LOWCOST

Projeto portal Low Cost _ documentação

1. Montando o Ambiente

Antes de criar o ambiente para o na sua máquina local, verifique se possui :

No Windows:

- PHP 8.2 ou 8.3 configurado no PATH da máquina
- Docker Desktop
- Composer (https://getcomposer.org)
- Node Js para uso do @Vite.

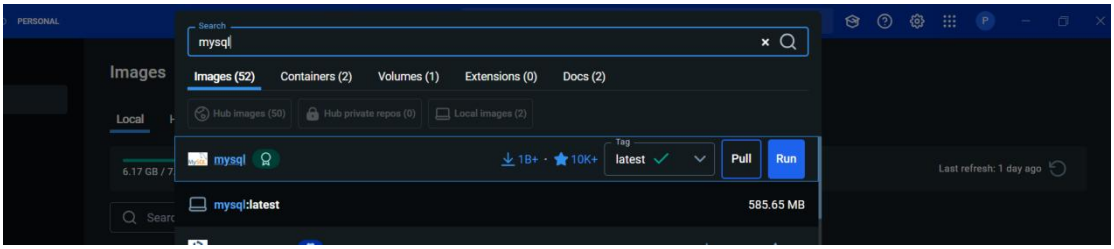
No Linux ou WSL Ubuntu:

- Docker Desktop

No Linux ou WSL para criar ambiente docker basta executar no shell curl -s https://laravel.build/name_of_project | bash, ele criara todo ambiente necessário no docker, incluindo acesso a porta 3306 do Mysql e irá montar dentro do ambiente linux o path com o container docker e ao alterar o código no ambiente externo ele será sincronizado com o ambiente docker. ***Pule para página 4 em diante.***

-> Criando ambiente no Docker

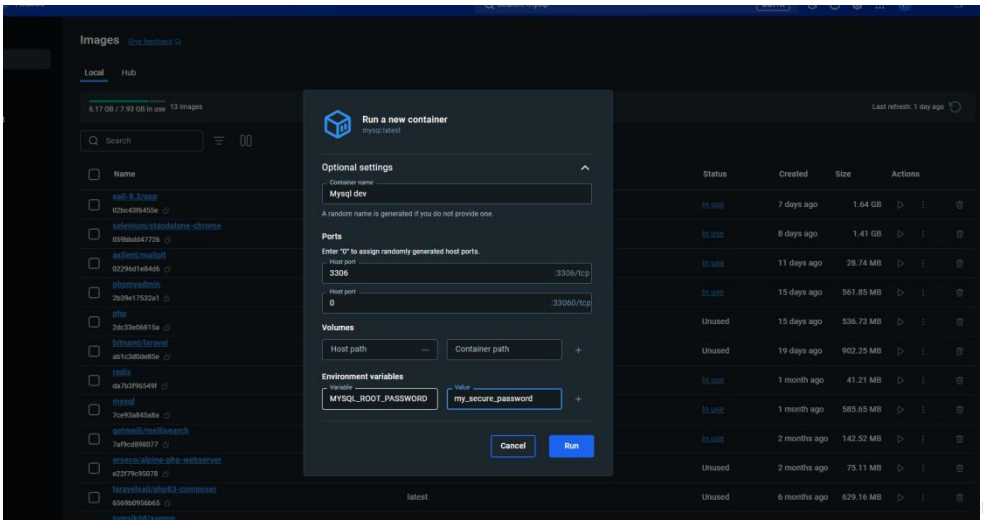
Criando o ambiente com o docker desktop aberto acessse o menu lateral Imagens e na barra de busca digite mysql.



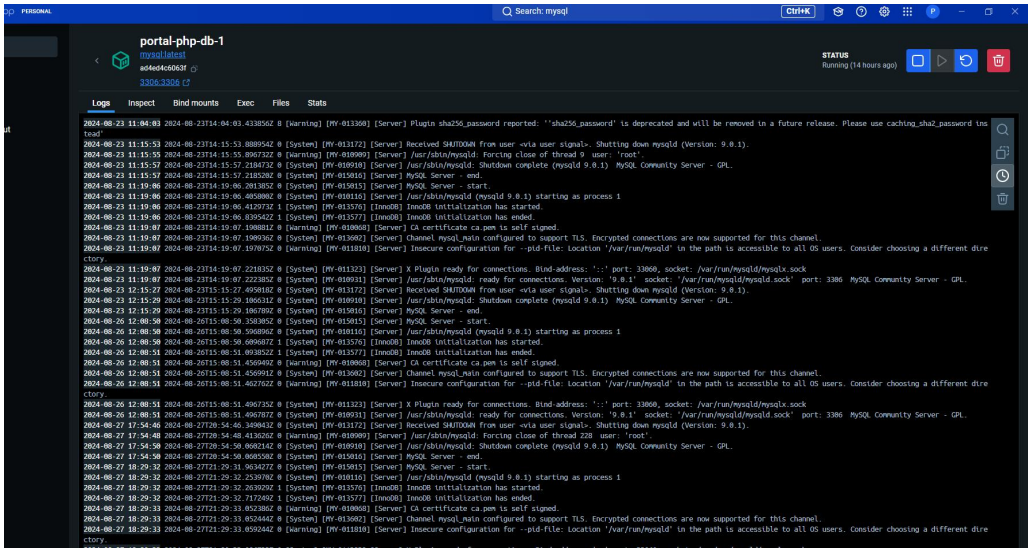
Na imagem Mysql:latest clique em pull, o docker ira copiar a imagem na sua máquina.

