

Curso Básico de Laravel.

Prof. Glauco Todesco

Versão 1.0

Setembro/2020

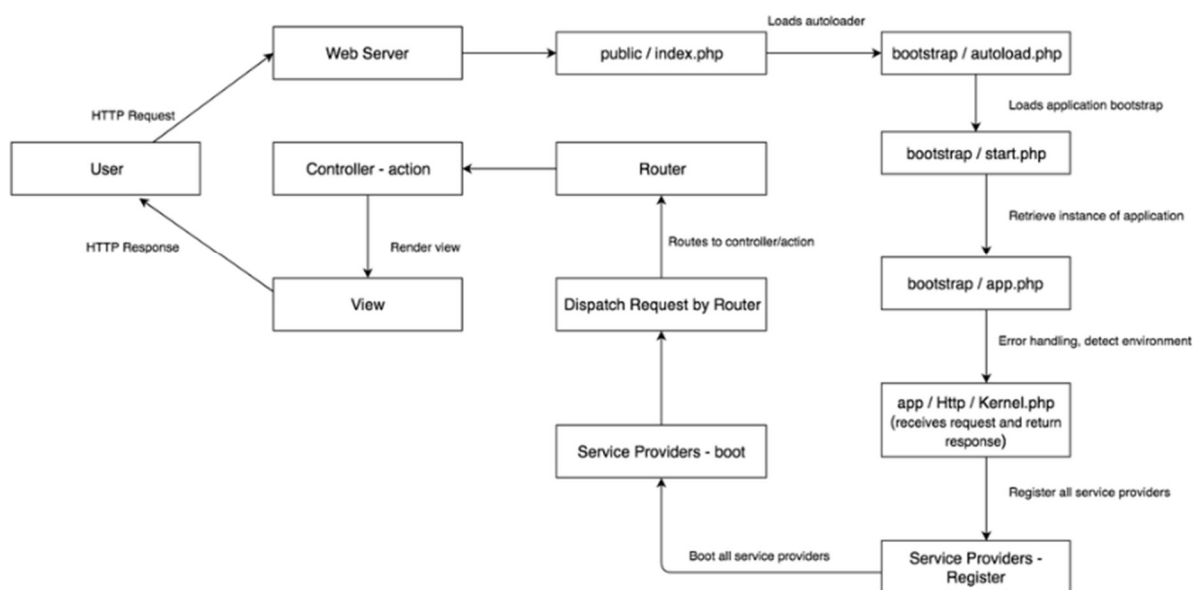
<https://github.com/glaucotodesco/curso-laravel-blog>

