Alunos: Nicolas Santana, Henrique White, Daniel Martins.

Sesi Senai - Curso Técnico Desenvolvimento de Sistemas

Documentação sobre uso do Software ‘Caldenorte’

Caldenorte

Caldenorte é um sistema de gestão de vendas e fornecimento de materiais, projetado para facilitar o processo de venda, controle de estoque, emissão de pedidos e gerenciamento de fornecimento de produtos. O Software permite que os usuários registrem vendas, acompanhem os estoques de materiais, definam prazos de entrega, atribuam prioridades aos pedidos e monitorem o progresso de cada etapa do processo de fornecimento.
A interface do sistema é simples e intuitiva, com funcionalidades como filtragem por status do pedido, categoria de material e data de entrega, oferecendo uma experiência de usuário eficiente e organizada.
O principal objetivo da Caldenorte é otimizar a gestão de vendas e fornecimento de materiais, proporcionando uma plataforma online acessível em qualquer dispositivo com acesso à internet. O sistema auxilia empresas a gerenciar com mais eficiência o ciclo de vendas e garantir o fornecimento de materiais dentro dos prazos estabelecidos, resultando em uma maior produtividade e satisfação do cliente.

Visão Geral do Sistema

- *Requisitos Funcionais:*

RF1 - O Sistema deve permitir o cadastro do tipo de usuário.
RF2 - O Cliente poderá ver os Produtos e alterar suas informações
RF3 - O Funcionário poderá fazer o Cadastro, alteração, ou exclusão dos Produtos, Transportadora, Fornecedor e Funcionário.
RF4 - O Cliente poderá fazer o Cadastro.
RF5 - Para realizar o cadastro, todos os dados têm que estar preenchidos.
RF6 - Para realizar o Pedido, todos os dados têm que estar preenchidos.

- *Requisitos Não Funcionais:*

RNF1 - A interface deve ser acessível e entendível a todos os usuários.
RNF2 - O Software deve ser capaz de rodar nos principais sistemas operacionais (Windows, macOS e Linux).
RNF3 - O sistema deve ser altamente protegido contra ataques.

- *Requisitos do painel administrativo:*

- O Sistema vai mostrar uma tela de Login.
- Se a pessoa não tiver conta, ela vai para uma tela de cadastro.

Menu Cliente:

- O Cliente poderá ver e acessar os produtos listados.
- O Cliente poderá alterar seus dados cadastrados.

Menu ADM:

- O ADM poderá criar o pedido do cliente.
- O ADM poderá fazer o cadastro de produtos.
- O ADM pode cadastrar um Cliente.
- O ADM pode fazer o cadastro de Transportadora.
- O ADM poderá fazer o cadastro de fornecedores.
- Cadastro de um novo funcionário será possível na aba ‘Funcionários’.

Tecnologias utilizadas

- Frontend: Python
- Backend: MySQL
- Banco de Dados: MySQL

Instalação - Pré-Requisitos

-Python (Versão 3.12.5)
-MySQL (Versão 8.0)

Funções

```python
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
import mysql.connector
from mysql.connector import Error
from PIL import Image, ImageTk
```

Bibliotecas:

tkinter para criar as interfaces gráficas, mensagens de erro e botões;
mysql.connector para conectar ao banco de dados;
Error para problemas de conexão com o banco;
PIL para abrir e configurar imagens.

```python
class SistemaLogin:
    def __init__(self):
        self.janela = tk.Tk()
        self.janela.title("Login")
        self.janela.geometry("500x500")
        self.janela.configure(bg="#151e70")

        self.inicializar_login()
        self.janela.mainloop()
```

Criação de classe SistemaLogin;
Método __init__:

self.janela para criar a janela inicial, com título, tamanho e cor;
self.inicializar_login para chamar o método;
.mainloop para manter a janela aberta.

```python
    def conectar_banco(self):
        try:
            return mysql.connector.connect(
                host='localhost',
                user='usuario',
                password='senha',
                database='caldenorte_db'
            )
        except Error as e:
            messagebox.showerror("Erro de Conexão", f"Erro ao conectar ao banco: {e}")
            return None
```

Função para conectar ao banco de dados, localhost, com usuário, senha e nome do banco.

```python
    def registrar_usuario(self, usuario, senha, tipo):
        try:
            conexao = self.conectar_banco()
            if not conexao:
                return False

            cursor = conexao.cursor()
            cursor.execute(
                "INSERT INTO usuarios (usuario, senha, tipo) VALUES (%s, %s, %s)",
                (usuario, senha, tipo)
            )
            conexao.commit()
            conexao.close()
            return True
        except Error:
            messagebox.showerror("Erro", "Erro ao registrar usuário.")
            return False
```

Função para registrar o usuário:

parâmetros usuario, senha e tipo;
testa a conexão com o banco;
executa o registro na tabela SQL.

```python
def autenticar_login(self, usuario, senha):
    try:
        conexao = self.conectar_banco()
        if not conexao:
            return None

        cursor = conexao.cursor()
        cursor.execute("SELECT tipo, id FROM usuarios WHERE usuario = %s AND senha = %s", (usuario, senha))
        resultado = cursor.fetchone()
        conexao.close()

        return resultado if resultado else None
    except Error:
        messagebox.showerror("Erro", "Erro ao autenticar.")
        return None
```

Função para autenticar o login:

parâmetros usuario e senha;

conecta com o banco e consulta se o usuario e senha correspondem;

```python
def inicializar_login(self):
    frame_login = tk.Frame(self.janela, bg="#e1e2e8", width=300, height=250)
    frame_login.place(relx=0.5, rely=0.5, anchor="center")

    logo_imagem = Image.open("icons/logo.png")
    logo_imagem = logo_imagem.resize((100, 100), Image.Resampling.LANCZOS)
    logo = ImageTk.PhotoImage(logo_imagem)

    logo_label = tk.Label(self.janela, image=logo)
    logo_label.image = logo
    logo_label.pack(pady=10)

    titulo = tk.Label(frame_login, text="Login", bg="#e1e2e8", font=("Arial", 14, "bold"))
    titulo.place(x=20, y=10)

    tk.Label(frame_login, text="Usuário:", bg="#e1e2e8", font=("Arial", 12, "bold")).place(x=20, y=60)
    self.entrada_usuario = tk.Entry(frame_login)
    self.entrada_usuario.place(x=100, y=60, width=150)

    tk.Label(frame_login, text="Senha:", bg="#e1e2e8", font=("Arial", 12, "bold")).place(x=20, y=100)
    self.entrada_senha = tk.Entry(frame_login, show="*")
    self.entrada_senha.place(x=100, y=100, width=150)

    botao_login = tk.Button(frame_login, text="Login", bg="#151e70", fg="white", command=self.processar_login)
    botao_login.place(x=20, y=160, width=120)

    botao_registrar = tk.Button(frame_login, text="Registrar", bg="#151e70", fg="white", command=self.abrir_tela_registro)
    botao_registrar.place(x=150, y=160, width=120)
```

Inicializa o login:

cria a interface de login com caixas de entrada e botões.

```
93      def abrir_tela_registro(self):
94          janela_registro = tk.Toplevel()
95          janela_registro.title("Registrar Usuário")
96          janela_registro.geometry("300x300")
97
98          tk.Label(janela_registro, text="Usuário:").pack(pady=5)
99          entrada_usuario_registro = tk.Entry(janela_registro)
100         entrada_usuario_registro.pack(pady=5)
101
102         tk.Label(janela_registro, text="Senha:").pack(pady=5)
103         entrada_senha_registro = tk.Entry(janela_registro, show="*")
104         entrada_senha_registro.pack(pady=5)
105
106         tk.Label(janela_registro, text="Tipo de Usuário:").pack(pady=5)
107         tipo_usuario_var = tk.StringVar(value="cliente")
108         tk.Radiobutton(janela_registro, text="Funcionário", variable=tipo_usuario_var, value="funcionario").pack()
109         tk.Radiobutton(janela_registro, text="Cliente", variable=tipo_usuario_var, value="cliente").pack()
111         def processar_registro():
112             usuario = entrada_usuario_registro.get()
113             senha = entrada_senha_registro.get()
114             tipo = tipo_usuario_var.get()
115
116             if not usuario or not senha:
117                 messagebox.showerror("Erro", "Todos os campos são obrigatórios!")
118                 return
119
120             if self.registrar_usuario(usuario, senha, tipo):
121                 messagebox.showinfo("Sucesso", "Usuário registrado com sucesso!")
122                 janela_registro.destroy()
123             else:
124                 messagebox.showerror("Erro", "Erro ao registrar o usuário.")
125
126         tk.Button(janela_registro, text="Registrar", command=processar_registro).pack(pady=20)
```

Abre a tela de registro e processa o registro:

cria e abre a tela de registro, com caixas de texto e botões;

processa as informações de registro.

```
128     def processar_login(self):
129         usuario = self.entrada_usuario.get()
130         senha = self.entrada_senha.get()
131
132         usuario_info = self.autenticar_login(usuario, senha)
133         if usuario_info:
134             self.abrir_menu(usuario_info[0], usuario_info[1])
135         else:
136             messagebox.showerror("Erro", "Usuário ou senha inválidos!")
```

Processa o login:

recebe as informações digitadas e autentica o login.

```
138     def abrir_menu(self, tipo_usuario, id_usuario):
139         self.janela.destroy()
140         if tipo_usuario == "funcionario":
141             self.criar_menu_funcionario()
142         elif tipo_usuario == "cliente":
143             self.criar_menu_cliente(id_usuario)
```

Abre o menu conforme o tipo de usuário.

```python
    def criar_menu_funcionario(self):
        janela_menu = tk.Tk()
        janela_menu.title("Menu Funcionário")
        janela_menu.geometry("800x600")

        frame_topo = tk.Frame(janela_menu, bg="white", height=100)
        frame_topo.pack(side="top", fill="x")

        logo_imagem = Image.open("icons/logo.png")
        logo_imagem = logo_imagem.resize((100, 100), Image.Resampling.LANCZOS)
        logo = ImageTk.PhotoImage(logo_imagem)

        label_logo = tk.Label(frame_topo, image=logo, bg="white")
        label_logo.image = logo
        label_logo.pack(side="left", padx=25, pady=10)

        titulo = tk.Label(
            frame_topo,
            text="Bem-vindo ao Sistema",
            font=("Arial", 18, "bold"),
            bg="white"
        )
        titulo.pack(pady=20)

        frame_esquerdo = tk.Frame(janela_menu, bg="white", width=200)
        frame_esquerdo.pack(side="left", fill="y")

        label_menu_adm = tk.Label(
            frame_esquerdo,
            text="MENU ADM",
            font=("Arial", 14, "bold"),
            bg="white",
            fg="#151e70"
        )
        label_menu_adm.pack(pady=10)

        self.frame_principal = tk.Frame(janela_menu, bg="white")
        self.frame_principal.pack(side="right", fill="both", expand=True)

        botoes = [
            ("Funcionários", "funcionarios"),
            ("Clientes", "clientes"),
            ("Produtos", "produtos"),
            ("Pedidos", "pedidos"),
            ("Transportadoras", "transportadoras"),
            ("Fornecedores", "fornecedores"),
        ]

        for texto, tabela in botoes:
            botao = tk.Button(frame_esquerdo,
                              text=texto,
                              bg="#151e70",
                              fg="white",
                              font=("Arial", 12, "bold"),
                              relief="raised",
                              width=15,
                              command=lambda t=tabela: self.visualizar_tabela(t))
            botao.pack(pady=10)

        botao_sair = tk.Button(
            frame_esquerdo,
            text="SAIR",
            bg="#FF0000",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            width=15,
            command=lambda: self.sair_funcionario()
        )
        botao_sair.pack(pady=10)

        self.janela_menu = janela_menu
```

Cria o menu do funcionário:

cria a interface do menu do funcionário;
cria botões para visualizar as tabelas em um menu lateral;
e um botão para fazer logout.

```python
    def sair_funcionario(self):
        self.janela_menu.destroy()
        self.__init__()
```

