### **PROMPT MESTRE FINAL PARA GERAÇÃO DE APLICAÇÃO WEB**

**Atue como:** Desenvolvedor Web Sênior focado em soluções práticas e eficientes.

**Sua tarefa é:** Gerar o código completo para uma aplicação web monolítica chamada **"GestokPro"**. O sistema deve ser totalmente autocontido, criando seu próprio banco de dados localmente, e deve se autoavaliar, exibindo métricas de desempenho diretamente em sua interface.

### **1. Especificações Gerais e Tecnologias**

* **Framework Backend:** Python 3.10+ com **Flask**.
* **Renderização de Frontend:** **Jinja2** (templates renderizados no servidor).
* **Estilização:** **Tailwind CSS** (usando o CDN para simplicidade).
* **Banco de Dados:** **SQLite**. O banco de dados será um único arquivo (gestokpro.db) no diretório do projeto, criado automaticamente pela aplicação. Isso elimina qualquer necessidade de configuração de banco de dados pelo usuário.
* **ORM:** **SQLAlchemy** com Flask-SQLAlchemy.
* **Autenticação:** Baseada em sessão do Flask (flask-login).

### **2. Estrutura do Projeto Autocontido**

Gere a seguinte estrutura de arquivos. Note que não há mais o arquivo .env, pois a configuração do banco de dados é local e fixa.

/  
|-- app.py # Arquivo principal com rotas e lógica  
|-- models.py # Definição dos modelos SQLAlchemy (tabelas)  
|-- forms.py # Definição dos formulários com Flask-WTF  
|-- templates/  
| |-- base.html # Layout principal com menu e métricas  
| |-- login.html  
| |-- dashboard.html  
| |-- produtos.html  
| |-- produto\_form.html  
|-- static/  
| |-- css/  
| | |-- styles.css  
|-- requirements.txt  
|-- init\_db.py # Script para criar e popular o banco de dados inicial

### **3. Detalhamento do Backend (Python/Flask)**

#### **3.1. Modelos de Dados (models.py)**

* **Usuario**:
  + Campos: id, email, senha\_hash, is\_admin (Boolean).
  + Métodos: set\_password(password) e check\_password(password).
* **Produto**:
  + Campos: id, nome, sku, descricao, quantidade, preco\_venda.

#### **3.2. Rotas e Lógica Principal (app.py)**

Este arquivo conterá toda a lógica da aplicação.

* **Configuração:**
  + Inicialize o Flask.
  + Configure o SQLAlchemy para usar SQLite: app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///gestokpro.db'.
  + Antes da primeira requisição, use db.create\_all() para criar o arquivo de banco de dados e as tabelas automaticamente se não existirem.
* **Autenticação:**
  + Implemente o fluxo padrão do flask-login com as rotas /login e /logout.
* **Rotas Protegidas** (@login\_required):
  + /dashboard: Além de renderizar a página, esta rota deve calcular métricas como "Total de Produtos" e "Quantidade Total em Estoque" e passá-las para o template.
  + /produtos, /produtos/novo, /produtos/editar/<id>, /produtos/excluir/<id>: Implemente o CRUD completo para produtos.
  + /produtos/movimentar/<int:id> (POST): Rota única para adicionar ou subtrair do estoque, recebendo a quantidade (positiva para entrada, negativa para saída).

### **4. Medição e Exibição de Desempenho (Autoavaliação)**

Esta é a parte central da nova especificação.

#### **4.1. Medição do Tempo de Resposta no Backend**

* **No app.py**: Use os decoradores de aplicação do Flask para medir o tempo de cada requisição.  
  import time  
  from flask import g  
    
  @app.before\_request  
  def before\_request\_func():  
   # Armazena o tempo de início no objeto 'g' do Flask,  
   # que persiste durante uma única requisição.  
   g.start\_time = time.time()  
    
  @app.after\_request  
  def after\_request\_func(response):  
   # Calcula o tempo total e o adiciona a um cabeçalho  
   # para fins de depuração na aba Network.  
   diff\_ms = (time.time() - g.start\_time) \* 1000  
   response.headers['X-Request-Time-Ms'] = f"{diff\_ms:.2f}"  
   return response  
    
