Delfina Lúcia Nhavoto

Guião

Licenciatura em Informática

Universidade Save

Chongoene

2024

Funcionalidades

- **Comunicação de Aplicações usando APIs**
- **Autenticação e autorização**
- **Criptografia e proteção de dados**
- **Transações em Aplicações Laravel**
- **Testes unitários com PhpUnit**
- **Logs de auditoria**

1. Comunicacao de Aplicacoes usando APIs

Nesta funcionalidade vamos integrar API da gemine para criar um chatbot. E pra isso vamos seguir os seguites passos:

1. Obter as Credenciais da API da Gemini

- **Cria uma conta na Gemini.**
- **Gera uma chave API na área de configurações da conta.**
- **Anota a API Key e o API Secret.**

2. Configuração do projeto node pra backend

1. Instalar node no site oficial

2. Inicialize um novo projeto Node.js

mkdir gemini-backend – criar nova pasta

cd gemini-backend

```
npm init -y
```

3. Instale a biblioteca da google

```
npm install @google/generative-ai
```

4. Crie um ficheiro **`.env`** na raiz do projeto para armazenar sua chave de API

```
API_KEY=YOUR_API_KEY
```

5. Instale a biblioteca dotenv para carregar a chave de API:

```
npm install dotenv
```

6. Criar o arquivo server.js pra o servidor node

```
const express = require('express');
const { GoogleGenerativeAI } = require('@google/generative-ai');
const dotenv = require('dotenv');

dotenv.config();
const app = express();
const PORT = process.env.PORT || 3000;

// Configurar o middleware para JSON
app.use(express.json());

// Inicializar o cliente Gemini
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

// Endpoint para gerar texto
app.post('/generate-text', async (req, res) => {
    const { prompt } = req.body;

    // Verificar se o prompt foi fornecido
    if (!prompt) {
        return res.status(400).json({ error: 'Prompt é obrigatório.' });
    }

    try {
        const result = await model.generateContent(prompt);
        res.json({ text: result.response.text() });
    } catch (error) {
        // Caso o erro seja relacionado à API Gemini
        if (error.response) {
            console.error('Erro na API Gemini:', error.response.data);
            res.status(500).json({ error: 'Erro ao gerar conteúdo com a API Gemini.' });
        } else if (error.code === 'ENOTFOUND') {
            // Se o erro for de rede (ex: sem conexão com a internet)
            console.error('Erro de rede:', error.message);
            res.status(503).json({ error: 'Erro de conexão. Verifique sua internet e tente novamente.' });
        } else {
            // Outros tipos de erro inesperado
            console.error('Erro desconhecido:', error.message);
            res.status(500).json({ error: 'Erro inesperado. Tente novamente mais tarde.' });
        }
    }
});

app.listen(PORT, () => {
    console.log(`Servidor Node.js a correr na porta ${PORT}`);
});
```

2. Configurar o Frontend Laravel

1. Criar rotas no Laravel:

```
Route::get('/chat', [ChatController::class, 'chatView']);
Route::post('/chat', [ChatController::class, 'sendMessage'])->name('chat.send');
```

2. No `ChatController`, configurar para chamar o backend Node.js:

```
namespace App\Http\Controllers;

use Illuminate\Support\Facades\Http;
use Illuminate\Http\Request;

class ChatController extends Controller
{
    public function chatView()
    {
        return view('chat');
    }

    public function sendMessage(Request $request)
    {
        $message = $request->input('message');

        // Enviar a requisição para o backend Node.js
        try {
            $response = Http::post('http://localhost:3000/generate-
text', [
                'prompt' => $message,
            ]);

            // Verificar se a resposta foi bem-sucedida
            if ($response->successful()) {
                $geminiResponse = $response->json('text');
            } else {
                // Se a API retornar um erro
                $geminiResponse = 'Erro ao gerar resposta. Tente
novamente mais tarde.';
            }
        } catch (\Exception $e) {
            // Se houver erro de rede ou de conexão
            $geminiResponse = 'Erro de conexão com o servidor.
Verifique sua internet.';
        }

        return view('chat', compact('message', 'geminiResponse'));
    }
}
```