Função que executa o botão para fazer logout do funcionario e volta para a tela de login.

```python
    def visualizar_tabela(self, tabela):
        for widget in self.frame_principal.winfo_children():
            widget.destroy()

        conexao = self.conectar_banco()
        if not conexao:
            return

        cursor = conexao.cursor()
        try:
            cursor.execute(f"SELECT * FROM {tabela}")
            dados = cursor.fetchall()
            colunas = [desc[0] for desc in cursor.description]
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao carregar dados da tabela: {e}")
            return
        finally:
            conexao.close()

        style = ttk.Style()
        style.configure(
            "Custom.Treeview",
            background="#e1f5fe",
            foreground="black",
            rowheight=25,
            fieldbackground="#e1f5fe",
            font=("Arial", 12)
        )
        style.configure("Custom.Treeview.Heading", font=("Arial", 14, "bold"))

        frame_pesquisa = tk.Frame(self.frame_principal, bg="white")
        frame_pesquisa.pack(fill="x", padx=10, pady=5)

        tk.Label(frame_pesquisa, text="Digite para pesquisar:", font=("Arial", 12, "bold"), bg="white").pack(side="left", padx=5)

        self.entry_pesquisa = tk.Entry(frame_pesquisa, font=("Arial", 12))
        self.entry_pesquisa.pack(side="left", padx=5)

        botao_pesquisa = tk.Button(
            frame_pesquisa,
            text="Pesquisar",
            bg="#151e70",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            command=lambda: self.pesquisar_dados(tabela, colunas)
        )
        botao_pesquisa.pack(side="left", padx=5)

        scrollbar_y = ttk.Scrollbar(self.frame_principal, orient="vertical")
        scrollbar_x = ttk.Scrollbar(self.frame_principal, orient="horizontal")

        treeview = ttk.Treeview(
            self.frame_principal,
            columns=colunas,
            show="headings",
            style="Custom.Treeview",
            yscrollcommand=scrollbar_y.set,
            xscrollcommand=scrollbar_x.set
        )

        for coluna in colunas:
            treeview.heading(coluna, text=coluna)
            treeview.column(coluna, width=150, anchor="center")

        scrollbar_y.config(command=treeview.yview)
        scrollbar_x.config(command=treeview.xview)

        treeview.pack(side="top", fill="both", expand=True)
        scrollbar_y.pack(side="right", fill="y")
        scrollbar_x.pack(side="bottom", fill="x")

        self.treeview = treeview

        for linha in dados:
            treeview.insert("", "end", values=linha)

        frame_botoes = tk.Frame(self.frame_principal, bg="white")
        frame_botoes.pack(fill="x", padx=10, pady=10)
```

```
        botao_adicionar = tk.Button(
            frame_botoes,
            text="Adicionar",
            bg="#151e70",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            width=15,
            command=lambda: self.adicionar_registro(tabela, colunas, treeview)
        )
        botao_adicionar.pack(side="left", padx=5)

        botao_alterar = tk.Button(
            frame_botoes,
            text="Alterar",
            bg="#151e70",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            width=15,
            command=lambda: self.alterar_registro(tabela, colunas, treeview)
        )
        botao_alterar.pack(side="left", padx=5)

        botao_excluir = tk.Button(
            frame_botoes,
            text="Excluir",
            bg="#FF0000",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            width=15,
            command=lambda: self.excluir_registro(tabela, colunas, treeview)
        )
        botao_excluir.pack(side="left", padx=5)

```

Função para visualizar as tabelas:

usa o treeview para mostrar a tabela;
cria uma barra de pesquisa com botão;
cria botões para criar, alterar ou excluir informações das tabelas.

```
    def pesquisar_dados(self, tabela, colunas):
        for widget in self.treeview.get_children():
            self.treeview.delete(widget)

        filtro = self.entry_pesquisa.get()

        conexao = self.conectar_banco()
        if not conexao:
            return

        cursor = conexao.cursor()
        try:
            query = f"SELECT * FROM {tabela} WHERE {' OR '.join([f'{coluna} LIKE %s' for coluna in colunas])}"
            valores = [f"%{filtro}%" for _ in colunas]
            cursor.execute(query, valores)
            dados = cursor.fetchall()
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao buscar dados: {e}")
            return
        finally:
            conexao.close()

        for linha in dados:
            self.treeview.insert("", "end", values=linha)
```

