UNIT-4

PHP Framework

Framework

- Set of rules, idea, beliefs which we use in order to deal with problems or decide what to do.
- S/w Framework:
  - Platform for developing s/w applications.
  - Includes Class, Functions, DB config, Packages

Laravel

- It is an open source PHP framework, developed by Taylor Otwell in 2011.
- It follows MVC(Model-View-Controller) architecture.
- Has features like:
  - Built in session management
  - DB management
  - Composer
  - Eloquent ORM (Object Relational Mapper)
- Full stack framework to develop web application from scratch using DB wrapper called Eloquent ORM and its own templating engine called Blade.
- It has welcoming and supportive community.

Laravel

- Advantages:
  - Web applications becomes more scalable.
  - It reuses the components from other framework in developing web applications that saves time.
  - Includes namespaces and interfaces to organize and management of resources.
  - Websites developed in Laravel is secure and prevents several web attacks.

Laravel

- Features:
  - Modularity: provides 20 built in libraries and modules to enhance the application.
  - Testability: includes features for maintaining code as per requirement and helper to test various test cases.
  - Routing: helps to scale the application in better way.
  - Configuration Management: Web application developed in Laravel will be running on different environment.

Laravel

- Features:
  - Query Builder & ORM: incorporates query builder to querying DB, Provides ORM (handles DB records and working as layer of abstraction) and ActiveRecord implementation called Eloquent.
  - Schema Builder: maintains the DB definition and schema in PHP. Also keep track of changes with respect to DB migration.
  - Template Engine: Uses “Blade Template Engine”- lightweight template language to design hierarchical blocks and layouts with predefined blocks for dynamic content.
  - E-mail: include mail class for sending mail with rich content and attachments.

Laravel

- Features:
  - Authentication: Eases designing authentication as it includes features like register, forgot password and send password reminders.
  - Redis: Uses it to connect to an existing session and general purpose cache. It interacts directly with session.
  - Queues: include queue service like emailing large number of users or specified scheduled job. They help in completing task w/o waiting for previous task to be completed.

Laravel

- Features:
  - Event and Command Bus: Command bus helps in executing commands and dispatch events in simple way.
  - Artisan Command Tool: offers a range of commands for common tasks

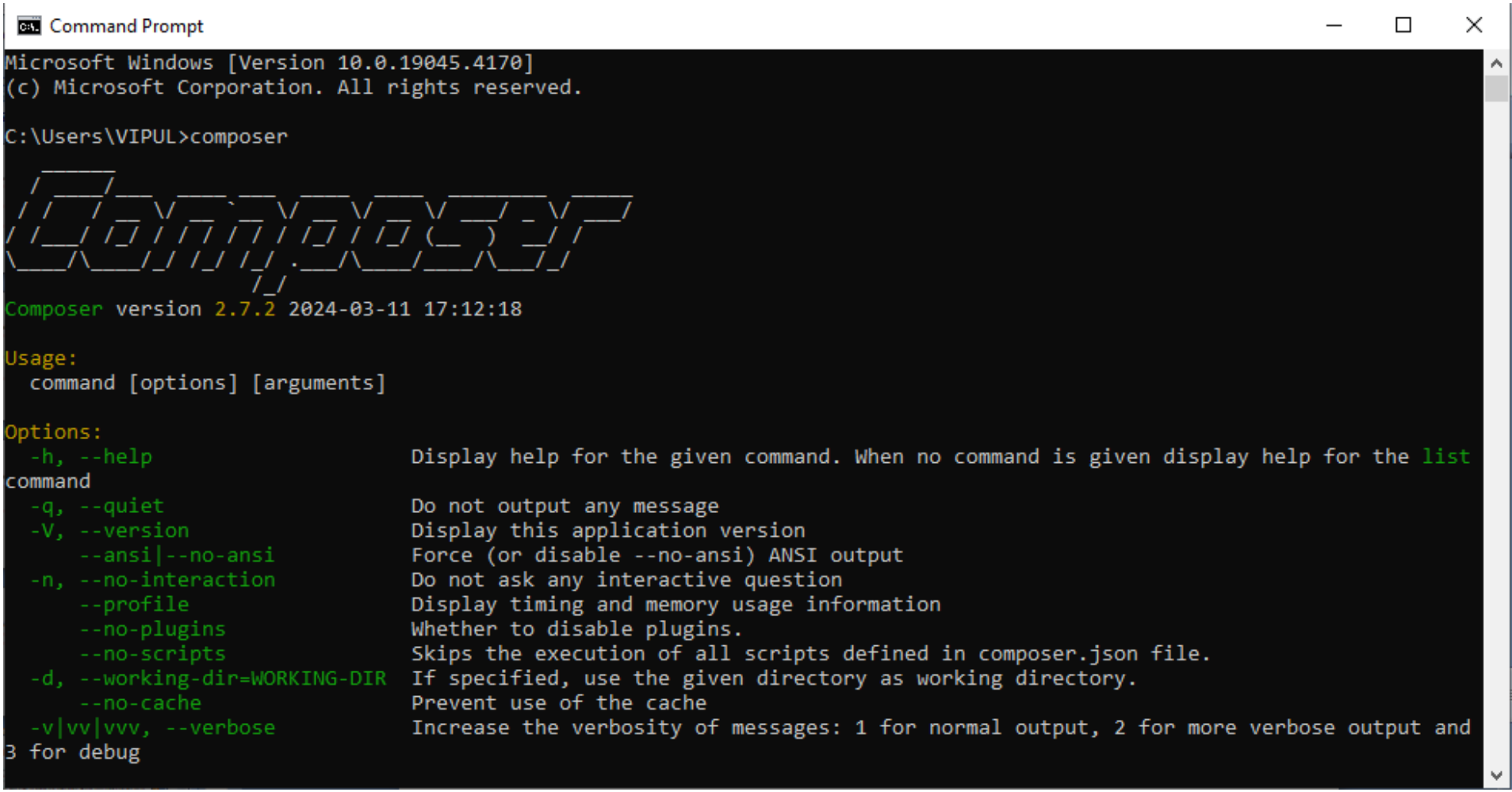
Installing Laravel

- Two methods to install Laravel
  - Composer
  - New Laravel Installer

Install Composer on Windows

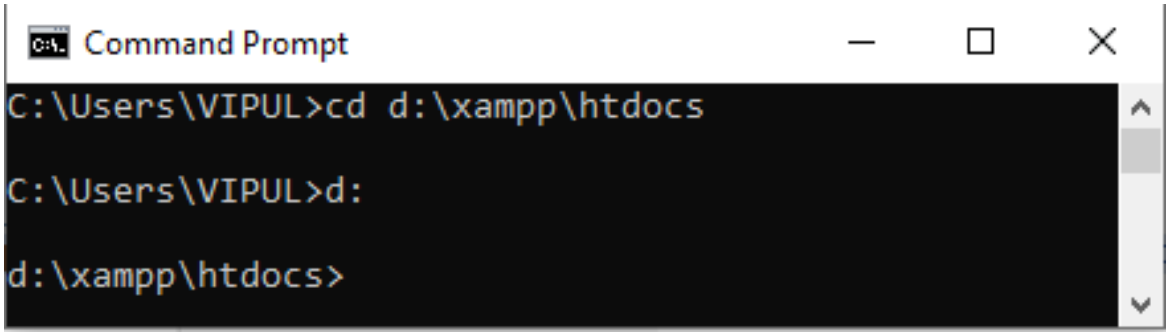
- For managing dependencies, Laravel uses composer. Make sure you have a Composer installed on your system before you install Laravel. In this chapter, you will see the installation process of Laravel.
- You will have to follow the steps given below for installing Laravel onto your system −
  - **Step 1** − Visit the following URL and download composer to install it on your system. https://getcomposer.org/download/
  - **Step 2 −** After the Composer is installed, check the installation by typing the Composer command in the command prompt as shown in the following screenshot.

Install Composer on Windows



Install Laravel on Windows

1. Open the Windows command prompt and navigate to the “htdocs” directory in xampp.

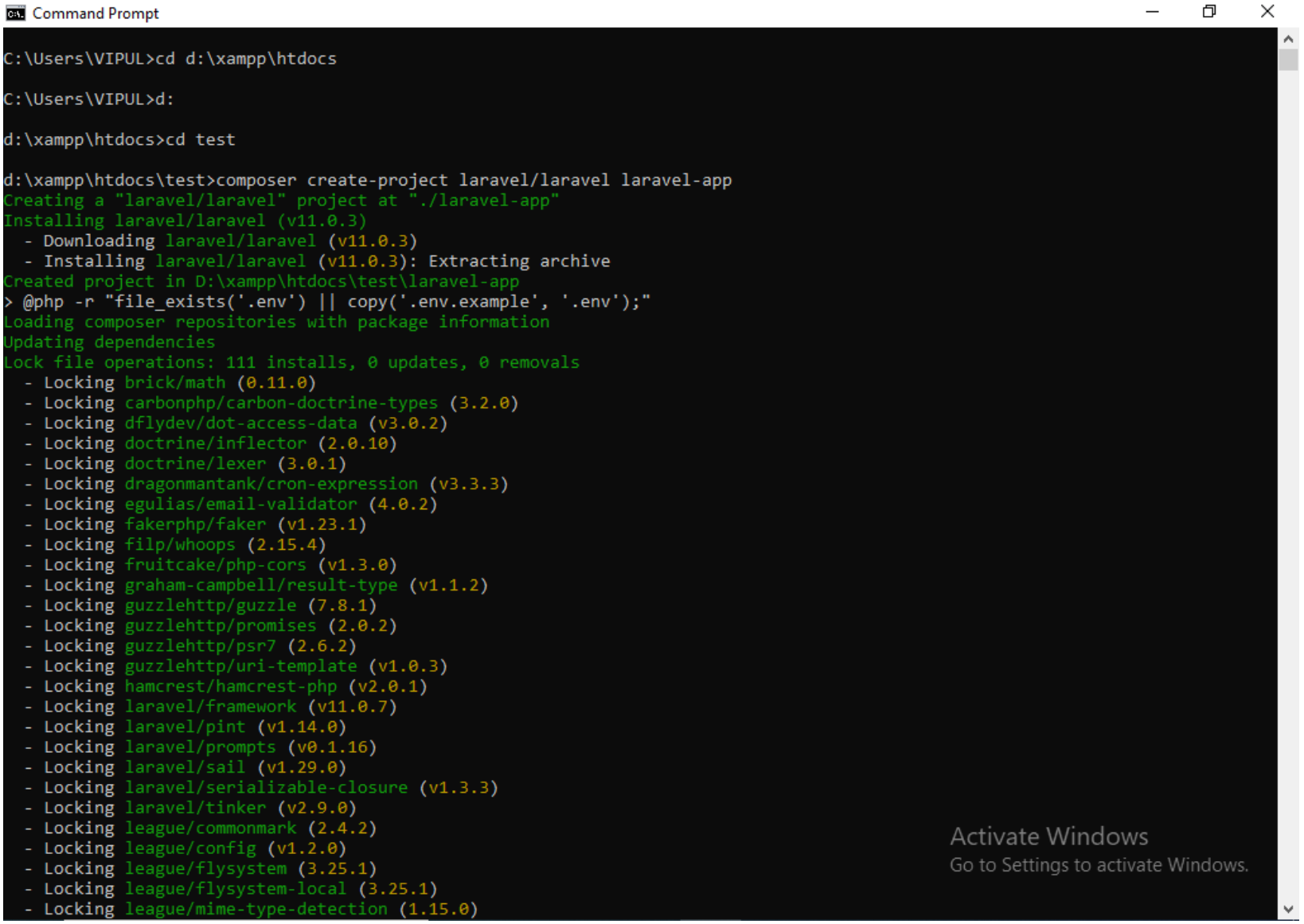


2. Run the following command to install Laravel in D:\xampp\htdocs\test\laravel-app directory.

```
composer create-project laravel/laravel laravel-app
```

