

ENSINO MÉDIO INTEGRADO - INFORMÁTICA
Disciplina de Desenvolvimento Web | Backend
Aula 03: Framework Laravel - Autenticação e Autorização

Gil Eduardo de Andrade

Conceitos Preliminares

(<https://laravel.com/>)

Introdução

O Laravel é um dos frameworks PHP mais populares e poderosos da atualidade, conhecido pela sintaxe elegante, ecossistema robusto e foco na experiência do desenvolvedor. Dois dos pilares fundamentais de qualquer aplicação web moderna são a autenticação, que verifica a identidade de um usuário, e a autorização, que determina quais ações esse usuário pode realizar.

Neste documento iremos explorar os dois conceitos no Laravel 12, versão mais recente do framework (*novembro de 2025*). Para isso utilizaremos o Laravel Breeze, uma implementação rápida e minimalista do sistema de autenticação, juntamente com as Policies, que gerenciam de forma organizada e granular as regras de autorização da aplicação.

Autenticação com Laravel Breeze

O Laravel oferece diversos pacotes e ferramentas para lidar com a autenticação. O Laravel Breeze se destaca por ser uma implementação minimalista e simples de todas as funcionalidades de autenticação do Laravel, incluindo login, registro, redefinição de senha, verificação de e-mail e confirmação de senha.

O que é o Laravel Breeze?

Breeze é um starter kit que fornece um ponto de partida limpo para construir uma aplicação Laravel. Ele publica rotas de autenticação, controladores e views em sua aplicação, que podem ser facilmente customizadas. O Breeze utiliza o Laravel Fortify, por baixo dos panos, para as funcionalidades de autenticação, mas abstrai sua complexidade inicial. No Laravel 13, o Breeze não é mais uma opção padrão durante a criação de um novo projeto com o instalador “*laravel new*”. No entanto, ele continua totalmente compatível e pode ser instalado manualmente.

Instalação e Configuração

Para adicionar o Breeze num projeto Laravel 13, execute os seguintes comandos:

Adiciona a Dependência

Shell

```
composer require laravel/breeze --dev
```

Instala o Pacote/Scaffolding do Breeze

Shell

```
php artisan breeze:install blade --no-interaction
```

Autorização com Laravel Policies

Uma vez que um usuário está autenticado, torna-se necessário definir uma maneira de controlar o que ele tem ou não permissão de efetuar dentro da aplicação. O Laravel oferece um sistema de autorização com duas abordagens principais: “*Gates*” e “*Policies*”.

Gates são ideais para ações que não estão atreladas a um modelo específico. Policies, por outro lado, agrupam a lógica de autorização para um modelo ou recurso particular.

Neste documento, focaremos nas Policies, visto que elas oferecem uma maneira mais estruturada de gerenciar permissões em aplicações complexas.

Gerando uma Policy

“*Policies*” são classes simples que organizam a lógica de autorização. Para gerar uma nova policy, utilizamos o comando Artisan make:policy. Por convenção, as “*policies*” são armazenadas no diretório app/Policies.

Vamos criar uma “*PostPolicy*” para um modelo “*Post*”. Usando a flag --model=Post, o Artisan irá gerar automaticamente os métodos básicos para as ações de CRUD (Create, Read, Update, Delete).



Shell

```
php artisan make:policy PostPolicy --model=Post
```

Isso criará o arquivo “*app/Policies/PostPolicy.php*” com a seguinte estrutura:

Arquivo: “app/Policies/PostPolicy.php”

PHP

```
<?php
```

```
namespace App\Policies;
```

```
use App\Models\Post;
```

```
use App\Models\User;
```

```
use Illuminate\Auth\Access\HandlesAuthorization;
```

```
class PostPolicy
```

```
{
```

```
    use HandlesAuthorization;
```

```
    public function viewAny(User $user): bool
```

```
    {
```

```
        //
```

```
    }
```

```
    public function view(User $user, Post $post): bool
```

```
    {
```

```
        //
```

```
    }
```

```
    public function create(User $user): bool
```

```
    {
```

```
        //
```

```
    }
```

```
    public function update(User $user, Post $post): bool
```

```
    {
```

```
        //
```

```
    }
```

```
    public function delete(User $user, Post $post): bool
```

```
    {
```

```
//  
}  
  
public function restore(User $user, Post $post): bool  
{  
    //  
}  
  
public function forceDelete(User $user, Post $post): bool  
{  
    //  
}  
}
```

Registrando a Policy

O Laravel possui um recurso de “*policy discovery*” que encontra automaticamente as “*policies*”, desde que o modelo e a “*policy*” sigam as convenções de nomenclatura padrão do Laravel (ex: “*Post*” model e “*PostPolicy*”).

