

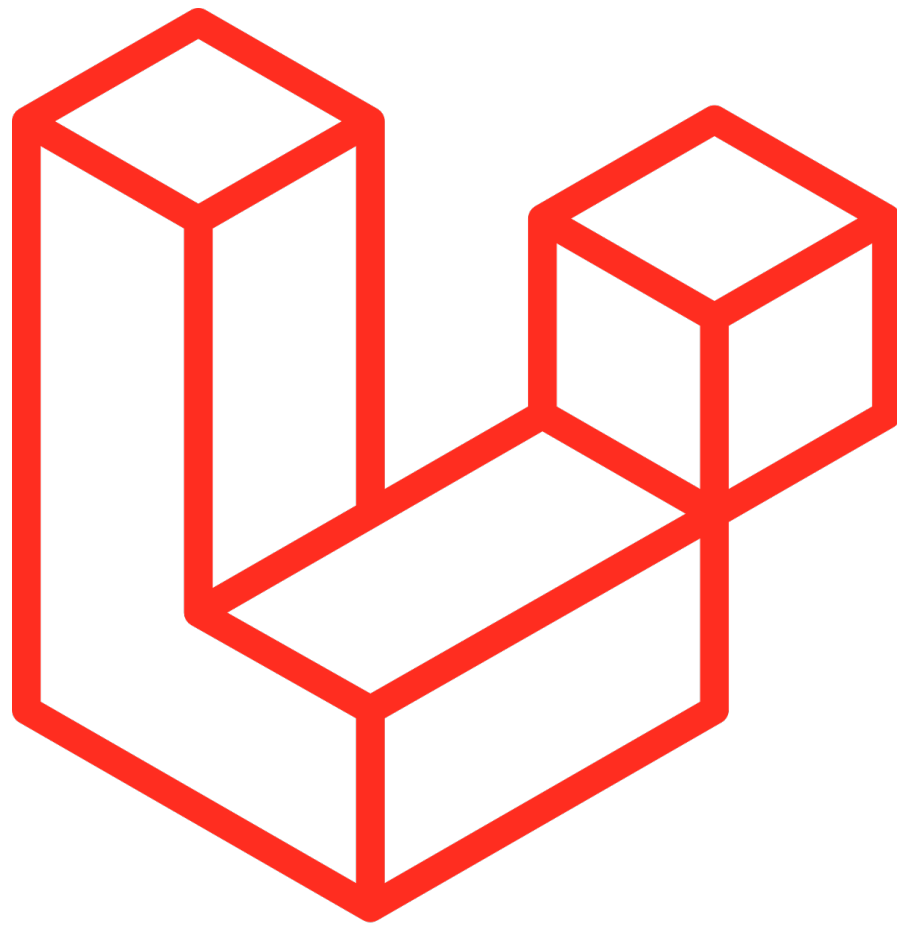


Programação para a WEB - servidor (server-side)

Sérgio da Silva Nogueira

P H P

Laravel

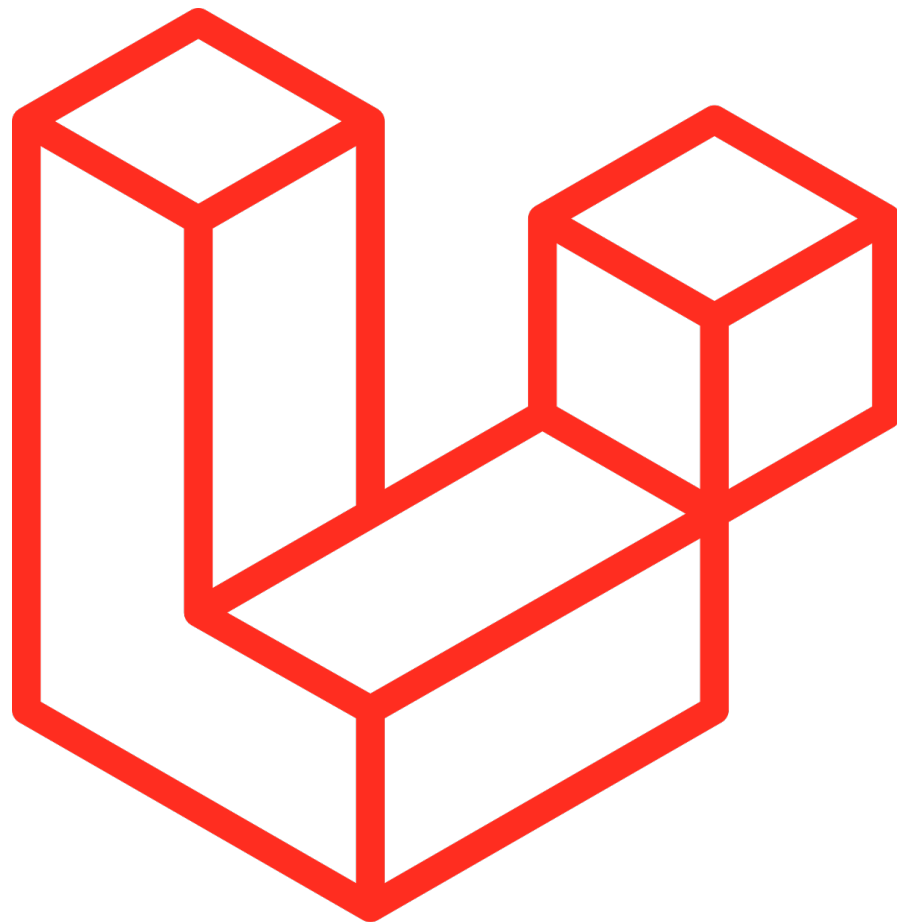


Laravel é uma framework gratuita, de código aberto, com foco para web e feito em PHP. Foi criado por Taylor Otwell e tem como principal função desenvolver aplicações web com padrão de arquitetura MVC. Algumas características nativas do Laravel são sua sintaxe simples e concisa, um sistema modular com gestor de dependências dedicado, várias formas de acesso a base de dados relacionais e vários utilitários indispensáveis no auxílio ao desenvolvimento e manutenção de sistemas.

O código fonte do Laravel está alojado no GitHub e licenciado sob os termos da licença MIT

P H P

MVC



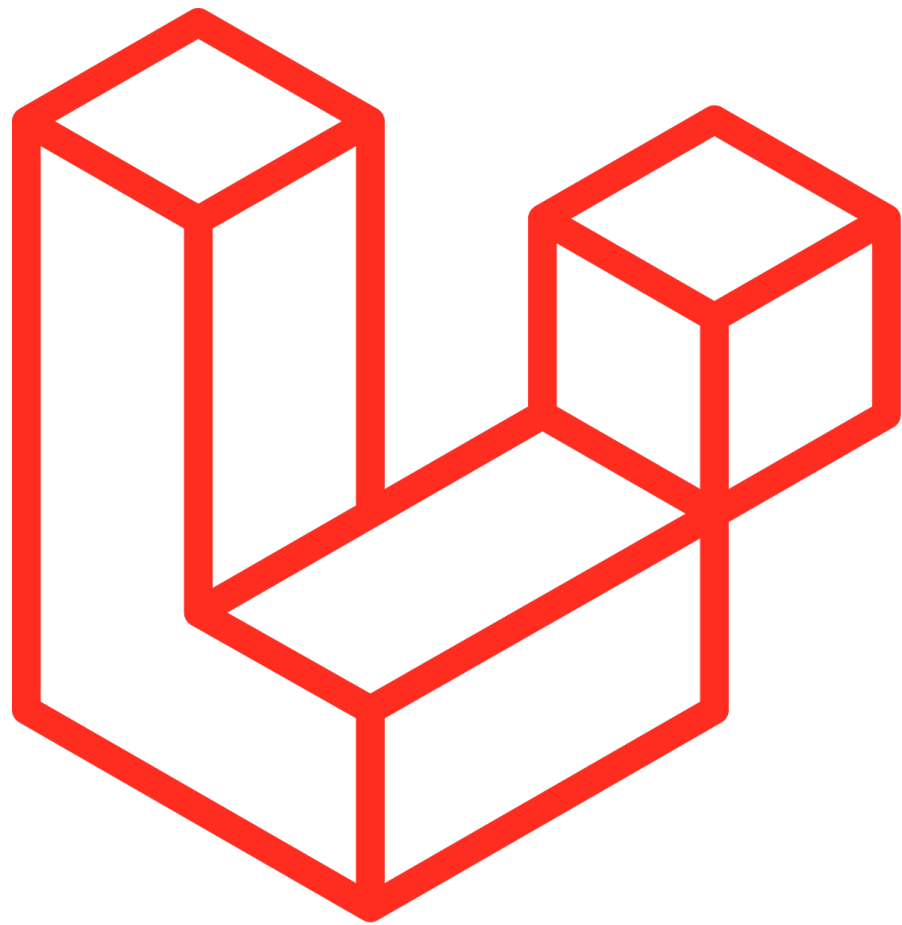
O Laravel utiliza o padrão MVC (Model, View e Controller) que, basicamente, funciona da seguinte forma:

Model é a camada responsável pela parte lógica da aplicação, ou seja, todos os recursos da sua aplicação (consultas ao BD, validações, notificações, etc), mas ele não sabe quando isso deve ser feito, a camada de model apenas tem o necessário para que tudo aconteça, mas não sabe quando irá executar.

View é a camada responsável por exibir dados para o usuário, seja em páginas HTML, JSON, XML, etc. A camada View não possui responsabilidade de saber quando vai exibir os dados, apenas como irá exibi-los.

Controller é o famoso “meio-de-campo” da aplicação. Essa é a camada que sabe quem chamar e quando chamar para executar determinada ação.

