

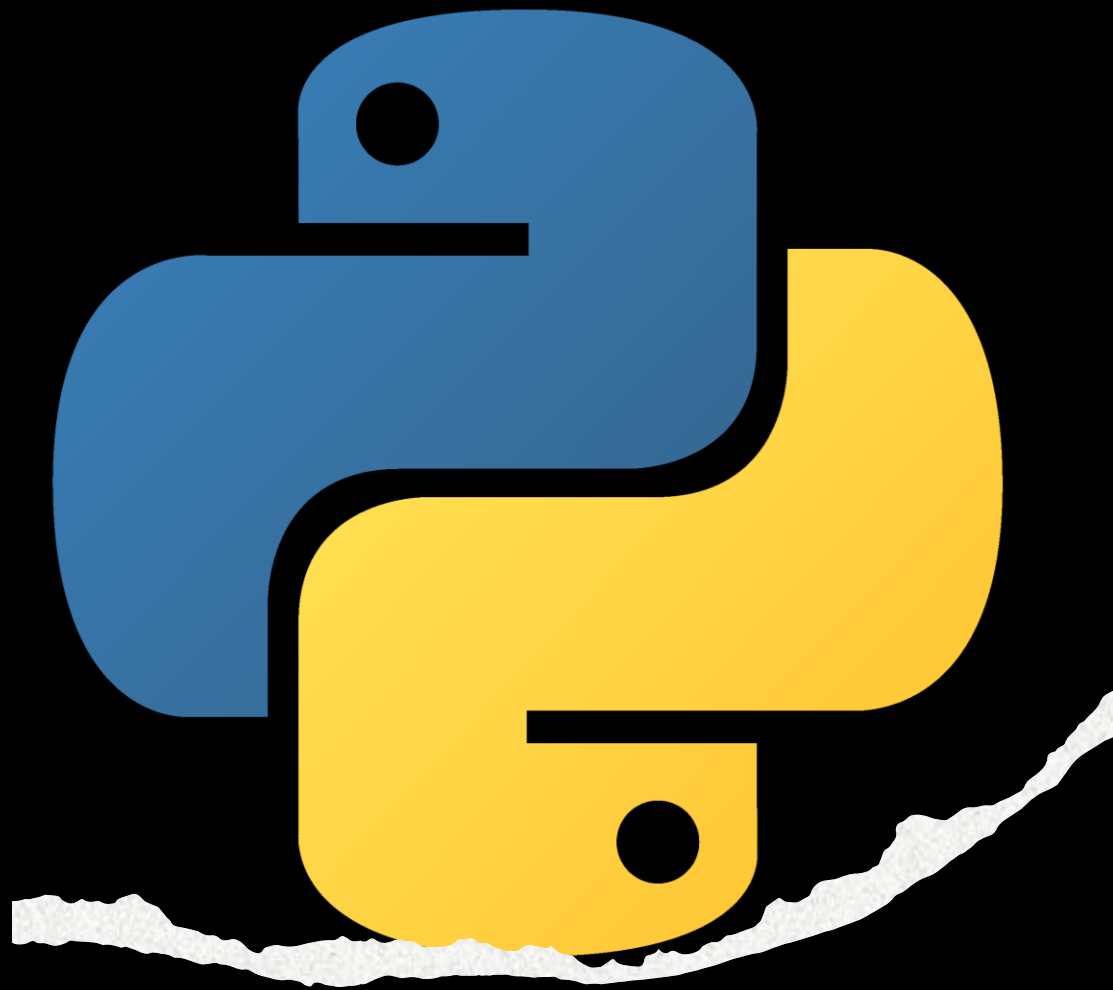
python

Casa do Norte

Sistema de Cadastro em Python

Atenção!

- Para que tudo funcione corretamente, lembre-se de deixar o código indentado de forma correta. Isso é prioridade na linguagem Python.





Briefing

Um restaurante/fornecedor de comidas típicas nordestinas enfrenta desafios importantes na gestão de estoque devido à falta de um sistema adequado.

