# 

**Instituição**: Uceff

**Nome do Projeto:** Sistema de controle de estoque

**Período:** 17.11.2025 — 06.12.2025

**Aluno:** Cristian Finger - [cristianfinger98@gmail.com](mailto:cristianfinger98@gmail.com)

**GitHub:** https://github.com/Cristianfinger/projeto\_integrador-controle\_de\_estoque-desktop

**Professor:** Caio Vinicio Koch dos Santos - caio@uceff.edu.br

# Projeto: Controle de Estoque

Desenvolvimento de um Sistema Desktop para Controle de Estoque com Python e SQLite

### Resumo

Este trabalho apresenta o desenvolvimento de um sistema desktop para controle de estoque, destacando o processo de modelagem de dados, arquitetura da aplicação e resultados obtidos. O sistema foi implementado em Python com interface gráfica em Tkinter e banco de dados SQLite.

### Introdução

Apresenta a relevância do controle de estoque em pequenas empresas e a necessidade de soluções simples e de baixo custo.

### Objetivo

Criar um sistema de controle de estoque confiável/seguro e que atenda as necessidades da empresa.

### Objetivo especifico

* Desenvolver programa desktop
* Integrar com banco de dados
* Integrar ao sistema da empresa

### Problema/Justificativa

A empresa onde trabalho não possui um sistema de controle de estoque confiável, utilizando métodos antigos (anotação em cadernos e planilhas excel) gerando confusão e dificuldade de fazer o controle.

### Desenvolvimento/requisitos funcionais

* Cadastrar produto
* Classificar por classe de produto
* Alerta de estoque
* Apagar produto
* Gerar relatório

### Desenvolvimento/requisitos não funcionais

* Tecnologias utilizadas: Python 3.13, kinter, SQLite, CSV, PyInstaller (para gerar executável).

### Metodologia

* Levantamento de requisitos com stakeholders
* Modelagem conceitual (MER) e lógica
* Implementação modular em Python
* Testes funcionais e validação com casos de uso

### Resultados

* Sistema funcional com cadastro de categorias, produtos e movimentações
* Histórico de movimentações e exportação para CSV
* Mecanismo básico de alertas de estoque mínimo

## Índice

1. Visão geral do projeto
2. MER (Modelo Entidade-Relacionamento)
3. Diagramas UML
   * Caso de Uso
   * Diagrama de Sequência
   * Diagrama de Banco de Dados (Classe/Modelo)
4. Documento de Apresentação
5. Organização e Código-Fonte
   * Código modularizado
6. Conclusão

## 1. Visão geral do projeto

Aplicativo desktop para controle de estoque desenvolvido em Python com interface Tkinter e banco de dados SQLite. Objetivos principais: - Registrar categorias, produtos e movimentações (entradas/saídas). - Manter histórico de movimentações com data e observação. - Alertar produtos com estoque abaixo do mínimo. - Exportar catálogo de produtos para CSV.

## 2. MER (Modelo Entidade-Relacionamento)

Abaixo está o MER em sintaxe. Ele mostra as entidades principais e seus atributos.

MerDiagram  
 CATEGORIAS {  
 INTEGER id PK  
 TEXT nome UNIQUE  
 }  
 PRODUTOS {  
 INTEGER id PK  
 TEXT nome  
 INTEGER categoria\_id FK  
 REAL preco  
 INTEGER quantidade  
 INTEGER min\_estoque  
 }  
 MOVIMENTACOES {  
 INTEGER id PK  
 INTEGER produto\_id FK  
 TEXT tipo  
 INTEGER quantidade  
 TEXT data  
 TEXT observacao  
 }  
  
 CATEGORIAS ||--o{ PRODUTOS : contem  
 PRODUTOS ||--o{ MOVIMENTACOES : registra

## 3. Diagramas UML

### 3.1 Caso de Uso

ator Usuário: U  
 U --> (Gerenciar Categorias)  
 U --> (Gerenciar Produtos)  
 U --> (Registrar Movimentação)  
 U --> (Exportar CSV)  
 U --> (Consultar Movimentações)  
 U --> (Receber Alertas de Estoque)

**Atores:** - Usuário (operador do sistema)

**Casos principais:** - Gerenciar categorias: criar e listar categorias. - Gerenciar produtos: criar, editar, excluir e buscar produtos. - Registrar movimentação: entrada/saída com observação. - Exportar catálogo: gerar CSV. - Consultar movimentações: visualizar histórico. - Alertas: verificar produtos com estoque baixo.

