LT Cloud

Laravel + Livewire

Documentacao Tecnica Completa

Explicacao Detalhada da Implementacao

Lourenco Henrique Neves Pereira
Desenvolvedor Full Stack
Email: lourencohenriquenp@gmail.com

18 de agosto de 2025

Conteúdo

1 Introducao

Ola! Sou Lourenco Henrique Neves Pereira e desenvolvi este projeto como parte do processo seletivo da LT Cloud. Esta documentacao apresenta uma explicacao completa de como implementei a mini-aplicacao Laravel + Livewire, detalhando cada aspecto tecnico e as decisoes arquiteturais tomadas durante o desenvolvimento.

O projeto foi desenvolvido com foco em demonstrar dominio das tecnologias Laravel, Livewire, design responsivo, boas praticas de seguranca e versionamento Git. Ao longo desta documentacao, explico como cada requisito foi atendido e como estruturei o codigo para garantir escalabilidade, manutenibilidade e seguranca.

1.1 Objetivos Tecnicos Alcancados

- **CRUD completo** para Desenvolvedores e Artigos
- **Relacionamento muitos-para-muitos** entre entidades
- **Autenticacao robusta** com Laravel Breeze
- **Interface responsiva** com Tailwind CSS
- **Filtros em tempo real** usando Livewire
- **Sistema de permissoes** com Policies
- **Tema escuro/claro persistente** como diferencial

2 Arquitetura e Estrutura do Projeto

2.1 Decisoes Arquiteturais

Para este projeto, optei por uma arquitetura que combina o padrao MVC tradicional do Laravel com a reatividade do Livewire. Esta escolha me permitiu criar uma aplicacao com comportamento SPA (Single Page Application) mantendo a simplicidade do desenvolvimento backend.

2.1.1 Stack Tecnologica Escolhida

- **Laravel 10**: Framework PHP robusto para o backend
- **Livewire 3**: Para componentes reativos sem JavaScript complexo
- **Tailwind CSS**: Framework CSS utility-first para design responsivo
- **Alpine.js**: JavaScript minimalista para interacoes simples
- **SQLite**: Banco de dados para desenvolvimento (compativel com MySQL/PostgreSQL)

2.2 Estrutura de Diretorios

Organizei o projeto seguindo as convencoes do Laravel, com algumas customizacoes para melhor organizacao:

```
app/
        Http/Controllers/          # Controllers tradicionais
        Livewire/                  # Componentes Livewire
            Developer/             # CRUD de Desenvolvedores
            Article/               # CRUD de Artigos
            ThemeToggle.php        # Toggle de tema
        Models/                    # Modelos Eloquent
        Policies/                  # Politicas de autorizacao
        View/Components/           # Componentes Blade
```

3 Implementacao dos Models

3.1 Model Developer

O model `Developer` foi o primeiro que implementei, focando na estrutura de dados e relacionamentos:

Listing 1: app/Models/Developer.php - Estrutura principal

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Developer extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'email',
        'seniority',
        'skills',
        'user_id',
    ];

    protected $casts = [
        'skills' => 'array',  // Converte JSON para array automaticamente
    ];
```

3.1.1 Decisoes Tecnicas do Model Developer

1. **Campo skills como JSON**: Optei por armazenar as skills como JSON no banco, usando o cast automatico do Laravel para array. Isso me permitiu flexibilidade para adicionar/remover skills sem precisar de uma tabela relacionada.

2. **Relacionamento com User**: Implementei um relacionamento belongsTo para garantir que cada desenvolvedor pertenca a um usuario especifico, fundamental para o isolamento de dados.

3. **Enum para senioridade**: Defini as opcoes Jr/Pl/Sr diretamente no migration como enum, garantindo consistencia de dados.

3.2 Model Article

O model `Article` implementa funcionalidades mais complexas, incluindo geracao automatica de slug:

Listing 2: app/Models/Article.php - Funcionalidades principais