Função para filtrar as informações inseridas na barra de pesquisa.

```python
    def adicionar_registro(self, tabela, colunas, treeview):
        for widget in self.frame_principal.winfo_children():
            widget.destroy()

        tk.Label(
            self.frame_principal,
            text=f"Adicionar Registro em {tabela.capitalize()}",
            bg="white",
            fg="#151e70",
            font=("Arial", 14, "bold")
        ).pack(pady=10)

        entradas = {}
        for coluna in colunas:
            tk.Label(self.frame_principal, text=coluna).pack(pady=5)
            entrada = tk.Entry(self.frame_principal)
            entrada.pack(pady=5)
            entradas[coluna] = entrada

        def salvar():
            valores = [entrada.get() for entrada in entradas.values()]
            conexao = self.conectar_banco()
            if not conexao:
                return
            try:
                cursor = conexao.cursor()
                placeholders = ", ".join(["%s"] * len(valores))
                cursor.execute(
                    f"INSERT INTO {tabela} ({', '.join(colunas)}) VALUES ({placeholders})",
                    valores
                )
                conexao.commit()

                messagebox.showinfo("Sucesso", "Registro adicionado com sucesso!")
                conexao.close()

                self.visualizar_tabela(tabela)

            except Exception as e:
                messagebox.showerror("Erro", f"Não foi possível adicionar o registro.\n{e}")
            finally:
                conexao.close()

        tk.Button(self.frame_principal, text="Salvar", command=salvar, bg="#151e70", fg="white").pack(pady=20)
```

Função para criar um registro na tabela:

Abre uma tela para criar um novo registro;

Verifica se o registro pode ser inserido e registra no banco.

```python
    def alterar_registro(self, tabela, colunas, treeview):
        item_selecionado = treeview.selection()
        if not item_selecionado:
            messagebox.showerror("Erro", "Nenhum registro selecionado!")
            return

        valores_selecionados = treeview.item(item_selecionado, "values")

        for widget in self.frame_principal.winfo_children():
            widget.destroy()

        tk.Label(
            self.frame_principal,
            text=f"Alterar Registro em {tabela.capitalize()}",
            bg="white",
            fg="#151e70",
            font=("Arial", 14, "bold")
        ).pack(pady=10)

        entradas = {}
        for i, coluna in enumerate(colunas):
            tk.Label(self.frame_principal, text=coluna).pack(pady=5)
            entrada = tk.Entry(self.frame_principal)
            entrada.pack(pady=5)
            entrada.insert(0, valores_selecionados[i])
            entradas[coluna] = entrada

        def salvar():
            novos_valores = [entrada.get() for entrada in entradas.values()]
            conexao = self.conectar_banco()
            if not conexao:
                return
            try:
                cursor = conexao.cursor()
                sets = ", ".join([f"{coluna} = %s" for coluna in colunas])
                cursor.execute(
                    f"UPDATE {tabela} SET {sets} WHERE {colunas[0]} = %s",
                    novos_valores + [valores_selecionados[0]]
                )
                conexao.commit()

                messagebox.showinfo("Sucesso", "Registro alterado com sucesso!")
                conexao.close()

                self.visualizar_tabela(tabela)

            except Exception as e:
                messagebox.showerror("Erro", f"Erro ao atualizar registro: {e}")
            finally:
                conexao.close()

        tk.Button(self.frame_principal, text="Salvar", command=salvar, bg="#151e70", fg="white").pack(pady=20)
```

Função para alterar o registro:

Verifica se o usuário selecionou o registro para alterar;
Abre uma tela para alterar as informações;
Salva as informações alteradas no banco.

```python
    def excluir_registro(self, tabela, colunas, treeview):
        item_selecionado = treeview.selection()
        if not item_selecionado:
            messagebox.showerror("Erro", "Nenhum registro selecionado!")
            return

        valores_selecionados = treeview.item(item_selecionado, "values")

        confirmar = messagebox.askyesno("Confirmar", "Tem certeza que deseja excluir este registro?")
        if not confirmar:
            return

        conexao = self.conectar_banco()
        if not conexao:
            return
        cursor = conexao.cursor()
        try:
            cursor.execute(f"DELETE FROM {tabela} WHERE {colunas[0]} = %s", (valores_selecionados[0],))
            conexao.commit()
            messagebox.showinfo("Sucesso", "Registro excluído com sucesso!")
            treeview.delete(item_selecionado)
        except Error as e:
            messagebox.showerror("Erro", f"Erro ao excluir registro: {e}")
        finally:
            conexao.close()

