
Laravel

Conhecendo a estrutura do Laravel

Vamos conhecer as pastas e os arquivos que fazem parte da estrutura do nosso projeto:

O Laravel possui algumas pastas base como a pasta `app`, `bootstrap`, `database`, `config`, `public`, `resources`, `storage`, `routes`, `tests` e a `vendor`. Vamos ver o que cada pasta destas representa ou armazena em sua estrutura.

```
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
```

app

A pasta app conterá todo o conteúdo do nosso projeto como os models, controllers, serviços, providers, middlewares e outros. 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 conexões com bando, onde estão os drivers para armazenamento de arquivos, configurações de autenticação, mailers, serviços, sessions e outros.

resources

Nesta pasta temos algumas subpastas que são: `views`, `js`, `lang` & `sass`. A priori esta pasta salva os assets referentes as nossas views e também nossas views ou templates.

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 vamos 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.

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 popular nossas tabelas enquanto estamos desenvolvendo.

bootstrap

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

public

Esta é nossa pasta principal, a que fica exposta para a web e que contêm 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.

Outros arquivos da raiz do projeto

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

Temos também o package.json que contêm 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 contêm as configurações para nossa execução dos testes unitários, funcionais e etc em nossa aplicação.

Deixe por último o arquivo .env que contêm as variáveis de ambiente para cada configuração de nossa aplicação como os parâmetros para conexão com o banco e também o aplication key hash único para nossa aplicação e outras configurações a mais além destas.

Laravel: Artisan CLI

O Laravel possui uma interface de comandos ou command line interface (CLI) chamada de artisan.

Por meio dela podemos melhorar bastante nossa produtividade enquanto desenvolvemos 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:

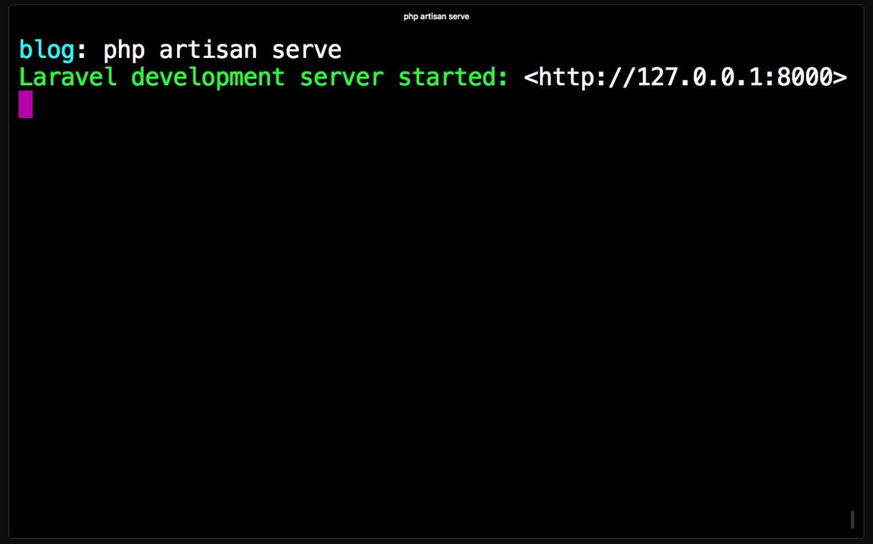
```
blog: php artisan
Laravel Framework 6.0.3

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 norm
al 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
```

