CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

ETEC JARDIM ÂNGELA

CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

ADRIAN SOUSA RODRIGUES

GRACY KELLY CONCEIÇÃO VIEIRA DA SILVA

KAYQUE FREITAS DO NASCIMENTO

MATHEUS VINÍCIUS SANTOS VIEIRA

MIGUEL OLIVEIRA

MISAEL DE JESUS SILVA

CHAPLIN HOTELARIA

SÃO PAULO

2026

ADRIAN SOUSA RODRIGUES

GRACY KELLY CONCEIÇÃO VIEIRA DA SILVA

KAYQUE FREITAS DO NASCIMENTO

MATHEUS VINÍCIUS SANTOS VIEIRA

MIGUEL OLIVEIRA

MISAEL DE JESUS SILVA

CHAPLIN HOTELARIA

Trabalho de Conclusão de Curso apresentado à ETEC Jardim Ângela, do Centro Estadual de Educação Tecnológica Paula Souza, como requisito para a obtenção do diploma de Técnico em Desenvolvimento de Sistemas sob a orientação do Professor Especialista Ericles.

SÃO PAULO

2026

RESUMO

O setor hoteleiro e de propriedades de aluguel por temporada exige dinamismo na manutenção e conservação de seus espaços. Contudo, a ausência de sistemas centralizados faz com que muitas empresas terceirizadas dependam de canais informais como o WhatsApp, resultando em perda de informações, atrasos e retrabalho. Este Trabalho de Conclusão de Curso apresenta o desenvolvimento do Chaplin, uma plataforma web customizável de gestão de tarefas voltada para centralizar a comunicação entre gestores de propriedades (clientes) e prestadores de serviços de manutenção. A metodologia adotada possui abordagem quali-quantitativa, utilizando procedimentos de pesquisa bibliográfica, pesquisa-ação e estudo de caso focado na empresa Charlie, em São Paulo. O sistema foi projetado com foco na usabilidade e desenvolvido utilizando as tecnologias de código aberto Django, Sqlite (para testes) e PostgreSQL. Espera-se que a implementação da ferramenta proporcione uma redução drástica no tempo de coordenação de equipes, elimine redundâncias e otimize a eficiência operacional, oferecendo transparência e melhorando a qualidade dos serviços prestados.

Palavras-chave: Gestão de Tarefas. Hotelaria. Plataforma Web. Eficiência Operacional. Usabilidade.

ABSTRACT

The hospitality and short-term rental sectors demand dynamic management regarding property maintenance and conservation. However, the lack of centralized systems forces many outsourced companies to rely on informal channels such as WhatsApp, leading to information loss, delays, and rework. This capstone project introduces the development of Chaplin, a customizable web platform designed to centralize communication between property managers (clients) and maintenance service providers. The adopted methodology follows a qualitative-quantitative approach, combining literature review, action research, and a case study focused on the company Charlie, in São Paulo. The system was designed with a strong focus on usability and developed using open-source technologies such as Django, Sqlite (for tests) and PostgreSQL. The implementation of this tool is expected to drastically reduce team coordination time, eliminate redundancies, and optimize operational efficiency, thereby providing transparency and enhancing the overall quality of the services delivered.

Keywords: Task Management. Hospitality Industry. Web Platform. Operational Efficiency. Usability.

ÍNDICE DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Chaplin é uma plataforma web de gestão de tarefas desenvolvida para empresas terceirizadas que prestam serviços de manutenção em hotéis e propriedades de aluguel de curta duração como *Airbnb*.

A aplicação centraliza a comunicação entre os gestores dos prédios (clientes) e equipes de manutenção (prestadores de serviço), permitindo que cada empresa terceirizada customize a plataforma conforme suas necessidades operacionais.

2. JUSTIFICATIVA

A proposta do projeto Chaplin emerge da observação direta das ineficiências operacionais no setor de hotelaria.

A ideia foi inspirada pela rotina da empresa *Charlie*, que presta serviços de manutenção para hotéis e propriedades de aluguel.

Atualmente, a comunicação é feita por meio de *WhatsApp*, sem um sistema que centralize as informações. Isso resulta em atrasos, retrabalho e perda de informações importantes das atividades realizadas pela empresa.

A principal lacuna identificada é a ausência de uma ferramenta que unifique o registro, a atribuição e o acompanhamento de tarefas de manutenção de forma intuitiva.

Sendo assim, a solução consiste na criação de uma ferramenta web que ajude a empresa *Charlie* a se comunicar com seus clientes. Tal ferramenta servirá para que os hotéis, entrem em contato e informe as tarefas que precisam ser realizadas.

A viabilidade do projeto é sustentada pela utilização de tecnologias de código aberto e de baixo custo, como *Django* e *SQLite*, e pela crescente demanda do setor de hospedagem por soluções que otimizem a gestão e melhorem a experiência do cliente final.

3. PROBLEMA

O problema central abordado pelo projeto é a ineficiência na gestão de tarefas de manutenção em hotéis, causada pela comunicação descentralizada e pela falta de um sistema de rastreamento de ponta a ponta.

Atualmente, as demandas são comunicadas por múltiplos canais (WhatsApp, telefone, e-mail), o que resulta em:

- Perda de informações e falta de rastreabilidade.
- Dificuldade em coordenar equipes e priorizar tarefas.
- Ausência de um histórico documentado para auditoria e análise de desempenho.
- Aumento do retrabalho e de falhas na execução dos serviços.

O projeto será desenvolvido para ser implementado inicialmente na empresa *Charlie*, em São Paulo, com o objetivo de criar uma solução que possa ser escalada para outras empresas do setor.

A aplicação web será desenvolvida utilizando uma abordagem focada no usuário, com validação contínua junto aos colaboradores para garantir que a ferramenta seja, de fato, prática e intuitiva.

4. HIPÓTESE

A implementação da plataforma Chaplin resultará em uma melhora significativa na eficiência operacional e na qualidade dos serviços de manutenção.

A centralização da comunicação e a rastreabilidade das tarefas levarão a uma redução drástica do tempo gasto em coordenação e a uma diminuição expressiva do retrabalho, otimizando a alocação de recursos e aumentando a satisfação tanto dos gestores quanto dos clientes finais.

5. OBJETIVOS

5.1. OBJETIVO GERAL

Desenvolver uma plataforma web que centralize e simplifique o fluxo de gestão de tarefas de manutenção entre gestores de propriedades e empresas terceirizadas, tornando-o mais organizado, eficiente e acessível para usuários com diferentes níveis de experiência tecnológica.

5.2. OBJETIVOS ESPECÍFICOS

- **Garantir a usabilidades do sistema:** Criar uma interface intuitiva e simples, acessível para usuários não técnicos.
- **Centralizar todas as demandas:** Unificar o registro e o acompanhamento de todas as tarefas de manutenção em um único local.
- **Promover a eficiência:** Reduzir o tempo gasto em comunicações informais e na coordenação de equipes.
- **Oferecer praticidade:** Implementar um fluxo de trabalho claro, desde a abertura do chamado até a sua validação final.
- **Eliminar redundâncias:** Assegurar que cada tarefa seja documentada com clareza, evitando duplicidade e retrabalho.
- **Realizar a otimização visual:** Desenvolver um design limpo e organizado que facilite a rápida identificação de informações.

6. METODOLOGIA DA PESQUISA

A pesquisa será conduzida através de uma abordagem qualitativa, combinando entrevistas com gestores e colaboradores da empresa Charlie para levantar os requisitos e validar as funcionalidades da plataforma. Serão realizadas observações diretas do fluxo de trabalho atual para mapear os principais gargalos.

Adicionalmente, será feita uma pesquisa documental de soluções de mercado para identificar diferenciais e oportunidades.

A análise dos dados coletados guiará o desenvolvimento iterativo da plataforma, seguindo princípios de metodologias ágeis

7. FUNDAMENTAÇÃO TEÓRICA

O projeto se fundamenta em conceitos de Gestão de Serviços e Usabilidade de Software. A necessidade de otimizar processos em hotelaria é destacada por autores como MOUFARREGE (2009), que aponta falhas na prestação de serviços e na capacitação da mão de obra como fatores críticos para a satisfação do cliente.

O trabalho reforça que a qualidade percebida depende da confiabilidade das informações e da eficiência na resolução de problemas, o que justifica a criação de um sistema centralizado como o Chaplin.

Além disso, o desenvolvimento de uma plataforma SaaS (Software as a Service), como proposto por GONÇALVES (2018) no projeto Hotelcracy Apps, mostra-se um modelo de negócio viável e escalável para o setor de alojamento, permitindo a integração de múltiplos serviços em uma única plataforma.