Com a imagem salva vamos configurar o container, na pagina de imagens no docker desktop procure a image Mysql, e clique em play, e abrira um pop -up de configurações.

No painel de configurações vamos preencher os itens, container name, ports, environment variables, como na imagem abaixo.

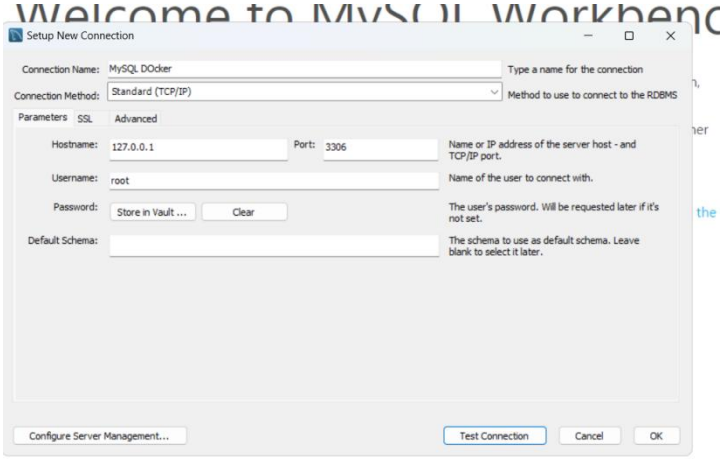


ntainer rodando e o ícone dele estará na cor verde, se der algum erro o icone ficara na cor laranja ou se o erro for grave exemplo não teve permissão para bindar a porta o status dele será exited (1).



-> Configurando o acesso via Mysql Workbench ao docker mysql.

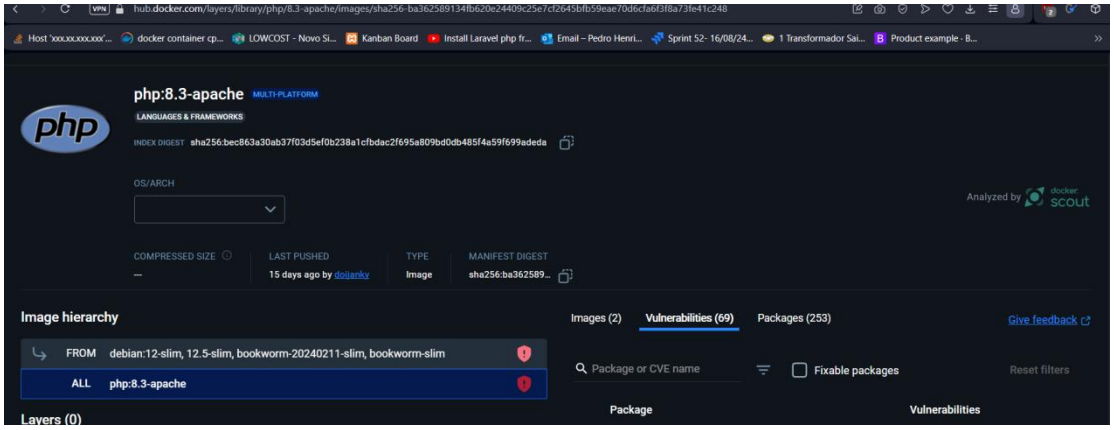
Abra o Mysql Workbench ou outro gestor de banco de dados , na tela de configuração de conexão



Após preencher os dados de conexão clique em testar conexão, se aparecer a tela com a mensagem “Connection Warning” clique em “Continue Anyway”, esse erro é porque a versão de driver do Mysql Workbench é menor do que versão da imagem, algumas funcionalidades podem não funcionar.

-> Iniciando projeto Laravel.

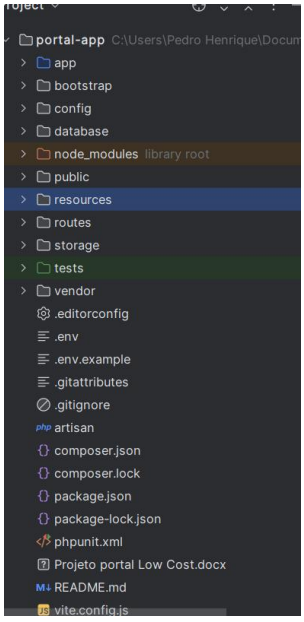
Para iniciar o projeto Laravel fora do docker ou seja o servidor e arquivos estarão no seu computador, podemos utilizar tanto o Laravel installer ou o composer, o installer vai sempre utilizar a última versão, no caso do Laravel 11 ele necessita PHP 8.3 que foi lançado recentemente, mas tem o seguinte problema ainda não tem versão dos servidores Apache ou Nginex estáveis as que existem Xampp, Mampp possuem algumas vulnerabilidades.