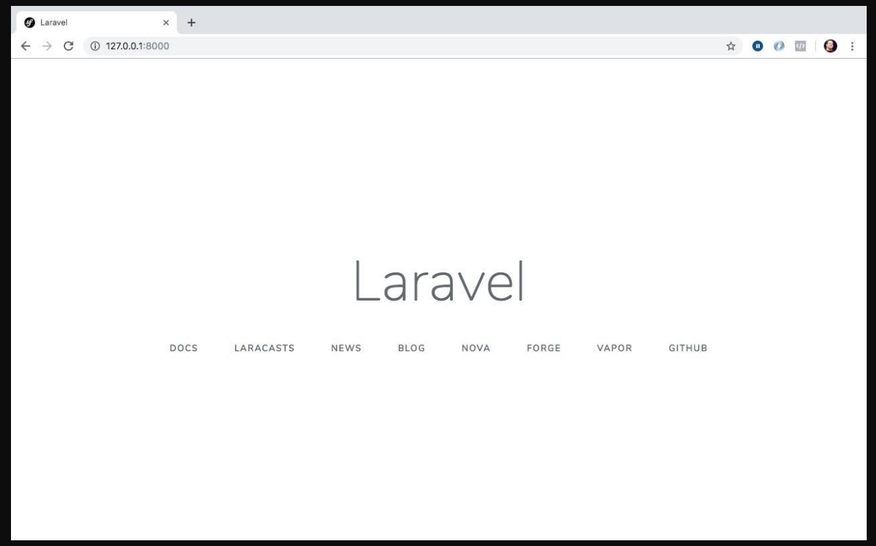
Executando a Aplicação

Para concluirmos nosso primeiro módulo, vamos iniciar nossa aplicação e testá-la em nosso browser.
Caso você esteja usando o laragon é só usar a pretty url, se estiver com outro servidor, acesse o seu projeto via terminal ou cmd no Windows e na raiz execute o comando abaixo:

```
php artisan serve
```

O comando acima levantará sua aplicação em seu ip local 127.0.0.1 e disponibilizará a porta 8000 para que você possa acessar sua aplicação no browser. Agora vamos acessar em nosso browser o seguinte link: http://127.0.0.1:8000:





Hello World com Laravel

Este módulo visa mostrar o Laravel de uma forma geral, o que definirá nosso fluxo de trabalho durante todo o desenvolvimento.

O projeto que iremos executar e que nos traz todo o aparato para entendermos cada parte de um framework fullstack será a criação de um blog com gerenciamento de posts e autores, além do sistema de comentários. Então vamos lá colocar a mão na massa e dar início de fato ao conhecimento por meio do nosso primeiro Hello World com o framework.

MVC

Antes de continuarmos precisamos conhecer um pouco do modelo base utilizado no mapeamento das classes dentro do nosso projeto Laravel, esse modelo ou padrão é o famoso MVC, ou, Model- View-Controller.

A maioria dos frameworks atuais utilizam esta estrutura para organização de seus componentes, além de termos um `Front Controller` que recebe as requisições enviadas para nossa aplicação e entrega/delega para quem vai resolver aquela requisição. Geralmente este `Front Controller` se encontra na pasta public(Directory Root da aplicação) no index.php dos frameworks, como o é no Laravel.

O modelo MVC possui três camadas base, as que comentei acima, o Model, o Controller e a View. Vamos entender o que é cada parte:

Model

A camada do Model ou Modelo é a camada que conterá nossas regras de negócio, geralmente possuem as entidades que representam nossas tabelas no banco de dados, podem conter classes que contêm regras de negócio especificas e até podem conter classes que realizam algum determinado serviço.

Dentro do Laravel nossos models serão as classes que representam alguma tabela no banco de dados com poderes de manipulação referentes a esse tratamento com o banco. Você pode encontrar, por default, as classes model dentro da pasta `app/Model`.

Controller

A camada do Controller ou Controlador é a camada mais fina digamos assim, ele recebe a requisição e demanda para o model 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, evite adicionar regras e complexidade em seus Controllers. Dentro do Laravel estes controllers encontram-se na pasta `app/Http/Controllers`.

View

A camada de View ou visualização é a camada de interação do usuário. Onde nossos templates vão existir, com as telas do nosso sistema e as páginas de nossos sites. Nesta camada, também, não é recomendado colocar regra de negócios, consultas ao banco 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 aproveitamente do usuário.

No Laravel nossas views encontram-se na pasta `resources/views/`. Em nossas views utilizaremos o template engine, que falarei sobre ele mais a frente, chamado de Blade.

Roteiro para nosso primeiro Hello World

Para criamos nosso hello world, precisaremos seguir os passos abaixo para este momento:

- Criar um controller para execução ao chamarmos nossa rota em questão;
- Criar uma view para envio da nossa string: `Hello World` a ser exibida como resultado do acesso;
- Criar nossa rota para chamada e acesso em nosso browser.

A priori passos bem simples e que vão nos dar um panorama inicial do framework para a partir daí prosseguirmos com os conceitos individualmente.

Iniciando Hello World

Acesse o projeto iniciado no módulo passado pelo seu terminal ou cmd no Windows. Na certeza de estar na raiz do seu projeto execute o seguinte comando abaixo:

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

Ao executar o comando acima teremos o seguinte resultado, para o sucesso da criação do nosso primeiro controller, exibido em nosso terminal ou cmd:

```
Controller created successfully.
```

Após isso teremos o nosso controller adicionado dentro da pasta do nosso projeto, especificamente: app/Http/Controllers e o arquivo HelloWorldController.php. Não esqueça de acessar o projeto em seu editor ou IDE de código de sua preferência.

