Relatório Técnico do Sistema

1. Visão Geral

O sistema é dividido em três classes principais, conforme o padrão MVC:

- **Produto (Model):** Representa a entidade de dados, que é o produto em si. Armazena as informações (ID, Nome, Preço e Quantidade) e provê métodos para acesso e modificação (getters/setters).
- **ProdutoController (Controller/Service):** Contém a lógica de negócios e o gerenciamento da coleção de produtos. Ele atua como o intermediário entre a View e o Model, manipulando os dados e aplicando as regras de CRUD.
- **ProdutoView (View):** É a interface gráfica do usuário (GUI), construída com Tkinter. Ela lida com a entrada de dados do usuário, exibe a lista de produtos (usando Treeview) e chama os métodos apropriados no Controller em resposta às ações do usuário.

2. Análise de Componentes

Model: Produto.py

- **Propriedades:** _id, _nome, _preco, _quantidade.
- **Controle de ID:** Utiliza uma variável de classe (_proximo_id) para **auto-incrementar** e atribuir um ID único a cada nova instância de Produto, simulando o comportamento de uma chave primária em um banco de dados.
- **Encapsulamento:** Os atributos são prefixados com _ (convenção Python para atributos protegidos/privados), acessíveis via métodos **Getters e Setters**.
- **Funcionalidades:** Inclui um método to_tuple() crucial para formatar os dados do produto para exibição na tabela Treeview da interface.

Controller/Service: Service.py

- **Estrutura de Dados:** Armazena os objetos Produto em uma lista interna (self._produtos).
- **Métodos CRUD:**
    - adicionar(produto): Adiciona o objeto Produto à lista.
    - listar(): Retorna a lista completa de produtos.
    - buscar_por_id(id_produto): Implementa a busca linear na lista.
    - atualizar(id_produto, novo_produto): Localiza o produto pelo ID e copia os atributos (Nome, Preço, Quantidade) do objeto novo_produto para a instância existente.
    - remover(id_produto): Localiza e remove o objeto Produto da lista.
- **Independência:** O Controller não tem conhecimento da View (ProdutoView) e lida apenas com objetos Produto, garantindo uma boa separação de responsabilidades.

View: Interface.py

- **Interface:** Utiliza **Tkinter** com a biblioteca **ttk (Themed Tkinter)** para widgets modernos.
- **Campos de Entrada:** Usa variáveis de controle (tk.StringVar, tk.DoubleVar, tk.IntVar) para vincular as entradas de texto (Entry) aos dados.
- **Tabela (Treeview):** Exibe a lista de produtos com colunas formatadas e alinhadas.

- **Eventos:**
    - selecionar_item(): Preenche os campos de entrada com os dados do produto selecionado na tabela, e armazena o selected_id.
- **Validação:** O método validar_campos() realiza checagens básicas de preenchimento (Nome obrigatório, Preço > 0, Quantidade ≥ 0) e exibe messagebox em caso de erro.
- **Interação com o Controller:** Todos os métodos de ação (adicionar_produto, atualizar_produto, remover_produto) chamam o Controller, atualizam a interface (listar_produtos()) e utilizam messagebox para feedback ao usuário.

3. Conclusão

A arquitetura MVC foi bem aplicada, resultando em um código **modular**, **manutenível** e com **separação clara de responsabilidades**. O sistema cumpre todos os requisitos de um CRUD básico, usando a memória RAM para persistência de dados (através da lista no Controller).