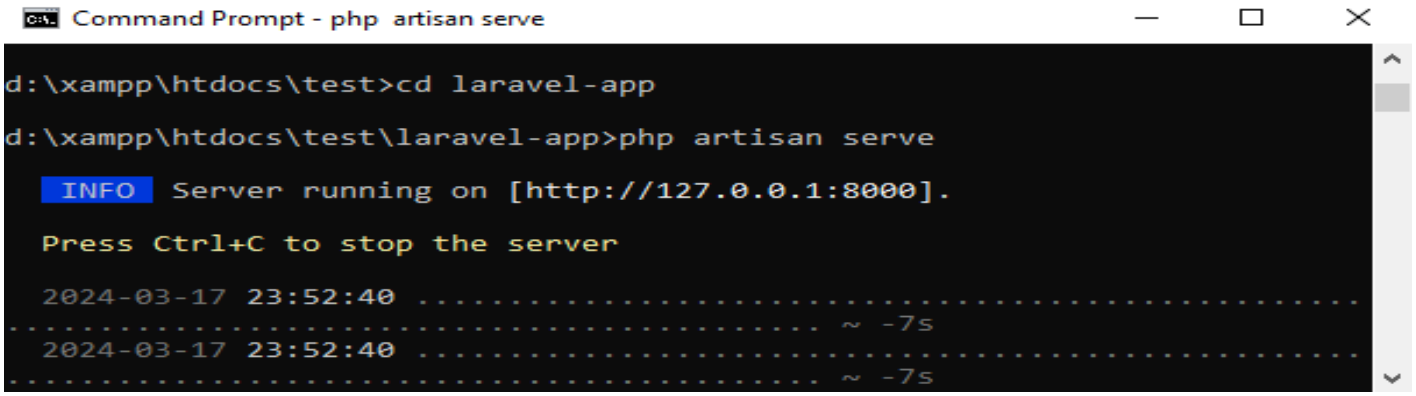
Laravel installation will start, wait for completion.

Install Laravel on Windows



Test Laravel Installation on Windows

- You may start Laravel local development server using the following command.



- Run the development URL on the browser (http://localhost:8000/).If Laravel is installed successfully on the Windows server, the following screen will appear.

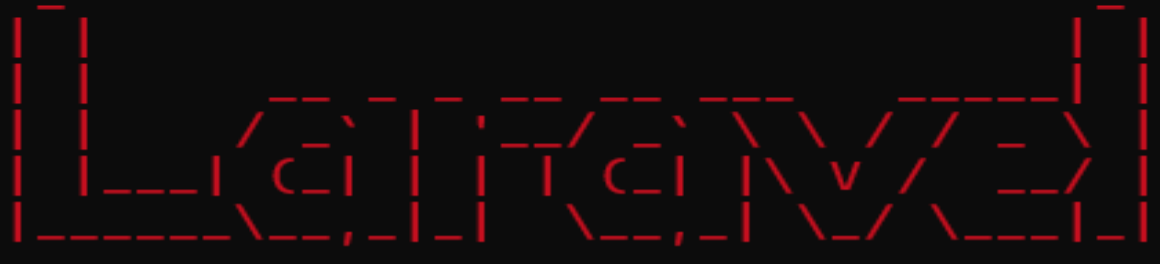
Test Laravel Installation on Windows

- You can install laravel globally by following command.

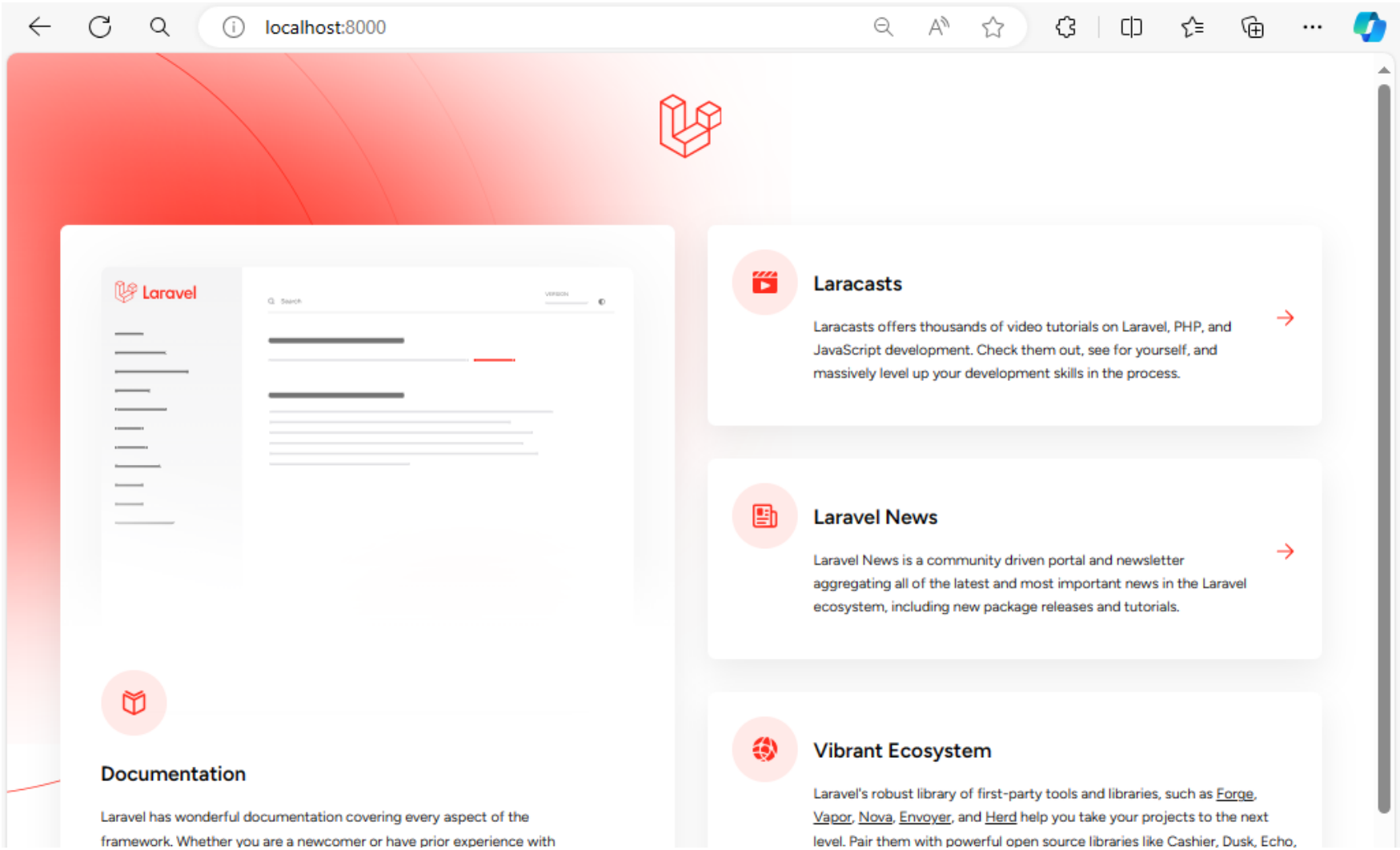
```
C:\Users\vskjh>composer global require laravel/installer
Changed current directory to C:/Users/vskjh/AppData/Roaming/Composer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
```

```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

F:\xamp\htdocs\IIIT_WEB>laravel new lara-app-global
```



Test Laravel Installation on Windows

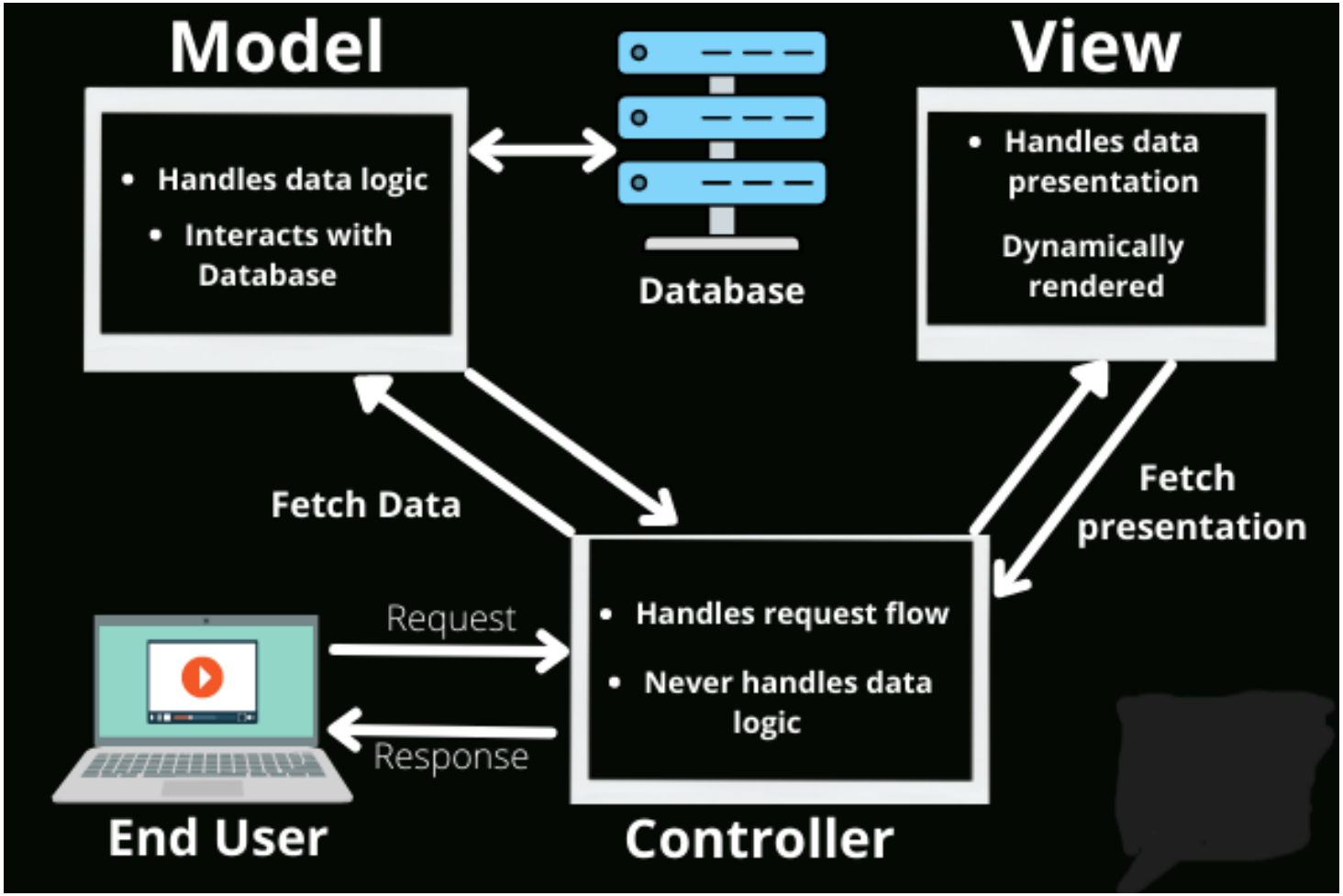


Laravel

- MVC:
  - Its an architecture pattern that enforces separation between Models (information), Controller (User interaction) and View (model's display)
  - It isolates the business logic and presentation layer from each other.
  - MVC helps us to change, extend and maintain applications easily.