O código do nosso controller encontra-se abaixo:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HelloWorldController extends Controller
{
    //
}
```

O resultado acima é a classe do nosso controller que estende do Controller base e já traz um import para nós, o nosso Request que também conheceremos ele no decorrer do nosso bimestre.

Vamos criar nosso primeiro método para execução posterior ao acessarmos nossa rota, ainda não criada. Abaixo segue o conteúdo do nosso primeiro método:

```php
public function index()
{
        $helloWorld = 'Hello World';

        return view('hello_world.index', compact('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.

Antes de entendermos como criaremos a view com base no primeiro parâmetro, preciso te falar que a função compact é uma função do PHP que pega as variáveis informadas e joga dentro de um array associativo sendo a chave o nome da variável e seu valor o valor contido na variável, foi o que fizemos para enviar $helloWorld para nossa view.

Agora como podemos criar nossa view de forma que o primeiro parâmetro da função helper view seja satisfeito? Vamos entender!

Temos o seguinte valor: hello_world.index.
A última parte da string apresentada após . será o nome da nossa view que deve respeitar o seguinte nome de arquivo, se depois do ponto tenho index precisarei criar uma view chamada index.blade.php, só que antes do ponto temos o valor hello_world que neste caso será a pasta onde nosso index.blade.php estará, então, no fim das contas precisaremos criar lá dentro da pasta resources/views uma pasta chamada hello_world e dentro desta pasta nosso arquivo index.blade.php chegando ao caminho completo e o arquivo: resources/views/hello_world/index.blade.php.

Se você quiser chamar uma view diretamente que esteja dentro da pasta de views, basta informar apenas o nome da views em questão. Se você tiver mais níveis em questão de pastas até chegar na view, é necessário informá-los até o arquivo da view em questão. Tanto o caminho base até a pasta views quanto a extensão .blade.php que o próprio Laravel adciona automáticamente para nós.

Com isso crie o arquivo index.blade.php e sua pasta hello_world dentro da pasta views. Com o seguinte conteúdo abaixo:

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

Acima temos, dentro do elemento h1, o nosso primeiro contato com o template engine Blade. Usamos acima a notação de print, abre {{ fecha }}, dentro da nossa view e pegamos a nossa variável $helloWorld vinda lá do nosso controller e exibimos seu valor dentro do elemento html.

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

Como faremos isso? Simples, no momento, vamos criar uma rota que apontará para o método do nosso controller!

Vamos lá que vou te mostrar!

Abra seu arquivo web.php que se encontra na pasta routes na raiz do projeto. Teremos o seguinte conteúdo:

```php
<?php

/*
|--------------------------------------------------------------------------
| 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');
});
```

Este arquivo web.php conterá todas as rotas da nossa aplicação que trabalham com a exibição de htmls com resultados em nosso projeto. Tudo que for UI ou User Interface(Interface do Usuário) terá suas rotas definidas neste arquivo.

De cara já vemos a primeira 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 executada ao acessarmos 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:

```
1  Route::get('hello-world', [HelloWorldController::class,'index']);
2
```

Acima temos nossa primeira rota definida, a rota escolhida foi `hello-world` que executará o método index do controller HelloWorldController, blz, como isto está definido? Vamos lá!

O segundo parâmetro da função `get` do `Route` é o executavél para esta rota, que pode ser uma função anônima como vimos na definição da rota principal já existente ou uma string que respeite [`Controller::class,método`] e foi como definimos [`HelloWorldController::class,index`].

O Laravel vai chamar o namespace base até o nosso controller e trabalhar em cima da string dada separando o controller do método, criando a instância deste controller e efetuando a chamada do método informado.

Com nossa rota definda, chegamos ao passo 3 e final do nosso roteiro e já podemos iniciar nosso webserver e temos o Hello World em nossa tela!

Rotas & Controllers

As rotas e controllers são integrantes bem importantes em nossas aplicações Laravel. Neste módulo vamos conhecer as opções que as rotas nos proporcionam e como podemos trabalhar com controllers conhecendo um pouco além do que vimos na aula passada.

Vamos as rotas então!

