Projeto: Sistema de Controle de Estoque (SCE)
Pedro Lucas Dutra

1. Documentação do Projeto

1.1. Descrição do Cenário/Problema:

Pequenas e médias empresas (PMEs) frequentemente enfrentam desafios no gerenciamento de seus estoques. O controle manual, realizado por meio de planilhas ou cadernos, é um processo lento, suscetível a erros humanos e que não fornece informações em tempo real. Essa falta de controle pode levar a problemas como:

- Ruptura de estoque: A falta de um produto quando um cliente deseja comprá-lo, resultando em perda de vendas e insatisfação do cliente.
- Excesso de estoque: Produtos parados por muito tempo, representando capital imobilizado e risco de perdas por obsolescência ou validade.
- Dificuldade na tomada de decisão: Sem dados precisos, é difícil saber quais produtos têm maior giro, qual o momento certo de comprar e como planejar promoções.

O Sistema de Controle de Estoque (SCE) visa solucionar esses problemas, oferecendo uma plataforma digital simples e centralizada para o gerenciamento de produtos e movimentações de estoque. O sistema permitirá o cadastro de produtos, o registro de entradas e saídas, e a visualização de relatórios, automatizando o controle e fornecendo dados confiáveis para uma gestão mais eficiente.

1.2. Levantamento e Especificação dos Requisitos

Requisitos Funcionais (RF)

- RF01: O sistema deve permitir o cadastro de novos produtos, informando nome, descrição, preço de venda e quantidade inicial.
- RF02: O sistema deve permitir a alteração dos dados de um produto existente (nome, descrição, preço). A quantidade não pode ser alterada diretamente, apenas por meio de movimentações.
- RF03: O sistema deve permitir a exclusão de um produto do catálogo.
- RF04: O sistema deve permitir a listagem de todos os produtos cadastrados, exibindo todas as suas informações, incluindo a quantidade atual em estoque.
- RF05: O sistema deve permitir a busca de um produto específico pelo seu nome.
- RF06: O sistema deve permitir o registro de entrada de itens no estoque para um produto específico.
- RF07: O sistema deve permitir o registro de saída de itens do estoque para um produto específico. O sistema não deve permitir que uma saída deixe o estoque com quantidade negativa.

- RF08: O sistema deve gerar um relatório simples de estoque, listando todos os produtos e suas quantidades atuais.

Requisitos Não Funcionais (RNF)

- RNF01: O sistema deve possuir uma interface de linha de comando (CLI) para interação com o usuário.
- RNF02: Os dados do sistema devem ser persistidos de forma que não se percam ao fechar a aplicação.
- RNF03: O sistema deve ser desenvolvido na linguagem Python.
- RNF04: A resposta do sistema para as operações de cadastro e consulta deve ser rápida, idealmente em menos de 2 segundos para uma base de até 10.000 produtos.
- RNF05: A interface deve ser intuitiva e guiar o usuário através de menus de opções.

1.3. Arquitetura do Sistema

O padrão de arquitetura adotado será a Arquitetura em Camadas (Layered Architecture):

- Camada de Apresentação (Presentation Layer): Responsável pela interação com o usuário. Neste projeto, será a interface de linha de comando (CLI), que exibe menus, coleta dados e apresenta resultados.
- Camada de Lógica de Negócio (Business Logic Layer / Service Layer): Contém as regras de negócio do sistema. Orquestra as operações, valida dados e coordena o fluxo de informações entre a apresentação e o acesso a dados. Por exemplo, é nesta camada que a regra "não permitir estoque negativo" é validada.
- Camada de Acesso a Dados (Data Access Layer): Responsável pela comunicação com a fonte de dados (banco de dados). Abstrai os detalhes de como os dados são armazenados e recuperados, implementando o padrão DAO.

Justificativa da Escolha:

A arquitetura em camadas foi escolhida por sua clareza na separação de responsabilidades (Separation of Concerns). Cada camada tem um papel bem definido, o que resulta em:

- Manutenibilidade: Alterações na interface (ex: trocar a CLI por uma interface web no futuro) não impactam a lógica de negócio ou o acesso a dados.
- Testabilidade: Cada camada pode ser testada de forma isolada.
- Reusabilidade: A lógica de negócio e o acesso a dados podem ser reutilizados por diferentes interfaces de apresentação.
- Organização: O código fica mais limpo, organizado e fácil de entender.

1.4. Modelagem UML (Usando PlantUML)

Diagrama de Casos de Uso:

Este diagrama ilustra as interações do ator "Gestor de Estoque" com as principais funcionalidades do sistema.

```
@startuml
left to right direction
actor "Gestor de Estoque" as gestor

rectangle "Sistema de Controle de Estoque (SCE)" {
  gestor -- (Cadastrar Produto)
  gestor -- (Editar Produto)
  gestor -- (Excluir Produto)
  gestor -- (Listar Produtos)
  gestor -- (Buscar Produto)
  gestor -- (Registrar Entrada de Estoque)
  gestor -- (Registrar Saída de Estoque)
  gestor -- (Gerar Relatório de Estoque)
}
@enduml
```

Diagrama de Classes:

Este diagrama mostra a estrutura estática das classes do sistema, seus atributos, métodos e os relacionamentos entre elas, refletindo a arquitetura em camadas e o padrão DAO.

```
@startuml

class MainApp {
  - produto_service: ProdutoService
  + main_loop()
  + exibir_menu()
}

class ProdutoService {
  - produto_dao: IProdutoDAO
  + adicionar_produto(nome, desc, preco, qtd)
  + atualizar_produto(id, nome, desc, preco)
  + remover_produto(id)
  + listar_todos()
  + registrar_entrada(id, quantidade)
  + registrar_saida(id, quantidade)
}

interface IProdutoDAO {
  + salvar(produto)
  + atualizar(produto)
  + deletar(id)
  + buscar_por_id(id)
  + listar_todos()
}

class ProdutoDAOSQLite {
  - conexao
  + salvar(produto)
  + atualizar(produto)
  + deletar(id)
  + buscar_por_id(id)
  + listar_todos()
}

class Produto {
  - id_produto: int
  - nome: str
  - descricao: str
  - preco: float
  - quantidade: int
}
```

```
class Database {
  + conectar()
  + criar_tabela()
}

MainApp o--> ProdutoService
ProdutoService o--> IProdutoDAO
ProdutoDAOSQLite ..|> IProdutoDAO
ProdutoService ..> Produto
IProdutoDAO ..> Produto
ProdutoDAOSQLite o--> Database

@enduml
```

1.5. Persistência de Dados

Estratégia Adotada:

A persistência de dados será realizada utilizando um banco de dados relacional, especificamente o SQLite.

Justificativa Técnica:

- Simplicidade e Portabilidade: O SQLite é um banco de dados serverless, ou seja, não requer um processo de servidor separado. Ele armazena o banco de dados inteiro em um único arquivo no disco, tornando a aplicação autocontida e fácil de distribuir.
- Biblioteca Padrão: O Python possui suporte nativo ao SQLite através do módulo sqlite3, eliminando a necessidade de instalar drivers ou bibliotecas externas.
- Adequação ao Escopo: Para um sistema de pequeno a médio porte, como o proposto, o desempenho e a capacidade do SQLite são mais do que suficientes.
- Facilidade de Backup: O backup do banco de dados pode ser feito simplesmente copiando um único arquivo.

Modelagem do Banco de Dados:

O sistema utilizará uma única tabela para armazenar as informações dos produtos.

Nome da Coluna	Tipo de Dado	Restrições	Descrição
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Identificador único do produto.
nome	TEXT	NOT NULL	Nome do produto.

descricao	TEXT		Descrição detalhada do produto.
preco	REAL	NOT NULL	Preço de venda unitário.
quantidade	INTEGER	NOT NULL DEFAULT 0	Quantidade atual em estoque.

SQL para Criação da Tabela:

```sql
CREATE TABLE IF NOT EXISTS produtos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL,
    descricao TEXT,
    preco REAL NOT NULL,
    quantidade INTEGER NOT NULL DEFAULT 0
);
```

1.6. Padrão de Projeto DAO (Data Access Object)

O padrão DAO será central na implementação da camada de acesso a dados. Ele tem como objetivo desacoplar a lógica de negócio da lógica de persistência.

A Camada de Lógica de Negócio (ProdutoService) não saberá como os dados são salvos (se em SQLite, PostgreSQL, ou um arquivo de texto). Ela apenas interage com uma interface (IProdutoDAO).

A classe ProdutoDAOSQLite será a implementação concreta dessa interface, contendo todo o código SQL específico para o SQLite.

Isso permite que, no futuro, a forma de armazenamento possa ser trocada (ex: migrar para um banco de dados em nuvem) alterando apenas a classe DAO, sem impactar o resto do sistema.