Laravel

- MVC:


Model
Handles data logic
Interacts with Database
Database
View
Handles data presentation
Dynamically rendered
Fetch Data
Fetch presentation
Request
Response
Handles request flow
Never handles data logic
End User
Controller

Laravel

- Model:
  - The **Model** component corresponds to all the data-related logic that the user works with.
  - This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
  - It can add or retrieve data from the database.
  - It responds to the controller's request because the controller can't interact with the database by itself.
  - The model interacts with the database and gives the required data back to the controller.

Laravel

- View:
  - The **View** component is used for all the UI logic of the application.
  - It generates a user interface for the user.
  - Views are created by the data which is collected by the model component but these data aren't taken directly but through the controller.
  - It only interacts with the controller.

Laravel

- Controller:
  - The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary.
  - The controller doesn’t have to worry about handling data logic, it just tells the model what to do.
  - It processes all the business logic and incoming requests, manipulates data using the **Model** component, and interact with the **View** to render the final output.

Laravel

- Features of MVC:
  - It provides a clear separation of **business logic, Ul logic, and input logic.**
  - It offers full control over your HTML and URLs which makes it easy to design web application architecture.
  - It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs.
  - It supports **Test Driven Development (TDD).**

MVC Routing

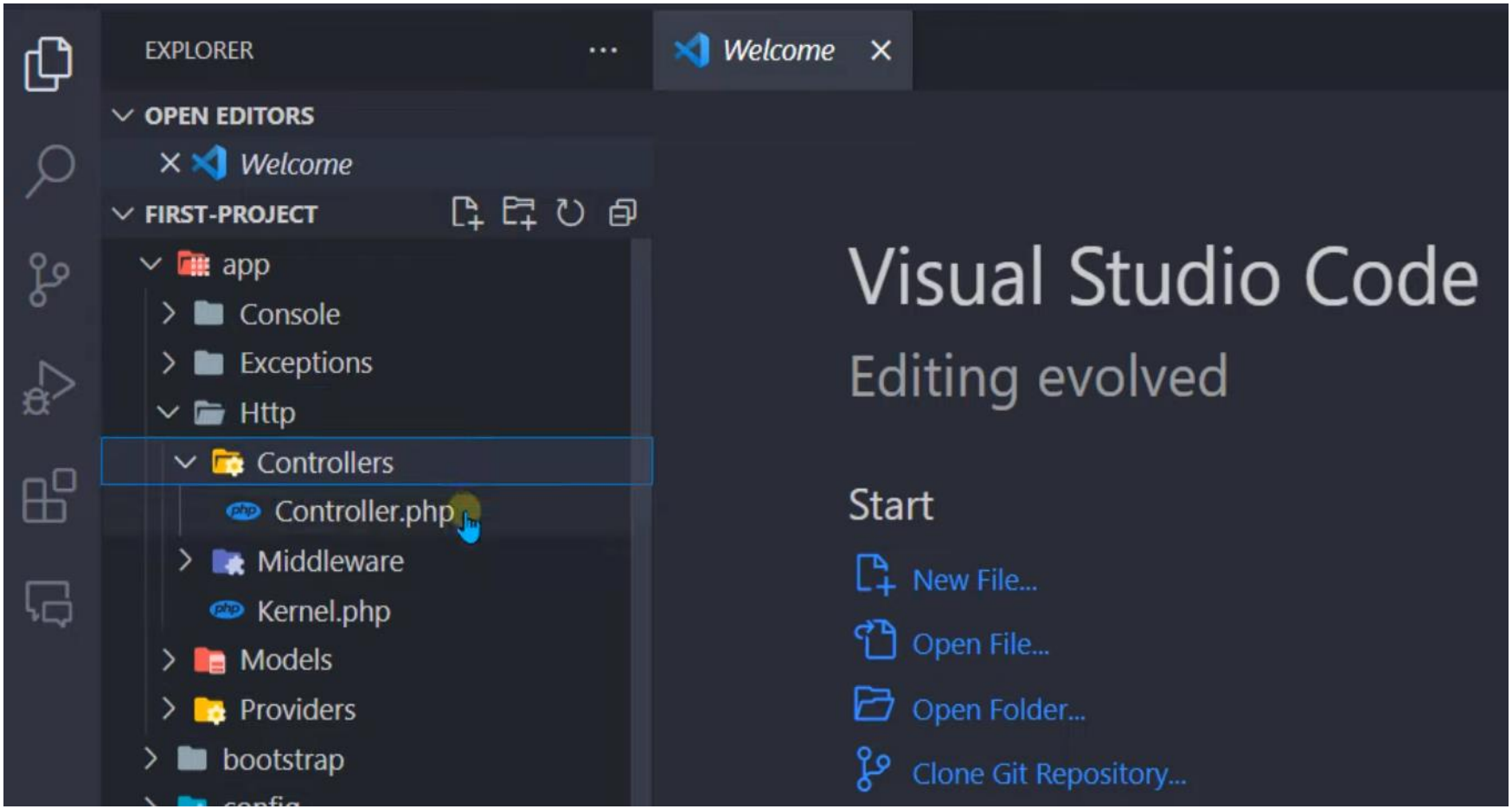
- It is a process of mapping the browser request to the controller action and return response back.
- Each MVC application has default routing.
- We can set custom routing for newly created controller.
- We can go to routes/web.php to modify application's routes.
- In the newer version, Laravel 5.3, we have a folder named 'routes', where we can find the following files:
  - api.php
  - console.php
  - web.php
- For this new version, the routes for your controllers, you can put inside web.php file

Laravel Folder & File Structure

1. Model Folder → Database / SQL Queries Handling Files
2. Controller Folder → Business Logics Files
3. View Folder → HTML Files
4. Routing Folder → URL Defining Files
5. Assets Folder (Public Folder) → Images / Fonts / Music / Videos Files
   CSS / JavaScript Files

