Relatório de Reestruturação do Código ERP

1. Introdução

Este relatório detalha as melhorias e reestruturações aplicadas ao código do sistema ERP, originalmente desenvolvido em Python com SQLite e Tkinter. O objetivo principal foi aprimorar a modularidade, segurança, desempenho e manutenibilidade do sistema.

2. Estrutura Geral e Arquitetura

O código foi reestruturado para seguir um padrão mais modular, separando as responsabilidades em diferentes diretórios e arquivos:

- `erp_refatorado/`
- `database/`
  - `database_manager.py`: Gerencia a conexão com o banco de dados SQLite e as operações de criação de tabelas.
- `models/`
  - `models.py`: Define as classes de modelo de dados (Cliente, Usuário, Fornecedor, Produto) usando `dataclasses` para clareza e tipagem.
- `business_logic/`
  - `client_manager.py`: Contém a lógica de negócios para operações CRUD de clientes.
  - `user_manager.py`: Contém a lógica de negócios para operações CRUD de usuários, incluindo hashing de senhas.
  - `supplier_manager.py`: Contém a lógica de negócios para operações CRUD de fornecedores.

    - `product_manager.py`: Contém a lógica de negócios para operações CRUD de produtos e gerenciamento de estoque.
- `gui/`
    - `gui_components.py`: Contém classes e funções para componentes de interface gráfica reutilizáveis.
    - `main_app.py`: A aplicação principal Tkinter, responsável pela construção da interface e integração com as classes de gerenciamento.

Essa separação de responsabilidades melhora a organização do código, facilita a manutenção e permite que futuras modificações sejam feitas em módulos específicos sem afetar outras partes do sistema.

3. Implementação do Banco de Dados SQLite

3.1. Gerenciamento de Conexão

Foi implementada a classe `DatabaseManager` para centralizar a conexão com o SQLite. Esta classe utiliza um gerenciador de contexto (`__enter__` e `__exit__`) para garantir que a conexão seja aberta e fechada corretamente, prevenindo vazamentos de recursos e garantindo a integridade das transações.

3.2. Normalização e Consistência

As tabelas foram revisadas para garantir uma melhor normalização, reduzindo a redundância de dados. As operações de criação de tabelas foram movidas para o `DatabaseManager`, assegurando que o esquema do banco de dados seja inicializado de forma consistente.

3.3. Prevenção de SQL Injection

Todas as consultas SQL que envolvem entrada do usuário agora utilizam parâmetros (`?`) em vez de concatenação direta de strings. Isso previne ataques de SQL Injection, tornando o sistema mais seguro.

4. Modelos de Dados

As entidades do sistema (Cliente, Usuário, Fornecedor, Produto) foram representadas por classes Python no arquivo `models.py` utilizando `dataclasses`. Isso proporciona:

- **Clareza**: A estrutura dos dados é explicitamente definida.
- **Tipagem**: Facilita a validação de dados e melhora a legibilidade do código.
- **Manutenibilidade**: Alterações na estrutura dos dados são mais fáceis de gerenciar.

5. Lógica de Negócios

A lógica de negócios foi encapsulada em classes de gerenciamento (`ClientManager`, `UserManager`, `SupplierManager`, `ProductManager`). Cada classe é responsável pelas operações CRUD (Criar, Ler, Atualizar, Deletar) de sua respectiva entidade. Isso promove:

- **Separação de Preocupações**: A lógica de negócios está desacoplada da interface do usuário e do acesso ao banco de dados.
- **Reusabilidade**: As classes de gerenciamento podem ser reutilizadas em diferentes contextos (por exemplo, em uma API REST futura).
- **Testabilidade**: Facilita a escrita de testes de unidade para a lógica de negócios.

6. Interface Gráfica Tkinter

6.1. Modularização da GUI

A interface gráfica foi reestruturada para ser mais modular. Componentes reutilizáveis (como campos de entrada com validação) podem ser definidos em `gui_components.py`.

6.2. Validação de Entrada

Foram adicionadas validações de entrada para garantir que os dados inseridos pelos usuários estejam no formato correto antes de serem processados ou armazenados no

banco de dados. Isso melhora a robustez do sistema e a qualidade dos dados.

6.3. Limpeza de Entradas

As funções de limpeza de campos de entrada foram refatoradas para serem mais eficientes e consistentes em toda a aplicação.

7. Segurança

7.1. Hashing de Senhas

Foi implementado o hashing de senhas utilizando a biblioteca `bcrypt` para a tabela de usuários. Isso garante que as senhas não sejam armazenadas em texto simples no banco de dados, protegendo-as contra acessos não autorizados. A função de login agora verifica a senha fornecida contra o hash armazenado.

8. Testes

Devido às limitações do ambiente de sandbox, testes de interface gráfica não puderam ser executados. No entanto, a modularização do código facilita a implementação de testes de unidade e integração para as camadas de lógica de negócios e banco de dados em um ambiente de desenvolvimento adequado.

9. Conclusão

A reestruturação do código resultou em um sistema ERP mais organizado, seguro e fácil de manter. As melhorias na modularidade, segurança do banco de dados e tratamento de senhas são passos importantes para a robustez da aplicação. Recomenda-se a implementação de testes automatizados e a consideração de um framework de GUI mais moderno para futuras expansões.