```

Função para excluir um registro na tabela:

Verifica se o usuário selecionou o registro para excluir;

Exclui o registro do banco;

```python
    def criar_menu_cliente(self, id_cliente):
        self.janela_cliente = tk.Tk()
        self.janela_cliente.title("Menu Cliente")
        self.janela_cliente.geometry("800x600")

        frame_topo_cliente = tk.Frame(self.janela_cliente, bg="white", height=100)
        frame_topo_cliente.pack(side="top", fill="x")

        try:
            logo_imagem = Image.open("icons/logo.png")
            logo_imagem = logo_imagem.resize((100, 100), Image.Resampling.LANCZOS)
            logo = ImageTk.PhotoImage(logo_imagem)

            label_logo = tk.Label(frame_topo_cliente, image=logo, bg="white")
            label_logo.image = logo
            label_logo.pack(side="left", padx=25, pady=10)
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao carregar o logo: {e}")

        titulo_cliente = tk.Label(
            frame_topo_cliente,
            text="Bem Vindo ao Sistema",
            font=("Arial", 14, "bold"),
            bg="white"
        )
        titulo_cliente.pack(pady=20)

        frame_esquerdo_cliente = tk.Frame(self.janela_cliente, bg="white", width=200)
        frame_esquerdo_cliente.pack(side="left", fill="y")

        label_menu_cliente = tk.Label(
            frame_esquerdo_cliente,
            text="MENU CLIENTE",
            font=("Arial", 14, "bold"),
            bg="white",
            fg="#151e70"
        )
        label_menu_cliente.pack(pady=10)

        botoes_cliente = [
            ("Visualizar Produtos", self.visualizar_produtos_cliente),
            ("Alterar Informações", lambda: self.abrir_tela_alterar_info(id_cliente))
        ]

        for texto, comando in botoes_cliente:
            botao_cliente = tk.Button(
                frame_esquerdo_cliente,
                text=texto,
                bg="#151e70",
                fg="white",
                font=("Arial", 12, "bold"),
                relief="raised",
                width=15,
                command=comando
            )
            botao_cliente.pack(pady=5)

```

```
        botao_sair = tk.Button(
            frame_esquerdo_cliente,
            text="SAIR",
            bg="#FF0000",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            width=15,
            command=lambda: self.sair_cliente()
        )
        botao_sair.pack(pady=5)

        self.janela_cliente = self.janela_cliente

        self.frame_direita = tk.Frame(self.janela_cliente, bg="white")
        self.frame_direita.pack(side="right", fill="both", expand=True)

```

Cria o menu do cliente:

cria a interface do menu do cliente;

cria botões para visualizar os produtos e alterar as informações em um menu lateral;

e um botão para fazer logout.

```
    def sair_cliente(self):
        self.janela_cliente.destroy()
        self.__init__()