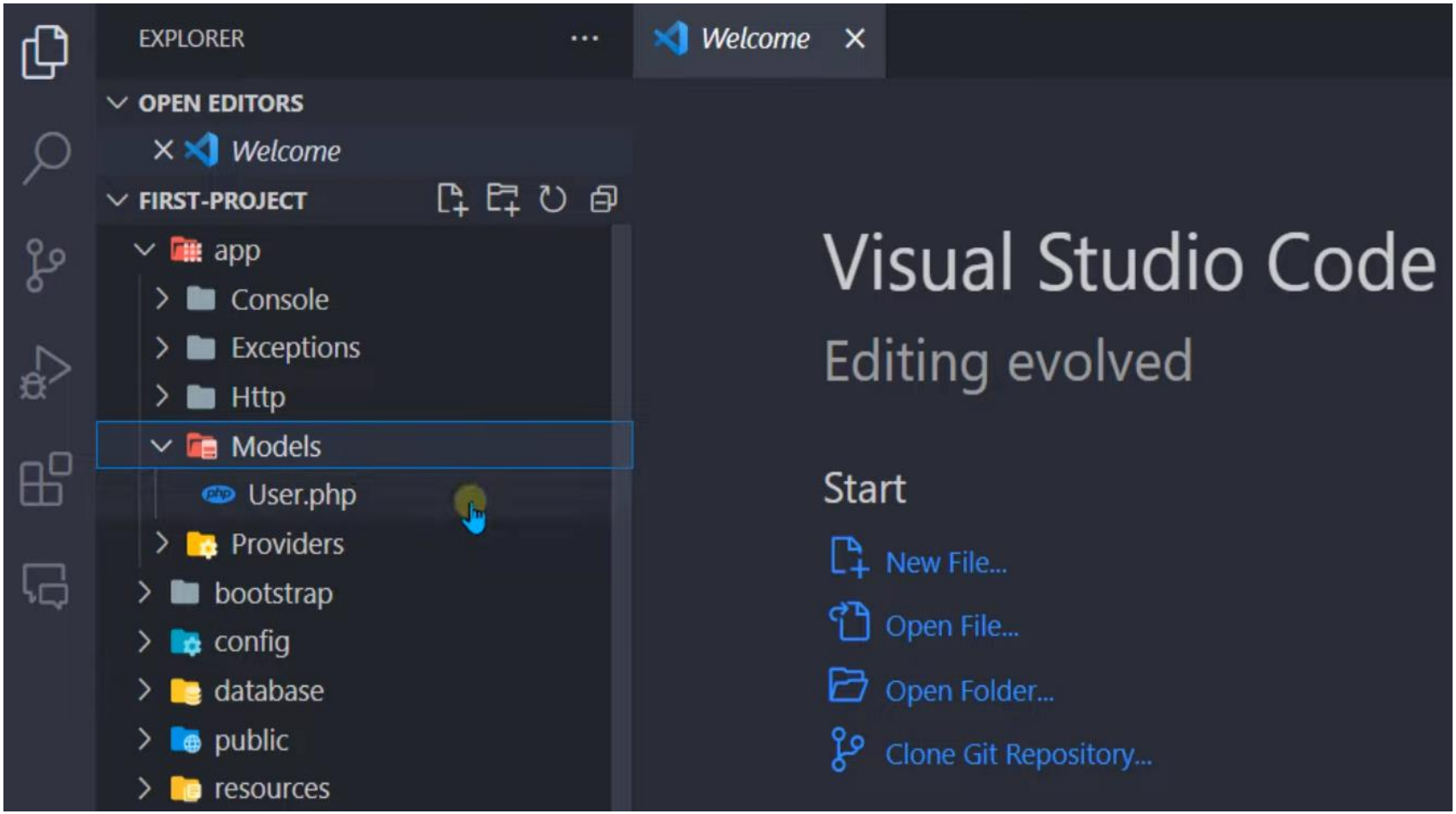
Laravel Folder & File Structure

- Controller



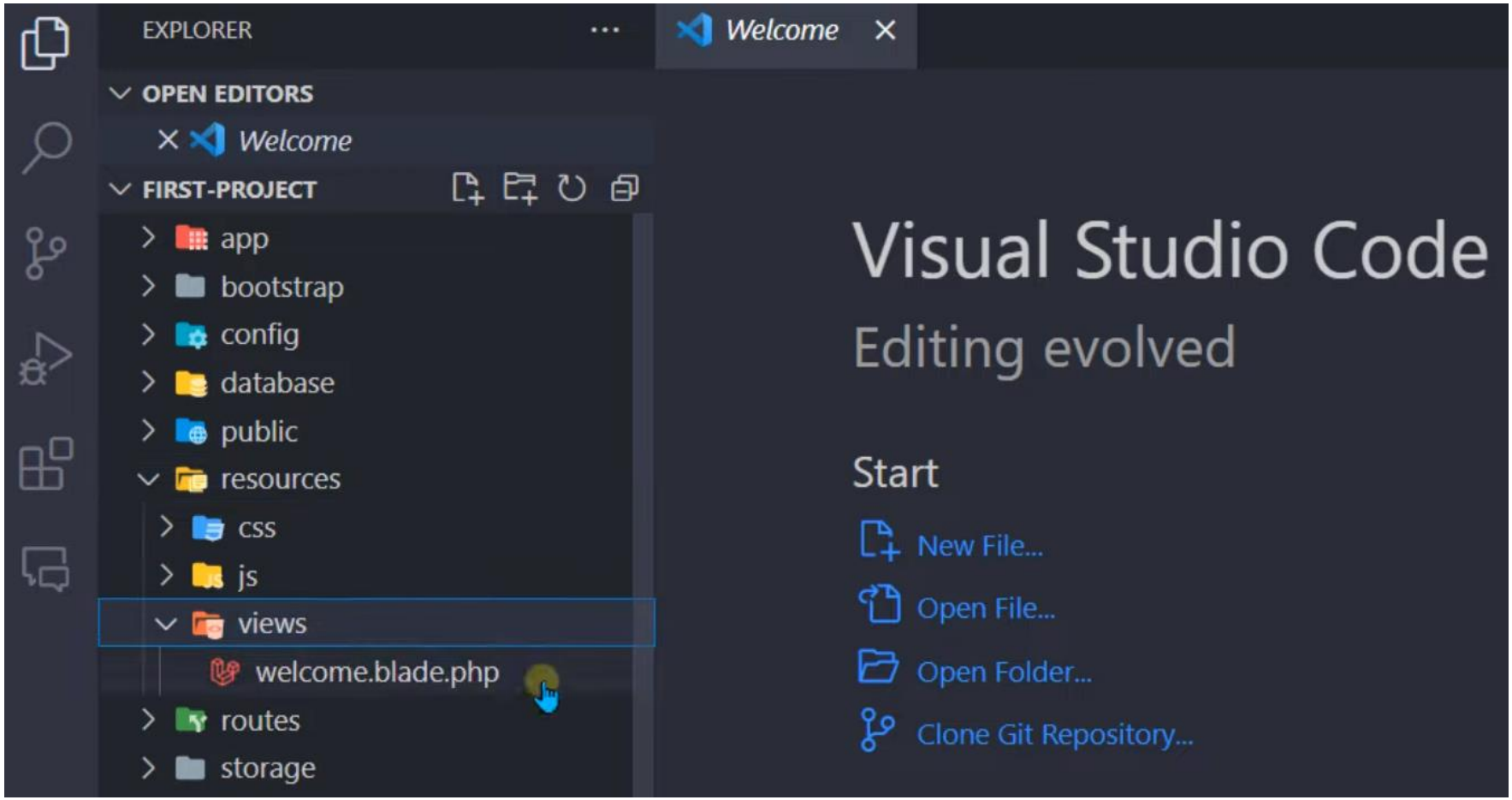
Laravel Folder & File Structure

- Model



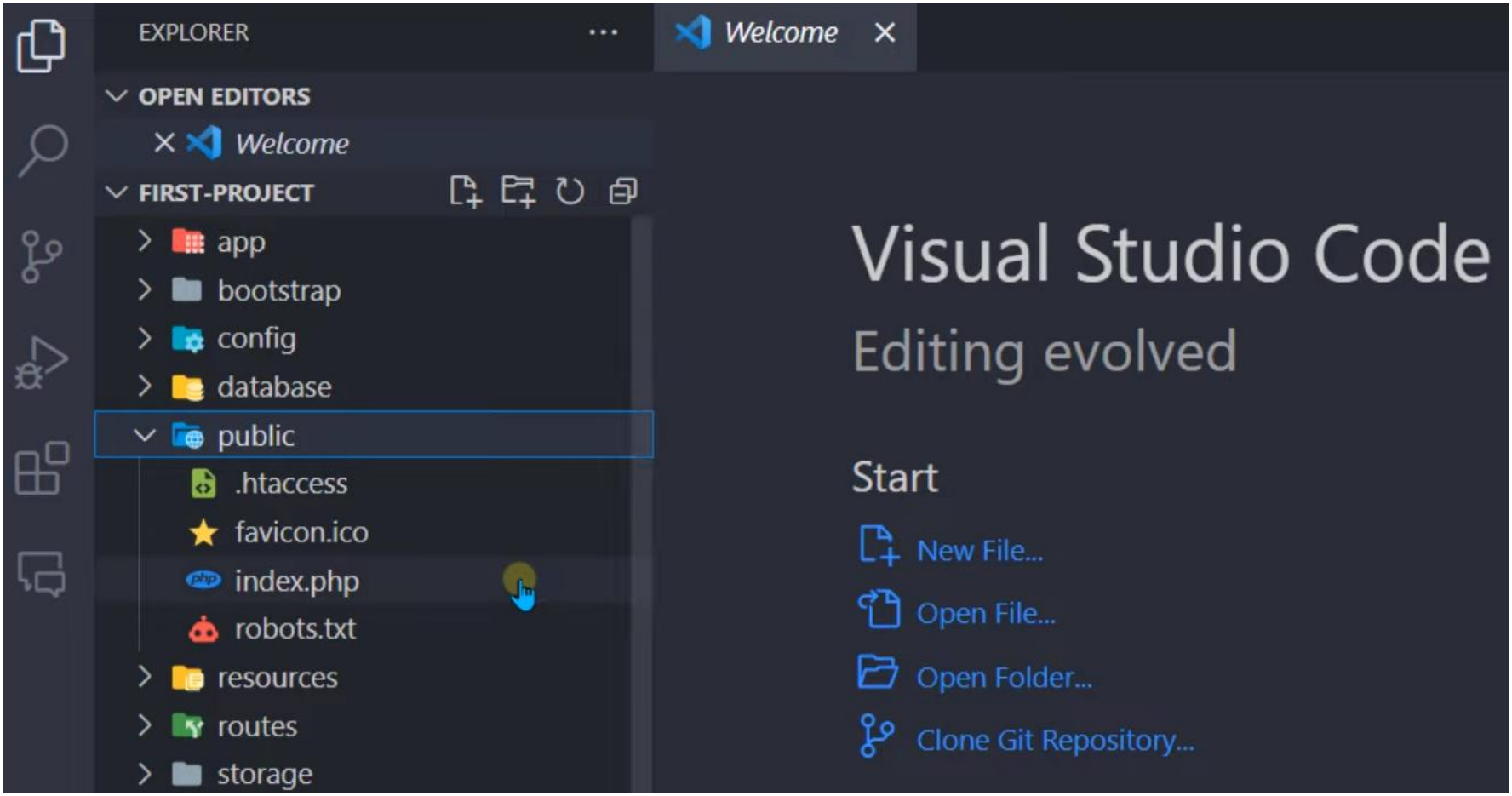
Laravel Folder & File Structure

- Views



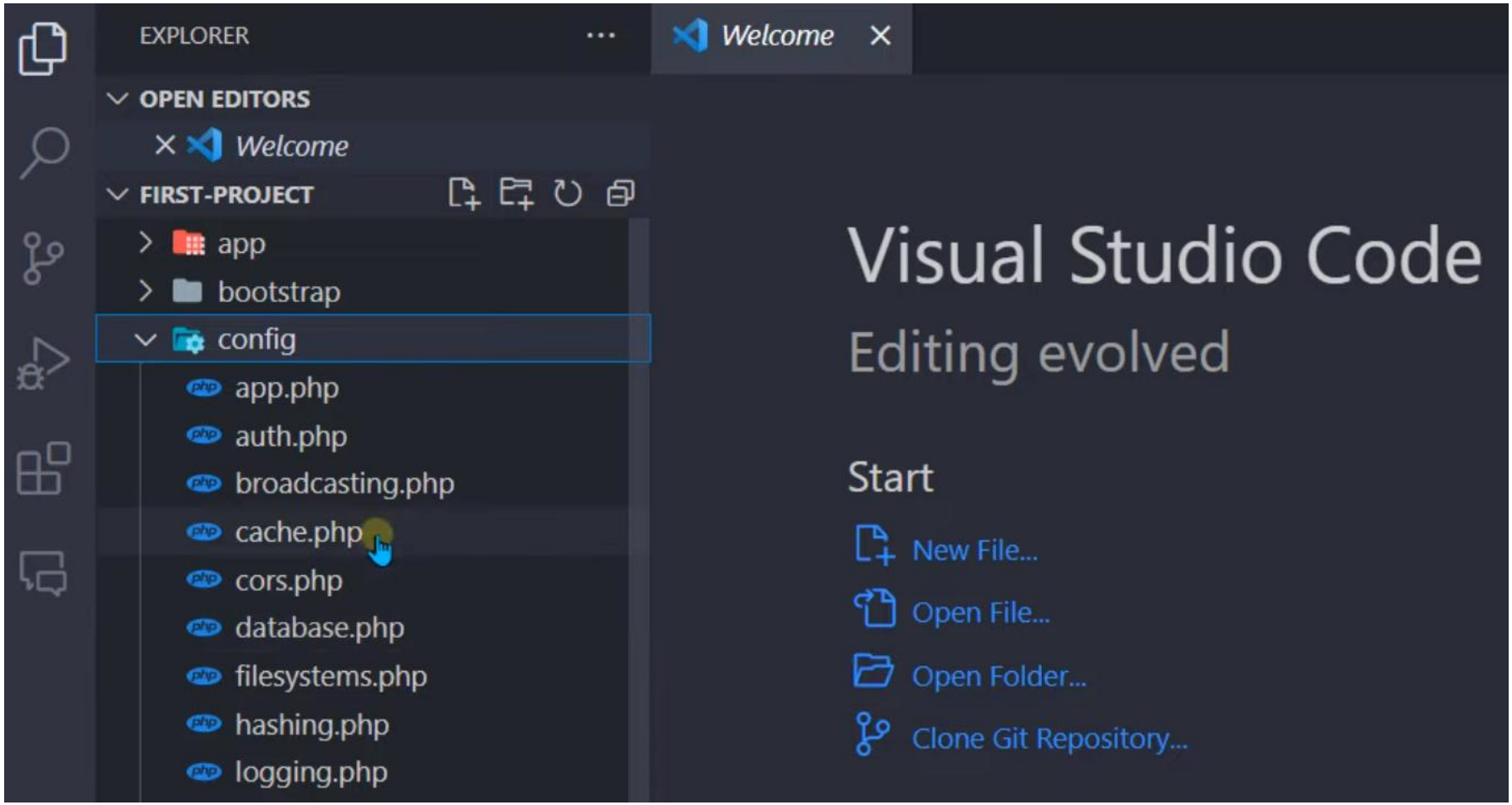
Laravel Folder & File Structure

- Public



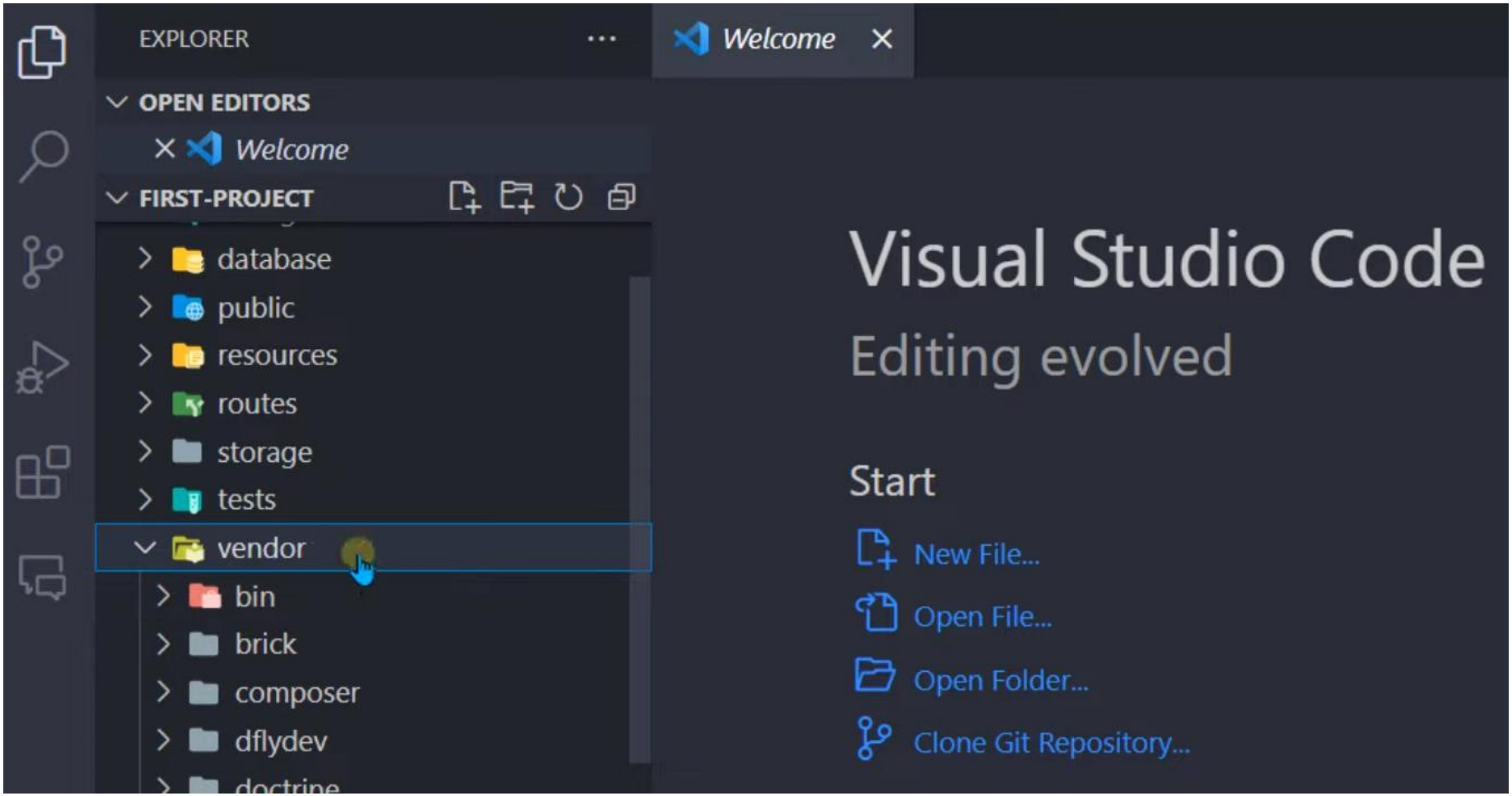
Laravel Folder & File Structure

- config



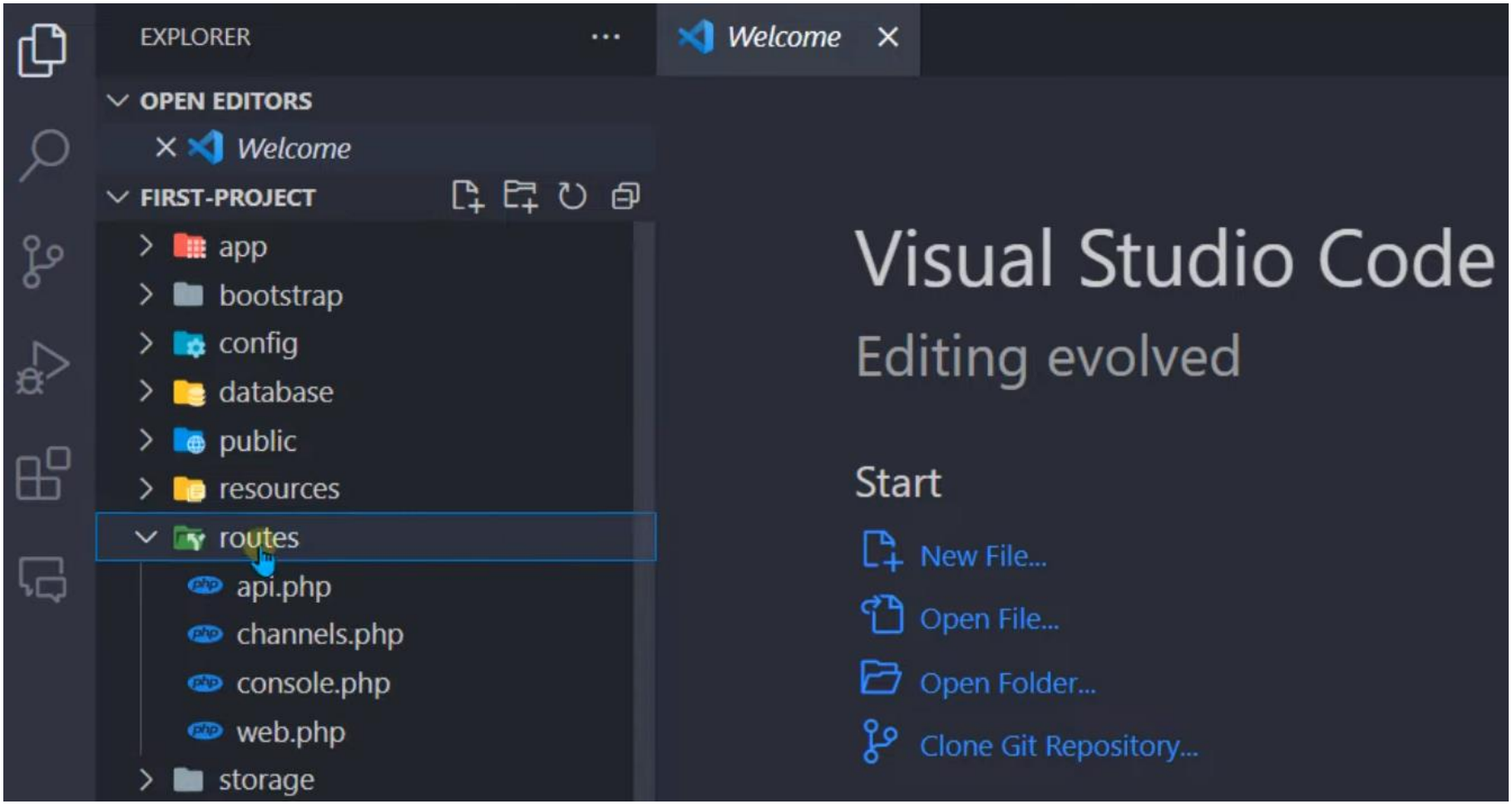
Laravel Folder & File Structure

- vendor

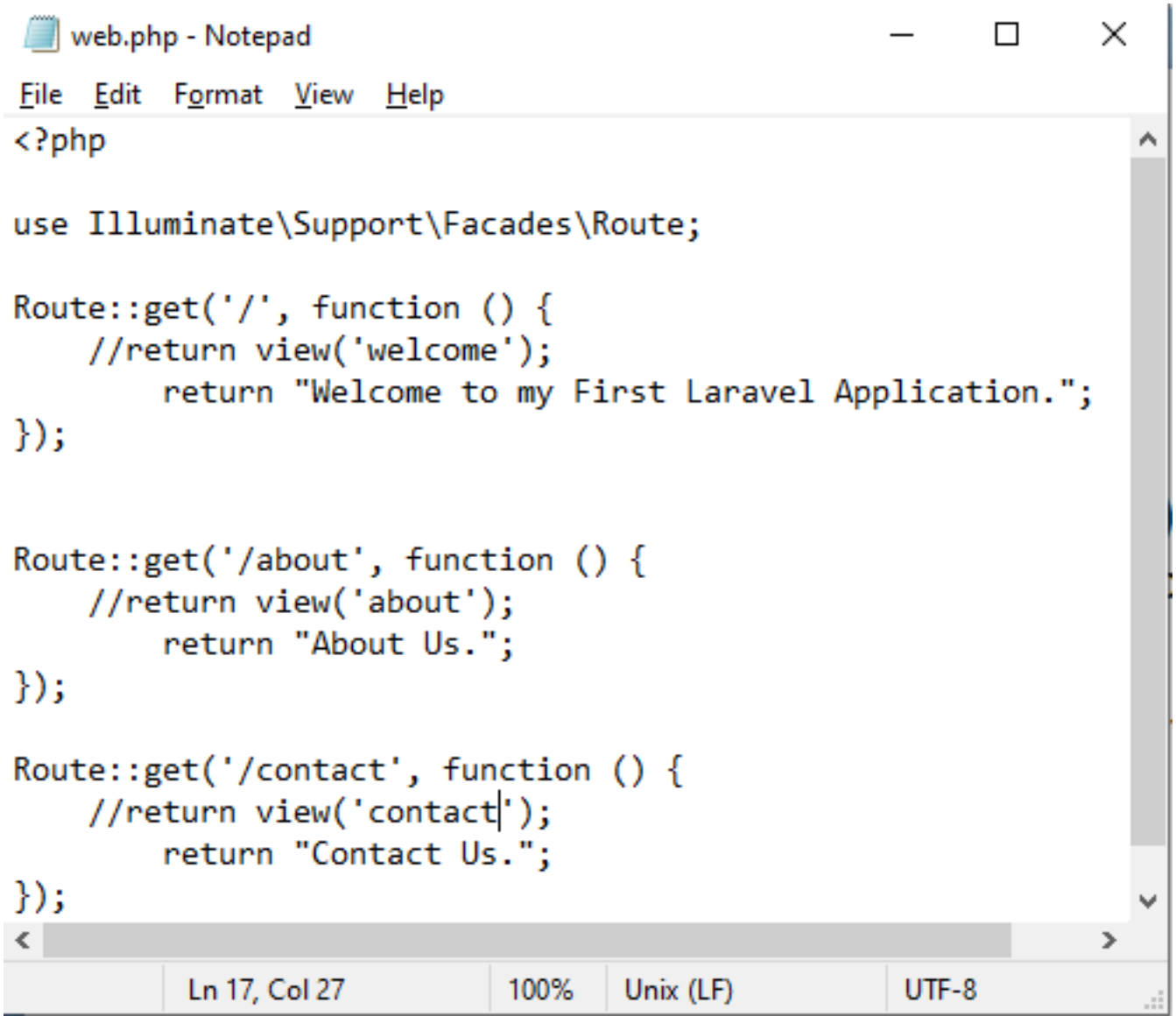


Laravel Folder & File Structure

- routes



Routing


web.php - Notepad
File Edit Format View Help
<?php
use Illuminate\Support\Facades\Route;
Route::get('/', function () {
//return view('welcome');
return "Welcome to my First Laravel Application.";
});
Route::get('/about', function () {
//return view('about');
return "About Us.";
});
Route::get('/contact', function () {
//return view('contact');
return "Contact Us.";
});
Ln 17, Col 27
100%
Unix (LF)
UTF-8

Routing

Basic GET Route

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

Basic POST Route

```
Route::post('foo/bar', function()
{
    return 'Hello World';
});
```

Routing

Registering A Route For Multiple Verbs

```
Route::match(array('GET', 'POST'), '/', function()
{
    return 'Hello World';
});
```

Registering A Route Responding To Any HTTP Verb

```
Route::any('foo', function()
{
    return 'Hello World';
});
```

Routing

Forcing A Route To Be Served Over HTTPS

```
Route::get('foo', array('https', function()
{
    return 'Must be over HTTPS';
}));
```

Often, you will need to generate URLs to your routes, you may do so using the `URL::to` method:

```
$url = URL::to('foo');
```

Route Parameters

Route Parameters

```php
Route::get('user/{id}', function($id)
{
    return 'User '.$id;
});
```

Optional Route Parameters

```php
Route::get('user/{name?}', function($name = null)
{
    return $name;
});
```

Route Parameters

```php
Route::get('/', function () {
    return view('welcome');
});

Route::get('/post/{id?}', function (string $id = null) {
    if($id){
        return "<h1>Post ID : ". $id ."</h1>";
    }else{
        return "<h1>No ID Found</h1>";
    }
```

Route Parameters

Optional Route Parameters With Defaults

```
Route::get('user/{name?}', function($name = 'John')
{
    return $name;
});
```