Por esse motivos ultilizaremos o Laravel V10 que necessita do PHP 8.2 que tem tanto para xampp como docker e está estavel.

Com o composer instalado crie uma pasta onde desejar eu sugiro no ‘C:/ ‘ porque iremos sincronizar com docker mais a frente e rode no CMD o comando:

composer create-project laravel/laravel:10.0^ portal-app.



Com o Laravel instalado você terá esta estrutura de arquivos,
ainda com o CMD aberto monte a posta do seu projeto e execute

```
npm run dev
```

Ele instalara os recursos para uso do @Vite e irá aparecer no seu projeto a pasta node_modules assim como está nesta imagem.

-> Instalando extensões no VSCode

No vs code iremos instalar umas exetensões para facilicar trabalhar com PHP / Laravel, atém do VS Code é possivel utilizar o PHPStorm e/ou Netbeans.

-> Laraphense (Laravel PHP Inteliphense) : publisher Porifa
-> Laravel blade : publisher amirmarmul
-> Laravel Extra Intellisense : publisher amiralizadeh9480
-> PHP Intelephense : publisher bmewburn
-> Docker : publihser ms-azuretools

No PHPStorm não precisa das extensões ele vem configurado, mas ele permite usar sem registro somente por 30 dias.

1.2 Configurando e Conehcendo o Laravel

-> Configurando o Laravel

Para nosso projeto rodar temos que configurar no Laravel, o banco de dados isso é feito editando dois arquivos, o .ENV e os outros na pasta config.

No arquivo .ENV vamos alterar o APP_NAME, e os campos DB_* como exemplo:

```
APP_NAME=Low_Cost_Portal
APP_ENV=local
APP_KEY=base64:VM+R5RtdbJiUMcacaDHMeQylqjQj4yqLxUd0jbmdmHg=
APP_DEBUG=true
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=www_portal
DB_USERNAME=root
DB_PASSWORD=''
```

Ja na pasta config iremos acessar o aquivo app.php e database.php. No app.php, alteraemos a variavel “name” ,“timezone” e a “locale” e ficara assim

```
'name' => env('APP_NAME', 'low Cost Portal'),
```

```
'timezone' => 'America/Sao_Paulo',
'locale' => 'pt_BR',
```

Em config/database.php iremos alterar os campos , “default” e “connections”=>”mysql”.

```
'default' => env('DB_CONNECTION', 'mysql'),
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'www_portal'),
    'username' => env('DB_USERNAME', 'root'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
```

Para evitar problemas ao migrar o ambiente vamos limitar o número de caracteres que podem ser salvos no banco de dados, se no servidor houver um mysql mais antigo não haverá erro. Na pasta app/Providers abra o arquivo AppServiceProvider.php e procure o metodo boot(), dentro ele coloque

```
Schema::defaultStringLength(191)
```

Veja se no inicio do arquivo depois da declaração de namespace foi importado Illuminate\Support\Facedes\Schema.

LOWCOST

-> Iniciando o Laravel

Com seu ambiente configurado abra sua IDE no projeto e abra o terminal (cmd ou powershell) e digite **php artisan serve --port=9000**

Se a saída do comando acima for :

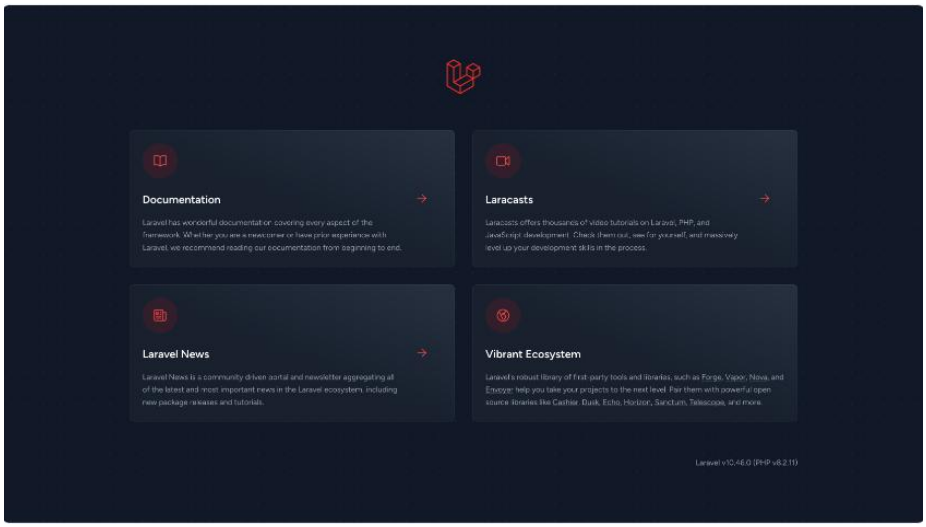
WARN Failed to listen on 127.0.0.1:9000 (reason: ?).