Rotas

As rotas em nossa aplicação Laravel nos ajudam a termos mais previsibilidade sobre nossas urls. Como nós mapeamos nossas URLs dentro dos arquivos de rotas fica mais fácil termos controle do que será exposto e também fica mais fácil de customizarmos as rotas como queremos.

Dentro do Laravel temos os arquivos de rotas bem separados, que nos ajuda a organizarmos melhor tais rotas que dependendo da aplicação podem se tornar bem grande no quesito de definições dentro do arquivo de rotas em questão.

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

web.php

O arquivo web.php conterá as rotas de sua aplicação com as interfaces para o usuário. Todas as rotas que têm esse fim deverão ser definidas neste arquivo.

api.php

Se você for trabalhar com APIs, expondo endpoints para que outras aplicações possam consumir seus recursos você deverá definir suas rotas com este fim no arquivo 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.

Definição de Rotas

Como vimos no último módulo, abordei duas formas de definição de rotas em nosso Hello World. Uma era a rota que já existia no arquivo web.php e a outra foi nossa definição de rota para nosso Hello World.

Vamos da uma revisada, uma relembrada:

```
1 Route::get('/', function () {
2     return view('welcome');
3 });
```

e

```
1 Route::get('hello-world', [HelloWorldController::class,'index']);
```

Acima temos duas formas de definição para rotas, com respeito ao que será executado. Lembrando que o primeiro parâmetro do método get é a rota em questão e o segundo parâmetro será um callable: uma função anônima ou uma string que respeite `Controller@método` que no fim das contas também virará um callable dada a instância do controller e a chamada do método deste controller.

Na rota inicial, que já tinhamos no arquivo web.php, vemos a utilização da função anônima e em nossa rota usamos a definição de chamada do controller e seu método diretamente.

Um primeiro ponto que podemos abordar sobre os métodos do `Route`, como vimos o `get` até o momento, é que teremos métodos respeitando os verbos http, como:

- GET;
- POST;
- PUT;
- PATCH;
- DELETE;
- OPTIONS;

Podendo utilizá-los da seguinte maneira:

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

Podemos usar os métodos conforme os verbos http mostrados, tendo sempre o primeiro parâmetro, a rota em si, e o segundo parâmetro o callable ou executável para esta rota ao ser acessada.

Route 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 bate com todos os verbos HTTP mencionados anteriormente';
});
```

View Routes

Em determinados momentos você vai precisar apenas renderizar determinadas views como resultado do acesso a sua rota. Para isso temos o método `view` do `Route` que nos permite setar 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.

Veja como é simples:

```
//Exibindo somente a view

Route::view('/bem-vindo', 'bemvindo');