  @app.context\_processor  
  def inject\_performance\_metrics():  
   # Esta função torna a métrica de tempo disponível para TODOS os templates Jinja2.  
   # Assim, podemos exibi-la facilmente no layout base.  
   if 'start\_time' in g:  
   diff\_ms = (time.time() - g.start\_time) \* 1000  
   return dict(response\_time\_ms=f"{diff\_ms:.2f}")  
   return dict()

#### **4.2. Exibição das Métricas no Frontend**

* **No base.html**:
  + Adicione um <footer> fixo na parte inferior da página.
  + Dentro do rodapé, exiba a métrica de desempenho injetada pelo context\_processor.  
    <footer class="bg-gray-100 text-center text-xs p-2 fixed bottom-0 w-full">  
     <!-- A variável 'response\_time\_ms' vem diretamente do backend (app.py) -->  
     Página gerada em: <span class="font-bold text-blue-600">{{ response\_time\_ms }} ms</span>  
    </footer>
* **No dashboard.html**:
  + Crie uma seção de "Métricas do Sistema".
  + Renderize cards com as métricas calculadas na rota /dashboard.  
    <div class="grid grid-cols-1 md:grid-cols-2 gap-4">  
     <div class="bg-white p-4 rounded-lg shadow">  
     <h3 class="font-bold text-lg">Total de Produtos</h3>  
     <!-- A variável 'total\_produtos' vem da rota /dashboard -->  
     <p class="text-3xl">{{ total\_produtos }}</p>  
     </div>  
     <div class="bg-white p-4 rounded-lg shadow">  
     <h3 class="font-bold text-lg">Itens em Estoque</h3>  
     <p class="text-3xl">{{ total\_itens\_estoque }}</p>  
     </div>  
    </div>

### **5. Instruções Finais de Geração**

* Gere o código para cada arquivo, garantindo que a lógica de criação do banco de dados e a medição de desempenho estejam implementadas conforme descrito.
* O código deve ser limpo, bem comentado e autocontido, funcionando "out-of-the-box" após a instalação das dependências.
* Comece a geração pelo app.py para estabelecer a estrutura principal da aplicação.

### **6. Prompt Final para Geração no Replit**

Copie e cole o texto abaixo na ferramenta de IA do Replit para gerar o projeto completo.

Gere uma aplicação web completa em Flask chamada "GestokPro".  
  
\*\*Requisitos Principais:\*\*  
1. \*\*Backend:\*\* Use Python com Flask, Flask-SQLAlchemy, Flask-Login e Flask-WTF.  
2. \*\*Banco de Dados:\*\* Use SQLite. Crie um script `init\_db.py` que, quando executado, apaga o banco de dados existente (`gestokpro.db`), recria as tabelas e o popula com dados de exemplo.  
3. \*\*Modelos (`models.py`):\*\* Crie os modelos `Usuario` (id, email, senha\_hash, is\_admin) e `Produto` (id, nome, sku, descricao, quantidade, preco\_venda).  
4. \*\*Frontend:\*\* Use templates Jinja2 com Tailwind CSS (via CDN). Crie um `base.html` e templates para login, dashboard e um CRUD completo de produtos (`produtos.html` e `produto\_form.html`).  
5. \*\*Funcionalidades:\*\*  
 \* Sistema de login e logout baseado em sessão.  
 \* CRUD completo para a entidade "Produto".  
 \* Rotas para movimentar estoque (entrada/saída).  
6. \*\*Autoavaliação de Desempenho:\*\*  
 \* Em `app.py`, use `before\_request` e `after\_request` para medir o tempo de resposta de cada página.  
 \* Use um `context\_processor` para tornar a métrica de tempo (em ms) disponível em todos os templates.  
 \* Exiba essa métrica em um rodapé no `base.html` com o texto: "Página gerada em: {{ response\_time\_ms }} ms".  
7. \*\*Dados Iniciais (Seeding):\*\* O script `init\_db.py` deve inserir um usuário administrador padrão (ex: `admin@gestokpro.com` / senha `admin`) e pelo menos 3 produtos de exemplo para que o banco de dados não comece vazio.