+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Comando artisan para criar as classes model

php artisan make:model Produto

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Eloquent é um framework que já vem com o laravel ORM ele ajuda a mapear o mundo relacional com a orientação a objetos do PHP, podemos começar a remover as instruções SQL do nosso código.

Vamos remover o a instrução SQL pelo nosso model Produto.php que foi herdado

Alterando de

public function lista()

{

$produtos = DB::select('select \* from produtos');

return view('listagem')->with('produtos', $produtos);

}

Para: (Adicionando a classe com o produto usando use)

use App\Produto;

public function lista()

{

//Fazendo a listagem com a ORM(Eloquent)

$produtos = Produto::all();

return view('listagem')->with('produtos', $produtos);

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para fazer uma busca pode-se usar uma função chamada FIND

public function mostra()

{

$id = Request::route('id');

//$produto = DB::select('select \* from produtos where id = ?', [$id]);

$produto = Produto::find($id);

//return view('detalhes')->with('p', $produto[0]);

//Agora vai retornar só um produto sem array

return view('detalhes')->with('p', $produto);

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora chegou a hora de adicionar os produtos primeiro alteramos o nosso método adiciona() para o Eloquent deixando da seguinte forma.

public function adiciona()

{

$produto = new Produto();

//pegar as informações do formulario

//$nome = Request::input('nome');

//$quantidade = Request::input('quantidade');

//$valor = Request::input('valor');

//$descricao = Request::input('descricao');

$produto->nome = Request::input('nome');

$produto->quantidade = Request::input('quantidade');

$produto->valor = Request::input('valor');

$produto->descricao = Request::input('descricao');

//salvar no banco de dados

//DB::insert('insert into produtos (nome, quantidade, valor, descricao) values (?,?,?,?)', array($nome, $quantidade, $valor, $descricao));

$produto->save();

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

Mas antes para evitar o erro (SQLSTATE[42S22]:Column not found:1054 = updated\_at, created\_at) acontece pq o Eloquent por padrão espera o campo de ultima atualização updated\_at e campo de data de criação do registro created\_at, então faremos o seguinte para evitar esse problema.

Vamos adicionar no modelo Produto.php

class Produto extends Model

{

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Um problema comum é como o Eloquent sabe que a nossa classe Produto está relacionado com a tabela produtos, é porque ela sabe que vai ser o nome da classe só que com a primeira letra minúscula e no plural é um padrão do Eloquent.

Caso o nome da tabela fosse diferente do PADRÂO do Eloquent precisamos fazer o seguinte

class Produto extends Model

{

//Caso nome da tabela fosse diferente da classe, $table+nomedatabela

protected $table = 'produtos';

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para evitar no método adicionar ter todo o trabalho de inserir campo por campo podemos usar um recurso do Laravel o $params = Request::all(); e tabela fica muito mais simples principalmente em tabelas com um grande numero de colunas

Então alteramos no método adiciona() para:

public function adiciona()

{

//pegar as informações do formulario

$params = Request::all();

$produto = new Produto($params);

//$nome = Request::input('nome');

//$quantidade = Request::input('quantidade');

//$valor = Request::input('valor');

//$descricao = Request::input('descricao');

//$produto->nome = Request::input('nome');

//$produto->quantidade = Request::input('quantidade');

//$produto->valor = Request::input('valor');

//$produto->descricao = Request::input('descricao');

//salvar no banco de dados

//DB::insert('insert into produtos (nome, quantidade, valor, descricao) values (?,?,?,?)', array($nome, $quantidade, $valor, $descricao));

$produto->save();

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Ainda para o método adiciona() funcionar e evitar o erro(MassAssignmentException in Model.php line 444: \_token) esse error nada mais é do que a necessidade de passar um token pelo fato de o Request::all() coletar um número grande de campos, isso será feito no model Produto.php

class Produto extends Model

{

//Caso nome da tabela fosse diferente da classe, $table+nomedatabela

protected $table = 'produtos';

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

//para evitar o erro do massAssignmentException all() do controller

//o fillable ignora token ou senha e só pega os campos no array

protected $fillable =

array('nome', 'descricao', 'quantidade', 'valor');