3. Interface do chatbot "chat.blade.php"

```
@extends('layouts.app')

@section('title', 'Chat')

@section('header')
<h1>Chat com Gemini</h1>
@endsection

@section('content')

<div class="container" style="padding-bottom: 80px;">
    <h2>Converse com o Chatbot</h2>

    <!-- Exibição de mensagens de sucesso -->
    @if (session('success'))
    <div class="alert alert-success">
        {{ session('success') }}
    </div>
```

```
    @endif

    <!-- Exibição de mensagens de erro -->
    @if (isset($geminiResponse) && strpos($geminiResponse, 'Erro')
!== false)
    <div class="alert alert-danger">
        <strong>Erro:</strong> {{ $geminiResponse }}
    </div>
    @endif

    <!-- Exibição das mensagens -->
    <div id="messages" class="mb-4" style="height: 400px; overflow-
y: auto;">
        <!-- Aqui vamos iterar para exibir as interações anteriores
-->
        @if (isset($message) && isset($geminiResponse) &&
strpos($geminiResponse, 'Erro') === false)
        <div class="bg-light p-4 mt-2 rounded-lg shadow-sm">
            <h4 class="font-semibold">Última interação:</h4>
            <div class="mt-2">
                <p><strong>Você:</strong> {{ $message }}</p>
                <p><strong>Gemini:</strong> {{ $geminiResponse
}}</p>
            </div>
        </div>
        @elseif (!isset($geminiResponse))
        <p class="text-muted mt-4">Nenhuma interação ainda.</p>
        @endif
    </div>

</div>

<!-- Rodapé com o formulário de envio -->
<div class="fixed-bottom bg-white p-3 shadow" style="z-index:
1050;">
    <form action="{{ route('chat.send') }}" method="POST" class="d-
flex align-items-center">
        @csrf
        <!-- Campo de mensagem -->
        <div class="mr-2" style="flex-grow: 1;">
            <textarea name="message" id="message" rows="2"
class="form-control" required placeholder="Digite sua
mensagem..."></textarea>
        </div>
        <!-- Botão de envio -->
        <button type="submit" class="btn btn-
primary">Enviar</button>
    </form>
</div>

<script>
    // Script para garantir que a área de mensagens role
automaticamente para baixo
    const messagesDiv = document.getElementById('messages');
    messagesDiv.scrollTop = messagesDiv.scrollHeight;
</script>

@endsection
```

3.executar o servidor

node server.js

2. Transações em Laravel

Nesta funcionalidade vou mostrar o beginTransantion, Commit e Rollback num controlador que criar atualiza e elimina dados numa tabela de pacientes.

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\User;
use App\Providers\RouteServiceProvider;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;
use Illuminate\View\View;
use App\Services\LogService; // Importação do LogService

class RegisteredUserController extends Controller
{
    /**
     * Exibir todos os utilizadores.
     */
    public function index(): View
    {
        $users = User::all();
        return view('pacientes.user_index', compact('users')); // Substituir pela view correta
    }

    /**
     * Exibir a página de registo.
     */
    public function create(): View
    {
        return view('auth.register');
    }

    /**
     * Criar um novo utilizador.
     */
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:users,email'],
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
            'nivel_acesso' => ['required', 'in:admin,gerente,operador'], // Validação para o nível de acesso
        ]);

        DB::beginTransaction();
        try {
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
                'nivel_acesso' => $request->nivel_acesso,
```

```
            ]);

            event(new Registered($user));

            // Registar log da criação do utilizador
            LogService::record([
                'user_id' => Auth::id(),
                'user_name' => Auth::user()->name,
                'model' => User::class,
                'model_id' => $user->id,
                'action' => 'create',
                'changes' => $user->getAttributes(),
                'message' => "Utilizador {$user->name} criado.",
                'ip_address' => $request->ip(),
                'status' => 'success',
            ]);

