# Resumo Estratégico do Projeto FACILITA CHVC

## Ideia Central do Projeto

O projeto FACILITA CHVC nasceu da necessidade de criar um ponto de acesso centralizado, intuitivo e visualmente agradável para os diversos sistemas, ferramentas e recursos utilizados no contexto do Centro Hospitalar Virtual Completo (CHVC). A ideia principal é simplificar a experiência do usuário final, que muitas vezes precisa navegar por múltiplas plataformas ou lembrar de diversos endereços web. Ao consolidar esses acessos em um único portal, o FACILITA CHVC busca otimizar o tempo, reduzir a complexidade e melhorar a eficiência no acesso às informações e ferramentas essenciais para as atividades diárias.

O portal não se limita a ser um simples agregador de links. Ele foi concebido para oferecer uma experiência de usuário moderna e fluida, utilizando conceitos visuais como glassmorphism e gradientes suaves (conforme a versão preferida pelo usuário) para criar um ambiente digital sofisticado e convidativo. A organização dos links por categorias, a funcionalidade de busca e a paginação são elementos chave que visam facilitar a localização rápida do recurso desejado, mesmo com um grande volume de informações.

Além da interface pública, o projeto contempla uma área administrativa robusta e segura, acessível apenas por usuários autorizados. Este painel de controle permite o gerenciamento completo dos links (criação, edição, exclusão), das categorias (criação, edição, exclusão com validação de uso) e da personalização visual dos cards (cores, imagens de fundo, ícones), garantindo que o portal possa ser mantido atualizado e adaptado às necessidades da organização de forma autônoma e eficiente.

Em suma, o FACILITA CHVC é uma solução estratégica que combina funcionalidade prática com um design cuidadosamente elaborado, visando ser a porta de entrada principal e preferencial para os recursos digitais do CHVC, promovendo organização, agilidade e uma experiência de usuário superior.

---

# Arquitetura e Funcionamento Básico

O projeto FACILITA CHVC adota uma arquitetura moderna de aplicação web, separando claramente as responsabilidades entre o frontend (interface do usuário) e o backend (lógica de negócio e dados).

## Arquitetura Geral

1. **Frontend:** Desenvolvido com **React** (utilizando **Vite** como ferramenta de build e **TypeScript** para tipagem estática), responsável por toda a apresentação visual e interação com o usuário. Utiliza **Tailwind CSS** para estilização e **Framer Motion** para animações, garantindo uma interface responsiva, moderna e fluida. A navegação entre páginas é gerenciada pelo **React Router**.
2. **Backend:** Construído em **Python** com o microframework **Flask**, serve como uma API RESTful para o frontend. É responsável pela lógica de negócio, autenticação de usuários administrativos e persistência de dados. Utiliza **SQLAlchemy** como ORM para interagir com um banco de dados **SQLite**, tornando o projeto autocontido e fácil de executar localmente. A segurança da área administrativa é garantida por meio de **JSON Web Tokens (JWT)**.
3. **Comunicação:** O frontend se comunica com o backend exclusivamente através de requisições HTTP para a API RESTful. O **Axios** é utilizado no frontend para facilitar essas chamadas. O backend responde com dados em formato JSON.
4. **Banco de Dados:** Um arquivo **SQLite** ( `facilita.sqlite` , criado automaticamente na pasta `instance` do backend) armazena todas as informações: usuários administrativos, links, categorias e configurações de personalização.

## Funcionamento Básico

1. **Portal Público (Frontend - `/`)**:
    - Ao acessar a página inicial, o frontend busca todos os links e categorias disponíveis na API do backend.
    - Os links são exibidos em formato de cards, organizados em um grid responsivo e ordenados alfabeticamente.
    - O usuário pode filtrar os links por categoria através de um menu suspenso.
    - Uma barra de busca permite filtrar links pelo título ou categoria.
    - A paginação é ativada automaticamente caso o número de links filtrados exceda o limite por página.
    - Os cards podem exibir imagens de fundo ou cores sólidas, conforme cadastrado.
    - Clicar em um card abre o URL correspondente em uma nova aba.

- O rodapé contém um link para a área administrativa.

