cara untuk mengisi table dengan data-data contoh seperti yang telah ditetapkan dalam **Factory**.



```
C:\laragon\www\aplikasi
λ php artisan tinker
Psy Shell v0.10.8 (PHP 7.4.18 - cli) by Justin Hileman
>>> Book::find(10)
[!] Aliasing 'Book' to 'App\Models\Book' for this Tinker session.
=> App\Models\Book {#4261
  id: 10,
  title: "Est hic voluptatem aut voluptatem delectus.",
  synopsis: ""
  Asperiores quaerat perspiciatis et rerum. Est ad ab quia eveniet aut occaecati sed nihil. Non
  est sint porro totam eum earum error.
  \n
  Hic ab maiores repellat earum est illo porro. Ullam possimus non et dolores ipsa.
  ""
  price: "201.18",
  created_at: "2021-05-06 00:46:48",
  updated_at: "2021-05-06 00:46:48",
}
>>>
```

Arahan melihat contoh data dijana melalui factory mengguna fungsi **make()**.

```
>>> Book::factory()->make()
```

Arahan menjana data rawak melalui factory mengguna fungsi **create()**.

```
>>> Book::factory()->create()
```

Arahan menjana banyak data rawak melalui factory mengguna fungsi **create()**.

```
>>> Book::factory(10)->create()
```

Mengakses data melalui Model

```
>>> Book::all()
```

Melaksanakan arahan SQL dengan Query Builder

```
>>> DB::select('select * from books')
```

Ralat **Model not found**

Sekiranya menghadapi masalah ketika menggunakan Tinker untuk memanggil factory yang telah dibina dengan rata **“Model not found”**, cuba keluar daripada Tinker dan laksanakan arahan berikut :

```
$ composer dump-autoload
```

Ini akan membina semula fail autoloader dan mengambil kira lokasi model-model baru yang telah dibina.

Database & Query Builder

Rujuk dokumentasi rasmi <https://laravel.com/docs/8.x/database> dan juga <https://laravel.com/docs/8.x/queries>

Query Builder adalah *library* lain untuk komunikasi dengan *database*. Malahan Eloquent Model dibina dengan asas daripada Query Builder.

Sekiranya perlu melaksanakan arahan SQL terus kepada database, kita boleh menggunakan Query Builder.

Untuk mula menggunakan Query Builder, pastikan anda mengimport library DB Facade dengan use.

```
use Illuminate\Support\Facades\DB;
```

Melaksanakan Arahan SQL

Melaksanakan arahan SELECT

```
$users = DB::select('select * from users');
```

Melaksanakan arahan SELECT

```
$users = DB::table('users')->get();
```

Pernyataan SQL untuk update

```
$affected = DB::update(
    'update users set votes = 100 where name = ?',
    ['Anita']
);
```

\$affected akan memulangkan berapa rekod yang terkesan dengan hasil pernyataan SQL update.

Pernyataan SQL untuk insert

```
DB::insert('insert into users (id, name) values (?, ?)',  
[1, 'Marc']);
```

Pernyataan SQL untuk delete

```
$deleted = DB::delete('delete from users');
```

Mengguna Nilai dari *Variable*

Keadah Name Binding

```
$users = DB::select('select * from users where id = :id',  
[  
    'id' => 1  
]);
```

Keadah Name Binding Untuk Insert dengan Simbol (?)

```
DB::insert('insert into users (id, name) values (?, ?)',  
[1, 'Marc']);
```

Pelbagai *Database Connection*

Sekiranya ada keperluan untuk sambungan ke database yang selain daripada utama, database boleh dinyatakan sebelum pernyataan SQL.

Definisi beberapa sambungan *database* boleh dibuat dalam fail `/config/database.php`.

Database Connection

```
$users = DB::connection('sqlite')->select(...);
```

Memaparkan Rekod dengan *Pagination*

Dokumentasi rasmi di <https://laravel.com/docs/8.x/pagination>.

Pagination membantu dalam mengawal data yang dihantar ke *browser* pengguna. Sekiranya table mempunyai 100,000 rekod, terlalu banyak data yang perlu dihantar dan pengguna terpaksa menunggu masa yang lama untuk mula membaca rekod-rekod.

Pagination memecah data kepada kumpulan kecil mengikut halaman dan turutan.

Berikut adalah contoh melaksanakan Pagination.

Controller, /app/Http/Controllers/BookController.php

```
<?php
namespace App\Http\Controllers;

use App\Models\Book;
use App\Http\Controllers\Controller;

class BookController extends Controller
{
    public function index()
    {
        return view('book.index', [
            // untuk memaparkan 5 rekod dalam satu halaman
            'books' => Book::paginate(5)
        ]);
    }
}
```

Perhatikan view “**book.index**”. Ini bermakna fail view berada dalam **folder** **/app/views/book/index.blade.php**

View, /resources/views/book/index.blade.php

```
<h3>Books in the Shop</h3>
@foreach($books as $book)
    <p><strong>Title: </strong>{{ $book->title }}
    <br><strong>Price: </strong>{{ $book->price }}</p>
    <hr>
@endforeach

{{ $books->links() }}
```

Laravel akan menghasilkan kod HTML yang serasi untuk kerangka **CSS Tailwind**. Rujuk dokumentasi Tailwind <https://tailwindcss.com/>.

Menggunakan Pagination dengan Bootstrap CSS

Untuk menggunakan Bootstrap CSS bersama Pagination, tambahkan kod berikut kepada file `/app/Providers/AppServiceProviders.php` di dalam fungsi **`boot()`**. Tambahkan juga **`Namespace`** kepada **`Paginator`** dengan **`use`** di awal fail.

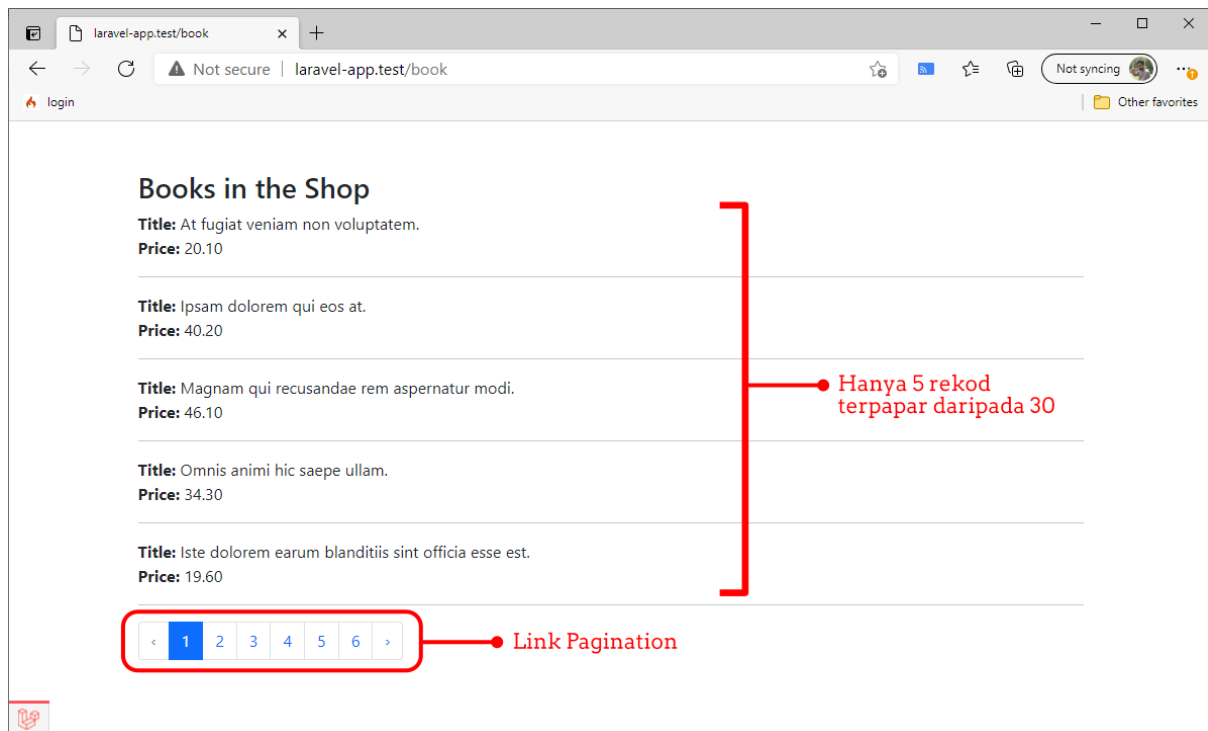
`/app/Providers/AppServiceProviders.php`

```
<?php
namespace App\Providers;

use Illuminate\Pagination\Paginator;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Paginator::useBootstrap();
    }
}
```



Borang & Memproses Input Pengguna

Menerima Data Dari Borang

Aplikasi web pastinya akan menerima data dari borang yang dibina. Membina borang dan memproses data dari borang selalunya dibuat dengan dua router.

Router untuk borang dan menerima input

```
// memaparkan borang untuk diisi
Route::get('/book/create', [BookController::class,
'create'])->name('book.create');

// untuk menerima dan memproses
Route::post('/book', [BookController::class,
'store'])->name('book.store');
```

/app/Http/Controllers/BookController

```
<?php
namespace App\Http\Controllers;
```

```
use App\Models\Book;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class BookController extends Controller
{
    // fungsi untuk memaparkan borang
    public function create()
    {
        return view('book.create');
    }

    // fungsi untuk memproses data
    public function store(Request $request)
    {
        // menyimpan data dengan Model
        Book::create([
            'title' => $request->title,
            'price' => $request->price
        ]);

        return redirect('book.index');
    }
}
```

Perhatikan bahagian yang berwarna kuning.

- **use App\Models\Book;**
Pastikan Model telah diimport dengan use.
- **use Illuminate\Http\Request;**
Pastikan *class Request* telah diimport dengan use. Ini bakal digunakan bersama fungsi untuk menerima input pengguna.
- **public function store(Request \$request)**
Kita mengguna class Request untuk menerima input daripada borang. Object \$request mengandungi data yang diinginkan.

- `redirect('book.index');`
Pengguna dihantarkan ke laman senarai buku.

View `/resources/views/book/create.blade.php` untuk memaparkan borang

```
<form action="{{ route('book.store') }}" method="POST">
<h4>Create a New Book</h4>
@csrf
    <p>
        <label>Title</label>
        <input type="text" name="title" />
    </p>
    <p>
        <label>Price</label>
        <input type="text" name="price" />
    </p>
</form>
```

Perhatikan bahagian yang berwarna kuning.

- `{{ route('book.store') }}`
route() yang digunakan dengan nama daripada router akan menjana URL yang betul untuk borang
- `@csrf`
Ini merupakan ciri keselamatan untuk Cross Site Request Forgery. Laravel akan menjana token khusus untuk ciri-ciri keselamatan.
- Pasti input HTML mempunyai nama yang betul.

Validation

Validation adalah proses memeriksa data yang diterima daripada borang, mengesahkannya dan memaparkan mesej sekiranya terdapat sebarang isu dengan data tersebut.

Validation memudahkan kita untuk menetapkan syarat-syarat untuk input yang ingin diterima.