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para deixar ainda mais simple a criação de um registro no nosso método adiciona()

public function adiciona()

{

//salvar no banco de dados

Produto::create(Request::all());

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Vamos criar um método para remover produtos

Primeiro criamos a rota em \app\Http\routes.php

Route::get('/produtos/remove/{id}', 'ProdutoController@remove');

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Criamos o método remove no nosso controller ProdutoController.php

public function remove()

{

$id = Request::route('id');

$produto = Produto::find($id);

$produto->delete();

return redirect('/produtos');

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

E por último adicionamos o ícone lixeirinha no nosso listagem.blade.php

<td>

<a href="/produtos/remove/{{$p->id}}">

<span class="glyphicon glyphicon-trash" aria-hidden="true">

</span>

</a>

</td>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

No método mostra() vamos remover a linha ($id = Request::route('id');) e passar o id como parâmetro no método mostra como abaixo

public function mostra($id)

{

$produto = Produto::find($id);

return view('detalhes')->with('p', $produto);

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Fazer a mesma alteração no método remove

public function remove($id)

{

//$id = Request::route('id');

$produto = Produto::find($id);

$produto->delete();

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

No método remove() fazer alteração na linha do redirect tirando o (/produtos) e adicionando a rota direto com o action(‘ProdutoController@lista’);

public function remove($id)

{

//$id = Request::route('id');

$produto = Produto::find($id);

$produto->delete();

//return redirect('/produtos');

return redirect()->action('ProdutoController@lista');

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Vamos adicionar um campo novo na tabela produtos com a migration, a migration funciona como um controle de versão do banco de dados. Vamos criar uma migration com o nome adiciona\_tamanho\_no\_produto para isso digitamos no terminal.

php artisan make:migration adiciona\_tamanho\_no\_produtos

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Faça as alterações na migration criada

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class AdicionaTamanhoNoProdutos extends Migration

{

public function up()

{

//adicionamos table pq a tabela já existe

Schema::table('produtos', function($table){

//adiciona campo tamanho do tipo string, no maximo 100 caracteres

$table->string('tamanho', '100');

});

}

public function down()

{

Schema::table('produtos', function($table){

//desfazer a alteração e apagar a coluna tamanho

$table->dropColumn('tamanho');

});

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos rodar todas as nossas migrations para fazer as alterações, para isso rodamos no terminal o comando.

php artisan migrate

Agora a tabela produtos terá um campo novo, caso queira desfazer a migração voltando para o estado anterior digite o comando

php artisan migrate:rollback

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Vamos adicionar o campo para a essa nova informação “tamanho” em formulário.blade.php

<div class="container">

<form action="/produtos/adiciona" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control" />

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input name="quantidade" class="form-control"/>

</div>

<div class="form-group">

<label>Tamanho</label>

<input name="tamanho" class="form-control"/>

</div>

<div class="form-group">

<label>Descrição</label>

<input name="descricao" class="form-control"/>

</div>

<button class="btn btn-primary" type="submit">Adicionar</button>

</form>

</div>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Alterar o listagem.blade.php

<div class="container">

<h1>Listagem de produtos</h1>

<table class="table">

@foreach($produtos as $p)

<tr class="{{ $p->quantidade <=1 ? 'danger': ''}}">

<td>{{ $p->nome }}</td>

<td>{{ $p->valor }}</td>

<td>{{ $p->descricao }}</td>

<td>{{ $p->quantidade }}</td>

<td>{{ $p->tamanho }}</td>

<td>

<a href="/produtos/mostra/{{$p->id}}">

<span class="glyphicon glyphicon-search" aria-hidden="true">

</span>

</a>

</td>

<td>

<a href="/produtos/remove/{{$p->id}}">

<span class="glyphicon glyphicon-trash" aria-hidden="true">

</span>

</a>

</td>

</tr>

@endforeach

</table>

@if(old('nome'))

<div class="alert alert-success">

<!-- mantém os dados da requisição anterior -->

Produto {{old('nome')}} adicionado com sucesso!