### 3.2 Diagrama de Sequência (ex.: registrar saída de produto)

participant U as Usuario  
 participant GUI as App/GUI  
 participant DB as SQLite  
  
 U->>GUI: Seleciona produto e escolhe 'Registrar Saída'  
 GUI->>U: Pergunta quantidade (simpledialog)  
 U->>GUI: Informa quantidade  
 GUI->>DB: Inserir movimentação (produto\_id, 'saida', quantidade, data, obs)  
 DB-->>GUI: Confirma inserção  
 GUI->>DB: Atualizar produtos SET quantidade = quantidade - X  
 DB-->>GUI: Confirma atualização  
 GUI->>U: Exibe mensagem de sucesso e atualiza listas

### 3.3 Diagrama de Banco de Dados (Classe / Modelo)

classDiagram  
 class Categoria{  
 +int id  
 +str nome  
 }  
 class Produto{  
 +int id  
 +str nome  
 +int categoria\_id  
 +float preco  
 +int quantidade  
 +int min\_estoque  
 }  
 class Movimentacao{  
 +int id  
 +int produto\_id  
 +str tipo  
 +int quantidade  
 +str data  
 +str observacao  
 }  
 Categoria "1" -- "\*" Produto : contém  
 Produto "1" -- "\*" Movimentacao : registra

## 4. Documento de Apresentação

### 4.1 Resumo

O sistema permite o controle do inventário de uma pequena/média empresa, registrando produtos, categorias e movimentações. Possui alertas de estoque mínimo e exportação para CSV.

Na empresa onde trabalho existe uma grande entrada e saída de material/estoque, e foi constatado por mim a dificuldade de se manter um registro estável e confiável pois ainda e realizado o controle do estoque em cadernos e em planilhas excel, tornando todo o trabalho de controle de estoque confuso e desorganizado. Vendo a necessidade de um sistema de estoque seguro, confiável, pratico e fácil, decidi desenvolver o mesmo para a empresa e o projeto integrador.

O sistema tem um funcionamento simples, mas atende as necessidades da empresa, possui: entrada e saída de estoque, registro e exclusão de produto, busca e alerta de estoque e ainda e possível exportar relatórios do estoque. Com todas as funcionalidades necessárias para um melhor e mais seguro controle de estoque pretendo atender as necessidades e melhorar a forma como e feito o controle do estoque da empresa.

### 4.2 Objetivos

* Controle preciso de quantidades e valores.
* Histórico de movimentações para auditoria.
* Interface simples e intuitiva para operadores.

### 4.3 Fluxos de processo

1. Cadastro de categorias
2. Cadastro de produtos vinculados a categorias
3. Registro de entradas e saídas
4. Monitoramento de níveis mínimos de estoque
5. Geração de relatórios (CSV)

### 4.4 Tecnologias e justificativas

* **Python**: rápido desenvolvimento e portabilidade
* **Tkinter**: GUI nativa leve para aplicações desktop
* **SQLite**: banco embarcado, sem necessitar servidor
* **PyInstaller**: empacotamento em executável para distribuição

### 4.5 Desafios enfrentados

* Validação de entradas do usuário (tipos, limites)
* Evitar inconsistências (movimentações que levem a estoque negativo)
* Exportação e compatibilidade de encoding para CSV

### 4.6 Roadmap / Melhorias futuras

* Autenticação de usuários e níveis de permissão
* Relatórios em PDF com gráficos (valores por categoria)
* Interface mais moderna
* Integração com sistema web para aplicar a futuras filiais

## 5. Código-Fonte

# controle\_estoque\_app.py

# Aplicativo de controle de estoque usando linguagem python, Tkinter e SQLite

# Autor: Cristian Finger

# data\_inicio: 17.11.2025

#data\_termino: 06.12.2025

from tkinter import \*

from tkinter import ttk, messagebox, simpledialog, filedialog

import sqlite3

import os

import csv

from datetime import datetime

BASE\_DIR = os.path.dirname(\_\_file\_\_)

DB\_PATH = os.path.join(BASE\_DIR, "dados", "estoque.db")

def ensure\_db():

    os.makedirs(os.path.join(BASE\_DIR, "dados"), exist\_ok=True)

    conn = sqlite3.connect(DB\_PATH)

    c = conn.cursor()

    c.execute("""CREATE TABLE IF NOT EXISTS categorias (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    nome TEXT UNIQUE NOT NULL

                );""")

    c.execute("""CREATE TABLE IF NOT EXISTS produtos (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    nome TEXT NOT NULL,

                    categoria\_id INTEGER,

                    preco REAL DEFAULT 0,

                    quantidade INTEGER DEFAULT 0,

                    min\_estoque INTEGER DEFAULT 0,

                    FOREIGN KEY(categoria\_id) REFERENCES categorias(id)

                );""")

    c.execute("""CREATE TABLE IF NOT EXISTS movimentacoes (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    produto\_id INTEGER NOT NULL,

                    tipo TEXT NOT NULL,

                    quantidade INTEGER NOT NULL,

                    data TEXT NOT NULL,

                    observacao TEXT,

                    FOREIGN KEY(produto\_id) REFERENCES produtos(id)