Contoh, fungsi dari controller

```
public function store(Request $request)
{
```

```
$validated_data = $request->validate([
    'title' => 'required|min:5|max:255',
    'price' => 'required|numeric'
]);
```

```
Book::create($validated_data);
redirect('book.index');
}
```

Perhatikan ruangan berwarna kuning :

- Ada pelbagai syarat yang boleh ditetapkan untuk *validation*. Rujuk dokumentasi <https://laravel.com/docs/8.x/validation#available-validation-rules>.
- Sekiranya *validation* gagal, Laravel akan menghantar pengunjung kembali ke laman borang, dalam contoh ini, **/book/create**. Rujuk bahagian bawah untuk keadah memaparkan mesej ralat dengan view.
- Sekiranya *validation* data hasil daripada *validation* akan dipulangkan, dalam contoh ini kepada *variable* **\$validated_data**.
- Variable **\$validated_data** boleh terus digunakan bersama Model untuk membuat rekod baru.

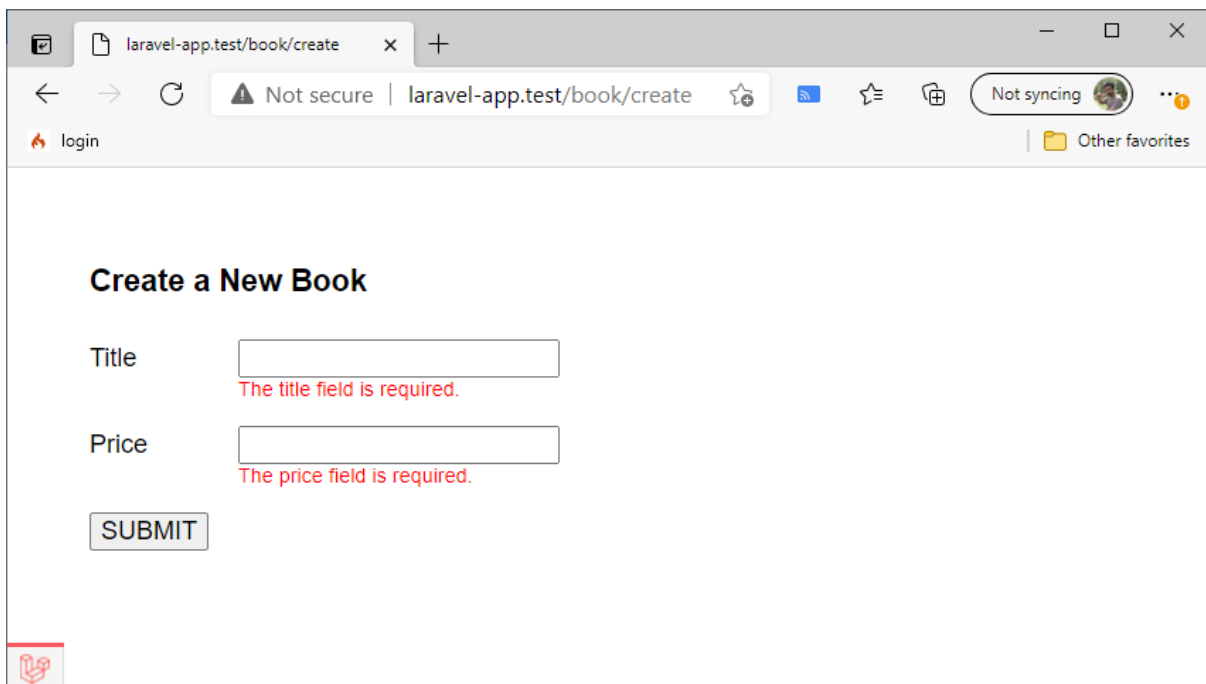
Memaparkan Mesej Ralat Daripada Validation

```
<form action="{{ route('book.store') }}" method="POST">
<h4>Create a New Book</h4>
@csrf
    <p>
        <label>Title</label>
        <input type="text" name="title" />
        @error('title')
            <br><small style="color:red">{{ $message }}</small>
        @enderror
    </p>
    <p>
        <label>Price</label>
        <input type="text" name="price" />
```

```
@error('price')
<br><small style="color:red">{{ $message }}</small>
@enderror

</p>
<button type="submit">SUBMIT</button>
</form>
```

- Mesej daripada *validation* boleh dipaparkan di antara **@error('nama-field')** dan **@enderror**
- Mesej yang sebenar dipaparkan dengan **{{ \$message }}**



Untuk memaparkan nilai yang digunakan dahulu oleh pelawat, kita boleh gunakan fungsi blade **{{ old('name input') }}**.

Contoh : memaparkan nilai terdahulu.

```
<form action="{{ route('book.store') }}" method="POST">
<h4>Create a New Book</h4>
@csrf
<p>
```

```
<label>Title</label>
<input type="text" name="title" value="{{old('title')}}" />
@error('title')
<br><small style="color:red">{{ $message }}</small>
@enderror
</p>
<p>
<label>Price</label>
<input type="text" name="price" value="{{old('price')}}" />
@error('price')
<br><small style="color:red">{{ $message }}</small>
@enderror
</p>
<button type="submit">SUBMIT</button>
</form>
```

Session

Dokumentasi penuh di <https://laravel.com/docs/8.x/session>.

Session membantu menyimpan data dari saya laman atau proses untuk digunakan di laman yang lain. Contoh yang paling kerap adalah untuk menyimpan data *Shopping Cart* dalam web e-commerce.

Session boleh digunakan dengan dua kaedah :

- Request object, pastikan Request \$request telah diisytiharkan sebagai *parameter* dan *Illuminate\Http\Request* telah digunakan dengan *use* di awal fail.
- Session global helper

Menyimpan data ke session dengan objek Request

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function show(Request $request, $id)
    {
```



```
        $request->session()->put('key', 'value');
    }
}
```

Menyimpan data dengan *helper global session()*

```
session(['key' => 'value']);
```

```
session([
    'data1' => 'ya',
    'data2' => 'tidak',
]);
```

Mendapatkan data dari session. Nilai *default* adalah tidak wajib.

```
$value = $request->session()->get('key', 'default');
```

```
$value = session('key', 'default');
```

Mendapatkan semua data session.

```
$data = $request->session()->all();
```

Memeriksa kewujudan nilai session, mengabaikan *null*

```
if ($request->session()->has('users')) {
    // return false even when the value is null
}
```

Memeriksa kewujudan nilai session, walaupun null

```
if ($request->session()->exists('users')) {
    // will return true even the value is null
}
```

Menyimpan data array

```
$request->session()->put('user.teams', ['Ali', 'Siti']);
$request->session()->push('user.teams', 'Karim');
```

Memadamkan session

```
$request->session()->forget('key');
```

```
$request->session()->forget(['key1', 'key2']);
```

Membuang semua session

```
$request->session()->flush();
```

Flash Data

Session secara amnya akan kekal hingga pengguna tidak menggunakan sistem dalam tempoh masa tertentu. Tetapi ini ada di dalam `/config/session.php`. Nilai asal yang ditetapkan adalah 120 saat.

Flash Data adalah sejenis data *session* yang hidup hanya untuk laman yang berikutnya. Kegunaan biasa adalah untuk mesej bagi notifikasi kejayaan, kegagalan ataupun ralat dalam operasi.

Menetapkan *flash data*

```
$request->session()->flash('status', 'Task was successful!');
```

Di view, kita boleh gunakan kod seperti berikut.

Menetapkan *flash data*

```
@if( session('status') )  
<div class="alert alert-info">  
    {{ session('status') }}  
</div>  
@endif
```

Sekiranya flash data ingin dikekalkan terus dari laman berikutnya, di laman tersebut, kita boleh melaksanakan fungsi `keep`.

Mengekalkan *flash data*

```
$request->session()->keep('status');
```

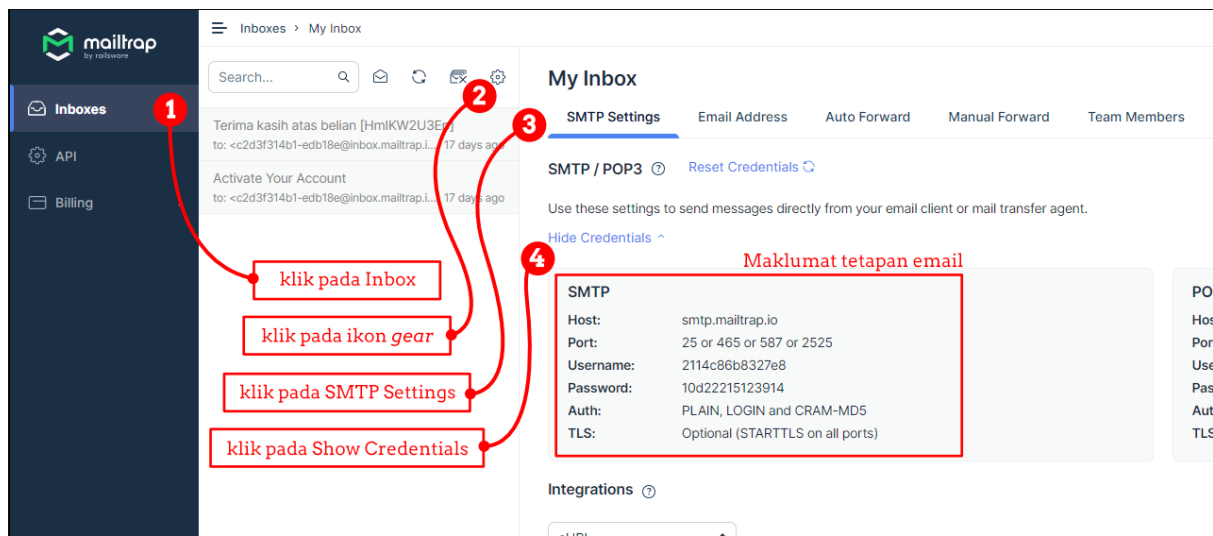
Menghantar Email

Mailtrap.io

Dalam tempoh pembangunan, kita perlukan satu kaedah untuk menghantar dan memeriksa email yang telah dihantar. Mailtrap.io adalah sebuah platform untuk tujuan tersebut.

Email yang dihantar dengan tetapan yang diberikan akan dapat kita lihat di dalam akaun di Mailtrap.

- Daftar akaun percuma di <https://mailtrap.io/>
- Dapatkan tetapan untuk menghantar email



Tetapan

Sebelum menghantar email, kita perlu membuat tetapan tentang kaedah email dihantar, server penghantar email, alamat email penghantar, dan sebagainya. Ini boleh dibuat melalui fail `.env`. Rujuk bahagian **Konfigurasi**.

Dapat maklumat tetapan email dari Mailtrap.io untuk digunakan dalam fail `.env`.

Berikut adalah tetapan email dalam fail `.env`.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=2114c86b8327e8
MAIL_PASSWORD=10d22215123914
MAIL_ENCRYPTION=TLS
MAIL_FROM_ADDRESS=kelas@kelasprogramming.com
MAIL_FROM_NAME="${APP_NAME}"
```

Pengenalan

Laravel mempunyai kaedah yang khusus untuk menghantar email. Ianya membantu modifikasi dan kekemasan kod.

