1. DESCRIÇÃO DO PROJETO

Este projeto é uma aplicação web desenvolvida em Flask para o gerenciamento completo de um sistema de controle de estoque. Ele permite o cadastro de produtos, fornecedores, controle de usuários com diferentes níveis de permissão, e o registro de movimentações de estoque (entradas e saídas), além de uma tela de vendas para registrar as operações comerciais.

A aplicação foi construída utilizando uma arquitetura modular com *Blueprints*, o padrão *Application Factory* e uma interface de usuário responsiva baseada em Bootstrap.

2. FUNCIONALIDADES

2.1. **Autenticação de Usuários**

Sistema completo de login, registro e logout de usuários.

2.2. **Controle de Acesso Baseado em Funções (RBAC)**

- **Admin:** Acesso total ao sistema, incluindo gerenciamento de usuários.
- **Gerente:** Pode gerenciar produtos, fornecedores e realizar vendas.
- **Usuário:** Nível de acesso mais restrito (pode ser personalizado).

2.3. **Dashboard Principal**

Exibe estatísticas rápidas sobre o estoque, como total de produtos, valor total em estoque e alertas de produtos com baixa quantidade.

2.4. **Gerenciamento de Produtos (CRUD)**

- Cadastro, listagem, edição e exclusão de produtos.
- Busca dinâmica e paginação na lista de produtos.

2.5. **Gerenciamento de Fornecedores (CRUD)**

- Cadastro, listagem, edição e exclusão de fornecedores.

2.6. **Movimentação de Estoque**

- Registro de entradas e saídas para cada produto.

- Histórico detalhado de todas as movimentações por produto.

2.7. **Tela de Vendas**

- Interface dinâmica para buscar produtos e adicioná-los a uma venda.
- Cálculo automático do valor total.
- Ao finalizar a venda, o estoque dos produtos é atualizado automaticamente.

2.8. **API Interna**
Um endpoint para busca assíncrona de produtos, utilizado na tela de vendas para uma melhor experiência do usuário.

3. TECNOLOGIAS UTILIZADAS

A aplicação foi construída com as seguintes tecnologias e bibliotecas principais:

3.1. **Backend**

- Python
- Flask
- Flask-SQLAlchemy (ORM para interação com o banco de dados)
- Flask-Login (Gerenciamento de sessões de usuário)
- Flask-WTF (Formulários e validação)
- Werkzeug (Segurança de senhas - hashing)
- python-dotenv (Gerenciamento de variáveis de ambiente)

3.2. **Frontend**

- HTML5
- CSS3
- Bootstrap 5
- Jinja2 (Template Engine do Flask)
- JavaScript (para interatividade na tela de vendas)

3.3. **Banco de Dados**

- SQLite (padrão de desenvolvimento), mas pode ser facilmente trocado para PostgreSQL ou MySQL via SQLAlchemy.

4. INSTALAÇÃO E CONFIGURAÇÃO

Siga os passos abaixo para configurar e executar o projeto em um ambiente de

desenvolvimento local.

4.1. **Pré-requisitos**

- Python 3.8 ou superior
- pip (gerenciador de pacotes do Python)

4.2. **Passos para Instalação**
Clone o repositório:
git clone <URL_DO_SEU_REPOSITORIO>

1. cd <NOME_DA_PASTA_DO_PROJETO>

Crie e ative um ambiente virtual:
Para Windows

python -m venv venv

.\venv\Scripts\activate

Para macOS/Linux

python3 -m venv venv

2. source venv/bin/activate
3. **Instale as dependências:**
   O projeto utiliza as bibliotecas listadas no arquivo `requirements.txt`. Instale todas com um único comando:
   pip install -r requirements.txt
4. **Configure as variáveis de ambiente:**
   - Crie uma cópia do arquivo `.env.example` e renomeie-a para `.env`.
   - Abra o arquivo `.env` e preencha as variáveis. Para a `SECRET_KEY`, você pode gerar uma chave segura (por exemplo, usando o comando `python -c 'import secrets; print(secrets.token_hex())'`).

Arquivo: .env

SECRET_KEY=sua_chave_secreta_aqui

5. SQLALCHEMY_DATABASE_URI='sqlite:///estoque.db'
6. **Inicialize o Banco de Dados:**
   O Flask CLI foi configurado para criar as tabelas do banco de dados e popular dados iniciais (como as Funções de usuário e os Estados). Execute o seguinte comando:
   flask init-db
7. **Execute a Aplicação:**

Após a configuração, inicie o servidor de desenvolvimento do Flask:
flask run
A aplicação estará disponível em `http://127.0.0.1:5000`.