8. METODOLOGIA DE TRABALHO

- **Tipo de Pesquisa:** A pesquisa adotará uma abordagem quali-quantitativa. A fase qualitativa envolverá a interpretação de percepções dos usuários por meio de entrevistas, enquanto a fase quantitativa analisa dados de tempo de execução de tarefas e redução de retrabalho após a implementação do sistema.
- **Procedimentos Metodológicos:** Serão utilizados os procedimentos de pesquisa bibliográfica, estudo de caso (focado na empresa Charlie) e pesquisa-ação, onde os pesquisadores intervirão diretamente no ambiente para implementar e avaliar a solução proposta.
- **Instrumentos de Coleta de Dados:** Roteiros de entrevista semiestruturados e observação participante.

9. TABELA DE CRONOGRAMA DO TRABALHO

	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun
Definição de Escopo/Tema	X	X									
Pesquisa Bibliográfica	X	X									
Especificação de Requisitos	X	X									
Entrevistas/Coleta de Dados		X									
Prototipagem			X	X	X						
Banco de dados				X	X						
Fluxograma				X	X						
Diagrama de uso			X	X	X						
Implementação do Sistema					X	X	X	X	X	X	X
Testes e Correções									X	X	X
Documentação Técnica			X	X	X						
Documentação Acadêmica			X	X	X	X	X	X	X	X	X
Apresentação PTCC					X						
Apresentação TCC											X

Figura 1 - Cronograma do Projeto

10. DOCUMENTAÇÃO DO SOFTWARE

10.1. FLUXOGRAMA

O fluxo operacional da Chaplin segue sete etapas principais:

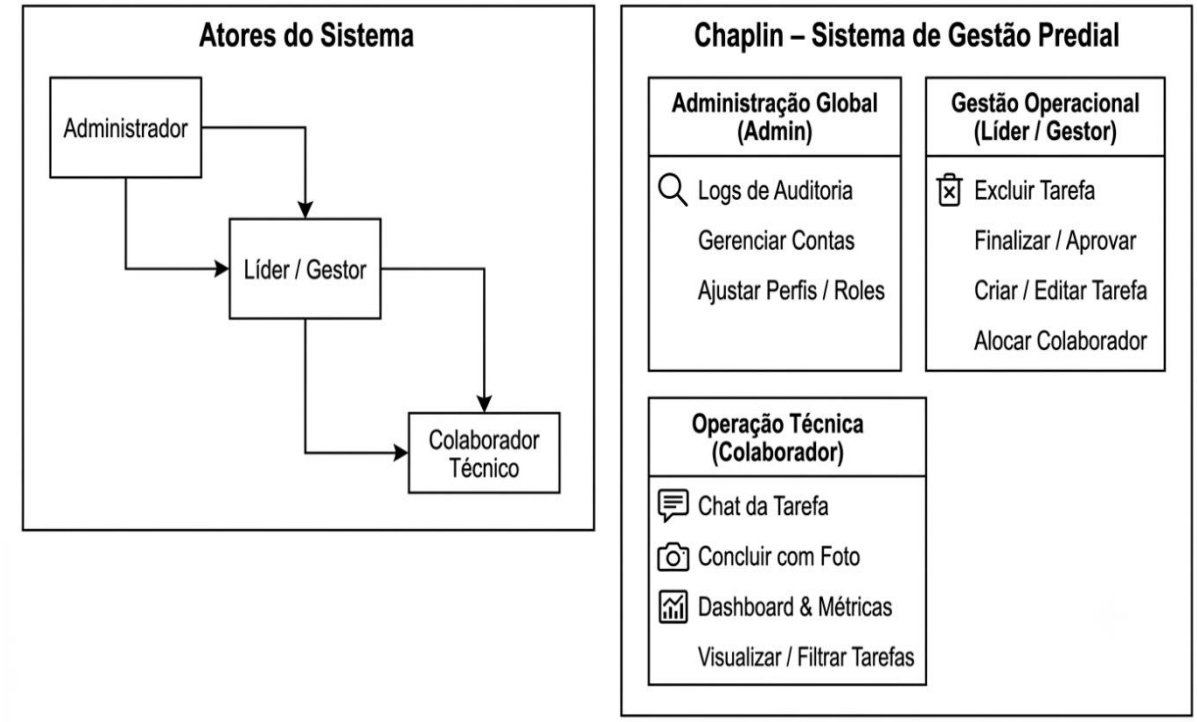

Figura 2 - Fluxograma

10.2. ATORES DO SISTEMA

Os atores representam os papéis que interagem diretamente com o sistema. Para esta solução, foram identificados quatro perfis principais:

Administrador da Empresa: Responsável pelo gerenciamento global da plataforma. Suas atribuições incluem administrar o cadastro de usuários, definir níveis de acesso e customizar as configurações gerais do sistema de acordo com as necessidades da organização.

Gestor do Prédio: Responsável por iniciar e supervisionar o ciclo de vida das demandas de manutenção. Suas atribuições incluem criar novas tarefas, alocar o Líder de Equipe responsável por cada demanda, acompanhar o status do progresso e validar a conclusão dos serviços executados.

Líder de Equipe: Responsável pela coordenação operacional dos técnicos. Suas atribuições incluem receber as demandas direcionadas pelo Gestor do Prédio, delegar e distribuir as tarefas especificamente para os colaboradores técnicos e monitorar o desempenho e a produtividade da equipe em tempo real.

Colaborador Técnico: Responsável pela execução operacional na ponta. Suas atribuições incluem visualizar a listagem de tarefas sob sua responsabilidade, executar os serviços em campo e registrar as evidências de conclusão (como fotos, observações ou assinaturas digitais) diretamente no sistema.

