Documentação Técnica do Projeto
Gestão de Obras

Gerado por Gemini

3 de novembro de 2025

1 Visão Geral do Projeto

Este documento descreve a arquitetura e implementação de um sistema Full-Stack de Gestão de Obras. O objetivo da aplicação é permitir o controlo de múltiplos projetos de construção, incluindo gestão de funcionários, finanças, inventário e documentação.

O sistema é dividido em duas componentes principais:

- **Backend:** Uma API RESTful construída em Python com o framework Flask.
- **Frontend:** Uma Single Page Application (SPA) construída em React.js.

O sistema implementa um controlo de acesso robusto baseado em três níveis de permissão (Roles): **Administrador**, **Gestor** e **Prestador**.

2 Stack Tecnológica

2.1 Backend (Python)

- **Framework:** Flask
- **Base de Dados (ORM):** SQLAlchemy
- **Migrações de BD:** Flask-Migrate
- **Autenticação:** Flask-JWT-Extended (JSON Web Tokens)
- **Hashing de Senhas:** Flask-Bcrypt
- **CORS:** Flask-CORS
- **Servidor WSGI (Dev):** Werkzeug (nativo do Flask)

2.2 Frontend (JavaScript)

- **Biblioteca:** React.js (com Hooks)
- **Build Tool:** Vite
- **Comunicação API:** Axios
- **Estilização:** Tailwind CSS
- **Ícones:** Lucide-React

2.3 Base de Dados

- **Desenvolvimento:** SQLite (ficheiro `instance/dev.db`)
- **Produção (Planeada):** PostgreSQL (configurável via `config.py`)

3 Estrutura do Projeto

O projeto está organizado numa estrutura monorepo, com pastas distintas para o backend e o frontend.

```
/gestao-obras
|-- /backend
|   |-- /instance
|   |   |-- dev.db        (Base de dados SQLite local)
|   |-- /routes           (Blueprints do Flask)
|   |   |-- auth.py       (Login)
|   |   |-- obras.py      (Gestão de obras)
|   |   |-- users.py      (Gestão de utilizadores)
|   |   |-- ... (financeiro.py, inventario.py, etc.)
|   |-- __init__.py       (Fábrica da aplicação - create_app)
|   |-- config.py         (Configuração de ambiente e chaves)
|   |-- extensions.py     (Instâncias das extensões, ex: db, jwt)
|   |-- models.py         (Definição de todos os modelos SQLAlchemy)
|   |-- seed.py           (Script para popular a BD com dados iniciais)
|-- /frontend             (Aplicação React)
|   |-- /src
|   |   |-- App.jsx       (Ficheiro ÚNICO com toda a lógica do frontend)
|   |-- package.json
|   |-- vite.config.js
|-- .env                  (Ficheiro de variáveis de ambiente, ex: SECRET_KEY)
|-- run.py                (Ponto de entrada para iniciar o servidor backend)
```

4 Configuração e Execução (Setup)

4.1 Backend

1. **Ambiente Virtual:** Criar e ativar um ambiente virtual Python (`venv`).
2. **Dependências:** Instalar as dependências (ex: `pip install -r requirements.txt`).
3. **Variáveis de Ambiente:** Criar um ficheiro `.env` na raiz do projeto e definir, no mínimo, a `SECRET_KEY`.
4. **Popular a Base de Dados:** Executar o script de seed para criar a BD, os cargos (Roles) e os utilizadores de teste.

   ```
   python -m backend.seed
   ```

5. **Executar o Servidor:** Iniciar o servidor Flask (localmente na porta 5000).

   ```
   python run.py
   ```

4.2 Frontend

1. **Dependências:** Instalar as dependências do Node.js.

   ```
   npm install
   ```

2. **Executar o Servidor (Vite):** Iniciar o servidor de desenvolvimento (localmente na porta 5173).

   ```
   npm run dev
   ```

5 Detalhes do Backend

5.1 Autenticação e Segurança (JWT)

A autenticação é gerida por JSON Web Tokens (JWT) através da extensão `Flask-JWT-Extended`.

- **Login:** A rota `POST /api/auth/login` (`auth.py`) valida o `username` e a `password`. Se forem válidos, gera um `access_token` usando `create_access_token(identity=user.id)`.
- **Proteção de Rotas:** Endpoints sensíveis (ex: `obras.py`) são protegidos com o decorador `@jwt_required()`. Este decorador verifica o `Authorization: Bearer <token>` no cabeçalho (Header) da requisição.
- **Configuração da Chave Secreta (em `__init__.py`):** Para que a criação e a verificação do token funcionem corretamente, as chaves `SECRET_KEY` (do Flask) e `JWT_SECRET_KEY` (do Flask-JWT-Extended) devem ser idênticas. Isto é garantido no ficheiro `backend/__init__.py`:

```
# Em backend/__init__.py, dentro de create_app():

# 1. Obtém a chave principal (do config.py ou .env)
main_secret_key = app.config.get("SECRET_KEY")

# 2. Define AMBAS as chaves para serem idênticas
app.config["SECRET_KEY"] = main_secret_key
app.config["JWT_SECRET_KEY"] = main_secret_key
```

5.2 Sistema de Permissões (Roles)

O acesso às funcionalidades é controlado por três cargos definidos no modelo `Role` e atribuídos a cada `User`.