WARN Failed to listen on 127.0.0.1:9000 (reason: ?).

No terminal de a combinação de teclas para parar o processo CTRL + C até parar.

E rode o comando **php -S localhost:8080 -t public**

O servidor php será iniciado na pasta public do laravel, abra o seu navegador na url localhost:8080 e você vera pagina de Welcome do Laravel.



-> Paginas HTML do projeto

Antes de começar a criar rotas e controllers no Laravel, vou explicar como colocar sua pagina html no framework, o Laravel ele vem com padrão dois gestores de templates o breeze e o blade para trabalhar com paginas html, ele possui o jetstram que permite trabalhar com Vue, react.

As paginas deste projeto estamos utilizando Boostrap 5.5 via CDN (Content Delivery Network) ou seja os arquivos css e js do boostrap não são salvos localmente.

LOWCOST

-> Onde colocar as Webpages e assets

As paginas html devel ser renomeadas mantendo o nome da pagina sem espaço ou caracteres especiais e adicionadas ao nome a extensão ".blade.php", e elas são salvas na pasta resources/views, pode se criar subpastas na pasta views e colocar as paginas lá.

Os assets (imagens, logos, etc) são colocadas na pasta public, para serem acessados quando a view for renderizada.

-> Migrations e Seeders

O Laravel tem o recurso de criar de forma automatizada tabelas e a dados para teste, para isso basta acessar a pasta database/migrations e verá os aquivos que ja vem com o Laravel. No aqruivo 2014_10_12_000000_create_users_table.php e na dentro da função up() edite os campos da tabela "users" a ser criada. OBS todos os campos adicionados no arquivo de migrations devem ser referenciados na model.

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->char('active')->default(1);
    $table->string('level')->default('user');
    $table->string('type')->default('normal');
    $table->string('empresa')->nullable();
    $table->rememberToken();
    $table->timestamps();
});
```

Após modificar o migrations abra em app/models/User.php e iremos adicionar os campos acima a serem trazidos pela model. Na variável $fillable , colocaremos os campos que queremos que sejam trazidos quando for consultados os dados de um ou mais usuários.

```
protected $fillable = [
    'name',
    'email',
    'password',
    'active',
    'level',
    'type'
];
```

Acesse em database/seeders/DatabaseSeeder.php e descomente o conteudo que está na função run().

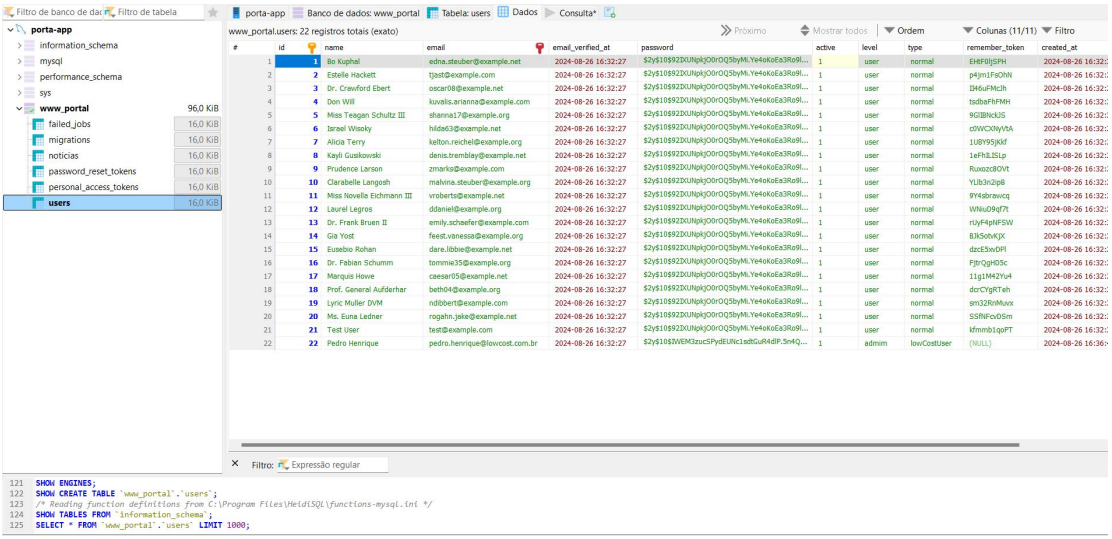
Deverá ficar assim:

```
public function run(): void
{
    \App\Models\User::factory(20)->create();  //qtde de usuarios a serem criados
```