</div>

@endif

</div>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora é preciso liberar os campos para que possam ser atribuídos dessa forma, alteramos o nosso model Produto adicionando o campo tamanho assim liberando o insert para o campo tamanho

class Produto extends Model

{

//Caso nome da tabela fosse diferente da classe, $table+nomedatabela

protected $table = 'produtos';

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

//para evitar o erro do massAssignmentException all() do controller

//o fillable ignora token ou senha e só pega os campos no array

protected $fillable =

array('nome', 'descricao', 'quantidade', 'valor', ‘tamanho’);

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos validar os produtos que nos estamos inserindo não permitindo inserção de campos vazios, mascaras e outros padrões.

Alteramos o nosso método adiciona() do ProdutoController.php

use Validator;

public function adiciona()

{

//pegar as informações do formulario

//$params = Request::all();

//$produto = new Produto($params);

//$nome = Request::input('nome');

//$quantidade = Request::input('quantidade');

//$valor = Request::input('valor');

//$descricao = Request::input('descricao');

//$produto->nome = Request::input('nome');

//$produto->quantidade = Request::input('quantidade');

//$produto->valor = Request::input('valor');

//$produto->descricao = Request::input('descricao');

//salvar no banco de dados

//DB::insert('insert into produtos (nome, quantidade, valor, descricao) values (?,?,?,?)', array($nome, $quantidade, $valor, $descricao));

//$produto->save();

//O validator ajuda a gente a fazer qualquer regra de validação

$validator = Validator::make(

//qual campo a validar+ qual valor quer validar

['nome' => Request::input('nome')],

//campo obrigatorio | no minino 3 caracteres

['nome' => 'required|min:3']

);

//validação com mensagens

if ($validator->fails()) {

$msg = $validator->messages();

dd($msg);

return redirect('/produtos/novo');

}

Produto::create(Request::all());

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Em formulário grande é recomendado fazer a validação de outra forma para evitar que o método adicionar() acabe ficando muito grande. Para isso vamos isolar a nossa regra de validação em uma classe especifica isso será feito com o FormRequest isso será feito com o artisan pelo terminal com o comando.

php artisan make:request ProdutoRequest

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

A request ficará em \app\Http\Requests\ProdutoRequest.php

No ProdutoRequest.php será feita as seguintes alterações para as validações

use App\Http\Requests\Request;

class ProdutoRequest extends Request

{

//aqui ele vai dizer se eu pode ou não validar essa autorização

public function authorize()

{

return true;

}

//aqui eu isolo todas as minha regras de validação

public function rules()

{

return [

'nome' => 'required|min:3',

'descricao' => 'required|max:255',

'valor' => 'required|numeric',

'quantidade' => 'required|numeric',

'tamanho' => 'required|max:100'

];

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

No ProdutoController.php fazemos as seguintes alterações no método adiciona() para aplicar as validações que estão na classe ProdutoRequest.php

public function adiciona(ProdutoRequest $request)

{

Produto::create($request->all());

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para mostrar os erros usamos um método chamado $errors com all() para mostrar todos os erros para fazermos isso usamos ele dentro de um foreach

Primeiro vamos alterar o formulario.blade.php

@extends('principal')

@section('conteudo')

<div class="alert alert-danger">

<ul>

@foreach($errors->all() as $error)

<li>{{$error}}</li>

@endforeach

</ul>

</div>

<div class="container">

<form action="/produtos/adiciona" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control" />

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input name="quantidade" class="form-control"/>

</div>

<div class="form-group">

<label>Tamanho</label>

<input name="tamanho" class="form-control"/>

</div>

<div class="form-group">

<label>Descrição</label>

<input name="descricao" class="form-control"/>

</div>

<button class="btn btn-primary" type="submit">Adicionar</button>

</form>

</div>

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos customizar as nossas mensagens criando um terceiro método no ProdutoRequest.php

namespace App\Http\Requests;

use App\Http\Requests\Request;

class ProdutoRequest extends Request

{

//aqui ele vai dizer se eu pode ou não validar essa autorização

public function authorize()

{

return true;

}

//aqui eu isolo todas as minha regras de validação

public function rules()

