Artigo interessante:

<http://www.vedovelli.com.br/web-development/o-que-e-o-artisan/>

Pra instalar:

- Php e Apache é só instalar o Xampp

- Baixar o composer

- Usar o Visual Studio Code como IDE

Ctrl + P dá pra pesquisar os arquivos

Tem terminal interno

- no cmd

composer global require "laravel/installer"

laravel new nomeDoProjeto

ou

composer create-project --prefer-dist laravel/laravel nomeDoProjeto

php artisan serve (pra subir o servidor)

php artisan make:model NomeDaModel

php artisan make:model NomeDaModel -m (cria a migration com o -m)

php artisan make:model Model\NomeDaModel (cria a pasta e a model)

migration serve pra criar a tabela no banco

é bom criar uma nova pasta antes pra as models não ficarem na raiz da pasta app

php artisan (mostra todos os comandos disponíveis)

Se fizer algum arquivo de migrations errado e apagar tem que digitar esse comando:

Composer dump-autoload (pra não ter problema com o autoload)

O autoload serve pra carregar para a memória as classes que vão ser utilizadas (pra não ter que ficar colocando um monte de require\_once)

O método timestamps() serve pra colocar a data e hora automaticamente no registro da tabela. Se não for usar isso, tem que colocar na model: public $timestamps = false;

Se o apache parar, tem que achar o processo que tá usando a porta 423 ou 8080

No cmd dar o comando “netstat -a -n -o” e depois taskkil /pid “numero do pid”

Para não dar problema com o tamanho dos campos das tabelas, tem que ir na classe AppServiceProviders, dentro da pasta Providers e add isso (linha pra importação e chamada do método)

use Illuminate\Support\Facades\Schema;

public function boot()

{

Schema::defaultStringLength('191');

}

Para definir os dados de acesso ao banco: Pasta Config, arquivo Database.

Tem que colocar os dados da conexão no arquivo .env

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=agora\_vai\_laravel

DB\_USERNAME=root

DB\_PASSWORD=

Para criar as tabelas:

php artisan migrate

php artisan migrate:rollback (volta ao estado anterior)

Para instalar o AdminLTE (framework pra login)

<https://github.com/jeroennoten/Laravel-AdminLTE>

composer require jeroennoten/laravel-adminlte

Para publicar os assets (css, javascript, etc.)

php artisan vendor:publish --provider="JeroenNoten\LaravelAdminLte\ServiceProvider" --tag=assets --force

Fica dentro da pasta public/vendor/adminlte

Para inserir um Seeder (Seeder serve para popular as tabelas pra facilitar o desenvolvimento)

Fica dentro de database/seeds

php artisan make:seed UsersTableSeeder

Esse comando apenas cria a classe UsersTableSeeder dentro da pasta database/seeds a lógica precisa ser implementada.

- Tem que dar um import (no php é “use”) pra usar a classe da model que vai ser criada na tabela. (neste caso a classe user está na raiz de app)

use App\User;

<?php

use Illuminate\Database\Seeder;

use App\User;

class UsersTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

User::create([

'name' => 'Guilherme Legramante',

'email' => 'guilhermelegramante@gmail.com',

'password' => brcrypt('stratocaster'),

]);

}

}

Para funcionar tem que chamar na classe DatabaseSeeder

$this->call(UsersTableSeeder::class);

Para criar os seeds

php artisan db:seed

Para gerar as Views do login com o AdminLTE

**php artisan make:adminlte**

o padrão do Laravel é o php artisan make:auth

São criadas as pastas auth e layouts dentro de resources/views

Também já criadas as rotas em web.php

As rotas servem para fazer direcionamento entre as páginas

Na classe api.php é definida a rota que vai direcionar para o início da aplicação

Essa rota é criada:

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

Está direcionando pra home, só que na controller exige que passe pela autenticação:

public function \_\_construct()

{

$this->middleware('auth');

}

Após isso, já está criada toda a parte de login

Para criar o arquivo de configuração do template AdminLTE:

php artisan vendor:publish --provider="JeroenNoten\LaravelAdminLte\ServiceProvider" --tag=config

É criado um arquivo “adminlte.php” dentro da pasta config.

'title' => 'Sistema de Saldo',

'title\_prefix' => '', (aqui é pra colocar prefixo na aba do navegador)

'title\_postfix' => '', (aqui é pra colocar o sufixo na aba do navegador)

Nesse arquivo tem várias configurações da exibição do template.

