Guia para Administração

Agora para Administradores do programa, aqui estão as informações e código essencial.

Índice

1 - Como funcionam as Rotas, Controladores e Blades

O Laravel segue o padrão MVC (Model-View-Controller), onde **rotas**, **controladores** e **blades** trabalham juntos para processar pedidos HTTP e processar respostas.

Rotas

As rotas, definidas em **routes/web.php**, mapeiam URLs para controladores ou ações.

Exemplo:

```
use App\Http\Controllers\PostController;
Route::get('/posts', [PostController::class, 'index'])-
>name('posts.index')->middleware('can:view-posts');
Route::post('/posts', [PostController::class, 'store'])-
>name('posts.store')->middleware('can:create-posts');
```

- **get** e **post** definem métodos HTTP.
- **name()** atribui um nome à rota para uso em links.
- **middleware('can:view-posts')** restringe acesso com permissões.

Controladores

Os controladores, em **app/Http/Controllers**, contêm a lógica para processar pedidos. Cada método corresponde a uma ação (ex.: listar, criar, editar).

Exemplo (**app/Http/Controllers/PostController.php**):

```
namespace App\Http\Controllers;
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $this->authorize('view-posts');
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }

    public function store(Request $request)
    {
        $this->authorize('create-posts');
        $post = Post::create($request->validate([
            'title' => 'required|string|max:255',
            'content' => 'required',
        ]));
        return redirect()->route('posts.index');
    }
}
```

- **index**: Lista posts, passando dados para a view.
- **store**: Cria um novo post após validação.

Blade

O motor de templates do Laravel, usado em **resources/views** para renderizar HTML dinâmico.

Exemplo (**resources/views/posts/index.blade.php**):

```
@extends('layouts.app')
@section('title', 'Lista de Posts')
@section('content')
    <h1>Posts</h1>
    @can('create-posts')
        <a href="{{ route('posts.create') }}">Novo Post</a>
    @endcan
    @forelse($posts as $post)
        <div>
            <h2>{{ $post->title }}</h2>
            @can('edit-posts')
                <a href="{{ route('posts.edit', $post->id)
}}">Editar</a>
            @endcan
        </div>
    @empty
        <p>Sem posts</p>
    @endforelse
@endsection
```

- A view recebe **$posts** do controlador e usa diretivas como **@can** e **@forelse**.

Fluxo Completo

1. O utilizador acede a **/posts** (ex.: **http://localhost/posts**).
2. A rota em **routes/web.php** direciona para **PostController@index**.
3. O controlador verifica permissões, consulta o modelo **Post** no SQLite, e retorna a view **posts.index**.
4. A view Blade renderiza o HTML com os dados recebidos, aplicando lógica condicional (ex.: **@can**).

2 - Gerir Migrações (SQLite)

As migrações gerem a estrutura da base de dados SQLite.

- **Criar uma migração**:

```
php artisan make:migration criar_tabela_posts
```

Exemplo em **database/migrations**:

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

class CriarTabelaPosts extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table)
{
            $table->id();
            $table->string('title');
            $table->text('content');
            $table->unsignedBigInteger('user_id');
            $table->string('status')->default('rascunho');
            $table->timestamps();
            $table->foreign('user_id')->references('id')-
>on('users');
        });
    }

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

- **Executar migrações**:

```
php artisan migrate
```

- **Outros comandos**:
  - Reverter: **php artisan migrate:rollback**
  - Apagar e aplicar novamente: **php artisan migrate:fresh**
  - Verificar estado: **php artisan migrate:status**

3 - Factories e Seeders

Factories

- **Criar**:

```
php artisan make:factory PostFactory
```

Exemplo:

```
use App\Models\Post;
use Illuminate\Database\Eloquent\Factories\Factory;

class PostFactory extends Factory
{
    protected $model = Post::class;

    public function definition()
    {
        return [
            'title' => $this->faker->sentence,
            'content' => $this->faker->paragraph,
            'user_id' => User::factory(),
            'status' => $this->faker-
>randomElement(['rascunho', 'publicado']),
            'created_at' => now(),
            'updated_at' => now(),
        ];
    }
}
```

Seeders

- **Criar**:

```
php artisan make:seeder PostsTableSeeder
```

Exemplo:

```
use App\Models\Post;
use Illuminate\Database\Seeder;

class PostsTableSeeder extends Seeder
{
    public function run()
    {
        Post::factory()->count(15)->create();
    }
}
```

- **Executar**:

```
php artisan db:seed
php artisan db:seed --class=PostsTableSeeder
php artisan migrate:fresh --seed
```

4 - Manipular Blades com Permissões e Diretivas

Permissões com @can

- **Exemplo**:

```
@can('editar-posts')
    <a href="{{ route('posts.edit', $post->id)
}}">Editar</a>
@else
    <span>Sem permissão</span>
@endcan
```