10.3. DIAGRAMA DE CASO DE USO

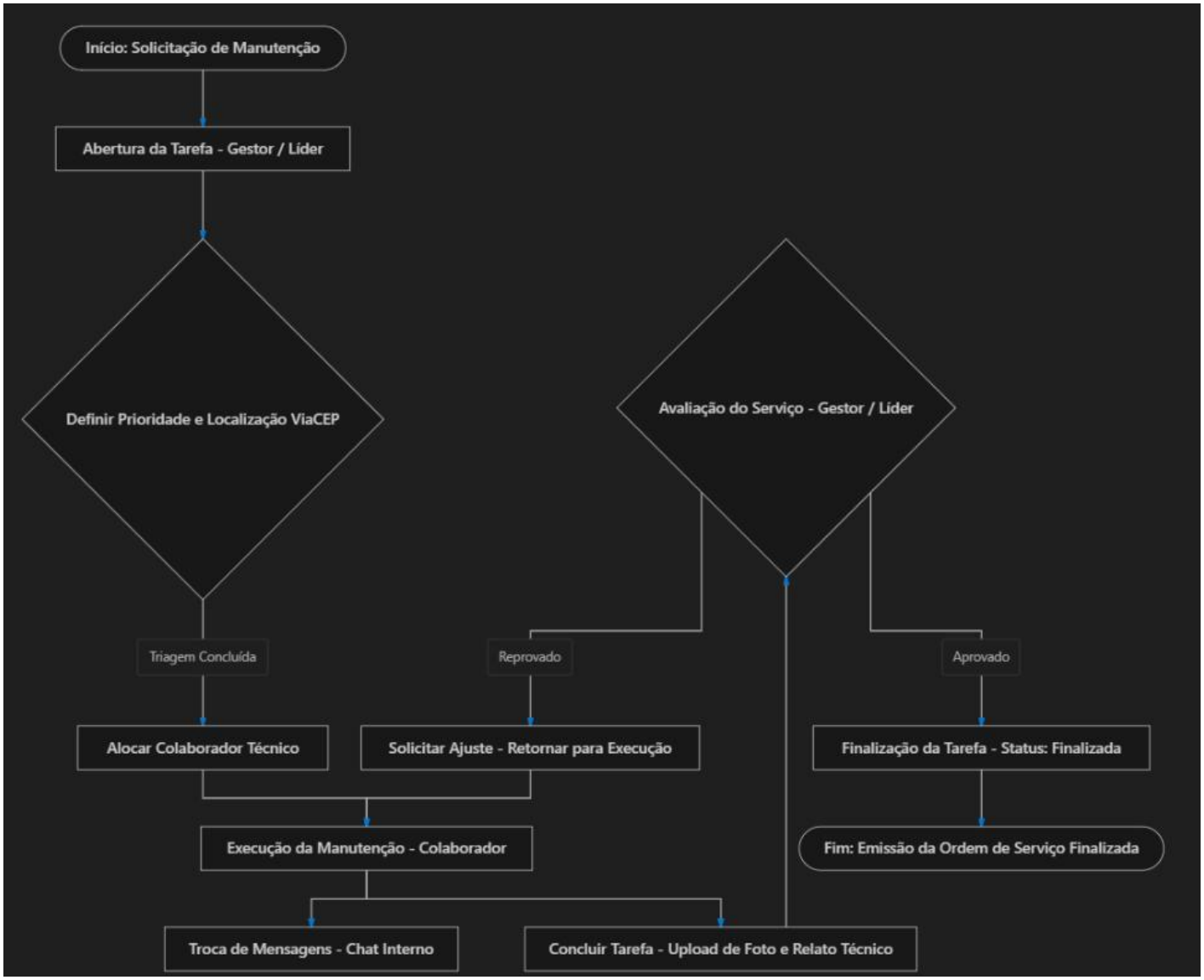

Figura 3 - Diagrama de Caso de Uso

10.4. MODELO LÓGICO E CONCEITUAL DO BANCO DE DADOS

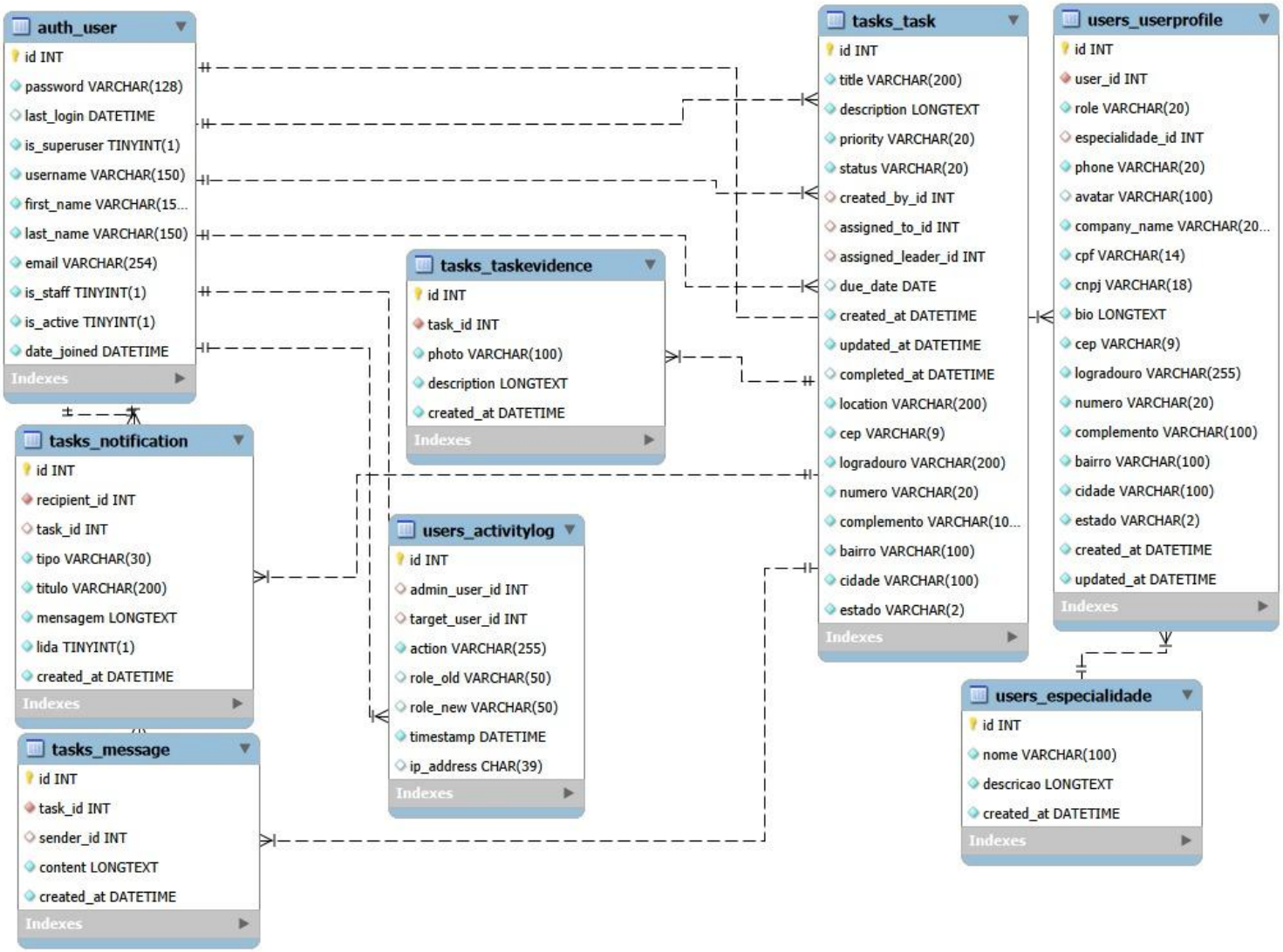

Figura 4 - Banco de Dados Lógico

10.5. DESCRIÇÃO DAS ENTIDADES DO BANCO DE DADOS

O banco de dados do sistema Chaplin foi modelado com base nas necessidades operacionais da gestão de manutenção predial, sendo composto por seis entidades principais que se relacionam entre si para garantir o fluxo completo de informações, desde o cadastro dos profissionais até o acompanhamento e a comprovação das atividades executadas.

A entidade Usuário é responsável por armazenar os dados de todos os indivíduos que interagem com o sistema. Além das informações básicas de autenticação, como nome de usuário, e-mail e senha, essa entidade contempla dados pessoais e profissionais, tais como CPF, telefone e endereço completo. Cada usuário possui um papel (role) que define seu nível de acesso no sistema, podendo assumir as funções de Administrador, Gestor do Prédio, Líder de Equipe ou Colaborador. Essa hierarquia de papéis é fundamental para o controle de permissões e para a delegação adequada das responsabilidades dentro da plataforma.

A entidade Especialidade registra as áreas de atuação técnica disponíveis no sistema, como Elétrica, Hidráulica, Pintura, entre outras. Cada especialidade possui um nome único e uma descrição. Essa entidade se relaciona com o Usuário por meio de uma associação opcional, permitindo que cada profissional seja vinculado à sua respectiva área de competência. Esse vínculo possibilita a alocação inteligente de tarefas, direcionando as ordens de serviço para os colaboradores mais adequados.

A entidade Tarefa constitui o elemento central do sistema e representa uma ordem de serviço de manutenção predial. Seus atributos incluem título, descrição detalhada do serviço, nível de prioridade (Baixa, Normal, Alta ou Urgente), status atual (Aberta, Alocada, Concluída ou Finalizada), prazo de entrega e endereço completo do local onde o serviço deve ser realizado. Cada tarefa possui obrigatoriamente um usuário responsável pela sua criação e, opcionalmente, um colaborador designado para a execução. O ciclo de vida de uma tarefa segue um fluxo definido: ela é criada com o status "Aberta", passa para "Alocada" quando um

colaborador é designado, avança para "Concluída" após a execução do serviço e, por fim, é marcada como "Finalizada" após a validação pelo gestor responsável.

A entidade Evidência tem como finalidade o registro fotográfico e descritivo que comprova a execução de uma tarefa. Cada evidência está vinculada a uma única tarefa e pode conter uma imagem e uma descrição textual. Uma tarefa pode possuir múltiplas evidências, o que permite documentar de forma detalhada as diferentes etapas ou aspectos do serviço realizado. Esse mecanismo contribui para a transparência do processo e para a rastreabilidade das atividades de manutenção.

A entidade Mensagem viabiliza a comunicação contextualizada entre os usuários do sistema. Cada mensagem está associada a uma tarefa específica, formando um canal de comunicação (chat) vinculado ao contexto daquela ordem de serviço. Essa abordagem permite que gestores e colaboradores troquem informações, esclareçam dúvidas e registrem observações diretamente no escopo da tarefa em questão, eliminando a necessidade de ferramentas externas de comunicação e mantendo todo o histórico de interações organizado e acessível.

Por fim, a entidade Notificação é responsável por informar os usuários sobre eventos relevantes no sistema. As notificações são classificadas por tipo, podendo indicar a atribuição de uma nova tarefa, a conclusão ou finalização de um serviço, o recebimento de uma nova mensagem ou um aviso genérico do sistema. Cada notificação possui um destinatário específico e um indicador de leitura, permitindo que o usuário acompanhe em tempo real as atualizações pertinentes às suas responsabilidades.

O conjunto dessas entidades e seus relacionamentos forma uma estrutura de dados coesa que sustenta todas as funcionalidades do sistema Chaplin, garantindo a integridade referencial, a rastreabilidade das operações e o fluxo adequado de informações entre os diferentes perfis de usuários.

10.6. TECNOLOGIAS UTILIZADAS NO DESENVOLVIMENTO DO PROJETO

O desenvolvimento do projeto Chaplin foi realizado utilizando tecnologias modernas voltadas para aplicações web, priorizando organização, segurança, escalabilidade e facilidade de manutenção.