Bagi setiap email yang kita akan hantar, kita perlukan

1. `class Mailable`, seperti sebuah Controller, tetapi untuk email
2. View untuk email tersebut, sebuah template untuk kandungan email

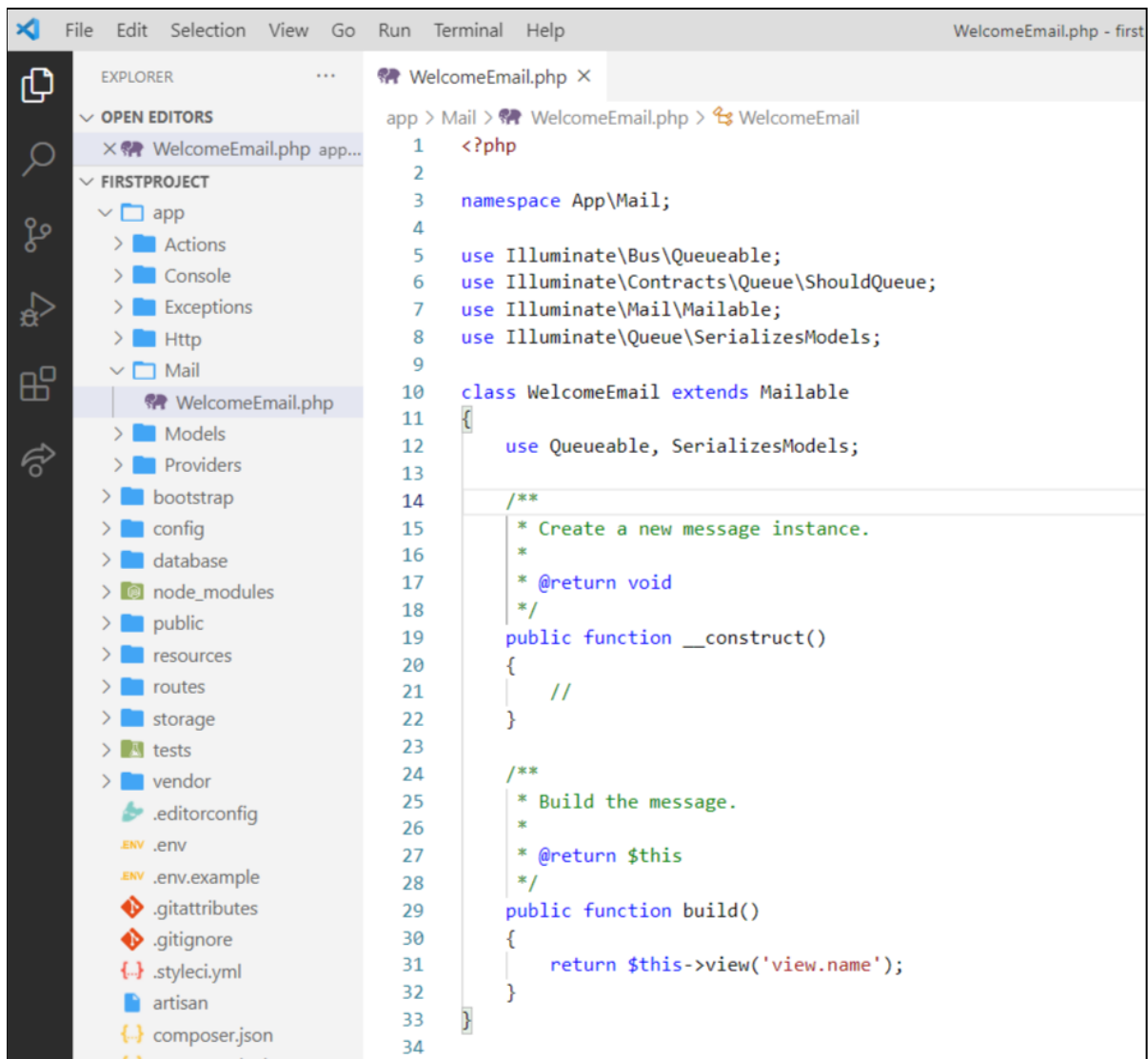
Apabila keduanya sudah tersedia, ianya akan digunakan di dalam Controller.

Membina Mailable dengan Artisan

Gunakan arahan berikut untuk membina *class* Mailable.

```
$ php artisan make:mail WelcomeEmail
```

Sekarang semak fail baru yang dijana di **`/app/Mail/WelcomeEmail.php`**



Perhatikan fungsi `build()`. Ini adalah di mana kandungan email dijana. Dan kita juga boleh mengubahsuai beberapa data berkaitan email seperti penghantar (*from*), perkara (*subject*), view dan *attachment*.

- Tanpa **`from()`**, alamat email yang akan digunakan adalah seperti yang ditetapkan dalam `.env`, ataupun `/config/mail.php`

- Tanpa **subject()**, Laravel akan mengambil nama class dan mengubahsuainya dari **WelcomeEmail** kepada **"Welcome Email"**.

Contoh : fungsi **build()** dalam sebuah class **Mailable**

```
public function build()
{
    return $this->view('mail.welcome')
        ->subject('Selamat Datang ke Kelas Programming')
        ->from('kp@kelasprogramming.com');
}
```

Membina View untuk Email

Amalan yang biasa adalah dengan membina satu folder baru bernama mail di dalam `/resources/views`. Ini akan memberikan folder `/resources/views/mail`. Dari contoh di atas, kita akan membina satu fail `/resources/views/mail/welcome.blade.php`.

Contoh : `/resources/views/mail/welcome.blade.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Selamat Datang</title>
    <style>
        body { background-color: #ddd;}
        table {
            width: 90%; margin: auto; max-width: 800px;
            background-color: #fff; border:1px solid #aaa;
            border-radius: 10px; font-family: arial; padding: 20px; }
        h1 { margin-top: 0px; }
    </style>
</head>
<body>
    <table cellpadding="0" cellspacing="0" border="0">
        <tr><td>
```

```
<h1>Selamat Datang</h1>
<p>Anda kini adalah ahli dalam KelasProgramming.com.</p>
</td></tr>
</table>
</body>
</html>
```

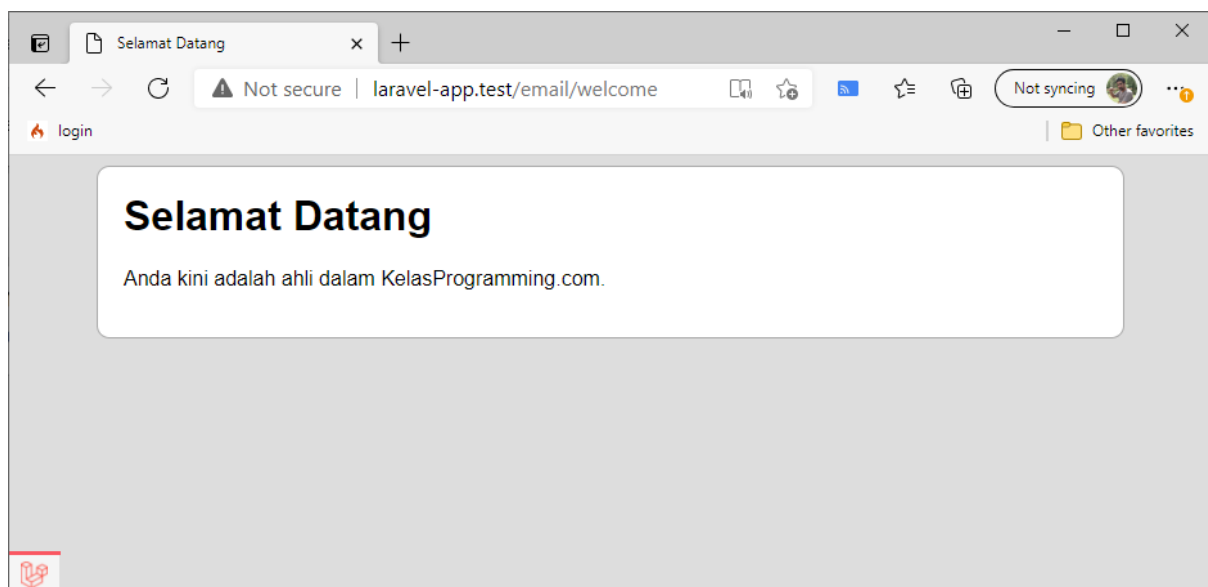
Menguji Email Dalam Browser

Kita boleh menyemak rupa bentuk email kita dalam *browser* dengan membina satu *router* baru di dalam `/routes/web.php`.

Contoh : router baru dalam `/routes/web.php`

```
Route::get('/email/welcome', function () {
    return new App\Mail\WelcomeEmail();
});
```

Dengan definisi sebegini, kita boleh akses <http://aplikasi.test/email/welcome>



Menghantar Email

Email boleh dihantar dari mana-mana Controller. Kita andaikan akan menghantar email selepas memasukkan maklumat pengguna ke dalam *database*.

Contoh : menghantar email dari Controller

```
<?php
namespace App\Http\Controllers;

use App\Models\User;
use App\Mail>WelcomeEmail;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Mail;

class AuthController extends Controller
{

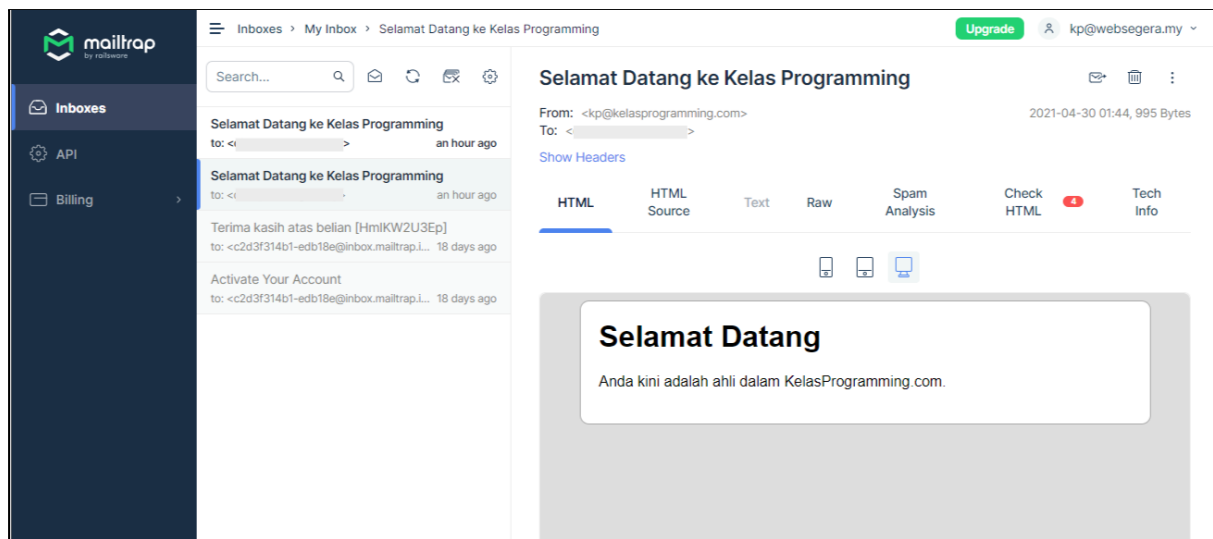
    public function store(Request $request) {
        $validated_data = $this->validate($request, [
            'email' => 'required|email|unique:users,email',
            'name' => 'required',
            'username' => 'required|alpha_dash|unique:users,username',
            'password' => 'required'
        ]);

        $user = User::create( $validated_data );
        Mail::to($user->email)->send( new WelcomeEmail );
    }
}
```

Perhatikan kawasan yang berwarna kuning.

- Pastikan telah mengimport **Illuminate\Support\Facades\Mail** dengan **use**.
- Pastikan telah mengimport **App\Mail>WelcomeEmail** dengan **use**.
- Hantar email dengan arahan
Mail::to(<email penerima>)->send(new <class Mailable>);

Dengan tetapan yang betul, kita boleh menyemak email yang dihantar di Mailtrap.io.



Menyertakan Data ke View

Email biasanya mempunyai maklumat yang khusus untuk penerima tersebut. Maka kita perlu boleh mengubah kandungan email mengikut keperluan.

Contoh : `/resources/views/email/welcome.blade.php`

```
...
<table cellpadding="0" cellspacing="0" border="0">
  <tr><td>
    <h1>Selamat Datang</h1>
    <p>{{ $name }}, anda kini adalah ahli dalam
KelasProgramming.com.</p>
  </td></tr>
</table>
...
```

Kita perlu mengubah supaya `class WelcomeEmail` kita boleh menerima parameter dalam `constructor`.

Contoh : Fail `/app/Mail/WelcomeEmail.php` yang telah diubahsuai

```
<?php
namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
```



```
use Illuminate\Queue\SerializesModels;
use Illuminate\Contracts\Queue\ShouldQueue;

class WelcomeEmail extends Mailable
{
    use Queueable, SerializesModels;

    public function __construct( Array $data )
    {
        $this->data = $data;
    }

    public function build()
    {
        return $this->view('mail.welcome')
            ->subject('Selamat Datang ke Kelas Programming')
            ->from('kp@kelasprogramming.com')
            ->with($this->data);
    }
}
```

Dan sekarang kita boleh mengemaskini Controller.

Contoh : fungsi store dalam Controller

```
...
public function store(Request $request) {
    $validated_data = $this->validate($request, [
        'email' => 'required|email|unique:users,email',
        'name' => 'required',
        'username' => 'required|alpha_dash|unique:users,username',
        'password' => 'required'
    ]);

    User::create( $validated_data );
}
```

```
Mail::to($validated_data['email'])
    ->send( new WelcomeEmail( $validated_data ) );
}
...
```

Kita juga boleh mengujinya melalui /routes/web.php dengan kod sedemikian.

Contoh : menguji email melalui /resources/web.php

```
Route::get('/email/welcome', function () {

    $data = [
        'name' => 'Ahmad Kasan',
        'email' => 'ahmad.kasan@getaranjiwa.com'
    ];

    Mail::to($data['email'])->send( new App\Mail\WelcomeEmail(
    $data ) );
    return new App\Mail\WelcomeEmail( $data );
});
```

Authentication

Dokumentasi rasmi di <https://laravel.com/docs/8.x/authentication>.

Pakej Authentication Laravel 8

Berikut adalah Authentication *package* yang disertakan bersama Laravel 8.

- **Fortify**

Pakej ini adalah untuk operasi login, logout, forget password dan lain-lain yang lazim kita temui bersama aplikasi web. Ia tiada antaramuka (UI). Tetapi kita bebas untuk membina antaramuka kita sendiri dengan hanya membina beberapa *view*. Kita juga bebas menggunakan apa sahaja kerangka CSS dan JS yang kita biasa, seperti Bootstrap, Bulma, Tailwind, VueJS, React, Angular atau apa sahaja.

<https://laravel.com/docs/8.x/fortify>

- **Sanctum**

Sanctum menawarkan operasi login tanpa *session*, tetapi dengan JSON Web Token. Ini lebih sesuai digunakan untuk pembangunan API dan dengan integrasi pengguna untuk aplikasi *mobile* dan *single-page-app* atau SPA.

<https://laravel.com/docs/8.x/sanctum>

- **Passport**

Passport menawarkan operasi OAuth2. Dengan OAuth2, kita boleh membenarkan aplikasi luar menawarkan login kepada pengguna mereka menggunakan *username* dan *password* dari aplikasi kita. Ini adalah seperti bila kita login ke aplikasi Waze dengan akaun Facebook.

<https://laravel.com/docs/8.x/passport>

Authentication juga boleh dibangunkan tanpa pakej yang disertakan, dengan hanya menggunakan library **Auth** dan **Session** daripada Laravel.

Laravel Starter Kit

- **Laravel Breeze**

Laravel dibangunkan tanpa pakej Authentication yang disertakan seperti Fortify. Segala proses boleh dikemaskini dan disemak bersama *Controller* yang dibina. Laravel Breeze baik untuk mempelajari bagaimana Authentication dilaksanakan. Antaramuka hadapan (UI) dibangunkan dengan kerangka CSS Tailwind. Ia juga menggunakan pendekatan *Component* dalam *View*, yang tidak dibincangkan dalam nota ini. Laravel Breeze juga menawarkan kaedah pembangunan Authentication dengan pantas.

<https://laravel.com/docs/8.x/starter-kits#laravel-breeze>

- **Laravel Jetstream**

Laravel Jetstream lebih baik tetapi lebih kompleks. Ianya menggunakan pakej Fortify sebagai enjin di belakangnya. Ianya menggunakan sistem pembangunan antaramuka dengan Livewire ataupun Inertia.js. Sekiranya anda mempunyai kemahiran dalam VueJS, dan mahukan sebuah cara yang mudah dan pantas membangunkan Authentication, boleh pertimbangkan Laravel Jetstream.

<https://jetstream.laravel.com/2.x/introduction.html>

Kita juga boleh membangunkan antaramuka (UI) *authentication* sendiri dengan menggunakan Fortify di belakangnya.

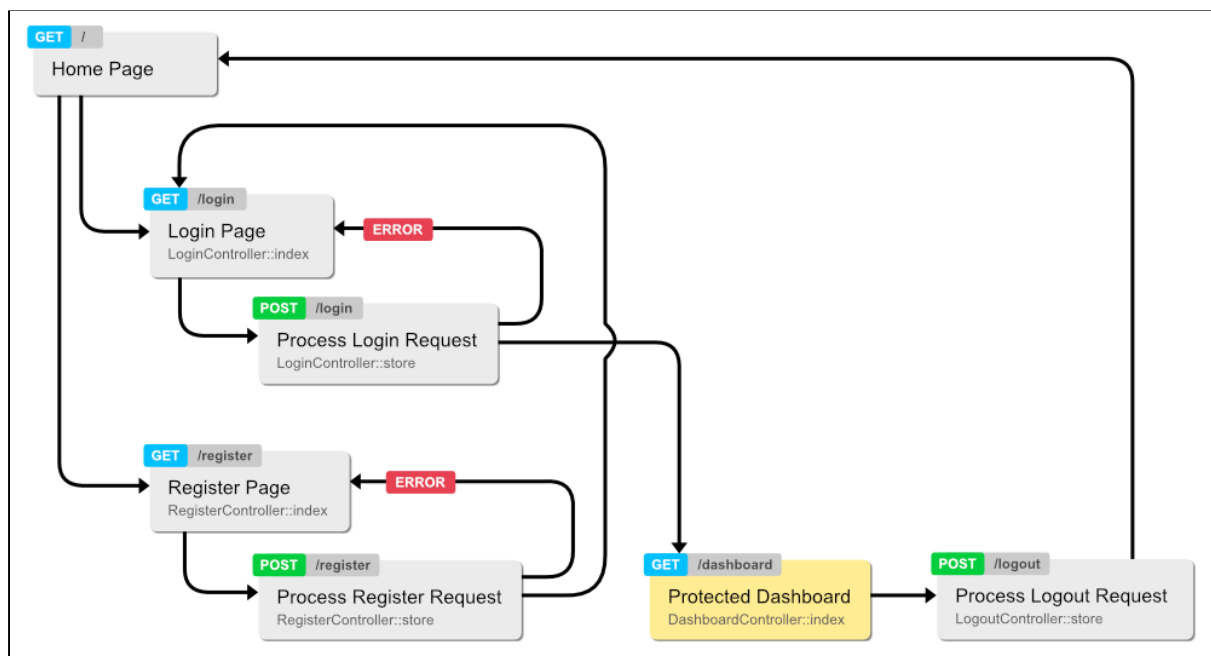
Terdapat juga beberapa pakej sumber terbuka yang menawarkan antaramuka untuk Fortify seperti **FortifyUI**.

<https://github.com/zacksmash/fortify-ui>

Membina Authentication Ringkas

Berikut adalah keadah membangunkan Authentication yang ringkas. Ianya mempunyai operasi berikut :

Nama	Fungsi	Protokol	URL	Controller	Fungsi
login	Paparkan Login	GET	/login	LoginController	index
	Proses Login	POST	/login	LoginController	store
register	Paparkan Pendaftaran	GET	/register	RegisterController	index
	Proses Pendaftaran	POST	/register	RegisterController	store
logout	Proses Logout	POST	/logout	LogoutController	store
dashboard	Laman Terkawal	GET	/dashboard	DashboardController	index



6 Langkah Membina Authentication Ringkas

1. Membina *table users*
2. Pendaftaran
3. Login

4. Logout
5. Dashboard
6. Routes

Dalam contoh di bawah, ianya mengandungi kod yang paling ringkas tanpa hiasan CSS. Ianya lebih bertujuan menunjukkan kaedah Authentication. View juga boleh diperbaiki dengan `@extend()` dan `@section()`. Dalam contoh di bawah, ianya dikomen.

LANGKAH 1: Membina *table users*

Langkah pertama adalah dengan melakukan arahan untuk membina *table users* dalam *database*.

Laravel didatangkan dengan arahan pantas untuk membina tables berkaitan pengguna. Laksanakan arahan berikut untuk membina *tables* yang diperlukan untuk pengurusan pengguna.