- **Adicionar permissão (Isto pode ser feito na GUI)**:

```
use Spatie\Permission\Models\Permission;
Permission::create(['name' => 'publicar-posts']);
$user->givePermissionTo('publicar-posts');
```

No Blade:

```
@can('publicar-posts')
    <form action="{{ route('posts.publicar', $post->id) }}"
method="POST">
        @csrf
        <button type="submit">Publicar</button>
    </form>
@endcan
```

- **Backend**:

```
public function publicar(Post $post)
{
    $this->authorize('publicar-posts');
    $post->update(['status' => 'publicado']);
    return redirect()->route('posts.index');
}
```

Diretivas Blade

- **@yield**:
  Layout em resources/views/layouts/app.blade.php:

```
<!DOCTYPE html>
<html>
<head>
    <title>@yield('titulo', 'App')</title>
    <link href="{{ asset('css/app.css') }}"
rel="stylesheet">
</head>
<body>
    <div class="container">
        @yield('conteudo')
    </div>
</body>
</html>
View filha:
@extends('layouts.app')
@section('titulo', 'Posts')
@section('conteudo')
    <h1>Lista de Posts</h1>
    @can('criar-posts')
        <a href="{{ route('posts.create') }}">Novo Post</a>
    @endcan
@endsection
```

- **@include**:

```
@include('components.cabecalho')
```

Arquivo **components/cabecalho.blade.php**:

```
<header>
    <a href="{{ route('home') }}">Início</a>
    @can('ver-painel')
        <a href="{{ route('painel') }}">Painel</a>
    @endcan
</header>
```

- **@if, @else, @elseif**:

```
@if($post->status === 'publicado')
    <span>Publicado</span>
@elseif($post->status === 'rascunho')
    <span>Rascunho</span>
@else
    <span>Desconhecido</span>
@endif
```

- **@forelse**:

```
@forelse($posts as $post)
    <div>{{ $post->title }}</div>
@empty
    <p>Sem posts</p>
@endforelse
```

- **@auth, @guest**:

```
@auth
    <p>Logado como {{ auth()->user()->name }}</p>
@guest
    <a href="{{ route('login') }}">Iniciar Sessão</a>
@endauth
```

- **@csrf, @method**:

```
<form action="{{ route('posts.update', $post->id) }}"
method="POST">
    @csrf
    @method('PUT')
    <input type="text" name="title" value="{{ $post->title
}}">
    <button type="submit">Atualizar</button>
</form>
```

- **@error**:

```
<input type="text" name="title">
@error('title')
    <span class="erro">{{ $message }}</span>
@enderror
```

- **@component**:
  Arquivo **components/alerta.blade.php**:

```
<div class="alerta alerta-{{ $tipo }}">
    {{ $slot }}
</div>
```

Uso:

```
@component('components.alerta', ['tipo' => 'sucesso'])
    Operação concluída!
@endcomponent
```

5 - Técnicas e Boas Práticas

- **Rotas**:

```
Route::resource('posts', PostController::class)-
>middleware('can:ver-posts');
```

- **Debugging**:

```
php artisan tinker
```

Exemplo: **User::find(1)->givePermissionTo('ver-posts')**

- **Testes**:

```
php artisan make:test PostTest
```

Exemplo:

```
public function test_criar_post()
{
    $user = User::factory()->create()-
>givePermissionTo('criar-posts');
    $this->actingAs($user);
    $response = $this->post(route('posts.store'), ['title'
=> 'Teste']);
    $response->assertStatus(200);
    $this->assertDatabaseHas('posts', ['title' =>
'Teste']);
}
```

- **Segurança**:
  - Valide as permissões no backend.
  - Use HTTPS em produção.
  - Faça backups de **database.sqlite**:

```
copy database\database.sqlite database\backup.sqlite
```

Resumo dos Comandos

Tarefa	Comando
Iniciar servidor	**php artisan serve**
Cache para produção	**php artisan config:cache**
Criar migração	**php artisan make:migration nome**
Executar migrações	**php artisan migrate**
Criar factory	**php artisan make:factory NomeFactory**
Criar seeder	**php artisan make:seeder NomeSeeder**
Popular banco de dados	**php artisan db:seed**
Limpar cache	**php artisan cache:clear**
Criar teste	**php artisan make:test NomeTest**

6 - Configurar Vite, app.css e app.js

Configurar o Vite

O Laravel, a partir da versão 9, usa o **Vite** como ferramenta padrão para compilar e gerir os recursos (CSS e JavaScript).

