José Onésimo Abel Nuvunga

Documentação da implementação de :

1- testes com php unit

2- Logs de acesso e ações do usuário

3- Autenticação de dois factores

4- Comunicação de Aplicações usando APIs

5- github single sign on

6- Autorizacao

1. Implementação de Testes com PHPUnit

Configuração do PHPUnit

Primeiramente, configurou-se o arquivo `phpunit.xml`, que já acompanha o framework Laravel, assegurando que as configurações estavam adequadas para o ambiente local. Criou-se um teste inicial com o seguinte comando:

```
php artisan make:test UserAuthenticationTest
```

Exemplo de Teste Unitário

O teste abaixo valida se o sistema retorna com sucesso ao verificar uma autenticação básica:

```
namespace Tests\Unit;

use Tests\TestCase;

class UserAuthenticationTest extends TestCase
{
    public function testSuccessfulLogin()
    {
        $response = $this->post('/login', [
            'email' => 'exemplo@dominio.com',
            'password' => 'senha123'
        ]);

        $response->assertStatus(200);
    }
}
```

Execução dos Testes

Os testes executam-se com o seguinte comando:

```
php artisan test
```

2. Logs de Acesso e Ações do Usuário

Criação de Middleware

Criou-se um middleware chamado `LogUserActions` para capturar logs de acessos e ações realizadas pelos usuários. A lógica implementada no middleware é apresentada a seguir:

```
use Illuminate\Support\Facades\Log;

public function handle($request, Closure $next)
{
    if (auth()->check()) {
        Log::info('Ação do usuário registrada:', [
            'user_id' => auth()->user()->id,
            'path' => $request->path(),
            'method' => $request->method(),
            'ip' => $request->ip(),
        ]);
    }

    return $next($request);
}
```

Registro do Middleware

Registrou-se o middleware no arquivo `app/Http/Kernel.php` dentro do grupo de middlewares `web`:

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\LogUserActions::class,
    ],
];
```

---

3. Autenticação de Dois Fatores

Instalação do Pacote

Instalou-se o pacote para autenticação de dois fatores utilizando o comando:

```
composer require pragmaRX/google2fa-laravel
```

Após a instalação, publicou-se as configurações do pacote:

```
php artisan vendor:publish --
provider="PragmaRX\Google2FALaravel\ServiceProvider"
```

Configuração

Adicionou-se o campo `two_factor_secret` na tabela de utilizadores:

```
php artisan make:migration add_two_factor_secret_to_users
```

Implementou-se a lógica para validar o código gerado pelo aplicativo de dois fatores:

```
use PragmaRX\Google2FA\Google2FA;

$google2fa = new Google2FA();
$valid = $google2fa->verifyKey(auth()->user()->two_factor_secret, $request->input('otp'));

if (!$valid) {
    return redirect()->back()->withErrors(['otp' => 'Código inválido.']);
}
```

---

4. Comunicação de Aplicações usando APIs

Criação de Controlador API

Criou-se o controlador `UserApiController` para gerenciar requisições de dados de utilizadores:

```
php artisan make:controller Api\UserApiController
```

Adicionou-se a lógica de resposta JSON no controlador:

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;

class UserApiController extends Controller
{
    public function index()
    {
        return response()->json([
            'users' => [
                ['id' => 1, 'name' => 'João'],
                ['id' => 2, 'name' => 'Maria'],
            ]
        ], 200);
    }
}
```

Configuração de Rotas

As rotas foram configuradas no arquivo `routes/api.php`:

```
Route::get('/users', [UserApiController::class, 'index']);
```

Autenticação com Passport

Para autenticar as APIs, implementou-se o Laravel Passport. O pacote foi instalado com os comandos:

```
composer require laravel/passport
php artisan passport:install
```

Registrou-se o middleware de autenticação no grupo de middleware `api`:

```
'api' => [
    \Laravel\Passport\Http\Middleware\CreateFreshApiToken::class,
],
```

---

5. GitHub Single Sign-On

Configuração Inicial

Implementou-se a funcionalidade de login com GitHub utilizando o pacote Laravel Socialite. Para isso, configurou-se o `.env` com as credenciais da API do GitHub:

```
.env
GITHUB_CLIENT_ID=github_client_id
GITHUB_CLIENT_SECRET=github_client_secret
GITHUB_REDIRECT_URL=http://localhost/auth/github/callback
```

Controlador de Autenticação

Adicionou-se os métodos necessários no controlador `SocialLoginController`:

```
namespace App\Http\Controllers;

use Laravel\Socialite\Facades\Socialite;

class SocialLoginController extends Controller
{
    public function redirectToProvider()
    {
        return Socialite::driver('github')->redirect();
    }

    public function handleProviderCallback()
    {
        $user = Socialite::driver('github')->user();
        // Lógica para criar ou autenticar o utilizador
    }
}
```

Rotas

Configuraram-se as rotas em `routes/web.php`:

```
Route::get('auth/github', [SocialLoginController::class,
'redirectToProvider']);
Route::get('auth/github/callback', [SocialLoginController::class,
'handleProviderCallback']);
```

---

6. Autorização

Políticas de Autorização

Criou-se uma política chamada `PostPolicy` para gerenciar permissões de acesso a postagens. A política foi gerada com o comando:

```
php artisan make:policy PostPolicy
```

Adicionou-se a lógica para permitir que apenas o autor da postagem possa visualizá-la:

```
namespace App\Policies;

use App\Models\User;
use App\Models\Post;

class PostPolicy
{
    public function view(User $user, Post $post)
    {
        return $user->id === $post->user_id;
    }
}
```

Uso nas Rotas ou Controladores

Implementou-se a autorização no controlador de postagens:

```
$this->authorize('view', $post);
```

Registro da Política

Registrou-se a política no arquivo `AuthServiceProvider.php`:

```
protected $policies = [
    Post::class => PostPolicy::class,
];
```