```
protected $fillable = [
    'title',
    'slug',
    'content',
    'cover_image',
    'published_at',
    'user_id',
];

protected $casts = [
    'published_at' => 'date',
];

public function setTitleAttribute($value)
{
    $this->attributes['title'] = $value;
    $this->attributes['slug'] = Str::slug($value);
}

public function developers(): BelongsToMany
{
    return $this->belongsToMany(Developer::class);
}
```

3.2.1 Implementacao do Slug Automatico

Implementei um mutator que gera automaticamente o slug baseado no titulo. Esta abordagem garante URLs amigaveis e SEO-friendly:

- **Mutator setTitleAttribute**: Intercepta a definicao do titulo e gera o slug automaticamente
- **Uso do Str::slug()**: Utiliza o helper do Laravel para conversao segura
- **Unique constraint**: O slug tem constraint unique no banco de dados

4 Sistema de Autenticacao

4.1 Laravel Breeze - Escolha e Implementacao

Para o sistema de autenticacao, optei pelo Laravel Breeze por ser uma solucao completa, leve e moderna. O Breeze me forneceu:

- Sistema completo de login/registro
- Reset de senha por email
- Verificacao de email
- Interface responsiva pronta
- Integracao com Tailwind CSS

4.2 Customizacoes Realizadas

Personalizei as views do Breeze para manter consistencia com o design do projeto:

Listing 3: resources/views/auth/login.blade.php - Customizacao

```
<x-guest-layout>
    <!-- Session Status -->
    <x-auth-session-status class="mb-4" :status="session('status')" />

    <form method="POST" action="{{ route('login') }}">
        @csrf

        <!-- Email Address -->
        <div>
            <x-input-label for="email" :value="__('Email')" />
            <x-text-input id="email" class="block mt-1 w-full"
                          type="email" name="email" :value="old('email')"
                          required autofocus />
            <x-input-error :messages="$errors->get('email')" class="mt-2
               " />
        </div>
```

5 Componentes Livewire

5.1 Arquitetura dos Componentes

Estruturei os componentes Livewire seguindo o padrao CRUD, com um componente para cada operacao principal:

- **Index**: Listagem com filtros e paginacao
- **Create**: Formulario de criacao
- **Edit**: Formulario de edicao

5.2 Developer Index Component

O componente de listagem dos desenvolvedores implementa funcionalidades avancadas de busca e filtro:

Listing 4: app/Livewire/Developer/Index.php - Filtros em tempo real

```php
<?php
namespace App\Livewire\Developer;

use App\Models\Developer;
use Livewire\Component;
use Livewire\WithPagination;

class Index extends Component
{
    use WithPagination;

    public $search = '';
    public $skillFilter = '';
    public $seniorityFilter = '';

    protected $queryString = [
        'search' => ['except' => ''],
        'skillFilter' => ['except' => ''],
        'seniorityFilter' => ['except' => ''],
    ];

    public function updatingSearch()
    {
        $this->resetPage();
    }

    public function render()
    {
        $developers = Developer::where('user_id', auth()->id())
            ->when($this->search, function ($query) {
                $query->where('name', 'like', '%' . $this->search . '%')
                      ->orWhere('email', 'like', '%' . $this->search . '%');
            })
            ->when($this->skillFilter, function ($query) {
                $query->whereJsonContains('skills', $this->skillFilter);
            })
            ->when($this->seniorityFilter, function ($query) {
                $query->where('seniority', $this->seniorityFilter);
            })
            ->withCount('articles')
            ->paginate(12);

        return view('livewire.developer.index', compact('developers'))
            ->layout('layouts.app');
    }
}
```

5.2.1 Funcionalidades Implementadas

1. **Busca em tempo real**: Utilizo o wire:model.live para atualizar resultados enquanto o usuario digita
2. **Filtro por skills**: Implementei busca em campo JSON usando whereJsonContains()
3. **Query String**: Os filtros sao mantidos na URL, permitindo compartilhamento e navegacao
4. **Isolamento de dados**: Sempre filtro por user_id para garantir seguranca
5. **Contagem de relacionamentos**: Uso withCount() para exibir quantos artigos cada desenvolvedor possui

5.3 Developer Create Component

O formulario de criacao implementa validacao robusta e interface dinamica para skills:

Listing 5: app/Livewire/Developer/Create.php - Validacao e Skills

```
protected $rules = [
    'name' => 'required|string|max:255',
    'email' => 'required|email|unique:developers,email',
    'seniority' => 'required|in:Jr,Pl,Sr',
    'skills' => 'required|array|min:1',
];

public function addSkill()
{
    if ($this->newSkill && !in_array($this->newSkill, $this->skills)) {
        $this->skills[] = $this->newSkill;
        $this->newSkill = '';
    }
}

public function removeSkill($index)
{
    unset($this->skills[$index]);
    $this->skills = array_values($this->skills);
}

public function save()
{
    $this->validate();