Aula00 – Ambiente e Conceitos

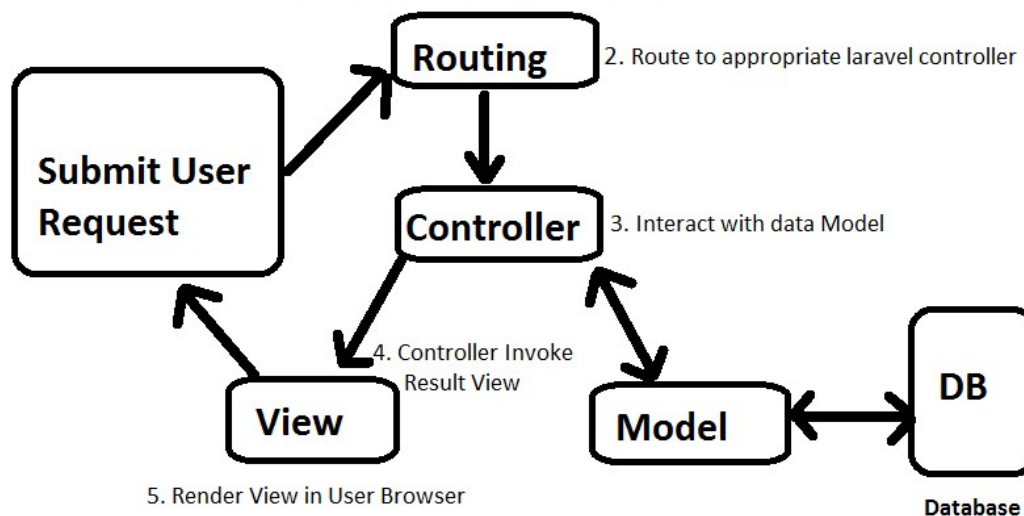
<https://laravel.com/docs/7.x>

- a. Documentação: <https://laravel.com/docs/7.x>
- b. Xampp
- c. VSCode:
 - i. Extensões: Laravel Blade Snippets
- d. Composer: <https://getcomposer.org/download/>
- e. Node.js: <https://nodejs.org/pt-br/download/>
- f. Ciclo de Vida:

Request Life Cycle of Laravel



Laravel Life Cycle Process



- g. Operações com Git (Pasta Vendor)
 - i. Ao fazer o clone executar o comando:
composer install
 - ii. Arquivo de Ambiente:
 - 1. Copiar o exemplo
 - 2. php artisan key:generate
 - 3. php artisan migrate (se tiver banco de dados)

Aula01- Hello

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

1. Criar o projeto Laravel:
 - a. Acessar o diretório a ser criado o projeto
 - b. Criar projeto:

composer create-project laravel/laravel blog

2. Configurar :
 - a. `C:\windows\system32\drivers\etc\hosts`

```
<VirtualHost *:80>
    DocumentRoot "C:/Users/GPU01/Temp/blog/public"
    <Directory "C:/Users/GPU01/Temp/blog/public">
        Options Indexes FollowSymLinks Includes ExecCGI
        AllowOverride All
        Require all granted
    </Directory>
    ServerName blog.localhost
</VirtualHost>
```

- b. `C:\xampp\apache\conf\extra\ httpd-vhosts.conf`

```
127.0.0.1 curso.localhost
```

3. Abrir o projeto no code.
4. Acessar o projeto pelo navegador:
 - a. `http://blog.localhost/`
5. Criando um novo View (/resource/views) (copia do welcome)
6. Configurando a rota para o novo view (/routes/web.php)
7. Criando um Controle para a nova view
 - a. **php artisan make:controller HelloController**
 - b. **construindo uma nova função**

```
public function index(){
    return view('hello');
}
```

8. Configurando a rota para o controlador:

```
Route::get('/hello', 'HelloController@index');
```

9. Acessando informações do .env no view

10. Abra **.env** e o **.env.example** e altere o chave APP_NAME
APP_NAME=Blog

11. Altere o view:

```
<title>{{config('app.name', 'Blog')}}</title>
```

Aula02- Exercício

1. Exercício:
 - a. Crie um novos view (Index)
 - b. Crie um novo diretório no view chamado pages
 - c. Crie dentro do diretório pages 3 novos views About, Projects e Services.
 - d. Crie um Controlador chamado PagesController
 - e. Mapear os recursos no controlador.
 - i. /about
 - ii. /projects
 - iii. /services
 - iv. /

Aula03-Layout

1. Layout

- a. Criar uma pasta em view chamada layouts
- b. Criar o arquivo app.blade.php dentro da para layouts

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{{config('app.name', 'CursoLaravel')}}</title>
</head>
<body>
    @yield('content')
</body>
</html>
```

- c. Aplicar o layout aos views

```
@extends('layouts.app')

@section('content')
<h1>About!</h1>
@endsection
```

Aula 04 – Enviado dados para o View

1. Passagem de informações do controlador para o view.

