# Meet Laravel

Laravel is a web application framework with expressive, elegant syntax. A web framework provides a structure and starting point for creating your application, allowing you to focus on creating something amazing while we sweat the details.

# Why Laravel?

There are a variety of tools and frameworks available to you when building a web application. However, we believe Laravel is the best choice for building modern, full-stack web applications.

# Installation Via Composer

Composer download link: [Composer (getcomposer.org)](https://getcomposer.org/download/)

If your local machine already has PHP and Composer installed, you may create a new Laravel project by using Composer directly. After the application has been created, you may start Laravel's local development server using the Artisan CLI's serve command:

composer create-project laravel/laravel example-app

cd example-app

php artisan serve

Once you have started the Artisan development server, you may access your application at <http://localhost:8000>.

# Initial Configuration

All of the configuration files for the Laravel framework are stored in the config directory. Each option is documented, so feel free to look through the files and get familiar with the options available to you.

# Directory Structure

The Root Directory

The app Directory

The bootstrap Directory

The config Directory

The database Directory

The lang Directory

The public Directory

The resources Directory

The routes Directory

The storage Directory

The tests Directory

The vendor Directory

The App Directory

The Broadcasting Directory

The Console Directory

The Events Directory

The Exceptions Directory

The Http Directory

The Jobs Directory

The Listeners Directory

The Mail Directory

The Models Directory

The Notifications Directory

The Policies Directory

The Providers Directory

The Rules Directory

Introduction

The default Laravel application structure is intended to provide a great starting point for both large and small applications. But you are free to organize your application however you like. Laravel imposes almost no restrictions on where any given class is located - as long as Composer can autoload the class.

The Root Directory

The App Directory

The app directory contains the core code of your application. We'll explore this directory in more detail soon; however, almost all of the classes in your application will be in this directory.

The Bootstrap Directory

The bootstrap directory contains the app.php file which bootstraps the framework. This directory also houses a cache directory which contains framework generated files for performance optimization such as the route and services cache files. You should not typically need to modify any files within this directory.

The Config Directory

The config directory, as the name implies, contains all of your application's configuration files. It's a great idea to read through all of these files and familiarize yourself with all of the options available to you.

The Database Directory

The database directory contains your database migrations, model factories, and seeds. If you wish, you may also use this directory to hold an SQLite database.

The Lang Directory

The lang directory houses all of your application's language files.

The Public Directory

The public directory contains the index.php file, which is the entry point for all requests entering your application and configures autoloading. This directory also houses your assets such as images, JavaScript, and CSS.

The Resources Directory

The resources directory contains your views as well as your raw, un-compiled assets such as CSS or JavaScript.

The Routes Directory

The routes directory contains all of the route definitions for your application. By default, several route files are included with Laravel: web.php, api.php, console.php, and channels.php.

The web.php file contains routes that the RouteServiceProvider places in the web middleware group, which provides session state, CSRF protection, and cookie encryption. If your application does not offer a stateless, RESTful API then it is likely that all of your routes will most likely be defined in the web.php file.

The api.php file contains routes that the RouteServiceProvider places in the api middleware group. These routes are intended to be stateless, so requests entering the application through these routes are intended to be authenticated via tokens and will not have access to session state.

The console.php file is where you may define all of your closure based console commands. Each closure is bound to a command instance allowing a simple approach to interacting with each command's IO methods. Even though this file does not define HTTP routes, it defines console based entry points (routes) into your application.

The channels.php file is where you may register all of the event broadcasting channels that your application supports.

The Storage Directory

The storage directory contains your logs, compiled Blade templates, file based sessions, file caches, and other files generated by the framework. This directory is segregated into app, framework, and logs directories. The app directory may be used to store any files generated by your application. The framework directory is used to store framework generated files and caches. Finally, the logs directory contains your application's log files.

The storage/app/public directory may be used to store user-generated files, such as profile avatars, that should be publicly accessible. You should create a symbolic link at public/storage which points to this directory. You may create the link using the php artisan storage:link Artisan command.

The Tests Directory

The tests directory contains your automated tests. Example PHPUnit unit tests and feature tests are provided out of the box. Each test class should be suffixed with the word Test. You may run your tests using the phpunit or php vendor/bin/phpunit commands. Or, if you would like a more detailed and beautiful representation of your test results, you may run your tests using the php artisan test Artisan command.