                );""")

    conn.commit()

    conn.close()

def run\_query(query, params=(), fetch=False):

    conn = sqlite3.connect(DB\_PATH)

    c = conn.cursor()

    c.execute(query, params)

    if fetch:

        rows = c.fetchall()

        conn.close()

        return rows

    conn.commit()

    conn.close()

def add\_categoria(nome):

    try:

        run\_query("INSERT INTO categorias (nome) VALUES (?)", (nome,))

        return True

    except sqlite3.IntegrityError:

        return False

def list\_categorias():

    return run\_query("SELECT id, nome FROM categorias ORDER BY nome", fetch=True)

def add\_produto(nome, categoria\_id, preco, quantidade, min\_estoque):

    run\_query("""INSERT INTO produtos (nome, categoria\_id, preco, quantidade, min\_estoque)

                 VALUES (?, ?, ?, ?, ?)""", (nome, categoria\_id, preco, quantidade, min\_estoque))

def update\_produto(pid, nome, categoria\_id, preco, quantidade, min\_estoque):

    run\_query("""UPDATE produtos SET nome=?, categoria\_id=?, preco=?, quantidade=?, min\_estoque=?

                 WHERE id=?""", (nome, categoria\_id, preco, quantidade, min\_estoque, pid))

def delete\_produto(pid):

    run\_query("DELETE FROM produtos WHERE id=?", (pid,))

def list\_produtos(search=None):

    if search:

        term = f"%{search}%"

        return run\_query("""SELECT p.id, p.nome, c.nome, p.preco, p.quantidade, p.min\_estoque

                            FROM produtos p LEFT JOIN categorias c ON p.categoria\_id=c.id

                            WHERE p.nome LIKE ? OR c.nome LIKE ?

                            ORDER BY p.nome""", (term, term), fetch=True)

    return run\_query("""SELECT p.id, p.nome, c.nome, p.preco, p.quantidade, p.min\_estoque

                        FROM produtos p LEFT JOIN categorias c ON p.categoria\_id=c.id

                        ORDER BY p.nome""", fetch=True)

def get\_produto(pid):

    res = run\_query("SELECT id, nome, categoria\_id, preco, quantidade, min\_estoque FROM produtos WHERE id=?", (pid,), fetch=True)

    return res[0] if res else None

def add\_movimentacao(produto\_id, tipo, quantidade, observacao=""):

    data = datetime.now().isoformat(sep=' ', timespec='seconds')

    run\_query("""INSERT INTO movimentacoes (produto\_id, tipo, quantidade, data, observacao)

                 VALUES (?, ?, ?, ?, ?)""", (produto\_id, tipo, quantidade, data, observacao))

    # Atualizar quantidade no produto

    prod = get\_produto(produto\_id)

    if not prod:

        return

    current\_qty = prod[4]

    new\_qty = current\_qty + quantidade if tipo == 'entrada' else current\_qty - quantidade

    run\_query("UPDATE produtos SET quantidade=? WHERE id=?", (new\_qty, produto\_id))

def list\_movimentacoes(limit=100):

    return run\_query("""SELECT m.id, p.nome, m.tipo, m.quantidade, m.data, m.observacao

                        FROM movimentacoes m JOIN produtos p ON m.produto\_id = p.id

                        ORDER BY m.data DESC LIMIT ?""", (limit,), fetch=True)

class App:

    def \_\_init\_\_(self, root):

        self.root = root

        root.title("Controle de Estoque - Python")

        root.geometry("900x600")

        self.create\_widgets()

        self.refresh\_produtos()

        self.check\_alerts()

    def excluir\_produto(self):

        sel = self.tree.selection()

        if not sel:

            messagebox.showwarning("Atenção", "Selecione um produto para excluir.")

            return

        item = sel[0]

        pid = int(self.tree.item(item, "values")[0])

        resp = messagebox.askyesno("Excluir Produto", f"Tem certeza que deseja excluir o produto ID {pid}?")

        if not resp:

            return

        delete\_produto(pid)

        messagebox.showinfo("Sucesso", "Produto removido com sucesso.")

        self.refresh\_produtos()

    def create\_widgets(self):

        header = Frame(self.root, bg="#10f21c", height=50)

        header.pack(fill=X)

        Label(

            header,

            text="AVITEC - Controle de Estoque",

            font=("Arial", 20),

            bg="#edda0e"

        ).pack(pady=10)

        frm = Frame(self.root)

        frm.pack(fill=X, padx=8, pady=6)

        Button(frm, text="Nova Categoria", command=self.nova\_categoria).pack(side=LEFT, padx=4)

        Button(frm, text="Novo Produto", command=self.novo\_produto).pack(side=LEFT, padx=4)

        Button(frm, text="Excluir Produto", command=self.excluir\_produto).pack(side=LEFT, padx=4)

        Button(frm, text="Registrar Entrada", command=lambda: self.registrar\_mov('entrada')).pack(side=LEFT, padx=4)

        Button(frm, text="Registrar Saída", command=lambda: self.registrar\_mov('saida')).pack(side=LEFT, padx=4)

        Button(frm, text="Exportar CSV", command=self.export\_csv).pack(side=LEFT, padx=4)

        # Busca

        self.search\_var = StringVar()

        Entry(frm, textvariable=self.search\_var).pack(side=LEFT, padx=6)