2. **Login Administrativo (Frontend - `/admin/login`)**:
   - Apresenta um formulário para inserção de usuário e senha.
   - Ao submeter, o frontend envia as credenciais para a rota `/api/auth/login` do backend.
   - O backend valida as credenciais contra os dados no banco SQLite.
   - Se válidas, o backend gera um token JWT e o retorna ao frontend.
   - O frontend armazena o token (geralmente no `localStorage`) e redireciona o usuário para o painel administrativo (`/admin`).

3. **Painel Administrativo (Frontend - `/admin`)**:
   - Esta é uma rota protegida. O frontend verifica a presença e validade do token JWT antes de renderizar o conteúdo.
   - Todas as requisições para a API a partir daqui incluem o token JWT no cabeçalho `Authorization`.
   - O painel permite:
     - **Gerenciar Links:** Listar, criar, editar (título, URL, categoria, cor, imagem) e excluir links.
     - **Gerenciar Categorias:** Listar, criar, editar (nome, cor, ícone) e excluir categorias (com validação para não excluir categorias em uso).
     - **Gerenciar Cores:** Adicionar cores personalizadas (HEX/RGB) e editar/excluir cores existentes (funcionalidade adicionada em iterações posteriores).
     - **Logout:** Limpa o token JWT armazenado e redireciona para a página de login.

4. **Backend (API RESTful)**:
   - Recebe as requisições do frontend.
   - Para rotas protegidas (gerenciamento), valida o token JWT recebido.
   - Interage com o banco de dados SQLite (via SQLAlchemy) para buscar, criar, atualizar ou deletar dados (usuários, links, categorias).
   - Retorna as respostas em formato JSON para o frontend.
   - Inclui uma rota `/api/ping` para verificação de conectividade.

---

# Prompt Detalhado para Recriação do Projeto FACILITA CHVC

**Objetivo:** Recriar o projeto FACILITA CHVC do zero, um portal web full-stack moderno e funcional para acesso centralizado a recursos, com um painel administrativo completo.

**Arquitetura e Tecnologias:**

1. **Tipo:** Aplicação Web Full-Stack.
2. **Estrutura:** Projeto monorepo com dois diretórios principais: `frontend` e `backend`.
3. **Backend:**
    - Linguagem: Python 3.x
    - Framework: Flask
    - Banco de Dados: SQLite (arquivo único, autocontido)
    - ORM: SQLAlchemy
    - Autenticação: Flask-JWT-Extended (JWT para rotas administrativas)
    - API: RESTful, com endpoints claros para cada recurso.
    - CORS: Configurado para permitir requisições do frontend (localhost:5173).
    - Dependências: Flask, Flask-SQLAlchemy, Flask-JWT-Extended, Flask-Cors, python-dotenv, Werkzeug (versões compatíveis).
4. **Frontend:**
    - Framework/Lib: React
    - Build Tool: Vite
    - Linguagem: TypeScript
    - Estilização: Tailwind CSS
    - Animações: Framer Motion
    - Roteamento: React Router DOM
    - Requisições HTTP: Axios
    - Notificações: React Hot Toast
    - Dependências: react, react-dom, react-router-dom, axios, tailwindcss, postcss, autoprefixer, @vitejs/plugin-react-swc, typescript, framer-motion, react-hot-toast, lucide-react (para ícones), etc.

**Funcionalidades Essenciais:**

1. **Portal Público (`/`)**
    - Exibir links cadastrados em formato de cards responsivos (grid).
    - Ordenar links alfabeticamente por título por padrão.
    - Permitir filtrar links por categoria (menu dropdown).
    - Implementar barra de busca para filtrar links por título ou categoria.

- Implementar paginação caso o número de links filtrados exceda um limite (e.g., 12 por página).
- Cards devem suportar imagem de fundo ( `imageUrl` ) ou cor sólida ( `color` ) como background. Se houver imagem, o título e categoria devem ser legíveis sobre ela (usar overlay/gradiente escuro na parte inferior).
- Clicar em um card abre o `url` correspondente em nova aba.
- Rodapé com informações básicas e link para `/admin/login` .
- **Garantir ausência total de scrollbars desnecessárias na página.**

2. **Login Administrativo ( `/admin/login` )**

- Formulário seguro para usuário e senha.
- Enviar credenciais para a API backend ( `/api/auth/login` ).
- Ao receber token JWT válido, armazená-lo (localStorage) e redirecionar para `/admin` .
- Tratar erros de login (credenciais inválidas, erro de servidor) com feedback visual (toast).
- **Corrigir problema de erro 415:** Garantir que o frontend envie `Content-Type: application/json` e que o backend Flask processe corretamente requisições JSON.
- **Corrigir problema de redirecionamento:** Garantir que todos os links e redirecionamentos usem caminhos relativos ou `localhost` , sem IPs hardcoded.