Se for necessário registrar uma “*policy*” manualmente, isso pode ser feito no método “*boot*” do seu “*AuthServiceProvider*” (app/Providers/AuthServiceProvider.php):

Arquivo: “app/Providers/AuthServiceProvider.php”

```
PHP  
use App\Models\Post;  
use App\Policies\PostPolicy;  
use Illuminate\Support\Facades\Gate;  
  
protected $policies = [  
    Post::class => PostPolicy::class,  
];
```

Escrevendo a Lógica da Policy

Cada método na policy recebe o usuário autenticado como primeiro argumento e, opcionalmente, a instância do modelo relevante. O método deve retornar “**true**” se a ação for autorizada e “**false**” caso contrário.

Vamos definir uma regra simples: um usuário só pode atualizar um “*post*” se ele for o autor do “*post*”.

Arquivo: “app/Policies/PostPolicy.php”

```
PHP
public function update(User $user, Post $post): bool
{
    return $user->id === $post->user_id;
}
```

Exemplo Prático: Unindo Tudo

Imagine um cenário onde temos “*posts*” de um blog, e apenas o autor de um “*post*” pode editá-lo ou excluí-lo.

1. **Controller:** No “*PostController*” é possível utilizar o método “***authorize***” para verificar se o usuário possui permissão, antes de executar a ação. O Laravel irá automaticamente chamar a “*policy*” correspondente.
2. **Blade Views:** no frontend é possível verificar as permissões diretamente nas views Blade, através da utilização das diretivas “***@can***” e “***@cannot***”.

APLICANDO OS CONCEITOS: AUTH BREEZE / POLICIES

Continuação da Aplicação Criada na Aula Anterior

1) Instalando e Utilizando Breeze

Criando um arquivo com as rotas da aplicação

Arquivo: “routes/app.php”

```
PHP
<?php

use App\Http\Controllers\AlunoController; //Adicionado
use App\Http\Controllers\CursoController;
use App\Http\Controllers\DisciplinaController; //Adicionado
use App\Http\Controllers\MatriculaController;
```



```
use Illuminate\Support\Facades\Route;

Route::get('/home', function () {
    return view('home');
})->name('home');

Route::resource('/curso', CursoController::class);
Route::resource('/disciplina', DisciplinaController::class);
Route::resource('/aluno', AlunoController::class);
Route::resource('/matricula', MatriculaController::class);
```

OBS.: as rotas que haviam sido adicionadas ao arquivo “/route/web.php”, nas aulas anteriores, foram deslocadas para cá.

Comando

(no terminal de comando - dentro da pasta da aplicação)

Adicionando a Dependência

Shell

```
docker compose exec app composer require laravel/breeze --dev
```

OBS.: antes de executar o comando a seguir, salve as rotas que já tiverem sido criadas no arquivo “/route/web.php”.

Instalando Pacote/Scaffolding do Breeze

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan
breeze:install blade --no-interaction
```

***Scaffolding** na programação é um processo de geração automática de código que cria a estrutura básica (esqueleto) para um aplicativo, economizando tempo dos desenvolvedores.

Efetuando a Migração Novamente - Tabelas Autenticação

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan
migrate:fresh --seed
```

Atualizando o arquivo de rotas principal

Arquivo: "routes/web.php"

```
PHP
<?php

use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;

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

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

require __DIR__.'/auth.php';
require __DIR__.'/app.php'; // Linha adicionada (importação)
```

2) Pastas e Arquivos Criados

Controllers:

- **/app/Http/Controllers/auth/AuthenticatedSessionController.php:** responsável por configurar a sessão de autenticação do usuário.
- **/app/Http/Controllers/auth/ConfirmablePasswordController.php:** responsável pelos eventos de confirmação de senha.
- **/app/Http/Controllers/auth/EmailVerificationNotificationController.php:** responsável pelos eventos de verificação de e-mail do usuário que se registrou recentemente.
- **/app/Http/Controllers/auth/NewPasswordController.php:** responsável pelos eventos de recebimento e atualização da nova senha do usuário.
- **/app/Http/Controllers/auth/PasswordResetLinkController.php:** responsável pelos eventos de solicitação de redefinição de senha - envio do link de reset.
- **/app/Http/Controllers/auth/RegisteredUserController.php:** responsável pelos eventos de registro do usuário.
- **/app/Http/Controllers/auth/VerifyEmailController.php:** responsável pelos eventos de verificação de e-mail do usuário que se registrou recentemente.