```
$ php artisan migrate
```

Table : users

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default
1	id	BIGINT	20	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT
2	name	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No default
3	email	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No default
4	email_verified_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
5	password	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No default
6	remember_token	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
7	created_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
8	updated_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL

LANGKAH 2: Pendaftaran

Langkah ini akan membina *Controller* dan *View* untuk Pendaftaran pengguna.

Perhatikan yang satu *folder* baru **Auth** dibuat di dalam *folder Controllers*.

Controller : `/app/Http/Controllers/Auth/RegisterController.php`

```
<?php
namespace App\Http\Controllers\Auth;

use App\Models\User;
```

```
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Hash;

class RegisterController extends Controller
{
    public function index()
    {
        return view('auth.register');
    }

    public function store(Request $request)
    {
        $data = $request->validate($request, [
            'name' => 'required|max:255',
            'username' => 'required|max:255',
            'email' => 'required|email|max:255',
            'password' => 'required|confirmed',
        ]);

        $data['password'] = Hash::make($data['password']);
        User::create($data);

        auth()->attempt($request->only('email', 'password'));

        return redirect()->route('login');
    }
}
```

- **Model User**

Model User memang telah disertakan dalam Laravel. Pastikan ianya diimport melalui ***App\Models\User*** menggunakan ***use***.

- **Password Pengguna**

Password pengguna tidak boleh disimpan dalam bentuk asal. Ianya mesti dihash atau diubah bentuk agar password sebenar tidak diketahui. Dalam Laravel ada library Hash Facade yang boleh digunakan. Pastikan anda telah mengimport

Illuminate\Support\Facades\Hash dengan `use`. Kemudian `Hash::make()` digunakan untuk mengubah *password* pengguna dari bentuk asal.

Dalam view di bawah, perhatikan satu folder **auth** baru dibina dalam `/resources/views`. View ini boleh dikembangkan lagi melalui `@extend` yang memanggil fail `/resources/views/layouts.app.blade.php`. Dalam contoh ini ianya dikomen.

View : `/resources/views/auth/register.blade.php`

```
{{--
@extends('layouts.app')

@section('content')
--}}

@if ($errors->any())
    <div class="status">Please check your form</div>
@endif

<form action="{{ route('register') }}" method="post">
    @csrf
    <div>
        <label for="name">Name</label>
        <input type="text" name="name" id="name" placeholder="Your
name" value="{{ old('name') }}">
        @error('name')
            <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>

    <div>
        <label for="email">Email</label>
        <input type="text" name="email" id="email"
placeholder="Your email" value="{{ old('email') }}">
        @error('email')
            <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>
</form>
```

```
</div>

<div>
    <label for="password">Password</label>
    <input type="password" name="password" id="password"
placeholder="Choose a password">
    @error('password')
        <div class="invalid-feedback">{{ $message }}</div>
    @enderror
</div>

<div>
    <label for="password_confirmation">Password again</label>
    <input type="password" name="password_confirmation"
id="password_confirmation"
        placeholder="Repeat your password" value="">
    @error('password_confirmation')
        <div class="invalid-feedback">{{ $message }}</div>
    @enderror
</div>

<div>
    <button type="submit">Register</button>
</div>
</form>

{{--
@endsection
--}}
```

LANGKAH 3: Login

Langkah ini adalah untuk membina operasi login untuk pengguna. Ia melibatkan pembangunan Controller dan View yang diperlukan.

Controller : /app/Http/Controllers/**Auth**/LoginController.php


```
<?php
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class LoginController extends Controller
{
    public function index()
    {
        return view('auth.login');
    }

    public function store(Request $request)
    {
        $this->validate($request, [
            'email' => 'required|email',
            'password' => 'required',
        ]);

        if (!auth()->attempt($request->only('email', 'password')))
        {
            return back()->with('status', 'Invalid login');
        }

        return redirect()->route('dashboard');
    }
}
```

View : /resources/views/auth/login.blade.php

```
{{--
@extends('layouts.app')

@section('content')
```

```
--}}

@if (session('status'))
    <div class="status">{{ session('status') }}</div>
@endif

<form action="{{ route('login') }}" method="post">
    @csrf

    <div>
        <label for="email">Email</label>
        <input type="text" name="email" id="email"
placeholder="Your email" value="{{ old('email') }}">

        @error('email')
            <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>

    <div>
        <label for="password">Password</label>
        <input type="password" name="password" id="password"
placeholder="Choose a password" value="">

        @error('password')
            <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>

    <div>
        <button type="submit">Login</button>
    </div>
</form>
```

```
{{--  
@endsection  
--}}
```

LANGKAH 4: Logout

Langkah ini adalah untuk membina Controller untuk operasi logout untuk pengguna. Tiada apa yang perlu dipaparkan ke pengguna, maka View tidak diperlukan.

/app/Http/Controllers/Auth/LogoutController.php

```
<?php  
namespace App\Http\Controllers\Auth;  
  
use App\Http\Controllers\Controller;  
use Illuminate\Http\Request;  
  
class LogoutController extends Controller  
{  
    public function store()  
    {  
        auth()->logout();  
        return redirect()->route('home');  
    }  
}
```

LANGKAH 5: Dashboard

Langkah ini akan membina *Controller* dan *View* untuk Dashboard sebagai contoh kepada laman yang dilindungi.

/app/Http/Controllers/DashboardController.php

```
<?php  
namespace App\Http\Controllers;  
  
use App\Http\Controllers\Controller;  
use Illuminate\Http\Request;
```

```
class DashboardController extends Controller
{
    public function index()
    {
        return view('dashboard');
    }
}
```

View : /resources/views/dashboard.blade.php

```
{{--
@extends('layouts.app')

@section('content')
--}}

<h1>Dashboard</h1>
<form action="{{ route('logout') }}" method="post">
    @csrf
    <button type="submit">Logout</button>
</form>

{{--
@endsection
--}}
```

LANGKAH 6: Routes

Akhir sekali, kita perlu menetapkan definisi untuk URL melalui *router*.

Router : /routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
```

```
use App\Http\Controllers\DashboardController;
use App\Http\Controllers\Auth\LoginController;
use App\Http\Controllers\Auth\LogoutController;
use App\Http\Controllers\Auth\RegisterController;

Route::get('/', function () {
    return view('home');
})->name('home');

// only logged in can access
Route::middleware('auth')->group(function () {
    Route::get('/dashboard', [DashboardController::class,
    'index'])->name('dashboard');
    Route::post('/logout', [LogoutController::class,
    'store'])->name('logout');
});

// only guest can access
Route::middleware('guest')->group(function () {
    Route::get('/login', [LoginController::class,
    'index'])->name('login');
    Route::post('/login', [LoginController::class, 'store']);
    Route::get('/register', [RegisterController::class,
    'index'])->name('register');
    Route::post('/register', [RegisterController::class,
    'store']);
});
```

Mencuba Fortify

Dokumentasi lengkap di <https://laravel.com/docs/8.x/fortify#introduction>.

Bahagian ini sekadar untuk memperkenalkan dan mencuba Fortify. Di bahagian lain, kita akan membina authentication menggunakan **FortifyUI**, yang lengkap dengan UI.

Fortify adalah pakej Authentication rasmi oleh Laravel. Ianya lengkap untuk pelbagai operasi dari login, logout, forget password, pendaftaran dan lain-lain. Namun, ianya tiada antaramuka, yang mana perlu dibina sendiri.

Terdapat juga pakej sumber terbuka yang menawarkan UI untuk Fortify seperti **FortifyUI**.

Pemasangan

Arahan untuk pemasangan Fortify (bukan FortifyUI). Arahan ini tidak diperlukan sekiranya akan menggunakan FortifyUI

```
$ composer require laravel/fortify
$ php artisan vendor:publish
--provider="Laravel\Fortify\FortifyServiceProvider"
```

Pastikan anda telah mengemaskini fail **.env** dengan tetapan *database* yang betul.

Sekiranya belum lagi melaksanakan arahan **artisan migrate**, laksanakan arahan berikut untuk membina *tables* yang diperlukan.

```
$ php artisan migrate
```

Selepas semua ini, kita akan dapat routes seperti berikut:

URL	Protokol	Nama
/api/user	GET HEAD	
/forgot-password	GET HEAD	password.request
/forgot-password	POST	password.email
/login	POST	
/login	GET HEAD	login
/logout	POST	logout
/register	GET HEAD	register
/register	POST	
/reset-password	POST	password.update
/reset-password/{token}	GET HEAD	password.reset

/two-factor-challenge	GET HEAD	two-factor.login
/two-factor-challenge	POST	
/user/confirm-password	POST	
/user/confirm-password	GET HEAD	password.confirm
/user/confirmed-password-status	GET HEAD	password.confirmation
/user/password	PUT	user-password.update
/user/profile-information	PUT	user-profile-information.update
/user/two-factor-authentication	POST	
/user/two-factor-authentication	DELETE	
/user/two-factor-qr-code	GET HEAD	
/user/two-factor-recovery-codes	GET HEAD	
/user/two-factor-recovery-codes	POST	

Routes ini semua boleh disemak dengan arahan berikut :

```
$ php artisan route:list
```

Konfigurasi

Konfigurasi untuk **Fortify** ada di **/config/fortify.php**. Boleh semak pelbagai ciri-ciri yang ada untuk Fortify.

Semak baris 134 hingga ke akhir fail. Ianya menunjukkan ciri-ciri yang ada yang boleh diguna ataupun diabaikan.

/config/fortify.php baris hingga akhir

```
'features' => [
    Features::registration(),
    Features::resetPasswords(),
    // Features::emailVerification(),
    Features::updateProfileInformation(),
    Features::updatePasswords(),
    Features::twoFactorAuthentication([
```

```
        'confirmPassword' => true,  
    ]),  
],  
];
```

Langkah seterusnya, kita perlu tetapkan di manakah *view* yang mempunyai *form* untuk login, pendaftaran dan lain-lain. Ini dilakukan melalui **Providers**.

Buka fail `/app/Providers/FortifyServiceProvers.php`. Di bahagian ***function boot()***, baris-baris berikut perlu ditambah.

```
...  
public function boot()  
{  
    // tetapkan lokasi view untuk login  
    Fortify::loginView(function () {  
        return view('auth.login');  
    });  
  
    // tetapkan lokasi view untuk register  
    Fortify::registerView(function () {  
        return view('auth.register');  
    });  
  
    // tetapkan lokasi view untuk forgot-password  
    Fortify::requestPasswordResetLinkView(function () {  
        return view('auth.forgot-password');  
    });  
  
    // tetapkan lokasi view untuk reset-password  
    Fortify::resetPasswordView(function ($request) {  
        return view('auth.reset-password', ['request' =>  
$request]);  
    });  
}
```



```
// bahagian yang lain boleh dipasang sekiranya perlu

// Fortify::verifyEmailView(function () {
//     return view('auth.verify-email');
// });

// Fortify::confirmPasswordView(function () {
//     return view('auth.confirm-password');
// });

// Fortify::twoFactorChallengeView(function () {
//     return view('auth.two-factor-challenge');
// });

}

...
```

Sebelum membina *view*, kita perlu mendaftarkan **FortifyServiceProvider** dalam **/config/app.php**. Buka fail tersebut dan cari bahagian seperti berikut.

Tambahkan FortifyServiceProvider dalam /config/app.php

```
...

Illuminate\Translation\TranslationServiceProvider::class,
Illuminate\Validation\ValidationServiceProvider::class,
Illuminate\View\ViewServiceProvider::class,

/*
 * Package Service Providers...
 */

App\Providers\FortifyServiceProvider::class,

/*
 * Application Service Providers...
 */
```

```
App\Providers\AppServiceProvider::class,  
App\Providers\AuthServiceProvider::class,  
...
```

Membina Register View

Berikut adalah syarat-syarat yang perlu dipenuhi apabila membina **Register View** untuk **Fortify**.

- <form> mesti menghantar method **POST**.
- <form>, untuk action mesti menggunakan action **{{ route('register') }}**
- <form> mesti menggunakan **@csrf**
- Mesti mempunyai medan berikut :
 - Nama **name**. Jenis *text*
 - Nama **email**. Jenis *text*
 - Nama **password**. Jenis *password*.
 - Nama **password_confirmation**. Jenis *password*.

Rujuk penerangan di dokumentasi di <https://laravel.com/docs/8.x/fortify#registration>

Kita boleh gunakan view yang sama seperti di bahagian **Membina Authentication Ringkas**. Rujuk bahagian **Langkah 2 : Pendaftaran** dari bab tersebut.

Apabila pendaftaran berjaya, pengguna akan dihantar ke **HOME** seperti dalam fail **/app/Providers/RouteServiceProvider.php**.