3. **Painel Administrativo ( `/admin` )**

- Rota protegida por JWT. Verificar token em cada acesso/requisição.
- Interface intuitiva para gerenciar:
  - **Links:** CRUD completo (Criar, Ler, Atualizar, Deletar). Campos: Título, URL, Categoria (seleção de existente), Cor (seleção de existente/padrão), URL da Imagem (opcional).
  - **Categorias:** CRUD completo. Campos: Nome, Cor (seleção), Ícone (seleção de biblioteca pré-definida).
    - **Validação:** Impedir exclusão de categoria se houver links associados a ela, retornando erro informativo.
  - **Cores:** CRUD completo. Permitir adicionar novas cores via input HEX ou RGB. Permitir editar/excluir cores existentes.
  - **Ícones:** Implementar uma biblioteca visual de ícones (usar `lucide-react` ou similar) para seleção ao criar/editar categorias e, opcionalmente, links.
- Botão de Logout (limpar token, redirecionar para login).
- Utilizar toasts para feedback de sucesso/erro em todas as operações CRUD.

4. **Backend API Endpoints:**
    - `/api/ping`: Rota GET pública para verificar status.
    - `/api/auth/login`: Rota POST pública para autenticação.
    - `/api/auth/verify`: Rota GET protegida para verificar validade do token.
    - `/api/links`: GET (listar todos), POST (criar - protegido).
    - `/api/links/<id>`: GET (obter um), PUT (atualizar - protegido), DELETE (excluir - protegido).
    - `/api/categories`: GET (listar todas), POST (criar - protegido).
    - `/api/categories/<id>`: GET (obter uma), PUT (atualizar - protegido), DELETE (excluir - protegido).
    - (Opcional/Avançado) `/api/colors`: Endpoints CRUD para cores personalizadas (protegido).

**Estilo Visual e Experiência do Usuário (UX):**

1. **Estilo Base:** Inspirado na **primeira versão otimizada** entregue anteriormente (referenciar arquivos da pasta `/home/ubuntu/optimized/` se necessário).
    - Utilizar conceitos de **Glassmorphism** nos cards (fundo semitransparente, blur, borda sutil).
    - **Fundo:** Cor sólida escura e moderna (e.g., azul/roxo escuro, como `rgb(20, 20, 30)`), **sem animação de gradiente no background.**
    - Esquema de cores moderno e coeso (usar variáveis CSS para fácil ajuste).
    - Tipografia legível e moderna (e.g., Inter, Poppins).
2. **Animações:** Utilizar **Framer Motion** para animações sutis e elegantes:
    - Fade-in e leve slide-up para elementos da página ao carregar.
    - Animação de hover nos cards (leve scale/translateY, sombra intensificada).
    - Animações de entrada/saída para listas (cards, itens no admin) ao adicionar/remover.
    - **NÃO** implementar animação no background da página.
3. **Responsividade:** Layout totalmente responsivo, adaptando-se a telas de desktop, tablet e mobile.
4. **Feedback:** Uso consistente de toasts (React Hot Toast) para feedback de ações e erros.
5. **Usabilidade:** Interface intuitiva, fácil navegação, formulários claros.
6. **Performance:** Código otimizado, carregamento rápido.

**Estrutura de Arquivos e Documentação:**

1. Manter separação clara entre `frontend` e `backend`.
2. Organizar componentes, páginas, serviços (frontend) e rotas, modelos (backend) de forma lógica.

3. Criar um `README.md` detalhado na raiz do projeto, explicando a arquitetura, tecnologias, como instalar dependências ( `npm install` , `pip install -r requirements.txt` ) e como executar ambos os servidores (backend e frontend).
4. Incluir credenciais padrão do admin ( `admin` / `admin123` ) no README.

**Entregável:**

- Código-fonte completo do projeto nos diretórios `frontend` e `backend` .
- Arquivo `README.md` .
- (Opcional) Arquivo ZIP do projeto (excluindo `node_modules` e `venv` ).