Regular Expression Route Constraints

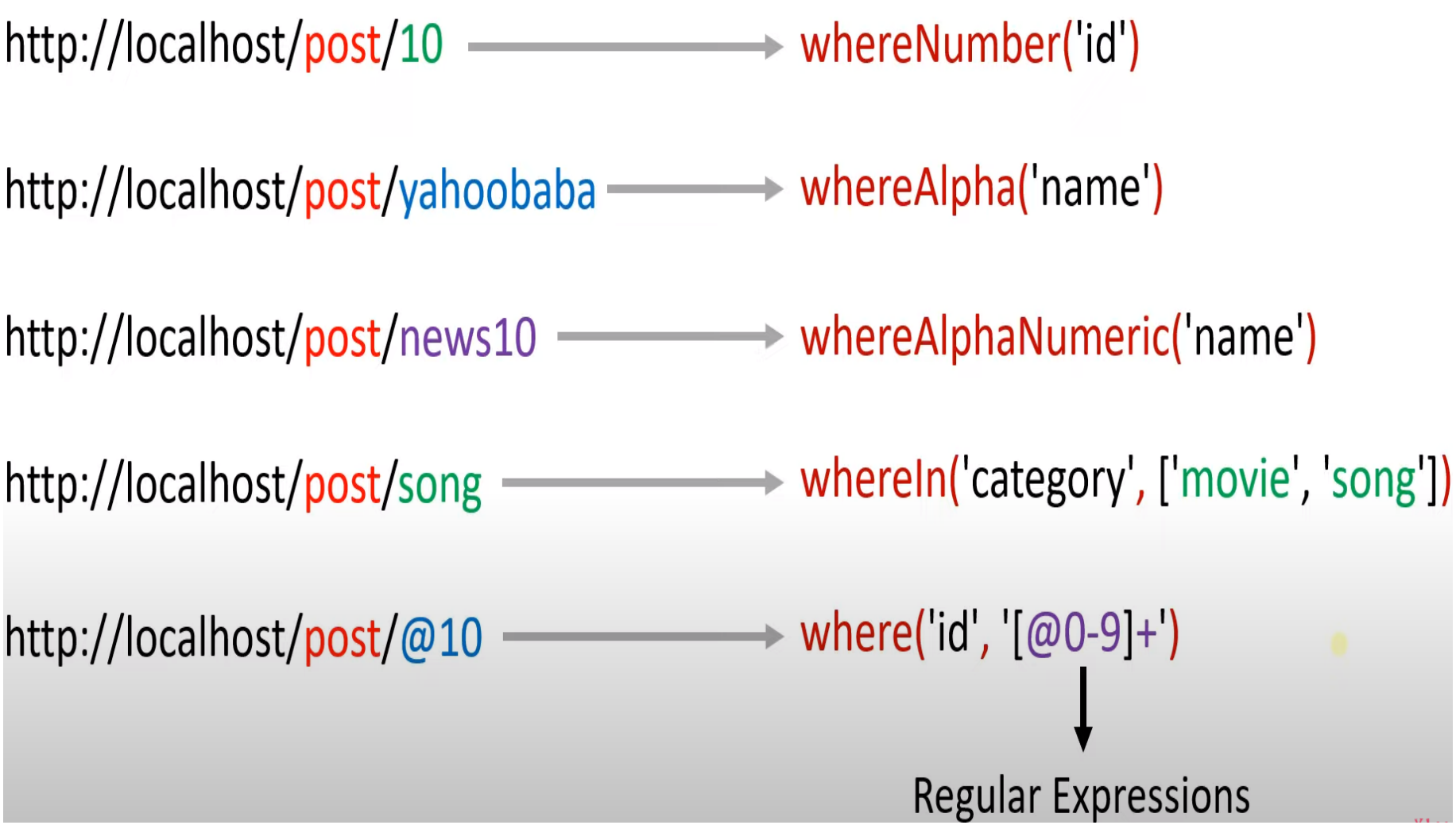
```
Route::get('user/{name}', function($name)
{
    //
})
->where('name', '[A-Za-z]+');

Route::get('user/{id}', function($id)
{
    //
})
->where('id', '[0-9]+');
```

Route Parameters

```php
Route::get('/post/{id?}/comment/{commentid?}', function (string $id = null, string $comment = null) {
    if($id){
        return "<h1>Post ID : ". $id ."</h1><h2>". $comment ."</h2>";
    }else{
        return "<h1>No ID Found</h1>";
    }
```

Route Parameter Constraints



Route Parameter Constraints

Passing An Array Of Wheres

Of course, you may pass an array of constraints when necessary:

```
Route::get('user/{id}/{name}', function($id, $name)
{
    //
})
->where(array('id' => '[0-9]+', 'name' => '[a-z]+'))
```

Defining Global Patterns

If you would like a route parameter to always be constrained by a given regular expression, you may use the `pattern` method:

```
Route::pattern('id', '[0-9]+');

Route::get('user/{id}', function($id)
{
    // Only called if {id} is numeric.
});
```

Route Parameter Constraints

```php
Route::get('/post/{id}/comment/{commentid}', function (string $id, string $comment) {
    if($id){
        return "<h1>Post ID : ". $id ."& Comment : ". $comment ."</h1>";
    }else{
        return "<h1>No ID Found</h1>";
    }
})->where('id', '[0-9]+')->whereAlpha('commentid');
```

Named Route

http://localhost/page/about

```
Route::get('/page/about-us', function () {
    return 'About Page';
})->name('about');
```

first.blade.php

```
<a href='{{ route('about') }}'>About</a>
```

second.blade.php

```
<a href='{{ route('about') }}'>About</a>
```

third.blade.php

```
<a href='{{ route('about') }}'>About</a>
```

Named Route

```
<h1>About Page</h1>

<a href="/">Home</a>
<a href="{{ route('mypost') }}">Post</a>
```

Route Redirection

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

Route::get('page/posts', function () {
    return view('post');
})->name('mypost');

Route::get('/test', function () {
    return view('about');
});

Route::redirect('/about','/test');
```

Route Groups

```
Route::prefix('page')->group(function () {
    Route::get('/post/1', function () {
    });
    Route::get('/about/', function () {
    });
    Route::get('/gallery/', function () {
    });
});
```

http://localhost/page/about

Route Groups

```php
Route::prefix('page')->group(function(){
    Route::get('/about', function () {
        return "<h1>About Page</h1>";
    });

    Route::get('/gallery', function () {
        return "<h1>Gallery Page</h1>";
    });

    Route::get('/post/firstpost', function ()
        return "<h1>First Post Page</h1>";
    });
});
```

Default Route

- Fallback method is used to redirect 404 error page to default route.

```
Route::fallback(function(){
    return "<h1>Page Note Found.</h1>";
});
```

HTTP Middleware Introduction

- Middleware acts as a layer between the user and the request.
- It means that when the user requests the server then the request will pass through the middleware, and then the middleware verifies whether the request is authenticated or not.
- If the user's request is authenticated then the request is sent to the backend.
- If the user request is not authenticated, then the middleware will redirect the user to the login screen.
- An additional middleware can be used to perform a variety of tasks except for authentication.
- Laravel middlewares are located in the **app/Http/Middleware**.
- We can say that middleware is an http request filter where you can check the conditions.

Creating Middleware

- Type the command **php artisan make:middleware 'name of the middleware‘**
- To see whether the middleware is created or not, go to your project.
- Our project name is laravelproject, so the path for the middleware would be: **C:\xampp\htdocs\laravelproject\app\Http\Middleware**

Applying Middleware

- Open the kernel.php file.
- If we want to apply the middleware to all the URLs, then add the path of the middleware in the array of middleware.
- Open the **php** file for the middleware, which you have created as a middleware and print a message using echo in handle function.

Applying Middleware to Specific Routes

- Open the kernel.php file.
- add the code, i.e.,
- **''age' => \App\Http\Middleware\CheckAge::class'**, where age is the name of the middleware.
- Now, we can use the 'age' middleware for some specific routes.
- Add the middleware code in the **web.php** file.

Applying Middleware to Specific Routes

```
Route::Get('/',function()
{
  return view('welcome');
})-> middleware('age');
Route::Get('user/profile',function()
{
  return "user profile";
});
```

- In the above code, we have added middleware in '/' root URL, and we have not added the middleware in the 'user/profile' URL.

Middleware Parameters

- In web.php file

```
Route::Get('/{age}',function($age)
{
  return view('welcome');
})-> middleware('age');
```

Middleware Parameters

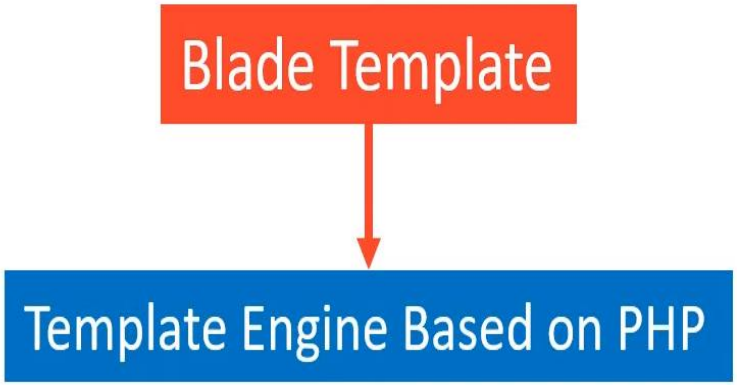
- In middleware file

```
   public function handle($request, Closure $next)

{
  //return "middleware";
if($request->age>10)
{
echo "Age is greater than 10";
}
else
{
echo"Age is not greater than 10";
}
return $next($request);

}}
```

Blade Template Introduction



Blade provides a clean and convenient way to create views in Laravel

Benefits : Create Dynamic and Reusable Templates

HTML & PHP

Blade Template Syntax

```php
<?php
    echo "Hello";
?>
```

```
{{ "Hello" }}
```

Prevent cross-site scripting (XSS) Attacks

```php
<?php
    echo $name;
?>
```

```
{{ $name }}
```

```php
<?php
    echo "<h1>Hello</h1>";
?>
```

```
{!! "<h1>Hello</h1>" !!}
```

Blade Template Syntax

```
<?php

?>
```

```
@php

@endphp
```

```
<?php
    // Comment
?>
```

```
{-- Comment --}
```

Blade Control Structure Syntax

```php
<?php
    if(condition){
        //Statement
    }elseif(condition){
        //Statement
    }else{
        //Statement
    }
?>
```

```
@if (condition)
    //Statement
@elseif (condition)
    //Statement
@else
    //Statement
@endif
```

Blade Control Structure Syntax

```php
<?php
    switch($i){
        case 1:
            First case...
        break;
        case 2:
            First case...
        break;
        case 3:
            First case...
        break;
    }
?>
```

```
@switch($i)
    @case(1)
            First case...
        @break

    @case(2)
            Second case...
        @break

    @default
            Default case...
@endswitch
```

Blade Control Structure Syntax

```php
<?php
    if(isset($records)){
        //Statement
    }
?>
```

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

```php
<?php
    if(emtpy($records)){
        //Statement
    }
?>
```

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

Blade Loop statement syntax

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

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

```
@while (condition)
    <p>Loop Statement</p>
@endwhile
```

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