O controle de estoque dos pratos e ingredientes tem sido um problema grande.

Atualmente, o registro de entradas e saídas é feito manualmente, tornando-se confuso e desorganizado.

Briefing

Com a implementação de um sistema, buscase ter um **controle preciso sobre o que entra e sai do estoque**, além de saber quais comidas e ingredientes estão disponíveis.

Também é importante **receber alertas** quando os níveis de **estoque** estiverem **baixos**.

É fundamental que as características das comidas sejam devidamente registradas, incluindo detalhes como nome, descrição, categoria, origem, ingredientes, porção e calorias, por exemplo: Baião de Dois – arroz com feijão verde, queijo coalho e carne seca.





Briefing

O sistema deve permitir o registro de campos específicos das comidas, como **nome, descrição, categoria, origem, ingredientes, porção, calorias, quantidade disponível e estoque mínimo configurável para alertas.**

Os alertas de estoque precisam ser **configuráveis** para cada comida cadastrada.



1ª Parte: estrutura

- Fazer o download da pasta com a estrutura
- Casa_norte-aluno
 - comidas.py
 - db.py
 - estoque.py
 - login.py
 - main.py
 - utils.py



1ª parte: estrutura

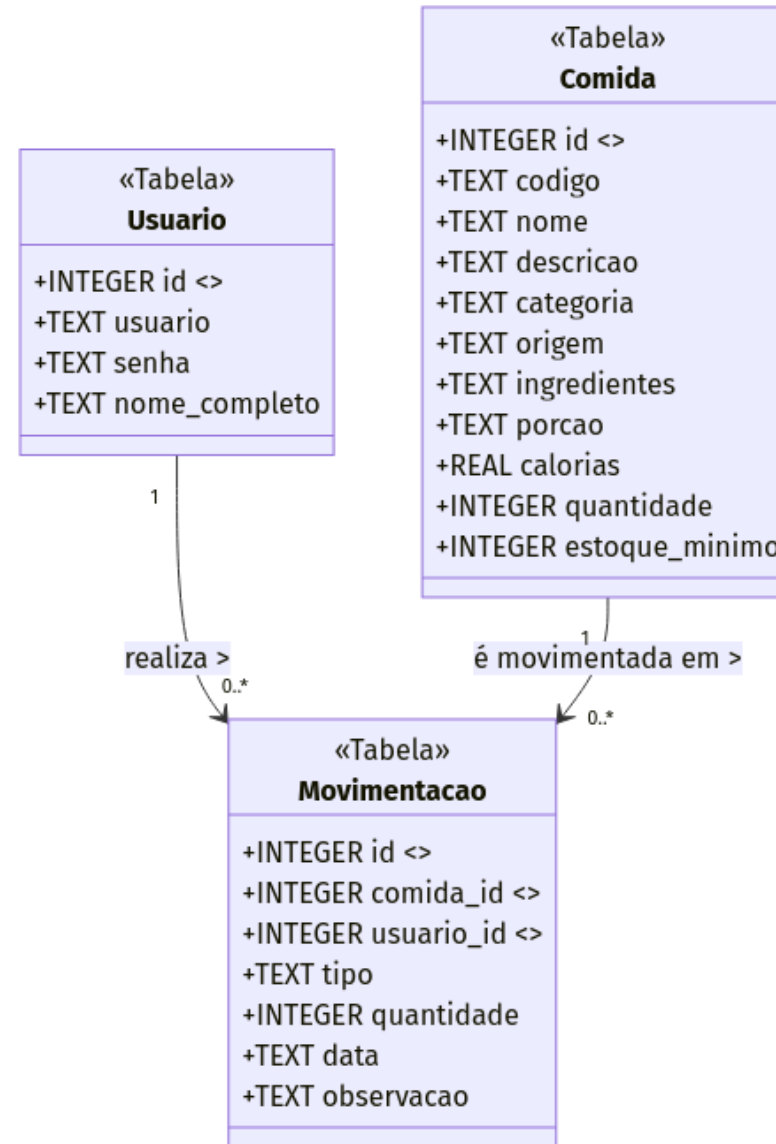


2ª Parte: Banco de dados

- Desenvolver todo o SCRIPT do banco de dados
- Desenvolver o diagrama UML



2ª parte:
Desenvolver o
SCRIPT do
banco de
dados



2ª parte: Desenvolver o SCRIPT do banco de dados

- Na pasta criar o arquivo “db_init.sql”

```
-- Ativa a verificação de integridade referencial (chaves estrangeiras).
```

```
PRAGMA foreign_keys = ON;
```

```
-- Tabela de usuários
```

```
CREATE TABLE IF NOT EXISTS usuarios (
```

```
    id INTEGER PRIMARY KEY AUTOINCREMENT, -- Identificador único gerado automaticamente
```

```
    usuario TEXT UNIQUE, -- Nome de usuário único (login)
```

```
    senha TEXT, -- Senha do usuário (em texto simples aqui)
```

```
    nome_completo TEXT -- Nome completo do usuário
```

```
);
```

2ª parte: Desenvolver o SCRIPT do banco de dados

-- Tabela de comidas nordestinas

```
CREATE TABLE IF NOT EXISTS comidas (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    codigo TEXT UNIQUE,  
    nome TEXT,  
    descricao TEXT,  
    categoria TEXT,  
    origem TEXT,  
    ingredientes TEXT,  
    porcao TEXT,  
    calorias REAL,  
    quantidade INTEGER DEFAULT 0,  
    estoque_minimo INTEGER DEFAULT 0  