Tetapan untuk **HOME** di **/app/Providers/RouteServiceProvider.php**

```
<?php  
  
namespace App\Providers;  
  
use Illuminate\Cache\RateLimiting\Limit;  
use Illuminate\Foundation\Support\Providers\RouteServiceProvider  
as ServiceProvider;
```

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\RateLimiter;
use Illuminate\Support\Facades\Route;

class RouteServiceProvider extends ServiceProvider
{
    public const HOME = '/home';
    ...
}
```

Membina Login View

Berikut adalah syarat-syarat yang perlu dipenuhi apabila membina **Login View** untuk **Fortify**.

- <form> mesti menghantar method **POST**.
- <form>, untuk action mesti menggunakan action **{{ route('login') }}**
- <form> mesti menggunakan **@csrf**
- Mesti mempunyai medan berikut :
 - Nama **email**. Jenis *text*
 - Nama **password**. Jenis *password*.

Rujuk penerangan di dokumentasi di <https://laravel.com/docs/8.x/fortify#authentication>

Kita boleh gunakan view yang sama seperti di bahagian **Membina Authentication Ringkas**. Rujuk bahagian **Langkah 3 : Login** dari bab tersebut.

Operasi Lanjutan dengan Fortify

Sekiranya kita perasan daripada Router yang dibina oleh Fortify, ada banyak lagi operasi yang boleh dibuat. Kebanyakannya hanya memerlukan view dan borang seperti yang telah diterangkan di bahagian yang sebelum ini.

Antara operasi Authentication lain yang boleh dibangunkan adalah :

- Proses penetapan semula *password*

- Proses pengesahan email semasa pendaftaran
- Proses **2-factor authentication**

Maklumat lanjut tentang ini semua <https://laravel.com/docs/8.x/fortify>

Namun untuk jalan pintas menggunakan Fortify, kita boleh gunakan pakej sumber terbuka **FortifyUI**.

Membina Authentication dengan FortifyUI

Dokumentasi rasmi **FortifyUI** di <https://github.com/zacksmash/fortify-ui>

Perlu diingatkan bahawa FortifyUI hanya membina View dengan kod HTML yang paling minima. Tiada hiasan dan tetapan CSS.

Pemasangan

Untuk menggunakan **FortifyUI**, mulakan dengan aplikasi Laravel yang baru.

Laksanakan arahan berikut di Terminal.

```
$ composer require zacksmash/fortify-ui
```

Arahan di atas turut disertakan dengan Fortify. Maka Fortify itu sendiri tidak perlu dipasangkan secara berasingan.

Seterusnya, laksanakan arahan berikut di Terminal.

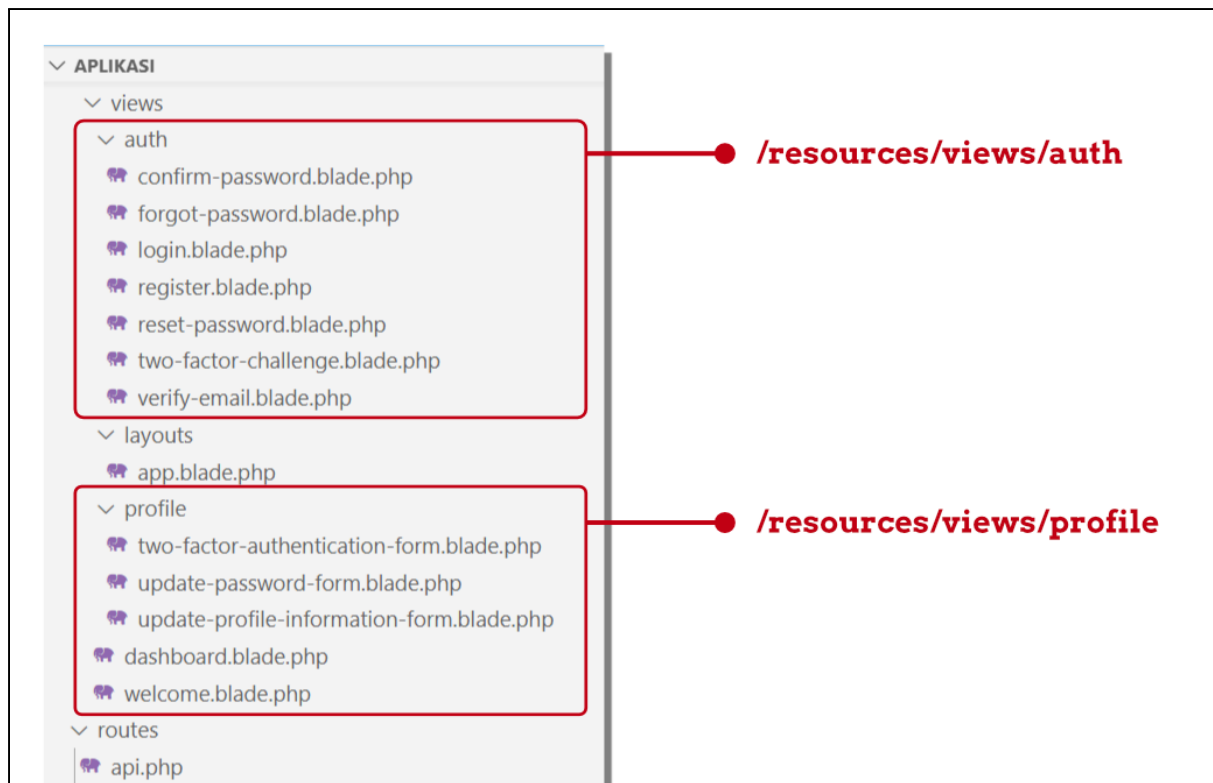
```
$ php artisan fortify:ui
```

Pastikan anda telah mengemaskini fail `.env` dengan tetapan *database* yang betul.

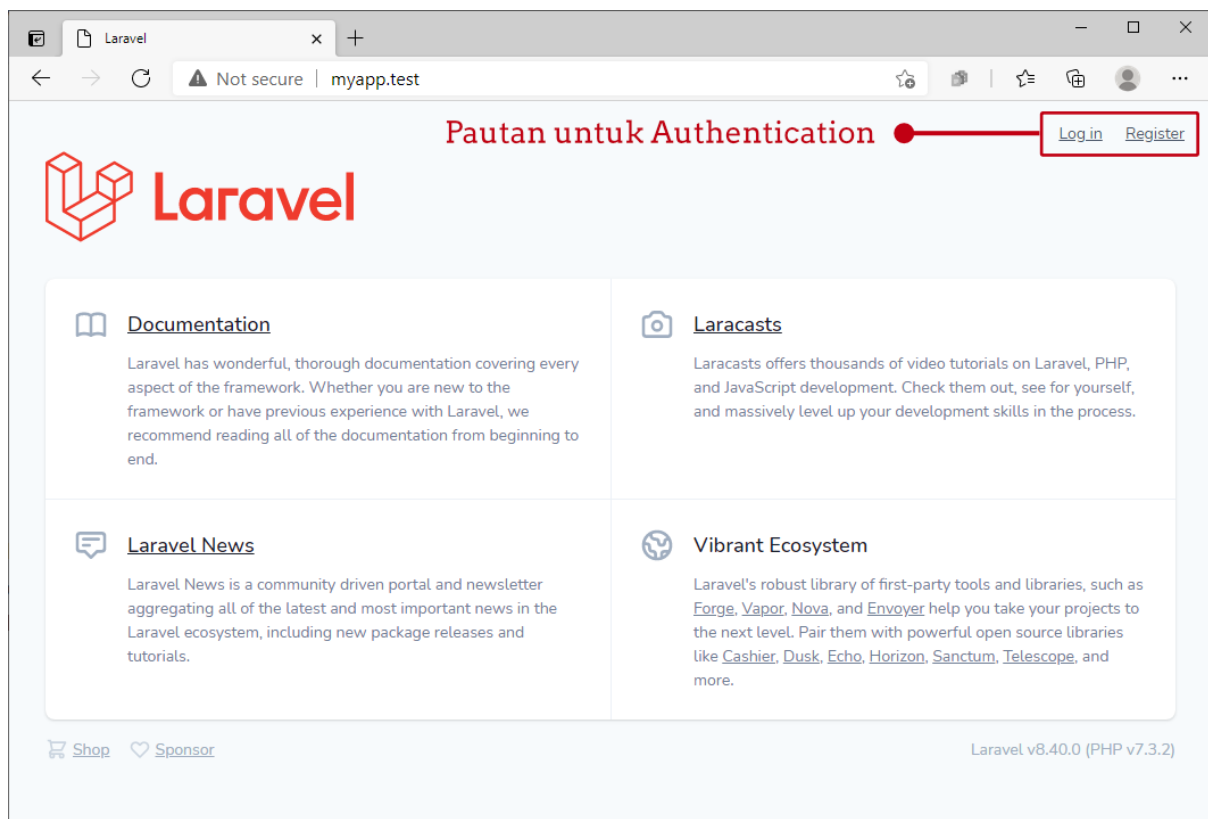
Laksanakan arahan Artisan untuk pemasangan *tables*.