        Button(frm, text="Buscar", command=self.on\_search).pack(side=LEFT, padx=4)

        Button(frm, text="Limpar", command=self.on\_clear\_search).pack(side=LEFT, padx=4)

        cols = ("ID","Nome","Categoria","Preço","Qtd","MinQtd")

        self.tree = ttk.Treeview(self.root, columns=cols, show='headings')

        for c in cols:

            self.tree.heading(c, text=c)

            self.tree.column(c, anchor=W, width=100)

        self.tree.pack(fill=BOTH, expand=True, padx=8, pady=6)

        self.tree.bind("<Double-1>", self.on\_edit\_produto)

        # Movimentações

        lbl = Label(self.root, text="Últimas movimentações:")

        lbl.pack(anchor=W, padx=8)

        self.mov\_text = Text(self.root, height=8)

        self.mov\_text.pack(fill=X, padx=8, pady=4)

    def nova\_categoria(self):

        nome = simpledialog.askstring("Nova Categoria", "Nome da categoria:")

        if nome:

            ok = add\_categoria(nome.strip())

            if ok:

                messagebox.showinfo("Sucesso", "Categoria adicionada.")

            else:

                messagebox.showwarning("Erro", "Categoria já existe.")

        self.refresh\_produtos()

    def novo\_produto(self):

        dlg = ProdutoDialog(self.root)

        self.root.wait\_window(dlg.top)

        if dlg.result:

            nome, cat\_id, preco, qtd, minq = dlg.result

            add\_produto(nome, cat\_id, preco, qtd, minq)

            messagebox.showinfo("Sucesso", "Produto cadastrado.")

        self.refresh\_produtos()

    def on\_edit\_produto(self, event):

        sel = self.tree.selection()

        if not sel: return

        pid = int(self.tree.item(sel[0])['values'][0])

        prod = get\_produto(pid)

        dlg = ProdutoDialog(self.root, produto=prod)

        self.root.wait\_window(dlg.top)

        if dlg.result:

            nome, cat\_id, preco, qtd, minq = dlg.result

            update\_produto(pid, nome, cat\_id, preco, qtd, minq)

            messagebox.showinfo("Sucesso", "Produto atualizado.")

        self.refresh\_produtos()

    def registrar\_mov(self, tipo):

        sel = self.tree.selection()

        if not sel:

            messagebox.showwarning("Seleção", "Selecione um produto na lista.")

            return

        pid = int(self.tree.item(sel[0])['values'][0])

        qty = simpledialog.askinteger("Quantidade", f"Quantidade para {tipo}:")

        if qty is None: return

        obs = simpledialog.askstring("Observação (opcional)", "Observação:")

        add\_movimentacao(pid, tipo, qty, obs or "")

        messagebox.showinfo("Registrado", f"{tipo.capitalize()} registrado.")

        self.refresh\_produtos()

    def export\_csv(self):

        path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files","\*.csv")])

        if not path: return

        rows = list\_produtos()

        with open(path, "w", newline='', encoding='utf-8') as f:

            w = csv.writer(f)

            w.writerow(["ID","Nome","Categoria","Preco","Quantidade","MinEstoque"])

            for r in rows:

                w.writerow(r)

        messagebox.showinfo("Exportado", f"Arquivo salvo em: {path}")

    def on\_search(self):

        term = self.search\_var.get().strip()

        self.refresh\_produtos(search=term)

    def on\_clear\_search(self):

        self.search\_var.set("")

        self.refresh\_produtos()

    def refresh\_produtos(self, search=None):

        for i in self.tree.get\_children():

            self.tree.delete(i)

        rows = list\_produtos(search=search)

        for r in rows:

            self.tree.insert("", "end", values=r)

        # atualizar movimentações

        self.mov\_text.delete("1.0", END)

        movs = list\_movimentacoes(20)

        for m in movs:

            self.mov\_text.insert(END, f"{m[4]} | {m[1]} | {m[2]} | {m[3]} | {m[5]}\n")

    def check\_alerts(self):

        rows = list\_produtos()

        alerts = []

        for r in rows:

            pid, nome, cat, preco, qtd, minq = r

            if minq is not None and qtd is not None and qtd <= minq:

                alerts.append(f"{nome} (Qtd: {qtd} / Min: {minq})")

        if alerts:

            messagebox.showwarning("Alerta de Estoque Mínimo", "Produtos com estoque baixo:\n" + "\n".join(alerts))

        self.root.after(10800000, self.check\_alerts)

class ProdutoDialog:

    def \_\_init\_\_(self, parent, produto=None):

        top = self.top = Toplevel(parent)

        top.title("Produto")

        Label(top, text="Nome:").grid(row=0, column=0, sticky=W)

        self.nome = Entry(top, width=40)

        self.nome.grid(row=0, column=1, padx=4, pady=2)

        Label(top, text="Categoria:").grid(row=1, column=0, sticky=W)

        self.cat\_cb = ttk.Combobox(top, values=[c[1] for c in list\_categorias()])

        self.cat\_cb.grid(row=1, column=1, padx=4, pady=2)