Views/Components:

- **/app/View/Components/AppLayout.php**: define um componente que será utilizado na renderização do layout da aplicação para usuários, com autenticação.
- **/app/View/Components/GuestLayout.php**: define um componente que será utilizado na renderização do layout da aplicação para visitantes, sem autenticação.
- **/resources/views/dashboard.blade.php**: tela apresentada logo após a autenticação ser efetuada com sucesso.
- **/resources/views/auth/confirm-password.blade.php**: tela utilizada para confirmação de senha.
- **/resources/views/forgot-password.blade.php**: tela utilizada para recuperação de senha, quando usuário a esqueceu.
- **/resources/views/auth/login.blade.php**: tela de autenticação (solicita e-mail e senha) apresentada ao usuário.
- **/resources/views/auth/register.blade.php**: tela de registro de novo usuário, apresentada quando um novo cadastro é solicitado.
- **/resources/views/auth/reset-password.blade.php**: tela que permite ao usuário redefinir sua nova senha.
- **/resources/views/auth/verify-email.blade.php**: tela que permite efetuar a confirmação de e-mail do usuário que se registrou recentemente.

3) Aplicando Middleware de Autenticação na Rotas

Arquivo: "routes/app.php"

```
PHP
Route::get('/', function () {
    return view('home');
})->name('home')->middleware(['auth', 'verified']);

Route::resource('/curso', CursoController::class)
    ->middleware(['auth', 'verified']);

Route::resource('/disciplina', DisciplinaController::class)
    ->middleware(['auth', 'verified']);

Route::resource('/aluno', AlunoController::class)
    ->middleware(['auth', 'verified']);

Route::resource('/matricula', MatriculaController::class)
    ->middleware(['auth', 'verified']);
```

4) Acertando Redirecionamento - Após Autenticação

Arquivo: "/app/Http/Controllers/auth/AuthenticatedSessionController.php"



PHP

```
public function store(LoginRequest $request): RedirectResponse
{
    $request->authenticate();

    $request->session()->regenerate();

    return redirect()->intended(route('home', absolute: false));
}
```

Arquivo: "/app/Http/Controllers/auth/RegisteredUserController.php"

PHP

```
...
public function store(Request $request): RedirectResponse
{
    ...

    event(new Registered($user));

    Auth::login($user);

    return redirect(route('home', absolute: false));
}
```

5) Adicionando o Serviço "Vite" ao Docker e Configurando-o

Arquivo: "/docker-compose.yml"

PHP

```
// Adicionar logo abaixo do serviço "laravel_app"
vite:
  image: node:22-alpine
  container_name: laravel_vite
  working_dir: /app
  ports:
    - "5173:5173"
  volumes:
    - ./app
  command: sh -c "npm install && npm run dev -- --host"
```



Arquivo: “/vite.config.js”

PHP

```
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';

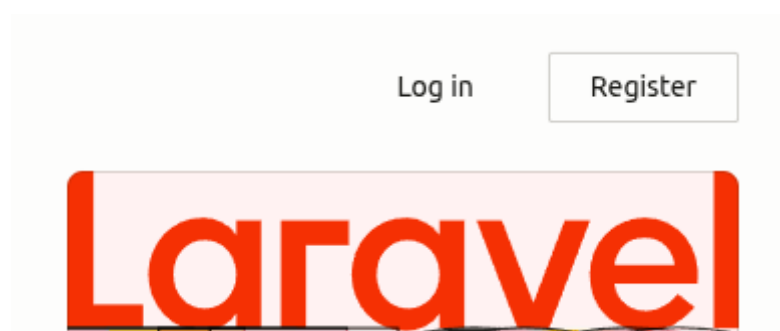
export default defineConfig({
  plugins: [
    laravel({
      input: ['resources/css/app.css', 'resources/js/app.js'],
      refresh: true,
    }),
  ],
  server: {
    hmr: {
      host: 'localhost',
    },
    watch: {
      usePolling: true,
    },
  },
});
```

6) Testando a Aplicação / Recursos de Autenticação

Executando a Aplicação

Shell

```
docker compose up
```



OBS.: clicar nos botões “Login” e “Register” para navegar entre as telas. Após isso tentar acessar a rota “/home”, para verificar que está protegida por autenticação. Ao final, efetue o registro de um novo usuário, e verifique para onde a aplicação irá navegar..

7) Atualizando Nome | Inserindo Logout

Arquivo: “resources/views/templates/main.php”

PHP

```
// Remover a linha:
<span class="ps-1 text-white">Visitante</span>

// Inserir as linhas:
@auth
    <span class="ps-1 text-white">
        {{
            Auth::user()
            ?
                explode(" ", Auth::user()->name)[0]
            :
                'Anônimo'
        }}
    </span>
@endauth
```



Sistema Aula



Alunos



Cursos



Disciplinas



Matrículas



Gil ▾

Arquivo: “resources/views/templates/main.php”

PHP

```
// Adaptar as linhas para:
<form method="POST" action="{{ route('logout') }}">
    @csrf
    <li>
        <a
            href=""
            onclick="event.preventDefault(); this.closest('form').submit();"
            class="dropdown-item"
        >
            <span class="ps-1 text-secondary ">Sair</span>
        </a>
    </li>
</form>
```

8) Criando as Models, Migrações e Seeders - Autorização

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:model Role -m
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:model Resource -m
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:model Permission -m
```

OBS.: Renomeie a migração de “Users” para que a tabela de usuários seja criada após a tabela de “Roles” (papéis do usuário).

Migrações

Arquivo: “database/migrations/create_roles_table.php”

PHP

```
public function up(): void  
{  
    Schema::create('roles', function (Blueprint $table) {  
        $table->id();  
        $table->string('name');  
        $table->timestamps();  
    });  
}
```

Arquivo: “database/migrations/create_resources_table.php”

PHP

```
public function up(): void  
{  
    Schema::create('resources', function (Blueprint $table) {  
        $table->id();  
        $table->string('name');  
        $table->timestamps();  
    });  
}
```



```
}
```

Arquivo: “database/migrations/create_permissions_table.php”

PHP

```
public function up(): void
{
    Schema::create('permissions', function (Blueprint $table) {
        $table->unsignedBigInteger('role_id');
        $table->foreign('role_id')->references('id')
            ->on('roles');
        $table->unsignedBigInteger('resource_id');
        $table->foreign('resource_id')->references('id')
            ->on('resources');
        $table->primary(['role_id', 'resource_id']);
        $table->boolean('permission');
    });
}
```

Arquivo: “database/migrations/create_users_table.php”

PHP

```
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->unsignedBigInteger('role_id');
    $table->foreign('role_id')->references('id')->on('roles');
    $table->rememberToken();
    $table->timestamps();
});
```

Models / Relacionamentos

Arquivo: “app/Models/Role.php”

PHP

```
public function resource() {
```



```
return $this->belongsToMany('\App\Models\Resource', 'permissions');  
}
```

Arquivo: “app/Models/Resource.php”

PHP

```
public function role() {  
    return $this->belongsToMany('\App\Models\Role', 'permissions');  
}
```

Arquivo: “app/Models/Permission.php”

PHP

```
public function role() {  
    return $this->belongsTo('\App\Models\Role');  
}  
  
public function resource() {  
    return $this->belongsTo('\App\Models\Resource');  
}
```

Arquivo: “app/Models/User.php”

PHP

```
#[Fillable(['name', 'email', 'password', 'role_id'])]  
  
...  
  
public function role() {  
    return $this->belongsTo('\App\Models\Role');  
}
```

Seeders

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:seeder RoleSeeder
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:seeder ResourceSeeder
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:seeder PermissionSeeder
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:seeder UserSeeder
```

Arquivo: "database/seeder/RoleSeeder.php"

```
PHP  
use Illuminate\Support\Facades\DB;  
...  
public function run(): void {  
    $data = [  
        ["name" => "PROFESSOR"],    // 1  
        ["name" => "COORDENADOR"], // 2  
    ];  
    DB::table('roles')->insert($data);  
}
```

Arquivo: "database/seeder/ResourceSeeder.php"

```
PHP  
use Illuminate\Support\Facades\DB;  
...  
public function run(): void  
{  
    $data = [  
        // CURSO  
        ["name" => "curso.index"],    // 1  
        ["name" => "curso.create"],   // 2  
        ["name" => "curso.show"],     // 3  
        ["name" => "curso.edit"],     // 4  
        ["name" => "curso.delete"],   // 5  
        // DISCIPLINA  
        ["name" => "disciplina.index"], // 6  
        ["name" => "disciplina.create"], // 7  
    ]  
}
```



```
[ "name" => "disciplina.show"],      // 8
[ "name" => "disciplina.edit"],      // 9
[ "name" => "disciplina.delete"],    // 10
// ALUNO
[ "name" => "aluno.index"],          // 11
[ "name" => "aluno.create"],         // 12
[ "name" => "aluno.show"],           // 13
[ "name" => "aluno.edit"],           // 14
[ "name" => "aluno.delete"],         // 15
// MATRICULA
[ "name" => "matricula.index"],      // 16
[ "name" => "matricula.create"],     // 17
[ "name" => "matricula.show"],       // 18
[ "name" => "matricula.edit"],       // 19
[ "name" => "matricula.delete"],     // 20
];
DB::table('resources')->insert($data);
}
```

Arquivo: "database/seeders/PermissionSeeder.php"

```
PHP
use Illuminate\Support\Facades\DB;
...
public function run(): void
{
    $data = [
        // PROFESSOR - DISCIPLINA
        [ "role_id" => 1, "resource_id" => 6 ],
        [ "role_id" => 1, "resource_id" => 7 ],
        [ "role_id" => 1, "resource_id" => 8 ],
        [ "role_id" => 1, "resource_id" => 9 ],
        // COORDENADOR - CURSO
        [ "role_id" => 2, "resource_id" => 1 ],
        [ "role_id" => 2, "resource_id" => 2 ],
        [ "role_id" => 2, "resource_id" => 3 ],
        [ "role_id" => 2, "resource_id" => 4 ],
        [ "role_id" => 2, "resource_id" => 5 ],
        // COORDENADOR - DISCIPLINA
        [ "role_id" => 2, "resource_id" => 6 ],
```



```
[ "role_id" => 2, "resource_id" => 7 ],  
[ "role_id" => 2, "resource_id" => 8 ],  
[ "role_id" => 2, "resource_id" => 9 ],  
[ "role_id" => 2, "resource_id" => 10 ],  
];  
DB::table('permissions')->insert($data);  
}
```

Arquivo: "database/seeders/UserSeeder.php"

```
PHP  
use Illuminate\Support\Facades\DB;  
use Illuminate\Support\Facades\Hash;  
...  
public function run(): void  
{  
    $data = [  
        [  
            "name" => "ANTÔNIO MARCOS SILVA",  
            'email' => "antonio@gmail.com",  
            "password" => Hash::make('@1234@5678'),  
            "role_id" => 1,  
        ],  
        [  
            "name" => "RAFAELA SANTOS",  
            'email' => "rafaela@gmail.com",  
            "password" => Hash::make('@1234@5678'),  
            "role_id" => 2,  
        ],  
    ];  
    DB::table('users')->insert($data);  
}
```

Arquivo: "database/seeders/DatabaseSeeder.php"

```
PHP  
public function run(): void {  
    $this->call(CursoSeeder::class);  
    $this->call(DisciplinaSeeder::class);  
    $this->call(AlunoSeeder::class);  
    $this->call(MatriculaSeeder::class);  
}
```

```
// Linhas Adicionadas
$this->call(RoleSeeder::class);
$this->call(ResourceSeeder::class);
$this->call(PermissionSeeder::class);
$this->call(UserSeeder::class);
}
```

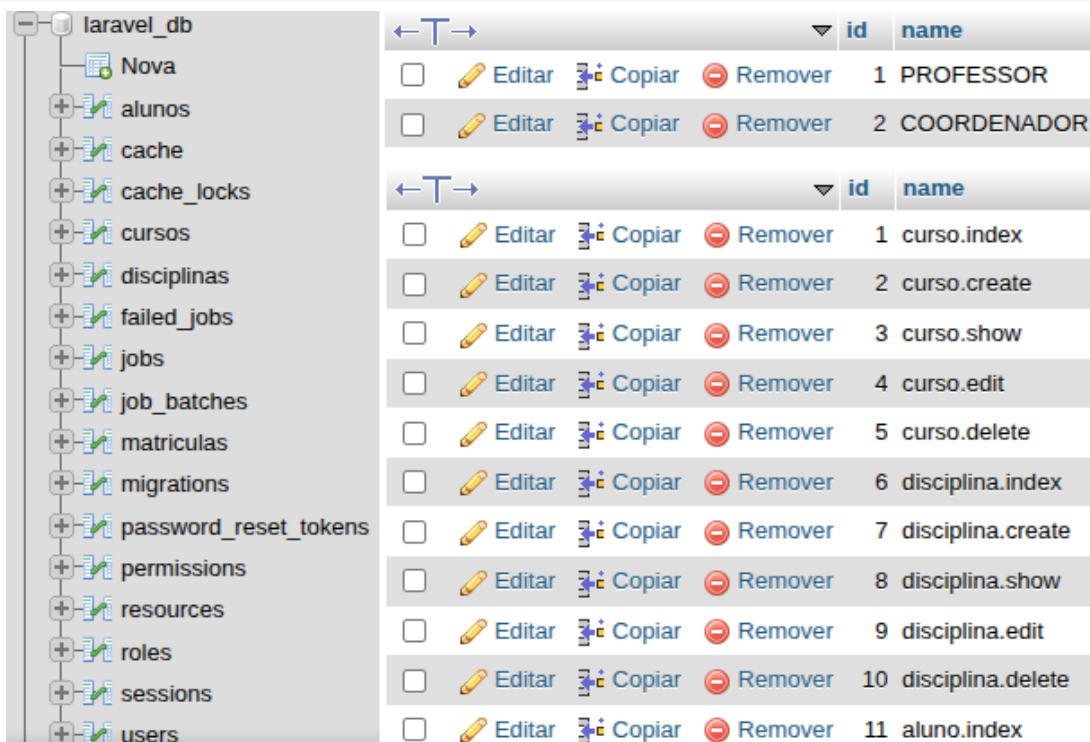
9) Executando as Migrations/Seeders na Base de Dados

