

BLADE: TEMPLATE ENGINE

Blade, Template Engine do Laravel

O Blade é o template engine default do Laravel e traz consigo diversas estruturas que simplificam muito nosso trabalho na criação de nossas views, estruturas condicionais, laços e também diversas diretivas que nos permitem criar controles de forma simples e rápida, por conta da pouca escrita para alcançar um resultado desejado.

O Blade ao longo do tempo veio e vem recebendo diversos incrementos e pretendo mostrar, na prática, estas possibilidades usando seu poder para construirmos nossas views.

Então vamos criar as views de nossa aplicação utilizando o poder do Blade!

Layouts

Vamos começar pelo layout, em nosso Hello World com Laravel nós tivemos um contato com o blade quando mandamos nossa mensagem (Hello World) para a view e exibimos ela com o interpolador

`{{ }}`, que nos permite retornar/exibir informações passadas a ele. Antes de entrarmos em pontos como este que comentei, vamos definir e organizar um template base usando o blade para melhor dispormos as views do nosso painel.

Primeiramente crie uma pasta chamada de `layouts` dentro da pasta de `views` e dentro da pasta `layouts` crie o arquivo `app.blade.php`. Adicione o seguinte conteúdo e logo após farei os comentários:

Layouts

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-\
7  scale=1.0, minimum-scale=1.0">
8      <meta http-equiv="X-UA-Compatible" content="ie=edge">
9      <title>Gerenciador de Posts</title>
10     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/\
11 css/bootstrap.min.css">
12 </head>
13 <body>
14     <nav class="navbar navbar-expand-lg navbar-light bg-light">
15         <a class="navbar-brand" href="/">Laravel 6 Blog</a>
16         <button class="navbar-toggler" type="button" data-toggle="collapse" data-tar\
17 get="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false" ari\
```

Layouts

```
18  a-label="Toggle navigation">
19      <span class="navbar-toggler-icon"></span>
20  </button>
21  <div class="collapse navbar-collapse" id="navbarNavDropdown">
22      <ul class="navbar-nav">
23          <li class="nav-item active">
24              <a class="nav-link" href="{{ route('posts.index') }}">Posts</a>
25          </li>
26      </ul>
27  </div>
28  </nav>
29  <div class="container">
30      @yield('content')
31  </div>
32  </body>
33  </html>
```

Layouts

Acima temos a definição do nosso layout base, aqui entramos no primeiro conceito do Blade, neste caso, na herança de templates. Se você perceber no layout, defini dentro do body na div.container uma diretiva chamada de `@yield` que me permite apontar onde os templates , que irão herdar deste layout, devem exibir seus conteúdos.

Por exemplo, vamos entender como isso acontece fazendo nossa view `create.blade.php` herdar do nosso layout `app.blade.php`. Veja as alterações que fiz na view `create.blade.php`:

```
1  @extends('layouts.app')
2
3  @section('content')
4      <form action="{{route('posts.store')}}" method="post">
5
6          @csrf
7
8          <div class="form-group">
9              <label>Titulo</label>
10             <input type="text" name="title" class="form-control" value="{{old('title\
11         ')}}">
12         </div>
13
14         <div class="form-group">
```

Layouts

```
15         <label>Descrição</label>
16         <input type="text" name="description" class="form-control" value="{{old(\
17 'description')}}">
18     </div>
19
20     <div class="form-group">
21         <label>Conteúdo</label>
22         <textarea name="content" id="" cols="30" rows="10" class="form-control">\
23 {{old('content')}}</textarea>
24     </div>
25
26     <div class="form-group">
27         <label>Slug</label>
28         <input type="text" name="slug" class="form-control" value="{{old('slug')\
29 }}">
30     </div>
31
32     <button class="btn btn-lg btn-success">Criar Postagem</button>
33 </form>
34 @endsection
```

Layouts

Aqui temos o conteúdo do formulário envolvido por uma diretiva chamada de `@section` que recebe o valor `content` e a definição de onde essa diretiva termina com o `@endsection`. O que isso quer dizer?!

A diretiva `@section` define o conteúdo que será substituído no layout principal, ou seja, quando eu acessar essa view ele vai herdar o que tem em `app.blade.php` e onde eu defini o `@yield('content')` será adicionado o conteúdo que temos na diretiva `@section`, no caso da view `create.blade.php` exibirá o conteúdo do formulário.

Mas onde está definido que o `create.blade.php` herda de `layout.blade.php`? Ele se encontra como sendo a primeira linha da nossa view, veja a definição que aponta de qual template `create.blade.php` herda por meio da diretiva `@extends` que recebe o layout pai da view em questão. Neste caso digo que a view `create.blade.php` herda de `app.blade.php` informando a diretiva `@extends` da seguinte maneira: `@extends('layouts.app')`.

Sendo que `layouts` é a pasta dentro de `views` e `app` o arquivo `app.blade.php`, onde sabemos que o Laravel irá incluir a extensão internamente e linkar o caminho completo até a pasta `layouts`.

Layouts

Altere também a view edit.blade.php, segue o conteúdo:

```
1  @extends('layouts.app')
2
3  @section('content')
4      <form action="{{route('posts.update', ['post' => $post->id])}}" method="post">
5
6          @csrf
7          @method("PUT")
8
9          <div class="form-group">
10              <label>Titulo</label>
11              <input type="text" name="title" class="form-control" value="{{ $post->title}}">
12
13          </div>
14
15          <div class="form-group">
16              <label>Descrição</label>
17              <input type="text" name="description" class="form-control" value="{{ $post->description}}">
18
19          </div>
20
21          <div class="form-group">
22              <label>Conteúdo</label>
```

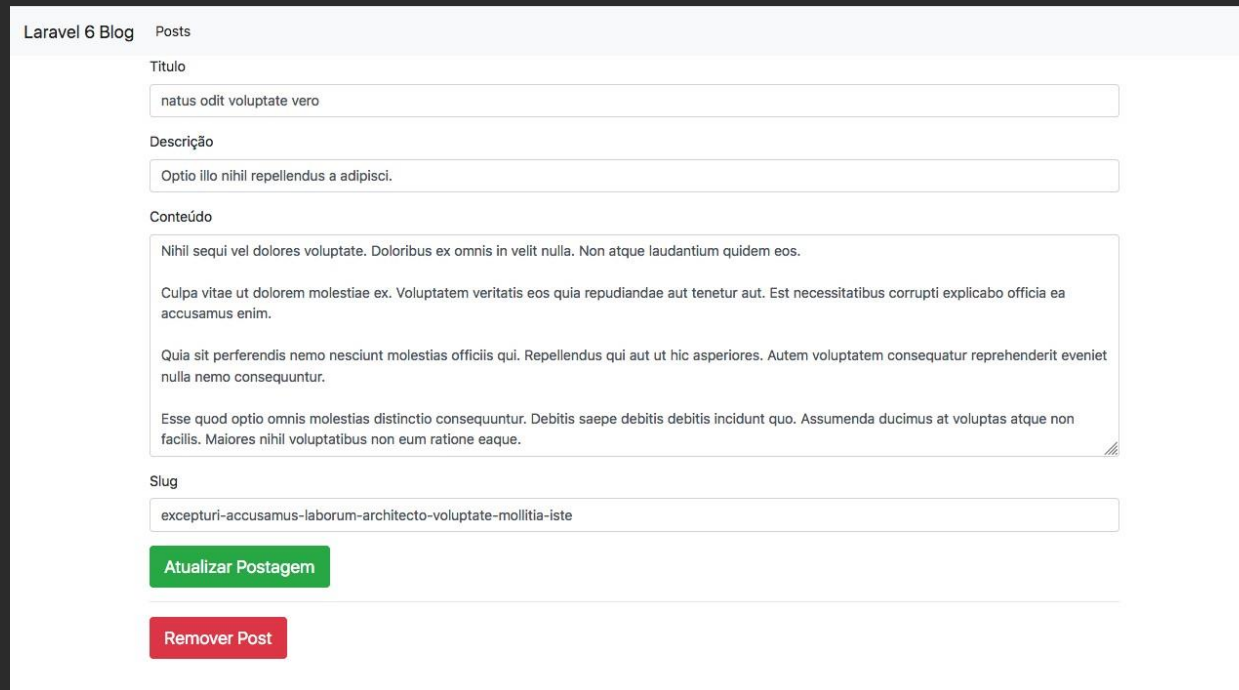
Layouts