{

return [

'nome' => 'required|min:3',

'descricao' => 'required|max:255',

'valor' => 'required|numeric',

'quantidade' => 'required|numeric',

'tamanho' => 'required|max:100'

];

}

//aqui vamos customizar as nossas mensagens

public function messages()

{

return[

//aqui vai o input + regra de validação

//ex: nome.validação obrigatoria

'required' => 'O :attribute é obrigatório!',

];

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para remover a div error adicionamos um contador na estrutura formulário.blade.php

@if (count($errors) > 0)

@foreach ($errors->all() as $error)

* {{ $error }}

@endforeach

@endif

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Adicione o {{ old(‘nome do campo’) }} para manter as informações digitas anteriormente

@extends('principal')

@section('conteudo')

@if (count($errors) > 0)

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<div class="container">

<form action="/produtos/adiciona" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control" value="{{ old('nome') }}" />

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control" value="{{ old('valor') }}"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input name="quantidade" class="form-control" value="{{ old('quantidade') }}"/>

</div>

<div class="form-group">

<label>Tamanho</label>

<input name="tamanho" class="form-control" value="{{ old('tamanho') }}"/>

</div>

<div class="form-group">

<label>Descrição</label>

<input name="descricao" class="form-control" value="{{ old('descricao') }}"/>

</div>

<button class="btn btn-primary" type="submit">Adicionar</button>

</form>

</div>

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos fazer autenticação para controle de usuários.

Primeiro vamos alterar o arquivo routes.php

Route::get('/', 'ProdutoController@lista');

Route::get('/produtos', 'ProdutoController@lista');

Route::get('/produtos/mostra/{id}', 'ProdutoController@mostra');

Route::get('/produtos/remove/{id}', 'ProdutoController@remove');

Route::get('/produtos/novo', 'ProdutoController@novo');

Route::post('/produtos/adiciona', 'ProdutoController@adiciona');

Route::get('/login', 'LoginController@form');

Route::get('home', 'HomeController@index');

Route::controllers([

'auth' => 'Auth\AuthController',

'password' => 'Auth\PasswordController',

]);

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Vamos criar um o nosso controller para o login com o comando no terminal, só que nesse caso adicionamos o –plain para que crie a classe sem os método padrão do crud

php artisan make:controller LoginController --plain

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

No nosso no controller LoginController adicionamos o nosso método form()

public function form()

{

return view('form\_login');

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos criar o nosso view form\_login.blade.php em resurces\views e adicionar conteúdo a ele

@extends('principal')

@section('conteudo')

@if (count($errors) > 0)

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<div class="container">

<form action="/login" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Email</label>

<input name="email" class="form-control" value="{{ old('email') }}" />

</div>

<div class="form-group">

<label>Senha</label>

<input type="password" name="password" class="form-control" value="{{ old('password') }}"/>

</div>

<button class="btn btn-primary" type="submit">Login</button>

</form>

</div>

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Fazer alteração em routes.php

Route::get('/', 'ProdutoController@lista');

Route::get('/produtos', 'ProdutoController@lista');

Route::get('/produtos/mostra/{id}', 'ProdutoController@mostra');

Route::get('/produtos/remove/{id}', 'ProdutoController@remove');

Route::get('/produtos/novo', 'ProdutoController@novo');

Route::post('/produtos/adiciona', 'ProdutoController@adiciona');

Route::post('/login', 'LoginController@form');

Route::get('home', 'HomeController@index');

Route::controllers([

'auth' => 'Auth\AuthController',

'password' => 'Auth\PasswordController',

]);

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos implementar o método login() no nosso controller LoginController.php, usamos a classe Auth que tem vários métodos para nos ajudar na autenticação.

namespace App\Http\Controllers;

use App\Http\Requests;

use App\Http\Controllers\Controller;

use Request;

use Auth;

class LoginController extends Controller

{

public function form()

{

return view('form\_login');

}

public function login()

{

//pegar credenciais apenas email e senha

$credenciais = Request::only('email', 'password');

//passar credenciais para o attempt(valida e entrar)

if (Auth::attempt($credenciais)) {

return 'usuario está logado com sucesso!';

}

return 'usuario não existe';

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora queremos que os usuários só abram a listagem depois de logados.