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker-compose exec app php artisan migrate:fresh --seed
```



The screenshot shows a database management interface. On the left, a tree view displays the database structure, including tables like 'alunos', 'cache', 'cache_locks', 'cursos', 'disciplinas', 'failed_jobs', 'jobs', 'job_batches', 'matriculas', 'migrations', 'password_reset_tokens', 'permissions', 'resources', 'roles', 'sessions', and 'users'. The 'disciplinas' table is selected, and its records are displayed in a table on the right.

id	name
1	PROFESSOR
2	COORDENADOR
1	curso.index
2	curso.create
3	curso.show
4	curso.edit
5	curso.delete
6	disciplina.index
7	disciplina.create
8	disciplina.show
9	disciplina.edit
10	disciplina.delete
11	aluno.index

10) Criando Classe de Controle - Permissões

Comando

(no terminal de comando - dentro da pasta da aplicação)



Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:controller PermissionController
```

Arquivo: “app/Http/Controllers/PermissionController.php”

PHP

```
...  
use App\Models\Permission;  
  
class PermissionController extends Controller {  
  
    public static function loadPermissions($role) {  
  
        $arr_permissions = Array();  
        $perm = Permission::with(['resource'])  
            ->where('role_id', $role)->get();  
  
        foreach($perm as $item) {  
            $arr_permissions[$item->resource->name] = true;  
        }  
        // dd($arr_permissions);  
        session(['user_permissions' => $arr_permissions]);  
    }  
  
    public static function isAuthorized($resource) {  
        $permissions = session('user_permissions');  
  
        if(array_key_exists($resource, $permissions)) {  
            return true;  
        }  
        return false;  
    }  
}
```

11) Criando “Event” e “Listener” - Carregar Permissões

OBS.: Eventos e listeners, no Laravel, são uma implementação do padrão Observer, ideal para desacoplar a lógica da aplicação. Observer é um padrão de projeto comportamental que define uma dependência um-para-muitos entre objetos, permitindo que um objeto (publicador ou sujeito) notifique automaticamente outros objetos (assinantes ou observadores) sobre mudanças em seu estado.



Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:event AuthenticationEvent
```

Arquivo: “app/Events/AuthenticationEvent.php”

PHP

```
...  
public $data;  
/**  
 * Create a new event instance.  
 */  
public function __construct($data) {  
    $this->data = $data;  
}  
...
```

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:listener AuthenticationListener --event=AuthenticationEvent
```

Arquivo: “app/Listeners/AuthenticationListener.php”

PHP

```
use App\Http\Controllers\PermissionController;  
...  
/**  
 * Handle the event.  
 */  
public function handle(AuthenticationEvent $event): void {  
    PermissionController::loadPermissions($event->data);  
}  
...
```

Registrando o Evento

Arquivo: "app/Http/Controllers/AuthenticatedSessionController.php"

```
PHP
...
use App\Events\AuthenticationEvent;
...
public function store(LoginRequest $request): RedirectResponse {

    $request->authenticate();
    $request->session()->regenerate();

    // Registra o Evento de Autenticação - Permissão
    event(new AuthenticationEvent(Auth::user()->role_id));

    return redirect()->intended(route('home', absolute: false));
}
```

Arquivo: "app/Http/Controllers/RegisteredUserController.php"

```
PHP
...
use App\Events\AuthenticationEvent;
...
public function store(LoginRequest $request): RedirectResponse {

    ...

    // Registra o Evento de Autenticação - Permissão
    event(new AuthenticationEvent(Auth::user()->role_id));

    return redirect(route('home', absolute: false));
}
```

12) Testado o carregamento das Permissões / Após Autenticação

Descomentar a linha (para teste)

Arquivo: "app/Http/Controllers/PermissionController.php"

```
PHP
...
```



```
class PermissionController extends Controller {  
  
    public static function loadPermissions($role) {  
  
        $arr_permissions = Array();  
        $perm = Permission::with(['resource'])  
            ->where('role_id', $role)->get();  
  
        foreach($perm as $item) {  
            $arr_permissions[$item->resource->name] = true;  
        }  
  
        dd($arr_permissions);  
        session(['user_permissions' => $arr_permissions]);  
    }  
  
    ...  
}
```