            DB::commit(); // Confirma as alterações
            return redirect()->route('users.index')->with('success', 'Utilizador criado com sucesso!');
        } catch (\Exception $e) {
            DB::rollBack(); // Reverte as alterações

            // Registar log de erro
            LogService::record([
                'user_id' => Auth::id(),
                'user_name' => Auth::user()->name,
                'model' => User::class,
                'action' => 'create',
                'message' => "Erro ao criar utilizador: {$e->getMessage()}",
                'ip_address' => $request->ip(),
                'status' => 'error',
            ]);

            return redirect()->back()->with('error', 'Erro ao criar utilizador: ' . $e->getMessage());
        }
    }

    /**
     * Atualizar um utilizador existente.
     */
    public function update(Request $request, User $user): RedirectResponse
    {
        $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:users,email,' . $user->id],
            'nivel_acesso' => ['required', 'in:admin,gerente,operador'], // Validação para o nível de acesso
            'password' => ['nullable', 'confirmed', Rules\Password::defaults()],
        ]);

        DB::beginTransaction();
        try {
            $oldAttributes = $user->getAttributes();

            $user->update([
                'name' => $request->name,
                'email' => $request->email,
                'nivel_acesso' => $request->nivel_acesso,
                'password' => $request->password ? Hash::make($request->password) : $user->password,
            ]);

            // Registar log da atualização
            LogService::record([
                'user_id' => Auth::id(),
```

```
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'model_id' => $user->id,
            'action' => 'update',
            'changes' => ['before' => $oldAttributes, 'after' => $user->getAttributes()],
            'message' => "Utilizador {$user->name} atualizado.",
            'ip_address' => $request->ip(),
            'status' => 'success',
        ]);

        DB::commit(); // Confirma as alterações
        return redirect()->route('users.index')->with('success', 'Utilizador atualizado com sucesso!');
    } catch (\Exception $e) {
        DB::rollBack(); // Reverte as alterações

        // Registar log de erro
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'action' => 'update',
            'message' => "Erro ao atualizar utilizador: {$e->getMessage()}",
            'ip_address' => $request->ip(),
            'status' => 'error',
        ]);

        return redirect()->back()->with('error', 'Erro ao atualizar utilizador: ' . $e->getMessage());
    }
}

/**
 * Excluir um utilizador.
 */
public function destroy(User $user): RedirectResponse
{
    DB::beginTransaction();
    try {
        $userName = $user->name;
        $user->delete();

        // Registar log da exclusão
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'model_id' => $user->id,
            'action' => 'delete',
            'message' => "Utilizador {$userName} excluído.",
            'ip_address' => request()->ip(),
            'status' => 'success',
        ]);

        DB::commit(); // Confirma as alterações
        return redirect()->route('users.index')->with('success', 'Utilizador excluído com sucesso!');
    } catch (\Exception $e) {
        DB::rollBack(); // Reverte as alterações

        // Registar log de erro
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'action' => 'delete',
```

```
                'message' => "Erro ao excluir utilizador: {$e->getMessage()}",
                'ip_address' => request()->ip(),
                'status' => 'error',
            ]);

            return redirect()->back()->with('error', 'Erro ao excluir utilizador: ' . $e->getMessage());
        }
    }
}
```

3. Logs de acesso

1. Modelo de logs

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Log extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',     // ID do usuário que realizou a ação
        'user_name',   // Nome do usuário que realizou a ação
        'ip_address',  // Endereço IP do usuário
        'model',       // Nome do modelo afetado
        'model_id',    // ID do registro afetado
        'action',      // Ação realizada (create, update, delete, etc.)
        'changes',     // Alterações feitas
        'status',      // Status da ação (success, error, info)
        'message',     // Descrição da ação
    ];

    protected $casts = [
        'changes' => 'array', // Converte o campo 'changes' em array
    ];

    // Relacionamento com o usuário
    public function user()
    {
        return $this->belongsTo(User::class, 'user_id');
    }

    /**
     * Obter o nome do usuário associado ao log.
     */
    public function getUserNameAttribute()
    {
        return $this->user->name ?? $this->user_name ?? 'Sistema';
    }

    /**
     * Formatar a data de criação do log.
```

```
     */
    public function getFormattedCreatedAtAttribute()
    {
        return $this->created_at->format('d/m/Y H:i:s');
    }
}
```