PHP MVC

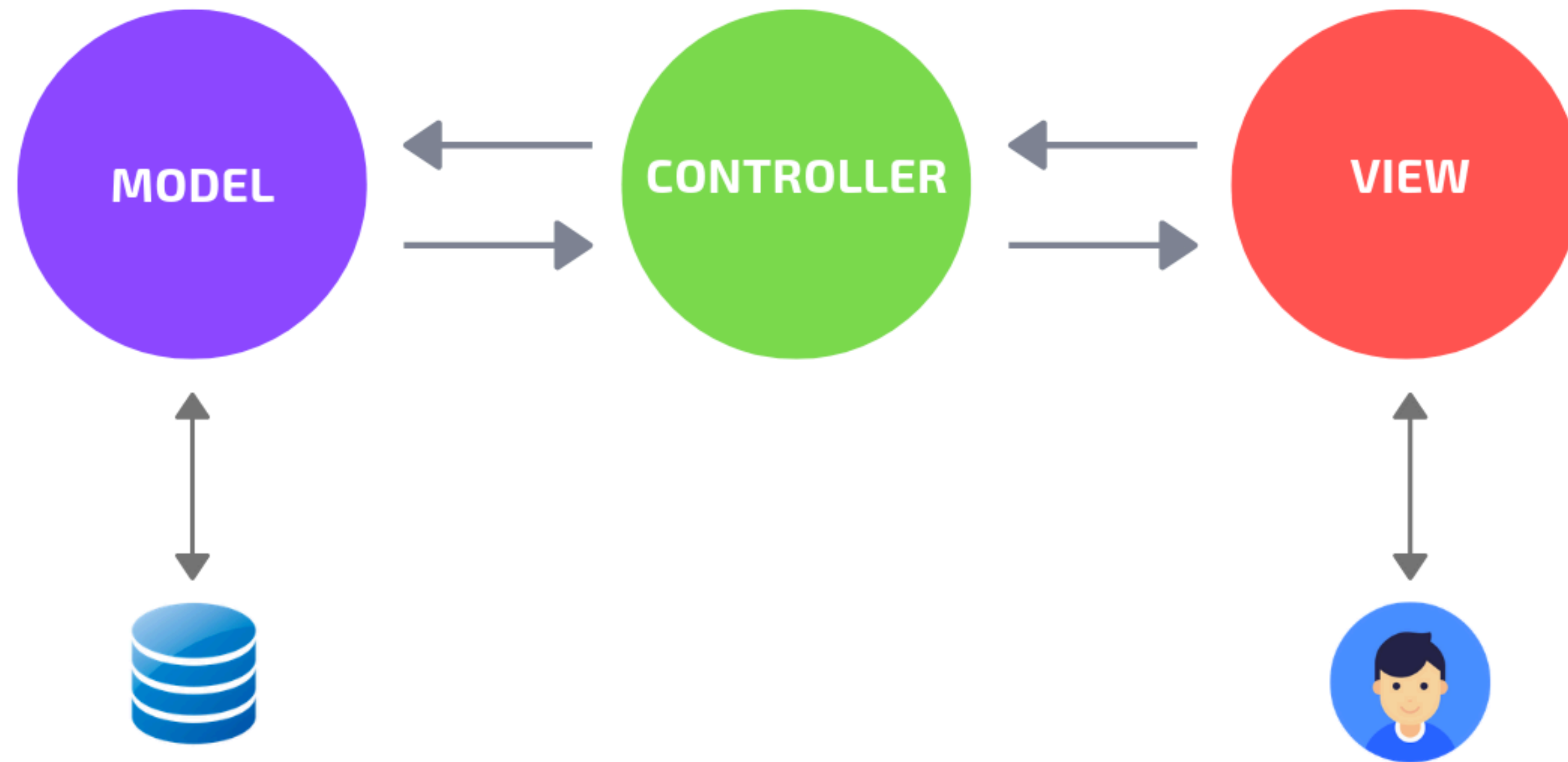


Basicamente, o MVC funciona da seguinte forma:

Ao receber uma requisição, o Controller solicita ao Model as informações necessárias (que virão do banco de dados), que as obtém e retorna ao Controller. De posse dessas informações, o Controller as envia para a View que irá renderizá-las.

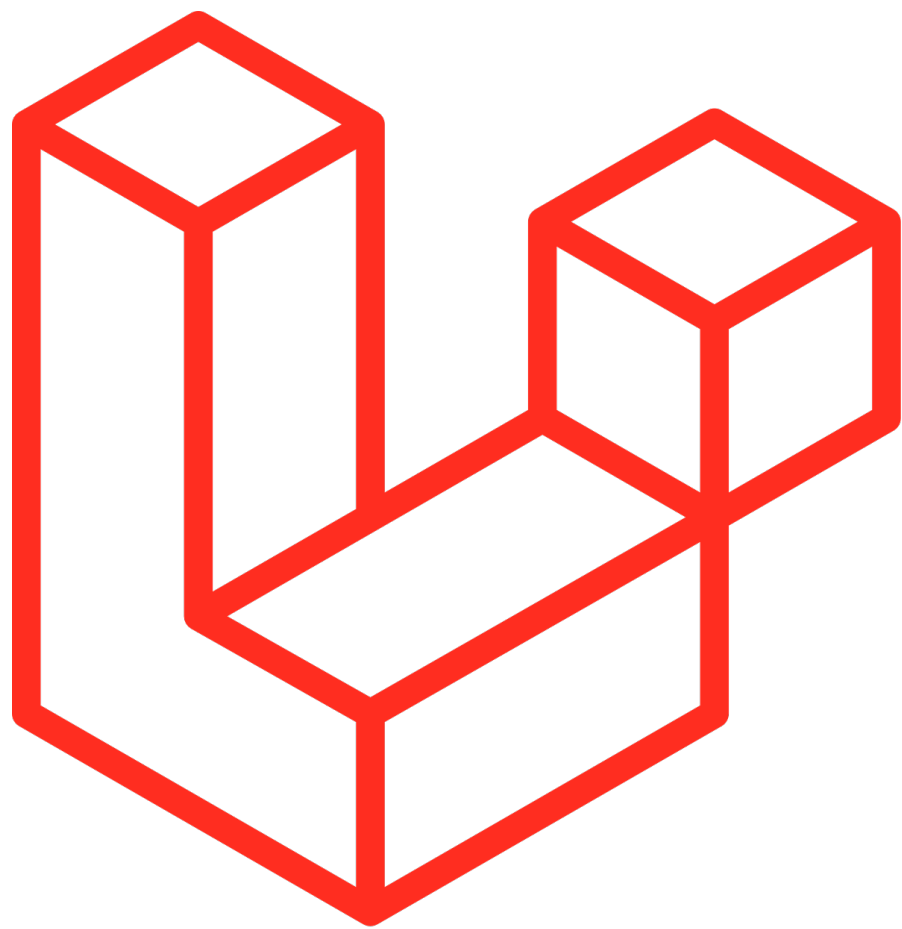
P H P

MVC



P H P

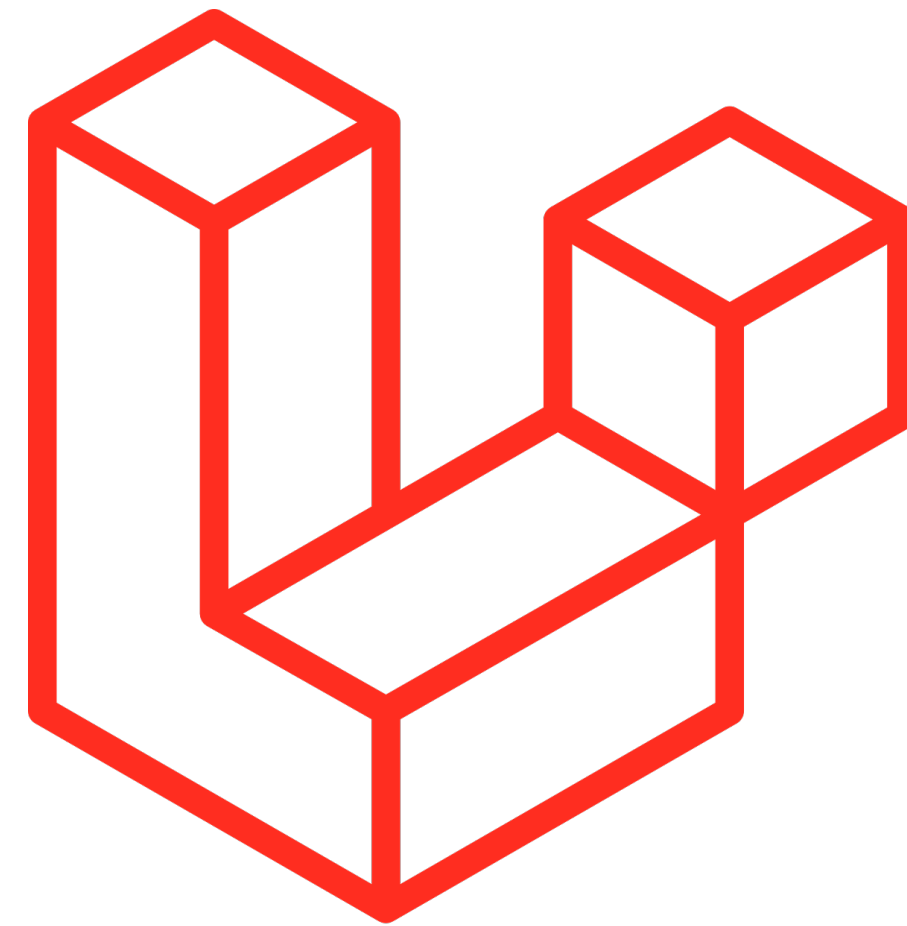
Instalação



- PHP \geq 7.2.5
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

P H P

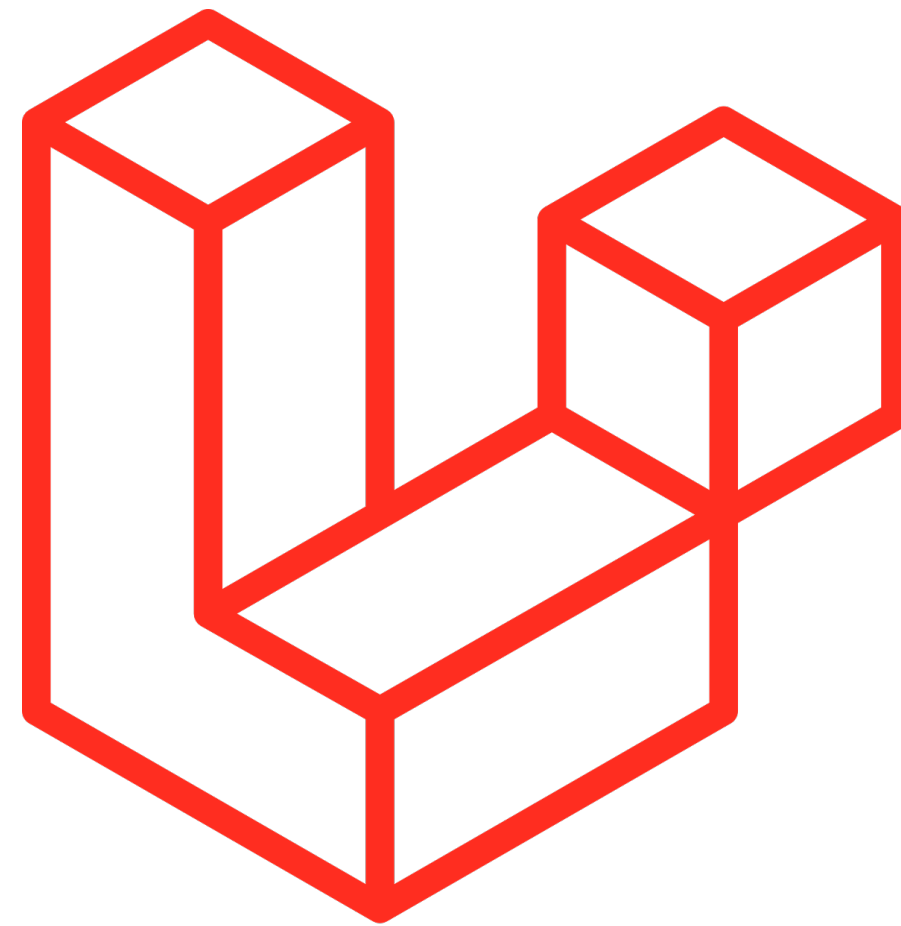
Instalação



```
composer global require laravel/installer
```

P H P

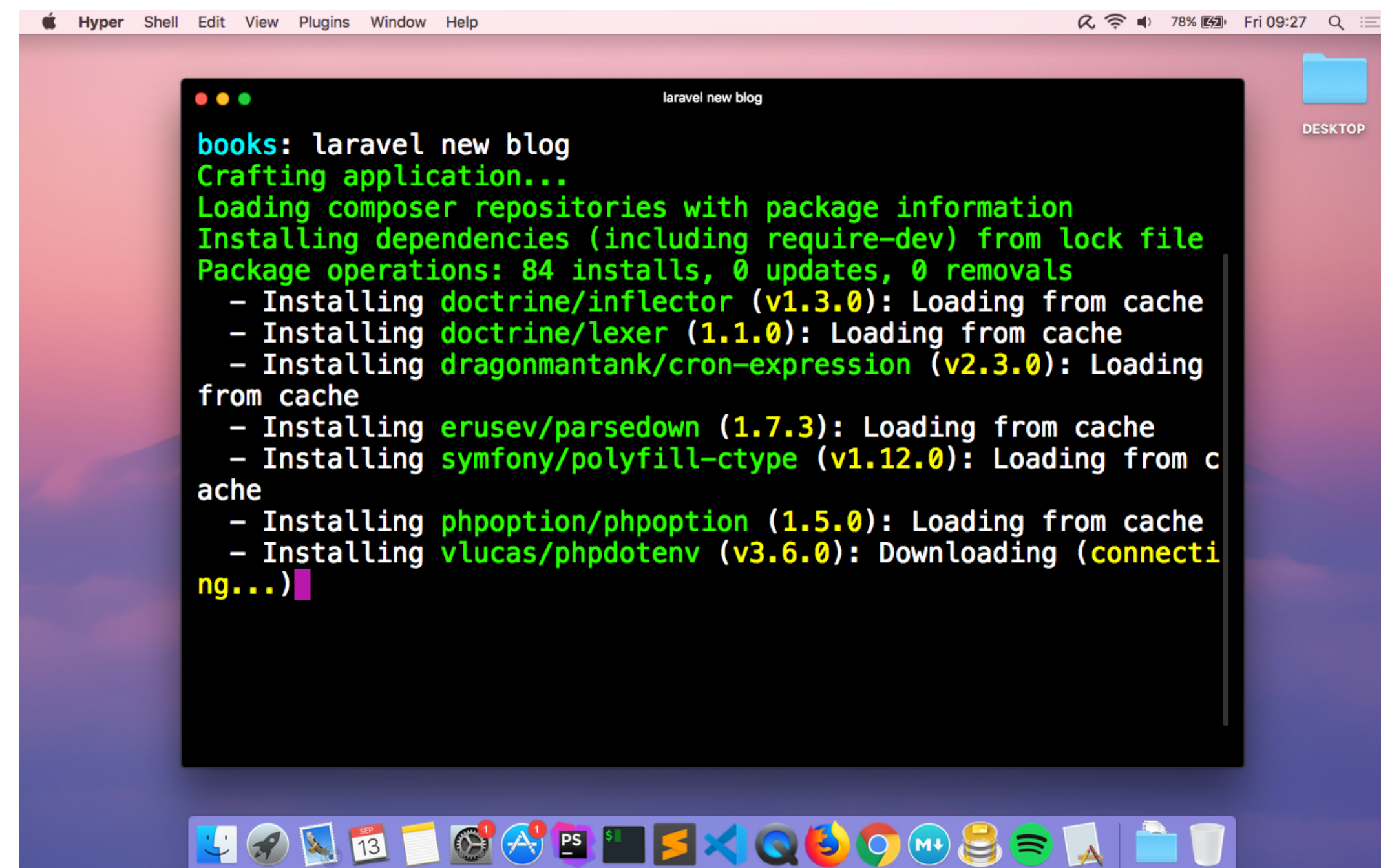
Instalação



```
composer create-project --prefer-dist laravel/laravel:^7.0 blog
```


P H P

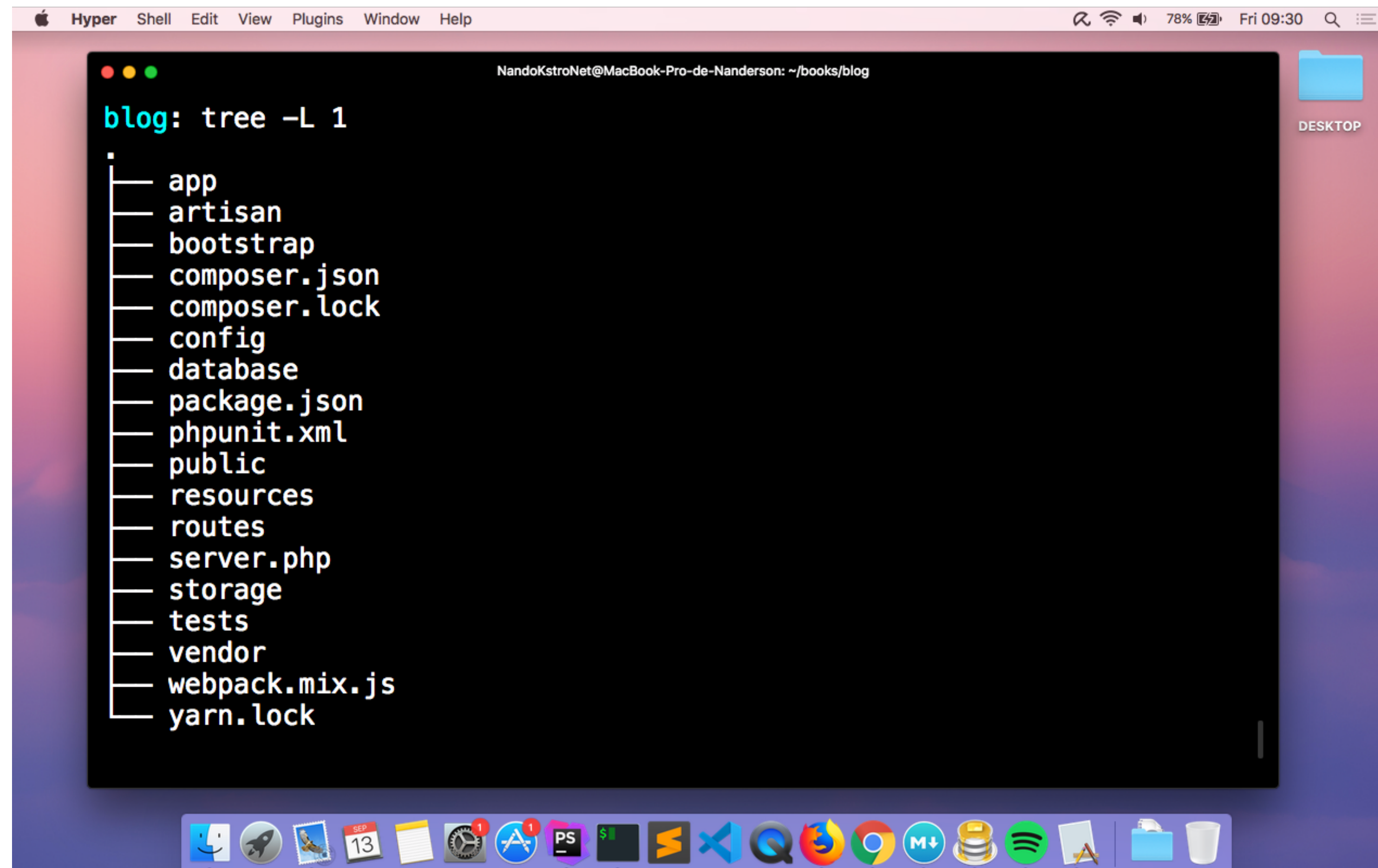
Instalação

A screenshot of a macOS desktop environment. The desktop background is a purple and blue gradient. In the center, a terminal window titled 'laravel new blog' is open. The terminal shows the command 'books: laravel new blog' and its output, which includes 'Crafting application...', 'Loading composer repositories with package information', 'Installing dependencies (including require-dev) from lock file', and 'Package operations: 84 installs, 0 updates, 0 removals'. A list of packages being installed is shown, including doctrine/inflector, doctrine/lexer, dragonmantank/cron-expression, erusev/parsedown, symfony/polyfill-ctype, phpoption/phpooption, and vlucas/phpdotenv. The desktop has a dock at the bottom with various application icons like Finder, Safari, Mail, and others. The top of the screen shows the macOS menu bar with the Apple logo, system status icons, and the time 'Fri 09:27'.

