

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

```
1 import tkinter as tk
2 from tkinter import messagebox
3 from tkinter import ttk
4 import mysql.connector
5 from mysql.connector import Error
6 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.

```
8 class SistemaLogin:
9     def __init__(self):
10         self.janela = tk.Tk()
11         self.janela.title("Login")
12         self.janela.geometry("500x500")
13         self.janela.configure(bg="#151e70")
14
15         self.inicializar_login()
16         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.

```
18 def conectar_banco(self):
19     try:
20         return mysql.connector.connect(
21             host='localhost',
22             user='usuario',
23             password='senha',
24             database='caldenorte_db'
25         )
26     except Error as e:
27         messagebox.showerror("Erro de Conexão", f"Erro ao conectar ao banco: {e}")
28     return None
```

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

```
30 def registrar_usuario(self, usuario, senha, tipo):
31     try:
32         conexao = self.conectar_banco()
33         if not conexao:
34             return False
35
36         cursor = conexao.cursor()
37         cursor.execute(
38             "INSERT INTO usuarios (usuario, senha, tipo) VALUES (%s, %s, %s)",
39             (usuario, senha, tipo)
40         )
41         conexao.commit()
42         conexao.close()
43         return True
44     except Error:
45         messagebox.showerror("Erro", "Erro ao registrar usuário.")
46     return False
```

Função para registrar o usuário:

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

```

48  def autenticar_login(self, usuario, senha):
49      try:
50          conexao = self.conectar_banco()
51          if not conexao:
52              return None
53
54          cursor = conexao.cursor()
55          cursor.execute("SELECT tipo, id FROM usuarios WHERE usuario = %s AND senha = %s", (usuario, senha))
56          resultado = cursor.fetchone()
57          conexao.close()
58
59          return resultado if resultado else None
60      except Error:
61          messagebox.showerror("Erro", "Erro ao autenticar.")
62          return None

```

Função para autenticar o login:
 parâmetros usuario e senha;
 conecta com o banco e consulta se o usuario e senha correspondem;

```

64  def inicializar_login(self):
65      frame_login = tk.Frame(self.janela, bg="#e1e2e8", width=300, height=250)
66      frame_login.place(relx=0.5, rely=0.5, anchor="center")
67
68      logo_imagem = Image.open("icons/logo.png")
69      logo_imagem = logo_imagem.resize((100, 100), Image.Resampling.LANCZOS)
70      logo = ImageTk.PhotoImage(logo_imagem)
71
72      logo_label = tk.Label(self.janela, image=logo)
73      logo_label.image = logo
74      logo_label.pack(pady=10)
75
76      titulo = tk.Label(frame_login, text="Login", bg="#e1e2e8", font=("Arial", 14, "bold"))
77      titulo.place(x=20, y=10)
78
79      tk.Label(frame_login, text="Usuário:", bg="#e1e2e8", font=("Arial", 12, "bold")).place(x=20, y=60)
80      self.entrada_usuario = tk.Entry(frame_login)
81      self.entrada_usuario.place(x=100, y=60, width=150)
82
83      tk.Label(frame_login, text="Senha:", bg="#e1e2e8", font=("Arial", 12, "bold")).place(x=20, y=100)
84      self.entrada_senha = tk.Entry(frame_login, show="*")
85      self.entrada_senha.place(x=100, y=100, width=150)
86
87      botao_login = tk.Button(frame_login, text="Login", bg="#151e70", fg="white", command=self.processar_login)
88      botao_login.place(x=20, y=160, width=120)
89
90      botao_registrar = tk.Button(frame_login, text="Registrar", bg="#151e70", fg="white", command=self.abrir_tela_registro)
91      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()
110
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.

```

145     def criar_menu_funcionario(self):
146         janela_menu = tk.Tk()
147         janela_menu.title("Menu Funcionário")
148         janela_menu.geometry("800x600")
149
150         frame_topo = tk.Frame(janela_menu, bg="white", height=100)
151         frame_topo.pack(side="top", fill="x")
152
153         logo_image = Image.open("icons/logo.png")
154         logo_image = logo_image.resize((100, 100), Image.Resampling.LANCZOS)
155         logo = ImageTk.PhotoImage(logo_image)
156
157         label_logo = tk.Label(frame_topo, image=logo, bg="white")
158         label_logo.image = logo
159         label_logo.pack(side="left", padx=25, pady=10)
160
161         titulo = tk.Label(
162             frame_topo,
163             text="Bem-vindo ao Sistema",
164             font=("Arial", 18, "bold"),
165             bg="white"
166         )
167         titulo.pack(pady=20)
168
169         frame_esquerdo = tk.Frame(janela_menu, bg="white", width=200)
170         frame_esquerdo.pack(side="left", fill="y")
171
172         label_menu_adm = tk.Label(
173             frame_esquerdo,
174             text="MENU ADM",
175             font=("Arial", 14, "bold"),
176             bg="white",
177             fg="#151e70"
178         )
179         label_menu_adm.pack(pady=10)
180
181         self.frame_principal = tk.Frame(janela_menu, bg="white")
182         self.frame_principal.pack(side="right", fill="both", expand=True)
183
184         botoes = [
185             ("Funcionários", "funcionarios"),
186             ("Clientes", "clientes"),
187             ("Produtos", "produtos"),
188             ("Pedidos", "pedidos"),
189             ("Transportadoras", "transportadoras"),
190             ("Fornecedores", "fornecedores"),
191         ]
192
193         for texto, tabela in botoes:
194             botao = tk.Button(frame_esquerdo,
195                             text=texto,
196                             bg="#151e70",
197                             fg="white",
198                             font=("Arial", 12, "bold"),
199                             relief="raised",
200                             width=15,
201                             command=lambda t=tabela: self.visualizar_tabela(t))
202             botao.pack(pady=10)
203
204             botao_sair = tk.Button(
205                 frame_esquerdo,
206                 text="SAIR",
207                 bg="#ff0000",
208                 fg="white",
209                 font=("Arial", 12, "bold"),
210                 relief="raised",
211                 width=15,
212                 command=lambda: self.sair_funcionario()
213             )
214             botao_sair.pack(pady=10)
215
216         self.janela_menu = janela_menu
217

```

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.

```

218     def sair_funcionario(self):
219         self.janela_menu.destroy()
220         self.__init__()

```

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

```

222     def visualizar_tabela(self, tabela):
223         for widget in self.frame_principal.winfo_children():
224             widget.destroy()
225
226         conexao = self.conectar_banco()
227         if not conexao:
228             return
229
230         cursor = conexao.cursor()
231         try:
232             cursor.execute(f"SELECT * FROM {tabela}")
233             dados = cursor.fetchall()
234             colunas = [desc[0] for desc in cursor.description]
235         except Exception as e:
236             messagebox.showerror("Erro", f"Erro ao carregar dados da tabela: {e}")
237             return
238         finally:
239             conexao.close()
240
241         style = ttk.Style()
242         style.configure(
243             "Custom.Treeview",
244             background="#e1f5fe",
245             foreground="black",
246             rowheight=25,
247             fieldbackground="#e1f5fe",
248             font=("Arial", 12)
249         )
250         style.configure("Custom.Treeview.Hheading", font=("Arial", 14, "bold"))
251
252         frame_pesquisa = tk.Frame(self.frame_principal, bg="white")
253         frame_pesquisa.pack(fill="x", padx=10, pady=5)
254
255         tk.Label(frame_pesquisa, text="Digite para pesquisar:", font=("Arial", 12, "bold"), bg="white").pack(side="left", padx=5)
256
257         self.entry_pesquisa = tk.Entry(frame_pesquisa, font=("Arial", 12))
258         self.entry_pesquisa.pack(side="left", padx=5)
259
260         botoao_pesquisa = tk.Button(
261             frame_pesquisa,
262             text="Pesquisar",
263             bg="#151e70",
264             fg="white",
265             font=("Arial", 12, "bold"),
266             relief="raised",
267             command=lambda: self.pesquisar_dados(tabela, colunas)
268         )
269         botoao_pesquisa.pack(side="left", padx=5)
270
271         scrollbar_y = ttk.Scrollbar(self.frame_principal, orient="vertical")
272         scrollbar_x = ttk.Scrollbar(self.frame_principal, orient="horizontal")
273
274         treeview = ttk.Treeview(
275             self.frame_principal,
276             columns=colunas,
277             show="headings",
278             style="Custom.Treeview",
279             yscrollcommand=scrollbar_y.set,
280             xscrollcommand=scrollbar_x.set
281         )
282
283         for coluna in colunas:
284             treeview.heading(coluna, text=coluna)
285             treeview.column(coluna, width=150, anchor="center")
286
287         scrollbar_y.config(command=treeview.yview)
288         scrollbar_x.config(command=treeview.xview)
289
290         treeview.pack(side="top", fill="both", expand=True)
291         scrollbar_y.pack(side="right", fill="y")
292         scrollbar_x.pack(side="bottom", fill="x")
293
294         self.treeview = treeview
295
296         for linha in dados:
297             treeview.insert("", "end", values=linha)
298
299         frame_botoes = tk.Frame(self.frame_principal, bg="white")
300         frame_botoes.pack(fill="x", padx=10, pady=10)
301

```

```

302         botao_adicionar = tk.Button(
303             frame_botoes,
304             text="Adicionar",
305             bg="#151e70",
306             fg="white",
307             font=("Arial", 12, "bold"),
308             relief="raised",
309             width=15,
310             command=lambda: self.adicionar_registro(tabela, colunas, treeview)
311         )
312         botao_adicionar.pack(side="left", padx=5)
313
314         botao_alterar = tk.Button(
315             frame_botoes,
316             text="Alterar",
317             bg="#151e70",
318             fg="white",
319             font=("Arial", 12, "bold"),
320             relief="raised",
321             width=15,
322             command=lambda: self.alterar_registro(tabela, colunas, treeview)
323         )
324         botao_alterar.pack(side="left", padx=5)
325
326         botao_excluir = tk.Button(
327             frame_botoes,
328             text="Excluir",
329             bg="#ff0000",
330             fg="white",
331             font=("Arial", 12, "bold"),
332             relief="raised",
333             width=15,
334             command=lambda: self.excluir_registro(tabela, colunas, treeview)
335         )
336         botao_excluir.pack(side="left", padx=5)
337

```

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.