    Developer::create([
        'name' => $this->name,
        'email' => $this->email,
        'seniority' => $this->seniority,
        'skills' => $this->skills,
        'user_id' => auth()->id(),
    ]);

    session()->flash('message', 'Desenvolvedor criado com sucesso!');
    return redirect()->route('developers.index');
}
```

5.3.1 Interface Dinamica para Skills

Implementei um sistema que permite adicionar e remover skills dinamicamente:

- **addSkill()**: Adiciona nova skill se nao existir na lista
- **removeSkill()**: Remove skill por indice e reorganiza array
- **Validacao**: Garante que pelo menos uma skill seja informada
- **Interface reativa**: Atualiza em tempo real sem refresh da pagina

6 Upload de Arquivos

6.1 Implementacao do Upload de Imagens

Para o upload de imagens de capa dos artigos, implementei um sistema robusto usando as funcionalidades nativas do Livewire:

Listing 6: app/Livewire/Article/Create.php - Upload de imagens

```
use Livewire\WithFileUploads;

class Create extends Component
{
    use WithFileUploads;

    public $cover_image;

    protected $rules = [
        'cover_image' => 'nullable|image|max:2048',
    ];

    public function save()
    {
        $this->validate();

        $coverImagePath = null;
        if ($this->cover_image) {
            $coverImagePath = $this->cover_image->store('covers', '
               public');
        }

        $article = Article::create([
            'cover_image' => $coverImagePath ? basename($coverImagePath)
                : null,
            'user_id' => auth()->id(),
        ]);
    }
}
```

6.1.1 Seguranca e Validacao

Implementei varias camadas de seguranca para upload:

1. **Validacao de tipo**: Apenas imagens sao aceitas

2. **Limite de tamanho**: Maximo 2MB por arquivo
3. **Storage seguro**: Arquivos salvos no disco public com nomes unicos
4. **Cleanup**: Remocao de arquivos quando artigo e deletado

7 Sistema de Relacionamentos

7.1 Relacionamento Many-to-Many

O relacionamento entre Artigos e Desenvolvedores foi implementado usando a tabela pivot padrao do Laravel:

Listing 7: database/migrations/article$_d$eveloper$_t$able.php

```
Schema::create('article_developer', function (Blueprint $table) {
    $table->id();
    $table->foreignId('article_id')->constrained()->onDelete('cascade');
    $table->foreignId('developer_id')->constrained()->onDelete('cascade');
    $table->timestamps();
});
```

7.2 Implementacao no Frontend

No formulario de criacao/edicao de artigos, implementei uma interface para selecao multipla:

Listing 8: resources/views/livewire/article/create.blade.php - Selecao multipla

```
<div>
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
        Desenvolvedores *
    </label>
    <div class="space-y-2 max-h-48 overflow-y-auto border rounded-md p-3">
        @foreach($developers as $developer)
            <label class="flex items-center space-x-2">
                <input type="checkbox"
                       wire:model="selectedDevelopers"
                       value="{{ $developer->id }}"
                       class="rounded border-gray-300">
                <span class="text-sm">{{ $developer->name }}</span>
                <span class="text-xs text-gray-500">({{ $developer->seniority }})</span>
            </label>
        @endforeach
    </div>
</div>
```

7.2.1 Sincronizacao dos Relacionamentos

No backend, utilizo o metodo sync() do Eloquent para gerenciar os relacionamentos:

Listing 9: Sincronizacao de relacionamentos

```
// Na criacao
$article->developers()->sync($this->selectedDevelopers);

// Na edicao
$this->article->developers()->sync($this->selectedDevelopers);
```

8 Sistema de Seguranca e Policies

8.1 Implementacao das Policies

Criei Policies para garantir que usuarios so acessem seus proprios dados:

Listing 10: app/Policies/DeveloperPolicy.php

```
<?php
namespace App\Policies;

use App\Models\Developer;
use App\Models\User;

class DeveloperPolicy
{
    public function view(User $user, Developer $developer): bool
    {
        return $user->id === $developer->user_id;
    }

    public function update(User $user, Developer $developer): bool
    {
        return $user->id === $developer->user_id;
    }

    public function delete(User $user, Developer $developer): bool
    {
        return $user->id === $developer->user_id;
    }
}
```

8.2 Aplicacao das Policies

As policies sao aplicadas automaticamente nos componentes Livewire:

Listing 11: Verificacao de permissoes nos componentes

```
// No Edit component
public function mount(Developer $developer)
{
    if ($developer->user_id !== auth()->id()) {
        abort(403);
    }

    $this->developer = $developer;
}

// Nas queries sempre filtro por user_id
```

```
$developers = Developer::where('user_id', auth()->id())
    ->when($this->search, function ($query) {
        // ... filtros
    })
    ->paginate(12);
```

9 Interface Responsiva

9.1 Design System com Tailwind

Implementei um design system consistente usando classes utilitarias do Tailwind CSS:

Listing 12: Exemplo de card responsivo