//Exibindo a view e mandando parâmetros para ela

Route::view('/bem-vindo', 'bemvindo', ["name" => "Joãozinho"]);
```

Bem simples mesmo, ao acessarmos a rota `bem-vindo` em nosso browser carregaremos a view `bemvindo.blade.php` diretamente como resultado.

Parâmetros dinâmicos

Continuando, vamos conhecer um ponto bem importante sobre rotas que é a possibilidade de informamos parâmetros dinâmicos. Parâmetros estes que podem servir para identificar determinado recurso como uma postagem em um blog por exemplo.

Veja a rota definida abaixo:

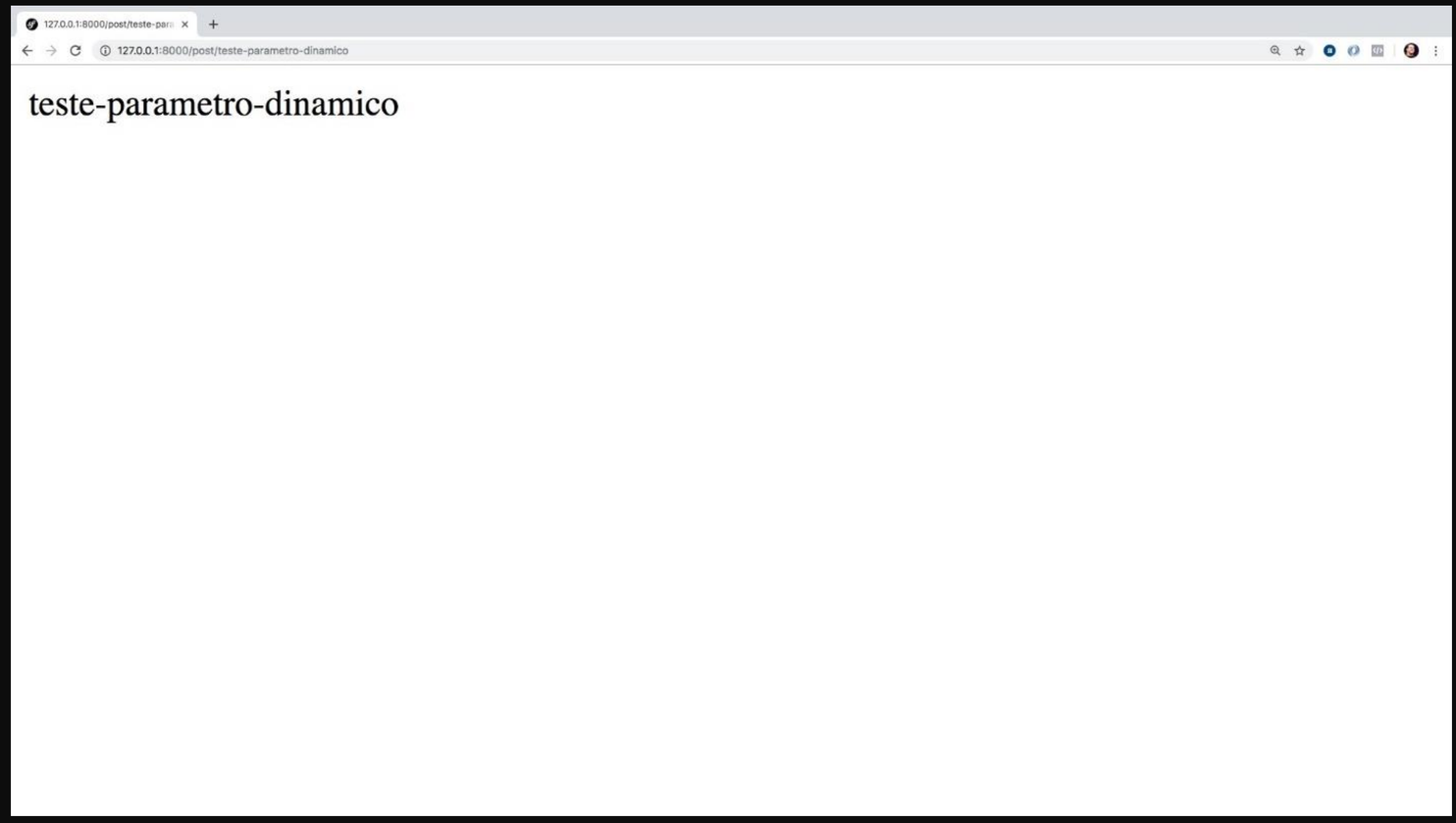
```
1  Route::get('/post/{slug}', function($slug) {
2      return $slug;
3  });
```

Temos a rota `/post/` após isso definimos um parâmetro dinâmico chamado `slug` dentro de chaves como é solicitado pelo componente de rotas. Em nossa função anônima passamos o parâmetro `$slug` que receberá o valor dinâmico e assim, poderemos utilizar ele dentro da nossa função.

Se eu estiver usando um controller e seu método, basta informarmos no método o parâmetro correspondente como informamos na função anônima e utilizarmos tranquilamente.

Com esta rota defininda em nosso arquivo web.php e nosso server levantado, podemos acessar em nosso browser a seguinte url: http://127.0.0.1:8000/post/teste-parametro-dinamico.

Resultado:



Como retornamos o parâmetro informado, teremos o valor dinâmico exibido em nosa tela como mostra a imagem acima.

Parâmetros Opcionais

Se você precisar definir parâmetros opcionais para sua rota em questão, basta adicionar a `?` antes do fechamento da última chave do parâmetro na definição da rota. Veja abaixo:

```
1  Route::get('/post/{slug?}', function($slug = null) {
2
3      return !is_null($slug) ? $slug : 'Comportamento sem a existência do param slug';
4
5  });
```

Agora nosso parâmetro `slug` é opcional ou seja não será necessário informá-lo na rota e isso nos abre precedente para validarmos ou exibirmos os resultados com base na não existência do parâmetro.

Mais um ponto bem útil nas rotas no Laravel.

Regex em parâmetros

Podemos validar o formato dos parâmetros aceitos em nossas rotas por meio de expressões regulares para um melhor controle. Para isso podemos utilizar o método `where` com este fim:

```
1  Route::get('/user/{id}', function($slug) {
2      return $slug;
3  })
4  ->where(['id' => '[0-9]+']);
```

O método `where` espera um array associativo, sendo a chave o nome do parâmetro e o valor a expressão regular (Regex) a ser validada no parâmetro informado.

Se você tiver mais de um parâmetro dinâmico e quiser informar uma regex para tal, basta ir adicionando no array, respeitando o pensamento chave sendo o parâmetro e valor sendo a experessão regular.

Apelido para rotas

Outro ponto bem importante em nossas rotas são seus apelidos. Mas para que servem? Até agora conhecemos o valor real ou nome real da rota mas podemos chamá-las por meio de seus apelidos também, isso nos ajuda quando precisamos, em um futuro, alterar o nome real das rotas.

Quando fazemos referência aos apelidos, podemos alterar tranquilamente o nome real da rota que o peso desta modificação não será tão impactante assim no quesito negativo. E como utilizar este apelido?

Vamos pegar nossa última rota do parâmetro dinâmico:

```
1 Route::get('/post/{slug}', function($slug) {
2     return $slug;
3 })
4 ->name('post.single');
```

Perceba a adição simples que fiz após o método `get` antes de fechar com o `;`. Chamei o método `name` que me permite adicionar um apelido para a rota em questão, neste caso agora posso chamar o apelido `post_single` toda vez que eu precisar usar a rota `post/{slug}`.

Em um link em nossa view ao invés de usarmos desta maneira:

```
1 <a href="/post/primeiro-post">Primeiro Post</a>
```

Vamos utilizar desta maneira:

```
1 <a href="{{route('post.single', ['slug' => 'primeiro-post'])}}">Primeiro Post</a>
```

Perceba acima que estamos em uma suposta view e utilizamos um método helper do Blade que é o `route` em nosso atributo href da âncora. O primeiro parâmetro do `route` é o apelido da rota e se a rota tiver parâmetros dinâmicos, que é o nosso caso, agente informa isso dentro de um array no segundo parâmetro do helper, informando o nome do parâmetro dinâmico e o seu valor.

O método route irá gerar a url correta, informando o parâmetro dinâmico no local correto. Com isso fica mais simples se precisarmos futuramente modificar o nome real da rota por esta questão, de chamarmos a rota pelo apelido ao invés de seu nome real.

Grupo de Rotas & Prefixo

Podemos definir determinadas configurações para um grupo especifico de rotas, para conhecermos o poder do group decidi mostrar ele aqui com a definição de um prefixo, método existente no Route também.

Vamos ao código abaixo:

```
Route::prefix('posts')->group(function(){

    Route::get('/', 'PostController@index')->name('posts.index');

    Route::get('/create', 'PostController@create')->name('posts.create');

    Route::post('/save', 'PostController@save')->name('posts.save');

});
```

Perceba no set de rotas acima que utilizei incialmente o método `prefix` e me utilizei do método `group` para definir esse prefixo para um grupo de rotas específico. Esse grupo de rotas é adicionado dentro do método `group` por meio de uma função anônima.

Agora, as rotas dentro do group serão prefixadas com o `posts`, ficando desta maneira:

- /posts/;
- /posts/create;
- /posts/save.

Duas rotas acessivéis via GET e uma acessível via POST.

O grupo nos permite esse tipo de configuração, quando precisamos organizar melhor determinadas configurações que se repetirão para mais de um set de rota. Isso melhora até a escrita dos nossos arquivos de rotas e definições.

Grupo de Rotas & Apelidos

Vamos melhorar ainda mais nosso set de rotas do momento passado. Podemos, também, definir um apelido base para um grupo de rotas, então vamos melhorar nosso grupo anterior.

Veja como ficou:

```
Route::prefix('posts')->name('posts.')->group(function(){

    Route::get('/', 'PostController@index')->name('index');

    Route::get('/create', 'PostController@create')->name('create');

    Route::post('/save', 'PostController@save')->name('save');

});
```

Perceba que agora isolei a parte `posts.` referente ao apelido das rotas após a definição do prefixo. Agora as rotas deste grupo além de receberem um prefixo, irão receber um apelido base que será concatenado com os apelidos de cada rota do grupo.

Os apelidos ficarão desta forma:

- posts.index;
- posts.create;
- posts.save.

Esses serão os apelidos das rotas gerados, o mesmo que seria anteriormente mas agora com o detalhe de termos organizado e isolado o que era repetido, ou seja, o `posts.`.