2. Migração da tabela de logs

2.1 Criar a tabela

php artisan make:migration create_logs_table

2.2. Faser migração

php artisan migrate

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('logs', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id')->nullable();  // ID do usuário (FK)
            $table->string('user_name')->nullable();            // Nome do usuário
            $table->string('ip_address')->nullable();           // Endereço IP do usuário
            $table->string('model');                            // Nome do modelo afetado
            $table->unsignedBigInteger('model_id')->nullable(); // ID do registro afetado
            $table->string('action');                           // Ação realizada (create, update, delete, etc.)
            $table->json('changes')->nullable();                // Alterações realizadas
            $table->string('status')->default('info');          // Status (success, error, info)
            $table->text('message')->nullable();                // Mensagem descritiva
            $table->timestamps();

            // Chave estrangeira opcional
            $table->foreign('user_id')->references('id')->on('users')->onDelete('set null');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('logs');
    }
};
```

2. Criar o serviço `LogService`

```
namespace App\Services;

use App\Models\Log;
use Illuminate\Support\Facades\Auth;

class LogService
{
    public static function record(array $data)
    {
        $userId = $data['user_id'] ?? Auth::id();
        $userName = $data['user_name'] ?? (Auth::check() ?
Auth::user()->name : 'Sistema');

        Log::create([
            'user_id' => $userId,
            'user_name' => $userName,
            'model' => $data['model'] ?? null,
            'model_id' => $data['model_id'] ?? null,
            'action' => $data['action'] ?? 'undefined',
            'changes' => $data['changes'] ?? [],
            'status' => $data['status'] ?? 'info',
            'message' => $data['message'] ?? 'Nenhuma mensagem
fornecida.',
            'ip_address' => $data['ip_address'] ?? request()->ip(),
            'created_at' => now(),
        ]);
    }
}
```

2. controlador de logs

```
namespace App\Http\Controllers;

use App\Models\Log;

class LogController extends Controller

{

   public function index()

   {

      $logs = Log::latest()->paginate(20); // Recupera os logs com paginação

      return view('logs.index', compact('logs')); // Passa os logs para a view

   }

}

3. usuando o logservice nos controladores

use App\Services\LogService;

class PacienteController extends Controller
{
    public function store(Request $request)
```