5. ESTRUTURA DO PROJETO

O código está organizado de forma modular para facilitar a manutenção e escalabilidade:

```
├── app/                # Contém o núcleo da aplicação
│   ├── auth/           # Blueprint de autenticação (login, registro)
│   │   ├── forms.py
│   │   └── routes.py
│   ├── main/           # Blueprint principal da aplicação (produtos, vendas, etc.)
│   │   ├── forms.py
│   │   └── routes.py
│   ├── static/         # Arquivos estáticos (CSS, JS, Imagens)
│   ├── templates/      # Arquivos HTML (Jinja2)
│   ├── __init__.py     # Fábrica da aplicação (create_app)
│   ├── db_file.py      # Inicialização do objeto SQLAlchemy
│   ├── decorators.py   # Decoradores customizados (ex: permission_required)
│   └── models.py       # Modelos de dados do SQLAlchemy
├── venv/               # Pasta do ambiente virtual (ignorada pelo .gitignore)
```

1. Descrição do Projeto

Este projeto é uma aplicação web desenvolvida em Flask para o gerenciamento completo de um sistema de controle de estoque. Ele permite o cadastro de produtos, fornecedores, controle de usuários com diferentes níveis de permissão, e o registro de movimentações de estoque (entradas e saídas), além de uma tela de vendas para registrar as operações comerciais.

A aplicação foi construída utilizando uma arquitetura modular com *Blueprints*, o padrão *Application Factory* e uma interface de usuário responsiva baseada em Bootstrap.

2. Funcionalidades

- **Autenticação de Usuários:** Sistema completo de login, registro e logout de

usuários.
- **Controle de Acesso Baseado em Funções (RBAC):**
  - **Admin:** Acesso total ao sistema, incluindo gerenciamento de usuários.
  - **Gerente:** Pode gerenciar produtos, fornecedores e realizar vendas.
  - **Usuário:** Nível de acesso mais restrito (pode ser personalizado).
- **Dashboard Principal:** Exibe estatísticas rápidas sobre o estoque, como total de produtos, valor total em estoque e alertas de produtos com baixa quantidade.
- **Gerenciamento de Produtos (CRUD):**
  - Cadastro, listagem, edição e exclusão de produtos.
  - Busca dinâmica e paginação na lista de produtos.
- **Gerenciamento de Fornecedores (CRUD):**
  - Cadastro, listagem, edição e exclusão de fornecedores.
- **Movimentação de Estoque:**
  - Registro de entradas e saídas para cada produto.
  - Histórico detalhado de todas as movimentações por produto.
- **Tela de Vendas:**
  - Interface dinâmica para buscar produtos e adicioná-los a uma venda.
  - Cálculo automático do valor total.
  - Ao finalizar a venda, o estoque dos produtos é atualizado automaticamente.
- **API Interna:** Um endpoint para busca assíncrona de produtos, utilizado na tela de vendas para uma melhor experiência do usuário.

3. Tecnologias Utilizadas

A aplicação foi construída com as seguintes tecnologias e bibliotecas principais:

- **Backend:**
  - Python
  - Flask
  - Flask-SQLAlchemy (ORM para interação com o banco de dados)
  - Flask-Login (Gerenciamento de sessões de usuário)
  - Flask-WTF (Formulários e validação)
  - Werkzeug (Segurança de senhas - hashing)
  - python-dotenv (Gerenciamento de variáveis de ambiente)
- **Frontend:**
  - HTML5
  - CSS3
  - Bootstrap 5
  - Jinja2 (Template Engine do Flask)
  - JavaScript (para interatividade na tela de vendas)

- **Banco de Dados:**
  - SQLite (padrão de desenvolvimento), mas pode ser facilmente trocado para PostgreSQL ou MySQL via SQLAlchemy.

4. Instalação e Configuração

Siga os passos abaixo para configurar e executar o projeto em um ambiente de desenvolvimento local.

Pré-requisitos

- Python 3.8 ou superior
- pip (gerenciador de pacotes do Python)

Passos para Instalação

1. **Clone o repositório:**
   git clone <URL_DO_SEU_REPOSITORIO>
   cd <NOME_DA_PASTA_DO_PROJETO>

2. **Crie e ative um ambiente virtual:**
   # Para Windows
   python -m venv venv
   .\venv\Scripts\activate

   # Para macOS/Linux
   python3 -m venv venv
   source venv/bin/activate

3. Instale as dependências:
   O projeto utiliza as bibliotecas listadas no arquivo requirements.txt. Instale todas com um único comando:
   pip install -r requirements.txt

4. **Configure as variáveis de ambiente:**
   - Crie uma cópia do arquivo .env.example e renomeie-a para .env.
   - Abra o arquivo .env e preencha as variáveis. Para a SECRET_KEY, você pode gerar uma chave segura (por exemplo, usando o comando python -c 'import secrets; print(secrets.token_hex())').

   # Arquivo: .env
   SECRET_KEY=sua_chave_secreta_aqui
   SQLALCHEMY_DATABASE_URI='sqlite:///estoque.db'

5. Inicialize o Banco de Dados:
   O Flask CLI foi configurado para criar as tabelas do banco de dados e popular dados iniciais (como as Funções de usuário e os Estados). Execute o seguinte comando:
   flask init-db

6. Execute a Aplicação:
   Após a configuração, inicie o servidor de desenvolvimento do Flask:
   flask run

   A aplicação estará disponível em http://127.0.0.1:5000.