        Label(top, text="Preço:").grid(row=2, column=0, sticky=W)

        self.preco = Entry(top); self.preco.grid(row=2, column=1, padx=4, pady=2)

        Label(top, text="Quantidade:").grid(row=3, column=0, sticky=W)

        self.qtd = Entry(top); self.qtd.grid(row=3, column=1, padx=4, pady=2)

        Label(top, text="Min. Estoque:").grid(row=4, column=0, sticky=W)

        self.minq = Entry(top); self.minq.grid(row=4, column=1, padx=4, pady=2)

        btn = Button(top, text="Salvar", command=self.on\_save)

        btn.grid(row=5, column=0, columnspan=2, pady=6)

        self.result = None

        if produto:

            pid, nome, cat\_id, preco, qtd, minq = produto

            self.nome.insert(0, nome)

            cats = list\_categorias()

            cat\_names = [c[1] for c in cats]

            if cat\_id:

                for i,c in enumerate(cats):

                    if c[0] == cat\_id:

                        self.cat\_cb.current(i)

                        break

            self.preco.insert(0, str(preco))

            self.qtd.insert(0, str(qtd))

            self.minq.insert(0, str(minq))

    def on\_save(self):

        nome = self.nome.get().strip()

        cat\_name = self.cat\_cb.get().strip()

        cat\_id = None

        for c in list\_categorias():

            if c[1] == cat\_name:

                cat\_id = c[0]; break

        try:

            preco = float(self.preco.get() or 0)

        except:

            preco = 0.0

        try:

            qtd = int(self.qtd.get() or 0)

        except:

            qtd = 0

        try:

            minq = int(self.minq.get() or 0)

        except:

            minq = 0

        if not nome:

            messagebox.showwarning("Validação", "Informe o nome do produto.")

            return

        self.result = (nome, cat\_id, preco, qtd, minq)

        self.top.destroy()

def main():

    ensure\_db()

    root = Tk()

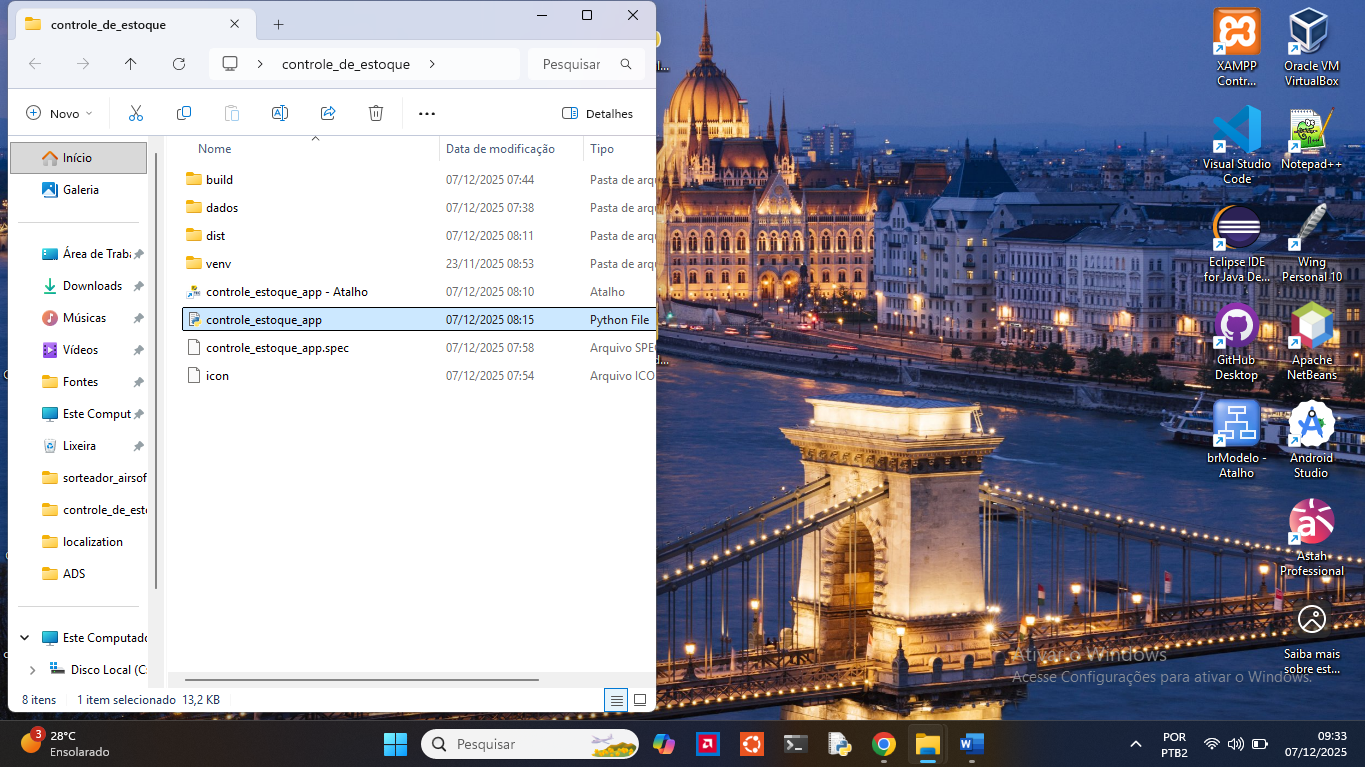
    app = App(root)