```
$ php artisan migrate
```

Berikut adalah fail *views* yang telah dijana oleh **FortifyUI**.

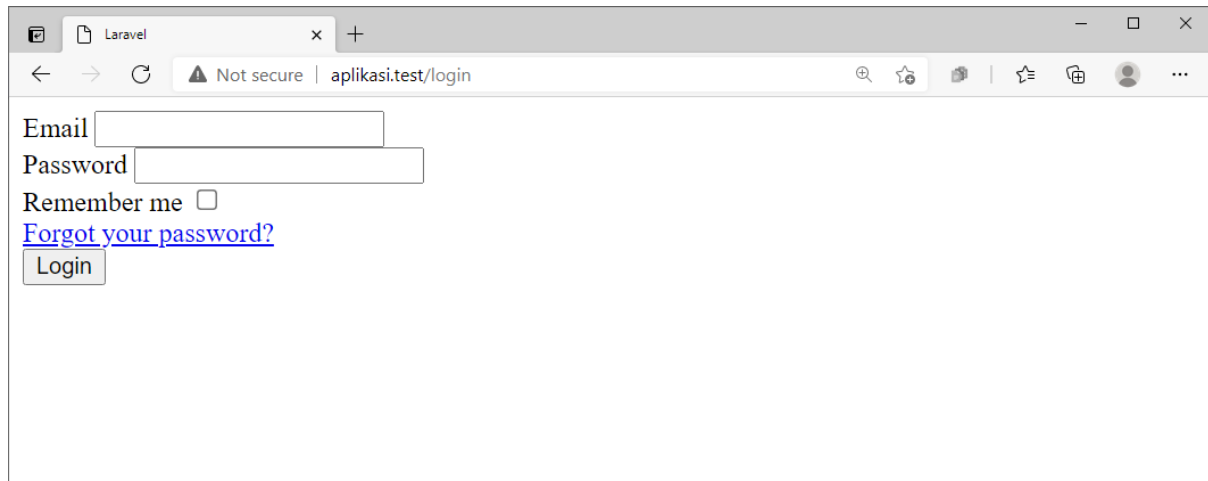


Kita boleh menyemak aplikasi sekarang. Ianya sekarang mempunyai pautan ke **Login** dan **Register**.



Perlu diingatkan bahawa FortifyUI hanya membina View dengan kod HTML yang paling minima. Tiada hiasan dan tetapan CSS.

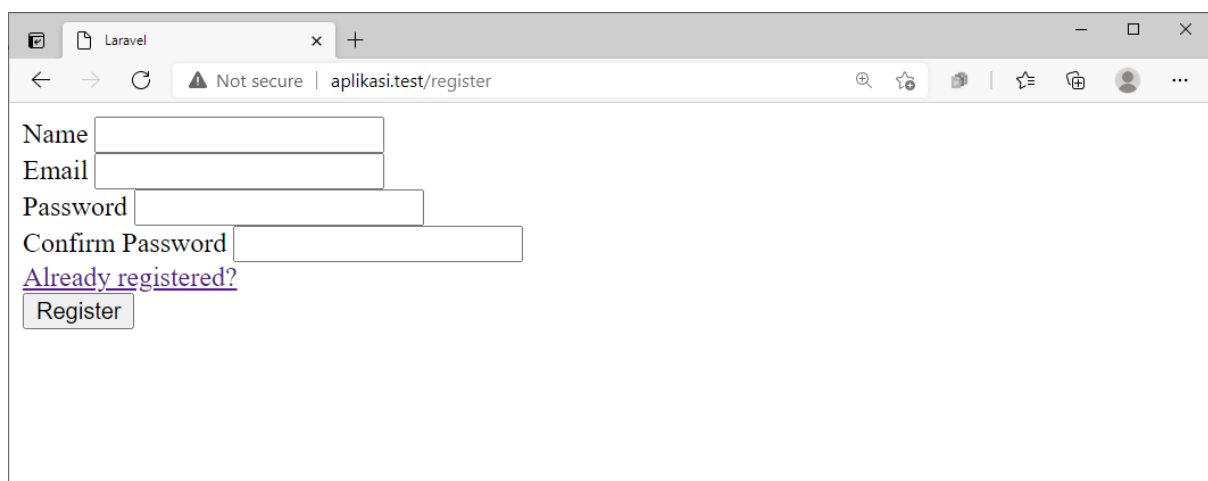
Semak laman Login.



Browser: Laravel x +
Address bar: Not secure | aplikasi.test/login

Email
Password
Remember me ☐
[Forgot your password?](#)

Semak laman **Register**



Browser: Laravel x +
Address bar: Not secure | aplikasi.test/register

Name
Email
Password
Confirm Password
[Already registered?](#)

Dari sini kita boleh mula menghias dan mengubahsuai View mengikut keperluan. Namun sekiranya anda mahukan hiasan CSS segera, anda boleh meneruskan dengan memasang pula **FortifyUI Preset**.

FortifyUI Preset : Hiasan CSS Pantas

FortifyUI menawarkan beberapa pilihan untuk rekabentuk dan hiasan CSS dengan pantas. Berikut adalah Preset yang ditawarkan oleh pembangun FortifyUI dan komuniti.

- **FortifyUIKit**
<https://github.com/zacksmash/fortify-uikit>
- **FortifyUITabler**
<https://github.com/Proxeuse/fortify-tabler>
- **FortifyBulma**
<https://github.com/mikeburrelljr/fortify-bulma>
- **FortifyUITailwind**
<https://github.com/pradeep3/fortify-ui-tailwindcss>

Mencuba FortifyUIKit

Setelah memasang FortifyUI, kita boleh meneruskan pemasangan FortifyUIKit yang menggunakan library UI Kit untuk menghasilkan UI yang cantik dan kemas.

- **FortifyUIKit**
<https://github.com/zacksmash/fortify-uikit>
- **UI Kit**
<https://getuikit.com>

Teruskan dengan arahan berikut di Terminal untuk memasang FortifyUIKit.

```
$ composer require zacksmash/fortify-uikit
```

Arahan di atas memuat-turun FortifyUIKit. Seterusnya, laksanakan arahan pemasangan.

```
$ php artisan fortify:uikit
```


Selepas pemasangan FortifyUIKit, laksanakan arahan berikut agar Laravel boleh membina fail CSS dan Javascript ke *folder* public.


```
$ npm install  
$ npm run dev
```


Kita boleh mencuba hasilnya di <http://aplikasi.test/register>


APLIKASI

Register

Name 

Email 

Password 


Confirm Password 


[REGISTER](#)

[Already registered? Sign in here.](#)

APLIKASI

Login

Email 

Password 

☐ Remember Me

[LOGIN](#)

[Forgot your password? | Sign up](#)

Memasang Semua Ciri-Ciri Fortify

Ciri-ciri Fortify turut dilengkapi dengan berikut :

- *Verify* email semasa pendaftaran
- Forget password
- Reset Password
- Verify with Password
- Two Factor Authentication

Berikut adalah langkah-langkah untuk mempersiapkan kesemuanya. Perhatikan kod bertanda kuning untuk ditambah atau dikemaskini.

/app/Models/User.php

```
<?php

namespace App\Models;

use Illuminate\Notifications\Notifiable;
use Laravel\Fortify\TwoFactorAuthenticatable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable implements MustVerifyEmail
{
    use HasFactory, Notifiable, TwoFactorAuthenticatable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];
}
```

```
];  
...
```

/config/fortify.php -- di hujung, pastikan semua tidak berkomen

```
...  
    'features' => [  
        Features::registration(),  
        Features::resetPasswords(),  
        Features::emailVerification(),  
        Features::updateProfileInformation(),  
        Features::updatePasswords(),  
        Features::twoFactorAuthentication([  
            'confirmPassword' => true,  
        ]),  
    ],
```

/app/Providers/FortifyUIServiceProviders.php - di hujung fail, pastikan semuanya tidak berkomen

```
/**  
 * Bootstrap any application services.  
 *  
 * @return void  
 */  
public function boot()  
{  
    Fortify::loginView(function () {  
        return view('auth.login');  
    });  
  
    Fortify::registerView(function () {  
        return view('auth.register');  
    });  
  
    Fortify::requestPasswordResetLinkView(function () {
```

```
        return view('auth.forgot-password');
    });

    Fortify::resetPasswordView(function ($request) {
        return view('auth.reset-password', ['request' =>
$request]);
    });

    Fortify::verifyEmailView(function () {
        return view('auth.verify-email');
    });

    Fortify::confirmPasswordView(function () {
        return view('auth.confirm-password');
    });

    Fortify::twoFactorChallengeView(function () {
        return view('auth.two-factor-challenge');
    });
}
}
```

Pastikan fail .env juga telah dikemaskini untuk penghantaran email pengesahan.

Operasi Authentication

Berikut adalah beberapa operasi yang lazim digunakan bersama *Authentication*.

Mendapatkan Maklumat Pengguna

Pengguna yang telah boleh diakses dengan Facade Auth. Pastikan namespace untuk facade **Auth** telah diimport dengan **use**.

```
use Illuminate\Support\Facades\Auth;

$user = Auth::user();
$id = Auth::id();
```

Ini juga boleh digunakan dalam Views

```
Selamat datang {{ Auth::user()->name }}!
```

Memeriksa Pengguna Telah Login

Cara berikut, kita boleh memeriksa sama ada pelawat telah login ataupun tidak. Pastikan namespace untuk facade **Auth** telah diimport dengan **use**.

```
use Illuminate\Support\Facades\Auth;

if (Auth::check()) {
    // The user is logged in...
}

if (!Auth::check()) {
    // The user is NOT logged in...
}
```

Dalam view, kita juga boleh gunakan seperti berikut :

```
@auth
    // The user is authenticated...
@endauth

@guest
    // The user is not authenticated...
@endguest
```

Menghantar Pengguna Belum Login Ke Laman Lain

Apabila pelawat belum login cuba mengakses laman yang dilindungi, mereka akan dihantar ke lama login. Ini boleh diubah dengan mengubahsuai fail `Route::get('profile', function () {`

```
    // Only authenticated users may enter...
})->middleware('auth');/app/Http/Middleware/Authenticate.php. Kemaskini fungsi redirectTo().
```

/app/Http/Middleware/Authenticate.php

```
/**
 * Get the path the user should be redirected to.
 *
 * @param  \Illuminate\Http\Request  $request
```

```
* @return string
*/
protected function redirectTo($request)
{
    return route('login');
}
```

Melindungi URL

Lebih mudah untuk mengawal akses ke website dengan mengawal URL di router. Kita boleh mengubahsuai fail `/routes/web.php` seperti berikut :

`/routes/web.php`

```
Route::get('profile', function () {
    // Only authenticated users may enter...
})->middleware('auth');
```

Debugging

Debugging adalah satu proses untuk mengetahui ralat yang berlaku dan bagaimana untuk memeriksa dan memperbaikinya.

Fungsi dd()

Fungsi **dd()** sangat berguna untuk menyemak kandungan *variable* ataupun objek. **dd()** adalah ringkas untuk **dump and die**, yang membawa maksud dump variable and die application. Selepas melaksanakan fungsi ini, aplikasi akan terhenti dan tidak akan diteruskan kepada kod yang seterusnya.

Contoh : menggunakan fungsi dd()

```
<?php
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;

class TestController extends Controller
{
    public function show(User $user) {
        dd($user);
    }
}
```

```
App\Models\User {#1065}
  #fillable: array:3 [▶]
  #hidden: array:2 [▶]
  #casts: array:1 [▶]
  #connection: "mysql"
  #table: "users"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #attributes: array:10 [▶]
  #original: array:10 [▶]
  #changes: []
  #classCastCache: []
  #dates: []
  #dateFormat: null
  #appends: []
  #dispatchesEvents: []
  #observables: []
  #relations: []
  #touches: []
  +timestamps: true
  #visible: []
  #guarded: array:1 [▶]
  #rememberTokenName: "remember_token"
}
```

Debugbar

Dokumentasi penuh di <https://github.com/barryvdh/laravel-debugbar>

Debugbar memberikan maklumat yang lebih terperinci apabila kita sedang membangunkan aplikasi. Debugbar perlu dipasang berasingan.

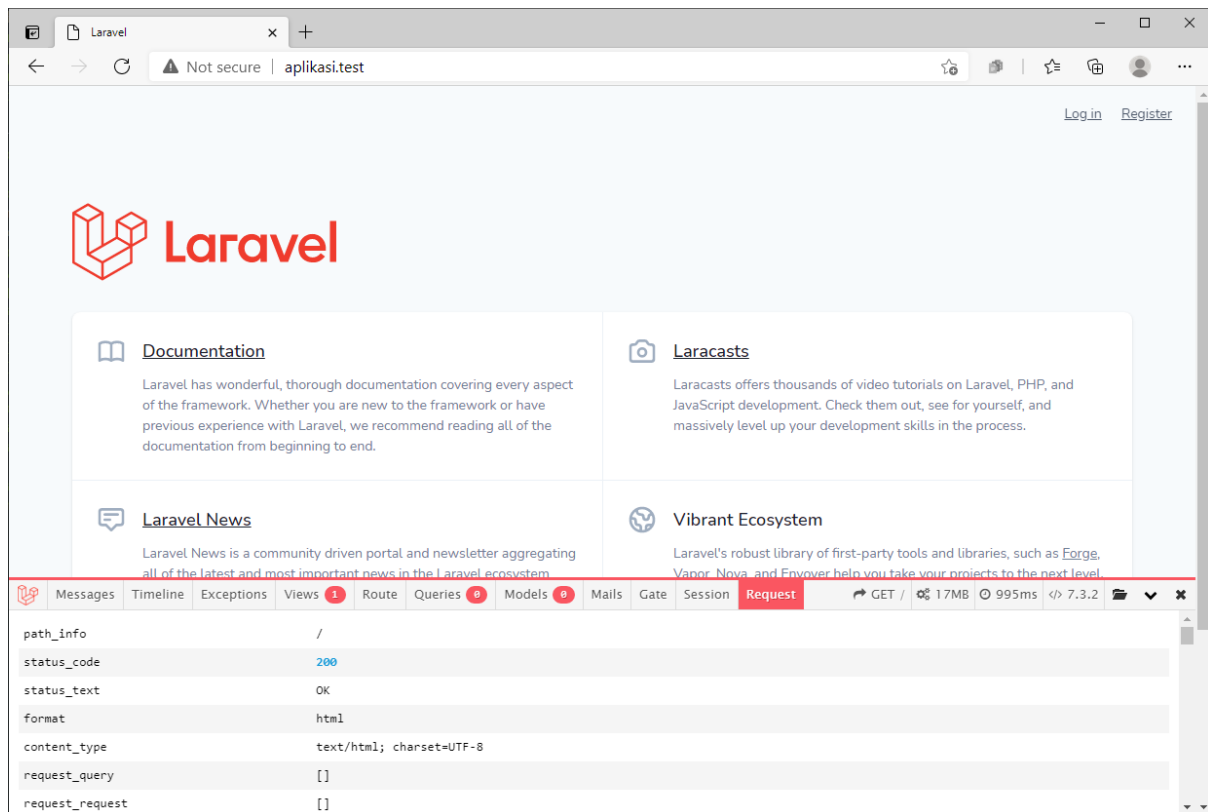
Laksanakan arahan berikut di Terminal.