);
```

```
-- Identificador único da comida  
-- Código único do prato/comida  
-- Nome da comida  
-- Descrição detalhada  
-- Categoria (ex: prato principal, salgado)  
-- Origem regional da comida  
-- Ingredientes usados  
-- Porção sugerida (ex: 300g)  
-- Quantidade de calorias  
-- Quantidade atual em estoque (padrão 0)  
-- Quantidade mínima para alerta (padrão 0)
```


2ª parte: Desenvolver o SCRIPT do banco de dados

```
-- Tabela de movimentações (histórico de estoque)
CREATE TABLE IF NOT EXISTS movimentacoes (
    id INTEGER PRIMARY KEY AUTOINCREMENT, -- Identificador único da movimentação
    comida_id INTEGER, -- ID da comida movimentada
    usuario_id INTEGER, -- ID do usuário que realizou movimentação
    tipo TEXT, -- Tipo de movimentação ("entrada" ou "saída")
    quantidade INTEGER, -- Quantidade movimentada
    data TEXT, -- Data da movimentação (formato texto)
    observacao TEXT, -- Observação adicional

    FOREIGN KEY(comida_id) REFERENCES comidas(id),
    FOREIGN KEY(usuario_id) REFERENCES usuarios(id)
);
```



3ª Parte: Testar armazenamento de dados

- Inserir os dados necessários para popular o banco de dados
- Inserir no mínimo 3 entradas para cada tabela

3ª parte: Inserir os dados para popular o bd


```
-- Inserção de usuários iniciais (sistema terá esses usuários cadastrados)
INSERT OR IGNORE INTO usuarios (id, usuario, senha, nome_completo) VALUES
(1, 'vendedor1', '123', 'Helena Silva'),
(2, 'vendedor2', '123', 'Cecilia Lima'),
(3, 'vendedor3', '123', 'Ravi Santos');
```


3ª parte: Inserir os dados para popular o bd

```
-- Inserção de comidas iniciais no sistema
INSERT OR IGNORE INTO comidas (id, codigo, nome, descricao, categoria, origem,
ingredientes, porcao, calorias, quantidade, estoque_minimo) VALUES
(1, 'C001', 'Baião de Dois', 'Arroz com feijão verde, queijo coalho e carne seca.',
'Prato Principal', 'Ceará', 'Arroz, feijão, queijo coalho, carne seca', '300g', 480,
15, 3),
(2, 'C002', 'Carne de Sol com Macaxeira', 'Carne de sol acebolada servida com macaxeira
frita ou cozida.', 'Prato Principal', 'Nordeste (Geral)', 'Carne de sol, macaxeira,
manteiga de garrafa, cebola roxa', '450g', 700, 25, 8),
(3, 'C003', 'Vatapá', 'Creme de pão com camarão seco, leite de coco e azeite de
dendê.', 'Acompanhamento', 'Bahia', 'Pão, camarão seco, leite de coco, azeite de dendê,
amendoim', '250g', 450, 18, 6);
```