- **Administrador:** Acesso total. O único que pode aceder à página global de "Funcionários" para criar, editar ou remover utilizadores do sistema.
- **Gestor:** Acesso de gestão de obras. Pode criar novas obras, editar obras existentes, adicionar/remover funcionários de uma obra (vínculos) e gerir finanças, inventário, etc. Não pode gerir utilizadores globais.
- **Prestador:** Acesso limitado de "leitura". Só pode ver o Dashboard (com as obras a que está vinculado) e o módulo de Checklist (para ver e marcar as suas tarefas). Não pode criar obras, adicionar funcionários, gerir finanças ou fazer downloads de documentos.

5.3 Modelos de Dados (models.py)

Os principais modelos da base de dados são:

- `User`: Armazena dados de login (`username`, `password_hash`) e perfil (`nome`, `email`, etc.). Contém a `role_id` que define a permissão.
- `Role`: Tabela simples que armazena os nomes dos cargos (ID 1: Administrador, 2: Gestor, 3: Prestador).
- `Obras`: O modelo central que define um projeto/obra.
- `ObraFuncionarios`: Tabela-pivot que liga um `User` a uma `Obras`. Crucialmente, também permite adicionar funcionários "não cadastrados" (guardando `nome_nao_cadastrado` e `cpf_nao_cadast` se o utilizador não tiver uma conta no sistema.
- `AuditLog`: Regista alterações importantes (criação, atualização, remoção) em modelos críticos, como `ObraFuncionarios`.
- *Outros:* `FinanceiroTransacoes`, `InventarioItens`, `ChecklistItem`, `ChecklistAnexo`, `Documentos`.

6 Detalhes do Frontend (App.jsx)

Toda a aplicação frontend está contida num único ficheiro, `App.jsx`, para simplicidade de desenvolvimento neste ambiente.

6.1 Estrutura de Componentes

O ficheiro contém múltiplos componentes funcionais, incluindo:

- `App()`: O componente raiz. Gere o estado de autenticação.
- `LoginPage`: Ecrã de login.
- `DashboardPage`: Página principal que lista as obras.
- `ObraDetailPage`: Página que mostra os detalhes de uma obra (com abas).
- `Sidebar` e `Header`: Componentes de navegação.
- `*Tab` (ex: `FuncionariosTab`, `FinanceiroTab`): Componentes para cada aba dentro de `ObraDetailPage`.
- `*Page` (ex: `GlobalFuncionariosPage`): Páginas de gestão global (Inventário, Finanças, etc.).
- `*Modal` (ex: `NovaObraModal`, `AdicionarFuncionarioModal`): Pop-ups para criação e edição de dados.

6.2 Gestão de Estado e Roteamento

- **Estado de Autenticação:** Gerido no componente `App()` com `useState`. O `user` e o `access_token` são guardados no `localStorage` para persistir a sessão.
- **Roteamento:** Não é usado `react-router-dom`. Em vez disso, um "router" simples baseado em estado é implementado no `App()`. As variáveis de estado `view` e `selectedObraId` determinam qual página/componente é renderizado.

6.3 Comunicação com API (Axios Interceptor)

O Axios é configurado com uma instância global `api` que aponta para `http://127.0.0.1:5000/api`.

Um **Axios Interceptor** é configurado para automatizar a autenticação. Este interceptor "intercepta" *todas* as requisições antes de serem enviadas e anexa automaticamente o token de acesso (lido do `localStorage`) ao cabeçalho de autorização.

```
// Em App.jsx
api.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Esta é a lógica que permite ao frontend aceder às rotas protegidas `@jwt_required()` do backend.

6.4 Renderização Condicional (Permissões no Frontend)

As regras de permissão (Roles) são aplicadas no frontend lendo o objeto `user` (guardado no estado). O `user.role` é verificado para esconder ou desativar botões e links.

- **Sidebar:** O componente `Sidebar` filtra a lista de links visíveis com base no `user.role` (ex: Prestador só vê Dashboard e Checklist).
- **Botões:** Botões de "Criar Obra", "Adicionar Funcionário" ou "Remover" são condicionalmente renderizados usando variáveis como `canManage` (que verifica se o `user.role` não é 'Prestador').
- **Funcionalidade:** A lógica é duplicada (ex: `handleDownloadClick` no `DocumentosTab`) para impedir ações, mesmo que o botão esteja visível.