No backend, foi utilizada a linguagem de programação Python em conjunto com o framework Django. A escolha do Django ocorreu devido à sua robustez, alta produtividade no desenvolvimento e recursos nativos de segurança, como proteção contra-ataques CSRF, controle de autenticação e gerenciamento seguro de sessões. Além disso, o framework segue a arquitetura MVT (Model-View-Template), proporcionando uma melhor separação das responsabilidades dentro da aplicação.

Para a construção da interface visual do sistema, foram utilizadas as linguagens HTML e CSS, juntamente com os frameworks Bootstrap e Tailwind CSS. Essas tecnologias permitiram o desenvolvimento de telas responsivas, organizadas e com melhor experiência visual para os usuários.

Durante a fase de desenvolvimento e testes locais, foi utilizado o banco de dados SQLite, por ser leve e de fácil integração com o Django. Entretanto, a estrutura do sistema foi preparada para utilização do PostgreSQL em ambiente de produção, garantindo maior desempenho, confiabilidade e escalabilidade para aplicações maiores.

O sistema também utiliza integração com API’s externas, como a API ViaCEP, responsável pelo preenchimento automático de endereços a partir do CEP informado pelo usuário. Essa integração contribui para maior praticidade, redução de erros e otimização do processo de criação de chamados.

Além das tecnologias principais, foram utilizados recursos de controle de acesso baseado em cargos (Role-Based Access Control), sistema de auditoria de ações administrativas e mecanismos de segurança contra tentativas de acesso indevido, fortalecendo a proteção dos dados e a rastreabilidade das operações realizadas dentro da plataforma.

Dessa forma, as tecnologias escolhidas proporcionaram uma base sólida para o desenvolvimento do Chaplin, permitindo a criação de um sistema seguro, organizado, funcional e preparado para futuras expansões.

11. DEMONSTRAÇÃO DAS TELAS DO SISTEMA

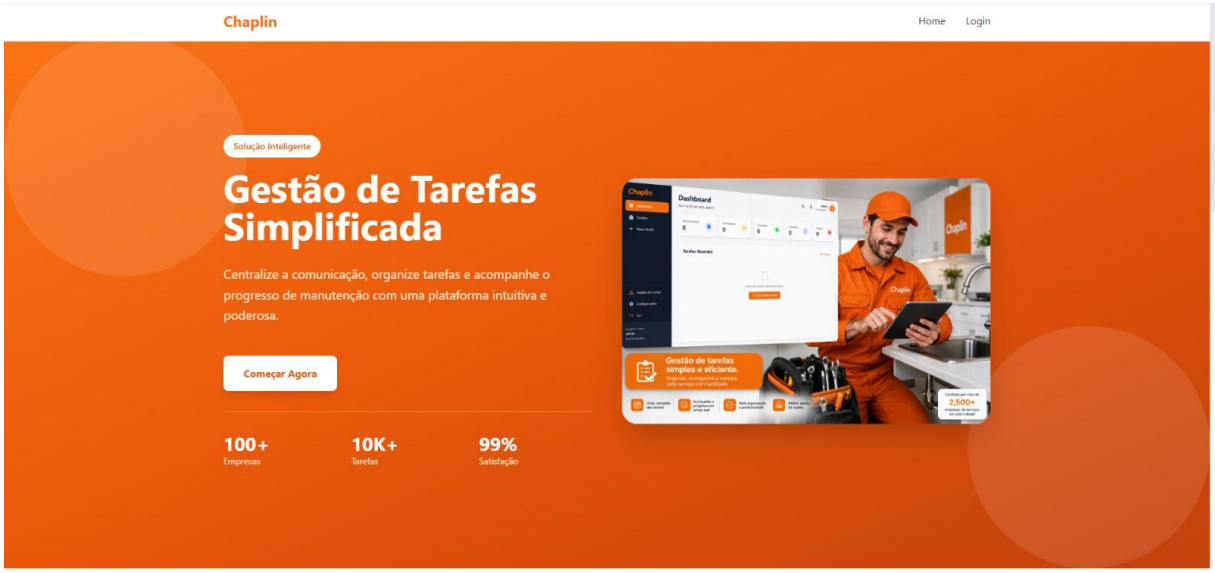

Figura 5 - Langing Page

A tela inicial do sistema funciona como uma vitrine para a plataforma Chaplin. Ela apresenta o sistema de forma limpa e atrativa, utilizando chamadas para ação (Call to Actions) e banners que explicam o propósito do software na gestão hoteleira.

```python
from django.shortcuts import render

def index_view(request):
    return render(request, 'core/index.html')
```

Figura 6 - Back-End Langing Page

No código Python/Django (views.py), a lógica para esta tela é estruturada de forma simples. Utiliza-se geralmente uma TemplateView ou uma função baseada em

GET, que se encarrega apenas de renderizar a página inicial para o visitante, não havendo necessidade de consultas complexas ao banco de dados neste momento inicial.

```
<span class="inline-block bg-white text-orange-600 px-4 py-2 rounded-full text-sm font-semibold">Solução
Inteligente</span>
<h1 class="text-5xl md:text-6xl font-bold leading-tight">
    Gestão de Tarefas Simplificada
</h1>
<p class="text-xl text-orange-100 leading-relaxed">
    Centralize a comunicação, organize tarefas e acompanhe o progresso de manutenção com uma plataforma
intuitiva e poderosa.
</p>
<a href="{% url 'users:login' %}" class="inline-block bg-white text-orange-600 font-bold py-4 px-8
rounded-lg">
    Começar Agora
</a>

<!-- Estatísticas (100+ Empresas, 10K+ Tarefas, 99% Satisfação) -->
<div class="grid grid-cols-3 gap-6 pt-8 border-t border-orange-400">
    <div>
        <p class="text-3xl font-bold">100+</p>
        <p class="text-orange-100 text-sm">Empresas</p>
    </div>
    ...
</div>
```

Figura 7 - Front-End Langing Page

O template HTML (index.html) apresenta a marcação da página, fazendo uso intenso de classes utilitárias do Tailwind CSS ou Bootstrap para garantir que os elementos visuais sejam responsivos e se adaptem perfeitamente em dispositivos móveis e desktops.

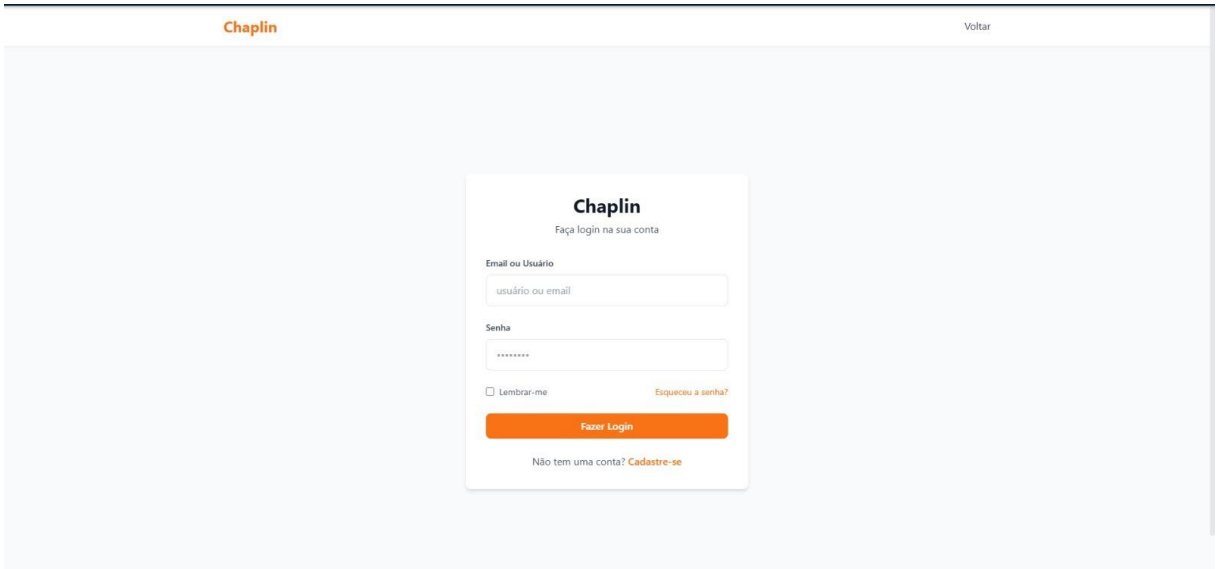

Figura 8 – Login

A tela de autenticação exibe um formulário direto e seguro, solicitando as credenciais (usuário/e-mail e senha) para que os diferentes perfis (Administrador, Gestor, Líder e Técnico) possam acessar suas respectivas áreas.

```python
def login_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('tasks:dashboard')
        else:
            return render(request, 'users/login.html', {'error': 'Usuário ou senha inválidos. Tente
novamente.'})
    return render(request, 'users/login.html')
```