Após executar os comandos o instalador fará o download de todo o skeleton do Laravel e instalará as dependências do nosso projeto

P H P

Estrutura



The screenshot shows a macOS desktop environment. At the top is a menu bar with the Apple logo, the application name 'Hyper', and menus for 'Shell', 'Edit', 'View', 'Plugins', 'Window', and 'Help'. On the right side of the menu bar are system status icons: a refresh icon, Wi-Fi, speaker, 78% battery, and the date/time 'Fri 09:30'. A search icon and a hamburger menu icon are also present. The desktop background is a purple-to-pink gradient. On the right side, there is a 'DESKTOP' folder icon. At the bottom is a dock with various application icons: Finder, Launchpad, Photos, Calendar (showing 'SEP 13'), Notes, System Settings, App Store, Photoshop, a terminal icon, VS Code, a magnifying glass icon, Firefox, Google Chrome, a music icon, a database icon, Spotify, a document icon, a folder icon, and a trash can icon. A terminal window is open in the center, titled 'NandoKstroNet@MacBook-Pro-de-Nanderson: ~/books/blog'. The terminal shows the command 'blog: tree -L 1' and its output, which is a directory tree listing the project structure.

```
blog: tree -L 1
.
├── app
├── artisan
├── bootstrap
├── composer.json
├── composer.lock
├── config
├── database
├── package.json
├── phpunit.xml
├── public
├── resources
├── routes
├── server.php
├── storage
├── tests
├── vendor
├── webpack.mix.js
└── yarn.lock
```

P H P

Estrutura

app

A pasta app contem todo o conteúdo do nosso projeto como os models, controllers, serviços, providers, middlewares e outros. É nela que concentraremos diretamente nossos esforços durante a criação do nosso projeto.

config

Os arquivos de configurações do nosso projeto laravel encontram-se nesta pasta. Configurações de ligação a base de dados, onde estão os drivers para armazenamento de ficheiros, configurações de autenticação, mailers, serviços, sessions e outros.

resources

Nesta pasta temos algumas subpastas que são: views, js, lang & sass. Na sua maioria esta pasta guarda os assets referentes a views ou templates.

P H P

Estrutura

storage

Nesta pasta salvamos arquivos como sessions, caches, logs e também é utilizada para armazenar arquivos mediante upload em nosso projeto.

routes

Nesta pasta vamos encontrar os arquivos para o mapeamento das rotas de nosso projeto. Rotas estas que permitirão o nosso usuário acessar determinada url e ter o conteúdo processado e esperado. Mais a frente vamo conhecer melhor essas rotas mas as mesmas são divididas em rotas de api, as rotas padrões no arquivo web, temos ainda rotas para channels e console.

tests

Nesta pasta teremos as classes para teste de nossa aplicação. Testes Unitários, Funcionais e outros.

Estrutura

database

Aqui teremos os arquivos de migração de nossas tabelas, vamos conhecer eles mais a frente também, teremos os arquivos para os seeds e também as factories estes para criação de dados para popularmos nossas tabelas enquanto estamos desenvolvendo.

bootstrap

Na pasta bootstrap teremos os arquivos responsáveis por inicializar os participantes do framework Laravel, ecaminhando as coisas a serem executadas.

public

Esta é nossa pasta principal, a que fica exposta para a web e que contém nosso front controller. Por meio desta pasta é que recebemos nossas requisições, especificamente no index.php, e a partir daí que o laravel direciona as requisições e começa a executar as coisas.

vendor

A vendo como conhecemos, é onde ficam os pacotes de terceiros dentro de nossa aplicação mapeados pelo composer.

Estrutura

Temos ainda alguns arquivos na raiz do nosso projeto, como o `composer.json` e o `composer.lock` onde estão definidas as nossas dependências e as versões descarregadas respectivamente.

Temos também o `package.json` que contém algumas definições de dependências do frontend. Temos também os arquivos de configuração para o webpack, pacote responsável por criar os builds do frontend.

Temos ainda também o `server.php` que nos permite emular o `mod_rewrite` do apache.

Temos também o `phpunit.xml` que contém as configurações para nossa execução dos testes unitários, funcionais e etc em nossa aplicação.

Por último o arquivo `.env` que contém as variáveis de ambiente para cada configuração de nossa aplicação como os parâmetros para ligação a base de dados e também o `application key hash` único para nossa aplicação e outras configurações além das referidas.

P H P

Laravel: Artisan CLI

O Laravel possui uma interface de comandos ou command line interface (CLI) chamada de artisan. Ao utilizarmos a CLI podemos melhorar bastante nossa produtividade enquanto developer como por exemplo: Gerar models, controllers, gerar a interface de autenticação e muitas outras opções que conheceremos.

Para conhecer todos os comando disponíveis no Artisan, basta executar na raiz do seu projeto o seguinte comando:

```
php artisan
```

Veja o resultado, a lista de comandos e opções disponíveis no cli:

P H P

Laravel: Artisan CLI

```
NandoKstroNet@MacBook-Pro-de-Nanderson: ~/books/blog

blog: php artisan
Laravel Framework 6.0.3

Usage:
  command [options] [arguments]

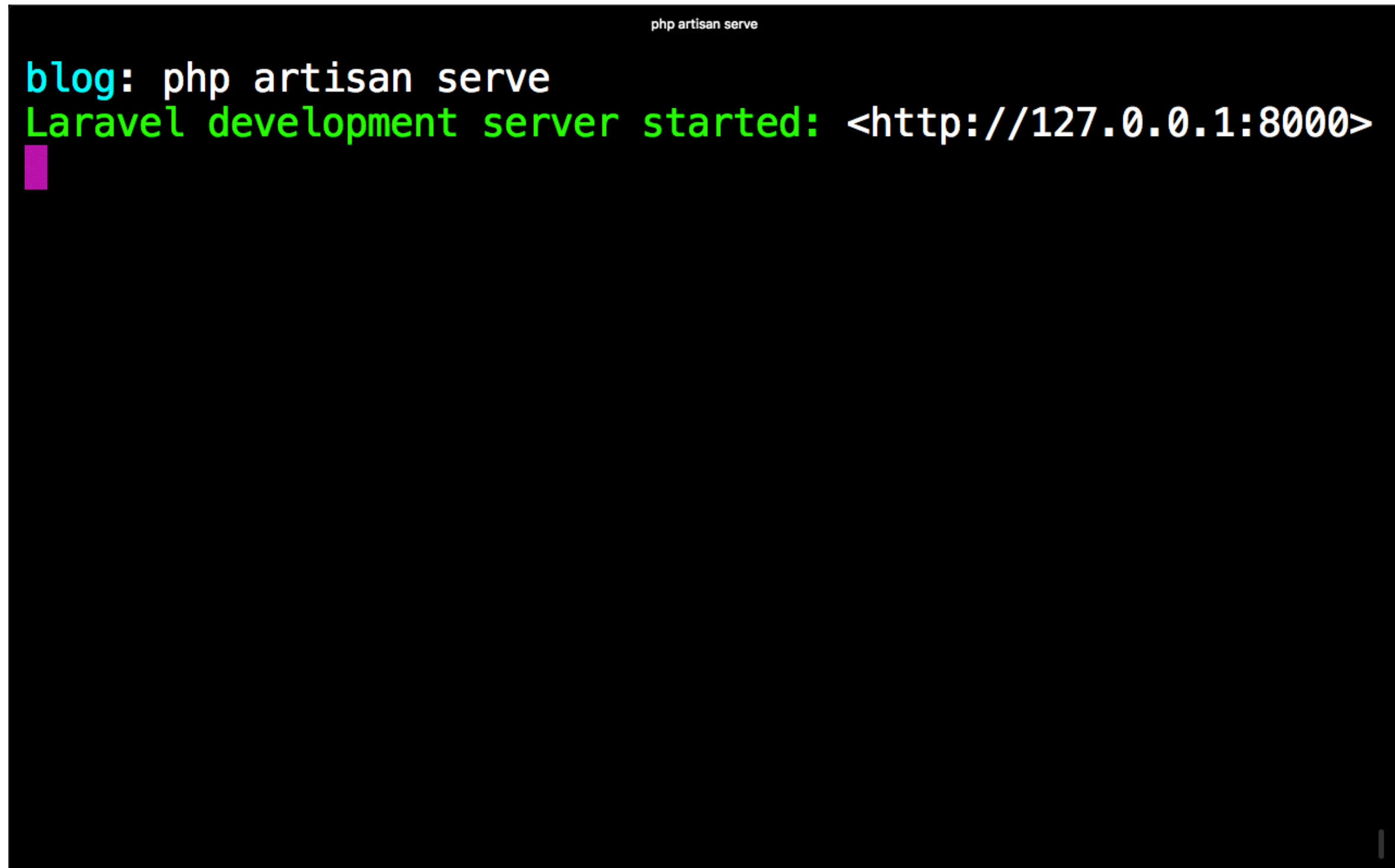
Options:
  -h, --help            Display this help message
  -q, --quiet            Do not output any message
  -V, --version          Display this application version
  --ansi                Force ANSI output
  --no-ansi             Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
  --env[=ENV]           The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:
  clear-compiled  Remove the compiled class file
  down           Put the application into maintenance mode
  env           Display the current framework environment
  help         Displays help for a command
  inspire      Display an inspiring quote
  list        Lists commands
  migrate      Run the database migrations
  optimize     Cache the framework bootstrap files
  preset       Swap the front-end scaffolding for the application
  serve       Serve the application on the PHP development server
  tinker       Interact with your application
  up          Bring the application out of maintenance mode
  auth
  auth:clear-resets  Flush expired password reset tokens
  cache
  cache:clear        Flush the application cache
  cache:forget       Remove an item from the cache
  cache:table        Create a migration for the cache database table
  config
  config:cache       Create a cache file for faster configuration loading
  config:clear       Remove the configuration cache file
  db
  db:seed            Seed the database with records
  db:wipe            Drop all tables, views, and types
  event
  event:cache        Discover and cache the application's events and listeners
  event:clear        Clear all cached events and listeners
  event:generate      Generate the missing events and listeners based on registration
```