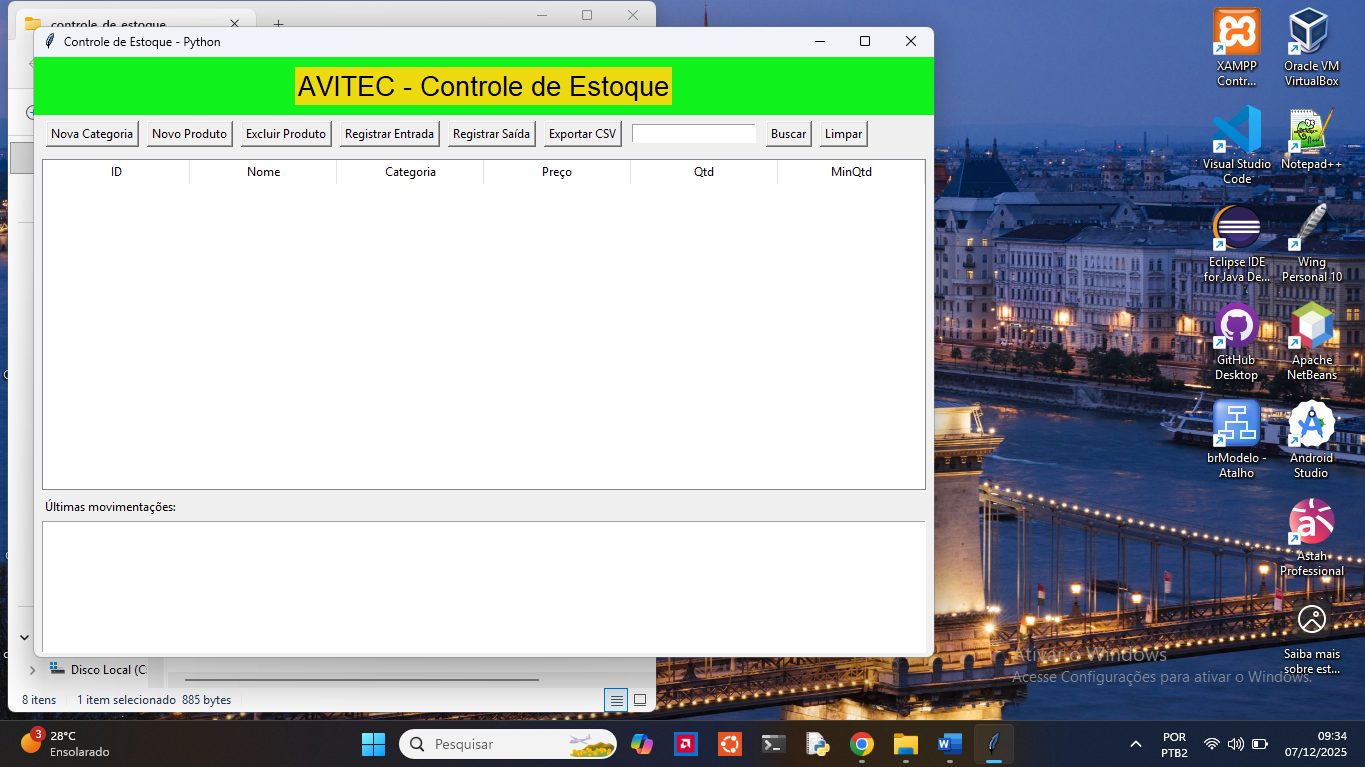
    root.mainloop()