```
$ composer require barryvdh/laravel-debugbar --dev
```

Pastikan tetapan di fail .env untuk **APP_DEBUG** ditetapkan kepada **true**.

```
APP_NAME=APLIKASI
APP_ENV=local
APP_KEY=base64:reimFRuu6GkOw7dHccQzrsTX+ItyMtVGEDPLUj8v2Fw=
APP_DEBUG=true
APP_URL=http://aplikasi.test
```

Semak aplikasi untuk melihat Debugbar.



Antara maklumat yang boleh diberikan oleh Debugbar adalah :

- Jumlah *database queries*
- Tempoh masa yang diambil
- Views yang digunakan
- *Variable* yang wujud
- Maklumat di Session
- dan lain-lain lagi.

Kita juga boleh meletakkan mesej tertentu dari dalam aplikasi untuk dipaparkan oleh Debugbar.

Kemaskini fail `/config/app.php` di bahagian aliases.

```
'aliases' => [
    ...
    ...
    'Debugbar' => Barryvdh\Debugbar\Facade::class,
],
```

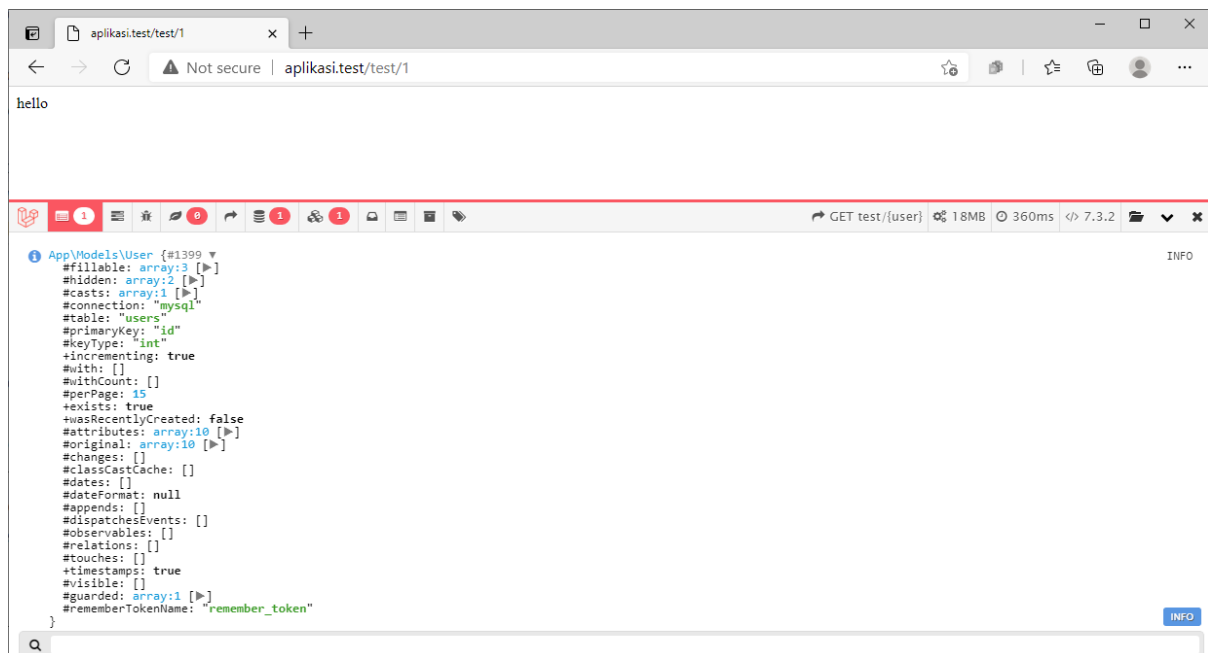
Contoh : memaparkan info ke Debugbar

```
<?php
```

```
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;

class TestController extends Controller
{
    public function show(User $user) {
        echo "hello";
        \Debugbar::info($user);
    }
}
```



Fungsi lain dalam Debugbar

```
Debugbar::info($object);
Debugbar::error('Error!');
Debugbar::warning('Watch out...');
Debugbar::addMessage('Another message', 'mylabel');
Debugbar::startMeasure('render', 'Time for rendering');
```



```
Debugbar::stopMeasure('render');
Debugbar::addMeasure('now', LARAVEL_START, microtime(true));
Debugbar::measure('My long operation', function() {
    // Do something...
});
```

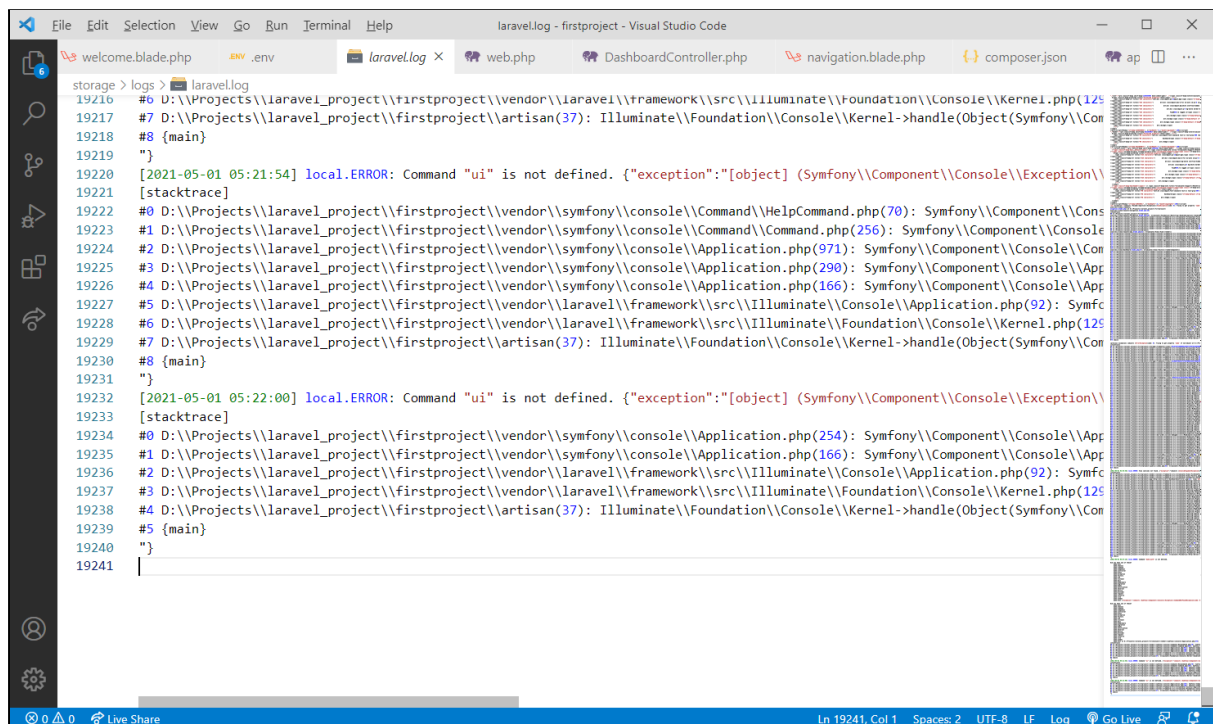
Membaca Log

Dalam persekitaran *production*, kita akan pastikan kita telah menetapkan fail `.env` dengan **production**.

Contoh : fail `.env`

```
APP_NAME=APLIKASI
APP_ENV=production
APP_KEY=base64:reimFRuu6GkOw7dHccQzrsTX+ItyMtVGEDPLUj8v2Fw=
APP_DEBUG=false
APP_URL=http://aplikasi.test
```

Dalam keadaan begini, Debugbar tidak akan muncul. Paparan ralat juga tidak akan muncul. Untuk mendapatkan sebarang mesej ralat, kita boleh menyemak fail `/storage/logs/laravel.log`.



Rujukan Tambahan

Helper

Helper adalah beberapa fungsi yang digunapakai dalam framework Laravel itu sendiri. Kita juga menggunakannya mengikut keperluan. Antara Helper yang berguna adalah seperti untuk manipulasi teks dan *string*, operasi Array dan lain-lain.

Antara helper yang sering kita gunakan ada seperti `route()`, `dd()`.

Senarai penuh ada di dokumentasi rasmi <https://laravel.com/docs/8.x/helpers>

Facade

Facade adalah library dengan pelbagai fungsi yang boleh diakses secara statik (contohnya: **ClassName::function()**).

Facade yang kita gunakan adalah seperti DB, Auth, Hash, Request, Response.

Senarai penuh ada di dokumentasi rasmi <https://laravel.com/docs/8.x/facades>

Service Provider

Di peringkat asas, kita mungkin buat sedikit modifikasi dalam *service provider*. Tapi kita tak membina *service provider* kita sendiri. Service provider mengandungi tetapan untuk fungsi-fungsi yang khusus. Antaranya adalah `DebugBarServiceProvider`, `AppServiceProvider`, `AuthServiceProvider` dan lain-lain.

Rujukan rasmi berkenaan Service Providers <https://laravel.com/docs/8.x/providers>

Tarikh dan Masa dengan Carbon

Dokumentasi rasmi tentang Carbon di <https://carbon.nesbot.com/docs/>

Apabila menerima data dari Model, dan ada data berbentuk datetime atau timestamp, ianya akan dipulangkan sebagai objek Carbon. Carbon adalah library yang memudahkan proses memformatkan tarikh dan masa.

Contoh : Memaparkan tarikh dalam form yang lebih mudah dibaca

```
<table>
  <thead>
    <tr>
```

```
        <th>Name</th>
        <th>Created</th>
    </tr>
</thead>
<tbody>
    @foreach($users as $user)
        <tr>
            <td>{{ $user->name }}</td>
            <td>{{ $user->created_at->diffForHumans() }}</td>
        </tr>
    @endforeach
</tbody>
</table>
```