3ª parte: Inserir os dados para popular o bd

```
-- Inserção de movimentações iniciais no estoque
INSERT OR IGNORE INTO movimentacoes (id, comida_id, usuario_id, tipo, quantidade,
data, observacao) VALUES
(1, 1, 1, 'entrada', 5, '2025-09-01', 'Reposição estoque mensal Baião de Dois'),
(2, 2, 2, 'saída', 2, '2025-09-05', 'Venda para evento do dia 10/10'),
(3, 3, 3, 'entrada', 10, '2025-09-10', 'Novo lote de camarão');
```



4ª Parte: Desenvolver o arquivo de conexão

- Inserir os dados necessários para popular o banco de dados
- Inserir no mínimo 3 entradas para cada tabela



4ª parte: arquivo de conexão com o bd

Necessário importar as bibliotecas necessárias para interagir com o banco de dados e o sistema de arquivos

1º passo: Importar o módulo 'sqlite3', que é a biblioteca padrão do Python para trabalhar com bancos de dados SQLite.

```
import sqlite3
```

2º passo: Importar o módulo 'os', que permite interagir com o sistema operacional, como verificar se um arquivo existe.

```
import os
```

3º passo: Definir o nome do arquivo que será usado para o banco de dados.

```
DBFILENAME = "comidasdb.sqlite"
```

4ª parte: arquivo de conexão com o bd

```
# --- Função para Obter Conexão ---  
# Toda vez que o programa precisar ler ou escrever dados, ele chamará esta função.  
  
def getconnection():  
    """Abre uma conexão com o banco de dados SQLite."""  
  
    # Estabelece comunicação com o arquivo do banco de dados  
    conn = sqlite3.connect(DBFILENAME)  
    conn.row_factory = sqlite3.Row  
    # Executa um comando PRAGMA no SQLite para garantir que as regras de chave  
    estrangeira sejam sempre aplicadas.  
    # Isso é crucial para manter a integridade dos dados (ex: não permitir uma  
    movimentação para uma comida que não existe).  
    conn.execute("PRAGMA foreign_keys = ON")  
    return conn
```

4ª parte: arquivo de conexão com o bd

```
# --- Função para Garantir a Existência do Banco ---
# Esta função é chamada uma única vez, no início da aplicação.
# Ela verifica se o banco de dados já foi criado. Se não, ela o cria.

def ensuredb():

    # Verifica se o arquivo do banco de dados já existe
    if not os.path.exists(DBFILENAME):
        scriptpath = os.path.join(os.path.dirname(__file__), "db_init.sql")
        if os.path.exists(scriptpath):
            with getconnection() as conn:
                with open(scriptpath, "r", encoding="utf-8") as f:
                    conn.executescript(f.read())
        else:
            # Gera erro caso o arquivo db_init.sql não esteja disponível
            raise FileNotFoundError("db_init.sql não encontrado. Coloque db_init.sql
na mesma pasta.")
```



5ª Parte: Desenvolver o arquivo de funcionalidades

- Desenvolver o arquivo que tem funções extras como criar IDs personalizados em ordem numérica para os produtos que forem cadastrados.



5ª parte: arquivo de funcionalidades

- No arquivo 'utils.py'

```
# --- Função para Gerar Código de Produto ---  
# Esta função foi projetada para criar um código único para cada nova comida (ex:  
# C001, C002).  
  
def gerarcodigo():  
    # Usa o banco de dados para determinar o último id cadastrado  
    with getconnection() as conn:  
        # Busca o id mais alto da tabela de comidas  
        cur = conn.execute("SELECT id FROM comidas ORDER BY id DESC LIMIT 1")  
        row = cur.fetchone()  
        # Se não houver cadastros, retorna o primeiro código padrão  
        if not row:  
            return "C001"  
        # Gera o próximo código baseado no maior id atual  
        # f"C{...:03d}" formata o número para ter sempre 3 dígitos, preenchendo com  
        # zeros à esquerda (ex: 5 -> 005).  
        return f"C{row['id']+1:03d}"
```



6ª Parte: Desenvolver a página de login

- Desenvolver layout da tela de login para input de usuário e senha



6ª parte: página de login

```
# --- Definição da Função Principal ---  
# Esta função é responsável por desenhar e controlar toda a tela de login.  
# Ela recebe 'app' como argumento, que é a janela principal da nossa  
# aplicação.  
  
def showlogin(app):  
  
    # Limpa a tela para exibir apenas o login  
    for w in app.winfo_children():  
        w.destroy()  
  
    # Centraliza a janela principal na tela, largura=400px e altura=250px  
    centralizarjanela(app, 400, 250)  
  
    # Cria um frame (área interna) com preenchimento, que organiza os widgets de login  
    frm = ttk.Frame(app, padding=20)  
    frm.pack(expand=True)
```

6ª parte: página de login

```
# ainda na mesma função
# Cria um título "Login" centralizado, usando fonte maior
ttk.Label(frm, text="Login", font=("TkDefaultFont", 16)).grid(column=0,
row=0, columnspan=2, pady=10)

# Cria o texto "Usuário" à direita na grid (linha 1, coluna 0)
ttk.Label(frm, text="Usuário").grid(column=0, row=1, sticky="e")

# Cria um campo de entrada de texto para digitar o nome de usuário
userent = ttk.Entry(frm, width=25)
userent.grid(column=1, row=1)

# Cria e posiciona o rótulo "Senha" à direita na grid (linha 2, coluna 0)
ttk.Label(frm, text="Senha").grid(column=0, row=2, sticky="e")

# Cria campo de entrada para senha, ocultando os caracteres digitados
pwdent = ttk.Entry(frm, show="*", width=25)
pwdent.grid(column=1, row=2)
```


6ª parte: página de login

```
# ainda na mesma função
# Função interna para realizar login ao clicar no botão "Entrar"
def attempt_login():
    username = userent.get().strip() # Obtém o nome de usuário removendo espaços extras
    password = pwdent.get().strip()  # Obtém a senha e remove espaços extras

    # Verifica se ambos os campos foram preenchidos
    if not username or not password:
        # Alerta se faltou algum campo
        messagebox.showwarning("Falha", "Preencha usuário e senha.")
        return

    # Executa a consulta no banco de dados para verificar usuário e senha
    with getconnection() as conn:
        cur = conn.execute(
            "SELECT * FROM usuarios WHERE usuario=? AND senha=?",
            (username, password)
        )
        row = cur.fetchone() # Obtém o primeiro resultado (se existir)
```

6ª parte: página de login

```
# ainda na mesma função
```

```
# Se encontrou usuário válido, armazena dados em app e direciona para tela principal
```

```
if row:
```

```
    app.currentuser = dict(row)
```

```
    app.showmain()
```

```
else:
```

```
    messagebox.showerror("Falha", "Usuário ou senha inválidos.")
```

```
# Cria o botão "Entrar" que chama attempt_login quando pressionado
```

```
ttk.Button(frm, text="Entrar", command=attempt_login).grid(column=0, row=3, columnspan=2,  
pady=10)
```

Atenção na indentação!



7ª Parte: Desenvolver a página principal

- Desenvolver layout da tela principal onde aparece o nome do usuário atual e o botão para sair do sistema
- No primeiro acesso a tela deve redirecionar para a tela de login de usuário



7ª parte: página principal

Uma 'class' é como uma base fundamental para construir um objeto.
Nossa classe 'App' é a base para construir a janela principal do nosso programa.
Ela 'herda' de 'tk.Tk', o que significa que ela já nasce sabendo tudo que uma janela normal do Tkinter sabe fazer.

```
class App(tk.Tk):  
    # O método construtor '__init__' é o que acontece assim que a janela é criada.  
    def __init__(self):  
        # Inicializa objeto Tk (janela principal).  
        super().__init__()  
        self.title("Controle de Estoque: Comidas Nordestinas")  
        ensuredb()  
        self.currentuser = None  
        centralizarjanela(self, 400, 250)  
        showlogin(self)
```

7ª parte: página principal

```
# Este método é responsável por mostrar o menu principal depois
que o login dá certo.
def showmain(self):
    # Antes de desenhar a tela nova, precisamos limpar a antiga (a de login).
    for w in self.wininfo_children():
        w.destroy()

    # Centraliza e expande janela para modo principal.
    centralizarjanela(self, 1000, 550)

    # Frame superior com dados do usuário e botões de navegação.
    top = ttk.Frame(self, padding=2)
    top.pack(fill="x")

    # Exibe quem está logado (usa 'nome_completo' do banco).
    ttk.Label(top, text=f" Usuário
logado:{self.currentuser.get('nome_completo', '')}").pack(side="left")
```


7ª parte: página principal

```
# Ainda no mesmo método...
```

```
# Botão para deslogar do sistema
```

```
ttk.Button(top, text="Sair", command=lambda: showlogin(self)).pack(side="right")
```

```
# Botão para acessar cadastro de comidas.
```

```
ttk.Button(top, text="Cadastro de Comidas", command=lambda:  
showcomidas(self)).pack(side="right", padx=6)
```

```
# Botão para acessar tela de gestão de estoque.
```

```
ttk.Button(top, text="Gestão de Estoque", command=lambda:  
showgestaoestoque(self)).pack(side="right", padx=6)
```

7ª parte: página principal

```
# Ainda no mesmo método...

# Frame central para mensagem de boas-vindas e instruções.
center = ttk.Frame(self, padding=40)
center.pack(expand=True)
msg = (
    f"Olá, {self.currentuser.get('nome_completo', '')}}!\n"
    "Bem-vindo ao sistema de controle de estoque de Comidas Nordestinas.\n\n"
    "Aqui você pode:\n"
    " ** Cadastrar novos pratos e comidas típicas\n"
    " ** Consultar e editar o estoque\n"
    " ** Registrar entradas e saídas"
    "\n\nEscolha um botão na barra de opções acima para começar."
)
tk.Label(center, text=msg, font=("TkDefaultFont", 18), justify="left").pack()
```




8ª Parte: Testar telas desenvolvidas

- Inserir os dados necessários para testar todas as telas desenvolvidas
- Inserir no mínimo 2 entradas para cada tabela
- Realizar no mínimo 1 entrada e 1 saída de produto



Telas



A screenshot of a web application window titled "Controle de Estoque de Comidas Nordesti...". The window contains a login form with the word "Login" in a large, brown font. Below the title, there are two input fields: "Usuário" and "Senha". A blue button labeled "Entrar" is positioned below the "Senha" field.

