Directory Structure

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.

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 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, two route files are included with Laravel: `web.php` and `console.php`.

The `web.php` file contains routes that Laravel 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 all your routes will most likely be defined in the `web.php` file.

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. You may also schedule tasks in the `console.php` file.

Optionally, you may install additional route files for API routes (`api.php`) and broadcasting channels (`channels.php`), via the `install:api` and `install:broadcasting` Artisan commands.

The `api.php` file contains routes that 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 `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 Pest or 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 `/vendor/bin/pest` or `/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.

By default, the `app` directory contains the `Http`, `Models`, and `Providers` directories. However, over time, a variety of other directories will be generated inside the app directory as you use the make Artisan commands to generate classes. For example, the `app/Console` directory will not exist until you execute the `make:command` Artisan command to generate a command class.

Both the `Console` and `Http` directories are further explained in their respective sections below, but 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.

> [!NOTE] 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.

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 all of the custom exceptions for your application. These exceptions may be generated using the `make:exception` command.

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 the `AppServiceProvider`. 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.