```
<!-- Grid responsivo -->
<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols
    -4 gap-6">
    @foreach($developers as $developer)
        <div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-6
            hover:shadow-lg transition-shadow">
            <!-- Conteudo do card -->
        </div>
    @endforeach
</div>
```

9.2 Tema Escuro/Claro

Implementei um sistema de tema persistente usando cookies:

Listing 13: resources/views/livewire/theme-toggle.blade.php - Sistema de tema

```
<script>
function applyTheme(isDark) {
    const html = document.documentElement;

    if (isDark) {
        html.classList.add('dark');
    } else {
        html.classList.remove('dark');
    }

    // Salvar no cookie
    setCookie('darkMode', isDark ? 'true' : 'false', 365);
}

function toggleTheme() {
    const isDark = document.documentElement.classList.contains('dark');
    applyTheme(!isDark);
}
</script>
```

9.2.1 Configuracao do Tailwind para Dark Mode

No arquivo de configuracao do Tailwind, habilitei o modo escuro baseado em classe:

Listing 14: tailwind.config.js

```
export default {
    darkMode: 'class',  // Permite controle via classe CSS
    content: [
        './resources/views/**/*.blade.php',
    ],
    // ... resto da configuracao
};
```

10 Banco de Dados

10.1 Migrations

Criei migrations bem estruturadas para todas as entidades:

Listing 15: Migration dos desenvolvedores

```
Schema::create('developers', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->enum('seniority', ['Jr', 'Pl', 'Sr']);
    $table->json('skills');
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->timestamps();
});
```

10.2 Factories para Dados Fake

Implementei factories robustas usando Faker para gerar dados realistas:

Listing 16: database/factories/DeveloperFactory.php

```
public function definition(): array
{
    $skills = [
        'PHP', 'Laravel', 'JavaScript', 'Vue.js', 'React',
        'Node.js', 'MySQL', 'PostgreSQL', 'Docker', 'Git'
    ];

    return [
        'name' => fake()->name(),
        'email' => fake()->unique()->safeEmail(),
        'seniority' => fake()->randomElement(['Jr', 'Pl', 'Sr']),
        'skills' => fake()->randomElements($skills, fake()->
            numberBetween(2, 6)),
        'user_id' => 1,
    ];
}
```

10.3 Seeders

Criei seeders que geram dados de demonstracao consistentes:

Listing 17: database/seeders/DatabaseSeeder.php

```
public function run(): void
{
    // Criar usuarios de teste
    $user1 = User::factory()->create([
        'name' => 'Demo User',
        'email' => 'demo@ltcloud.com.br',
    ]);

    $user2 = User::factory()->create([
        'name' => 'Test User',
        'email' => 'test@example.com',
    ]);

    // Criar desenvolvedores para cada usuario
    Developer::factory(5)->create(['user_id' => $user1->id]);
    Developer::factory(3)->create(['user_id' => $user2->id]);

    // Criar artigos e relacionamentos
    Article::factory(4)->create(['user_id' => $user1->id]);
    Article::factory(2)->create(['user_id' => $user2->id]);
}
```

11 Funcionalidades Avancadas

11.1 Badges de Contagem

Implementei um sistema de badges que mostra relacionamentos:

Listing 18: Contagem de relacionamentos

```
<!-- No card do desenvolvedor -->
<span class="text-sm text-gray-600 dark:text-gray-300">
    {{ $developer->articles_count }} artigo(s)
</span>

<!-- No card do artigo -->
<span class="text-sm text-gray-600 dark:text-gray-300">
    {{ $article->developers_count }} desenvolvedor(es)
</span>
```

As contagens sao feitas usando withCount() do Eloquent, que e eficiente e nao gera queries N+1.

11.2 Busca em Campos JSON

Para buscar skills especificas, utilizo o metodo whereJsonContains():

Listing 19: Busca em campo JSON

```
->when($this->skillFilter, function ($query) {
    $query->whereJsonContains('skills', $this->skillFilter);
})
```

Esta funcionalidade permite buscar desenvolvedores que possuem uma skill especifica, mesmo estando armazenada em array JSON.

12 Otimizacoes de Performance

12.1 Eager Loading

Implementei eager loading para evitar queries N+1:

Listing 20: Prevencao de queries N+1

```
// Ao carregar artigos com desenvolvedores
$articles = Article::with('developers')
    ->where('user_id', auth()->id())
    ->paginate(12);

// Ao carregar desenvolvedores com contagem de artigos
$developers = Developer::withCount('articles')
    ->where('user_id', auth()->id())
    ->paginate(12);
```

12.2 Paginacao Eficiente

Utilizo paginacao em todas as listagens para garantir performance:

Listing 21: Paginacao com Livewire

```
use Livewire\WithPagination;

class Index extends Component
{
    use WithPagination;

    public function render()
    {
        $developers = Developer::where('user_id', auth()->id())
            ->paginate(12);  // 12 itens por pagina

        return view('livewire.developer.index', compact('developers'));
    }
}
```

13 Testes e Qualidade

13.1 Validacoes Robustas

Implementei validacoes em multiplas camadas:

Listing 22: Validacoes no frontend e backend

```
// Validacao no componente Livewire
protected $rules = [
    'name' => 'required|string|max:255',
    'email' => 'required|email|unique:developers,email',
    'seniority' => 'required|in:Jr,Pl,Sr',
    'skills' => 'required|array|min:1',
];

// Mensagens customizadas
```

```
protected $messages = [
    'name.required' => 'O nome e obrigatorio.',
    'email.unique' => 'Este email ja esta cadastrado.',
    'skills.min' => 'Pelo menos uma skill e obrigatoria.',
];
```

13.2 Tratamento de Erros

Implementei tratamento consistente de erros em toda aplicacao:

- **Validacao de formularios**: Mensagens claras para o usuario
- **Autorizacao**: Redirecionamento 403 para acessos negados
- **Upload de arquivos**: Validacao de tipo e tamanho
- **Feedback visual**: Mensagens de sucesso e erro

14 Deploy e Configuracao

14.1 Configuracao de Ambiente

Preparei o projeto para deploy em diferentes ambientes:

Listing 23: Configuracoes de ambiente

```
# .env para desenvolvimento
DB_CONNECTION=sqlite
DB_DATABASE=/absolute/path/to/database.sqlite

# .env para producao
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ltcloud
DB_USERNAME=root
DB_PASSWORD=
```

14.2 Assets e Build

Configurei o Vite para compilacao otimizada:

Listing 24: vite.config.js

```
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,
        }),
    ],
});
```

15 Conclusao

15.1 Requisitos Atendidos

Durante o desenvolvimento, consegui atender todos os requisitos especificados:

- **[OK] Autenticacao completa** com Laravel Breeze
- **[OK] CRUD de Desenvolvedores** com filtros em tempo real
- **[OK] CRUD de Artigos** com upload de imagens
- **[OK] Relacionamento N:N** entre entidades
- **[OK] Interface responsiva** com Tailwind CSS
- **[OK] Policies** para isolamento de dados
- **[OK] Migrations, Seeders e Factories**
- **[OK] Tema escuro/claro** como diferencial

15.2 Desafios Tecnicos Superados

Durante o desenvolvimento, enfrentei e resolvi diversos desafios:

1. **Gerenciamento de skills dinamicas**: Solucionei criando interface reativa para adicionar/remover skills em tempo real
2. **Upload de arquivos com Livewire**: Implementei sistema robusto com validacao e cleanup automatico
3. **Filtros complexos em JSON**: Utilizei whereJsonContains para busca eficiente em campos JSON
4. **Tema persistente**: Criei sistema com JavaScript vanilla e cookies para persistencia
5. **Isolamento de dados**: Implementei Policies e filtros consistentes em toda aplicacao

15.3 Aspectos Tecnicos Destacados

Alguns pontos que considero diferenciais na implementacao:

- **Codigo limpo e bem documentado**: Seguindo PSR-12 e boas praticas
- **Arquitetura escalavel**: Componentes reutilizaveis e bem estruturados
- **UX moderna**: Interface responsiva e intuitiva
- **Seguranca robusta**: Multiplas camadas de protecao
- **Performance otimizada**: Queries eficientes e caching adequado

15.4 Consideracoes Finais

Este projeto demonstra minha capacidade de desenvolver aplicacoes web completas usando tecnologias modernas. A implementacao combina boas praticas de desenvolvimento, arquitetura solida e experiencia do usuario otimizada.

A escolha do Laravel + Livewire se mostrou acertada, permitindo criar uma aplicacao reativa mantendo a simplicidade do desenvolvimento backend. O uso do Tailwind CSS possibilitou uma interface moderna e totalmente responsiva.

Estou satisfeito com o resultado final, que atende nao apenas aos requisitos tecnicos, mas tambem demonstra atencao aos detalhes, preocupacao com seguranca e foco na experiencia do usuario.