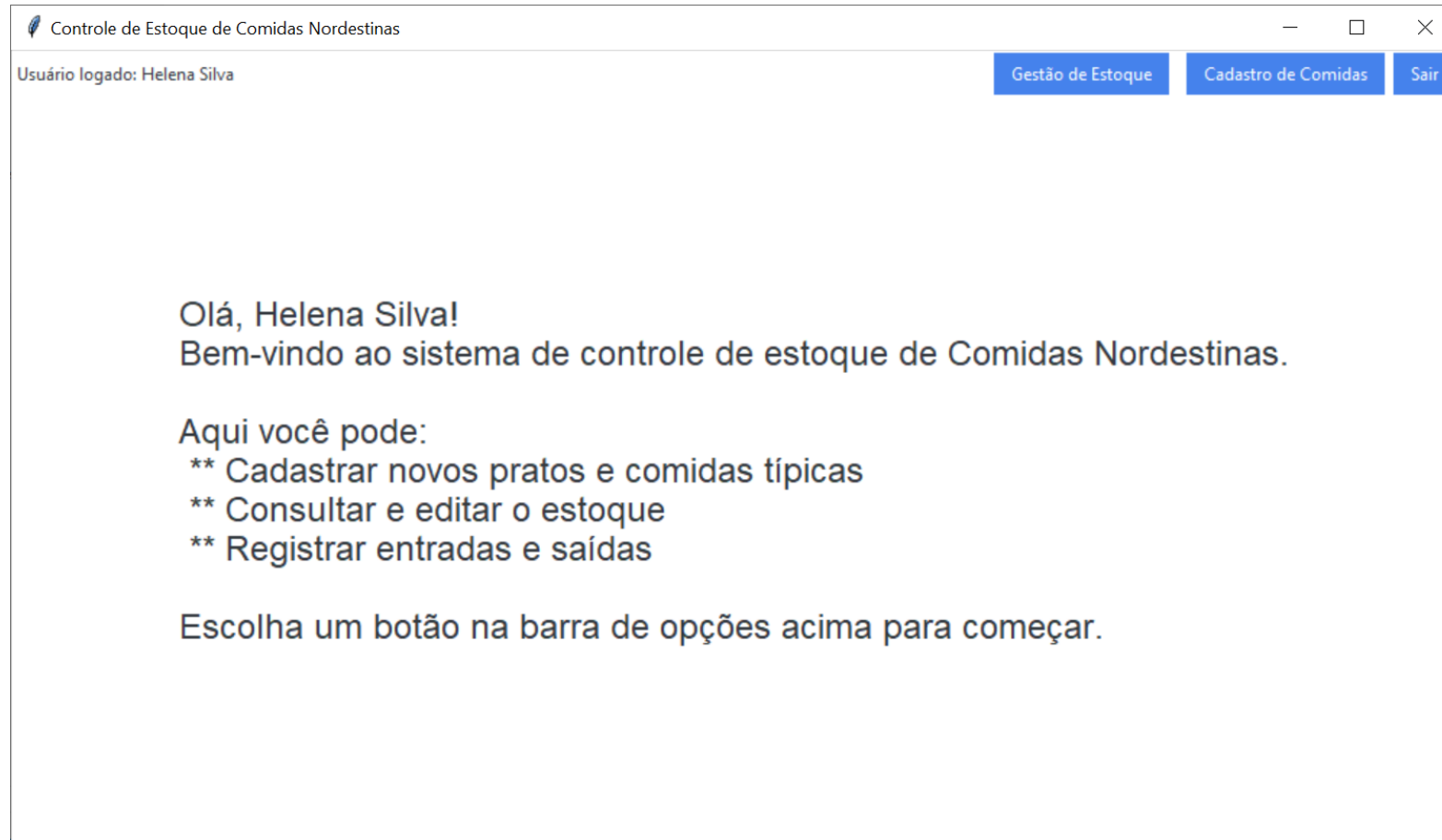
Controle de Estoque de Comidas Nordesti...

Login

Usuário

Senha

Telas



Telas

Controle de Estoque de Comidas Nordestinas

Voltar

ID	Nome	Quantidade	Estoque Mínimo
3	Acarajé	20	5
1	Baião de Dois	15	3
2	Sarapatel	10	3

Registrar Movimentação Ver Histórico

Telas

Controlador de Estoque de Alimentos Nordeste

Voltar

Nova Comida

Buscar

Buscar


Limpar

ID	Nome	Categoria	Origem	Porção	Calorias	Quantidade	Estoque Mínimo
3	Acarajé	Salgado	Bahia	150g	350.0	20	5
1	Baião de Dois	Prato Principal	Ceará	300g	480.0	15	3
2	Sarapatel	Prato Principal	Pernambuco	350g	600.0	10	3

Editar

Excluir

Telas

 Comida Nordestina

Nome

Descrição

Categoria

Origem

Ingredientes

Porção

Calorias

Quantidade

Estoque Mínimo

Salvar