```
23         <textarea name="content" id="" cols="30" rows="10" class="form-control">\
24     {{$post->content}}</textarea>
25     </div>
26
27     <div class="form-group">
28         <label>Slug</label>
29         <input type="text" name="slug" class="form-control" value="{{$post->slug\
30     }}">
31     </div>
32
33     <button class="btn btn-lg btn-success">Atualizar Postagem</button>
34 </form>
35 <hr>
36 <form action="{{$route('posts.destroy', ['post' => $post->id])}}" method="post">
37     @csrf
38     @method('DELETE')
39     <button type="submit" class="btn btn-lg btn-danger">Remover Post</button>
40 </form>
41 @endsection
```

Obs.: Fiz duas pequenas alterações além da definição da section e do extends. Adicionei a tag html hr entre o form de edição e o do botão de Remover Post e adicionei no button de remover posts as classes: btn btn-lg btn-danger.

Layouts

Veja o formulário na íntegra na imagem abaixo:



The image shows a web form for editing a blog post in a Laravel 6 application. The form is titled 'Laravel 6 Blog' and 'Posts'. It contains several input fields: 'Titulo' (Title) with the value 'natus odit voluptate vero', 'Descrição' (Description) with the value 'Optio illo nihil repellendus a adipisci.', and 'Conteúdo' (Content) with a large text area containing placeholder text. Below the content area is a 'Slug' field with the value 'excepturi-accusamus-laborum-architecto-voluptate-mollitia-iste'. At the bottom of the form are two buttons: 'Atualizar Postagem' (Update Post) in green and 'Remover Post' (Remove Post) in red.

Laravel 6 Blog Posts

Titulo

natus odit voluptate vero

Descrição

Optio illo nihil repellendus a adipisci.

Conteúdo

Nihil sequi vel dolores voluptate. Doloribus ex omnis in velit nulla. Non atque laudantium quidem eos.

Culpa vitae ut dolorem molestiae ex. Voluptatem veritatis eos quia repudiandae aut tenetur aut. Est necessitatibus corrupti explicabo officia ea accusamus enim.

Quia sit perferendis nemo nesciunt molestias officiis qui. Repellendus qui aut ut hic asperiores. Autem voluptatem consequatur reprehenderit eveniet nulla nemo consequuntur.

Esse quod optio omnis molestias distinctio consequuntur. Debitis saepe debitis debitis incidunt quo. Assumenda ducimus at voluptas atque non facilis. Maiores nihil voluptatibus non eum ratione eaque.

Slug

excepturi-accusamus-laborum-architecto-voluptate-mollitia-iste

Atualizar Postagem

Remover Post

Lembra que comentei que já tinha adicionado algumas classes do Bootstrap no Formulário, agora que linkamos o bootstrap.css lá no app.blade.php os estilos foram aplicados e nossa interface está mais amigável.

Agora vamos para a nossa listagem dos posts e conhecer mais possibilidades do Blade.

Laços de Repetição & Condicionais

Sabemos que para iterarmos em cima de uma coleção de dados precisamos usar laços de repetição, o Blade nos traz algumas possibilidades interessantes. A primeira delas é a possibilidade de utilização do `foreach`, como vemos abaixo:

```
1 @foreach($posts as $post)
2     <li>$post->title</li>
3 @endforeach
```

Desta maneira a iteração na coleção de posts vindas do banco de dados, será no mesmos moldes do `foreach` do PHP, e claro, no procesamento desta view essa diretiva se tornará um `foreach` nativo.

Mas aqui quero utilizar um `foreach` não muito convencional do dia a dia da forma que vamos escrever, mas é claro que com condicionais e combinando com os laços chegaríamos no mesmo resultado. Mas Nanderson, do que você está falando!

Por exemplo, poderíamos fazer um controle condicional pro caso de não existirem postagens na base e somente, se existirem, exibissemos a tabela com as postagens.

Laços de Repetição & Condicionais

Por exemplo, veja o trecho abaixo:

```
1  @if($posts)
2      @foreach($posts as $post)
3          <li>$post->title</li>
4      @endforeach
5  @else
6      <h2> Nenhuma postagem cadastrada!</h2>
7  @endif
```

Acima de cara te apresento o controle condicional ou como usar os se...senão (if...else) via diretivas do Blade. Primeiramente verificamos se o valor de \$posts é verdadeiro, se verdadeiro nós realizamos os loops, senão, exibimos uma mensagem padrão.

Laços de Repetição & Condicionais

Agora podemos melhorar ainda mais essa escrita usando blade, com a diretiva de loop chamada de @forelse. É ela que vamos utilizar então vamos ao conteúdo do nosso index.blade.php, que não existe ainda por isso crie o arquivo index.blade.php lá dentro da pasta das views das postagens.

Veja seu conteúdo abaixo:

```
1  @extends('layouts.app')
2
3  @section('content')
4      <div class="row">
5          <div class="col-sm-12">
6              <a href="{{route('posts.create')}}" class="btn btn-success float-right">\
7  Criar Postagem</a>
8              <h2>Postagens Blog</h2>
9              <div class="clearfix"></div>
10         </div>
11     </div>
12     <table class="table table-striped">
13         <thead>
14             <tr>
15                 <th>#</th>
16                 <th>Titulo</th>
17                 <th>Status</th>
18                 <th>Criado em</th>
```

Laços de Repetição & Condicionais

```
19         <th>Ações</th>
20     </tr>
21 </thead>
22 <tbody>
23 @forelse($posts as $post)
24     <tr>
25         <td>{{ $post->id }}</td>
26         <td>{{ $post->title }}</td>
27         <td>
28             @if($post->is_active)
29                 <span class="badge badge-success">Ativo</span>
30             @else
31                 <span class="badge badge-danger">Inativo</span>
32             @endif
33         </td>
34         <td>{{ date('d/m/Y H:i:s', strtotime($post->created_at)) }}</td>
35         <td>
36             <div class="btn-group">
37                 <a href="{{ route('posts.show', ['post' => $post->id]) }}" cla\
```

Laços de Repetição & Condicionais

```
38  ss="btn btn-sm btn-primary">
39      Editar
40  </a>
41  <form action="{{route('posts.destroy', ['post' => $post->id]\
42  }}" method="post">
43      @csrf
44      @method('DELETE')
45      <button type="submit" class="btn btn-sm btn-danger">Remo\
46  ver</button>
47  </form>
48  </div>
49  </td>
50  </tr>
51  @empty
52  <tr>
53      <td colspan="4">Nada encontrado!</td>
54  </tr>
55  @endforelse
56  </tbody>
57  </table>
58  @endsection
```


Laços de Repetição & Condicionais

Não esqueça de substituir, lá no controller PostController, no método index, a linha do dd pela linha abaixo:

```
1 return view('posts.index', compact('posts'));
```

Vamos aos pontos da view index.blade.php. Perceba que aqui usei a diretiva:

```
1 @forelse($posts as $post)
2     ...
3 @empty
4     ...
5 @endforelse
```

Para iterar as postagens vindas do controller, neste caso o @forelse fará o seguinte: Se não houverem postagens na base ele cairá na execução do bloco @empty, onde adicionamos um tr com td e a mensagem Nada Encontrado!.

Laços de Repetição & Condicionais

Veja o bloco do @empty abaixo:

```
1 @empty
2     <tr>
3         <td colspan="4">Nada encontrado!</td>
4     </tr>
5 @endforelse
```

Existindo posts, que é o nosso caso, teremos a exibição da tabela com as postagens. Vamos analisar o bloco do @forelse.

Veja somente ele abaixo:

```
1 @forelse($posts as $post)
2     <tr>
3         <td>{{ $post->id }}</td>
4         <td>{{ $post->title }}</td>
5         <td>
6             @if($post->is_active)
7                 <span class="badge badge-success">Ativo</span>
8             @else
9                 <span class="badge badge-danger">Inativo</span>
```

Laços de Repetição & Condicionais