P H P

Executando a Aplicação

```
php artisan serve
```



```
blog: php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

P H P

Laravel

Laravel

DOCS

LARACASTS

NEWS

BLOG

NOVA

FORGE

VAPOR

GITHUB

P H P

MVC - M

A camada do Model ou Modelo é a camada que possui as entidades que representam nossas tabelas na base de dados, podem conter classes que contêm regras de negócio específicas e até podem conter classes que realizam algum determinado serviço.

Dentro do Laravel os nossos models serão as classes que representam alguma tabela na base de dados com poderes de manipulação referentes a cada entidade. Podemos encontrar, por default, as classes model dentro da pasta app na raiz.

P H P

MVC - V

A camada de View ou visualização é a camada de interação com o utilizador. Onde nossos templates vão existir, com as interfaces do nosso sistema e as páginas de nossos sites. Nesta camada, também, não é recomendado colocar regra de negócios, pedidos a base de dados ou coisas deste tipo. Esta camada é exclusivamente para exibição de resultados e input de dados apenas, via formulários, além de interações Javascripts e outros processos já esperados para melhor a interação com o utilizador.

No Laravel as views encontram-se na pasta resources/views/. Nas views utilizaremos o template engine Blade.

P H P

MVC - C

A camada do Controller ou Controlador é a camada mais fina digamos assim, ele recebe o request e encaminha para o models em questão, caso necessário, e dada a resposta do model entrega o resultado para a view ou carrega diretamente um view caso não necessitemos de operações na camada do Model.

A ideia pro controller é que ele seja o mais simples possível, portanto, não é aconselhável adicionar regras e complexidade nos Controllers. Dentro do Laravel estes controllers encontram-se na pasta `app/Http/Controllers`.

P H P

HELLO WORLD

Aceda o projeto iniciado pelo seu terminal ou cmd no Windows. Na raiz do projeto execute o seguinte comando abaixo:

```
php artisan make:controller HelloWorldController
```

Ao executar o comando acima teremos o seguinte resultado, pro sucesso da criação do nosso primeiro controller, mostrando no nosso terminal ou cmd: Controller created successfully.

P H P

HELLO WORLD

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
class HelloWorldController extends Controller  
{  
    //  
}
```

P H P

HELLO WORLD

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HelloWorldController extends Controller
{
    public function index()
    {
        $helloWorld = 'Hello World';

        return view('hello_world.index', ['helloWorld' => $helloWorld]);
    }
}
```


P H P

HELLO WORLD

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HelloWorldController extends Controller
{
    public function index()
    {
        $helloWorld = 'Hello World';

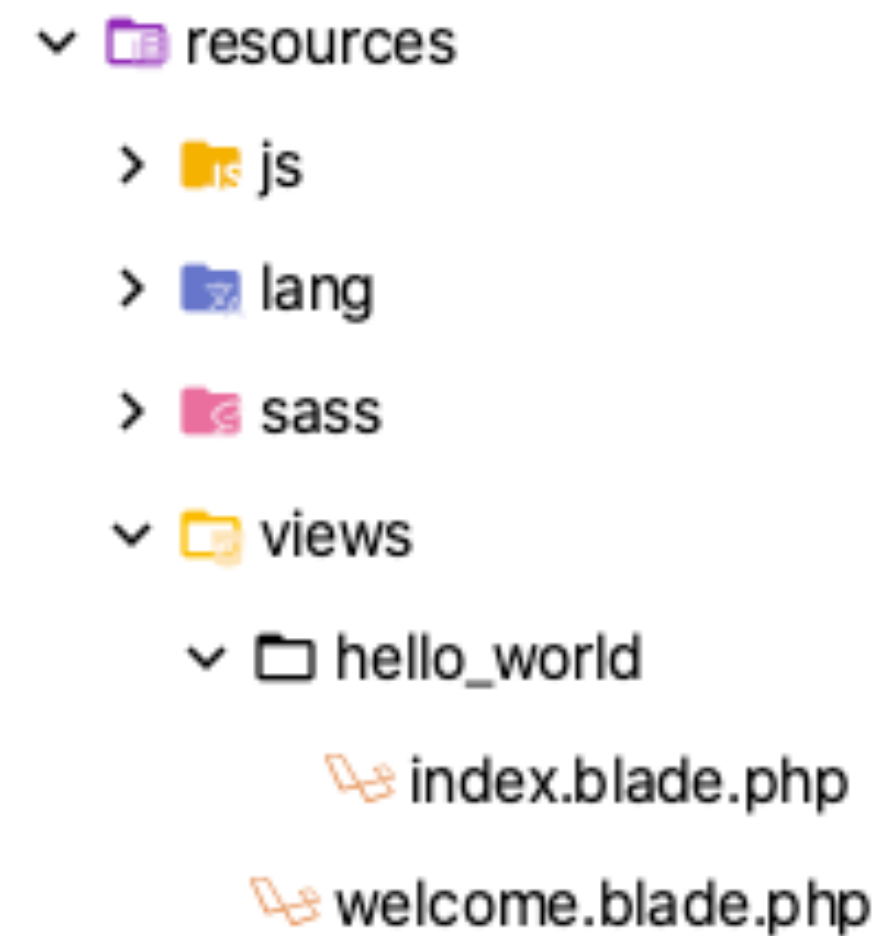
        return view('hello_world.index', ['helloWorld' => $helloWorld]);
    }
}
```

Do método acima temos um ponto bem interessante, o retorno da função helper chamada view que recebe como primeiro parâmetro a view desejada e o segundo parâmetro um array associativo com os valores a serem enviados para esta view.

P H P

HELLO WORLD

O hello_world é a pasta onde nosso index.blade.php estará, então, necessitamos de criar dentro da pasta resources/views uma pasta chamada hello_world e dentro desta pasta o nosso ficheiro index.blade.php chegando ao caminho completo e o arquivo: resources/views/hello_world/index.blade.php.



P H P

HELLO WORLD

```
<h1>{{ $helloWorld }} </h1>
```

P H P

HELLO WORLD

Agora, que já seguimos os dois passos referidos precisamos permitir o acesso e execução deste método, método index do nosso controller HelloWorldController, por parte dos nossos utilizadores e a definição de uma url para acesso.

Vamos criar uma rota que apontará para o método do nosso controller.

Acedemos ao ficheiro web.php que se encontra na pasta routes na raiz do projeto.

P H P

HELLO WORLD

```
<?php
```

```
use Illuminate\Support\Facades\Route;
```

```
/*
|-----
|  Web Routes
|-----
|
|  Here is where you can register web routes for your application. These
|  routes are loaded by the RouteServiceProvider within a group which
|  contains the "web" middleware group. Now create something great!
|
*/
```

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

P H P

HELLO WORLD

Este arquivo web.php recebe todas as rotas da nossa aplicação que define o encaminhamento dos pedidos do nosso projeto. Tudo que for UI ou User Interface(Interface do Utilizador) terá suas rotas definidas neste ficheiro.

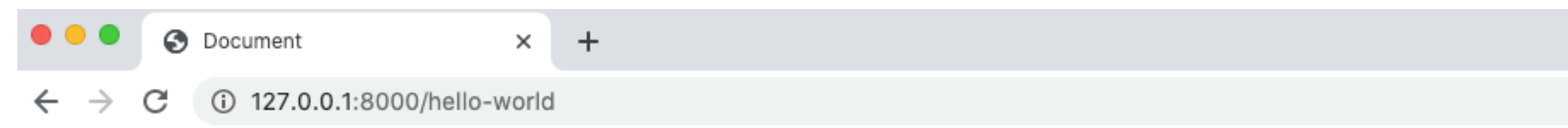
O template deste ficheiro possui uma definição de rota, a rota principal / que exibe a view welcome.blade.php que está lá dentro da pasta de views. Esta rota é a rota executado ao acedermos a página principal de uma aplicação Laravel recém instalada. Agora vamos definir nossa rota de hello world.

Adicione o código abaixo, após a definição da rota principal que já existe:

```
Route::get( 'hello-world' , 'HelloWorldController@index' );
```

P H P

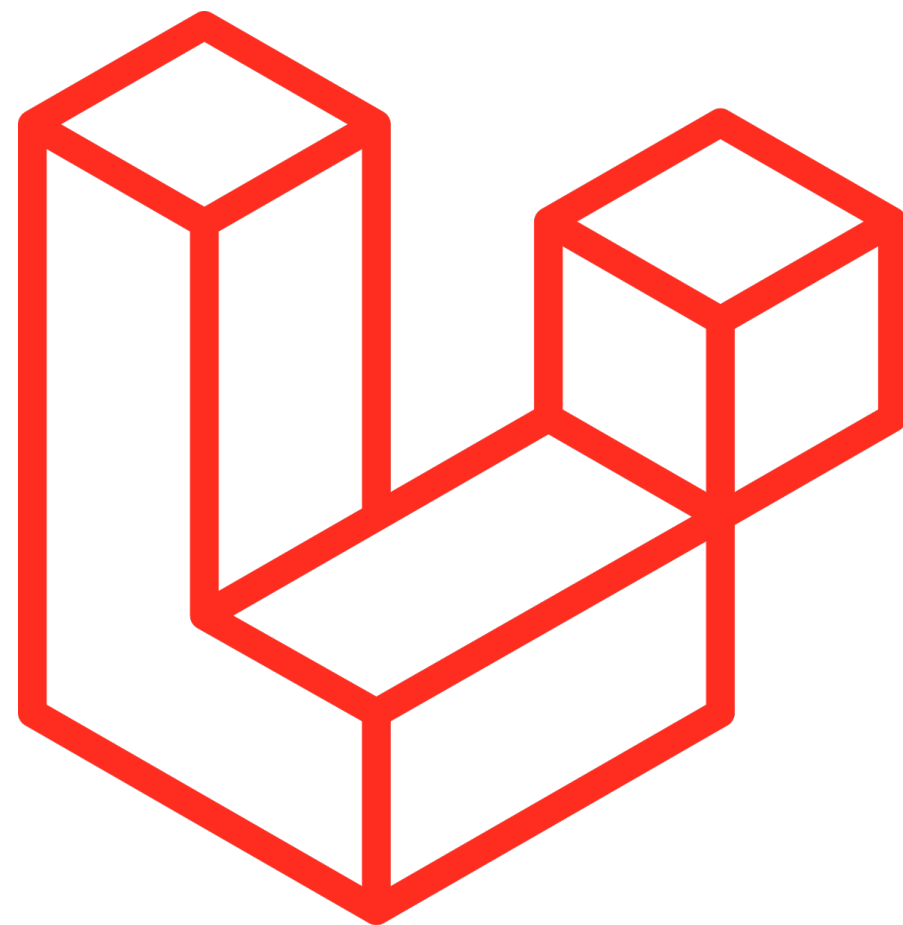
HELLO WORLD



Hello World

P H P

Rotas



As rotas em Laravel ajudam a termos uma visão mais clara e organizada sobre nossas urls. Como nós mapeamos nossas URLs dentro dos ficheiros de rotas fica mais fácil termos controlo do que será exposto e também é mais simples definirmos as rotas com a arquitetura desejada.