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose up
```

Efetuar Autenticação na Aplicação (veja resultado)

Email

antonio@gmail.com

Password

.....

☐ Remember me

[Forgot your password?](#)

LOG IN



```
← → ↻ ⓘ localhost:15000/login  
array:4 [▼ // app/Http/Controllers/Permi  
    "disciplina.index" => true  
    "disciplina.create" => true  
    "disciplina.show" => true  
    "disciplina.edit" => true  
]
```

OBS.: volte a comentar a linha descomentada anteriormente, para que seja possível continuar os procedimentos de aula.

OBS.: quando a aplicação for colocada em produção, recomenda-se que os eventos criados sejam colocados em cache, através do seguinte comando:

Shell

```
php artisan event:cache
```

13) Criando as Policies - Cursos e Alunos

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:policy CursoPolicy --model=Curso
```

```
docker compose exec --user $(id -u):$(id -g) app php artisan  
make:policy DisciplinaPolicy --model=Disciplina
```

Arquivo: “app/Policies/CursoPolicy.php”

PHP

```
...  
use App\Http\Controllers\PermissionController;  
  
class CursoPolicy {  
  
    public function viewAny(User $user): bool {  
        return PermissionController::isAuthorized('curso.index');  
    }  
  
    public function view(User $user, Curso $curso): bool {
```



```
        return PermissionController::isAuthorized('curso.show');
    }

    public function create(User $user): bool {
        return PermissionController::isAuthorized('curso.create');
    }

    public function update(User $user, Curso $curso): bool {
        return PermissionController::isAuthorized('curso.edit');
    }

    public function delete(User $user, Curso $curso): bool {
        return PermissionController::isAuthorized('curso.delete');
    }

    public function restore(User $user, Curso $curso): bool {
        return PermissionController::isAuthorized('curso.delete');
    }

    public function forceDelete(User $user, Curso $curso): bool
    {
        return PermissionController::isAuthorized('curso.delete');
    }
}
```

Arquivo: "app/Policies/DisciplinaPolicy.php"

PHP

```
...
use App\Http\Controllers\PermissionController;

class DisciplinaPolicy {
    public function viewAny(User $user): bool {
        return PermissionController::isAuthorized('disciplina.index');
    }

    public function view(User $user, Disciplina $disciplina): bool {
        return PermissionController::isAuthorized('disciplina.show');
    }
}
```



```
public function create(User $user): bool {
    return PermissionController::isAuthorized('disciplina.create');
}

public function update(User $user, Disciplina $disciplina): bool {
    return PermissionController::isAuthorized('disciplina.edit');
}

public function delete(User $user, Disciplina $disciplina): bool {
    return PermissionController::isAuthorized('disciplina.delete');
}

public function restore(User $user, Disciplina $disciplina): bool {
    return PermissionController::isAuthorized('disciplina.delete');
}

public function forceDelete(User $user, Disciplina $disciplina): bool {
    return PermissionController::isAuthorized('disciplina.delete');
}
}
```

14) Acertando a Autorização nas Controllers Curso e Disciplina

Arquivo: "app/Http/Controllers/CursoController.php"

```
PHP
...
use Illuminate\Support\Facades\Gate;

class CursoController extends Controller {

    public function index() {
        Gate::authorize('viewAny', Curso::class);
        ...
    }

    public function create() {
        Gate::authorize('create', Curso::class);
        ...
    }
}
```



```
public function store(CursoRequest $request) {
    Gate::authorize('create', Curso::class);
    ...
}

public function show(string $id) {
    $curso = Curso::find($id);
    Gate::authorize('view', $curso);
}

public function edit(string $id) {
    $curso = Curso::find($id);
    Gate::authorize('update', $curso);
    ...
}

public function update(CursoRequest $request, string $id) {
    $curso = Curso::find($id);
    Gate::authorize('update', $curso);
    ...
}

public function destroy(string $id) {
    $curso = Curso::find($id);
    Gate::authorize('delete', $curso);
    ...
}
```

Arquivo: "app/Http/Controllers/DisciplinaController.php"