The Vendor Directory

The vendor directory contains your Composer dependencies.

The App Directory

The majority of your application is housed in the app directory. By default, this directory is namespaced under App and is autoloaded by Composer using the PSR-4 autoloading standard.

The app directory contains a variety of additional directories such as Console, Http, and Providers. Think of the Console and Http directories as providing an API into the core of your application. The HTTP protocol and CLI are both mechanisms to interact with your application, but do not actually contain application logic. In other words, they are two ways of issuing commands to your application. The Console directory contains all of your Artisan commands, while the Http directory contains your controllers, middleware, and requests.

A variety of other directories will be generated inside the app directory as you use the make Artisan commands to generate classes. So, for example, the app/Jobs directory will not exist until you execute the make:job Artisan command to generate a job class.

Many of the classes in the app directory can be generated by Artisan via commands. To review the available commands, run the php artisan list make command in your terminal.

The Broadcasting Directory

The Broadcasting directory contains all of the broadcast channel classes for your application. These classes are generated using the make:channel command. This directory does not exist by default, but will be created for you when you create your first channel. To learn more about channels, check out the documentation on event broadcasting.

The Console Directory

The Console directory contains all of the custom Artisan commands for your application. These commands may be generated using the make:command command. This directory also houses your console kernel, which is where your custom Artisan commands are registered and your scheduled tasks are defined.

The Events Directory

This directory does not exist by default, but will be created for you by the event:generate and make:event Artisan commands. The Events directory houses event classes. Events may be used to alert other parts of your application that a given action has occurred, providing a great deal of flexibility and decoupling.

The Exceptions Directory

The Exceptions directory contains your application's exception handler and is also a good place to place any exceptions thrown by your application. If you would like to customize how your exceptions are logged or rendered, you should modify the Handler class in this directory.

The Http Directory

The Http directory contains your controllers, middleware, and form requests. Almost all of the logic to handle requests entering your application will be placed in this directory.

The Jobs Directory

This directory does not exist by default, but will be created for you if you execute the make:job Artisan command. The Jobs directory houses the queueable jobs for your application. Jobs may be queued by your application or run synchronously within the current request lifecycle. Jobs that run synchronously during the current request are sometimes referred to as "commands" since they are an implementation of the command pattern.

The Listeners Directory

This directory does not exist by default, but will be created for you if you execute the event:generate or make:listener Artisan commands. The Listeners directory contains the classes that handle your events. Event listeners receive an event instance and perform logic in response to the event being fired. For example, a UserRegistered event might be handled by a SendWelcomeEmail listener.

The Mail Directory

This directory does not exist by default, but will be created for you if you execute the make:mail Artisan command. The Mail directory contains all of your classes that represent emails sent by your application. Mail objects allow you to encapsulate all of the logic of building an email in a single, simple class that may be sent using the Mail::send method.

The Models Directory

The Models directory contains all of your Eloquent model classes. The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.

The Notifications Directory

This directory does not exist by default, but will be created for you if you execute the make:notification Artisan command. The Notifications directory contains all of the "transactional" notifications that are sent by your application, such as simple notifications about events that happen within your application. Laravel's notification feature abstracts sending notifications over a variety of drivers such as email, Slack, SMS, or stored in a database.

The Policies Directory

This directory does not exist by default, but will be created for you if you execute the make:policy Artisan command. The Policies directory contains the authorization policy classes for your application. Policies are used to determine if a user can perform a given action against a resource.

The Providers Directory

The Providers directory contains all of the service providers for your application. Service providers bootstrap your application by binding services in the service container, registering events, or performing any other tasks to prepare your application for incoming requests.

In a fresh Laravel application, this directory will already contain several providers. You are free to add your own providers to this directory as needed.

The Rules Directory

This directory does not exist by default, but will be created for you if you execute the make:rule Artisan command. The Rules directory contains the custom validation rule objects for your application. Rules are used to encapsulate complicated validation logic in a simple object. For more information, check out the validation documentation.

# The Basics

## Routing

Basic Routing

use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {

return 'Hello World';

});

## Middleware

Defining Middleware

php artisan make:middleware EnsureTokenIsValid

## CSRF Protection

## Controllers

Basic Controller

<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

use App\Models\User;

class UserController extends Controller

{

/\*\*

\* Show the profile for a given user.