Vamos criar um middleware para controle de usuários com o comando no terminal, middleware é uma camada que sempre será executada antes do código, podemos cria-lo digitando no terminal o seguinte comando.

php artisan make:middleware Autorizador

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Adicionar use Auth; no ProdutoController.php

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

O middleware possui um método handle com o parâmetros $request e Closure $next, funciona assim ele faz o $request se tiver autenticado ele executa o código a seguir caso não retorna a tela de login.

Faça a alteração no Middleware Autorizador na pasta Http\Middleware\Autorizador.php

namespace App\Http\Middleware;

use Closure;

class Autorizador

{

public function handle($request, Closure $next)

{

if (\Auth::guest()) {

//se não tiver logado eu quero mandar para a pagina de login

return redirect('/login');

}

return $next($request);

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Todo middleware precisa ser registrado no arquivo Kernel.php que fica em \Http\

protected $middleware = [

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,

\App\Http\Middleware\EncryptCookies::class,

\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

\Illuminate\Session\Middleware\StartSession::class,

\Illuminate\View\Middleware\ShareErrorsFromSession::class,

\App\Http\Middleware\VerifyCsrfToken::class,

\App\Http\Middleware\Autorizador::class,

];

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Não vai funcionar porque o middleware vai funcionar antes de todos os controllers, caso não esteja logado ele volta para a tela de login

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para o autorizado volta a funcionar faça a seguinte alteração no middleware Autorizador.php

namespace App\Http\Middleware;

use Closure;

class Autorizador

{

public function handle($request, Closure $next)

{

//A request não é para barra login e o cara é convidado

if (!$request->is('login') && \Auth::guest()) {

//se não para a barra login e o cara for convidado redirect para a barra /login

return redirect('/login');

}

return $next($request);

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Em Kernel a diferença do método $middleware para o $routeMiddleware é que sempre o $middleware rodará antes do código já no caso do $routeMiddleware só vai barrar em um controller especifico.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Vamos passar o nosso Autorizar.php do $middleware para o $routeMiddleware em nosso Kernel.php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

/\*\*

\* The application's global HTTP middleware stack.

\*

\* @var array

\*/

protected $middleware = [

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,

\App\Http\Middleware\EncryptCookies::class,

\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

\Illuminate\Session\Middleware\StartSession::class,

\Illuminate\View\Middleware\ShareErrorsFromSession::class,

\App\Http\Middleware\VerifyCsrfToken::class,

];

/\*\*

\* The application's route middleware.

\*

\* @var array

\*/

protected $routeMiddleware = [

'auth' => \App\Http\Middleware\Authenticate::class,

'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

//Dou um nome para ele e coloco um nome para classe

'autorizador' => \App\Http\Middleware\Autorizador::class,

];

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos adicionar um construtor em nosso ProdutoController.php para validar o nosso autorizador

<?php namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;

use Request;

use Validator;

use App\Produto;

use App\Http\Requests\ProdutoRequest;

use Auth;

class ProdutoController extends Controller

{

public function \_\_construct()

{

$this->middleware('autorizador');

}

public function lista()

{

//$produtos = DB::select('select \* from produtos');

if(Auth::guest())

{

return redirect('/login');

}

//Fazendo a listagem com a ORM(Eloquent)

$produtos = Produto::all();

return view('listagem')->with('produtos', $produtos);

}

public function mostra($id)

{

//$id = Request::route('id');

//$produto = DB::select('select \* from produtos where id = ?', [$id]);

$produto = Produto::find($id);

//return view('detalhes')->with('p', $produto[0]);

//Agora vai retornar só um produto sem array

return view('detalhes')->with('p', $produto);

}

public function novo()

{

return view('formulario');

}

public function remove($id)

{

//$id = Request::route('id');

$produto = Produto::find($id);

$produto->delete();

//return redirect('/produtos');

return redirect()->action('ProdutoController@lista');

}

public function adiciona(ProdutoRequest $request)

{

//pegar as informações do formulario

//$params = Request::all();

//$produto = new Produto($params);

//$nome = Request::input('nome');