```
10         @endif
11     </td>
12     <td>{{date('d/m/Y H:i:s', strtotime($post->created_at))}}</td>
13     <td>
14         <div class="btn-group">
15             <a href="{{route('posts.show', ['post' => $post->id])}}" cla\
16 ss="btn btn-sm btn-primary">
17                 Editar
18             </a>
19
19             <form action="{{route('posts.destroy', ['post' => $post->id]\
20 }}" method="post">
21                 @csrf
22                 @method('DELETE')
23                 <button type="submit" class="btn btn-sm btn-danger">Remo\
24 ver</button>
25             </form>
26         </div>
27     </td>
28 </tr>
29 ...
```

Laços de Repetição & Condicionais

O que vale destacar aqui é a utilização da diretiva de controle condicional para exibição da mensagem Ativo ou Inativo dentro de um badge provido pelo Bootstrap.css, usei também, no campo da data de criação da postagem, a função date do PHP para formatação.

Temos ainda, para a coluna de Ações, o botão de edição onde linkamos a rota de edição usando o apelido dela por meio da função helper route e passando o id dinâmico do post como segundo parâmetro.

Sobre o botão da remoção do post, apenas peguei o form que tínhamos criado para remoção da postagem no último capítulo alterando apenas a classe que estava btn-lg(para botões grandes) e coloquei btn-sm(para botões pequenos), além de agrupar os botões de edição e da remoção usando a div com a classe do bootstrap btn-group.

Laços de Repetição & Condicionais

Gerenciador de Posts

127.0.0.1:8000/admin/posts

Laravel 6 Blog Posts

Postagens Blog

Criar Postagem

#	Título	Status	Criado em	Ações
2	molestias commodi recusandae cum	Ativo	23/09/2019 20:34:19	Editar Remover
3	natus odit voluptate vero	Inativo	23/09/2019 20:34:19	Editar Remover
4	laboriosam molestiae sequi nihil	Inativo	23/09/2019 20:34:19	Editar Remover
5	dignissimos et rerum aut	Ativo	23/09/2019 20:34:19	Editar Remover
6	eos odit earum minima	Inativo	23/09/2019 20:34:19	Editar Remover
7	eos iste rem occaecati	Ativo	23/09/2019 20:34:19	Editar Remover
8	autem non est eaque	Ativo	23/09/2019 20:34:19	Editar Remover
9	architecto nesciunt consequuntur laborum	Ativo	23/09/2019 20:34:19	Editar Remover
10	repellat suscipit sed eaque	Ativo	23/09/2019 20:34:19	Editar Remover
11	occaecati et quia minima	Inativo	23/09/2019 20:34:19	Editar Remover
12	mollitia sint libero quia	Inativo	23/09/2019 20:34:19	Editar Remover
13	assumenda quo magnam mollitia	Ativo	23/09/2019 20:34:19	Editar Remover
14	aut omnis officiis fuga	Inativo	23/09/2019 20:34:19	Editar Remover
15	quidem repellat facere nesciunt	Inativo	23/09/2019 20:34:19	Editar Remover
16	aut consequatur consequuntur et	Inativo	23/09/2019 20:34:19	Editar Remover

Paginação na View

Veja o resultado da nossa tela de listagem das postagens:

Veja que temos apenas algumas postagens nesta tela, agora como fazer para exibirmos a paginação abaixo da tabela para navegação entre as telas da paginação?!

Como usamos o método `paginate` lá no nosso controller, podemos por meio da coleção de posts que recebemos na view, dentro da variável `$post`, exibir a paginação de forma bem simples e direta!

Adicione o trecho abaixo, logo após o fechamento da tag `table` da sua view `index.blade.php`:


```
1 {{ $posts->links() }}
```

Paginação na View

O seguinte trecho exibirá nosso html de navegação entre cada tela da paginação, veja o resultado obtido:

14	aut omnis officiis fuga	Inativo	23/09/2019 20:34:19	Editar	Remover
15	quidem repellat facere nesciunt	Inativo	23/09/2019 20:34:19	Editar	Remover
16	aut consequatur consequuntur et	Inativo	23/09/2019 20:34:19	Editar	Remover

[<](#) [1](#) [2](#) [3](#) [>](#)



Simple assim, agora já temos uma paginação tanto nos dados vindos do banco quando na exibição dos links da navegação de forma simples e direta!

DATABASE: SEEDS

A vertical white line is positioned to the right of the text, extending from the top of the word 'DATABASE:' down to the bottom of the word 'SEEDS'.

Seeds

Quando estamos em nosso ambiente de desenvolvimento nós precisaremos criar dados no banco para teste de nossas lógicas, os seeds nos permitem este processo.

Podemos por meio das seeds alimentar nosso banco de dados com informações para que possamos testar nossos CRUDs por exemplo.

Você pode encontrar os seeds do sistema dentro da pasta `database/seeds`, dentro desta pasta você vai encontrar o arquivo `DatabaseSeeder.php` que é o arquivo principal que executa todas as outras seeds que tivermos nesta pasta.

Seeds

Perceba que temos seu conteúdo mostrado abaixo:

```
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class DatabaseSeeder extends Seeder
6  {
7      /**
8       * Seed the application's database.
9       *
10      * @return void
11      */
12      public function run()
13      {
14          // $this->call(UsersTableSeeder::class);
15      }
16  }
```

Perceba a linha comentada do método `$this->call`, que contém a chamada para um arquivo de seed que ainda não existe mas já nos mostra como podemos registrar os arquivos de seed para serem executados, ou seja, por meio do método `call` e informando a classe de seed em questão.

Seeds

Vamos criar dois arquivos de seed para este momento, primeiro o arquivo que já temos referenciado acima, o UsersTableSeeder e também vamos criar o PostsTableSeeder.

Execute os comandos abaixo, primeiro executando a geração de um, depois do outro:

```
1 php artisan make:seeder UsersTableSeeder
```

Após a execução acima, execute a criação do outro arquivo seeder:

```
1 php artisan make:seeder PostsTableSeeder
```

Seeds

Veja o resultado:

```
blog: php artisan make:seeder UsersTableSeeder  
Seeder created successfully.  
blog: php artisan make:seeder PostsTableSeeder  
Seeder created successfully.  
blog: █
```

Abra o arquivo UsersTableSeeder.php e adicione o código abaixo no método run:

```
1  \DB::table('users')->insert([  
2      'name'      => 'Primeiro Usuário',  
3      'email'     => 'email@email.com',  
4      'password' => bcrypt('secret')  
5  ]);
```

Seeds

Acima temos o primeiro contato com o objeto `DB` que nos permite realizarmos a execução de queries SQL no mais baixo nível em relação a parte do ORM que veremos mais a frente. Então informo a tabela que quero realizar a inserção do dado, por meio do método `table` e logo após, aninhando, chamo o método `insert` informando um array respeitando as colunas que quero adicionar valor e seus valores em questão.

Perceba que no campo de senha do usuário utilizo o helper `bcrypt` para encriptar a senha do nosso usuário.

Após isso adicione o conteúdo do método `run` do `PostsTableSeeder`:

```
1  \DB::table('posts')->insert([
2      'title'      => 'Primeira Postagem',
3      'description' => 'Postagem teste com seeds',
4      'content'     => 'Conteúdo da postagem',
5      'is_active'   => 1,
6      'slug'        => 'primeira-postagem',
7      'user_id'     => 1
8  ]);
```

Seeds

Acima temos o mesmo pensamento com a diferença que estamos lidando com posts e respeitando o nome da tabela e os campos.

Perceba também que referenciei o `user_id` como 1, isso é pouco chato de se fazer assim mas neste caso se encaixa tranquilamente, uma vez que vamos executar a seed de users na ordem antes de posts onde teremos o usuário com id 1 para satisfazer com o `user_id` da postagem definida acima.

Agora vamos voltar lá no DatabaseSeeder e descomentar a linha que temos definida e adicionar mais uma para a chamada do PostsTableSeeder.

Seeds

O conteúdo do arquivo DatabaseSeeder ficará assim, veja ele todo na íntegra abaixo:

```
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class DatabaseSeeder extends Seeder
6  {
7      /**
8       * Seed the application's database.
9       *
10      * @return void
11      */
12      public function run()
13      {
14          $this->call(UsersTableSeeder::class);
15          $this->call(PostsTableSeeder::class);
16      }
17  }
```

Seeds

Temos a chamada descomentada do UsersTableSeeder e adicionamos a chamada para o PostsTableSeeder. Este passo feito, vamos ao terminal e conhecer mais um comando. Desta vez para execução dos nossos seeds.

Em seu terminal e na raiz do projeto execute o comando abaixo:

```
1  php artisan db:seed
```

Veja o resultado:

```
blog: php artisan db:seed
Seeding: UsersTableSeeder
Seeding: PostsTableSeeder
Database seeding completed successfully.
blog: █
```


Seeds

Se você for ao seu banco e consultar as tabelas verá que tens os dados lá como definimos nas classes de seed.

Se precisarmos de mais dados, como por exemplo, inserir 30 posts de primeira por meio dos seeds teríamos um trabalho grande mas este trabalho se simplifica por meio do que chamamos de Model Factories.

Então vamos aprender mais sobre o Laravel conhecendo mais este conceito/ferramenta que nos auxilia neste camada/etapa do desenvolvimento.

Vamos continuando...