**# Adicione esta variável global no início do código, com as outras variáveis**

saida\_produtos = [] # Lista para armazenar os produtos e quantidades de saída

**# Modifique a função adicionar\_item\_saida para usar o vetor**

def adicionar\_item\_saida():

global saida\_produtos

nome\_produto = entrada\_nome\_qtde.get().strip()

qtde\_retirada = entrada\_qtde\_retirada.get().strip()

if not nome\_produto or not qtde\_retirada:

showinfo("Aviso", "Preencha todos os campos!")

return

try:

qtde = int(qtde\_retirada)

if qtde <= 0:

showinfo("Aviso", "A quantidade deve ser maior que zero!")

return

except ValueError:

showinfo("Aviso", "Quantidade inválida!")

return

**# Verifica se o produto já está na lista**

encontrado = False

for i, item in enumerate(saida\_produtos):

if item['nome'] == nome\_produto:

saida\_produtos[i]['quantidade'] += qtde

encontrado = True

break

if not encontrado:

# Adiciona novo item ao vetor

saida\_produtos.append({

'nome': nome\_produto,

'quantidade': qtde

})

**# Atualiza a exibição na interface**

atualizar\_lista\_saida()

**# Limpa o campo de quantidade**

entrada\_qtde\_retirada.delete(0, tk.END)

**# Função para atualizar a exibição na interface de saída**

def atualizar\_lista\_saida():

# Limpa todos os widgets do frame\_scroll4

for widget in frame\_scroll4.winfo\_children():

widget.destroy()

**# Adiciona cada item do vetor à interface**

for i, item in enumerate(saida\_produtos):

label = customtkinter.CTkLabel(frame\_scroll4, text=f"{item['nome']} - -{item['quantidade']} unidades")

label.grid(row=i, column=0, pady=5, padx=0, sticky="w")

botao\_remover = customtkinter.CTkButton(

frame\_scroll4,

text="🗑️",

width=5,

command=lambda idx=i: remover\_item\_saida(idx)

)

botao\_remover.grid(row=i, column=1, pady=5, padx=5)

# Modifique a função remover\_item\_saida para trabalhar com o vetor

def remover\_item\_saida(indice):

global saida\_produtos

if 0 <= indice < len(saida\_produtos):

saida\_produtos.pop(indice)

atualizar\_lista\_saida() **LINHA 338**

# Atualize a função do botão salvar na tela de saída

def salvar\_saida():

global saida\_produtos

from datetime import datetime

if not saida\_produtos:

showinfo("Aviso", "Nenhum produto para salvar!")

return

try:

conexao = sqlite3.connect("dados.db")

terminal\_sql = conexao.cursor()

data\_hora = datetime.now().strftime("%d/%m/%Y %H:%M")

for produto in saida\_produtos:

# Verifica se o produto existe

terminal\_sql.execute("SELECT nome FROM itens WHERE nome = ?", (produto['nome'],))

if not terminal\_sql.fetchone():

showinfo("Erro", f"Produto '{produto['nome']}' não cadastrado!")

continue  **linha 359**

# Insere no banco de dados (você precisará criar uma tabela de saídas)

terminal\_sql.execute(

"INSERT INTO saidas (nome\_produto, quantidade, data\_hora) VALUES (?, ?, ?)",

(produto['nome'], produto['quantidade'], data\_hora)

)

conexao.commit()

showinfo("Sucesso", "Saída de produtos salva com sucesso!")  **Linha 368**

# Limpa o vetor e a interface

saida\_produtos = []

atualizar\_lista\_saida()

entrada\_nome\_qtde.delete(0, tk.END)

entrada\_qtde\_retirada.delete(0, tk.END)

except Exception as e:

conexao.rollback()

showinfo("Erro", f"Ocorreu um erro ao salvar: {str(e)}")

finally:

conexao.close() **linha 381**

# Atualize o botão salvar na tela de saída para usar a nova função

botao\_salvar.configure(command=salvar\_saida)

# Adicione esta função para carregar as saídas no relatório

def carregar\_saidas():

conexao = sqlite3.connect("dados.db")

terminal\_sql = conexao.cursor() **linha 377**

# Limpa a tabela

for item in tree\_saida.get\_children():

tree\_saida.delete(item**) linha 390**

# Busca as saídas no banco de dados

terminal\_sql.execute("""

SELECT nome\_produto, quantidade, data\_hora

FROM saidas

ORDER BY data\_hora DESC

""")

saidas = terminal\_sql.fetchall() **linha 398**

# Preenche a tabela com os dados

for saida in saidas:

tree\_saida.insert('', tk.END, values=saida)

conexao.close() **linha 401**

# Atualize a função abrir\_frame\_relatorio\_saida para carregar os dados

def abrir\_frame\_relatorio\_saida():

# Fecha frame

frame\_cadastrar.grid\_forget()

frame\_editar.grid\_forget()

frame\_saida.grid\_forget()

frame\_entrada.grid\_forget()

frame\_relatorio.grid\_forget()

frame\_relatorio\_entrada.grid\_forget()

# abre frame

frame\_relatorio\_saida.grid\_propagate(False)

frame\_relatorio\_saida.grid(row=0, column=1, padx=5, pady=5)

carregar\_saidas() # Carrega os dados ao abrir o frame **linha 552**

# Atualize a função criar\_banco para incluir a tabela de saídas

def criar\_banco():

conexao = sqlite3.connect("dados.db")

terminal\_sql = conexao.cursor()

terminal\_sql.execute("CREATE TABLE IF NOT EXISTS itens (nome text, preco decimal, descricao text)")

terminal\_sql.execute("""CREATE TABLE IF NOT EXISTS saidas (

id INTEGER PRIMARY KEY AUTOINCREMENT,

nome\_produto text,

quantidade integer,

data\_hora text,

FOREIGN KEY(nome\_produto) REFERENCES itens(nome)

)""")

conexao.commit()

conexao.close() **Linha 45**

**nome\_produto = entrada\_nome\_quantidade.get().strip()  
qtde\_adicionada = entrada\_qtde\_adicionada.get().strip()**