```
public function index(){  
    $title = "Sistema de Blogs 1.0";  
    return view('index')->with('Title', $title);  
}
```

```
@extends('layouts.app')  
  
@section('content')  
    <h1>Index Usando Layout!</h1>  
    <h2>{{ $title }}</h2>  
@endsection
```


Aula 05 – Exercício

1. Enviar do Controlador para o View **About** duas informações: autor e empresa.
2. Enviar do Controlador para o View **Services** um Array de Dados com os nomes dos serviços prestados.
 - Usar no View **@if(count(\$servicos) > 0) @endif** para ver se o array não está vazio.
 - Usar no View **@foreach (\$servicos as \$item) @endforeach** para processar o array.

Aula 06- Configurando BootStrap

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

1. Instalar o Node
2. Executar o comando:

composer require laravel/ui

3. Executar o comando:

php artisan ui bootstrap --auth

4. Executar o comando para compilar o BootStrap:

npm install && npm run dev

5. Teste a aplicação no navegador
6. Fix use for Auth em web.php

```
use Illuminate\Support\Facades\Auth;
```

Aula 07 – Exercício- Alterando o NavBar

1. Acesse o NOVO arquivo de layout
2. Procure pela diretiva @guest
3. Acrescente no NAV 4 novos itens: Home, Projetos, Serviços e Sobre.
4. Acrescente a classe container ao main:

```
<main class="py-4 container">  
    @yield('content')  
</main>
```

Aula 08 – Migrate

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

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

1. Criar o banco de dados BlogDB no MySQL
2. Configurar o arquivo .env Abra **.env** e o **.env.example**:

```
DB_DATABASE=blogdb
```

3. Acesse os arquivos de migração em database\migrations
4. Execute a migração e veja as tabelas criadas:

```
php artisan migrate
```

5. Faça um select na tabela migrations
6. Faça um roolback do banco e veja as tabelas criadas:

```
php artisan migrate:rollback
```

7. Faça um select na tabela migrations
8. Execute a migração:

```
php artisan migrate
```

9. O login e o registro de funcionários já estão funcionando. Teste e veja os controles e views criados.
10. Criando um modelo. Execute o comando abaixo e veja os arquivos criados.

```
php artisan make:model Post -m
```

11. Criando o migration do Post up

```
Schema::create('posts', function (Blueprint $table) {  
    $table->id();  
    $table->string('title');  
    $table->mediumText('body');  
    $table->timestamps();  
});
```

12. Rode o migrate e veja a nova tabela construída.

```
php artisan migrate
```

13. Criando um Post com o tinker:

1. Execute o comando:

php artisan tinker

2. Crie um novo Post:

```
$post = new App\Post();  
$post->title = 'My Post 01';  
$post->body='Body of my post 01';  
$post->save();
```

3. Faça um select na tabela Posts

14. Exercício:

Crie 3 Registro para na tabela Posts e teste os comandos abaixo:

1. `App\Post::all();`
2. `App\Post::count();`
3. `$post = App\Post::find(1);`
4. `$post = App\Post::find(1);`
`$post->delete();`
5. `App\Post::orderBy('title','asc')->get();`

Aula 09 – Add Controller to Posts

1. Crie um controlador para o model Post com a opção de recursos:

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

2. Veja as rotas já criadas:

```
php artisan route:list
```

3. Crie todas as rotas para o controlador automaticamente:

```
Route::resource('posts', 'PostsController');
```

4. Veja novamente as rotas criadas.

Aula10 – Get All Posts

1. Na função index do controlador de posts, retorne todos os Posts.

```
use App\Post;
```

```
public function index()  
{  
    return Post::all();  
}
```

2. Teste no navegador.

<http://blog.localhost/posts>

3. Crie uma pasta no view chamada posts
4. Crie um novo view (**index.blade.php**) para listar todos os Posts dentro da pasta criada no passo anterior.