Dentro do Laravel temos os ficheiros de rotas bem separados, que nos ajudam a organizar melhor as rotas que dependendo da aplicação podem se tornar bem grande no ficheiro de rotas em questão.

O Laravel possui os seguintes arquivos de rotas: **web.php**, **api.php**, **channels.php** e **console.php**.

P H P

Rotas - Ficheiros

web.php

O arquivo web.php conterá as rotas da aplicação com as interfaces para o utilizador. Todas as rotas que têm esse fim deverão ser definidas neste ficheiro.

api.php

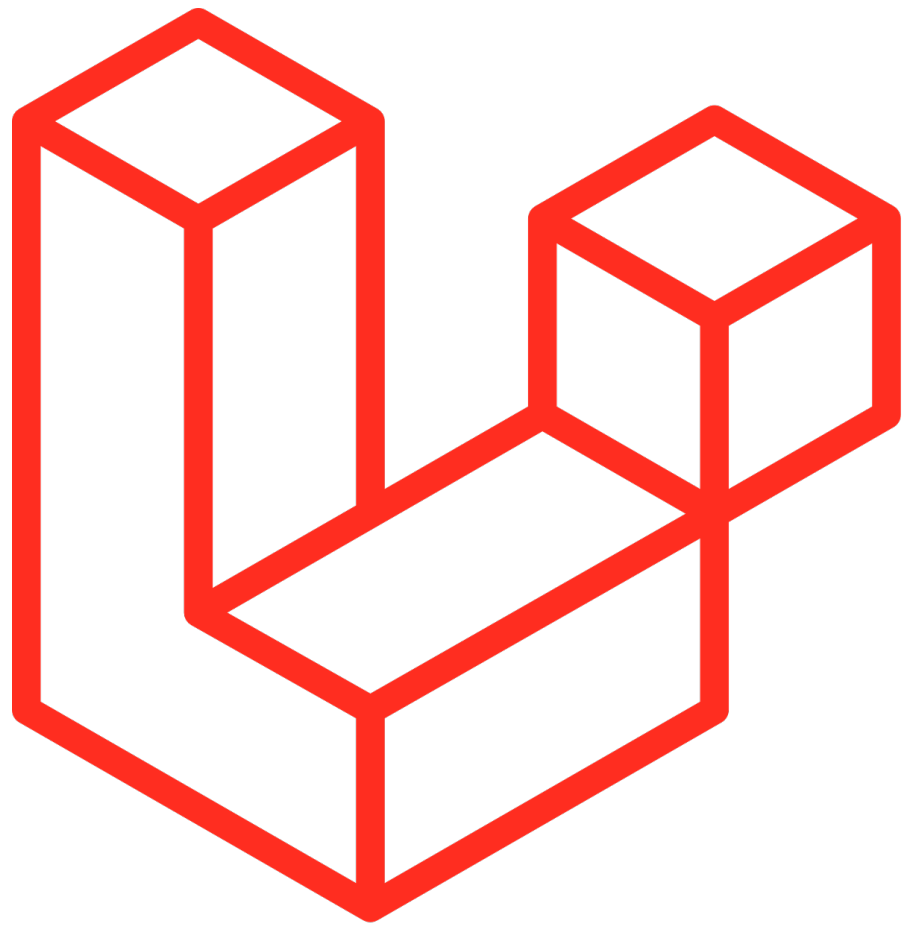
Ao trabalhar com APIs, são expostos endpoints para que outras aplicações possam consumir recursos, para isso é utilizado o ficheiro api.php.

channels.php

Se você for trabalhar com eventos de Broadcasting, Notificações e etc suas rotas deverão ser definidas neste arquivo.

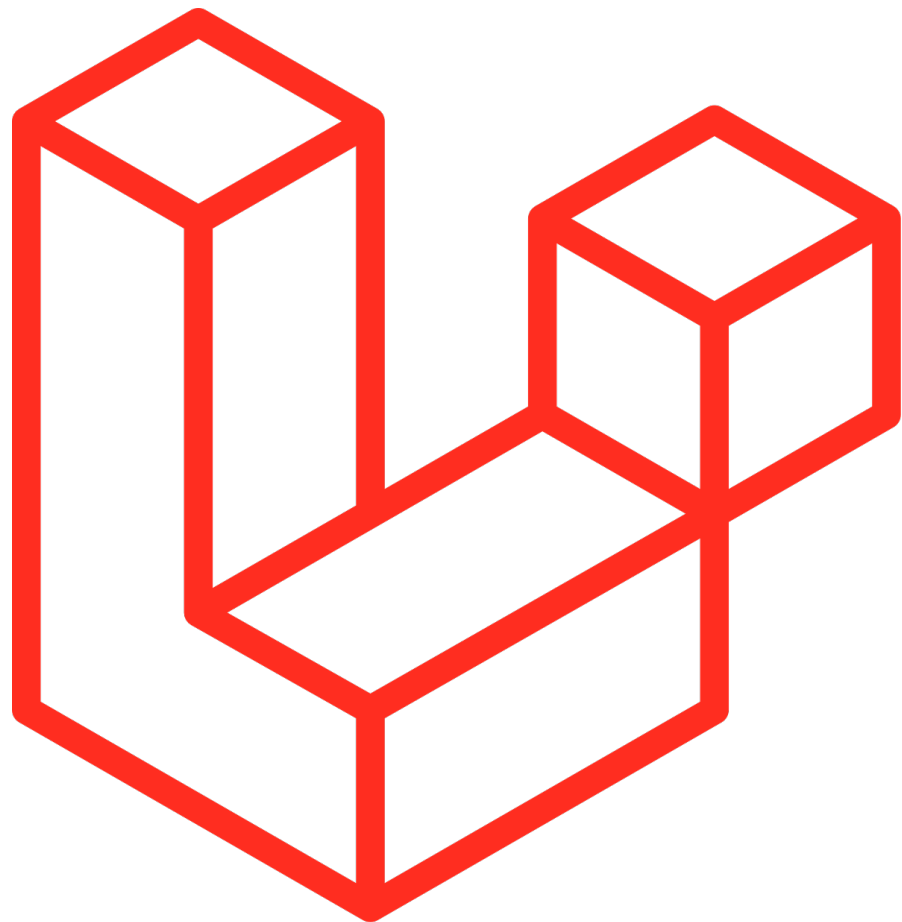
console.php

Arquivo para registro de comandos para o console e execução a partir do artisan.



P H P

Rotas



GET

O método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados.

HEAD

O método HEAD solicita uma resposta de forma idêntica ao método GET, porém sem conter o corpo da resposta.

POST

O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.

PUT

O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.

DELETE

O método DELETE remove um recurso específico.

CONNECT

O método CONNECT estabelece um túnel para o servidor identificado pelo recurso de destino.

OPTIONS

O método OPTIONS é usado para descrever as opções de comunicação com o recurso de destino.

TRACE

O método TRACE executa um teste de chamada *loop-back* junto com o caminho para o recurso de destino.

PATCH

O método PATCH é utilizado para aplicar modificações parciais em um recurso.

P H P

Rotas

SAFE METHODS NO ACTION ON SERVER	{	GET	HTTP/1.1 MUST IMPLEMENT THIS METHOD
		HEAD	INSPECT RESOURCE HEADERS
MESSAGE WITH BODY SEND DATA TO SERVER	{	PUT	DEPOSIT DATA ON SERVER — INVERSE OF GET
		POST	SEND INPUT DATA FOR PROCESSING
		PATCH	PARTIALLY MODIFY A RESOURCE
		TRACE	ECHO BACK RECEIVED MESSAGE
		OPTIONS	SERVER CAPABILITIES
		DELETE	DELETE A RESOURCE — NOT GUARANTEED