```
@continue
```

```
@break
```

Blade Loop variable for @foreach

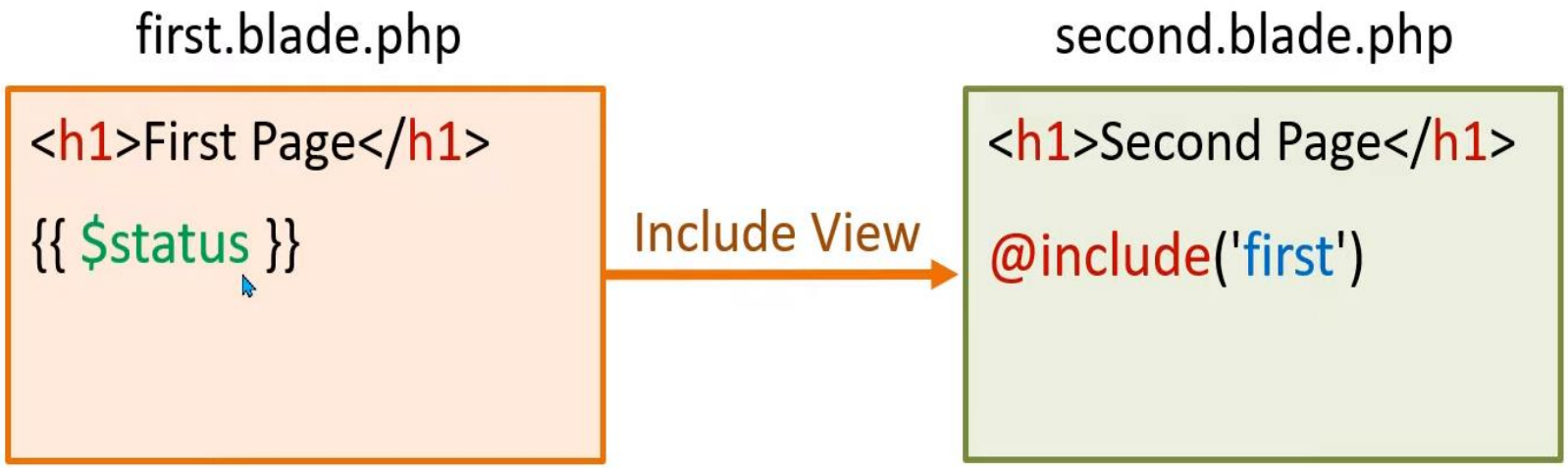
Property	Description
$loop->index	The index of the current loop iteration (starts at 0).
$loop->iteration	The current loop iteration (starts at 1).
$loop->remaining	The iterations remaining in the loop.
$loop->count	The total number of items in the array being iterated.
$loop->first	Whether this is the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

Blade template directives

- @include
- @section
- @extend
- @yield

Reusable Templates

Blade @include directives

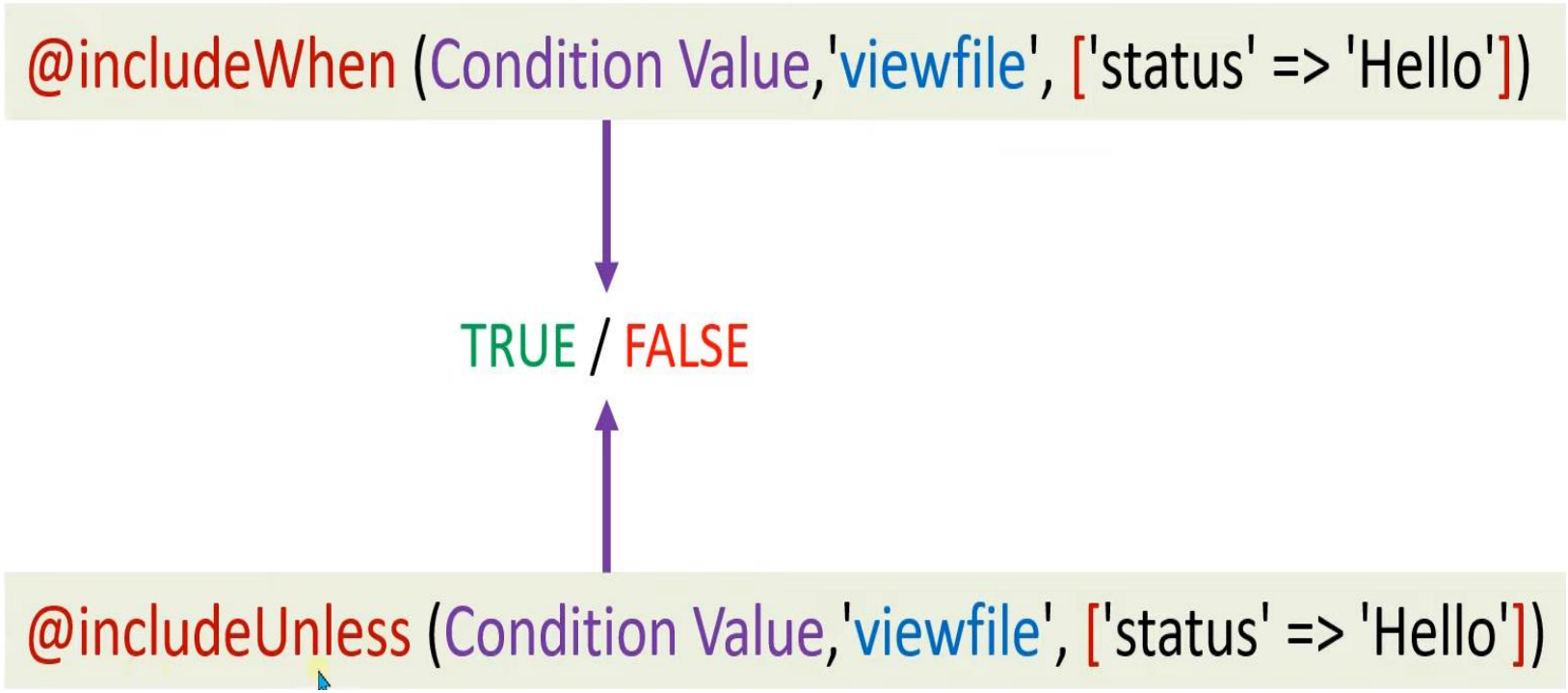


@include()

```
@include('first', ['status' => 'Hello'])
```

Blade @include directives

Blade @include directives



Blade @include directives

```
@php
  $fruits = ["one" => "Apple","two" => "Banana","three" =>"Orange"];
  $boolean = true;
@endphp

@includeWhen($boolean,'pages.header',['names' => $fruits])

<h1>Home Page</h1>

@include('pages.footer')

@includeIf('pages.content')
```

Blade template Inheritance

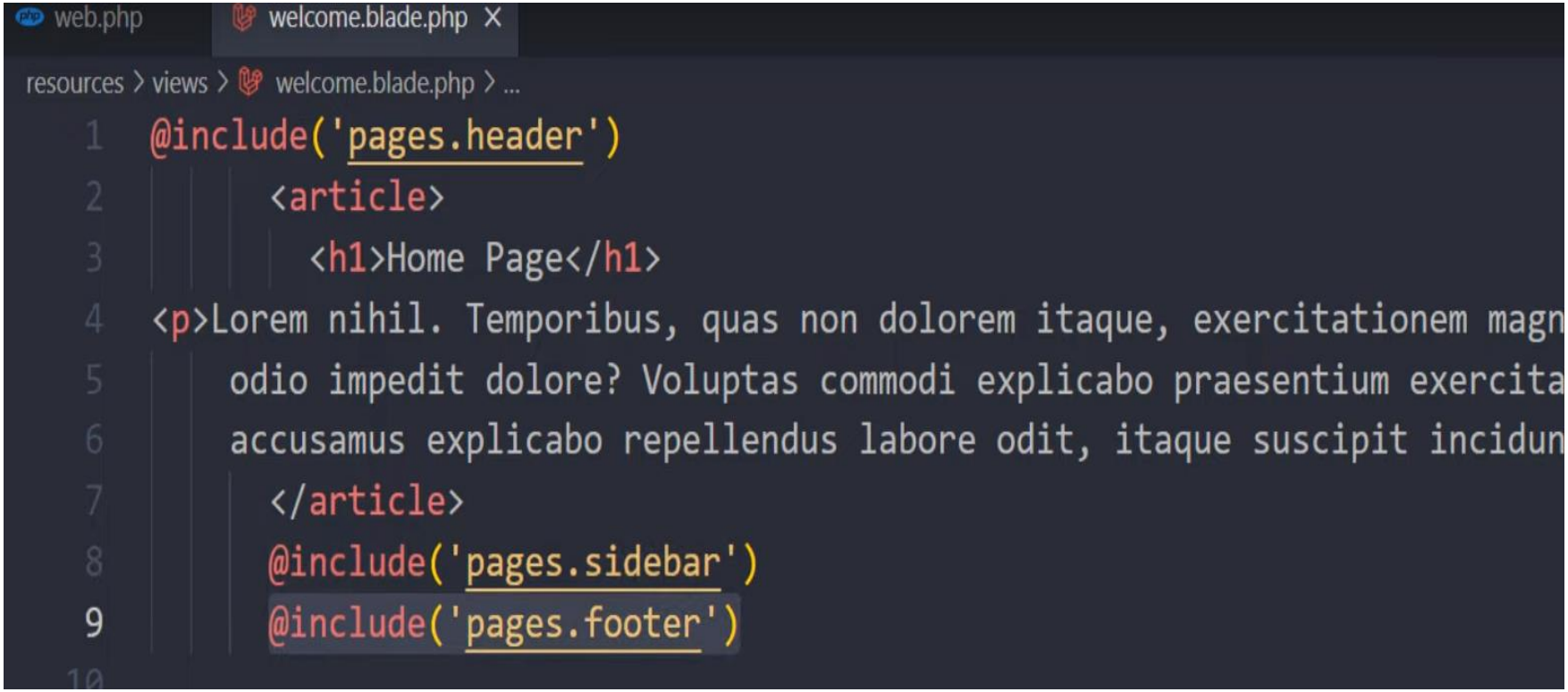
layout.blade.php

home.blade.php

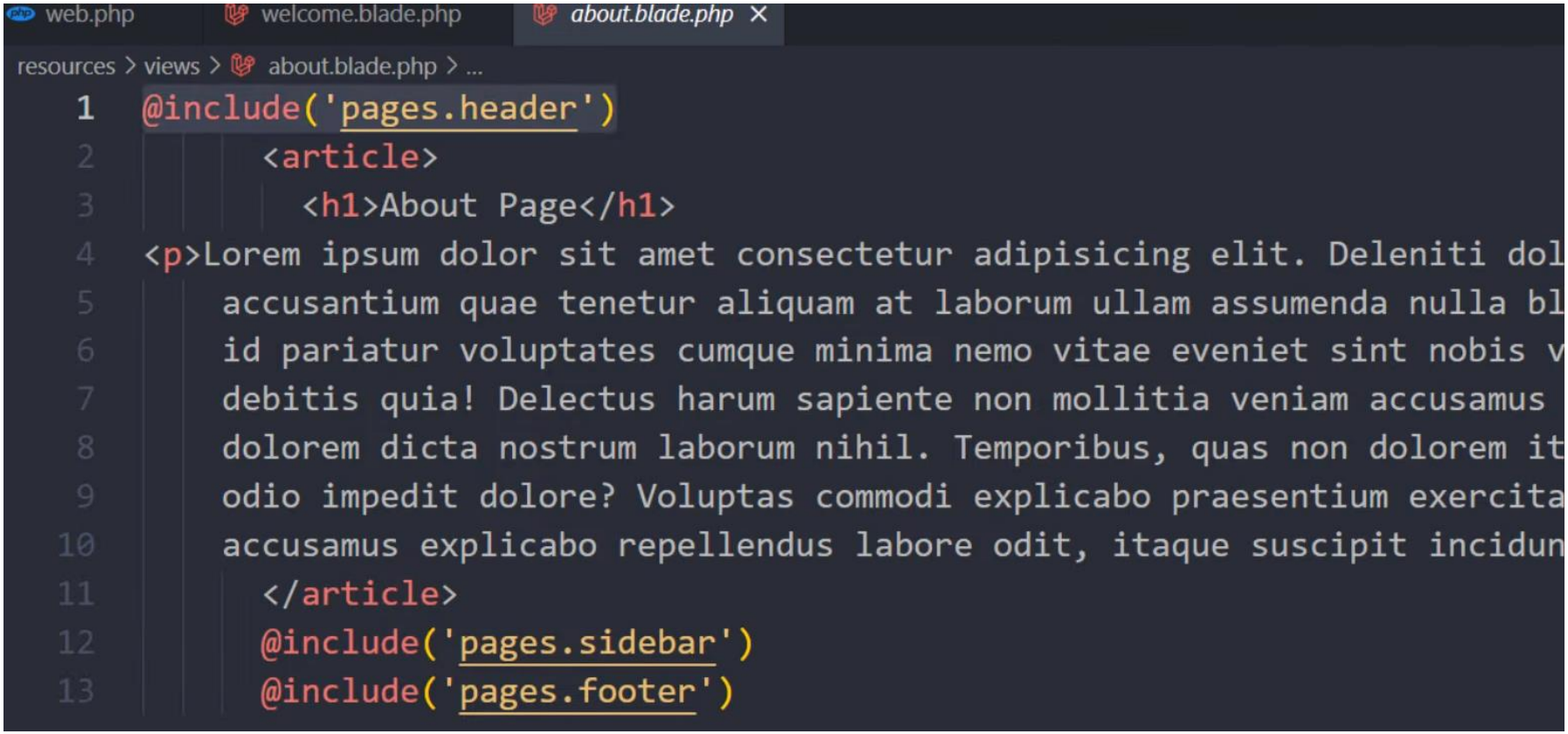
```
@extends('layout')