```
@extends('layouts.app')  
@section('content')  
    <h1>Postagens</h1>  
    @if(count($posts) > 1)  
        @foreach ($posts as $post)  
            <div class="card">  
                <h3>{{ $post->title }}</h3>  
            </div>  
        @endforeach  
    @endif  
@endsection
```

5. Altere o controlador para direcionar a requisição para o view:

```
public function index()  
{  
    $posts = Post::all();  
    return view('pages.posts.index')->with('posts', $posts);  
}
```

6. Teste no navegador
7. Exercícios. Altere o view do index do Post para:

- a. Mostrar a data de criação.
- b. Formatar a data dd/mm/aaaa hh:mm:ss.
- c. Mostrar somente os 10 primeiros caracteres do body.

Aula11 – Exercício BootStrap

1. Aplicando o BootStrap para visualizar todos os Posts
2. Acesse a documentação do BootStrap para a classe Card

<https://getbootstrap.com/docs/4.0/components/card/>

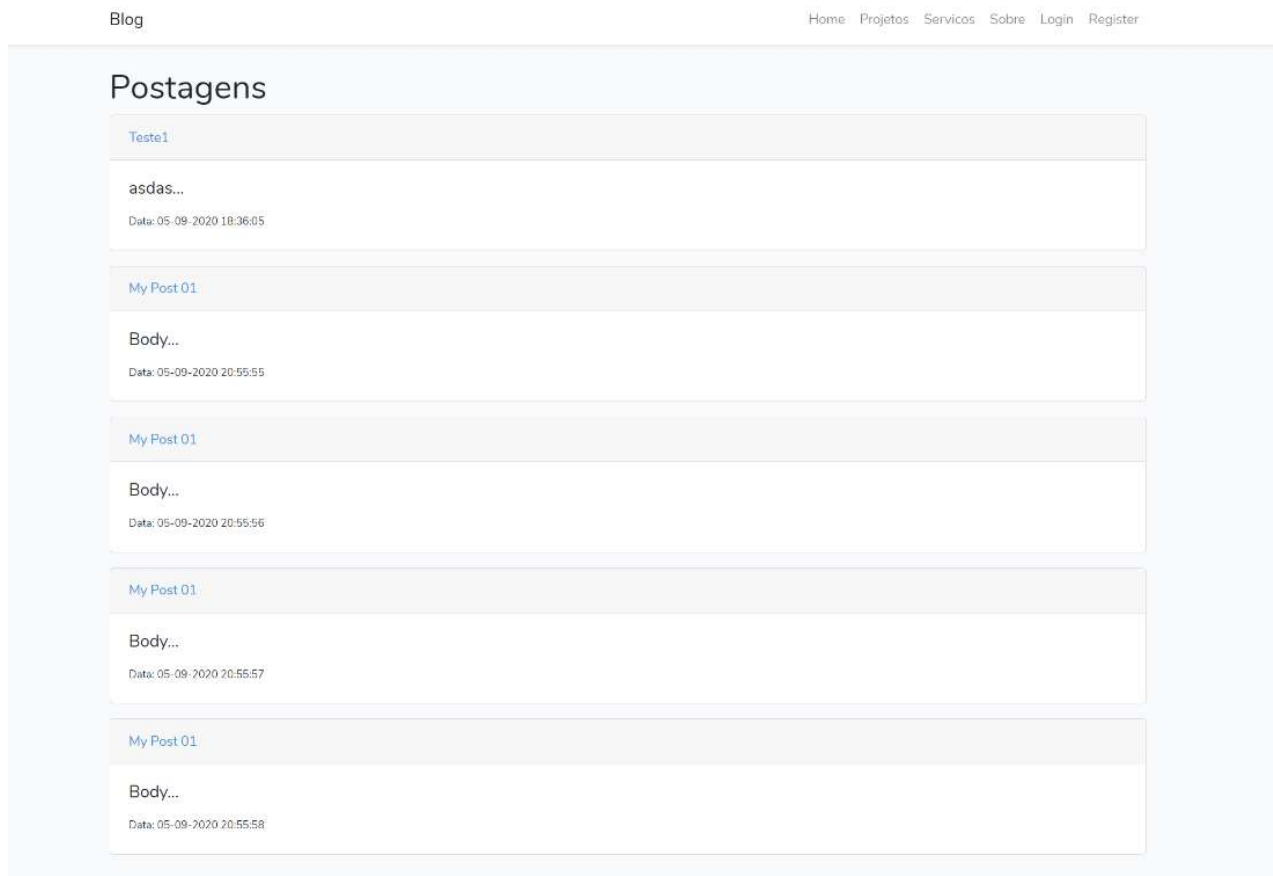
3. Use:

card-header para o título de um post com um link para “/posts/{{ \$post->id}}”

card-body para o corpo de um post

card-title para os 10 primeiros caracteres do post

card-text para a data e hora do post



Aula12 – Open Post

1. Altere o controlador para retornar um post:

```
public function show($id)
{
    return Post::find($id);
}
```

2. Teste no navegador:

<http://blog.localhost/posts/1>

3. Exercício: Mostrando um post.

- Crie um View para mostrar o post selecionado.
Arquivo: **/view/posts/show.blade.php**
- Adicione um botão para voltar aos posts.
- Altere o controlador para chamar o view com o post selecionado.

Aula 13- Exercício: Ordenando Posts

1. Altere o controlador para retornar posts ordenados:

- Título de forma ascendente
- Data de criação de forma descendente

Aula 14 – Executando Comandos SQL

1. Altere o controlador de Posts para usar a classe DB:

```
use Illuminate\Support\Facades\DB;
```

```
public function index()
{
    $posts = DB::select('Select * FROM Posts');
    return view('pages.posts.index')->with('posts', $posts);
}
```

2. Altere o View para formatar a data

```
<small class="card-text">
    Data: {{ date('d-m-Y H:i:s',
                strtotime(str_replace('-', '/', $post->created_at))
            )
    }}
</small>
```

Aula 15 – Salvando um Objeto

1. Alterando o Controle para direcionar ao View para criar um novo Post.

```
public function create()
{
    return view('posts.create');
}
```

2. Instalando os templates html

<https://laravelcollective.com/docs>

Execute o comando:

```
composer require laravelcollective/html
```

3. Abrindo um formulário no view:

```
{!! Form::open(['action' => 'PostsController@store']) !!}

{!! Form::close() !!}
```

4. Adicionando no form os inputs de um Post

```
@extends('layouts.app')
@section('content')
    <a href="/posts" class="btn btn-default">Voltar</a>

    <h1>Nova Postagem</h1>
    {!! Form::open(['action' => 'PostsController@store']) !!}
    <div class="form-group">
        {!! Form::label('title', 'Título') !!}
        {!! Form::text('title', '', ['class' => 'form-control', 'placeholder' => 'Título']) !!}
    </div>

    <div class="form-group">
        {!! Form::label('body', 'Mensagem') !!}
        {!! Form::textarea('body', '', ['class' => 'form-control', 'placeholder' => 'Mensagem']) !!}
    </div>

    {!! Form::submit('Salvar', ['class' => 'btn btn-primary']) !!}

    {!! Form::close() !!}
```

```
@endsection
```

5. Salvando no Banco de Dados:

```
public function store(Request $request)
{
    $post = new Post();
    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->save();

    return redirect('/posts');
}
```

Aula 16 – Validação de Formulário

1. Acrescente no Controlador de Posts@store a chamada para validar os dados:

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required',
        'body' => 'required',
    ]);

    $post = new Post();
    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->save();

    return redirect('/posts')->with('success', 'Sucesso.');
```

2. Crie um arquivo para apresentar todas as mensagens de erro (**messages.blade.php**)

```
@if (count($errors) > 0)
    @foreach ($errors->all() as $error)
        <div class="alert alert-danger">
            {{ $error }}
        </div>
    @endforeach
@endif

@if(session('success'))
    <div class="alert alert-success">
        {{session('success')}}
    </div>
@endif

@if(session('error'))
    <div class="alert alert-danger">
        {{session('error')}}
    </div>
@endif
```

3. Inclua o arquivo mensagens no layout:

```
<main class="py-4 container">
```

```
    @include('messages')  
    @yield('content')  
</main>
```

4. Teste com um formulário em branco.

Aula 17 – Tradução das mensagens de validação.