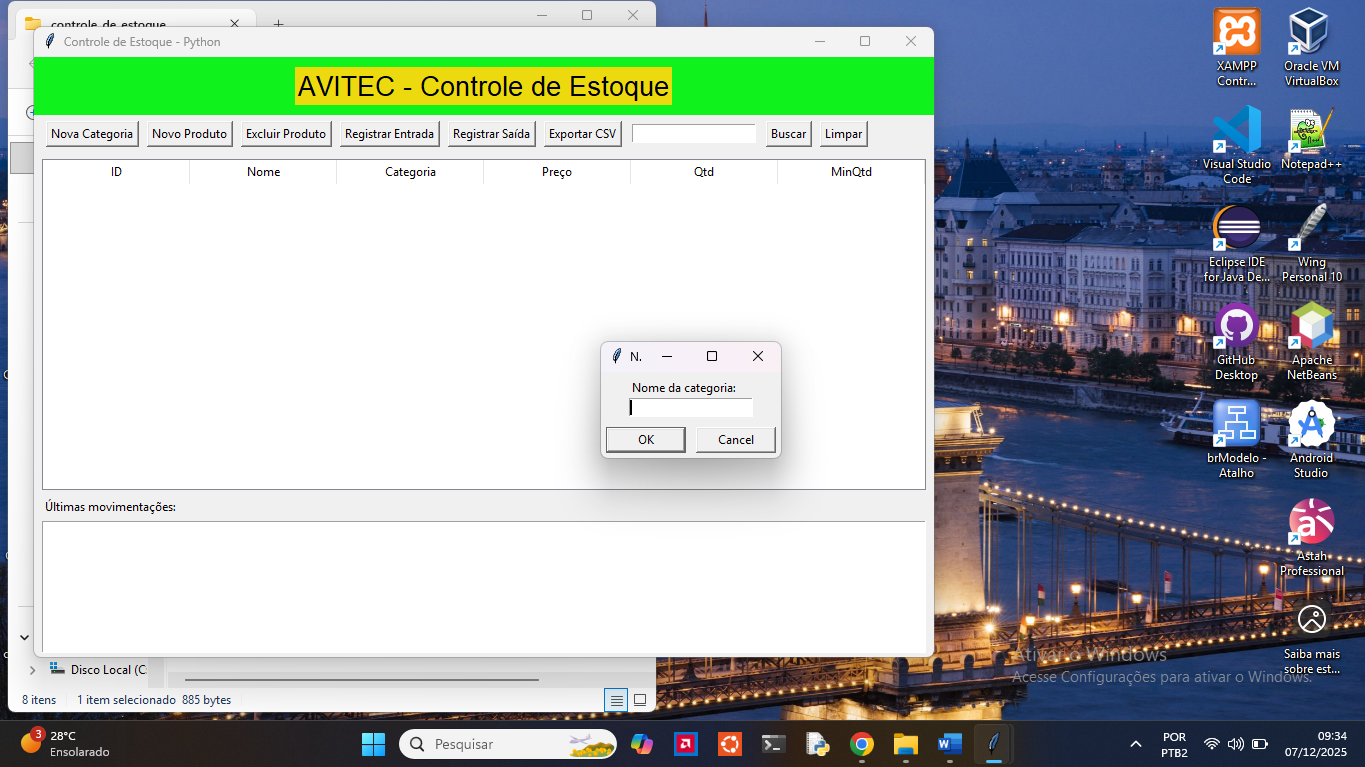
if \_\_name\_\_ == "\_\_main\_\_":

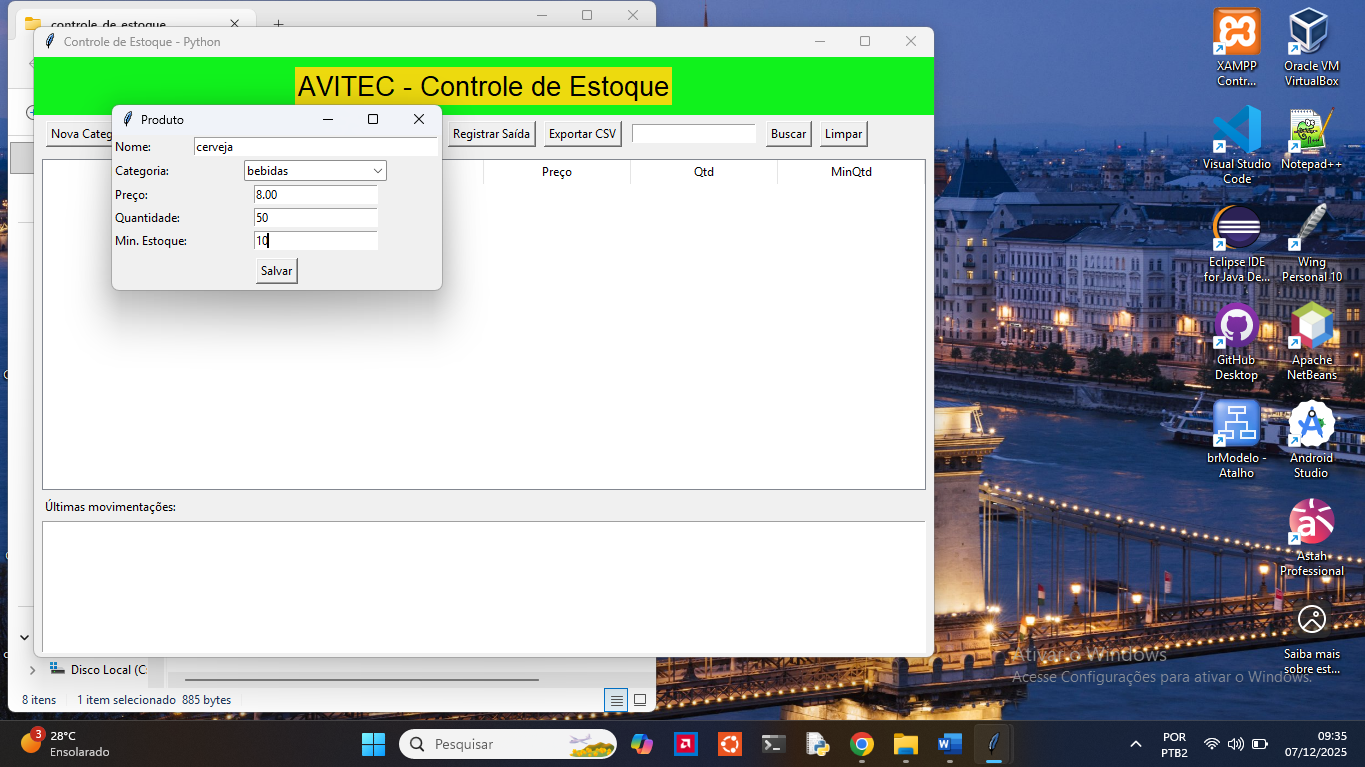
    main()