//$quantidade = Request::input('quantidade');

//$valor = Request::input('valor');

//$descricao = Request::input('descricao');

//$produto->nome = Request::input('nome');

//$produto->quantidade = Request::input('quantidade');

//$produto->valor = Request::input('valor');

//$produto->descricao = Request::input('descricao');

//salvar no banco de dados

//DB::insert('insert into produtos (nome, quantidade, valor, descricao) values (?,?,?,?)', array($nome, $quantidade, $valor, $descricao));

//$produto->save();

//O validator ajuda a gente a fazer qualquer regra de validação

/\*$validator = Validator::make(

//qual campo a validar+ qual valor quer validar

['nome' => Request::input('nome')],

//campo obrigatorio | no minino 3 caracteres

['nome' => 'required|min:3']

); \*/

/\*

//validação com mensagens

if ($validator->fails()) {

$msg = $validator->messages();

dd($msg);

return redirect('/produtos/novo');

}\*/

Produto::create($request->all());

//com a função withInput() mantem os parametros da função anterior;

return redirect('/produtos')->withInput();

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Ajuste feito principal.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Controle de estoque</title>

<link rel="stylesheet" type="text/css" href="/css/app.css">

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" crossorigin="anonymous">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" crossorigin="anonymous">

</head>

<body>

<!-- Image and text -->

<nav class="navbar navbar-light fixed-top" style="background-color: rgba(77,88,193,1);">

<span class="navbar-brand">

<img src="data:image/svg+xml;base64," width="35" height="35" class="d-inline-block align-top" alt="">

CRUD - Laravel

</span>

<span class="navbar-text">

<a href="/produtos/novo" class="btn btn-outline-danger" style="color:white; border:1px solid white;">Novo</a>

<a href="/auth/logout" class="btn btn-outline-danger" style="color:white; border:1px solid white;">Sair</a>

</span>

</nav>

<div class="container" style="margin-top: 100px;">

<div class="row">

@yield('conteudo')

</div>

</div>

<!-- Latest compiled and minified JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/js/bootstrap.min.js" crossorigin="anonymous"></script>

</body>

</html>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Alterações no form\_login.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Controle de estoque</title>

<link rel="stylesheet" type="text/css" href="/css/app.css">

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" crossorigin="anonymous">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" crossorigin="anonymous">

</head>

<body>

<!-- Image and text -->

<nav class="navbar navbar-light fixed-top" style="background-color: rgba(77,88,193,1);">

<span class="navbar-brand">

<img src="data:image/svg+xml;base64," width="35" height="35" class="d-inline-block align-top" alt="">

CRUD - Laravel

</span>

<span class="navbar-text">

<a href="/auth/register" class="btn btn-outline-danger" style="color:white; border:1px solid white;">Criar Usuario</a>

</span>

</nav>

<div class="container" style="margin-top: 56em;">

<div class="row">

@if (count($errors) > 0)

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<div style="position: absolute; height: 100px; top: 35%; width: 100%; left: 0;">

<div style="position: relative; height: 100px; top: -58px;">

<div class="container">

<form action="/login" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Email</label>

<input name="email" class="form-control" value="{{ old('email') }}" />

</div>

<div class="form-group">

<label>Senha</label>

<input type="password" name="password" class="form-control" value="{{ old('password') }}"/>

</div>

<button class="btn btn-outline-primary btn-lg btn-block" type="submit">Login</button>

</form>

</div>

</div>

</div>

</span>

<span class="navbar-text">

</span>

</nav>

<div class="container" style="margin-top: 56em;">

<div class="row">

@yield('conteudo')

</div>

</div>

<!-- Latest compiled and minified JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/js/bootstrap.min.js" crossorigin="anonymous"></script>

</body>

</html>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Alterações feitas no LoginController.php para fazer os redirecionamentos corretos

<?php

namespace App\Http\Controllers;

use App\Http\Requests;

use App\Http\Controllers\Controller;

use Request;

use Auth;

class LoginController extends Controller

{

public function form()

{

return view('form\_login');

}

public function login()

{

//pegar credenciais apenas email e senha

$credenciais = Request::only('email', 'password');