```
    {
        try {
            // Código de criação...
            LogService::record([
                'user_id' => auth()->id(),
                'model' => 'Paciente',
                'model_id' => $paciente->id,
                'action' => 'create',
                'changes' => $paciente->toArray(),
                'status' => 'success',
                'message' => 'Paciente criado com sucesso.',
            ]);
        } catch (\Exception $e) {
            LogService::record([
                'user_id' => auth()->id(),
                'model' => 'Paciente',
                'action' => 'create',
                'status' => 'error',
                'message' => $e->getMessage(),
            ]);
        }
    }
}
```

Exemplo de implementação num controlador de tabela de pacientes

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\User;
use App\Providers\RouteServiceProvider;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;
use Illuminate\View\View;
use App\Services\LogService; // Importação do LogService

class RegisteredUserController extends Controller
{
    /**
     * Exibir todos os utilizadores.
     */
    public function index(): View
    {
        $users = User::all();
        return view('pacientes.user_index', compact('users')); // Substituir pela view correta
    }

    /**
     * Exibir a página de registo.
     */
    public function create(): View
    {
        return view('auth.register');
    }

    /**
     * Criar um novo utilizador.
     */
```

```
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:users,email'],
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
            'nivel_acesso' => ['required', 'in:admin,gerente,operador'], // Validação para o nível de acesso
        ]);

        DB::beginTransaction();
        try {
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
                'nivel_acesso' => $request->nivel_acesso,
            ]);

            event(new Registered($user));

            // Registar log da criação do utilizador
            LogService::record([
                'user_id' => Auth::id(),
                'user_name' => Auth::user()->name,
                'model' => User::class,
                'model_id' => $user->id,
                'action' => 'create',
                'changes' => $user->getAttributes(),
                'message' => "Utilizador {$user->name} criado.",
                'ip_address' => $request->ip(),
                'status' => 'success',
            ]);

            DB::commit(); // Confirma as alterações
            return redirect()->route('users.index')->with('success', 'Utilizador criado com sucesso!');
        } catch (\Exception $e) {
            DB::rollBack(); // Reverte as alterações

            // Registar log de erro
            LogService::record([
                'user_id' => Auth::id(),
                'user_name' => Auth::user()->name,
                'model' => User::class,
                'action' => 'create',
                'message' => "Erro ao criar utilizador: {$e->getMessage()}",
                'ip_address' => $request->ip(),
                'status' => 'error',
            ]);

            return redirect()->back()->with('error', 'Erro ao criar utilizador: ' . $e->getMessage());
        }
    }

    /**
     * Atualizar um utilizador existente.
     */
    public function update(Request $request, User $user): RedirectResponse
    {
        $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:users,email,' . $user->id],
            'nivel_acesso' => ['required', 'in:admin,gerente,operador'], // Validação para o nível de acesso
            'password' => ['nullable', 'confirmed', Rules\Password::defaults()],
        ]);

        DB::beginTransaction();
        try {
            $oldAttributes = $user->getAttributes();

            $user->update([
```

```
            'name' => $request->name,
            'email' => $request->email,
            'nivel_acesso' => $request->nivel_acesso,
            'password' => $request->password ? Hash::make($request->password) : $user->password,
        ]);

        // Registar log da atualização
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'model_id' => $user->id,
            'action' => 'update',
            'changes' => ['before' => $oldAttributes, 'after' => $user->getAttributes()],
            'message' => "Utilizador {$user->name} atualizado.",
            'ip_address' => $request->ip(),
            'status' => 'success',
        ]);

        DB::commit(); // Confirma as alterações
        return redirect()->route('users.index')->with('success', 'Utilizador atualizado com sucesso!');
    } catch (\Exception $e) {
        DB::rollBack(); // Reverte as alterações

        // Registar log de erro
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'action' => 'update',
            'message' => "Erro ao atualizar utilizador: {$e->getMessage()}",
            'ip_address' => $request->ip(),
            'status' => 'error',
        ]);

        return redirect()->back()->with('error', 'Erro ao atualizar utilizador: ' . $e->getMessage());
    }
}

/**
 * Excluir um utilizador.
 */
public function destroy(User $user): RedirectResponse
{
    DB::beginTransaction();
    try {
        $userName = $user->name;
        $user->delete();

        // Registar log da exclusão
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
            'model' => User::class,
            'model_id' => $user->id,
            'action' => 'delete',
            'message' => "Utilizador {$userName} excluído.",
            'ip_address' => request()->ip(),
            'status' => 'success',
        ]);

        DB::commit(); // Confirma as alterações
        return redirect()->route('users.index')->with('success', 'Utilizador excluído com sucesso!');
    } catch (\Exception $e) {
        DB::rollBack(); // Reverte as alterações

        // Registar log de erro
        LogService::record([
            'user_id' => Auth::id(),
            'user_name' => Auth::user()->name,
```

```
                'model' => User::class,
                'action' => 'delete',
                'message' => "Erro ao excluir utilizador: {$e->getMessage()}",
                'ip_address' => request()->ip(),
                'status' => 'error',
            ]);

            return redirect()->back()->with('error', 'Erro ao excluir utilizador: ' . $e->getMessage());
        }
    }
}
```

Criptografia e proteção de dados

Para criptografia e proteção de dados usei os próprios pacotes de laravel pra codificar os dados
Exemplo de implementação de criptografia e descriptografia de dados

No modelo paciente implementei a codificação dos campos tanto a descodificação dos próprios campos.