PHP

```
...
use Illuminate\Support\Facades\Gate;

class DisciplinaController extends Controller {

    public function index() {
        Gate::authorize('viewAny', Disciplina::class);
        ...
    }
}
```



```
public function create() {
    Gate::authorize('create', Disciplina::class);
    ...
}

public function store(DisciplinaRequest $request) {
    Gate::authorize('create', Disciplina::class);
    ...
}

public function show(string $id) {
    $disciplina = Disciplina::find($id);
    Gate::authorize('view', $disciplina);
}

public function edit(string $id) {
    $disciplina = Disciplina::find($id);
    Gate::authorize('update', $disciplina);
    ...
}

public function update(DisciplinaRequest $request, string $id) {
    $disciplina = Disciplina::find($id);
    Gate::authorize('update', $disciplina);
    ...
}

public function destroy(string $id) {
    $disciplina = Disciplina::find($id);
    Gate::authorize('delete', $disciplina);
    ...
}
```

15) Acertando a Autorização na View Curso (index)

Arquivo: “resources/views/curso/index.blade.php”

PHP

```
@extends('templates/main',
```



```
[
    'titulo'=>"Sistema Aula",
    'cabecalho' => 'Lista de Alunos',
    'rota' => 'aluno.create',
    'class' => App\Models\Curso::class,
]
)
...
@can('view', $item)
    <a href="{{route('curso.show', $item->id)}}" ...>
        <svg xmlns="...">
            ...
        </svg>
    </a>
@endcan

@can('update', $item)
    <a href="{{route('curso.edit', $item->id)}}" ...>
        <svg xmlns="...">
            ...
        </svg>
    </a>
@endcan

@can('delete', $item)
    <form action="{{route('curso.destroy', $item->id)}}" ...>
        @csrf
        @method('delete')
        ...
    </form>
@endcan
...
```

Arquivo: "resources/views/disciplina/index.blade.php"

PHP

```
@extends('templates/main',
[
    'titulo'=>"Sistema Aula",
    'cabecalho' => 'Lista de Disciplinas',
    'rota' => 'disciplina.create',
```



```
'class' => App\Models\Disciplina::class,
    ]
)
...
@can('view', $item)
    <a href="{{route('disciplina.show', $item->id)}}" ...>
        <svg xmlns="...">
            ...
        </svg>
    </a>
@endcan

@can('update', $item)
    <a href="{{route('disciplina.edit', $item->id)}}" ...>
        <svg xmlns="...">
            ...
        </svg>
    </a>
@endcan

@can('delete', $item)
    <form action="{{route('disciplina.destroy', $item->id)}}" ...>
        @csrf
        @method('delete')
        ...
    </form>
@endcan
...
```

16) Acertando a Autorização na View Template (main)

Arquivo: "resources/views/templates/main.blade.php"

```
PHP
@can('viewAny', App\Models\Curso::class)
    <li class="nav-item me-2">
        <a href="{{ route('curso.index') }}" class="nav-link">
            <svg xmlns="...">
                </svg>
            <span class="ps-1 text-white">Cursos</span>
        </a>
    </li>
@endcan
```



```
</li>
@endcan

@can('viewAny', App\Models\Disciplina::class)
    <li class="nav-item me-2">
        <a href="" class="nav-link">
            <svg xmlns="...">
            </svg>
            <span class="ps-1 text-white">Disciplinas</span>
        </a>
    </li>
@endcan

...
@if($rota != '')
    @can("create", $class)
        <a href= "{{ route($rota) }}" ...>
            <svg xmlns="...">
                ...
            </svg>
        </a>
    @endcan
@endif
```

EXECUTANDO E TESTANDO A APLICAÇÃO

Comando

(no terminal de comando - dentro da pasta da aplicação)

Shell

```
docker compose up
```

AUTENTICAÇÃO - PROFESSOR



Email

antonio@gmail.com

Password

.....

☐ Remember me

[Forgot your password?](#)

LOG IN

HOME



Sistema Aula



Alunos



Disciplinas



Matrículas



ANTÔNIO ▾

Home

DISCIPLINA INDEX

Lista de Disciplinas



NOME	CARGA HORÁRIA	CURSO	AÇÕES
BANCO DE DADOS	2 aula(s)	TECNÓLOGO EM DESENVOLVIMENTO	 
FÍSICA	2 aula(s)	TÉCNICO EM INFORMÁTICA	 

AUTENTICAÇÃO - COORDENADOR



Sistema Aula



Alunos



Cursos



Disciplinas



Matrículas



RAFAELA ▾

Home

DISCIPLINA INDEX



INSTITUTO FEDERAL
Paraná
Campus Paranaguá



Ministério da Educação

Lista de Disciplinas



NOME	CARGA HORÁRIA	CURSO	AÇÕES
BANCO DE DADOS	2 aula(s)	TECNÓLOGO EM DESENVOLVIMENTO	  
FÍSICA	2 aula(s)	TÉCNICO EM INFORMÁTICA	  