1. Abra a pasta de mensagens de validação:
 \resource\lang\en
2. Acesse:
 <https://blog.especializati.com.br/traduzir-laravel/>

 ou

 <https://github.com/lucascudo/laravel-pt-BR-localization>
3. Faça o clone ou download de um dos projetos
4. Copie a pasta /src/pt_BR para o projeto
5. Configure o arquivo config/app.php para pt_BR

```
'locale' => 'pt_BR',
```

```
'fallback_locale' => 'pt_BR',
```

6. Teste.

Aula 18 – Exercício

1. Adicione um botão no index das postagens para criar um Post
2. Crie o botão seguindo as classes já utilizadas do BootStrap
3. Acrescente um novo Item no Nav Bar para chamar os Posts

Aula 19 – Editando um Post

1. **Exercício:** Adicione um link para editar uma postagem na view show. Veja as rotas já criadas para construir o link correto.
2. Adicione no controlador o código para pesquisar o post e direcionar para o view de edição.

```
public function edit($id)
{
    $post = Post::find($id);
    return view('posts.edit')->with('post',$post);
}
```

3. Crie o view acima, copiando o código do view para criar um Post, alterando o código para mostrar os dados do objeto post.

```
{!! Form::open(['action' => ['PostsController@update',
$post->id], 'method' => 'PUT' ]) !!}
```

```
{{ Form::text('title', $post->title, ['class' => 'form-control'
, 'placeholder' => 'Título']) }}
```

```
{{ Form::textarea('body', $post->body, ['class' => 'form-control'
, 'placeholder' => 'Mensagem']) }}
```

4. Altere a function update do controlador:

```
public function update(Request $request, $id)
{
    $this->validate($request, [
        'title' => 'required',
        'body' => 'required',
    ]);

    $post = Post::find($id);
    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->save();

    return redirect('/posts')->with('success', 'Mensagem Atualizada.');
```

5. Teste a funcionalidade de edição.

Aula 20 – Removendo Um Post

1. Altere o view show de um post acrescentando o código para remover um post:

```
<div>
  {!! Form::open(['action' => ['PostsController@destroy',
    $post->id], 'method' => 'DELETE', 'class' => 'pull-right' ]) !!}
  <a href="/posts/{{ $post->id }}/edit" class="btn btn-primary">Editar Mensagem</a>
  {!! Form::submit('Remover', ['class' => 'btn btn-danger']) !!}
  {!! Form::close() !!}
</div>
```

2. Implemente a function destroy no Controlador:

```
public function destroy($id)
{
    $post = Post::find($id);
    $post->delete();
    return redirect('/posts')->with('success','A Postagem foi removida!');
}
```

3. Teste a funcionalidade de remoção.

Aula21 – Relacionando Um Post ao Usuário

1. Veja a estrutura atual da tabela Posts
2. Execute o comando abaixo para criar um migration para acrescentar o id do usuário em um Post:

```
php artisan make:migration add_user_id_to_posts
```

3. Acrescente os códigos dos métodos up e down:

```
public function up()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->bigInteger('user_id')->unsigned();
        $table->foreign('user_id')->references('id')->on('users');
    });
}
```

```
public function down()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->dropForeign(['user_id']);
        $table->dropColumn('user_id');
    });
}
```

4. Execute o migrate

```
php artisan migrate
```

Obs: Apague todos os posts ou defina um id para os posts já criados.

5. Veja a nova estrutura da tabela Posts.
6. Ajuste no controlador para definir um usuário de um post ao criar um Post.

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required',
```

```
        'body' => 'required',
    ]));

    $post = new Post();
    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->user_id = auth()->user()->id;
    $post->save();

    return redirect('/posts')->with('success','Sucesso');
}
```

7. Teste e veja se o id do usuário foi associado corretamente ao novo post.

Atenção: Você precisa estar logado!

Aula22 – Associação Um-Para-Muitos

1. O próximo passo é estabelecer uma relação de um para muitos (Um usuário possui muitos posts e um post é de um usuário).
2. Alterando o Modelo Post:

```
class Post extends Model
{
    public function user(){
        return $this->belongsTo('App\User');
    }
}
```

3. Alterando o Modelo Users acrescentado:

```
public function posts(){
    return $this->hasMany('App\Post');
}
```

4. Teste:
 - a. Crie dois usuários, cada um com 2 posts
 - b. Usando o Tinker teste (Busque por um id que exista no Banco):

```
$u = User::find(1)
$u->posts()
```

```
$u = User::find(2)
$u->posts()
```

```
$p = Post::find(1);
$p->user
$p->user->name
```

Aula23 – Visualizando Posts de um Usuário

1. Altere a função index do HomeController

```
use App\User;
```

```
public function index()
{
    $user_id = auth()->user()->id;
    $user = User::find($user_id);
    return view('home')->with('posts',$user->posts);
}
```

2. Altere o view do home.blade.php

```
@if (count($posts) > 0)
    @foreach ($posts as $post)
        <div class="card mb-3">
            <div class="card-header">
                <a href="/posts/{{ $post->id }}">{{ $post->title }}</a>
            </div>
            <div class="card-body">
                <h5 class="card-title"> {{ Illuminate\Support\Str::limit($post->body, $limit = 5, $end = '...') }}</h5>
                <?php //<small class="card-text">Data: {{ $post->created_at->format('d-m-Y H:i:s') }}</small> ?>
                <small class="card-text">
                    Data: {{ date('d-m-Y H:i:s',strtotime(str_replace('-', '/', $post->created_at))) }}
                </small>
            </div>
        </div>
    @endforeach
@endif
```

3. Acrescente um item no dropdown do layout principal

```
<div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="{{ route('home') }}">Minhas Postagens</a>
```

4. Faça o login e teste.

Aula24 – Controle de Acesso

1. Criando um Post só quando estiver logado.
2. Acrescente o seguinte código no PostsController:

```
public function __construct()
{
    $this->middleware('auth');
}
```

3. Teste. Agora todo o controlador do Post possui controle de acesso.
4. Para ver todos os posts sem estar logado, é necessário liberar o acesso:

```
public function __construct()
{
    $this->middleware('auth',['except' => ['index','show']]);
}
```

5. Controlando o acesso aos botões Editar e Remover. Altere o view show de um post:

```
@if(!Auth::guest())
    {!! Form::open(['action' => ['PostsController@destroy', $post->id]
    , 'method' => 'DELETE', 'class' => 'pull-right' ]) !!}
    <a href="/posts/{{ $post->id }}/edit" class="btn btn-primary">Editar Mensagem</a>
    {!! Form::submit('Remover',['class' => 'btn btn-danger']) !!}
    {!! Form::close() !!}
@endif
```

6. Além de estar logado, para editar o post deve ser do usuário. Altere o if para:

```
@if(!Auth::guest() && Auth::user()->id == $post->user_id)
```

7. Alterando para permitir que somente o dono possa alterar um post:

```
public function edit($id)
{
    $post = Post::find($id);
    if(auth()->user()->id != $post->user_id)
        return redirect('/posts')->with('error','Acesso não autorizado!');
    else
        return view('posts.edit')->with('post',$post);
}
```

8. Alterando para permitir que somente o dono possa remover um post:

```
public function destroy($id)
{
    $post = Post::find($id);
    $post = Post::find($id);

    if(auth()->user()->id !== $post->user_id){
        return redirect('/posts')->with('error','Acesso não autorizado!');
    }
    else{
        $post->delete();
        return redirect('/posts')->with('success','A Postagem foi removida!');
    }
}
```