Documentação do Know Your Fan

Proposta: Rede social móvel focada em e-sports, oferecendo notícias de competições e marketplace de produtos oficiais da FURIA. Autenticação via contas de terceiros, registro de atividades e validação documental garantem credibilidade e personalização.

---

1. API Flask (CSV)

1.1 Visão Geral

- **Usuários: persistidos em `dados/dados.csv`.**
- **Atividades: registradas em `dados/atividades.csv`.**
- **Validação documental: CPF via upload.**
- **Autenticação: Google OAuth e Facebook Fake Data.**

1.2 Configuração

- **Porta: `5000`**
- **SECRET_KEY: variável de ambiente `SECRET_KEY` ou `'senha_secreta'`**
- **Diretório de dados: pasta `dados/` na raiz do projeto**

1.3 Endpoints Principais

GET `/api/usuario/<usuario_email>`

- **Descrição: busca usuário por e-mail.**
- **Respostas:**
    - `200 OK` ⇒ **JSON com dados do usuário**
    - `404 Not Found` ⇒ `{ "error": "Usuário não encontrado" }`

POST `/api/usuario`

- **Entrada: JSON com** `Nome`, `Email`
- **Ações: cria cabeçalho (se necessário) e adiciona registro em CSV**
- **Respostas:**
  - `201 Created` ⇒ **JSON do usuário**
  - `400 Bad Request` ⇒ `{ "error": "Dados incompletos" }`
  - `500 Internal Server Error` ⇒ `{ "error": "Erro no servidor" }`

POST `/api/atualizar_usuario`

- **Entrada: JSON com** `Email` **e campos a atualizar**
- **Ações: lê, atualiza e regrava CSV**
- **Respostas:**
  - `200 OK` ⇒ `{ "status": "sucesso", "updated": {...} }`
  - `400 Bad Request` ⇒ **erro de validação**
  - `404 Not Found` ⇒ `{ "error": "Usuário não encontrado" }`
  - `500 Internal Server Error` ⇒ `{ "error": "Falha ao gravar CSV" }`

POST `/api/activities`

- **Entrada: JSON com** `Email`, `link`
- **Ações: valida usuário e registra atividade com** `threading.Lock`
- **Respostas:**
  - `200 OK` ⇒ `{ "status": "Atividade registrada" }`

- `400/404/500` ⇒ erros correspondentes

POST `/api/validar_documento`

- **Entrada (multipart): arquivo** `documento`**, campo** `cpf`
- **Ações: chama** `validar_documento(documento, cpf)`
- **Respostas:**
  - `200 OK` ⇒ **resultado da validação**
  - `400 Bad Request` ⇒ **falta de campo ou arquivo**

---

2. Autenticação

2.1 Google OAuth (`src/auth/googlelogin.py`)

1. **init_google_oauth(app): configura credenciais e registra provedor Google (OpenID Connect).**
2. **login_google() (`/auth/login`): redireciona para autorização Google.**
3. **auth_callback() (`/auth/callback`): obtém tokens, decodifica perfil, armazena em sessão, registra usuário em CSV e gera cache JSON.**
4. **login_data() (`/auth/credentials`): lê e remove cache, retorna JSON com `Nome`, `Email`.**

2.2 Facebook (`src/auth/facebook`)

- **Função:** `fake_data(json_path, info_csv, activities_csv)`
- **Entradas: caminhos para JSON de perfil, CSV de usuários e CSV de atividades.**
- **Processo:**
  1. **Lê JSON com** `Nome`**,** `Email`**,** `CPF`**,** `Endereço` **e** `follow_facebook`**.**

2. Grava perfil em `info_csv` (cabeçalho opcional).
3. Registra atividades de follow em `atividades.csv`.

- Retorno: dicionário `info` com `Nome`, `Email`, `CPF`, `Endereço`.

---

3. Utilitários (`src/client/telas/utils.py`)

- show_popup(message): exibe `Popup` com título.
- cache_search(campo): retorna valor de `campo` em `cache/cache.json`.
- is_wsl(): identifica WSL via `/proc/version`.
- open_url(url): abre URL nativamente (Windows/macOS/WSL/Linux) com fallback para `webbrowser`.

Nota: confirme o caminho de `cache/cache.json` antes de usar.

---

4. Telas Kivy

4.1 TelaCadastro (`src/client/telas/tela_cadastro.py`)

- Componentes: Título, campos `Nome`, `Email`, `CPF`, `Endereço` e botão Cadastrar.
- Ação do botão Cadastrar: envia `POST /api/usuario`, exibe popup de sucesso/erro, grava cache e navega para `login`.

4.2 TelaDocumento (`src/client/telas/tela_documento.py`)

- Componentes: FileChooser, status label, botões Voltar e Submeter Documento.
- Ação do botão Voltar: retorna a `principal`.
- Ação do botão Submeter Documento: envia arquivo e CPF para `POST /api/validar_documento` e atualiza status.

4.3 TelaEditarPerfil (`src/client/telas/tela_editar_perfil.py`)

- **Componentes: campos `Nome`, `Link do Perfil`, `Email`, `CPF`, `Endereço`; botões Salvar e Cancelar.**
- **Ação do botão Salvar: grava cache, envia `POST /api/atualizar_usuario` em thread e vai para `perfil`.**
- **Ação do botão Cancelar: descarta alterações e vai para `perfil`.**

4.4 TelaPerfil (`src/client/telas/tela_perfil.py`)

- **Componentes: imagem de perfil, nome, blocos de informação (`Link`, `Email`, `CPF`, `Endereço`); botões Verificar, Editar, Voltar, Logout.**
- **Ações:**
  - **Verificar Perfil: vai para `documento`.**
  - **Editar Perfil: vai para `editar_perfil`.**
  - **Voltar: vai para `principal`.**
  - **Logout: limpa cache e vai para `login`.**

4.5 TelaProdutos (`src/client/telas/tela_produtos.py`)

- **Componentes: grid de produtos em `ScrollView`, botão Voltar.**
- **Ação do botão Voltar: retorna a `principal`.**
- **Fluxo: em `on_enter`, valida CPF/Endereço e exibe popup se necessário.**

4.6 TelaPrincipal (`src/client/telas/tela_principal.py`)

- **Componentes: painel lateral (foto, nome, botões Perfil, Loja, Atividades) e feed rolável (`FeedItem`).**
- **Ações dos botões: Perfil → `perfil`; Loja → `loja`; Atividades → (a implementar).**

- **Fluxo: `on_pre_enter` carrega nome e popula feed.**

4.7 TelaInicial (Login) (`src/client/telas/tela_inicial.py`)

- **Componentes: campos de usuário/senha e botões Entrar, Facebook, Google, Criar Conta.**
- **Ações:**
  - **Entrar: valida campos e faz `GET /api/usuario`; se `200`, grava cache e vai para `principal`.**
  - **Facebook: faz `GET /auth/facebook`, grava cache, importa fake data e vai para `perfil`.**
  - **Google: inicia OAuth (`/auth/login`), faz polling em `/auth/credentials`, grava cache e vai para `perfil`.**
  - **Criar Conta: vai para `cadastro`.**

4.8 FeedItem (`src/client/telas/feed_item.py`)

- **Componentes: thumbnail, título, descrição e botão Abrir.**
- **Ação do botão Abrir: abre link no navegador e envia `POST /api/activities` para registrar acesso.**
- **Fluxo: ao tocar em Abrir, dispara thread que executa `_handle_open`.**

---

5. Executando o código (Ubuntu)

- **Clone o repositório: https://github.com/LucasChagasMoreira/Desafio-tecnico-Furia.**
- **Dentro do diretorio “Desafio-Tecnico-Furia”, execute “make install”.**
- **Após isso, execute make “run-all” para iniciar tanto a api quanto o frontend da aplicação.**