5. Estrutura do Projeto

O código está organizado de forma modular para facilitar a manutenção e escalabilidade:

```
├── app/                # Contém o núcleo da aplicação
│   ├── auth/           # Blueprint de autenticação (login, registro)
│   │   ├── forms.py
│   │   └── routes.py
│   ├── main/           # Blueprint principal da aplicação (produtos, vendas, etc.)
│   │   ├── forms.py
│   │   └── routes.py
│   ├── static/         # Arquivos estáticos (CSS, JS, Imagens)
│   ├── templates/      # Arquivos HTML (Jinja2)
│   ├── __init__.py     # Fábrica da aplicação (create_app)
│   ├── db_file.py      # Inicialização do objeto SQLAlchemy
│   ├── decorators.py   # Decoradores customizados (ex: permission_required)
│   └── models.py       # Modelos de dados do SQLAlchemy
├── venv/               # Pasta do ambiente virtual (ignorada pelo .gitignore)
├── .env                # Arquivo com as variáveis de ambiente (local)
├── .env.example        # Exemplo de arquivo .env
├── requirements.txt    # Lista de dependências do Python
└── run.py              # Ponto de entrada para executar a aplicação
```

6. Descrição Detalhada dos Arquivos

6.1. Pasta Raiz

- run.py: Ponto de entrada da aplicação. Este script importa a fábrica de aplicações (create_app) e inicia o servidor de desenvolvimento do Flask.
- requirements.txt: Lista todas as bibliotecas Python necessárias para o projeto. O comando pip install -r requirements.txt instala todas elas de uma vez.
- .env: Arquivo local (ignorado pelo Git) que armazena variáveis de ambiente, como a chave secreta e a URI do banco de dados.
- .env.example: Um arquivo de exemplo que serve como modelo para o .env.
- .gitignore: Especifica quais arquivos e pastas devem ser ignorados pelo sistema de controle de versão Git (ex: venv, __pycache__, .env).

6.2. Pasta app/

Esta é a pasta principal que contém toda a lógica da aplicação.

- __init__.py: Implementa o padrão *Application Factory*. A função create_app() é responsável por:
  - Criar a instância principal do Flask.
  - Carregar as configurações a partir do arquivo .env.
  - Inicializar as extensões (SQLAlchemy, LoginManager).
  - Registrar os *Blueprints* (auth e main).
  - Definir comandos CLI customizados (como flask init-db).
- models.py: Define a estrutura do banco de dados usando classes Python. Cada classe representa uma tabela (ex: Usuario, Produto, Venda). O SQLAlchemy usa esses modelos para criar as tabelas e gerenciar os dados.
- db_file.py: Contém apenas a inicialização do objeto db = SQLAlchemy(). Isso evita problemas de importação circular, permitindo que db seja importado em outros arquivos sem criar dependências cíclicas.
- decorators.py: Contém decoradores Python customizados. O decorador @permission_required é usado para proteger rotas, garantindo que apenas usuários com funções específicas possam acessá-las.

6.2.1. Blueprint app/auth/

Módulo responsável por tudo relacionado à autenticação.

- routes.py: Define as rotas (URLs) para login (/login), registro (/registro) e logout (/logout). Contém a lógica para validar os dados do usuário e gerenciar a sessão.
- forms.py: Define os formulários de login e registro usando a biblioteca Flask-WTF. Inclui validações para os campos (ex: e-mail válido, senhas que coincidem).

6.2.2. Blueprint app/main/

Módulo principal que contém as funcionalidades centrais do sistema.

- routes.py: Define as rotas para o dashboard, gerenciamento de produtos,

fornecedores, movimentações de estoque e a nova tela de vendas. É o arquivo mais denso em lógica de negócios.
- forms.py: Define os formulários utilizados neste blueprint, como o formulário de produto, fornecedor e movimentação de estoque.

6.2.3. Pastas static/ e templates/

- static/: Armazena arquivos que não mudam, como CSS (style.css), imagens e arquivos JavaScript. O Flask os serve diretamente para o navegador.
- templates/: Contém os arquivos HTML que são renderizados pelo Flask usando o motor de templates Jinja2. A estrutura inclui templates base (base.html, dashboard_base.html) que são estendidos por outras páginas, evitando a repetição de código HTML.

7. Como Usar

1. Após iniciar a aplicação, acesse a URL http://127.0.0.1:5000.
2. Você será redirecionado para a tela de login.
3. Clique em "Registre-se aqui" para criar sua primeira conta. O primeiro usuário criado terá a função padrão (geralmente 'Admin' ou 'Usuário', dependendo da configuração inicial).
4. Faça login com a conta criada para acessar o dashboard principal.
5. Utilize o menu lateral para navegar entre as funcionalidades de Produtos, Fornecedores, Vendas e, se for Admin, Gerenciamento de Usuários.