\*

\* @param int $id

\* @return \Illuminate\View\View

\*/

public function show($id)

{

return view('user.profile', [

'user' => User::findOrFail($id)

]);

}

}

Route for controller

use App\Http\Controllers\UserController;

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

Controller Middleware in routes

Route::get('profile', [UserController::class, 'show'])->middleware('auth');

Middleware for controller in constructor

class UserController extends Controller

{

/\*\*

\* Instantiate a new controller instance.

\*

\* @return void

\*/

public function \_\_construct()

{

$this->middleware('auth');

$this->middleware('log')->only('index');

$this->middleware('subscribed')->except('store');

}

}

### Resource Controllers

Laravel resource routing assigns the typical create, read, update, and delete ("CRUD") routes to a controller with a single line of code. To get started, we can use the make:controller Artisan command's --resource option to quickly create a controller to handle these actions:

php artisan make:controller PhotoController –resource

----------------------------------------------------

use App\Http\Controllers\PhotoController;

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

Actions Handled By Resource Controller

Verb URI Action Route Name

GET /photos index photos.index

GET /photos/create create photos.create

POST /photos store photos.store

GET /photos/{photo} show photos.show

GET /photos/{photo}/edit edit photos.edit

PUT/PATCH /photos/{photo} update photos.update

DELETE /photos/{photo} destroy photos.destroy

API Resource Routes

php artisan make:controller PhotoController –api

-------------------------------------------------

use App\Http\Controllers\PhotoController;

Route::apiResource('photos', PhotoController::class);

## Blade Templates

Displaying Data

Hello, {{ $name }}.

Hello, {!! $name !!}.

Blade Directives

If Statements

@if (count($records) === 1)

I have one record!

@elseif (count($records) > 1)

I have multiple records!

@else

I don't have any records!

@endif

------------------------------

@unless (Auth::check())

You are not signed in.

@endunless

------------------------------

@isset($records)

// $records is defined and is not null...

@endisset

@empty($records)

// $records is "empty"...

@endempty

Authentication Directives

@auth

// The user is authenticated...

@endauth

@guest

// The user is not authenticated...

@endguest

-------------------------------

@auth('admin')

// The user is authenticated...

@endauth

@guest('admin')

// The user is not authenticated...

@endguest

Switch Statements

@switch($i)

@case(1)

First case...

@break

@case(2)

Second case...

@break

@default

Default case...

@endswitch

Loops

@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

The Loop Variable

@foreach ($users as $user)

@if ($loop->first)

This is the first iteration.

@endif

@if ($loop->last)

This is the last iteration.

@endif

<p>This is user {{ $user->id }}</p>

@endforeach

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.

Layouts Using Template Inheritance

<!-- resources/views/layouts/app.blade.php -->

<html>

<head>

<title>App Name - @yield('title')</title>

</head>

<body>

@section('sidebar')

This is the master sidebar.

@show

<div class="container">

@yield('content')

</div>

</body>

</html>

Stacks and push

@push('scripts')

This will be second...

@endpush

// Later...

@prepend('scripts')

This will be first...

@endprepend

Guide [How to Create Layout Using Laravel Blade [Ultimate Guide] (cloudways.com)](https://www.cloudways.com/blog/create-laravel-blade-layout/)

## Validation

# Demo App

Followed command will create a simple login and registration complete functionality

#### Laravel UI Installation

composer **require** laravel/ui

#### Generate basic scaffolding and login and registration for bootstrap

php artisan ui bootstrap

php artisan ui bootstrap --auth

After generating UI need to install npm dependencies.

npm run install && npm run dev

Next, Migrate the database

php artisan migrate

Questions & Answers:

1-how to override error of pattern Route?  
2-After and before middleware?

3-how to bind form requests with controller?  
in any controller or other classes just we pass our custom created from request instead of request default class.  
as follows:  
public function store(StorePostRequest $request)

{  
 // The incoming request is valid...

// Retrieve the validated input data...

$validated = $request->validated();

// Retrieve a portion of the validated input data...

$validated = $request->safe()->only(['name', 'email']);

$validated = $request->safe()->except(['name', 'email']);

}