**Step to start**

* Install via composer
* Wait until download complete.
* Open terminal in the directory and do “php artisan serve”
* Ctrl + c to quit
* Code . to call vs code

**ROUTE**

**Defining route**

Normal route

Route::get(‘/route, function(){

return view(‘view);

});

View Route, can be shortened with

Route::view( ‘/route’, ’view’, [‘key’ => ‘value’] );

Controller route

Route::get( ‘/route’, ‘ControllerName@methodName’ );

Route::get( ‘/route’, [ ControllerName::Class, ’methodName’ ] ); //in version 8

*Method will define to which view the route will go*

Route Name, you can give a route a name by :

Route::get( ‘/route’, ‘ControllerName@methodName’ )->name(‘theRouteName’);

you can use route name to navigate to each page by

{{ route(‘theRouteName’) }};

*If not using route name then you can use ->* {{ url(‘/where’) }}

**Prefix**

Prefix add prefix to the link before the route that is included in the group.

Route::group ( [ ‘prefix’ => ‘page’ ], function(){

Route::get( ‘/home, ‘ControllerName@home )->name(‘index’);

Route::get( ‘/about, ‘ControllerName@about )->name(‘aboutus’);

}); //when the link is clicked or the route is used, the url will show page/home or page/about instead of home and about

//caution! To use prefix you must use Route name in the url .

Directive

@section( ‘thisSection’ )

Section will group a certain block of codes to indicate those codes in their representative sections, e.q : content

@endsection

*@section(‘PageTitles’, ‘The titles’) -> you can also give parameters after it usually to define each views titles, meta description, meta keywords and any section which is unique, remember that section and yield is casesensitive so in this case yield should be @yield(‘PageTitles’).*

@yield( ‘thisSection’ )

Yield will indicate where the codes in @section will be placed.

@component(‘componentFile’)

@slot(‘theVariable’)

text

@endslot

@endComponent

Inside the component you can call your components by wrapping them in their slots. The text inside slot will replace the variable with the same name in the componentFile( {{$theVariable}} ). You can use this to change the attribute of style, text and many others to make them more dynamic.

**Database MIGRATION**

php artisan make:migration => to make migration

php artisan migrate => to migrate

php artisan migrate:rollback

**CHECK DOCUMENTATION => Database -> Migration**

create table

add column

modify column

renaming column

Note: to do task such as renaming or removing column you might require to put

doctrine/dbal = “\*” (default version). under the required-dev in composer.json then do **composer update** in the terminal or just ***require doctrine/dbal.*** But, just if you succeed in adding the dependencies but suffer some errors like

*‘"Doctrine\DBAL\Driver\PDOMySql\Driver" not found* ’ you will need to downgrade the doctrine/dbal = “^2.10“ (version 2)

**Authentication**

**CHECK DOCUMENTATION => Security -> authentication**

Install laravel/ui using ***composer require laravel/ui “^1.0”***

laravel/ui Compatibility

Version Laravel Version

1.x 5.8, 6.x

2.x 7.x

3.x 8.x

Laravel/ui

To edit the view or php files provided by the laravel/ui

Go to vendor>laravel>framework>src/illuminate>foundation>auth

There exist list of methods but you shouldn’t edit them, instead copy the methods to the controller files and override them there.

{{ Auth::user()->name }} means the name of the current authenticated user model’s name by placing it inside {{ }} means to output the user’s name there is also other attribute depends on the model

To edit user :

    public function profileUpdate(Request $request){

        $user = Auth::user();

        $user->name = $request['name'];

        $user->email = $request['email'];

        $user->save();

       return back();

    }

$request will store the data from Request object. Request is where to obtain an instance of the current HTTP request, in this example, is the user’s name and email from the post method

Back() is global method to redirect to previous view

To do validation just put it before the operation, if something went wrong the rest of the code won’t run

public function profileUpdate(Request $request){

        //validation

        $request->validate([

            'name' => 'required|min:2|string|max:255',

            'email' => 'required|email|max:255'

        ]);

        $user = Auth::user();

        $user->name = $request['name'];

        $user->email = $request['email'];

        $user->save();

       return back()->with(‘flashMessage’,’profile updated’);

    }

**Then you can use @error or @if ($error) method in the view to print the error. Look at the documentation in The Basics -> Validation to see how to implement @error , @if and other validation rules available**