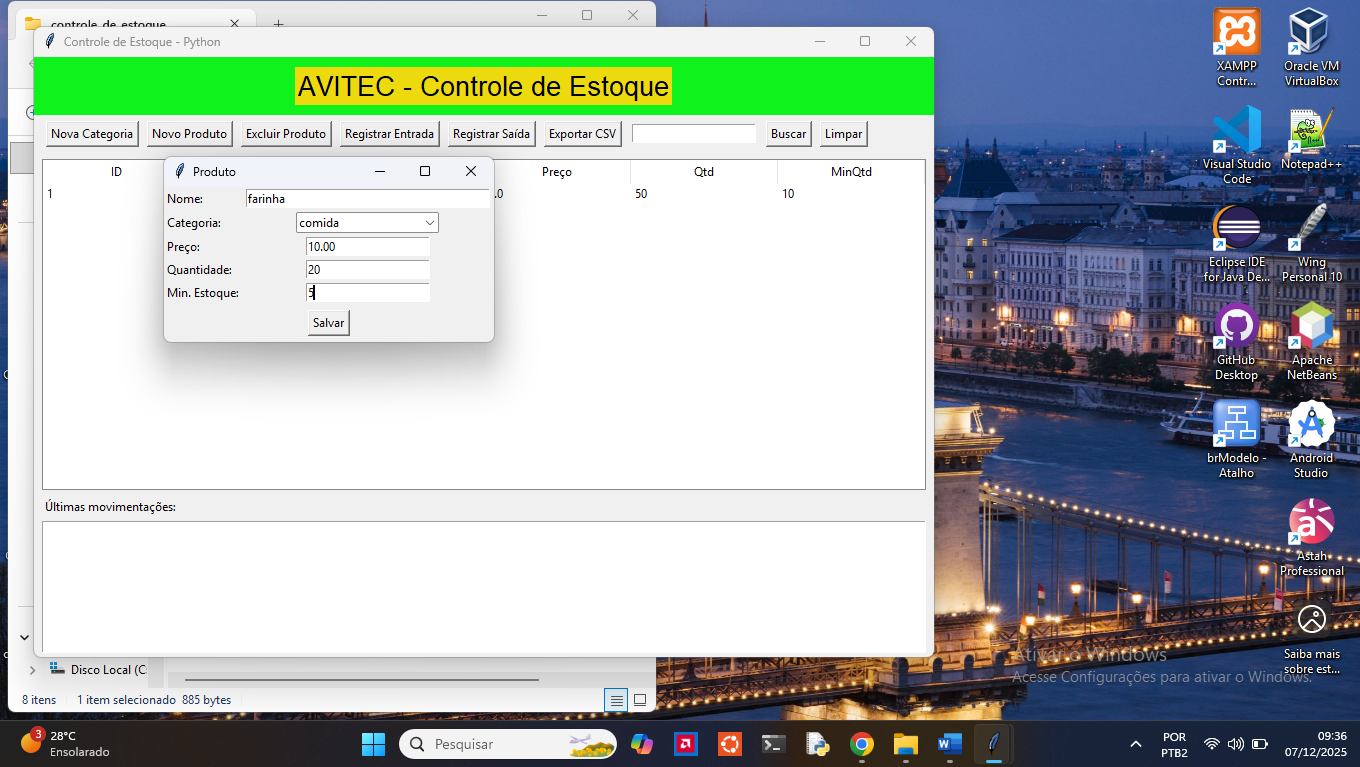
### Prints/projeto funcionalidades principais











### 6 Conclusão

A partir do exposto concluo que o objetivo principal (criação de um sistema de controle de estoque seguro e confiável) foi atendido com solução adequada para pequenos estabelecimentos, baixo custo de implantação e manutenção.

Considerando o projeto finalizado e concluído pretendo colocar em produção para testes, validação e implementação na empresa onde trabalho (Avitecsui) e estudar futuras implementações no sistema, como: Implementação de login com usuário e senha e implementação web para que o sistema possa ser usado entre filiais. Concluo então o projeto e iniciando a fase de implementação.