@section('content')
    <h1>Home Page</h1>
@endsection
```

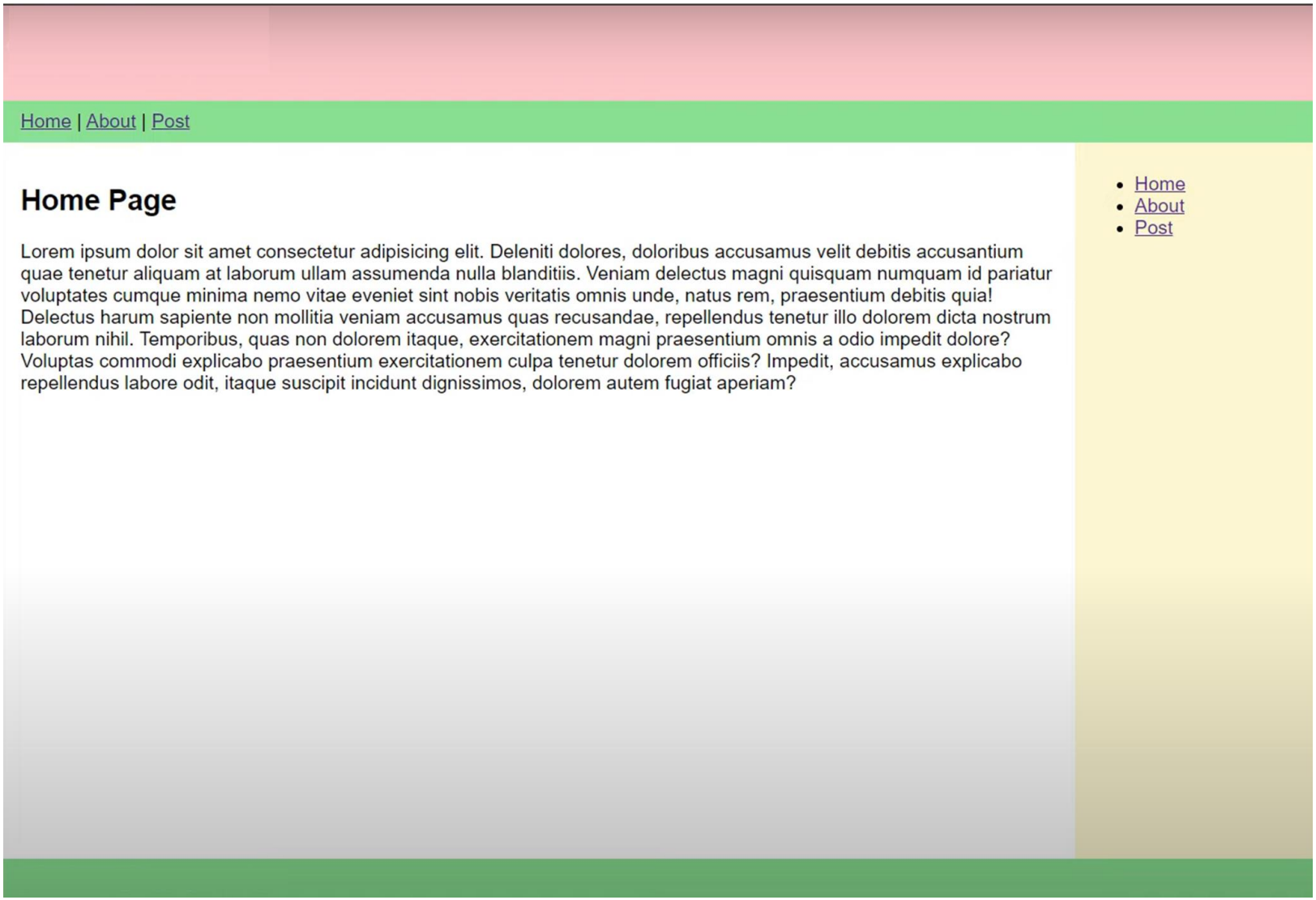
Blade template Inheritance



Blade template Inheritance



Blade template Inheritance

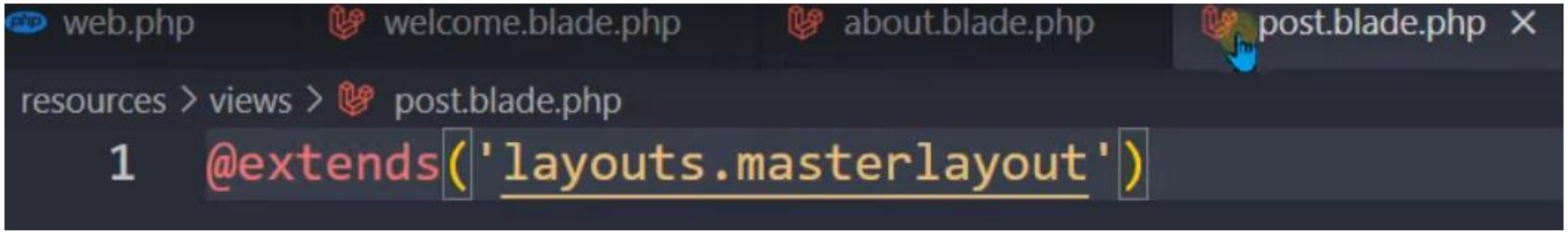


Blade template Inheritance

- Masterlayout.blade.php

```
<main>
  <article>
    @yield('content')
  </article>
```

- Post.blade.php/welcome.blade.php



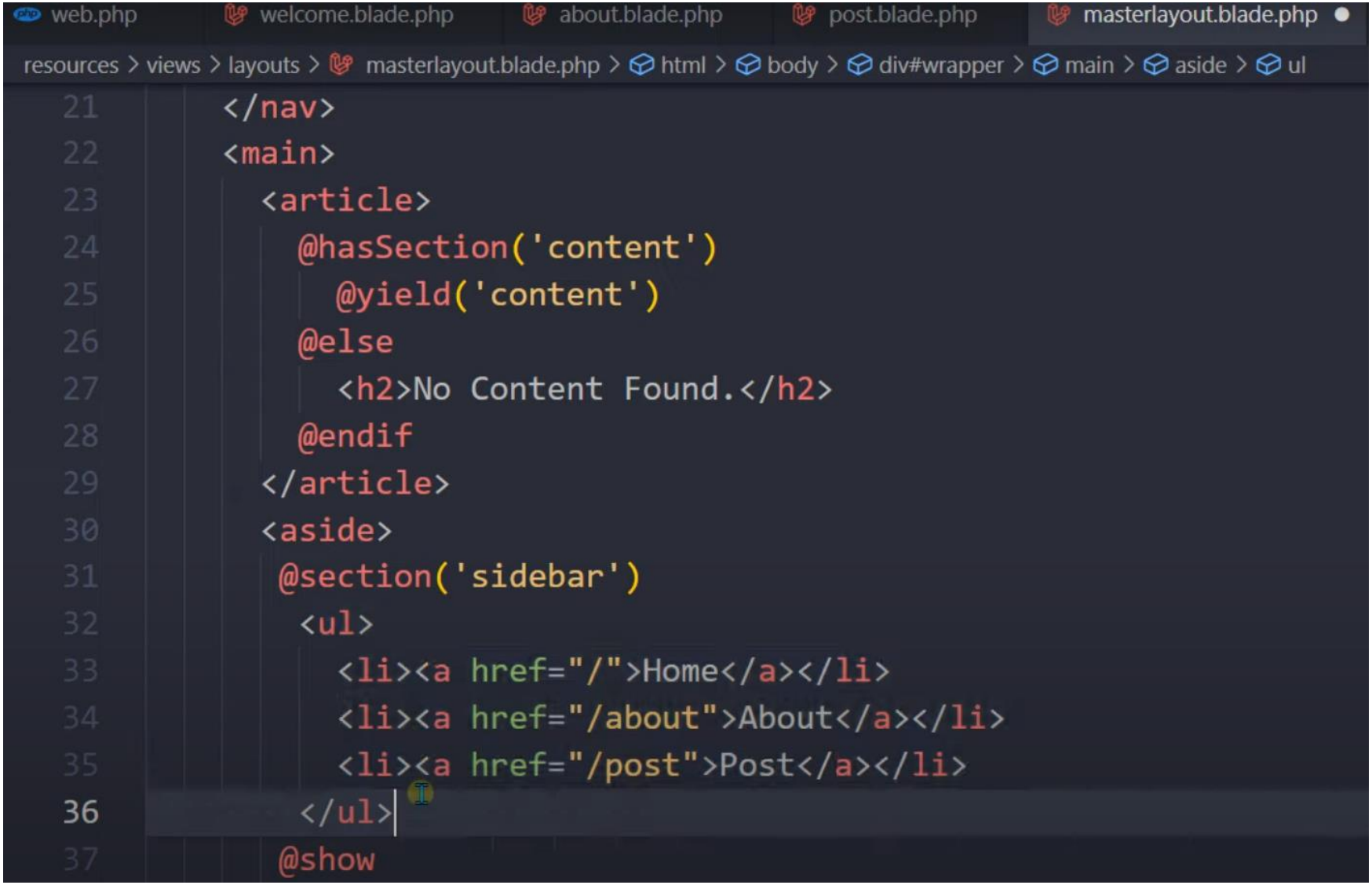
Blade template Inheritance

Blade template Inheritance

- @yield(‘identifier’, ‘default_identifier’)

Blade template Inheritance

Blade template Inheritance



Blade template Inheritance

web.php | welcome.blade.php | about.blade.php | post.blade.php ●

resources > views > post.blade.php > ...

```
1  @extends('layouts.masterlayout')
2
3  @section('content')
4      <h2>Post Page</h2>
5      <p>Lorem ipsum dolor sit amet consectetur, adip
6
7  @endsection
8
9  @section('sidebar')
10     @parent
11     <p>This is appended to the master sidebar.</p>
12 @endsection
```