No modelo

```
```

No controlador

```
```

Autenticação e autorização

Na auntenticao usei os pacotes de laravel quando faço o login

```php
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\LoginRequest;
use App\Mail\TwoFactorCodeMail;
use App\Providers\RouteServiceProvider;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Mail;
use Illuminate\View\View;

class AuthenticatedSessionController extends Controller
```

```php
{
    /**
     * Display the login view.
     */
    public function create(): View
    {
        return view('auth.login');
    }

    /**
     * Handle an incoming authentication request.
     */
    public function store(LoginRequest $request): RedirectResponse
    {
        $request->authenticate();

        // Verifica se há um ID armazenado para 2FA
        if (session('auth.id')) {
            return redirect()->route('auth.two-factor');
        }

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

        return redirect()->intended(RouteServiceProvider::HOME);
    }


    /**
     * Destroy an authenticated session.
     */
    public function destroy(Request $request): RedirectResponse
    {
        Auth::guard('web')->logout();

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

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

        return redirect('/');
    }
}
```

Na autorização usei **middleware** para fazer nivel de acessos para certas funcionalidades do sistema onde certos usuarios como operador, gerente não tem acesso total ao sistema. E integrei essa funcionalidade não rotas, onde nas rotas que tem

`'check.access:admin,gerente'` verifica o nivel de acesso do usuário logado, se não ter acesso não pode acessar essas rotas que tem essa verificação

1. Criando o Middleware

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Services\LogService; // Importação do LogService

class CheckAccess
{
    public function handle(Request $request, Closure $next, ...$roles)
    {
        // Verificar se o usuário está autenticado
        if (!Auth::check()) {
            // Registrar tentativa de acesso não autenticado
            LogService::record([
                'user_id' => null, // Usuário não autenticado
                'user_name' => 'Visitante',
                'model' => 'Middleware',
                'action' => 'unauthorized_access',
                'status' => 'error',
                'message' => 'Tentativa de acesso não autenticada.',
                'ip_address' => $request->ip(),
            ]);

            return redirect()->route('login')->with('error', 'Usuarios não
autenticado');
        }

        // Obter o nível de acesso do usuário autenticado
        $userRole = Auth::user()->nivel_acesso;

        // Verificar se o nível de acesso está permitido
        if (!in_array($userRole, $roles)) {
            // Registrar tentativa de acesso negado
            LogService::record([
                'user_id' => Auth::id(),
                'user_name' => Auth::user()->name ?? 'Usuário Desconhecido',
                'model' => 'Middleware',
                'action' => 'access_denied',
                'status' => 'error',
                'message' => "Tentativa de acesso negado para a rota: {$request-
>path()}.",
                'ip_address' => $request->ip(),
```

```
            ]);

            return redirect()->route('dashboard')->with('error', 'Acesso não
autorizado!');
        }

        // Passar para o próximo middleware ou controlador
        return $next($request);
    }
}
```

2. Configurando o middleware no arquivo Kernel.php para poder funcionar

pra adicionar a classe de middleware é só colar esta parte "`'check.access' => \App\Http\Middleware\CheckAccess::class,`" no arquivo kernel.php

```
protected $middlewareAliases = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'auth.session' =>
\Illuminate\Session\Middleware\AuthenticateSession::class,
        'cache.headers' =>
\Illuminate\Http\Middleware\SetCacheHeaders::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'password.confirm' =>
\Illuminate\Auth\Middleware\RequirePassword::class,
        'signed' => \App\Http\Middleware\ValidateSignature::class,
        'throttle' =>
\Illuminate\Routing\Middleware\ThrottleRequests::class,
        'verified' =>
\Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
        'check.access' => \App\Http\Middleware\CheckAccess::class,
        'check.device' => \App\Http\Middleware\CheckDevice::class,
        'restrict.ip' => \App\Http\Middleware\RestrictIP::class,
    ];
```

3. Implementando nas Rotas