```

Função que executa o botão para fazer logout do cliente e volta para a tela de login.

```
    def visualizar_produtos_cliente(self):
        conexao = self.conectar_banco()
        if not conexao:
            return

        for widget in self.frame_direita.winfo_children():
            widget.destroy()

        style = ttk.Style()
        style.configure(
            "Custom.Treeview",
            background="#e1f5fe",
            foreground="black",
            rowheight=25,
            fieldbackground="#e1f5fe",
            font=("Arial", 12)
        )
        style.configure("Custom.Treeview.Heading", font=("Arial", 14, "bold"))

        frame_pesquisa = tk.Frame(self.frame_direita, bg="white")
        frame_pesquisa.pack(fill="x", padx=10, pady=5)

        tk.Label(frame_pesquisa, text="Digite para pesquisar:", font=("Arial", 12, "bold"), bg="white").pack(side="left", padx=5)

        self.entry_pesquisa_produtos = tk.Entry(frame_pesquisa, font=("Arial", 12))
        self.entry_pesquisa_produtos.pack(side="left", padx=5)

        botao_pesquisa = tk.Button(
            frame_pesquisa,
            text="Pesquisar",
            bg="#151e70",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="raised",
            command=lambda: self.pesquisar_produtos()
        )
        botao_pesquisa.pack(side="left", padx=5)

        scrollbar_y = ttk.Scrollbar(self.frame_direita, orient="vertical")
        scrollbar_x = ttk.Scrollbar(self.frame_direita, orient="horizontal")

        treeview = ttk.Treeview(
            self.frame_direita,
            columns=("Produto", "Preço"),
            show="headings",
            style="Custom.Treeview",
            yscrollcommand=scrollbar_y.set,
            xscrollcommand=scrollbar_x.set
        )

        treeview.heading("Produto", text="Produto")
        treeview.heading("Preço", text="Preço")
        treeview.column("Produto", width=250)
        treeview.column("Preço", width=100)
```

```
        try:
            cursor = conexao.cursor()
            cursor.execute("SELECT nome, valor FROM produtos")
            produtos = cursor.fetchall()
            for produto in produtos:
                treeview.insert("", "end", values=produto)
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao buscar produtos: {e}")
        finally:
            conexao.close()

        scrollbar_y.config(command=treeview.yview)
        scrollbar_x.config(command=treeview.xview)

        treeview.pack(side="top", fill="both", expand=True)
        scrollbar_y.pack(side="right", fill="y")
        scrollbar_x.pack(side="bottom", fill="x")

        self.treeview_produtos = treeview
```

Função para visualizar a tabela de produto:

usa o treeview para mostrar a tabela;

cria uma barra de pesquisa com botão;

```
    def pesquisar_produtos(self):
        for widget in self.treeview_produtos.get_children():
            self.treeview_produtos.delete(widget)

        filtro = self.entry_pesquisa_produtos.get()

        conexao = self.conectar_banco()
        if not conexao:
            return

        cursor = conexao.cursor()
        try:
            query = "SELECT nome, valor FROM produtos WHERE nome LIKE %s"
            cursor.execute(query, (f"%{filtro}%",))
            produtos = cursor.fetchall()

            for produto in produtos:
                self.treeview_produtos.insert("", "end", values=produto)
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao buscar produtos: {e}")
        finally:
            conexao.close()
```

Função para filtrar as informações na barra de pesquisa na tabela de produtos;

```python
    def abrir_tela_alterar_info(self, id_cliente):
        for widget in self.frame_direita.winfo_children():
            widget.destroy()

        tk.Label(self.frame_direita, text="Alterar Informações", bg="white", fg="#151e70", font=("Arial", 14, "bold")).pack(pady=10)

        tk.Label(self.frame_direita, text="Novo Nome:").pack(pady=5)
        entrada_nome = tk.Entry(self.frame_direita)
        entrada_nome.pack(pady=5)

        tk.Label(self.frame_direita, text="Nova Senha:").pack(pady=5)
        entrada_senha = tk.Entry(self.frame_direita, show="*")
        entrada_senha.pack(pady=5)

        def processar_alteracao():
            nome = entrada_nome.get()
            senha = entrada_senha.get()

            conexao = self.conectar_banco()
            if not conexao:
                return

            try:
                cursor = conexao.cursor()
                if nome:
                    cursor.execute("UPDATE clientes SET nome = %s WHERE id = %s", (nome, id_cliente))
                if senha:
                    cursor.execute("UPDATE usuarios SET senha = %s WHERE id = %s", (senha, id_cliente))
                conexao.commit()
                messagebox.showinfo("Sucesso", "Informações alteradas com sucesso!")
            except Exception as e:
                messagebox.showerror("Erro", f"Erro ao atualizar informações: {e}")
            finally:
                conexao.close()

        tk.Button(self.frame_direita, text="Alterar", command=processar_alteracao, bg="#151e70", fg="white").pack(pady=20)
```

Função para alterar as informações do usuario cliente:

Cria uma tela para o usuário alterar seu nome e senha e salva no banco.

```python
    def run(self):
        self.janela.mainloop()
```

Função que mantém a interface gráfica ativa e permitindo as interações.

```python
if __name__ == "__main__":
    sistema = SistemaLogin()
    sistema.run()
```

Inicia o Sistema.