Pra trocar os ícones tem que buscar no site <https://fontawesome.com/icons> e trocar

'icon' => 'home',

Cada item de menu é um array

Criar uma pasta site dentro de resources/views (para a parte do site da aplicação onde os dados serão exibidos)

Dentro da pasta site criar uma pasta home e dentro desta pasta um arquivo “index.blade.php” (esse arquivo terá o layout da página inicial do site)

Criar uma pasta admin dentro de resources/views (para a parte do admin da aplicação)

Depois disso dá pra apagar o arquivo inicial “welcome.blade.php” (no curso ele copia o conteúdo da welcome pra index que foi criada)

Agora tem que arrumar a rota pra direcionar pro arquivo novo.

No arquivo web.php dentro da pasta routes a rota / deve ficar assim:

Route::get('/', 'SiteController@index');

Por enquanto tem que deixar comentada pra não dar erro, pois ainda não foi criada a controller que foi chamada nessa rota.

Para criar a controller:

php artisan make:controller SiteController (cria direto na raiz)

php artisan make:controller Site/SiteController (cria uma pasta site e o arquivo dentro dessa pasta)

Se não aparecer a classe, tem que clicar no ícone de atualizar no visual studio

Criar o método index dentro da controller:

class SiteController extends Controller

{

public function index()

{

return view('site.home.index');

}

}

Esse método vai chamar a view que está dentro da pasta home e a rota que foi criada vai funcionar

Aí dá pra descomentar a linha

Route::get('/', 'SiteController@index');

Da rota do arquivo web.php

Tem que mudar o endereço da rota para

Route::get('/', 'Site\SiteController@index');

Por causa da pasta Site que foi criada

Deletar a HomeController pra deixar mais organizado e criar uma pasta Admin dentro de Controllers

Criar a AdminController:

php artisan make:controller Admin\AdminController

Criar o método index na controller que foi criada:

class AdminController extends Controller

{

public function index()

{

return view('admin.home.index');

}

}

Criar as rotas no web.php:

$this->get('admin', 'Admin\AdminController@index')->name('admin.home');

$this->get('/', 'Site\SiteController@index')->name('home');

Auth::routes();

Agora tem que criar a view:

Criar as pastas admin\home dentro de resources\views e copiar o arquivo [home.blade.php](http://home.blade.php) e botar lá.

Renomear esse arquivo para index.blade.php

Fazer essa modificação nas rotas no web.php para exigir que passe por autenticação o grupo de rotas:

$this->group(['middleware' => ['auth'], 'namespace' => 'Admin'], function(){

$this->get('admin', 'AdminController@index')->name('admin.home');

});

Se remover essa linha a seguir(no arquivo da view “index.blade.php”), fica com o título padrão que foi colocado no arquivo de configuração adminlte.php

@section('title', 'AdminLTE')

Mudar a rota do login para \admin:

Dentro de app\http\Auth o arquivo LoginController.php e no RegisterController.php tem que trocar a propriedade redirectTo que está na rota \home para \admin

Fazer a mesma coisa para o RedirectAuthenticated.php que tá na pasta Middleware dentro de Controller.

Para fazer a tradução:

Clonar o pacote <https://github.com/caouecs/Laravel-lang.git> em algum lugar

Copiar a url, clicar com o direito e escolher a opção “Git bash here”

Dentro do terminal git dar o comando git clone <https://github.com/caouecs/Laravel-lang.git>

Colar a pasta pt-br dentro de resources\lang

Mudar a propriedade locale e fallback\_locale no arquivo app.php dentro de app\config

Para mudar as mensagens do login tem que acessar o arquivo auth.php dentro da pasta pt-BR

Exportar o arquivo de tradução:

php artisan vendor:publish --provider="JeroenNoten\LaravelAdminLte\ServiceProvider" --tag=translations

o arquivo é criado dentro de resources\lang\vendor\admilte\pt-br

o arquivo adminlte.php tem as configurações dos textos que são exibidos

Para customizar as views do adminLTE:

php artisan vendor:publish --provider="JeroenNoten\LaravelAdminLte\ServiceProvider" --tag=views

é criado dentro de resources\views\vendor\adminlte

O Blade Template Engine é parte do framework PHP Laravel. Blade permite que você use uma sintaxe discreta para escrever estruturas de controle PHP e mostrar os dados. Use esta tag para perguntas sobre problemas com a sintaxe e o uso do Blade. (usa o @ antes dos comandos tipo if, else, for, etc.)