```
<?php

use App\Models\Log;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;
use App\Http\Controllers\Auth\TwoFactorAuthController;
```

```php
use App\Http\Controllers\Auth\SettingsController;
use App\Http\Controllers\Auth\GithubController;
use App\Http\Controllers\PacienteController;
use App\Http\Controllers\LogController;
use App\Http\Controllers\DeviceController;
use App\Http\Controllers\ChatController;

Route::get('/auth/github', [GithubController::class,
'redirectToGithub'])->name('auth.github');
Route::get('/auth/github/callback', [GithubController::class,
'handleGithubCallback'])->name('auth.github.callback');

Route::get('/two-factor', [TwoFactorAuthController::class, 'show'])-
>name('auth.two-factor');
Route::post('/two-factor', [TwoFactorAuthController::class,
'verify']);
Route::post('/two-factor/resend', [TwoFactorAuthController::class,
'resend'])->name('auth.two-factor.resend');

Route::middleware(['restrict.ip'])->group(function () {

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

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



#, 'check.device'
Route::middleware('auth')->group(function () {

    Route::middleware('check.access:admin,gerente')->group(function
() {
        // Visualizar
        Route::get('pacientes/{paciente}',
[PacienteController::class, 'show'])->name('pacientes.show'); //
Detalhes
        Route::get('pacientes', [PacienteController::class,
'index'])->name('pacientes.index');
    });


    Route::get('/chat', [ChatController::class, 'chatView']);
    Route::post('/chat', [ChatController::class, 'sendMessage'])-
>name('chat.send');

    // Acesso exclusivo para administradores
```

```
    Route::middleware('check.access:admin')->group(function () {
        Route::post('pacientes', [PacienteController::class,
'store'])->name('pacientes.store'); // Criar
        Route::put('pacientes/{paciente}',
[PacienteController::class, 'update'])-
>name('pacientes.update');        // Atualizar
        Route::delete('pacientes/{paciente}',
[PacienteController::class, 'destroy'])->name('pacientes.destroy');
// Apagar
        Route::get('/logs', [LogController::class, 'index'])-
>name('logs.index');
        Route::get('/settings', [SettingsController::class,
'index'])->name('settings.index');
        Route::post('/settings/two-factor',
[SettingsController::class, 'toggleTwoFactor'])-
>name('settings.toggleTwoFactor');

            // Página inicial de dispositivos
        Route::get('/devices', [DeviceController::class, 'index'])-
>name('devices.index');

        // Rota para criar novo dispositivo
        Route::post('/devices', [DeviceController::class, 'store'])-
>name('devices.store');

        // Rota para editar dispositivo
        Route::put('/devices/{device}', [DeviceController::class,
'update'])->name('devices.update');

        // Rota para deletar dispositivo
        Route::delete('/devices/{device}', [DeviceController::class,
'destroy'])->name('devices.destroy');
    });

});

require __DIR__.'/auth.php';
```

Testes Unitarios com phpUnit

Criando um teste onde faz uma simulacao de criacao de pacientes com usuario que tem acesso e que nao tem acesso

php artisan make:test PacienteAccessTest –unit

```
<?php

namespace Tests\Unit;
```

```php
use App\Models\Paciente;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Support\Facades\Crypt;
use Tests\TestCase;

class PacienteAccessTest extends TestCase
{
    use RefreshDatabase;

    public function test_restricao_criacao_paciente()
    {
        // Criar um usuário gerente (não admin)
        $user = User::factory()->create([
            'nivel_acesso' => 'gerente', // ou 'operador'
        ]);

        // Simular login com o usuário gerente
        $this->actingAs($user);

        // Tentar criar um paciente
        $response = $this->post(route('pacientes.store'), [
            'nome' => 'Carlos Souza',
            'contacto' => '987654321',
            'idade' => 25,
            'sexo' => 'Masculino',
            'estado_medico' => 'Saudável',
            'consultas' => 'Nenhuma',
            'diagnostico' => 'Nenhum',
            'medicamentos' => 'Nenhum',
        ]);