1. **Configurar vite.config.js**:
   Editar **C:\daisy-scm\vite.config.js** para definir os ativos a compilar:

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

export default defineConfig({
    plugins: [
        laravel({
            input: [
                'resources/css/app.css',
                'resources/js/app.js',
                'resources/js/pages/login.js',
                'resources/js/pages/contacts.js',
                'resources/js/pages/users.js',
                'resources/js/pages/roles.js',
                'resources/js/pages/permissions.js',
            ],
            refresh: true,
        }),
        tailwindcss(),
    ],
});
```

   - **input**: Especifica os arquivos de entrada.
   - **refresh**: Ativa o refresh automático em desenvolvimento.

2. **Testar o Vite em desenvolvimento**:
   Execute:

```
npm run dev
```

   - O Vite inicia um servidor de desenvolvimento em **http://192.168.4.85:5173**, servindo os ativos em tempo real.
   - Acesse a aplicação em **http://192.168.4.85** (Nginx) para verificar se os ativos são carregados.

3. **Compilar para produção**:
   Gere os arquivos otimizados para produção:

```
npm run build
```

   - Os arquivos compilados são salvos em **C:\daisy-scm\public\build**, com um manifesto (**manifest.json**) para mapeamento.
   - ***Este comando deve ser efetuado sempre que se muda o ficheiro do vite ou nos ficheiros javascript.***

Configurar app.css

O arquivo **resources/css/app.css** é o ponto de entrada para estilos CSS.

1. **Criar ou editar app.css**:
   Em **C:\daisy-scm\resources\css\app.css**, adicione estilos:

```
@import 'tailwindcss';
@plugin "daisyui" {
    themes: emerald --default, dim --prefersdark;
    /* Sets the default theme to "emerald" and uses "dim"
if the user prefers a dark theme */
}
 *
 * Laravel's built-in pagination views */
@source
'../../vendor/laravel/framework/src/Illuminate/Pagination/r
esources/views/*.blade.php';
@source '../../storage/framework/views/*.php';
/* All Blade templates in the current project */
@source '../**/*.blade.php';
/* All JavaScript files in the project */
@source '../**/*.js';
@theme {
    --font-sans: 'Instrument Sans', ui-sans-serif, system-
ui, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji',
        'Segoe UI Symbol', 'Noto Color Emoji';
    /* Overrides the default sans-serif font stack with
'Instrument Sans' and standard fallbacks */
}
```

   - Estes estilos são aplicados às views Blade (ex.: lista de contactos).

2. **Integrar com Blade**:
   No layout principal (**resources/views/layouts/app.blade.php**), incluir o CSS compilado:

```
<!DOCTYPE html>
<html>
<head>
    <title>@yield('titulo', 'App')</title>
    @vite('resources/css/app.css')
</head>
<body>
    <div class="container">
        @yield('conteudo')
    </div>
</body>
</html>
```

   - A diretiva **@vite** carrega o CSS compilado pelo Vite. Em desenvolvimento, aponta para o servidor Vite (**192.168.4.85:5173**); em produção, usa os arquivos em **public/build**.

3. **Compilar**:
   - Em desenvolvimento: **npm run dev**
   - Em produção: **npm run build**

Configurar app.js

O arquivo **resources/js/app.js** é o ponto de entrada para JavaScript.

1. **Criar ou editar app.js**:
   Em **C:\daisy-scm\resources\js\app.js**, adicionar o código:

```
// Import Laravel Vite bootstrap configuration (e.g., CSRF
setup, Echo, etc.)
import './bootstrap';
// Import Alpine.js (used for declarative UI behavior)
import Alpine from 'alpinejs';

// Import individual utility functions
import { setupThemeToggle } from './utils/themeToggle.js';
import { setupToastTimer } from './utils/toastTimer.js';
import { setupSidebarHover } from
"./utils/sidebarHover.js";

// JS file to call all the JS scripts we want to run
globally

// Make Alpine available globally
window.Alpine = Alpine;
// Start Alpine to initialize its components
Alpine.start();

// Run scripts after the DOM is fully loaded
document.addEventListener('DOMContentLoaded', () => {
    // Enable sidebar hover interactivity (e.g., expand on
hover)
    setupSidebarHover();
    // Setup the theme toggle (light/dark mode switching)
    setupThemeToggle();
    // Setup auto-dismissal for toast notifications
(success, error, etc.)
    setupToastTimer();
});
```

   - O arquivo **bootstrap.js** (em **resources/js**) inicializa dependências como **Axios**.

2. **Integrar com Blade**:
   No layout **app.blade.php**, adicione:

```
@vite('resources/js/app.js')
```

   - Coloque após o **@vite** do CSS, dentro do **<head>** ou antes do **</body>**.

3. **Compilar**:
   - Desenvolvimento: **npm run dev**
   - Produção: **npm run build**

Integrar com Rotas, Controladores e Blade

O Vite integra-se com o fluxo rotas/controladores/Blade ao fornecer ativos compilados para as views.

- **Exemplo de fluxo**:

1. **Rota** (routes/web.php):