```
    \App\Models\User::factory()->create([
        'name' => 'Test User',
        'email' => 'test@example.com',
    ]);
}
```

Agora abra o CMD e digite php artisan migrate --seed, o docker mysql deve estar rodando.
Se a tabela users não existir ele ira perguntar no prompt se gostaria de criar a tabela.

Após execução do migrate e seed, abra o Mysql Workbench e veja quais tabelas foram criados no seu banco de dados deve ficar assim:



-> Rotas

O Laravel trabalhal com rotas e middleware além do MVC (Model View Controller), os aquivos de rotas ficam na pasta routes/ para criar ou alterar a resposta de alguma rota inicialmente usaremos o arquivo web.php, note que a versão 10 suporta roteamento para Api's via arquivo api.php.

A rota que vem no arquivo é '/' ou seja quando acessar o localhost:8080 ele vai interceptar o request e devolver a view welcome. Se a view estiver em um subpasta para retorna -la usamos **nome_da_pasta. nome_da_view**, automaticamente o rendereizer do blade entente que o '.' é um / na hora de acessar a view.

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

É possivel setar atributos das rotas como nomes e chamar controllers para realizaram ações antes de retornar a view, veremos a diante.

-> Laravel Gate e Policies.

Antes de continuar com as rotas, vamos configurar perfis de acesso que podem acessar rotas e funções nas views.

Acessa a pasta app/providers e abra o arquivo AuthServiceProvider dentro da função boot iremos utilizara a classe Gate, ela possui o controle de politicas, e um dos atributos dentro do Gate é a permissão chamada de can.

Importe no arquivo use Illuminate\Support\Facades\Gate; e use App\Models\User;

E dentro do boot(), colocaremos a chamda:

```
$this->registerPolicies();

Gate::define('level' , function(User $user){
        return $user->level=='admin';
        });

Gate::define('active, function(User $user){
        return $user->active=='1';
        });
```

-> Middlewares e niveis de acesso

Ainda em Routes/web.php, os grupos de middleware servem para controles de acesso a rotas e dados, por exemplo no nosso portal o visitante consegue acessar a pagina index e a login. Outras paginas como dashboard, cadastro somente se estiver logado e tiver a permissão de acesso.

No Laravel, o “convidado” refere-se a um usuário que não está autenticado ou logado na aplicação. É um conceito integrado fornecido pelo sistema de autenticação do Laravel. O middleware “convidado” e os protetores de autenticação são usados para lidar com solicitações de usuários não autenticados.

Veja como funciona o conceito de “convidado” no Laravel:

Middleware convidado:

O Laravel fornece um middleware chamado “convidado” que pode ser aplicado a rotas ou controladores. Este middleware permite que apenas usuários não autenticados acessem essas rotas. Se um usuário tentar acessar uma rota protegida pelo middleware “convidado” enquanto estiver sendo autenticado, ele será redirecionado para um local diferente, normalmente a página inicial do aplicativo.

Protetores de autenticação:

O sistema de autenticação do Laravel inclui protetores de autenticação que determinam como os usuários são autenticados. Por padrão, o Laravel fornece um protetor “web” que lida com a autenticação do usuário para aplicações baseadas na web.

Como criar rotas e grupos, para isso usamos a função Route::group assim como no exemplo a baixo.

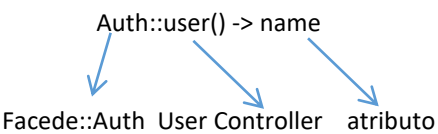
```
/** se o usuario não esta logado envia para o login */
Route::group(['middleware' => 'guest'], function () {
   /** se o usuario não esta logado envia para o login page */
   Route::get('/login', [UserController::class, 'showLoginForm'])->name('login');
   /** ao postar o form login na login.blade.php os dados request são enviados para o controller User na função loginPost */
   Route::post('/login', [UserController::class, 'loginPost']);
});
```

Como visto dentro do grupo as rotas, e ao invés de retornar a view são passadas em forma de um array a classe e o metodo que será executado, é possivel também utilizar validações dentro do middleware e dentro de cada rota. Como no exemplo abaixo.

```
/** acesso somente após login */
Route::group(['middleware' => 'auth'], function () {

  /** acesso a rota dashboard após login */
  Route::get('/dashboard', function ()
  {
    /** se user pertence a low cost envia para o dashboard @lowcost */
    if(Auth::user()->type=='lowCostUser')
    {
      return view('lowcost.dashboard');
    }
    /** se user não pertence a low cost envia para o dashboard @empresa */
    else
    {
      return view('auth.dashboard');
    }
  })->middleware(['auth', 'verified'])->name('dashboard');

  /** se user pertence a low cost cria roa  para o cadastra noticia*/
  Route::get('/add-news' , function(){
    if(Auth::user()->type=='lowCostUser')
    {
      return view('lowcost.addnews');
    }
  })->name('add-news');

});
```

-> Como a view interage com as rotas, dentro da view, as tags href é possivel usar o modo simples exemplo: Home , mas como muitos acesso são em níveis e pastas diferentes para usuario com acesso difretentes não acessem views e funções que não lhe pertencem usamos assim: Home.

A Facede Auth, é responsavel por “guardar” os dados de usuario logado, e/ou de outros controllers , então por ela podemos acessar dados de controllers diferentes utilizando a chamada:



-> Criando Helpers

Helpers são arquivos utilizados para uma função especifica ou varias como formatar datas, dar saudação em um ou mais view, para criar um helper é facil eu recomendo criar ele como uma classe com metodos estaticos, e criar um namespace.

Onde colocar app/, crie seu arquivo ele ficará assim

```
namespace App;
```

```
class Helpers
{
    /****/
}
```

Após criado seu arquivo Helpers iremos editar o arquivo composer.json, na chave autoload / PSR-4, coloque “App\\”:”App//”

```
"autoload": {
    "psr-4": {
        "App\\": "app/",
        "Database\\Factories\\": "database/factories/",
        "Database\\Seeders\\": "database/seeders/"
    }
```

E basta depois executar **composer dump -o**, depois que o composer rodar, execute **php artisan apitimize:clear**

Para acessar seu Helper em uma view basta **{!! \App\Helpers::greeting(); !!}**

-> Como criar Models e Controllers

No terminal basta digitar:

php artisan make:controller nome_do_controller

php artisan make:model nome_da_model

Os arquivos criados pelo artisan ja são automaticamente extensões das classes Controller e Model.

LOWCOST

-> Criando tabelas e migrations com Artisan/Laravel

Para criar migrations basta executar o comando

```
php artisan make:migration criar_tabela_nome--create=nome_table
```

Exemplo:

```
php artisan make:migration criar_tabela_alunos --create=aluno
```

A estrutura de uma migration

A classe criada é uma Migration que possui a estrutura básica de métodos e pacotes já definidas pelo Laravel, essa classe ficará responsável por criar uma tabela chamada alunos no banco de dados:

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CriarTabelaAlunos extends Migration
{
    public function up()
    {
            Schema::create('aluno', function (Blueprint $tabl
     {
                $table->increments('id');
                    $table->string('nome', 80);
                    $table->string('email', 80);
                    $table->integer('registro');
                    $table->timestamps();
            });
    }

    public function down()
    {
            Schema::dropIfExists('aluno');
    }
}
```

-> Executando uma migration

Com a classe responsável por executar a migration é possível executar o seguinte comando em um terminal:

php artisan migrate

O comando acima será responsável por executar as classes que estiverem localizadas no pacote migration/database/migrations/, e caso não seja identificado nenhum erro será criada a tabela Alunos na base de dados que a aplicação estiver conectada. A imagem abaixo demonstra o banco de dados após a execução da migration:

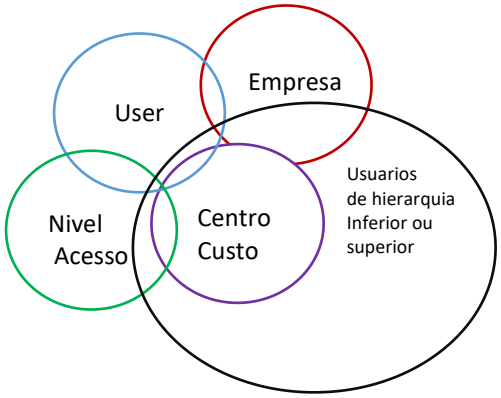
A tabela chamada migrations armazena o nome das classes de migration e cada vez que for criada uma nova seu nome será inserido nessa tabela.

2. Projeto Portal

Após configurar o ambiente de desenvolvimento vamos falar do portal.

-> Relacionamento user

O usuario cadatrado na tabela user tem relaciomento com Empresa, centro de custo e nível de acesso, e hierarquia. Ou seja um usuario com nivel 1 pode acessar os dados de um usuario nivel 2 que esteja dentro de um mesmo centro de custo e empresa**.



-> User_x_Empresa

O portal tem fluxos para tipos de usuarios, no tipo de usuario lowCostUser relacionado com a empresa Low Cost acessara as paginas e funções definidas a low Cost ou seja CRUD de noticias e dashboard com dados de noticias. E usuarios dentro de um empresa tenha acesso definido por nível, centro de custo e empresa

-> Sessão

Alterar as configurações do sistem para tempo máximo de sessão de 180 minutos ou 3 horas, para isto acesse config/session.php e adicione nas variaveis

‘Lifetime’ => 180,

‘expire_on_close’ => true

‘encrypt' => true,

Se a variavel driver estiver configurada para database altere para

'driver' => env('SESSION_DRIVER', 'file'),

LOWCOST

-> Login

Ao acessar a tela de login o usuario deve preencher a empresa que está inserido assim como email e senha. Se usuario estiver na empresa lowcost ele sera redirecionando para grupo de paginas da LowCost, aramazenadas em resources/views/lowcost.

Se o usuario for de outra empresas ele será redirecionado para outras views onde será carregado os dados referentes a sua empresa e centro de custo.

-> Paginas Publicas

As páginas publicas ficam salvas em resources/views, como o Laravel utuliza o blade que é uma gerenciador de template, as páginas estão separadas em body e header sendo que para incluir o header nelas basta utilizar @include('header'); Essas páginas no router estão dentro do middleware guest.