//passar credenciais para o attempt(valida e entrar)

if (Auth::attempt($credenciais)) {

return redirect('/produtos');

}

return redirect('/login');

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Adicionando link no detalhes.blade.php

@extends('principal')

@section('conteudo')

<div class="container">

<h1>Detalhes do produtos {{ $p->nome }}</h1>

<ul>

<li>

Valor: {{ $p->valor }}

</li>

<li>

Descrição: {{ $p->descricao or 'não têm descrição'}}

</li>

<li>

Quantidade em estoque: {{ $p->quantidade }}

</li>

</ul>

<a class="btn btn-outline-primary" href="/produtos">Voltar a listagem</a>

</div>

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Adicionando link no formulário.blade.php

@extends('principal')

@section('conteudo')

@if (count($errors) > 0)

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<div class="container">

<form action="/produtos/adiciona" method="post">

<!-- //precise ser o name \_token para identificar -->

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control" value="{{ old('nome') }}" />

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control" value="{{ old('valor') }}"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input name="quantidade" class="form-control" value="{{ old('quantidade') }}"/>

</div>

<div class="form-group">

<label>Tamanho</label>

<input name="tamanho" class="form-control" value="{{ old('tamanho') }}"/>

</div>

<div class="form-group">

<label>Descrição</label>

<input name="descricao" class="form-control" value="{{ old('descricao') }}"/>

</div>

<button class="btn btn-primary" type="submit">Adicionar</button>

<a href="/produtos/" class="btn btn-primary" type="submit">voltar</a>

</form>

</div>

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Faremos agora o relacionamento entre tabelas

Vamos criar uma classe nova um modelo faremos isso pelo terminal com o comando e também criar uma migration

php artisan make:model Categoria

Crie a migration para tabela categorias 2017\_08\_22\_175821\_create\_categorias\_table.php

<?php

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateCategoriasTable extends Migration {

public function up()

{

Schema::create('categorias', function(Blueprint $table)

{

$table->increments('id');

$table->string('nome');

$table->timestamps();

});

}

public function down()

{

//quando quiser desfazer essa migration então drop na tabela categorias

Schema::drop('categorias');

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos rodar a nossa migration no terminal com o comando

php artisan migrate

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Em ProdutoController.php alteramos o método novo() e adicionamos o use para Categoria

use App\Categoria;

public function novo()

{

//vai listar todas as categorias para gente

return view('formulario')->with('categorias', Categoria::all());

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

No form.blade.php adicionamos o campo categoria abaixo do campo tamanho

<div class="form-group">

<label>Categoria</label>

<select name="categoria\_id" class="form-control" id="">

@foreach($categorias as $c)

<option value="{{ $c->id }}">{{ $c->nome }}</option>

@endforeach

</select>

</div>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Apenas agora faremos o insert na mão

insert into categorias (nome) values(‘Eletronico’);

insert into categorias (nome) values(‘Eletrodomestico’);

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos criar um novo metodo dentro da classe Produto.php

public function categoria()

{

//mostrar que ele pertence a um categoria

$this->belongsTo('App\Categoria');

}

Fazer a mesma coisa na classe Categoria.php

public function produtos()

{

//ele vai pertencer a varios 1 para varios

$this->hasMany('App\Produto');

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora precisamos mudar no BD o relacionamento entre categorias e produtos, para isso faremos a migration

php artisan make:migration adiciona\_relacionamento\_produto\_categoria

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora abriremos a migration e faremos as alterações

<?php

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class AdicionaRelacionamentoProdutoCategoria extends Migration

{

public function up()

{

Schema::table('produtos', function(Blueprint $table)

{

//chave estrangeira //criar com valor 1 por padrão

$table->integer('categoria\_id')->default(1);

});

}

public function down()

{

Schema::table('produtos', function(Blueprint $table)

{

//chave estrangeira

$table->dropColumn('categoria\_id');

});

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Rodar a migration

php artisan migrate

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora faremos a alteração na classe Produto para permitir a inserção do campo categorias

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Produto extends Model

{

//Caso nome da tabela fosse diferente da classe, $table+nomedatabela

protected $table = 'produtos';