```
use App\Http\Controllers\PostController;
Route::get('/posts', [PostController::class, 'index'])-
>name('posts.index')->middleware('can:ver-posts');
```

2. **Controlador** (app/Http/Controllers/PostController.php):

```
namespace App\Http\Controllers;
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $this->authorize('ver-posts');
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }
}
```

3. **Blade** (resources/views/posts/index.blade.php):

```
@extends('layouts.app')
@section('titulo', 'Lista de Posts')
@section('conteudo')
    <h1>Posts</h1>
    @can('criar-posts')
        <a href="{{ route('posts.create') }}">Novo Post</a>
    @endcan
    @forelse($posts as $post)
        <div class="post">{{ $post->title }}</div>
    @empty
        <p>Sem posts</p>
    @endforelse
@endsection
```

1. O CSS em app.css estiliza a classe .post.
2. O JavaScript em app.js adiciona interatividade (ex.: alerta ao clicar em posts).
3. A diretiva @vite carrega os ativos compilados.

Configurar Arranque Automático com Nginx

O Nginx e o PHP-FPM já estão configurados como serviços via NSSM (ver guia anterior). Para garantir que os ativos do Vite estejam disponíveis na rede em produção:

- Execute antes de iniciar os serviços:

```
cd C:\daisy-scm
npm run build
```

- Confirme que o Nginx está configurado para 192.168.4.85:80 (ver nginx.conf do guia anterior).
- Verifique que o firewall permite a porta 80:

```
netsh advfirewall firewall add rule name="Laravel Nginx"
dir=in action=allow protocol=TCP localport=80
```

Boas Práticas

- **Cache em produção**: Sempre execute npm run build após alterar app.css ou app.js.
- **Estrutura de ativos**: Organize CSS e JS em subdiretórios (ex.: resources/css/components) e importe em app.css/app.js.
- **Debugging**: Verifique erros no console do navegador e em C:\daisy-scm\storage\logs\laravel.log.
- **Backup**: Faça cópias do database.sqlite:

```
copy C:\daisy-scm\database\database.sqlite C:\daisy-scm\database\backup.sqlite
```

Resumo dos Comandos

Tarefa	Comando
Instalar dependências	**npm install**
Iniciar Vite (desenvolvimento)	**npm run dev**
Compilar para produção	**npm run build**
Cache do Laravel	**php artisan config:cache**
Iniciar serviços	**nssm start php-fpm && nssm start nginx**

7 - Temas

Editar **resources/css/app.css**:

```
@plugin "daisyui" {
    themes: cmyk --default, dim --prefersdark;
}
```

Mudar ‘cmyk’, que será o tema claro e ‘dim’, que será o escuro.

Editar **resources/js/utils/themeToggle.js** com a mesma mudança:

```
ffunction toggleTheme() {
    // If switch is checked, use dark theme ("dim"),
otherwise use light ("emerald")
    const theme = themeSwitcher.checked ? 'dim' : 'cmyk';
    // Apply the selected theme to the HTML root element
    document.documentElement.setAttribute('data-theme',
theme);
    // Persist the theme selection in localStorage so it's
remembered on page reload
    localStorage.setItem('theme', theme);
}
```

Lista dos temas disponibilizados por o DaisyUI:

https://daisyui.com/docs/themes/

Se quisermos criar um tema personalizado:

https://daisyui.com/theme-generator/#theme=eJyVlPFugyAQxl-FmCzZkkoABXVvQ_VczawYsFm7Ze--ExtrizXZn3f3-77jDsJP1OkjRO-R3l-ci3ZRaVpjY1cewKcrbT8xG8dTfq8dxJwxrJjPtjy8CvZCGGU5Eax4ewTFAuRyA0yW4JZjaboBumGGiwnOFMkYVUW2FPS2OWp7ubFiZIWSCeFCrpCBu2TPFA6QrZbueUJFdj1MSkSR02RVEDRJt4W6LO9pRAkLiXA1eUB2cBqsbmckebLqK_evbTddbWYwS0eQKyISQZXij2DgLArvrFCB--DyztqdcEJ8nLO7h_HMhEtOM85X4LCDmjpIFAlaJOlS9KVt13Qft8tMae4nKJQgGCixQoc3uakCa42dWSUnMpc4MRUhGQ6QrSmsrpqTw-fVQjl4f2HhuCzVDbQV5nFj8qG0N2dfmPOu-Yal11Lja2tme2MrGGnen32ign44jLGPOtO48SdhGFVQ61OLM9W6dbCLegs1WOc_mSn3-wcK-0x_