2.1 - Área LowCost

As views da area lowcost estão salvas em /views/lowcost, para evitar ficar copiando o mesmo layout foram criados dois arquivos sendo eles:

-> header_lowcost.blade.php

-> left-menu.blade.php

Para incluilos na view basta

@include('lowcost.header_lowcost')

@include('lowcost.left-menu')

Ficando assim no arquivo

```
@include('lowcost.header_lowcost')

<title>Low Cost Portal.:Noticias</title>

<!--navbar end-->
```

LOWCOST

```
<div class="main">
    <!-- sidebar -->
        @include('lowcost.left-menu')
```

Dentro da área lowcost é permitido gerenciar noticias cadastradas, cadastrar novas noticias e se usuario for ‘admin’ ele pode cadastrar usuarios para acessar a plataforma.

2.2 Cadastro de usuarios

Ao cadastrar um novo usuario, deverar se informado , seu nome, email, empresa, centro de custo e devera selecionar seu nível de permissão.

Após cadastrado, o usuario receberá um email com um link para cadastrar sua senha de acesso.

Ao clicar no botão criar senha ele será redirecionando para a tela de alterar senha, se o usuario ja tiver cadastrado sua senha o token de acessso será invalidado e se usuario tentar acessar novamentoe ele receberá a mensagem de link expirado

Tela de erro:

2.3 Controller

O controller que realiza o controle do dados de usuario como cadastro, login logout é a classe UserControllers ela se encontra na app/html/controllers. Aqui veremos o metodo register()

Ao receber os dados do form cadastra usuarios ele salvara na tabela users, criará um token de acesso que é salvo na tabela password_reset_tokens, onde salvará o email informado o token e data hora do cadastro.

Após inserção no banco de dados ele chamará o MailController o metodo sendUserMail onde ele passara o email, o assunto, a url de acesso e o nome do usuario

```
public function register(Request $request)
{
    $request->validate([
        'name' => 'required',
        'email' => 'required|email|unique:users',
        'confirm_email'=> 'required|same:email',
        'empresa'=>'required',
        'centrocusto'=>'required',
```

```
        'tipo'=>'required'
    ]);

    // insere o usuario na tabela users
    $created_at= date('Y-m-d H:i:s', time());

    DB::table('users')->insert(['name'=>$request->name,
                                'email'=>$request->email,
                                'password'=> md5(time()),
                                'active'=>'1',
                                'level' => $request->tipo,
                                'created_at'=>$created_at,
                                'empresa' => strtolower($request->empresa),
                                'centro_de_custo' => strtolower($request->centrocusto)]);

    $token = md5(bin2hex(random_bytes(32))); // token de acesso para usuario cadastrar

    $created_at= date('Y-m-d H:i:s', time()); //data hora tabela password_reset_tokens

    DB::table('password_reset_tokens')->insert(['email'=>$request-
>email,'token'=>$token,'created_at'=>$created_at ]); //tabela do token de usuario

    $url = route("user-register",$token);  //gera o link do servidor

    if(MailController::sendUserMail($request->email, 'Cadastro de Usuario',$url,$request->name))
    {
        return redirect('/register-user')->with('success', 'Cadastro Realizado com Sucesso!');
    }
    else return redirect('/register-user')->with('error', 'Erro ao enviar email, por favor contate o
suporte.');
}
```

2.4 - Acesso via token

Quando o usuario acessar o endpoint passando o token o router chamará no UserController o método userRegister, esse método verifica se o token existe, se existir ele tras os dados do usuario e o envia para view password que está na pasta auth se o token não existir envia para a tela expired-link.

O acesso deste endpoint esta no middleware guest.

```
Route::group(['middleware' => 'guest'], function ()
{
    //Rotas de acesso para que o usuario cadastrado consiga cadastrar sua
senha de acesso.
    Route::get('/user-register/{token}',
```

```
[\App\Http\Controllers\UserController::class, 'userPassword'])->name('user-register');

    Route::post('/update-password', [\App\Http\Controllers\UserController::class, 'updatePassword'])->name('update-password');
});
```

3 Niveis de acesso e Flags

Os niveis de acesso e permissões podem ser modificados tanto as permissões de paginas, e links, e tanto quanto as operações CRUD.

Para isso basta acessar a pasta flags/Falgs.php nela estão os metodos que retornam as matrizes para a classe PermissionPolicy.

Função getFlags()

```
public static function getFlags()
{
    return [

        'admin'=>[
            'inventario'=> ['listar'=>1, 'chamado'=>1],
            'dashboard' => ['inventario'=>1,'chamadas'=>1,'tracking'=>1,'contato'=>1]
        ],
        9 =>[
            'inventario'=> ['listar'=>1],
            'dashboard' => ['inventario'=>0,'chamadas'=>1,'tracking'=>0,'contato'=>1]
        ],

        8=>[
            'inventario'=> ['listar'=>1, 'chamado'=>1,'troca'=>0]
        ]
    ];
            ......
}
```

Ela relaciona a página com links de acesso e função para pagina por exemplo se a pagina acessada for o dashboard e o inventario tiver valor 0 siginifica que o link para o inventario não será mostrado para o usuario nivel 9.

