

# 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

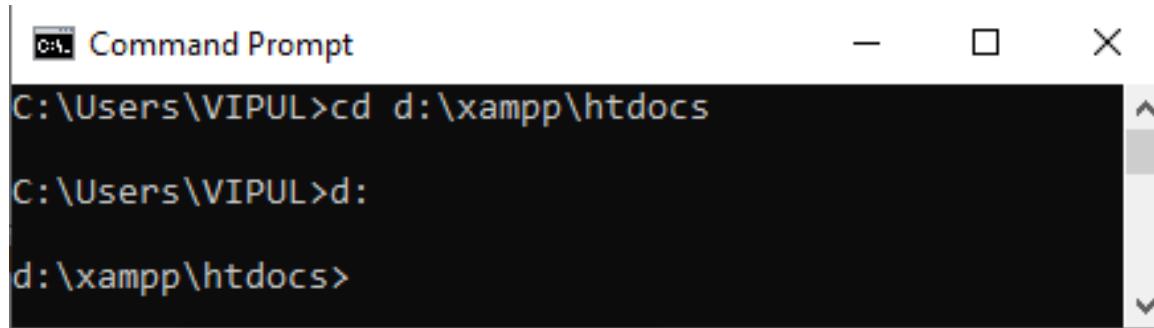
# Install Composer on Windows

- 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.



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:  
C:\Users\VIPUL>cd d:\xampp\htdocs  
C:\Users\VIPUL>d:  
d:\xampp\htdocs>

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

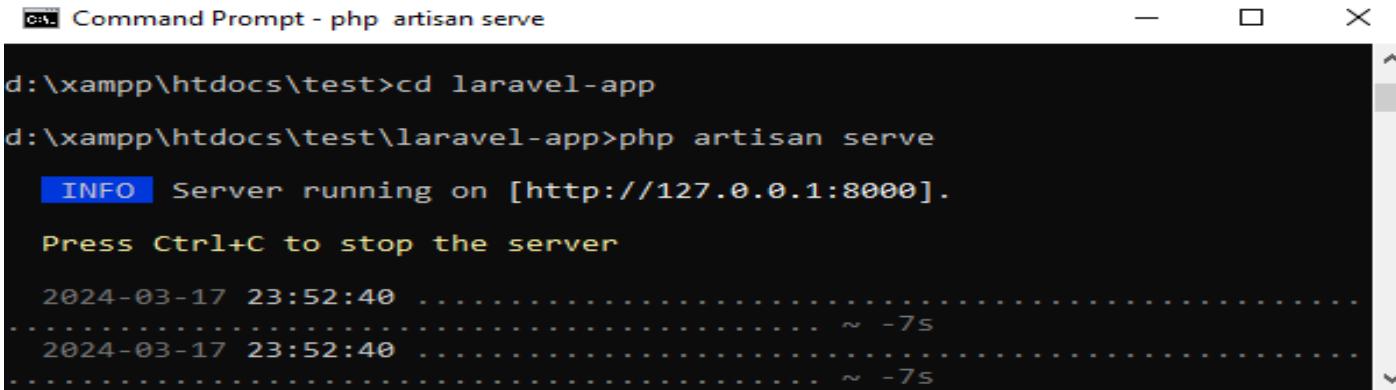
```
Command Prompt

C:\Users\VIPUL>cd d:\xampp\htdocs
C:\Users\VIPUL>d:
d:\xampp\htdocs>cd test
d:\xampp\htdocs\test>composer create-project laravel/laravel laravel-app
Creating a "laravel/laravel" project at "./laravel-app"
Installing laravel/laravel (v11.0.3)
- Downloading laravel/laravel (v11.0.3)
- Installing laravel/laravel (v11.0.3): Extracting archive
Created project in D:\xampp\htdocs\test\laravel-app
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 111 installs, 0 updates, 0 removals
- Locking brick/math (0.11.0)
- Locking carbonphp/carbon-doctrine-types (3.2.0)
- Locking dflydev/dot-access-data (v3.0.2)
- Locking doctrine/inflector (2.0.10)
- Locking doctrine/lexer (3.0.1)
- Locking dragonmantank/cron-expression (v3.3.3)
- Locking egulias/email-validator (4.0.2)
- Locking fakerphp/faker (v1.23.1)
- Locking filip/whoops (2.15.4)
- Locking fruitcake/php-cors (v1.3.0)
- Locking graham-campbell/result-type (v1.1.2)
- Locking guzzlehttp/guzzle (7.8.1)
- Locking guzzlehttp/promises (2.0.2)
- Locking guzzlehttp/psr7 (2.6.2)
- Locking guzzlehttp/uri-template (v1.0.3)
- Locking hamcrest/hamcrest-php (v2.0.1)
- Locking laravel/framework (v11.0.7)
- Locking laravel/pint (v1.14.0)
- Locking laravel/prompts (v0.1.16)
- Locking laravel/sail (v1.29.0)
- Locking laravel/serializable-closure (v1.3.3)
- Locking laravel/tinker (v2.9.0)
- Locking league/commonmark (2.4.2)
- Locking league/config (v1.2.0)
- Locking league/flysystem (3.25.1)
- Locking league/flysystem-local (3.25.1)
- Locking league/mime-type-detection (1.15.0)

Activate Windows
Go to Settings to activate Windows.
```

# Test Laravel Installation on Windows

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



```
Command Prompt - php artisan serve
d:\xampp\htdocs\test>cd laravel-app
d:\xampp\htdocs\test\laravel-app>php artisan serve
    INFO Server running on [http://127.0.0.1:8000].
    Press Ctrl+C to stop the server
2024-03-17 23:52:40 ..... ~ -7s
..... ~ -7s
2024-03-17 23:52:40 .....
```

- 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.

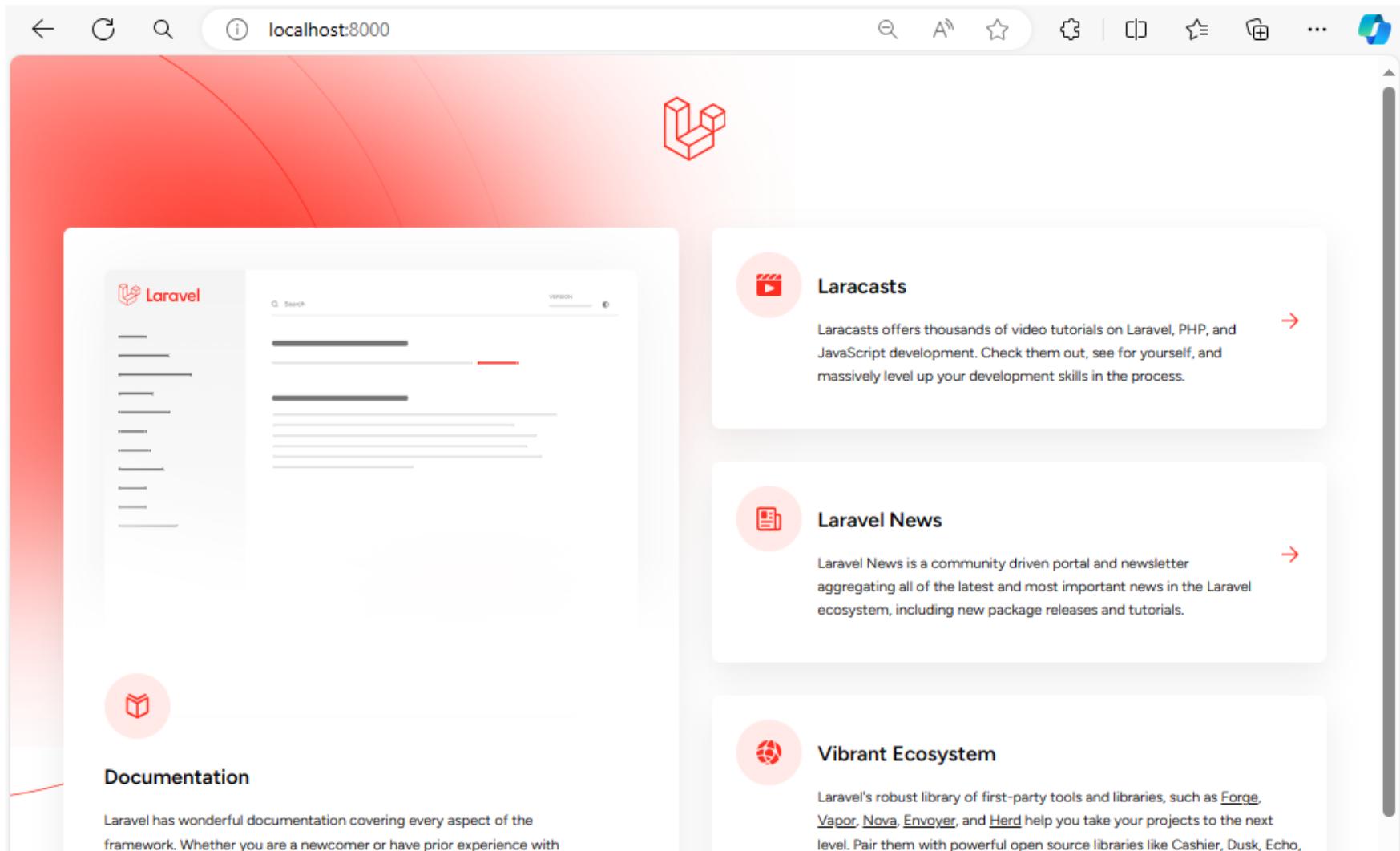
```
C:\Users\vskjh>composer global require laravel/installer
Changed current directory to C:/Users/vskjh/AppData/Roaming/Composer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
```

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

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



# Test Laravel Installation on Windows



The screenshot shows a web browser window with the URL `localhost:8000` in the address bar. The page content is a promotional page for Laravel, featuring the Laravel logo and several sections:

- Laravel**: Shows a screenshot of the Laravel documentation interface with a search bar and a list of articles.
- Documentation**: A section with a red circular icon containing a book icon, describing the comprehensive documentation available.
- Laracasts**: A section with a red circular icon containing a video camera icon, describing Laracasts video tutorials.
- Laravel News**: A section with a red circular icon containing a newspaper icon, describing the Laravel News community portal.
- Vibrant Ecosystem**: A section with a red circular icon containing a globe icon, describing the ecosystem of tools and libraries like Forge, Vapor, Nova, Envoyer, and Herd.

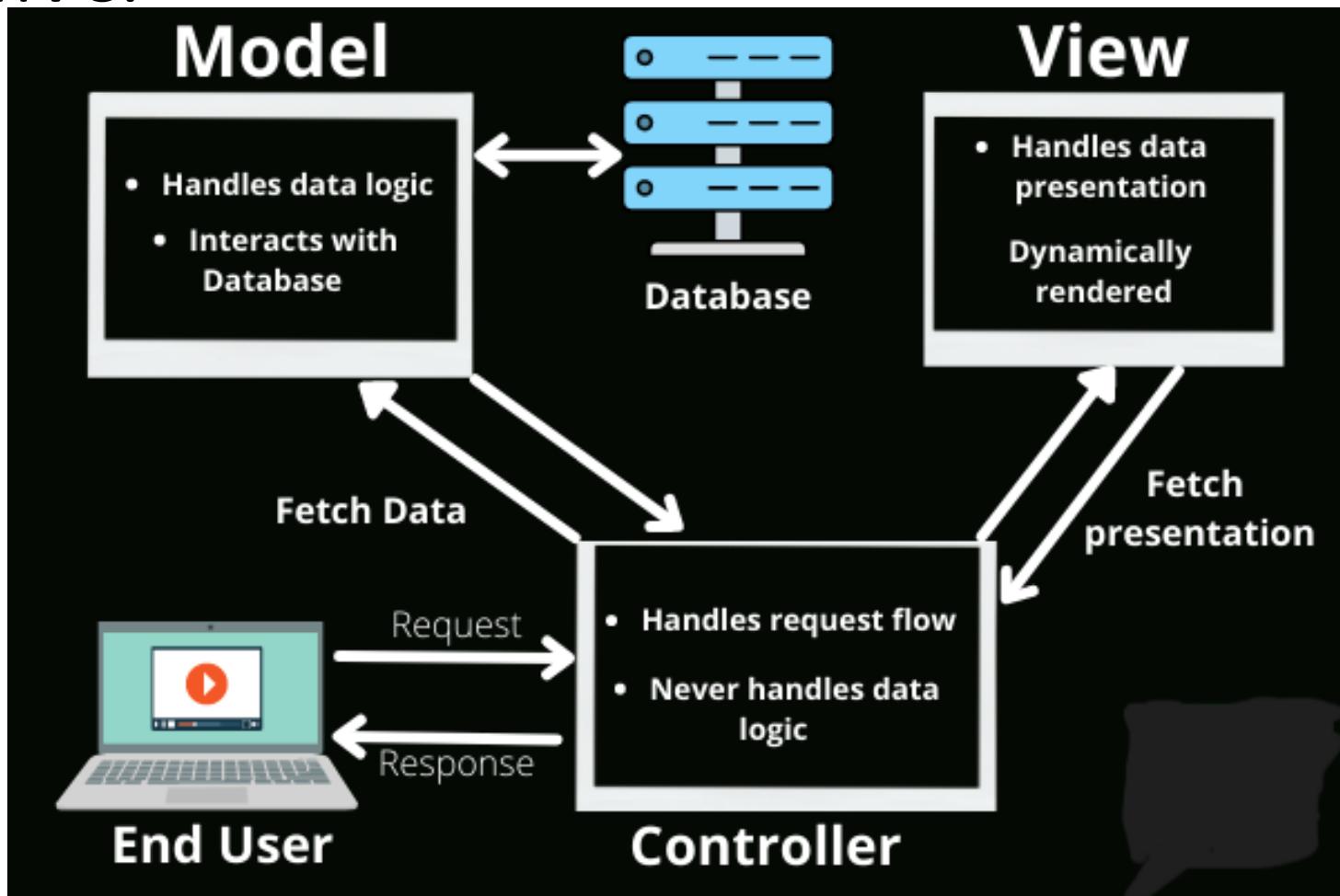
The browser interface includes a back button, forward button, search bar, and various toolbar icons.

# Laravel

- MVC:
  - It is 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:



# 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, UI 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

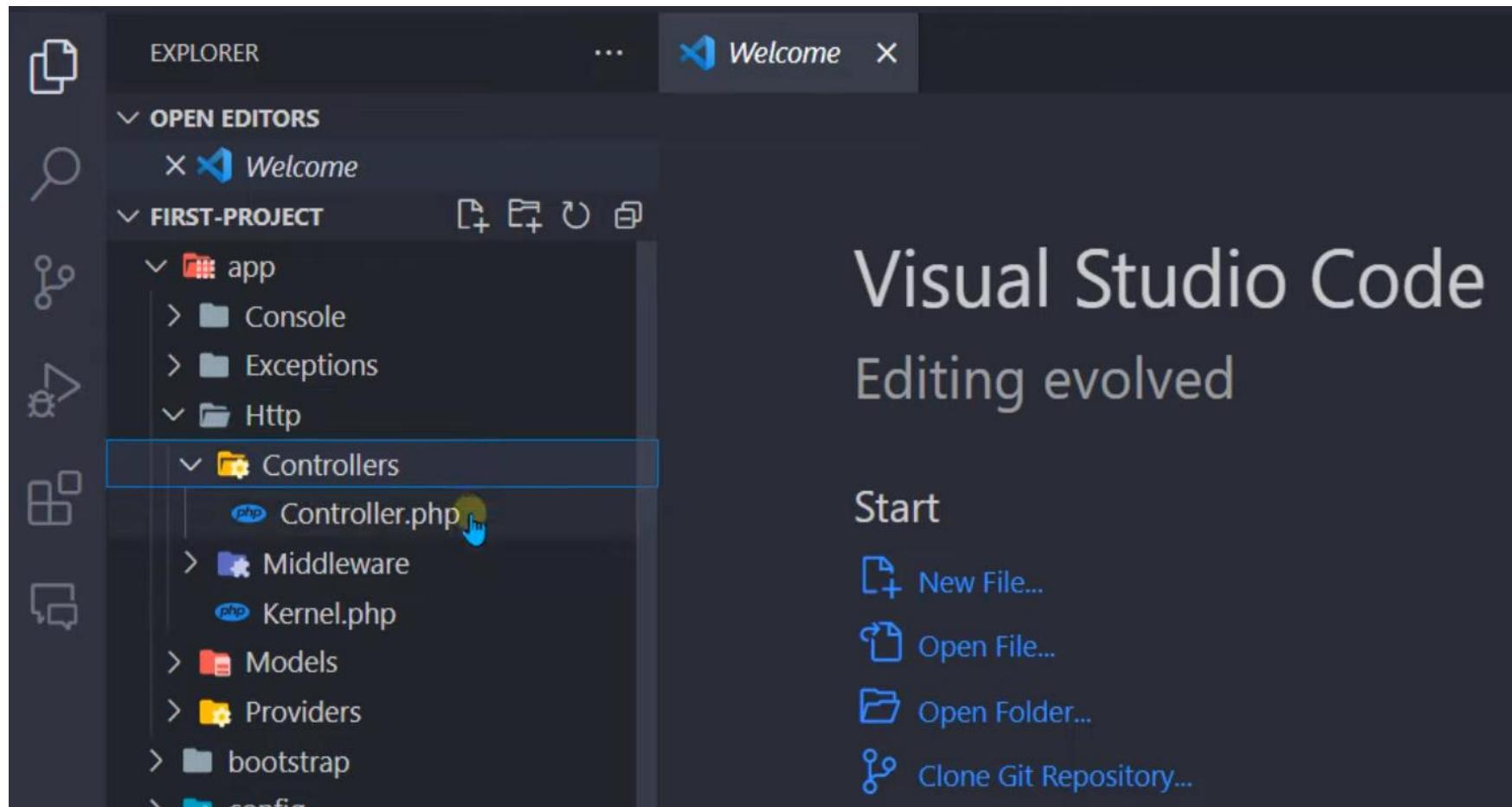
- It is a process of mapping the browser request to the controller action and return response back.
- Each MVC application has default routing.
- We can set custom routing for newly created controller.
- We can go to routes/web.php to modify application's routes.
- In the newer version, Laravel 5.3, 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

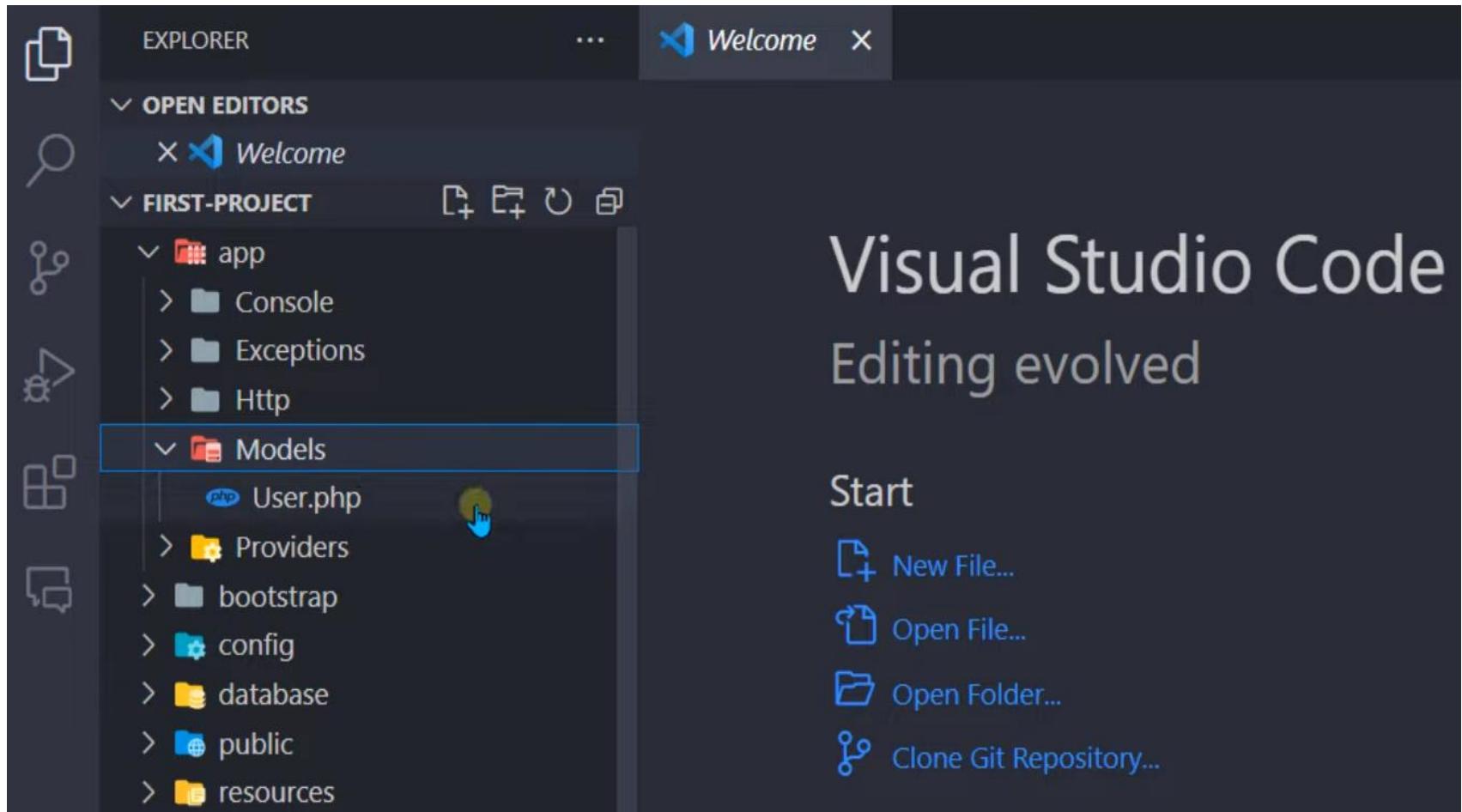
# Laravel Folder & File Structure

- Controller



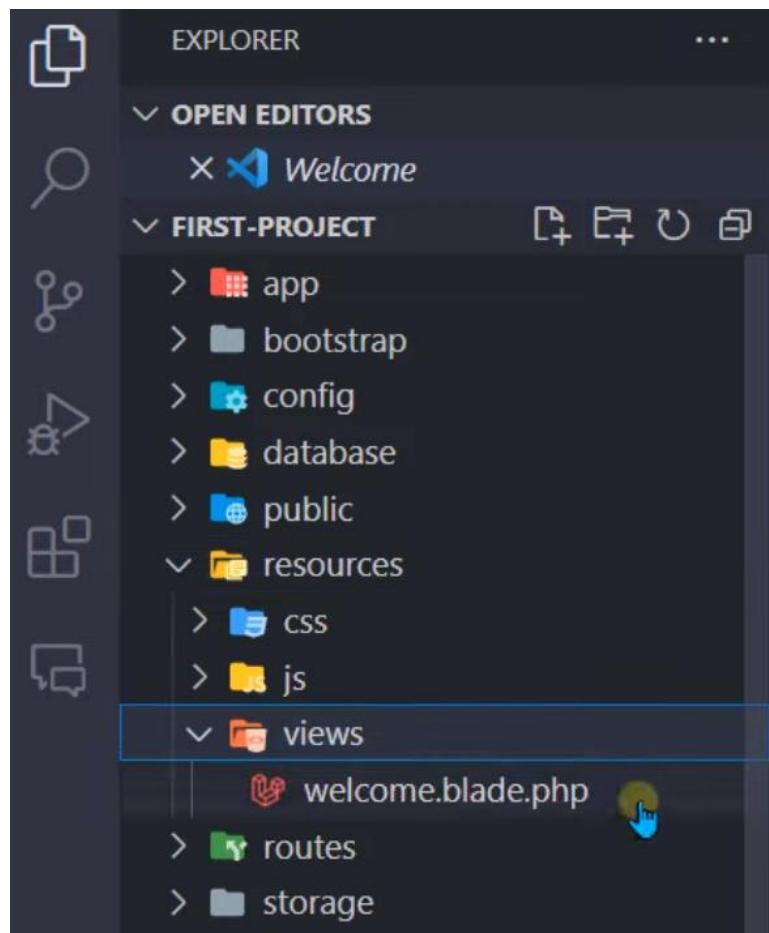
# Laravel Folder & File Structure

- Model



# Laravel Folder & File Structure

- Views



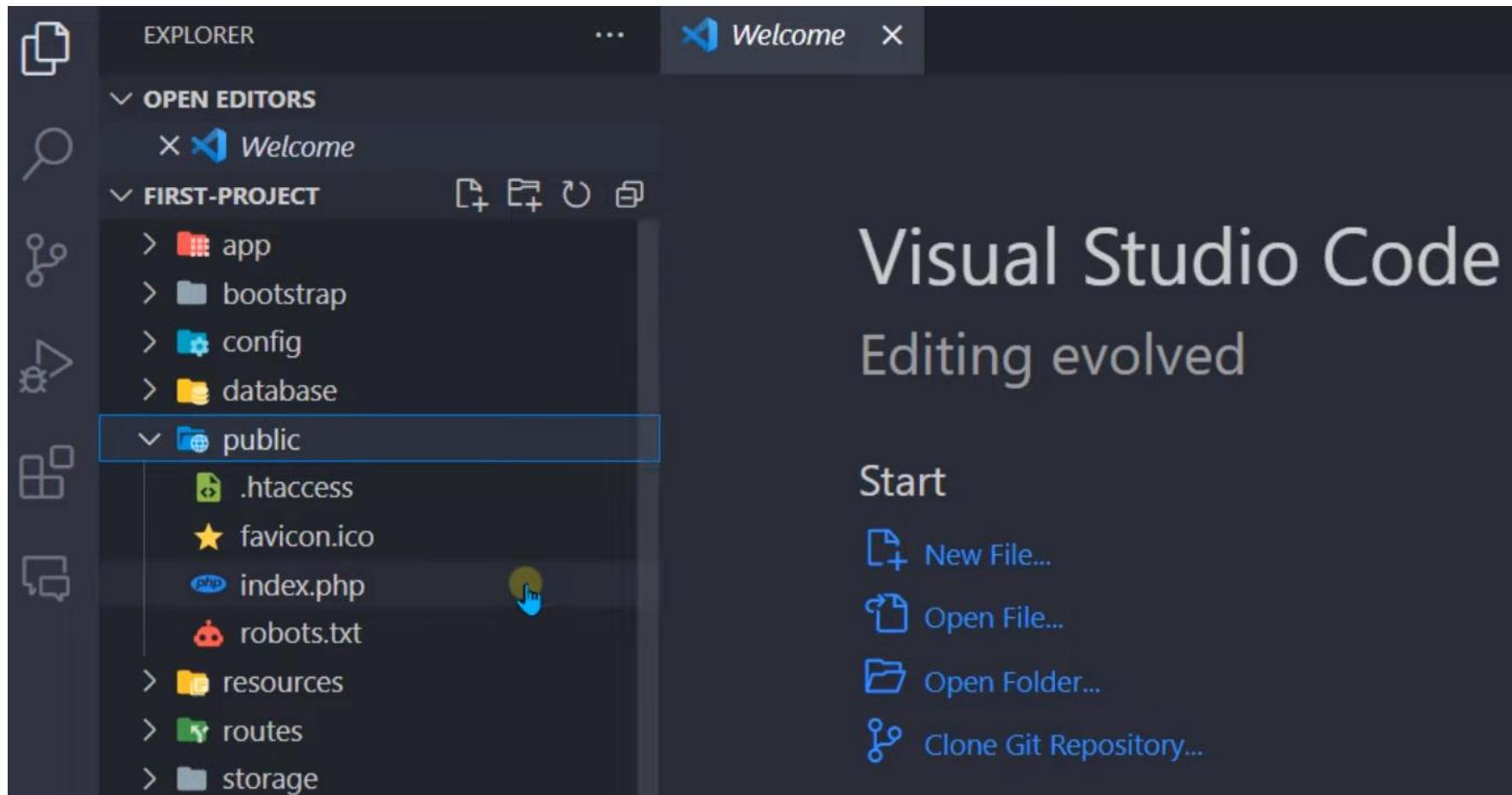
Visual Studio Code  
Editing evolved

Start

- New File...
- Open File...
- Open Folder...
- Clone Git Repository...

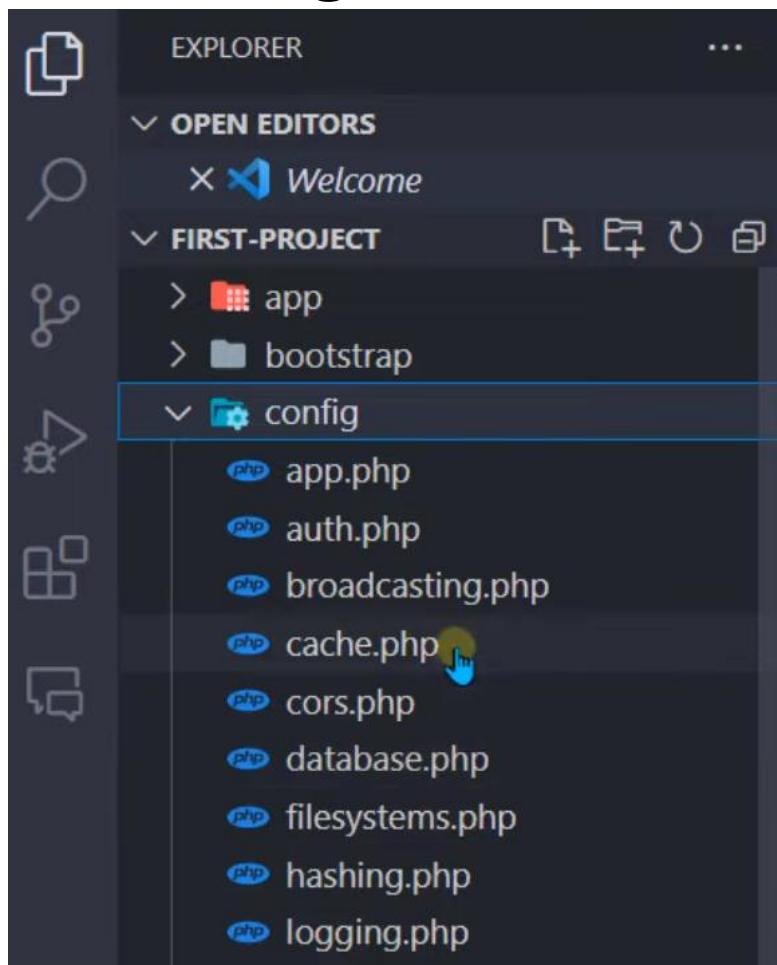
# Laravel Folder & File Structure

- Public



# Laravel Folder & File Structure

- config



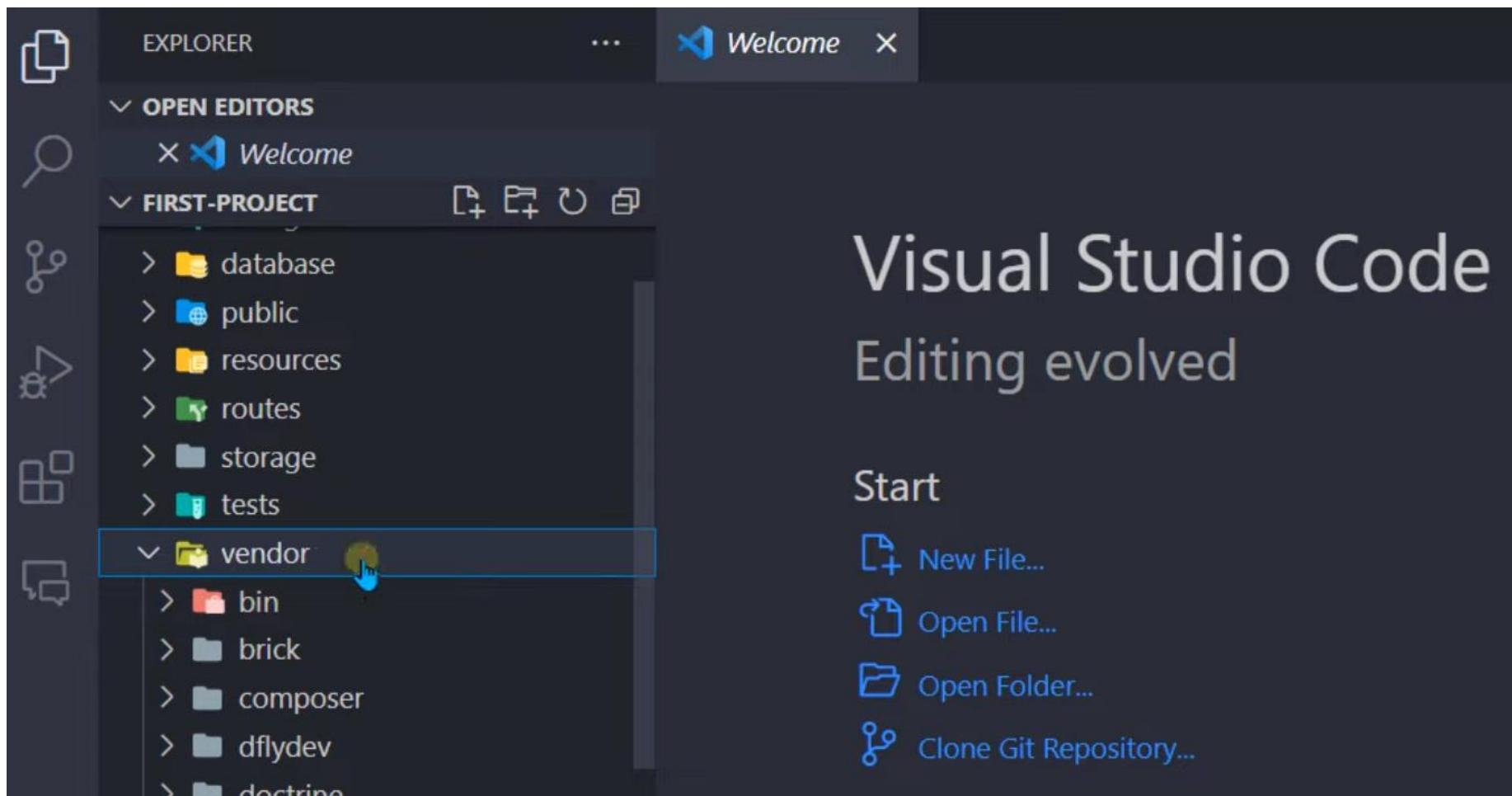
Visual Studio Code  
Editing evolved

Start

- New File...
- Open File...
- Open Folder...
- Clone Git Repository...

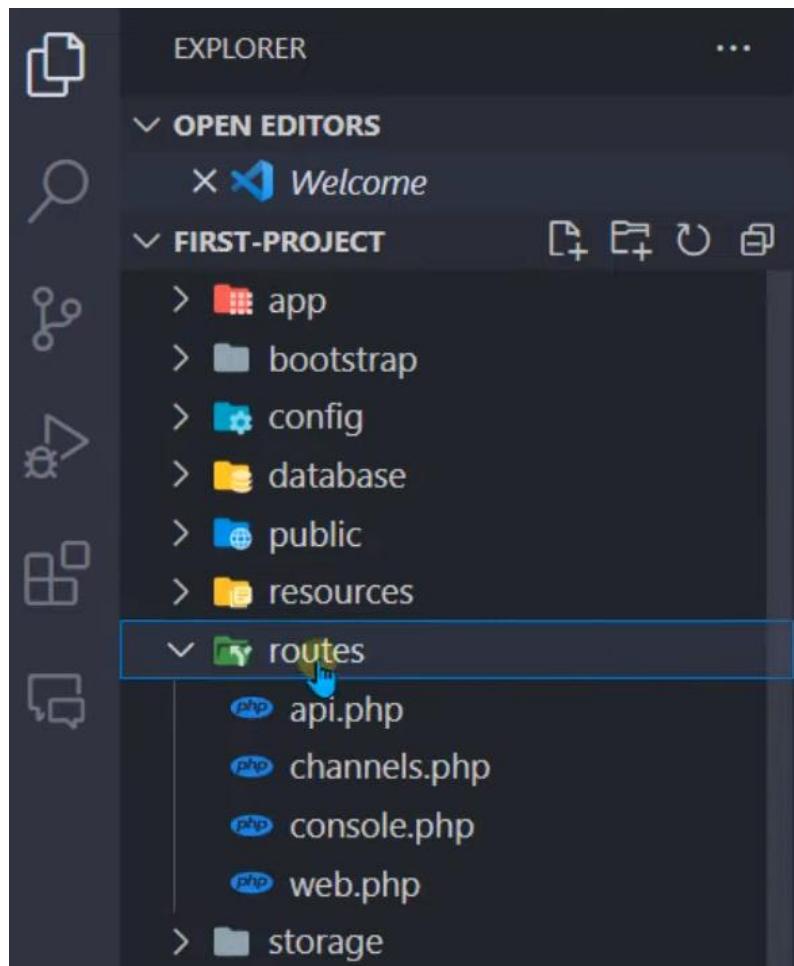
# Laravel Folder & File Structure

- vendor



# Laravel Folder & File Structure

- routes



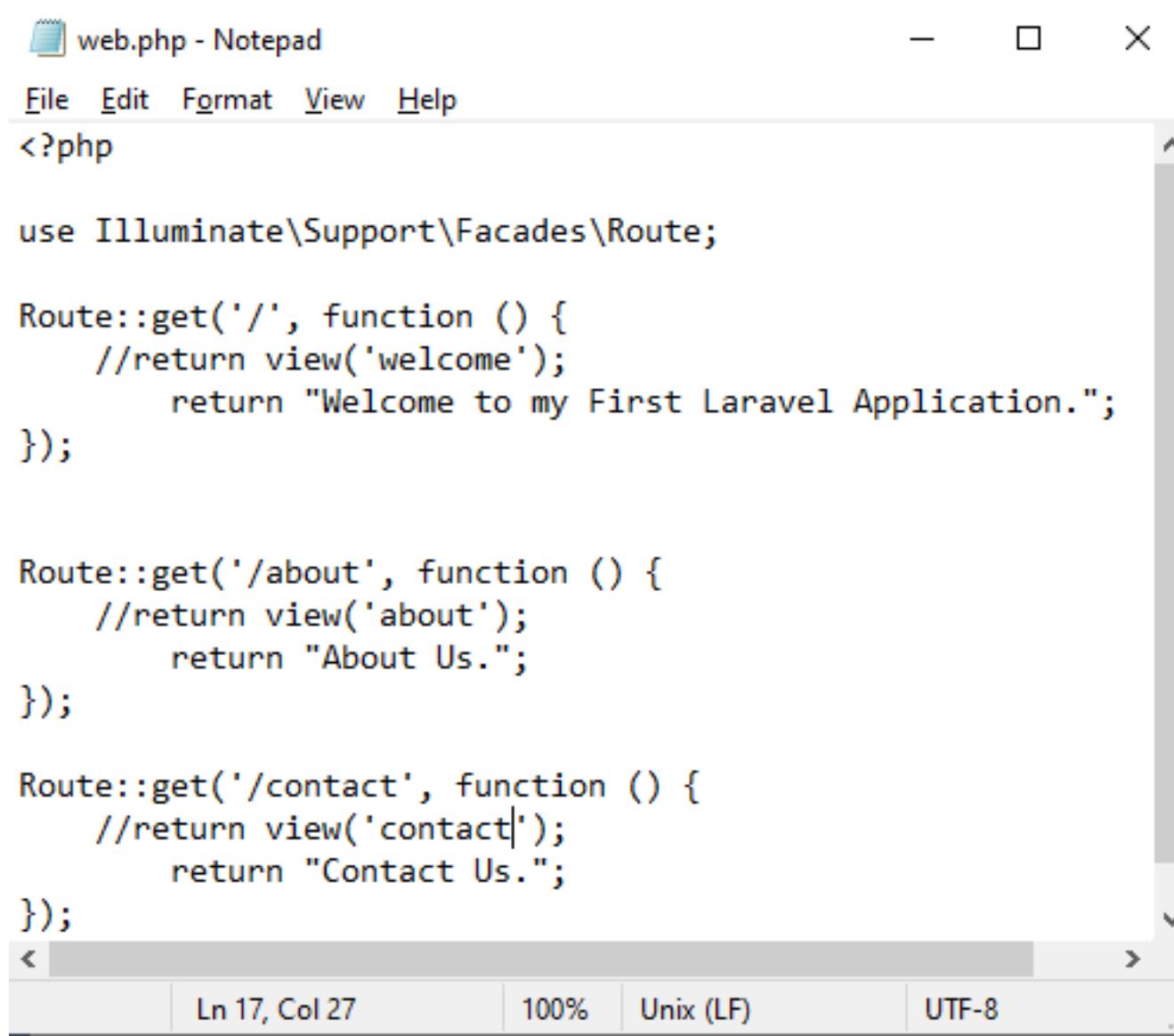
# Visual Studio Code

## Editing evolved

Start

-  New File...
  -  Open File...
  -  Open Folder...
  -  Clone Git Repository...

# 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

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



## Optional Route Parameters

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



# Route Parameters

```
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

```
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

http://localhost/post/10 → whereNumber('id')

http://localhost/post/yahoobaba → whereAlpha('name')

http://localhost/post/news10 → whereAlphaNumeric('name')

http://localhost/post/song → whereIn('category', ['movie', 'song'])

http://localhost/post/@10 → where('id', '[@0-9]+')

Regular Expressions

# 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

```
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

```
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

```
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');
```

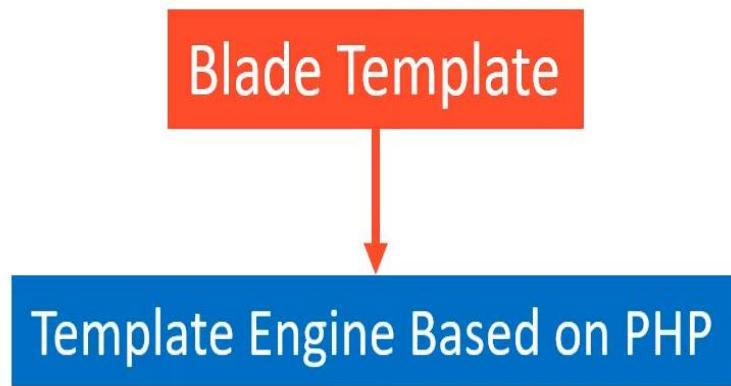
# Middleware Parameters

- 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  
    echo "Hello";  
?>
```

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

Prevent cross-site scripting (XSS) Attacks

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

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

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

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

# Blade Template Syntax

```
<?php
```

```
?>
```

```
@php
```

```
@endphp
```

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

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

# Blade Control Structure Syntax

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

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

# Blade Control Structure Syntax

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

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

```
<?php
  if(empty($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

first.blade.php

```
<h1>First Page</h1>
```

```
{{ $status }}
```

second.blade.php

```
<h1>Second Page</h1>
```

```
@include('first')
```

Include View



@include()

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

# Blade @include directives

```
@include('pages.header')
```

```
<h1>Home Page</h1>
```

```
@include('pages.footer')
```

# Blade @include directives

```
@includeWhen (Condition Value, 'viewfile', ['status' => 'Hello'])
```

TRUE / FALSE

```
@includeUnless (Condition Value, 'viewfile', ['status' => 'Hello'])
```

# Blade @include directives

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

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

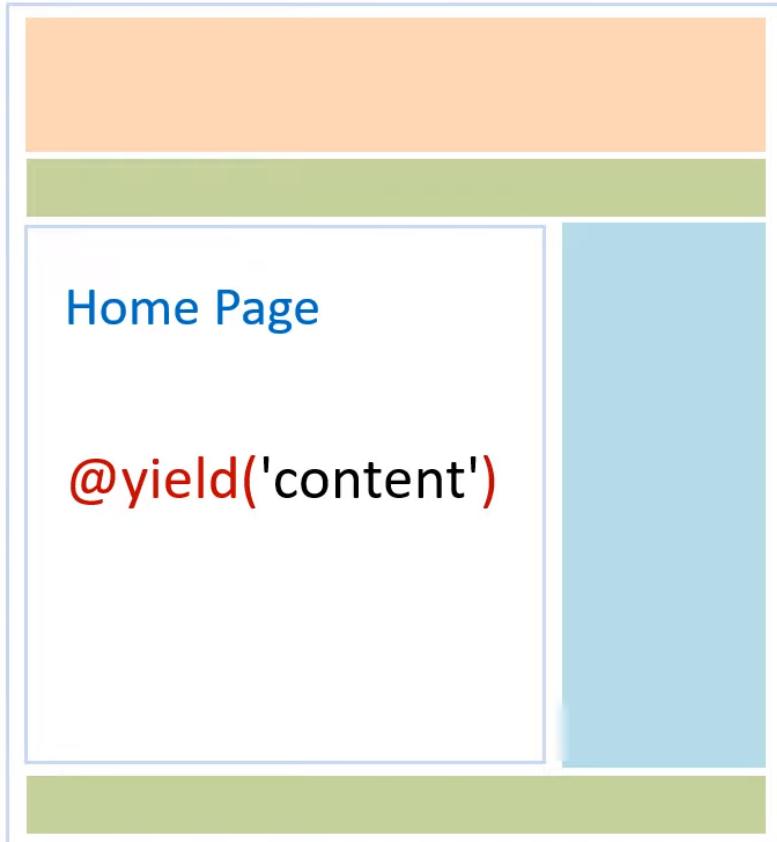
<h1>Home Page</h1>

@include('pages.footer')

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

# Blade template Inheritance

layout.blade.php

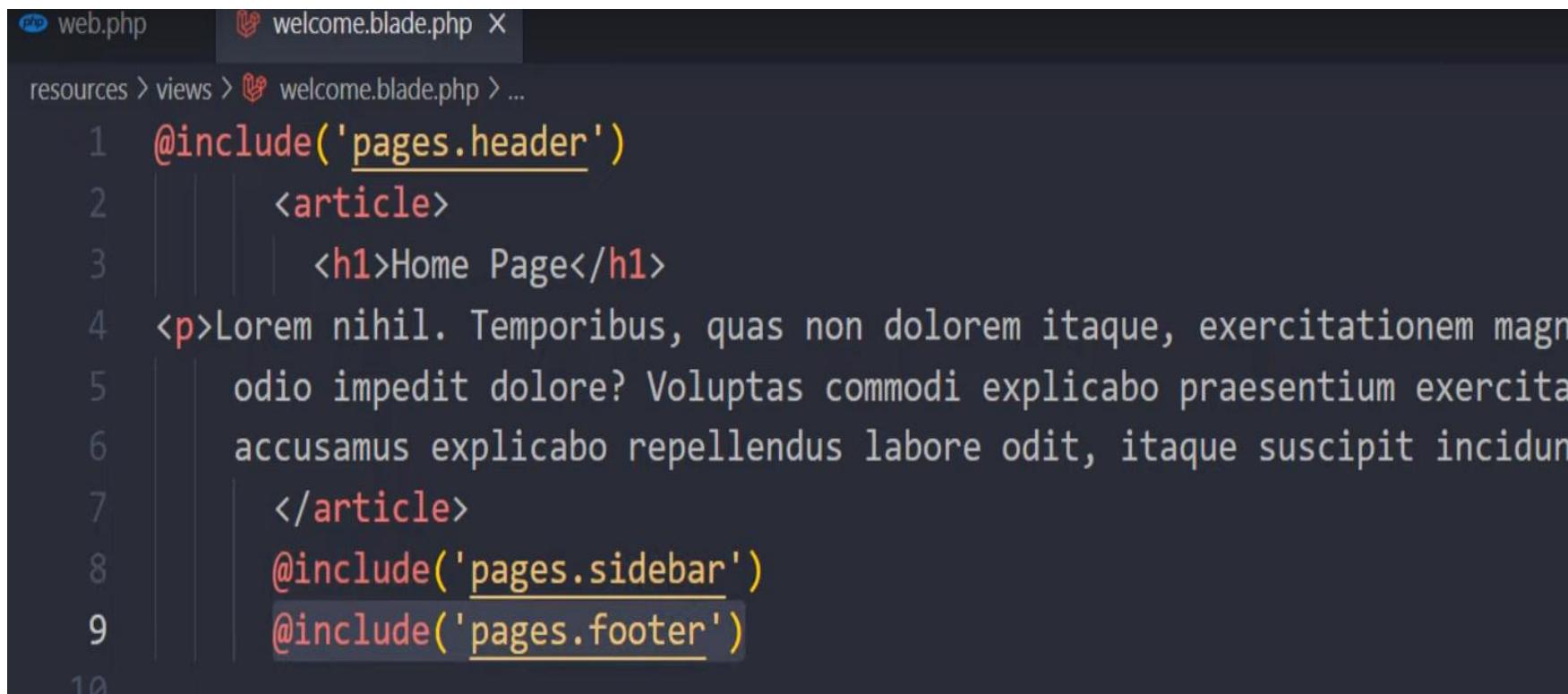


home.blade.php

```
@extends('layout')

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

# Blade template Inheritance

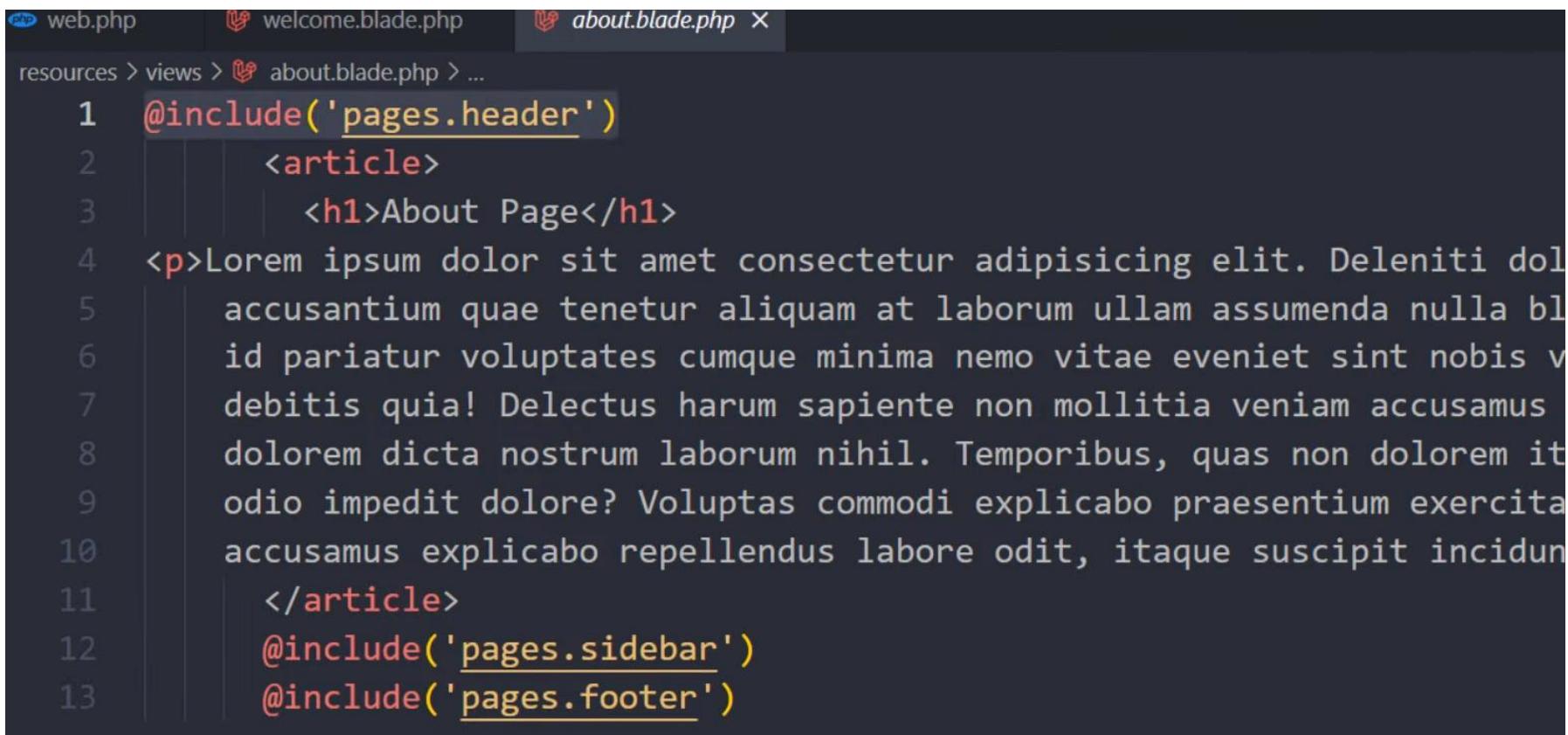


The screenshot shows a code editor with a dark theme. The top bar has tabs for 'web.php' and 'welcome.blade.php X'. Below the tabs, the file path is shown as 'resources > views > welcome.blade.php > ...'. The code itself is a Blade template with the following structure:

```
1 @include('pages.header')
2 <article>
3     <h1>Home Page</h1>
4     <p>Lorem nihil. Temporibus, quas non dolorem itaque, exercitationem magn
5         odio impedit dolore? Voluptas commodi explicabo praesentium exercita
6         accusamus explicabo repellendus labore odit, itaque suscipit incidun
7     </article>
8     @include('pages.sidebar')
9     @include('pages.footer')
```

The code is numbered from 1 to 10 on the left. The 'header', 'sidebar', and 'footer' sections are highlighted in yellow, indicating they are included from separate files.

# Blade template Inheritance



The screenshot shows a code editor with a dark theme. At the top, there are tabs for 'web.php', 'welcome.blade.php', and 'about.blade.php' (which is currently active). Below the tabs, the file structure is shown as 'resources > views > about.blade.php > ...'. The code itself is a Blade template with the following content:

```
1 @include('pages.header')
2 <article>
3     <h1>About Page</h1>
4     <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Deleniti dol
5         accusantium quae tenetur aliquam at laborum ullam assumenda nulla bl
6         id pariatur voluptates cumque minima nemo vitae eveniet sint nobis v
7         debitis quia! Delectus harum sapiente non mollitia veniam accusamus
8         dolorem dicta nostrum laborum nihil. Temporibus, quas non dolorem it
9         odio impedit dolore? Voluptas commodi explicabo praesentium exercita
10        accusamus explicabo repellendus labore odit, itaque suscipit incidun
11        </article>
12        @include('pages.sidebar')
13        @include('pages.footer')
```

# Blade template Inheritance

[Home](#) | [About](#) | [Post](#)

## Home Page

Quae tenetur aliquam at laborum ullam assumenda nulla blanditiis. Veniam delectus magni quisquam numquam id pariatur voluptates cumque minima nemo vitae eveniet sint nobis veritatis omnis unde, natus rem, praesentium debitibus quia! Delectus harum sapiente non mollitia veniam accusamus quas recusandae, repellendus tenetur illo dolorem dicta nostrum laborum nihil. Temporibus, quas non dolorem itaque, exercitationem magni praesentium omnis a odio impedit dolore? Voluptas commodi explicabo praesentium exercitationem culpa tenetur dolorem officiis? Impedit, accusamus explicabo repellendus labore odit, itaque suscipit incident dignissimos, dolorem autem fugiat aperiam?

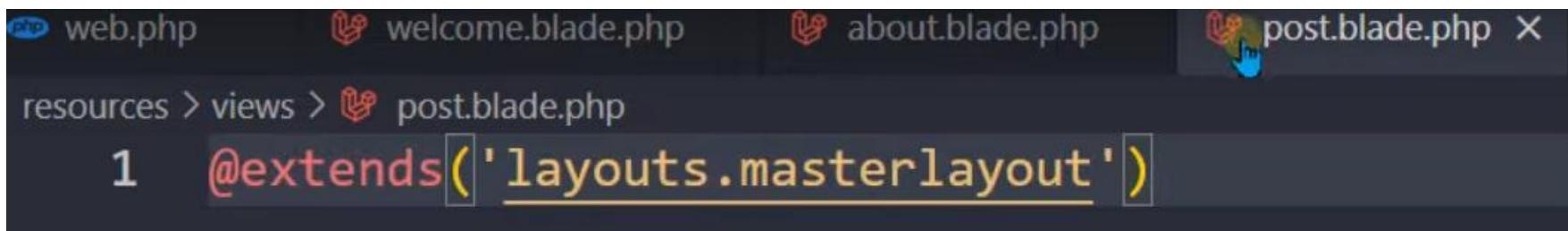
- [Home](#)
  - [About](#)
  - [Post](#)

# Blade template Inheritance

- Masterlayout.blade.php

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

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

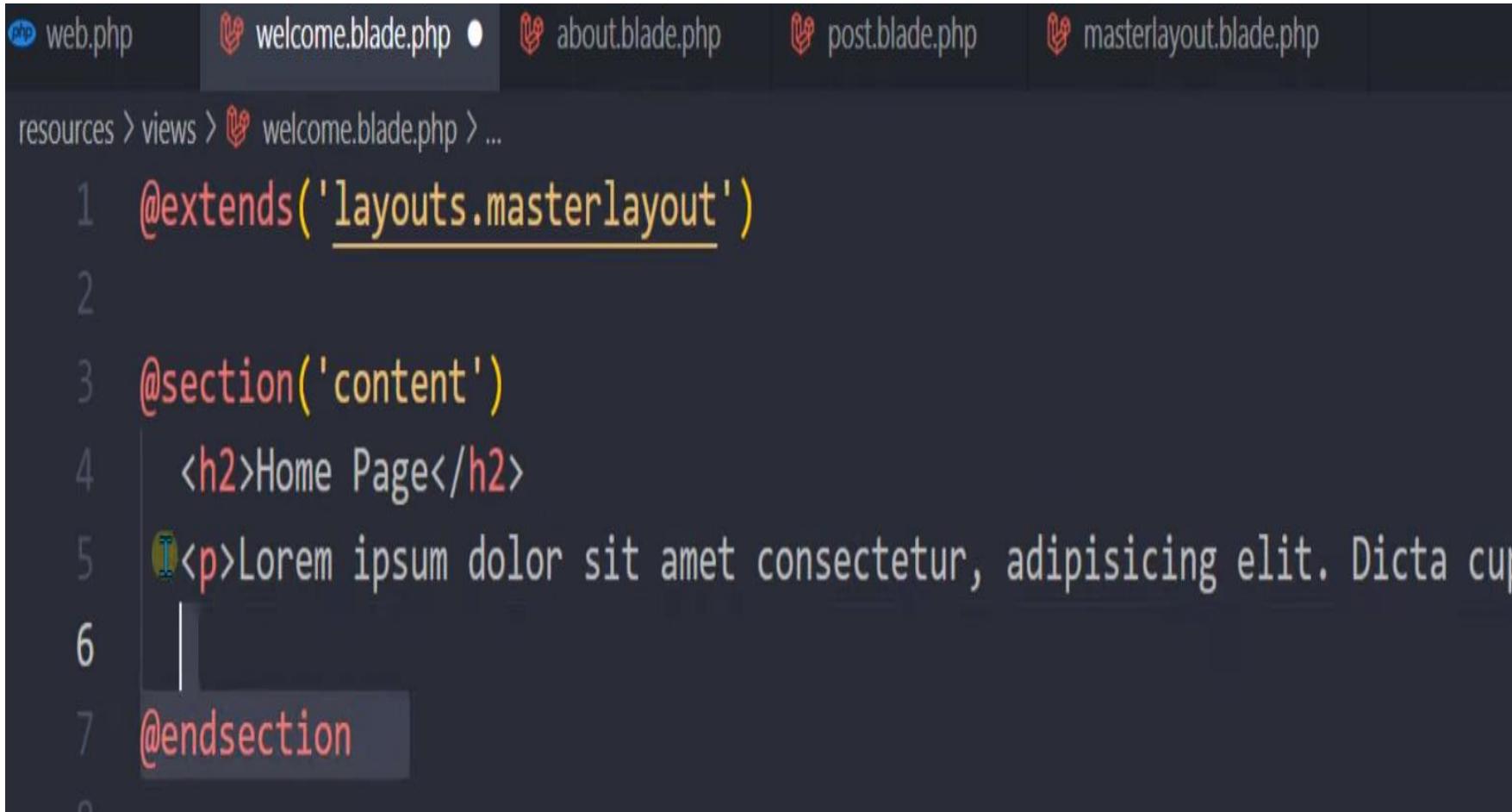


The screenshot shows a code editor with the following details:

- File tabs: web.php, welcome.blade.php, about.blade.php, post.blade.php (highlighted with a cursor icon).
- File path: resources > views > post.blade.php
- Code content:

```
1 @extends(['layouts.masterlayout'])
```

# Blade template Inheritance

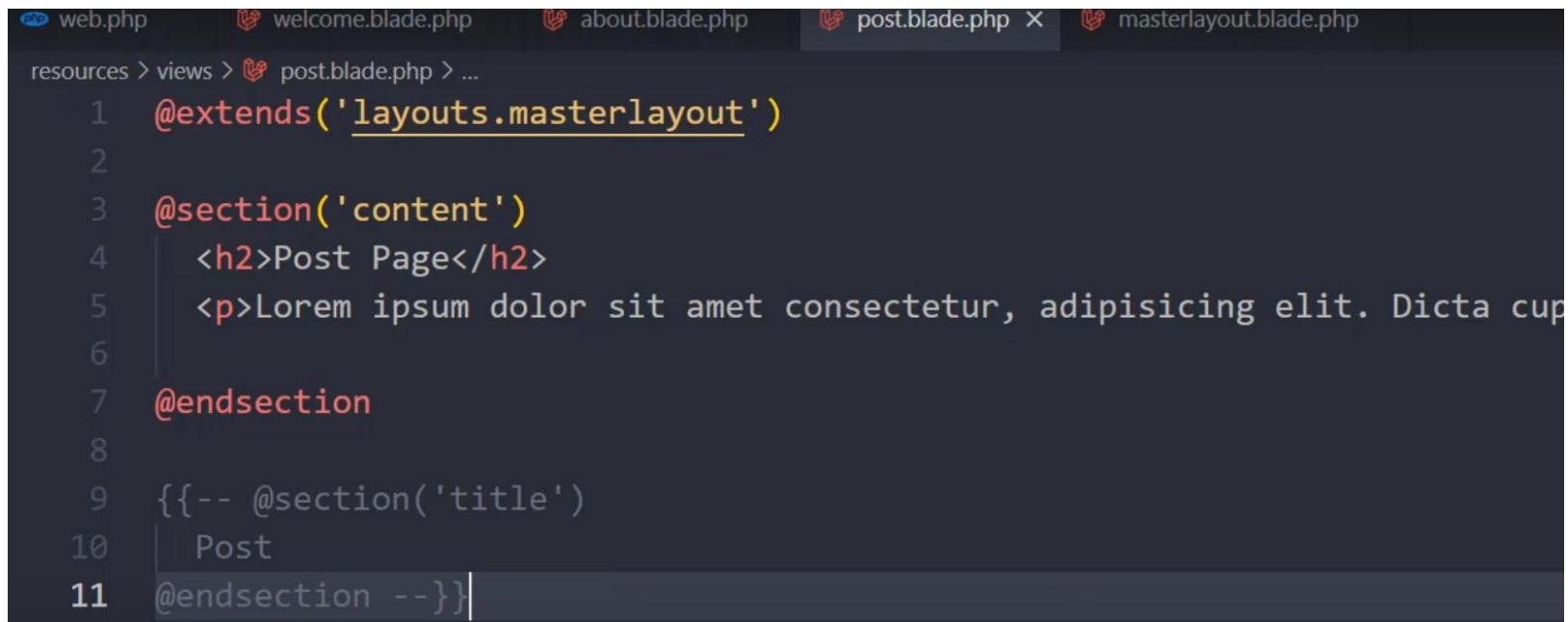


resources > views > welcome.blade.php > ...

```
1 @extends('layouts.masterlayout')
2
3 @section('content')
4     <h2>Home Page</h2>
5     <p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Dicta cup
6
7     @endsection
8
```

# Blade template Inheritance

- `@yield('identifier', 'default_identifier')`

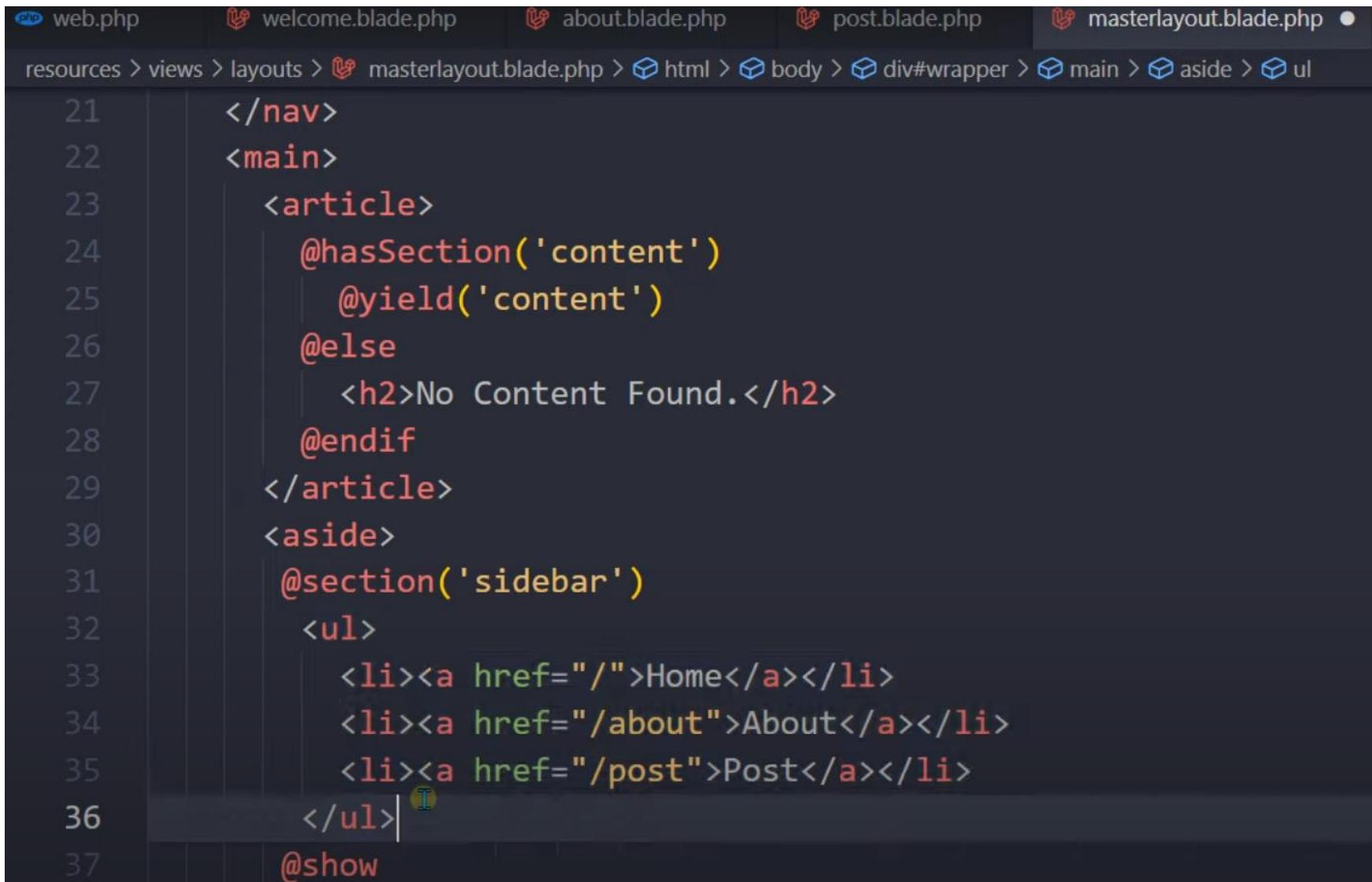


The image shows a code editor with a dark theme. The title bar has tabs for 'web.php', 'welcome.blade.php', 'about.blade.php', 'post.blade.php' (which is the active tab), and 'masterlayout.blade.php'. The file path 'resources > views > post.blade.php > ...' is visible. The code in the editor is:

```
1 @extends('layouts.masterlayout')
2
3 @section('content')
4     <h2>Post Page</h2>
5     <p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Dicta cup
6
7 @endsection
8
9 {{-- @section('title')
10     Post
11     @endsection --}}
```

# Blade template Inheritance

# Blade template Inheritance



The screenshot shows a code editor with a dark theme displaying a Blade template. The template structure includes a navigation bar, a main content area with an article section and a sidebar, and a sidebar section with a list of links. The code uses Blade syntax for sections and yields.

```
21  </nav>
22  <main>
23      <article>
24          @hasSection('content')
25              @yield('content')
26          @else
27              <h2>No Content Found.</h2>
28          @endif
29      </article>
30      <aside>
31          @section('sidebar')
32              <ul>
33                  <li><a href="/">Home</a></li>
34                  <li><a href="/about">About</a></li>
35                  <li><a href="/post">Post</a></li>
36              </ul> I
37          @show
```

# Blade template Inheritance

php web.php

blade welcome.blade.php

blade about.blade.php

blade 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
```