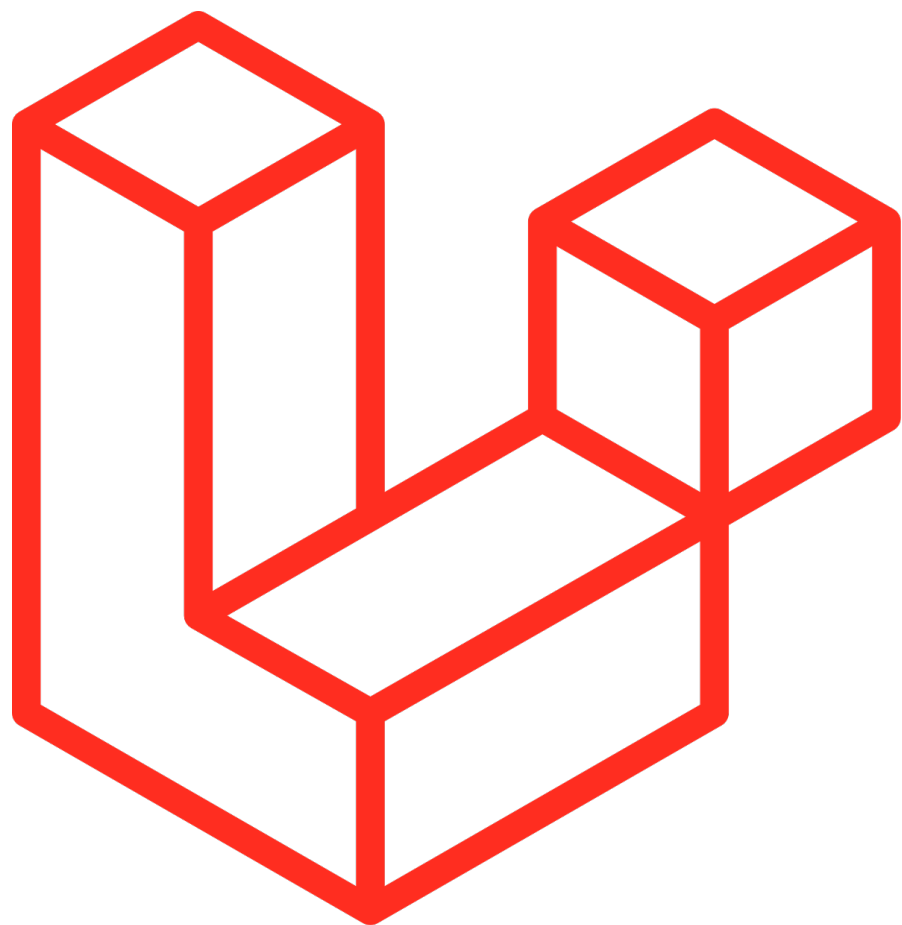
P H P

Rotas

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);  
Route::options($uri, $callback);
```

P H P

Rotas



Verb	URI	Typical Method Name	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

P H P

Rotas

```
<?php
```

```
use Illuminate\Support\Facades\Route;
```

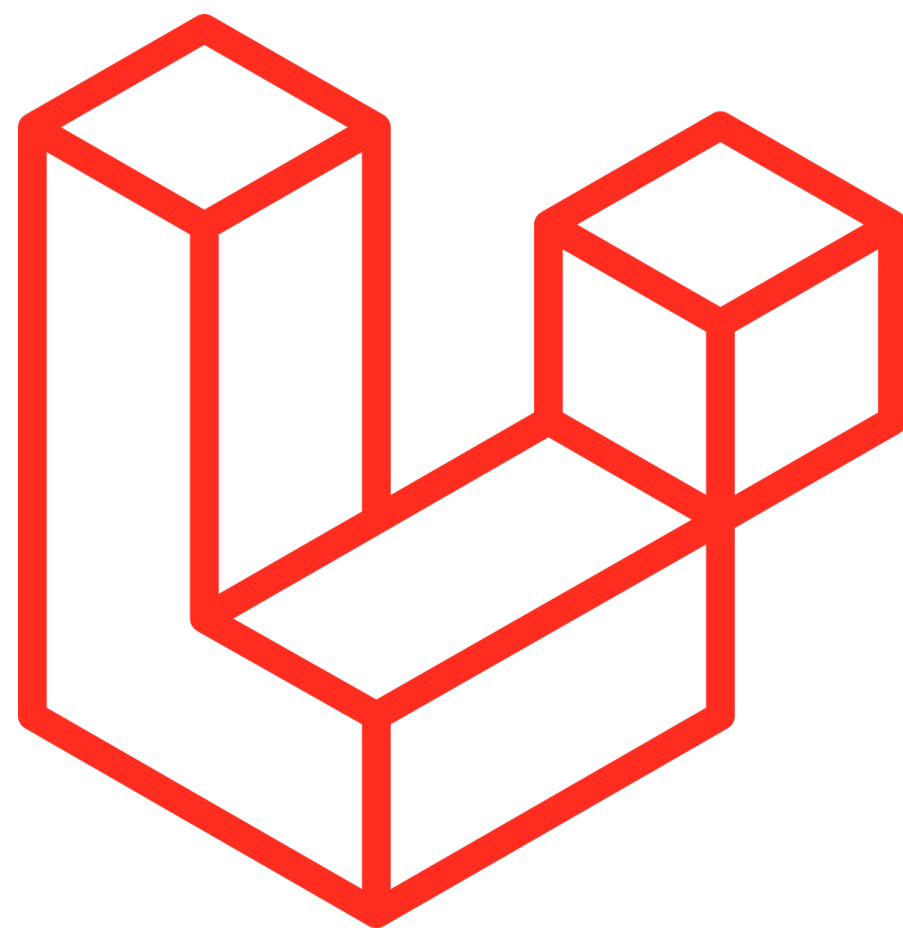
```
/*
|-----|
| Web Routes |
|-----|
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
```

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

```
Route::get('hello-world', 'HelloWorldController@index');
```


P H P

Rota match e Route any



Se precisarmos usar uma rota que responda a determinados tipos de verbos http, podemos usar o método match do Route. Como abaixo:

```
Route::match(['get', 'post'], 'posts/create', function() {  
    return 'Esta rota bate com o verbo GET e POST';  
});
```

Caso queira que uma rota responda para todos os verbos ao mesmo tempo, você pode usar o método any do Route:

```
Route::any('posts', function () {  
    return 'Esta rota serve todos os verbos HTTP mencionados  
    anteriormente';  
});
```

P H P

Rotas View

Em determinados momentos é necessário apenas renderizar determinadas views como resultado do pedido http. Para isso temos o método view do Route que nos permite definir uma rota, primeiro parâmetro, definir uma view, segundo parâmetro, e se preciso podemos passar algum valor para esta view sendo o terceiro parâmetro do método.

```
Route::view( 'welcome' , 'welcome' );  
Route::view( 'welcome' , 'folder.welcome' , [ 'name' => 'Taylor' ] );
```


P H P

Parâmetros em rotas

Por vezes é necessário utilizar dados da URL dentro da rota. Por exemplo, pode ser necessário guardar a identificação do utilizador. Para definir parâmetros de rota podemos consultar o seguinte exemplo:

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

```
Route::get( 'user/{id}', 'UserController@index' );
```

P H P

Parâmetros em rotas

O número de parâmetros de rota não possui limite

```
Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {  
    //  
});
```

P H P

Parâmetros em rotas opcionais

Ocasionalmente, pode ser necessário especificar um parâmetro de rota, mas tornar facultativa a presença desse parâmetro de rota. Pode ser feito colocando um ? depois do nome do parâmetro. Certifica de que a variável correspondente da rota possui um valor por defeito:

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

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

P H P

Grupos de rota

Os grupos de rota permitem-lhe partilhar atributos de rota, como intermediários ou espaços de nomes, num grande número de rotas sem necessidade de definir esses atributos em cada rota. Os atributos partilhados são especificados num formato de conjunto como primeiro parâmetro da rota::método do grupo.

```
Route::prefix('posts')->group(function () {  
    Route::get('/', 'PostController@index');  
    Route::get('/create', 'PostController@create');  
    Route::post('/save', 'PostController@save');  
});
```

P H P

Controllers

Podemos criar controllers com métodos para cada operação de CRUD e por meio de uma configuração de rota para termos também as rotas automáticas para cada um destes métodos.

Este tipo de controller chamamos de controller como recurso. Para criar um resource basta executar o seguinte comando:

```
php artisan make:controller UserController --resource
```

P H P

Controllers

O comando o parâmetro `--resource` criará um controller com os seguintes métodos:

- **index;**
- **create;**
- **store;**
- **show;**
- **edit;**
- **update;**
- **destroy;**

P H P

Controllers

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }
}
```

P H P

Controllers

```
/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    //
}

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    //
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    //
}
}
```


P H P

Controllers

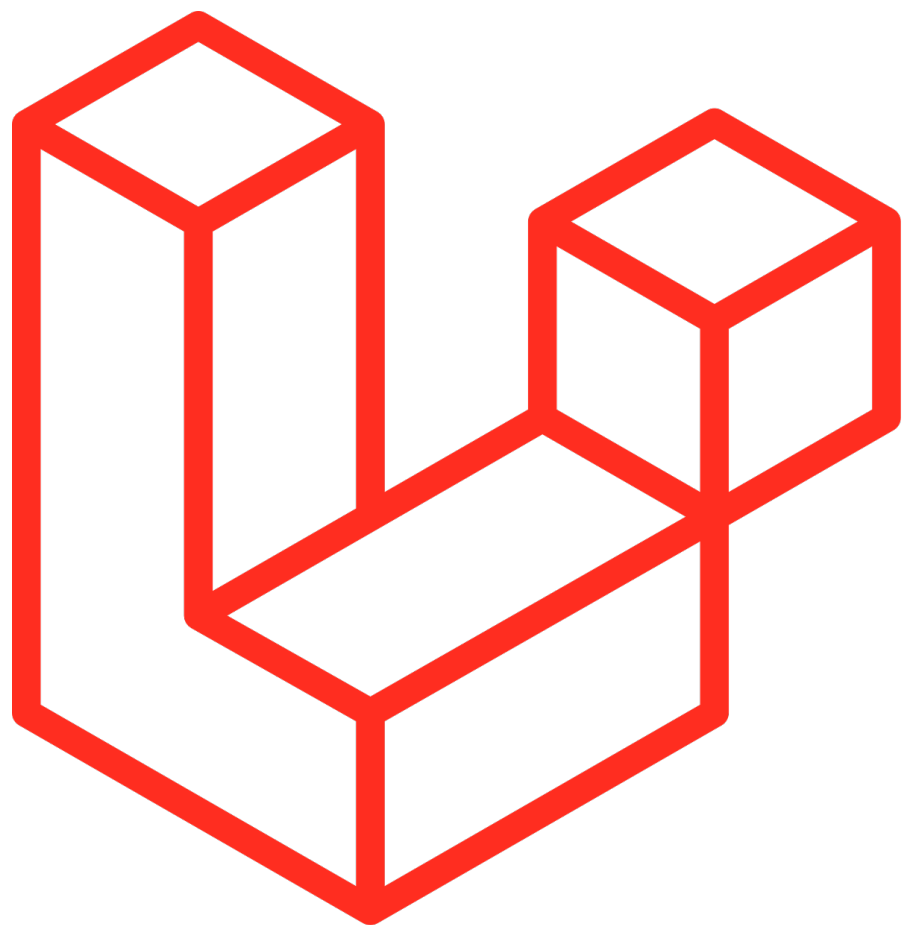
Agora com o controller como recurso criado, podemos expor rotas para cada um dos métodos acima. Neste caso, para facilitar temos um método dentro do Route chamado de resource que já expõe as rotas para cada um destes métodos simplificando ainda mais o desenvolvimento.