Ja a função getCrud() relaciona as permissoes de listar dados, atualizar e mesmo deletar

```
public static function  getCrud()
{
    return [
        'admin'=>['create'=>1,'update'=>1,'delete'=>0,'select'=>1],
                9=>['create'=>0,'update'=>0,'delete'=>0,'select'=>1],
                8=>['create'=>0,'update'=>0,'delete'=>0,'select'=>1],
                        .....
    ];
}
```

3.1 PermissionPolicy

Esta classe gera flags de acessso baseadas no usuarioe na pagina que ele está navegando. Ela é chamada na view assim:

```
@php($perm = new App\Policies\PermissionPolicy(Route::currentRouteName()))
```

A função Route::currentRouteName() passa para o metodo construct o nome da pagina, metodo contruct ele set o usuario pela Facede Auth, e chama o metodo populateAuth().

```
public function __construct($page)
{
    $this->setUser(Auth::user());
    $this->populateAuth($page);
}
```

A função populateAuth($page) por sua vez requisita os flags da classe Flags e extrai da matriz as flags pela pagina de acesso e pelo nivel de permissão do usuario.

```
$flags = (Flags::getFlags());
$userFlags = $flags[$this->getUser()->level];
$crudFlags = Flags::getCrud();
$crudFlags = $crudFlags[$this->getUser()->level];
$this->generateAuthPerPage($userFlags[$page]);
$this->generateCrudAuth($crudFlags);
```

3.1.1 - A função generateAuthPerPage

Ao receber o array com as flags ela excuta um busca utilizando a chave e o valor dela e setando os metodos setHas(nomedachave) exemplo: setHasInventario(bool $has);

```
foreach ($auths as $k=> $v)
{
    switch($k)
    {
        case 'inventario':
            $this->setHasInventario(boolval($v));
        break;
        case 'chamadas':
            $this->setHasChamadas(boolval($v));
        break;
        case 'tracking':
            $this->setHasTraking(boolval($v));
        break;
        case 'contato':
            $this->setHasContato(boolval($v));
        break;
    }

}
```

3.1.2 - Acessando as chaves de permissão na pagina

Como ja dissemos o Laravel utiliza o blade como gerenciador de templates, para utilziarmos qualquer codigo php devemos usar ou @php() ou @if() @endif, para pegar os falgs basta utilizar o @if exemplo:

```
@if($perm->getHasInventario())
<li class="nav-item"><a class="nav-link" href="#">Invent&aacuterio</a></li>
```

```
@endif
@if($perm->getHasChamadas())
<li class="nav-item"><a class="nav-link" href="#">Chamadas</a></li>
@endif
```

Se no **if getHasInventario()** for true o html '<li class="nav-item">Invent&aacuterio ' será impresso na pagina caso false, ele será ocultado.

4 Exibindo link de acesso para usuarios

Existem links que devemos mostrar somente depois do usuario logado, para isso usamos a Facede Auth, exemplo

```
@if(Route::has('login') & !auth()->user())
    <button class="btn " type="submit" onclick="location.href='{{route('login')}}'" style="font-size:14px !important;">
        <i class="fas fa-sign-in"></i> Login
    </button>
@endif
```

Como funciona se no aquivos de rota existir a rota login e o usuario não estiver logado mostre o link de login.

```
@if (Auth::check())
    <button class="btn" type="submit" onclick="location.href='{{route('logout')}}'" style="font-size:14px !important;">
        <i class="fa-solid fa-right-from-bracket fa-icon"></i> Logout
    </button>
@endif
```

Se o usuario estiver logado ao ínves de mostar o link login mostrará o logout.

4 - Acesso de usuarios de outras empresas

Ao acessar a pagina de login, como dito anteriormente, o usuario deverá informar seu empresa, seu email e sua senha.

Os arquivos que forma as views estão nas pastas /resources/views/auth, e como o Laravel utiliza o blade para facilitar a criação de templates e evitar ficar copiando varias vezes o mesmo arquivo os componentes header e o side menu estão separados das views salvas na pasta ../components.

Para incluilos nas views basta usar a tag @include;

Ex.:

```
@include('componets.header')
<!--navbar end-->
<div class="main">
    <!-- sidebar-->
    @include('componets.side_menu')
    <!--sidebar end-->
</div>
```

Lembrando que para acessar um arquivo em pasta basta utilizar **nome_da_pasta . nome_do_arquivo**

```
<svg xmlns="http://www.w3.org/2000/svg" style="display: none;"...>
<title>Low Cost Portal.:Dashboard</title>
@include('componets.header')
<!--navbar end-->
<div class="main">
    <!-- sidebar-->
    @include('componets.side_menu')
    <!--sidebar end-->
</div>
```