Figura 9 - Back-End Login

A tela de autenticação exibe um formulário direto e seguro, solicitando as credenciais (usuário/e-mail e senha) para que os diferentes perfis (Administrador, Gestor, Líder e Técnico) possam acessar suas respectivas áreas.

```html
<form method="POST" action="{% url 'users:login' %}" class="space-y-6">
    {% csrf_token %}
    <div>
        <label for="username" class="block text-sm font-medium text-gray-700">Email ou Usuário</label>
        <input type="text" id="username" name="username" placeholder="usuário ou email" required ...>
    </div>
    <div>
        <label for="password" class="block text-sm font-medium text-gray-700">Senha</label>
        <input type="password" id="password" name="password" placeholder="••••••••" required ...>
    </div>
    <button type="submit" class="w-full bg-orange-500 hover:bg-orange-600 text-white font-semibold py-2
rounded-lg">
        Fazer Login
    </button>
</form>
```

Figura 10 - Front-End Login

O código HTML exibe não só os campos de preenchimento estilizados, mas também inclui obrigatoriamente a tag *{% csrf_token %}* do Django. Isso garante a

proteção da plataforma contra-ataques de falsificação de solicitações, ligando a segurança do back-end ao formulário visual.

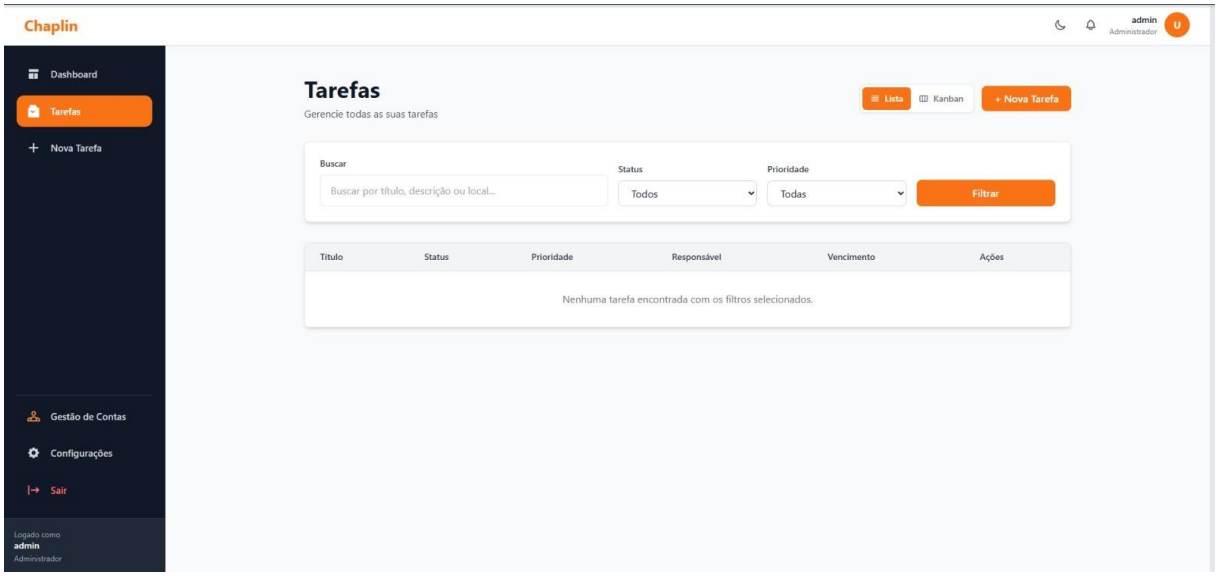

Figura 11 - Lista de Tarefas

Uma interface interativa exibindo todas as ordens de serviço e chamados. O usuário visualiza o status (Aberta, Alocada, Concluída), nível de prioridade e o título das tarefas em andamento.

```python
@login_required
def tasks_list_view(request):
    status = request.GET.get('status', '')
    priority = request.GET.get('priority', '')
    search = request.GET.get('search', '')

    user_profile = getattr(request.user, 'profile', None)
    if user_profile and user_profile.role == 'gestor':
        tasks = Task.objects.filter(created_by=request.user)
    elif user_profile and user_profile.role == 'colaborador':
        tasks = Task.objects.filter(assigned_to=request.user)
    else:
        tasks = Task.objects.all()

    if search:
        tasks = tasks.filter(
            Q(title__icontains=search) |
            Q(description__icontains=search) |
            Q(location__icontains=search)
        )
    if status:
        tasks = tasks.filter(status=status)
    if priority:
        tasks = tasks.filter(priority=priority)

    context = {
        'tasks': tasks,
        'current_status': status,
        'current_priority': priority,
        'current_search': search,
        'can_manage': _is_manager(request.user),
    }
    return render(request, 'tasks/list.html', context)
```

Figura 12 - Banck-End Lista de Tarefas

Aqui, o código realiza uma consulta (QuerySet) ao banco de dados na entidade Tarefa. A visualização (ListView ou função de View) implementa lógicas de filtragem para garantir que, por exemplo, um Colaborador Técnico receba e veja apenas as tarefas atribuídas a ele, garantindo a privacidade e organização dos dados.

```
<form method="GET" action="{% url 'tasks:list' %}" class="grid grid-cols-1 md:grid-cols-5 gap-4">
    <input type="text" name="search" placeholder="Buscar por título, descrição..." value="{{
current_search }}">
    <select name="status">...</select>
    <select name="priority">...</select>
    <button type="submit">Filtrar</button>
</form>

<table class="w-full">
    <thead>
        <tr>
            <th>Título</th>
            <th>Status</th>
            <th>Prioridade</th>
            <th>Responsável</th>
            <th>Vencimento</th>
            <th>Ações</th>
        </tr>
    </thead>
    <tbody>
        {% for task in tasks %}
        {% empty %}
            <tr>
                <td colspan="6" class="text-center">Nenhuma tarefa encontrada com os filtros
selecionados.</td>
            </tr>
        {% endfor %}
    </tbody>
</table>
```

Figura 13 - Front-End Tela de Tarefas

O template HTML faz uso das tags de repetição do Django *({% for tarefa in tarefas %})* para gerar linhas ou cartões na tela dinamicamente. As classes CSS estão condicionadas para exibir cores diferentes dependendo da prioridade (ex: vermelho para Urgente, verde para Normal).

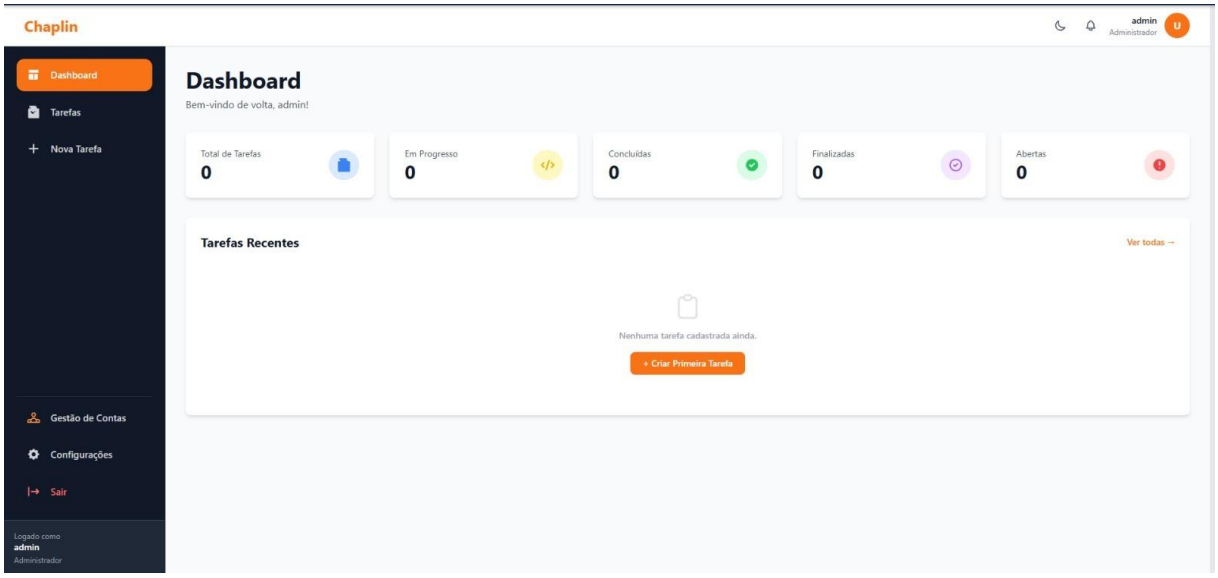

Figura 14 - Dashboard Administrador

Um painel gerencial rico, contendo indicadores de desempenho (KPIs), números totais de tarefas, andamento da equipe e status geral do prédio em tempo real.

```python
@login_required
def dashboard_view(request):
    user_profile = request.user.profile
    if user_profile.role == 'gestor':
        tasks = Task.objects.filter(created_by=request.user)
    elif user_profile.role == 'colaborador':
        tasks = Task.objects.filter(assigned_to=request.user)
    else:
        tasks = Task.objects.all()

    context = {
        'tasks': tasks,
        'total_tasks': tasks.count(),
        'open_tasks': tasks.filter(status='aberta').count(),
        'assigned_tasks': tasks.filter(status='alocada').count(),
        'completed_tasks': tasks.filter(status='concluida').count(),
        'finalized_tasks':
task$.filter(status='finalizada').count(),
    return render(request, 'tasks/dashboard.html', context)
```