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

//para evitar o erro do massAssignmentException all() do controller

//o fillable ignora token ou senha e só pega os campos no array

protected $fillable =

array('nome', 'descricao', 'quantidade', 'valor', 'tamanho', 'categoria\_id');

public function categoria()

{

//mostrar que ele pertence a um categoria

$this->belongsTo('App\Categoria');

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora alteramos o listagem.blade.php para exibir o campo nome de categoria

@extends('principal')

@section('conteudo')

<h1 class="text">Listagem de produtos</h1>

<table class="table">

@foreach($produtos as $p)

<tr class="{{ $p->quantidade <=1 ? 'danger': ''}}">

<td>{{ $p->nome }}</td>

<td>{{ $p->valor }}</td>

<td>{{ $p->descricao }}</td>

<td>{{ $p->quantidade }}</td>

<td>{{ $p->tamanho }}</td>

<td>{{ $p->categoria->nome }}</td>

<td>

<a href="/produtos/mostra/{{$p->id}}">

<span class="fa fa-search" aria-hidden="true">

</span>

</a>

</td>

<td>

<a href="/produtos/remove/{{$p->id}}">

<span class="fa fa-trash" aria-hidden="true">

</span>

</a>

</td>

</tr>

@endforeach

</table>

@if(old('nome'))

<div class="alert alert-success">

<!-- mantém os dados da requisição anterior -->

Produto {{old('nome')}} adicionado com sucesso!

</div>

@endif

@stop

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Para evitar erro de relacionamento faremos uma alteração na classe Categoria.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Categoria extends Model

{

public function produtos()

{

//ele vai pertencer a varios 1 para varios

return $this->hasMany('App\Produto');

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora tambem faça alteração na classe Produto.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Produto extends Model

{

//Caso nome da tabela fosse diferente da classe, $table+nomedatabela

protected $table = 'produtos';

//Para evitar os erros do campo data criação e atualização

public $timestamps = false;

//para evitar o erro do massAssignmentException all() do controller

//o fillable ignora token ou senha e só pega os campos no array

protected $fillable =

array('nome', 'descricao', 'quantidade', 'valor', 'tamanho', 'categoria\_id');

public function categoria()

{

//mostrar que ele pertence a um categoria

return $this->belongsTo('App\Categoria');

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora vamos trabalhar com seeds, vamos preparer o nosso DatabaseSeeder.php

<?php

use Illuminate\Database\Seeder;

use Illuminate\Database\Eloquent\Model;

use App\Categoria;

class DatabaseSeeder extends Seeder

{

public function run()

{

Model::unguard();

$this->call('CategoriaTableSeeder');

}

}

class CategoriaTableSeeder extends Seeder

{

public function run()

{

//Em run vou defenir o que vou inserir em meu BD

Categoria::create(['nome' => 'Eletrodomestico']);

Categoria::create(['nome' => 'Eletronica']);

Categoria::create(['nome' => 'Brinquedo']);

Categoria::create(['nome' => 'Esporte']);

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Agora rodaremos o nosso seeder pelo terminal com o commando

php artisan db:seed

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Um comando que pode deixar o serviço fora do ar enquanto damos manutenção no sistema para alguma correção de bug por exemplo usamos o seguinte comando no terminal

php artisan down

Para voltar ao normal é só digitar o comando inverso

php artisan up

Para comando do eloquente no terminal digite o seguinte comando

php artisan tinker

por exemplo

\DB::select(‘select \* from produtos’);

Ou

\App\Produto::all();

Para saber todas as rotas do aplicação digite o seguinte comando no terminal

php artisan route:list

Para mostrar a lista de comandos do artisan digite no terminal o comando

php artisan list

Quando precisar de ajuda em relação ao artisan digite no terminal o comando

php artisan help list

php artisan help route

etc

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**…or create a new repository on the command line**

echo "# Estoque-Laravel-CRUD-" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/diegocar448/Estoque-Laravel-CRUD-.git

git push -u origin master

**…or push an existing repository from the command line**

git remote add origin https://github.com/diegocar448/Estoque-Laravel-CRUD-.git

git push -u origin master