Para modificar o menu é no arquivo config\adminlte.php

Criar mais uma pasta “balance” dentro de resources\views\admin (para ficar todas as views relacionadas a gerenciamento de saldo)

O Visual Studio tem um terminal integrado “View\Integrated Terminal”

Criar o controller:

php artisan make:controller Admin\BalanceController

Essa controller vai ser a responsável por exibir o saldo do usuário

class BalanceController extends Controller

{

public function index()

{

return view('admin.balance.index');

}

}

Agora tem que criar esse index dentro da pasta resources\views\admin\balance:

Criar arquivo index.blade.php

Copiar a estrutura do arquivo index.blade.php que está dentro da pasta resources\views\admin\home

@extends('adminlte::page')

@section('title', 'Saldo')

@section('content\_header')

<h1>Saldo</h1>

<ol class="breadcrumb">

<li><a href="">Dashboard</a></li>

<li><a href="">Saldo</a></li>

</ol>

@stop

@section('content')

<p>Meu Saldo</p>

@stop

Agora tem que acertar a rota para que seja renderizada essa view (na pasta routes, arquivo web.php):

$this->group(['middleware' => ['auth'], 'namespace' => 'Admin'], function(){

$this->get('balance', 'BalanceController@index')->name('admin.balance');

$this->get('admin', 'AdminController@index')->name('admin.home');

});

Foi colocada a rota do ‘balance’ (saldo) na primeira linha do grupo de rotas

Arrumar a url no arquivo adminlte.php dentro da pasta config:

'submenu' => [

[

'text' => 'Saldo',

'url' => 'admin/balance',

'icon' => 'credit-card-alt',

],

[

'text' => 'Histórico',

'url' => 'admin/historic',

'icon' => 'history',

]

Para ficar com admin antes na url

Arrumar o arquivo de rotas web.php

$this->group(['middleware' => ['auth'], 'namespace' => 'Admin', 'prefix' => 'admin'], function(){

$this->get('balance', 'BalanceController@index')->name('admin.balance');

$this->get('/', 'AdminController@index')->name('admin.home');

});

Colocando o prefixo admin, todas as rotas vão ter o admin antes, aí não precisa mais passar o admin em cada rota e na rota que era admin vira só “/”.

Ajeitando o layout da tela de exibição do saldo (arquivo index.blade.php dentro de resources\views\admin\balance

@section('content')

<div class="box">

<div class="box-header">

Header

</div>

<div class="box-body">

Body

</div>

</div>

O adminLTE tem um site tipo o primefaces pra mostrar os componentes.

<https://adminlte.io/>

No curso o cara vai nesse site e inspeciona o elemento que ele deseja pra copiar o código. No exemplo ele pega o Bounce Rate.

Com F2 no inspetor do navegador, ele abre toda a div aí com Ctrl + A dá pra selecionar tudo

O layout da tela do saldo ficou assim:

@extends('adminlte::page')

@section('title', 'Saldo')

@section('content\_header')

<h1>Saldo</h1>

<ol class="breadcrumb">

<li><a href="">Dashboard</a></li>

<li><a href="">Saldo</a></li>

</ol>

@stop

@section('content')

<div class="box">

<div class="box-header">

<a href="" class="btn btn-primary"><i class="fa fa-cart-plus" aria-hidden="true"></i> Recarregar</a>

<a href="" class="btn btn-danger"><i class="fa fa-cart-arrow-down" aria-hidden="true"></i> Sacar</a>

</div>

<div class="box-body">

<div class="small-box bg-green">

<div class="inner">

<h3>R$ 90,00</h3>

</div>

<div class="icon">

<i class="ion ion-cash"></i>

</div>

<a href="#" class="small-box-footer">Histórico <i class="fa fa-arrow-circle-right"></i></a>

</div>

</div>

@stop

Agora tem que exibir o saldo na tela.

Implementação do método na model:

Tem que relacionar a classe User com a classe Balance. User tá na raiz da pasta app. Balance tá dentro de app/model

use App\Models\Balance;

Tem que dar esse use na classe que vai ser usada.

public function balance()

{

return $this->hasOne(Balance::class);

}

Como o relacionamento é 1:1 (um usuário só tem um saldo) é só colocar essa linha no método que já consegue acessar do banco.

Agora tem que codificar na controller (app/Http/Controllers/Admin/BalanceController.php)

Tem que pegar os dados do usuário logado.

O dd é utilizado pra debug. (se voltar lá na página e atualizar, vai mostrar os dados do usuário num JSON)

public function index()

{

dd( auth()->user());

return view('admin.balance.index');

}

Por exemplo, se quiser pegar o nome, é só fazer auth()->user()->name

dd( auth()->user()->balance);

Pega o saldo

dd( auth()->user()->balance()->get());

Pega o objeto inteiro do banco

O saldo vai retornar nulo pq ainda não tem nada persistido.