Figura 15 - Back-End Dashboard Administrador

Este código demonstra o uso das funções de agregação do ORM do Django (como Count para total de tarefas e contagem por status). Há também a proteção na View, utilizando decoradores (como @login_required e validação de grupo), garantindo que os cálculos estatísticos só sejam executados e acessados se o ator for um Administrador logado.

```
<p class="text-gray-600 text-sm">Total de Tarefas</p>
<p class="text-3xl font-bold">{{ total_tasks }}</p>

<p class="text-gray-600 text-sm">Em Progresso</p>
<p class="text-3xl font-bold">{{ assigned_tasks }}</p>

<div class="text-center py-10 text-gray-400">
    <p class="text-sm font-medium">Nenhuma tarefa cadastrada ainda.</p>
    <a href="{% url 'tasks:create' %}" class="inline-block mt-4 bg-orange-500 text-white py-2 px-5
rounded-lg">+ Criar Primeira Tarefa</a>
</div>
```

Figura 16 - Front-End Dashboard Administrador

Mostra como os dados agregados enviados pelo back-end são distribuídos na marcação HTML em "Cards" (cartões de estatística) e gráficos, permitindo ao gestor tomar decisões instantâneas baseadas nos números injetados dinamicamente no visual da página.

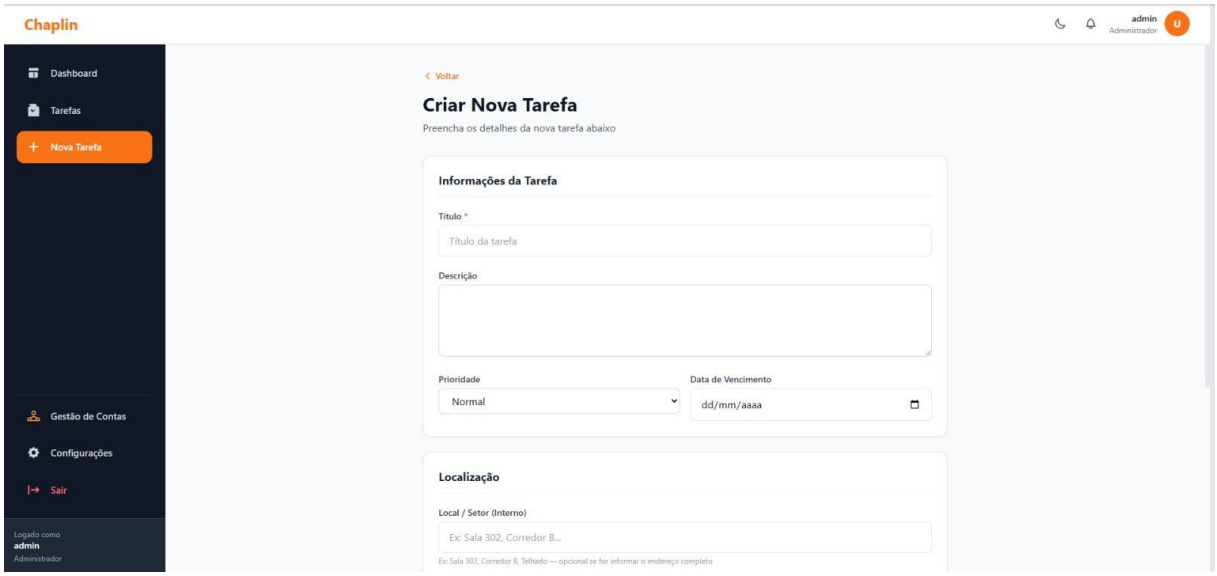

Figura 17 - Criação de novas Tarefas PT1

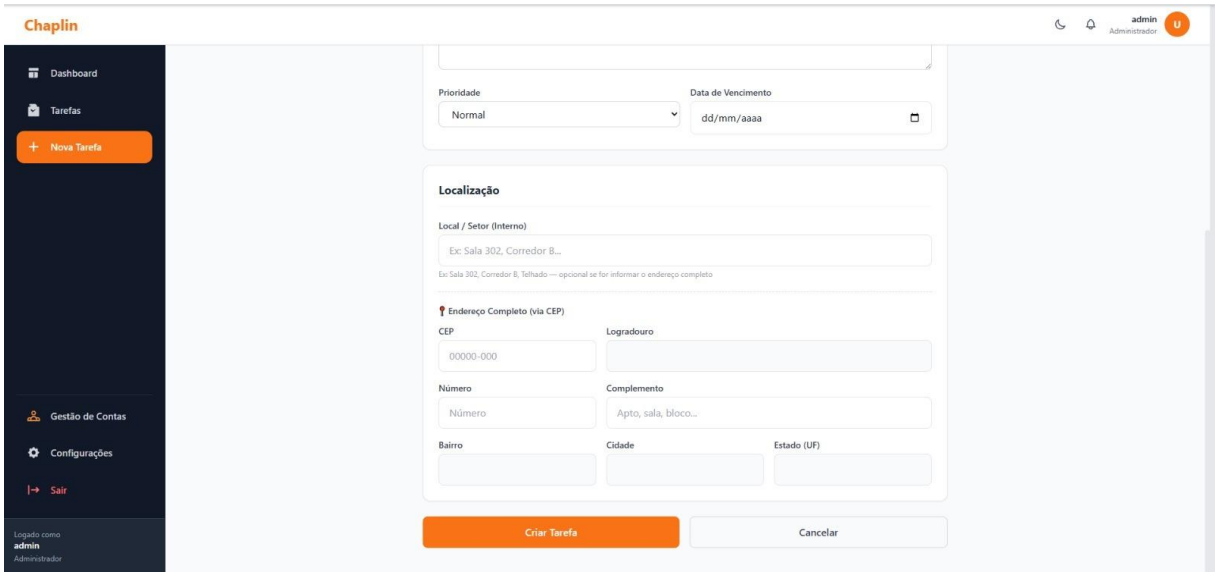

Figura 18 -Criação de novas Tarefas PT2

A tela divide em partes (PT1 e PT2) o formulário abrangente de abertura de um novo chamado (prioridade, especialidade exigida, fotos de evidência inicial e inserção inteligente do endereço do chamado).

```python
@login_required
def create_task_view(request):
    if not _is_manager(request.user):
        django_messages.error(request, 'Você não tem permissão para criar tarefas.')
        return redirect('tasks:list')
    if request.method == 'POST':
        form = TaskForm(request.POST, request.FILES)
        if form.is_valid():
            task = form.save(commit=False)
            task.created_by = request.user
            task.save()
            for user in AuthUser.objects.filter(profile__role__in=['gestor',
'admin']).exclude(pk=request.user.pk):
                _notify(user, f'Nova tarefa criada: {task.title}', tipo='tarefa_criada', task=task)
            return redirect('tasks:detail', pk=task.pk)
    else:
        form = TaskForm()

    return render(request, 'tasks/create.html', {'form': form})
```

Figura 19 - Back-End Criação de nova Tarefa

O código mostra como o Django recebe e valida múltiplos dados simultaneamente, garantindo a integridade relacional. Ele processa as imagens (Upload para evidências) e conecta a entidade Tarefa recém-criada ao Gestor do Prédio autenticado e à Especialidade correspondente.

```
<form method="POST" action="{% url 'tasks:create' %}" class="space-y-6">
    {% csrf_token %}
    <div class="bg-white rounded-xl p-6">
        <h2>Informações da Tarefa</h2>
        <div>
            <label>Título *</label>
            {{ form.title }}
        </div>
        <div>
            <label>Descrição</label>
            {{ form.description }}
        </div>
        <div class="grid grid-cols-2 gap-4">
            <div>
                <label>Prioridade</label>
                {{ form.priority }}
            </div>
            <div>
                <label>Data de Vencimento</label>
                {{ form.due_date }}
            </div>
        </div>
    </div>

    <div class="bg-white rounded-xl p-6">
        <h2>Localização</h2>
        <div>
            <label>Local / Setor (Interno)</label>
            {{ form.location }}
        </div>
        {{ form.cep }}
        {{ form.logradouro }}
        {{ form.numero }}
        {{ form.bairro }}
        {{ form.cidade }}
        {{ form.estado }}
    </div>
</form>
```

Figura 20 - Front-End Criação de nova Tarefa

No código da página, além do HTML do formulário configurado com enctype="multipart/form-data" (para permitir envio das fotos de evidência), observa-se a integração com o ViaCEP. Um trecho em JavaScript atua silenciosamente: ao usuário digitar o CEP, uma requisição assíncrona busca o logradouro e preenche o

formulário da tela automaticamente, refletindo em mais agilidade e usabilidade visual.

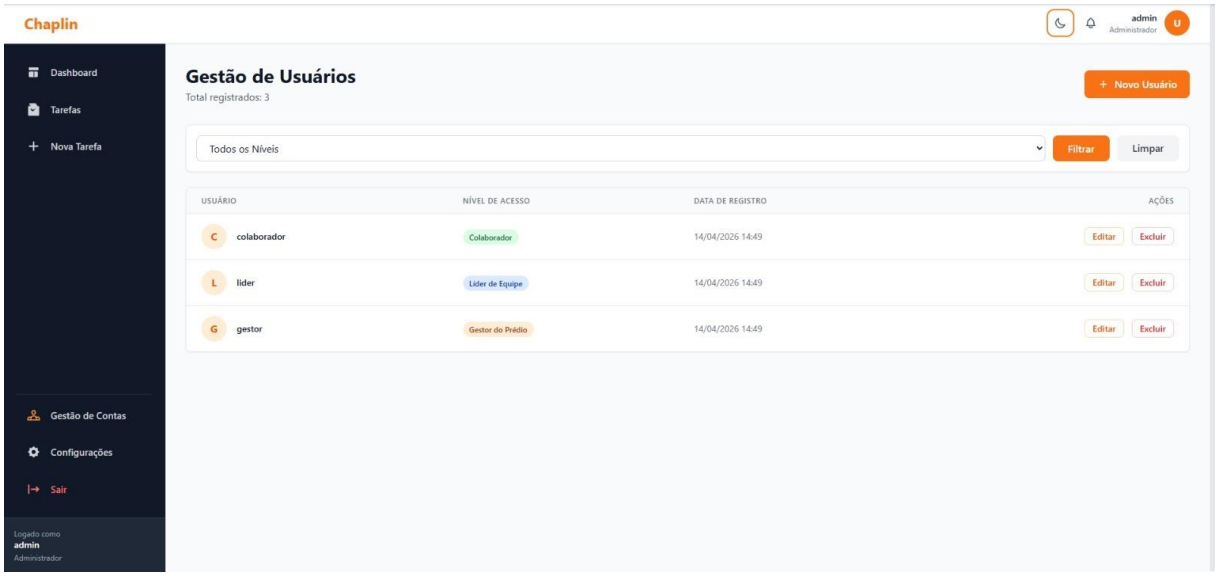

Figura 21 - Gestão de Usuários (Administrador)

Uma tabela organizacional demonstrando todos os colaboradores ativos cadastrados no sistema (Administradores, Gestores, Líderes e Técnicos), com botões que permitem edições e bloqueios.

```python
@login_required
@user_passes_test(is_admin)
def admin_users_list_view(request):
    query = request.GET.get('q', '')
    role_filter = request.GET.get('role', '')

    users_list = User.objects.filter(is_superuser=False).order_by('-
date_joined')
    if query:
        users_list = users_list.filter(
            Q(username__icontains=query) |
            Q(email__icontains=query) |
            Q(first_name__icontains=query)
        )
    if role_filter:
        users_list = users_list.filter(profile__role=role_filter)

    paginator = Paginator(users_list, 10)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    context = {
        'page_obj': page_obj,
        'query': query,
        'role_filter': role_filter,
        'roles': UserProfile.ROLE_CHOICES
    }
    return render(request, 'users/admin/list.html', context)
```

Figura 22 - Back-End Gestão de Usuários (Administrador)

O código destaca uma requisição ao banco (via *User.objects.all()*) para trazer a listagem. A regra de negócio garante que apenas ações vindas de administradores possam habilitar, desabilitar ou modificar papéis e especialidades na entidade Usuário.

```
<div>
    <h1 class="text-3xl font-bold text-gray-900 dark:text-white">Gestão de Usuários</h1>
    <p class="text-gray-600 dark:text-gray-400 mt-1">Total registrados: {{ page_obj.paginator.count }}
</p>
</div>

<form method="GET" class="flex gap-3">
    <select name="role">
        <option value="">Todos os Níveis</option>
        {% for value, label in roles %}
            <option value="{{ value }}" {% if role_filter == value %}selected{% endif %}>{{ label }}
</option>
        {% endfor %}
    </select>
    <button type="submit">Filtrar</button>
</form>

{% for user_obj in page_obj %}
<tr>
    <td>
        <div class="flex-shrink-0 h-10 w-10 bg-orange-100 text-orange-600 font-bold rounded-full flex
items-center justify-center">
            {{ user_obj.username|make_list|first|upper }}
        </div>
        <div>
            <div>{{ user_obj.get_full_name|default:user_obj.username }}</div>
            <div>{{ user_obj.email }}</div>
        </div>
    </td>
    <td>
        <span class="px-2.5 py-0.5 inline-flex text-xs font-semibold rounded-full
        {% if user_obj.profile.role == 'admin' %}bg-red-100 text-red-800
        {% elif user_obj.profile.role == 'gestor' %}bg-orange-100 text-orange-800
        {% elif user_obj.profile.role == 'lider' %}bg-blue-100 text-blue-800
        {% else %}bg-green-100 text-green-800{% endif %}">
            {{ user_obj.profile.get_role_display|default:'Colaborador' }}
        </span>
    </td>
    <td>{{ user_obj.date_joined|date:"d/m/Y H:i" }}</td>
    <td>
        <a href="{% url 'users:admin_user_edit' user_obj.id %}">Editar</a>
        {% if user_obj != request.user %}
        <a href="{% url 'users:admin_user_delete' user_obj.id %}">Excluir</a>
        {% endif %}
    </td>
</tr>
{% endfor %}
```

Figura 23 - Front-End Gestão de Usuários (Administrador)

A listagem em HTML recebe a lista do back-end. Botões de ação como "Editar" ou "Excluir" são gerados com caminhos (urls) apontando especificamente para o ID de cada usuário renderizado, combinando alertas de confirmação para prevenir exclusões acidentais na interface.

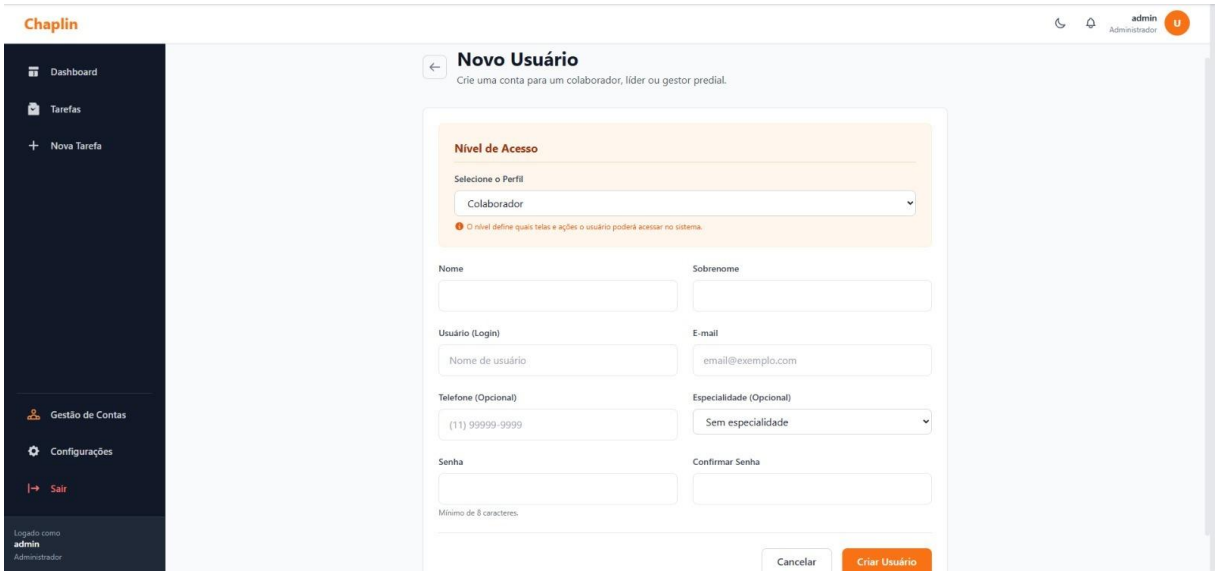

Figura 24 - Criação de novo Usuário

O formulário detalhado que cadastra novos integrantes na plataforma Chaplin. Nele se preenchem dados vitais (Nome, CPF, endereço, cargo) e senha.

```python
class AdminUserCreateForm(forms.ModelForm):
    INPUT_CSS = 'w-full px-4 py-2 bg-white dark:bg-gray-800 border rounded-lg ...'

    username = forms.CharField(max_length=30, label='Usuário', widget=forms.TextInput(attrs={'class':
INPUT_CSS, 'placeholder': 'Nome de usuário'}))
    email = forms.EmailField(max_length=254, label='E-mail', widget=forms.EmailInput(attrs={'class':
INPUT_CSS, 'placeholder': 'email@exemplo.com'}))
    first_name = forms.CharField(max_length=150, label='Nome', widget=forms.TextInput(attrs={'class':
INPUT_CSS}))
    last_name = forms.CharField(max_length=150, label='Sobrenome', required=False,
widget=forms.TextInput(attrs={'class': INPUT_CSS}))
    password = forms.CharField(min_length=8, label='Senha', widget=forms.PasswordInput(attrs={'class':
INPUT_CSS}))
    password_confirm = forms.CharField(min_length=8, label='Confirmar Senha',
widget=forms.PasswordInput(attrs={'class': INPUT_CSS}))

    ROLE_CHOICES = [
        ('colaborador', 'Colaborador'),
        ('lider', 'Líder de Equipe Técnica'),
        ('gestor', 'Gestor Predial (Anfitrião)'),
    ]
    role = forms.ChoiceField(choices=ROLE_CHOICES, label='Nível de Acesso', widget=forms.Select(attrs=
{'class': INPUT_CSS}))
    phone = forms.CharField(max_length=20, required=False, label='Telefone',
widget=forms.TextInput(attrs={'class': INPUT_CSS + ' mask-phone'}))

    class Meta:
        model = User
        fields = ['username', 'email', 'first_name', 'last_name', 'password']
```

Figura 25 - Back-End Criação de novo Usuário (PT1)

```python
@login_required
@user_passes_test(is_admin)
def admin_user_create_view(request):
    especialidades = Especialidade.objects.all()

    if request.method == 'POST':
        form = AdminUserCreateForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])
            user.is_active = True
            user.save()

            profile = user.profile
            profile.role = form.cleaned_data.get('role', 'colaborador')
            profile.phone = form.cleaned_data.get('phone', '')

            esp_id = request.POST.get('especialidade')
            if esp_id:
                profile.especialidade_id = esp_id
            profile.save()

            ActivityLog.objects.create(
                admin_user=request.user,
                target_user=user,
                action='CREATE_USER',
                role_new=profile.role,
                ip_address=request.META.get('REMOTE_ADDR')
            )

            messages.success(request, f'Usuário "{user.username}" criado com sucesso.')
            return redirect('users:admin_users_list')
    else:
        form = AdminUserCreateForm()

    return render(request, 'users/admin/create.html', {
        'form': form,
        'especialidades': especialidades,
    })
```

Figura 26 - Back-End Criação de novo Usuário (PT2)

Sendo um código mais extenso, dividido nas figuras, mostra a complexidade da criação do usuário. A Primeira Parte geralmente lida com a interceptação dos dados enviados e a validação fundamental do formulário (is_valid()). A Segunda Parte engloba o processo crítico de criptografia da senha (password hashing no Django), salvando as informações complementares de perfil (como telefones e endereço) e associando a especialidade técnica correta na estrutura do banco.

```
<form method="POST">
    {% csrf_token %}
    <div>
        <label>Selecione o Perfil</label>
        {{ form.role }}
    </div>

    <div class="grid grid-cols-2 gap-6">
        <div>
            <label>Nome</label>
            {{ form.first_name }}
        </div>
        <div>
            <label>Sobrenome</label>
            {{ form.last_name }}
        </div>
        <div>
            <label>Usuário (Login)</label>
            {{ form.username }}
        </div>
        <div>
            <label>E-mail</label>
            {{ form.email }}
        </div>
        <div>
            <label>Telefone (Opcional)</label>
            {{ form.phone }}
        </div>
        <div>
            <label>Especialidade (Opcional)</label>
            <select id="especialidade" name="especialidade">
                <option value="">Sem especialidade</option>
                {% for esp in especialidades %}
                    <option value="{{ esp.id }}">{{ esp.nome }}
</option>        {% endfor %}
            </select>
        </div>
    </div>

    <div class="grid grid-cols-2 gap-6">
        <div>{{ form.password }}</div>
        <div>{{ form.password_confirm }}</div>
    </div>
</form>
```

Figura 27 - Front-End Criação de novo Usuário

O template lida com o repasse visual de mensagens do servidor (Django Messages Framework). Caso ocorra um erro de validação (como um CPF já cadastrado) ou um sucesso na criação, o front-end reage disparando um alerta visual formatado na tela para manter o administrador a par do resultado da transação, finalizando a integração perfeita entre front e back.

12. CONSIDERAÇÕES FINAIS

O projeto Chaplin foi desenvolvido com o objetivo de solucionar problemas recorrentes enfrentados por empresas terceirizadas de manutenção hoteleira e predial. Muitas dessas empresas ainda utilizam ferramentas informais de comunicação, como aplicativos de mensagens, o que pode gerar desorganização, perda de informações e dificuldades no acompanhamento das tarefas realizadas.

Diante desse cenário, o Chaplin surge como uma plataforma web centralizada, capaz de organizar e monitorar todas as etapas do processo de manutenção, desde a abertura do chamado até a conclusão do serviço. O sistema proporciona maior controle operacional, rastreabilidade das atividades e mais transparência na comunicação entre gestores, líderes de equipe e técnicos.

Um dos principais diferenciais da solução é sua simplicidade e acessibilidade. A interface foi desenvolvida para ser intuitiva e fácil de utilizar, permitindo que colaboradores com diferentes níveis de experiência tecnológica consigam operar o sistema de forma eficiente. Isso contribui diretamente para maior produtividade e melhor organização das equipes.

Além disso, o desenvolvimento do projeto permitiu a aplicação prática de conhecimentos relacionados ao desenvolvimento web, banco de dados, segurança da informação, arquitetura de software e integração com APIs. Dessa forma, o Chaplin não apenas resolve um problema real do mercado, mas também demonstra o potencial da tecnologia como ferramenta de otimização e melhoria dos processos empresariais.

Espera-se que, com a implementação do Chaplin, as empresas possam otimizar seus processos internos, reduzir falhas operacionais, melhorar o gerenciamento das equipes técnicas e oferecer um serviço mais eficiente e transparente aos seus clientes. Dessa forma, a plataforma não apenas atende a uma necessidade real do mercado, mas também apresenta potencial para futuras expansões e melhorias, consolidando-se como uma solução tecnológica relevante para o segmento de manutenção predial e hoteleira.

13. REFERÊNCIAS

GONÇALVES, D. F. S. *Plataforma Integradora de Serviços para o Setor do Alojamento*. 2018. Trabalho acadêmico — Instituto Politécnico de Viseu, Viseu, 2018.

MOUFARREGE, C. N. *A Qualidade de Serviços na Hotelaria: um estudo de caso*. 2009. Trabalho acadêmico — Universidade de Brasília, Brasília, 2009.

DJANGO SOFTWARE FOUNDATION. *Django Documentation*. Disponível em: https://docs.djangoproject.com. Acesso em: 28 nov. 2025.

TAILWIND LABS. *Tailwind CSS*. Disponível em: https://tailwindcss.com. Acesso em: 05 dez. 2025.

AMAZON WEB SERVICES. *Amazon S3*. Disponível em: https://aws.amazon.com. Acesso em: 31 jan. 2026.

PYTHON SOFTWARE FOUNDATION. *Python Documentation*. Disponível em: https://docs.python.org/3/. Acesso em: 05 fev. 2026.

CHARLIE. A empresa Charlie foi utilizada como referência conceitual para o desenvolvimento deste projeto, contribuindo com inspirações relacionadas à inovação tecnológica, organização de processos e soluções voltadas à eficiência operacional no ambiente corporativo. Sua atuação no mercado paulista auxiliou na compreensão da importância da aplicação de tecnologias modernas para otimização de serviços e desenvolvimento de sistemas.

W3SCHOOLS. *W3Schools Online Web Tutorials*. Disponível em: https://www.w3schools.com/. Acesso em: 05 jun. 2026.