SPATIE COM LARAVEL E JETSTREAM

Após a instalação do projeto em Laravel e configuração do .env, realizamos o download do pacote Jetstream:

- composer require laravel/jetstream
- php artisan jetstream:install livewire
- php artisan migrate
- npm install
- npm run dev

Ao final desses comandos (via terminal, apontando para o projeto), é correto concluir que a instalação do Jetstream está completa. Para as seguintes instruções, seguindo a biblioteca Spatie, vamos presumir que você já possui conhecimento em Jetstream (domínio sobre as views criadas, tabelas, comandos, etc). Iniciaremos a biblioteca Spatie dessa forma, pelo próprio terminal:

- composer require spatie/laravel-permission
- php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
- php artisan optimize:clear
- php artisan migrate

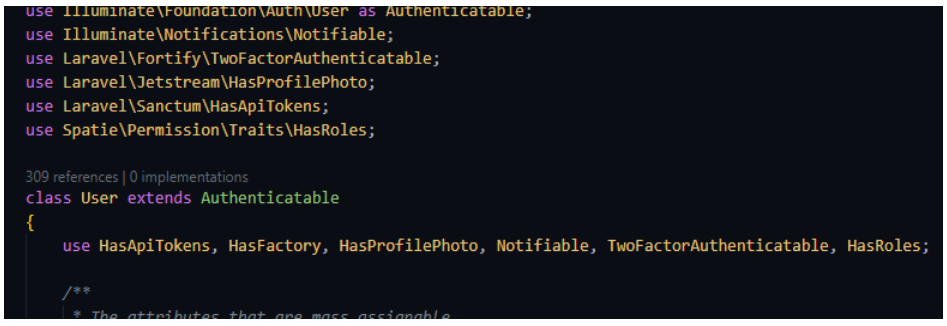
Seguindo esses comandos, na model de usuário (app/Models/User.php) vamos inserir:

use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable

{

use HasFactory, Notifiable, HasRoles;



Adicionamos no arquivo config/app.php, em '"providers":

- Spatie\Permission\PermissionServiceProvider::class,

Inserir os middlewares em app/Http/Kernel.php em '$routeMiddleware':

- 'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,
- 'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,
- 'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,

Neste ponto, já é possível executar comandos via terminal para a biblioteca Spatie. Confira os principais:

- php artisan permission:create-permission “exemplo” /* cria permissões */
- php artisan permission:create-role “exemplo” /* cria papéis */

Podemos atribuir, dentro do código, ao definir um usuário:

- $exemplo->assignRole('exemplo'); /* atribuindo uma role ao usuário */
- $exemplo ->givePermissionTo('exemplo'); /* atribuindo uma permissão ao usuário */

Lembrando que, para atribuí-los no momento da criação do usuário (contanto que esteja usando o Jetstream), deve-se implementar o código no arquivo (app/Actions/Fortify/CreateNewUser.php), veja:

```php
    */
    239 references | 0 overrides
    public function create(array $input)
    {
        Validator::make($input, [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
            'password' => $this->passwordRules(),
            'terms' => Jetstream::hasTermsAndPrivacyPolicyFeature() ? ['accepted', 'required'] : '',
        ])->validate();

        return User::create([
            'name' => $input['name'],
            'email' => $input['email'],
            'password' => Hash::make($input['password']),
        ])->assignRole('Regular'); /* Atribuindo papel/role regular ao usuário recém-cadastrado. */;
    }
}
```

Podemos proteger as views com o atributo @can, do próprio Laravel.

@can(‘exemplo_permissao’)

O conteúdo entre estes atributos só será visto

por usuários que tenham a permissão “exemplo_permissao”

@endcan

Verifique o exemplo:

```blade
        </a>
    </li>

    @can('Visualizar painel administrativo')
        <li class="nav-item">
            <a class="nav-link" href="/painel" style="color: #fff"><i class="bi bi-controller"></i>
                Controle de Acesso</a>
        </li>
    @endcan

    <form action="logout" method="post">
```

Além disso, também devemos proteger nossas rotas. Por costume particular, agrupo-as de acordo com seus controllers e adiciono a camada de proteção relacionada às permissões. Observe que atribuímos os nomes das permissões a elas:

```php
Route::controller(ContatosController::class)->group(function () {
    Route::get('/', 'index');
    Route::get('/contato', 'create')->middleware(['permission:Cadastrar contato']);
    Route::post('/contato/store', 'store')->middleware(['permission:Cadastrar contato']);
    Route::get('/contato/email/{id}', 'send_email')->middleware(['permission:Enviar e-mail']);
    Route::get('/contato/edit/{id}', 'edit')->middleware(['permission:Editar contato']);
    Route::put('/contato/update/{id}', 'update')->middleware(['permission:Editar contato']);
    Route::delete('/contato/delete/{id}', 'destroy')->middleware(['permission:Excluir contato']);
});

Route::controller(RolesController::class)->group(function () {
    Route::get('/painel', 'index')->middleware(['permission:Visualizar painel administrativo']);
    Route::get('/role', 'create')->middleware(['permission:Cadastrar perfil de acesso']);
    Route::post('/role/store', 'store')->middleware(['permission:Cadastrar perfil de acesso']);
    Route::get('/role/show/{id}', 'show')->middleware(['permission:Visualizar perfil de acesso']);
    Route::get('/role/edit/{id}', 'edit')->middleware(['permission:Editar perfil de acesso']);
    Route::put('/role/update/{id}', 'update')->middleware(['permission:Editar perfil de acesso']);
    Route::delete('/role/delete/{id}', 'destroy')->middleware(['permission:Excluir perfil de acesso']);
});
```

Podemos usar comandos como php artisan optimize:clear, php artisan config:clear, php artisan cache:clear para executar a limpeza do cache, caso algum dos procedimentos não apresente resultado.