No ficheiro de rotas web basta adicionar a rota abaixo:

```
Route::resource( '/users', 'UserController' );
```

P H P

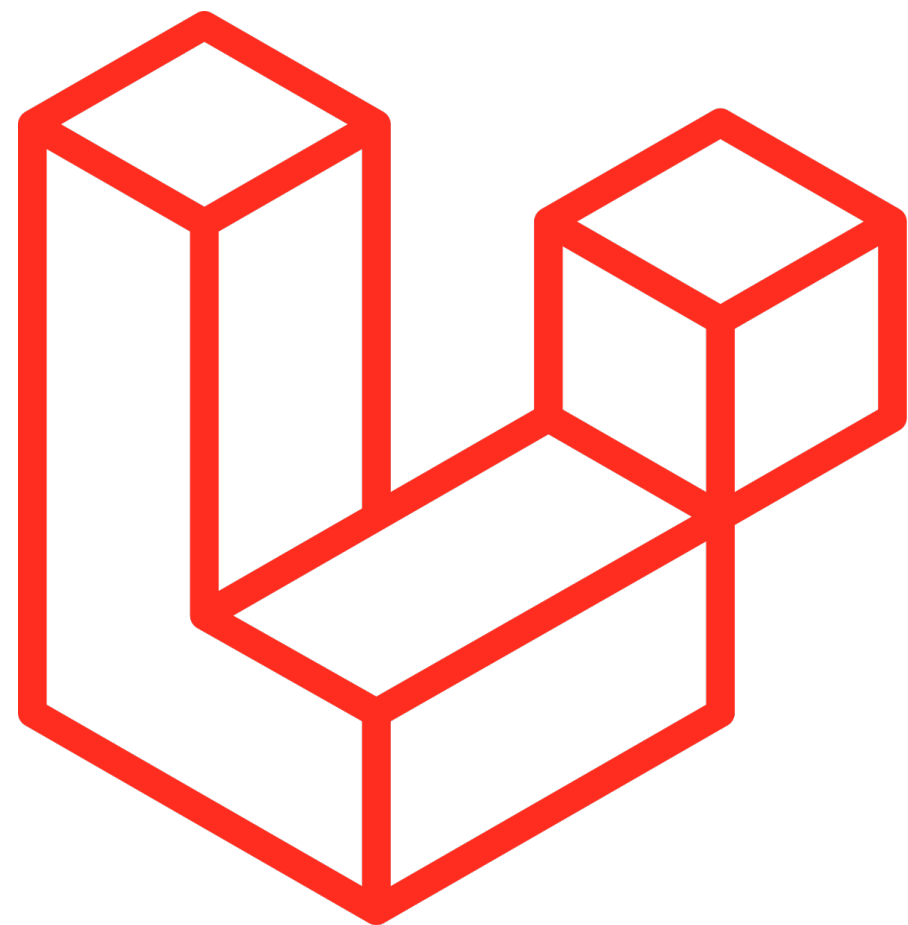
Rotas



Verb	URI	Typical Method Name	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

P H P

Definição de link URL



```
<a href="{{url( '/link' )}}">Link</a>
```

P H P

Blade template

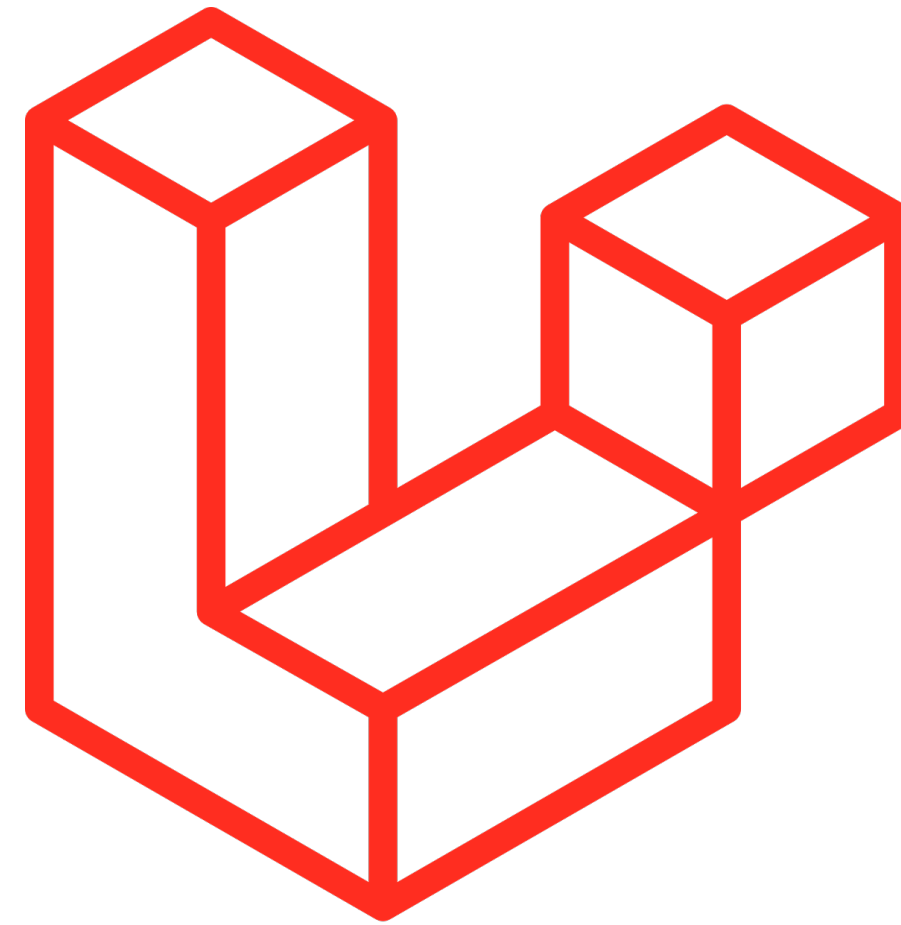
O blade é o motor moderado simples, mas poderoso, fornecido com Laravel. Ao contrário de outros motores de PHP populares, o Blade não te restringe de usar o código PHP simples apenas.

Documentação:

<https://laravel.com/docs/7.x/blade>

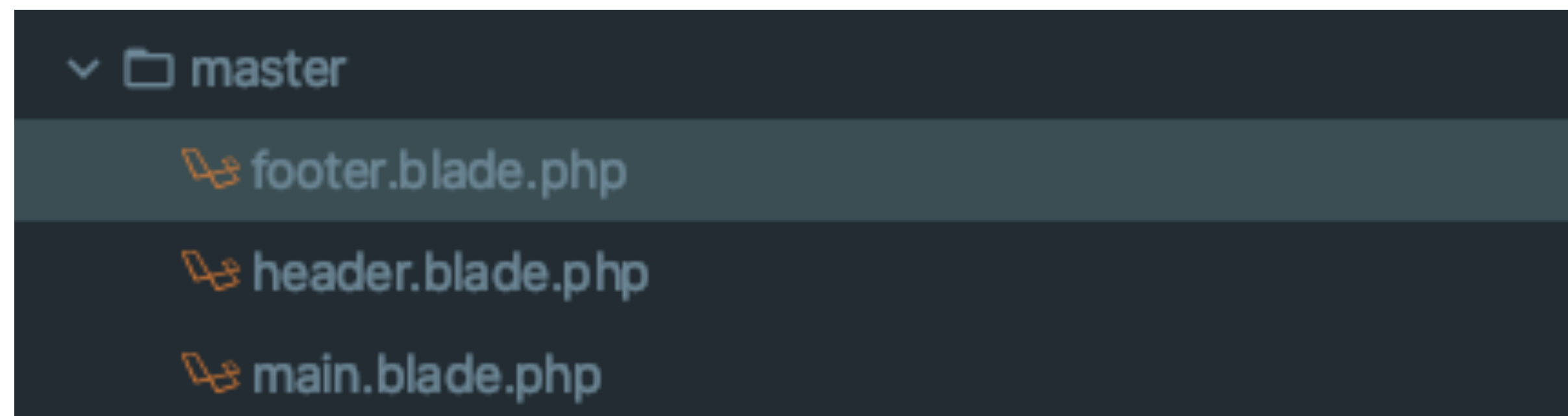
P H P

Master Pages



P H P

Master Pages



```
main.blade.php x footer.blade.php x header.blade.php x
1 <!doctype html>
2 <html lang="{{ app()->getLocale() }}">
3 <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no"/>
7     <meta name="csrf-token" content="{{ csrf_token() }}">
8     <title>Project </title>
9
10     {{-- STYLE SECTION --}}
11     <link href="{!! asset('css/app.css') !!}" media="all" rel="stylesheet" type="text/css" />
12     @yield('styles')
13     {{-- .STYLE SECTION --}}
14 </head>
15 <body>
16
17     {{-- Header --}}
18     @component('master.header')
19     @endcomponent
20     {{-- .Header --}}
21
22     {{-- content --}}
23     <main>
24         @yield('content')
25     </main>
26     {{-- .content --}}
27
28     @component('master.footer')
29     @endcomponent
30
31     <script src="{!! asset('js/app.js') !!}" type="text/javascript"></script>
32     @yield('scripts')
33     {{-- .SCRIPTS SECTION --}}
34 </body>
35 </html>
36 |
```

```
1  @extends('master.main')
2
3  @section('styles')
4      @stop
5
6  @section('scripts')
7      @stop
8
9  @section('content')
10      <div class="container">
11          <div class="row">
12              <h1>Hello World</h1>
13          </div>
14      </div>
15      @stop
16  |
```



```
1  @extends('master.main')
2
3  @section('styles')
4      @stop
5
6  @section('scripts')
7      @stop
8
9  @section('content')
10     @component('sliders.slider')
11     @endcomponent
12
13     @component('cards.cards')
14     @endcomponent
15
16     @component('content.article')
17     @endcomponent
18 @stop
19
```

P H P

Exercício Audi com master page