O método fica assim:

public function index()

{

$balance = auth()->user()->balance;

$amount = $balance ? $balance->amount : 0;

return view('admin.balance.index', compact('amount'));

}

Vai pegar o montante (amount) e passar para a view através do compact

$amount = $balance ? $balance->amount : 0;

Essa linha é pra retornar zero se não tiver nada (se for nulo o valor no banco)

Agora tem que exibir o amount lá na view (resourcer/views/admin/balance/index.blade.php)

<div class="box-body">

<div class="small-box bg-green">

<div class="inner">

<h3>R$ {{$amount}}</h3>

</div>

<div class="icon">

<i class="ion ion-cash"></i>

</div>

<a href="#" class="small-box-footer">Histórico <i class="fa fa-arrow-circle-right"></i></a>

</div>

Como a variável já foi enviada pelo método compact, agora é só chamar ela na view.

<h3>R$ {{ number\_format($amount, 2, ",", "")}}</h3>

Dá pra usar o método do php number\_format pra formatar. Nesse caso passa a variável, quantas casas quer mostrar após o separador (2), qual o caracter é vai ser o separador (,) e o último parâmetro é para a casa de milhar, se quiser colocar alguma coisa como espaço ou outra coisa, nesse caso não é colocado nada.

Pra inserir um registro no phpMyAdmin, é só ir na coluna desejada, depois na aba “insere”

Funcionalidade de incremento do saldo:

- Ir na view “index.blade.php” dentro de resources/views/admin/balance

<div class="box-header">

<a href="{{ route('balance.deposit')}}" class="btn btn-primary"><i class="fa fa-cart-plus" aria-hidden="true"></i> Recarregar</a>

<a href="" class="btn btn-danger"><i class="fa fa-cart-arrow-down" aria-hidden="true"></i> Sacar</a>

</div>

Colocar uma nova rota no href do botão “Recarregar”

- Ir no arquivo web.php (routes/web.php) para criar a rota:

$this->group(['middleware' => ['auth'], 'namespace' => 'Admin', 'prefix' => 'admin'], function(){

$this->get('balance/deposit', 'BalanceController@deposit')->name('balance.deposit');

$this->get('balance', 'BalanceController@index')->name('admin.balance');

$this->get('/', 'AdminController@index')->name('admin.home');

});

Adicionar essa rota ao grupo de rotas

- Criar o método de depósito na BalanceController:

public function deposit()

{

return view('admin.balance.deposit');

}

- Criar a view que o método deposit está retornando

Criar o arquivo deposit.blade.php dentro de resources/views/admin/balance

@extends('adminlte::page')

@section('title', 'Nova Recarga')

@section('content\_header')

<h1>Fazer Recarga</h1>

<ol class="breadcrumb">

<li><a href="">Dashboard</a></li>

<li><a href="">Saldo</a></li>

<li><a href="">Depositar</a></li>

</ol>

@stop

@section('content')

<div class="box">

<div class="box-header">

<h3>Fazer Recarga</h3>

</div>

<div class="box-body">

<form method="POST" action="{{ route('deposit.store') }}">

<div class="form-group">

<input type="text" placeholder="Valor Recarga" class="form-control">

</div>

<div class="form-group">

<button type="submit" class="btn btn-success">Recarregar</button>

</div>

</form>

</div>

</div>

@stop

Aqui foi criado o form da página

action="{{ route('deposit.store') }}" Agora tem que criar essa rota

- Criar a rota que o form da página criada chama no action:

$this->group(['middleware' => ['auth'], 'namespace' => 'Admin', 'prefix' => 'admin'], function(){

$this->post('deposit', 'BalanceController@depositStore')->name('deposit.store');

$this->get('deposit', 'BalanceController@deposit')->name('balance.deposit');

$this->get('balance', 'BalanceController@index')->name('admin.balance');

$this->get('/', 'AdminController@index')->name('admin.home');

});

OBS: Pode ser a mesma URL (1º Parâmetro) porque uma é o método post e outra é o get.

- Criar um campo oculto pra não dar erro quando enviar o form, pois precisa do token

<form method="POST" action="{{ route('deposit.store') }}">

{!! csrf\_field() !!}

<div class="form-group">

- Implementar o método DepositStore que a rota direciona:

Para testar a requisição (saber se tá chegando certo)

public function depositStore(Request $request)

{

dd($request->all());

}

Faltou colocar a propriedade name no campo de formulário de depósito, arquivo deposit.blade.php (dentro de resources/views/admin/balance)

<input type="text" name="value" placeholder="Valor Recarga" class="form-control">

- Implementar o método depositStore (que está na BalanceController) esse método é o que fará a recarga:

- Criar o método de depósito na model Balance: