

# Sistema de Estoque — Documentação do Projeto

Este documento descreve a arquitetura, organização de pastas, padrões de UI/UX e como executar o projeto.

## Sumário

- Visão geral
- Arquitetura e principais componentes
- Navegação e rotas
- Internacionalização (i18n)
- Estilos e temas
- Estrutura de pastas
- Convenções e dicas de layout (Flet)
- Como executar
- Roadmap (próximos passos)
- Performance e Boas Práticas (UI)
- Clients (grade, imagens e resize)

## Visão geral

Aplicativo de controle de estoque feito em Python com Flet. A navegação é baseada em um "shell" principal (Main View) com barra lateral (Sidebar). O conteúdo principal é carregado dinamicamente no centro sem troca de rota para a maioria das interações; rotas são usadas para estados principais como Login e a View principal.

## Arquitetura e principais componentes

- App (**lib/app.py**)
- Configura título e preferências de janela/tema via **AppConfig**.
- Inicializa navegação com **PageManager**.
- Core (**lib/src/core/**)
- **page\_manager.py**: responde a mudanças de rota (event.route), instancia e injeta Views.
- **routes.py**: centraliza constantes de rota (ex.: **/login**, **/main\_view**).
- Views (**lib/src/app/views/**)
- **widgets/side\_bar.py**: NavigationRail com labels traduzidas; emite seleção para atualizar conteúdo central.
- **pages/**: páginas de conteúdo como **welcome.py** e **products.py** (renderizadas dentro do shell principal).
- **login.py**: tela de autenticação.
- **home.py/main\_view.py** (shell): layout com Row (Sidebar + área de conteúdo). A Sidebar troca o conteúdo central via callback.
- i18n (**lib/utlis/**)
- **label\_keys.py**: enum com chaves (APP\_TITLE, MENU\_HOME, WELCOME\_TITLE etc.).
- **labels.py**: dicionários de traduções (pt/en) e helper **Labels.t**.

- Estilos (**lib/src/app/styles/**)
- **theme.py**: **ThemeManager** com cores e temas (dark/light).
- **image.py**: **ImagesAssets** e resolução de logos conforme tema.

## Navegação e rotas

- As rotas (Login/Main View) usam **PageManager** com **event.route** e **page.go(...)**.
- Dentro da Main View, a Sidebar não muda a rota; apenas notifica a seleção (ex.: **home**, **products**) para que o shell atualize o container central.
- Benefícios: histórico e deep-link para estados macro (login/main), com UX fluida no conteúdo interno.

## Internacionalização (i18n)

- Novas chaves para menu e WelcomePage: **MENU\_\***, **WELCOME\_TITLE**, **WELCOME\_INSTRUCTION**.
- **Labels.t(LabelKey.X)** retorna a string traduzida conforme **AppConfig.default\_lang**.

## Estilos e temas

- Uso de **ThemeManager** para cores, fundo, botões e fontes.
- Logos alternam conforme tema (claro/escuro).
- API atualizada do Flet: **ft.Colors** em vez de **ft.colors**.

## Estrutura de pastas (resumo)

config.ini  
main.py  
lib/  
app.py  
src/  
app/  
styles/  
views/  
pages/  
widgets/  
core/  
page\_manager.py  
routes.py  
config/  
app\_config.py  
utils/  
label\_keys.py  
labels.py  
assets/

images/  
storage/  
data/  
temp/

## Convenções e dicas de layout (Flet)

- Para ocupar todo o espaço disponível:
- Em layouts com **Row**, use **vertical\_alignment=ft.CrossAxisAlignment.STRETCH**.
- Em **Column**, use **expand=True** e **horizontal\_alignment=ft.CrossAxisAlignment.STRETCH**.
- Para conteúdo que deve crescer, use **ft.Expanded** ou **expand=True** e envolva tabelas em **ft.ListView/ft.Container** com **expand=True** para scroll.
- Sidebar
- Alterna **label\_type** (NONE/ALL) no hover; labels vêm do i18n.

## Performance e Boas Práticas (UI)

- Evite reconstruir a árvore de controles em handlers de UI (resize/hover/animação). Prefira alterar propriedades animáveis em controles já existentes (**scale**, **opacity**, **rotate**) e chamar **update()** neles.
- Em listas/grades com imagens:
- Evite re-encodar base64 a cada render. Calcule uma vez por card e reutilize.
- Prefira cache local em arquivo (**Image(src=...)**) quando possível, reduzindo payload enviado ao frontend.
- **ResponsiveRow** com **columns=12** e **col** em cada card permite responsividade sem recriar a grade no **on\_resized**.
- Async em UI (Flet):
- Não use **asyncio.run()** em callbacks/eventos. Use **page.run\_task(coro)** para agendar corrotinas no loop do Flet.
- Evite agendar a mesma task múltiplas vezes; guarde o handle (ex.: **self.\_load\_task**).

### ***Postmortem: uso elevado de memória e CPU na tela de Clientes***

Sintoma: ao executar animações ou redimensionar a janela na tela de Clientes, o consumo de memória chegava a ~2GB e CPU elevada. Na tela de Produtos não ocorria.

Causas:

- A cada resize/alteração de layout, a árvore de controles era (ou podia ser) reconstruída, recriando cartões e reprocessando imagens.
- Imagens eram serializadas como base64 e reenviadas ao frontend em cada **update()** quando os controles eram recriados.
- Em handlers de UI, operações pesadas de imagem (decoding/base64) poderiam ocorrer repetidamente.

Correções aplicadas:

- Responsividade com **ResponsiveRow** e **col** por card, sem reconstruir a grade no **on\_resized**.
- Cache de imagem por card:
- Preferência por salvar o BLOB/BASE64 em arquivo local (hash do conteúdo) e usar **Image(src=...)**.
- Fallback para **src\_base64** computado uma única vez por card.

- Reutilização do mesmo **CircleAvatar/Image** em animações (alterando apenas **scale**).
- Proteção contra múltiplos carregamentos: **get\_view()** guarda **self.\_load\_task** para não agendar a mesma coroutine várias vezes.

Prevenção futura:

- Não reconstruir **content** em **resize/hover**; usar controles responsivos e alterar apenas propriedades.
- Evitar base64 em hot-path de UI; preferir **Image(src=...)** com cache local e, se usar base64, calcular uma vez por instância.
- Centralizar agendamento assíncrono com **page.run\_task** e manter referências das tasks para evitar duplicidade.
- Opcional: Telemetria simples (tempo de render, contagem de updates) e limites nos tamanhos de imagem no backend (compressão/redimensionamento).

## Como executar

- Requisitos: Python 3.11+ e Flet.
- Opção 1 (CLI do Flet):
- Dentro do diretório do projeto, execute: **flet run**
- Opção 2 (Python):
- Ative a venv e rode **python main.py**.

## Roadmap (próximos passos)

- Mover páginas para **views/pages** e criar controladores por domínio (ex.: Produtos).
- Paginação/ordenção/CRUD na tabela de produtos.
- Testes de UI e linter/formatador (ruff/black) no repositório.

## Clients (grade, imagens e resize)

### *Grade Nx3 responsiva*

- Implementada com **ft.ResponsiveRow(columns=12)** e cartões recebendo **col={"xs": 12, "md": 6, "lg": 4}**.
- Em telas largas ( $\geq$  lg), exibem 3 colunas; conforme a largura reduz, a grade reflowa para 2 ou 1 coluna sem reconstrução.

### *Cartões e imagens (avatar)*

- Cada **ContactsCard** mantém cache:
- **self.\_photo\_src**: caminho de arquivo (cache) quando a foto vem como BLOB/base64.
- **self.\_photo\_b64**: base64 calculado uma única vez (fallback quando não é possível salvar em arquivo).
- **self.\_avatar**: controle **CircleAvatar** reutilizado (evita reenvio de imagem a cada **update**).
- A foto é normalizada assim:
  1. Se **bytes**/BLOB: salva em arquivo (hash para nome estável) e usa **Image(src=...)**.
  2. Se **str** em **data**: (data URL): decodifica e salva.
  3. Se **str** base64: decodifica e salva; se falhar, usa **src\_base64** (fallback) uma vez por card.

- Fallback para iniciais quando não há foto.

### ***Resize***

- O **on\_resized** não troca **content**. Apenas chama **update()** no container principal.
- O **ResponsiveRow** reflowa os mesmos cards, evitando recarga de imagens.

### ***Animações***

- As animações alteram apenas propriedades do controle existente (**animate\_scale + scale**), sem recriar o avatar/imagem.

### ***Backend / Banco***

- Campo **photo** como BLOB (MEDIUMBLOB recomendado). Envie bytes ao MySQL com parâmetros (sem f-strings).
- Se necessário, comprima/redimensione imagens no backend para manter tamanhos razoáveis.