        // Verificar que o usuário foi redirecionado de volta com erro
        $response->assertRedirect()->assertSessionHas('error', 'Acesso
não autorizado!');
    }

    public function test_admin_pode_criar_paciente()
    {
        // Criar um usuário administrador
        $user = User::factory()->create([
            'nivel_acesso' => 'admin',
        ]);

        // Simular login com o usuário administrador
        $this->actingAs($user);

        // Tentar criar um paciente
        $response = $this->post(route('pacientes.store'), [
            'nome' => 'Carlos Souza',
```

```
            'contacto' => '987654321',
            'idade' => 25,
            'sexo' => 'Masculino',
            'estado_medico' => 'Saudável',
            'consultas' => 'Nenhuma',
            'diagnostico' => 'Nenhum',
            'medicamentos' => 'Nenhum',
        ]);

        // Verificar que o usuário foi redirecionado com sucesso
        $response->assertRedirect(route('pacientes.index'))
            ->assertSessionHas('success', 'Paciente criado com
sucesso!');

        // Recuperar o paciente diretamente do banco de dados
        $paciente = Paciente::where('nome', 'Carlos Souza')->first();

        // Garantir que o paciente foi criado
        $this->assertNotNull($paciente);

        // Verificar os campos do paciente, incluindo desencriptação
para os campos encriptados
        $this->assertEquals('Carlos Souza', $paciente->nome);
        $this->assertEquals('987654321', $paciente->contacto);
        $this->assertEquals(25, $paciente->idade);
        $this->assertEquals('Masculino', $paciente->sexo);
        $this->assertEquals('Saudável', $paciente->estado_medico);
        $this->assertEquals('Nenhuma', $paciente->consultas);
        $this->assertEquals('Nenhum', $paciente->diagnostico);
        $this->assertEquals('Nenhum', $paciente->medicamentos);
    }

}
```

Teste que verifica a criacao de logs de sucessos e de erros

php artisan make:test PacienteAccessTest –unit

```
<?php

namespace Tests\Unit;

use App\Models\Paciente;
use App\Models\Log;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
```

```php
use Tests\TestCase;

class PacienteLogTest extends TestCase
{
    use RefreshDatabase;

    public function test_log_creation_paciente()
    {
        // Criar um usuário administrador
        $user = User::factory()->create([
            'nivel_acesso' => 'admin',
        ]);

        // Simular login do administrador
        $this->actingAs($user);

        // Dados do paciente a ser criado
        $pacienteData = [
            'nome' => 'Mariana Oliveira',
            'contacto' => '234567890',
            'idade' => 28,
            'sexo' => 'Feminino',
            'estado_medico' => 'Saudável',
            'consultas' => 'Nenhuma',
            'diagnostico' => 'Nenhum',
            'medicamentos' => 'Nenhum',
        ];

        // Enviar a requisição para criar o paciente
        $this->post(route('pacientes.store'), $pacienteData);

        // Recuperar o paciente recém-criado
        $paciente = Paciente::where('nome', 'Mariana Oliveira')->first();
        $this->assertNotNull($paciente);

        // Verificar se o log foi registrado
        $this->assertDatabaseHas('logs', [
            'model' => 'App\\Models\\Paciente',
            'model_id' => $paciente->id,
            'action' => 'create',
            'message' => 'Paciente criado com sucesso.',
        ]);

        // Validar que o log contém os dados corretos
        $log = Log::where('model', 'App\\Models\\Paciente')
            ->where('model_id', $paciente->id)
            ->first();

        $this->assertNotNull($log);
        $this->assertEquals('create', $log->action);
```

```
        $this->assertEquals('Paciente criado com sucesso.', $log-
>message);
        $this->assertEquals($paciente->id, $log->model_id);
        $this->assertEquals('App\\Models\\Paciente', $log->model);
    }
}
```

Para executar os testes

```
php artisan test
```