```

339     def pesquisar_dados(self, tabela, colunas):
340         for widget in self.treeview.get_children():
341             self.treeview.delete(widget)
342
343         filtro = self.entry_pesquisa.get()
344
345         conexao = self.conectar_banco()
346         if not conexao:
347             return
348
349         cursor = conexao.cursor()
350         try:
351             query = f"SELECT * FROM {tabela} WHERE {' OR '.join([f'{coluna} LIKE %s' for coluna in colunas])}"
352             valores = [f"%{filtro}%" for _ in colunas]
353             cursor.execute(query, valores)
354             dados = cursor.fetchall()
355         except Exception as e:
356             messagebox.showerror("Erro", f"Erro ao buscar dados: {e}")
357             return
358         finally:
359             conexao.close()
360
361         for linha in dados:
362             self.treeview.insert("", "end", values=linha)
363

```

Função para filtrar as informações inseridas na barra de pesquisa.


```

364     def adicionar_registro(self, tabela, colunas, treeview):
365         for widget in self.frame_principal.wininfo_children():
366             widget.destroy()
367
368         tk.Label(
369             self.frame_principal,
370             text=f"Adicionar Registro em {tabela.capitalize()}",
371             bg="white",
372             fg="#151e70",
373             font=("Arial", 14, "bold")
374         ).pack(pady=10)
375
376         entradas = {}
377         for coluna in colunas:
378             tk.Label(self.frame_principal, text=coluna).pack(pady=5)
379             entrada = tk.Entry(self.frame_principal)
380             entrada.pack(pady=5)
381             entradas[coluna] = entrada
382
383     def salvar():
384         valores = [entrada.get() for entrada in entradas.values()]
385         conexao = self.conectar_banco()
386         if not conexao:
387             return
388         try:
389             cursor = conexao.cursor()
390             placeholders = ", ".join(["%s"] * len(valores))
391             cursor.execute(
392                 f"INSERT INTO {tabela} ({', '.join(colunas)}) VALUES ({placeholders})",
393                 valores
394             )
395             conexao.commit()
396
397             messagebox.showinfo("Sucesso", "Registro adicionado com sucesso!")
398             conexao.close()
399
400             self.visualizar_tabela(tabela)
401
402         except Exception as e:
403             messagebox.showerror("Erro", f"Não foi possível adicionar o registro.\n(e)")
404         finally:
405             conexao.close()
406
407         tk.Button(self.frame_principal, text="Salvar", command=salvar, bg="#151e70", fg="white").pack(pady=20)
408

```

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.

```

409  def alterar_registro(self, tabela, colunas, treeview):
410      item_selecionado = treeview.selection()
411      if not item_selecionado:
412          messagebox.showerror("Erro", "Nenhum registro selecionado!")
413          return
414
415      valores_selecionados = treeview.item(item_selecionado, "values")
416
417      for widget in self.frame_principal.winfo_children():
418          widget.destroy()
419
420      tk.Label(
421          self.frame_principal,
422          text=f"Alterar Registro em {tabela.capitalize()}",
423          bg="white",
424          fg="#151e70",
425          font=("Arial", 14, "bold")
426      ).pack(pady=10)
427
428      entradas = {}
429      for i, coluna in enumerate(colunas):
430          tk.Label(self.frame_principal, text=coluna).pack(pady=5)
431          entrada = tk.Entry(self.frame_principal)
432          entrada.pack(pady=5)
433          entrada.insert(0, valores_selecionados[i])
434          entradas[coluna] = entrada
435
436  def salvar():
437      novos_valores = [entrada.get() for entrada in entradas.values()]
438      conexao = self.conectar_banco()
439      if not conexao:
440          return
441      try:
442          cursor = conexao.cursor()
443          sets = ", ".join([f"{coluna} = %s" for coluna in colunas])
444          cursor.execute(
445              f"UPDATE {tabela} SET {sets} WHERE {colunas[0]} = %s",
446              novos_valores + [valores_selecionados[0]]
447          )
448          conexao.commit()
449
450          messagebox.showinfo("Sucesso", "Registro alterado com sucesso!")
451          conexao.close()
452
453          self.visualizar_tabela(tabela)
454
455      except Exception as e:
456          messagebox.showerror("Erro", f"Erro ao atualizar registro: {e}")
457      finally:
458          conexao.close()
459
460      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.

```

462 def excluir_registro(self, tabela, colunas, treeview):
463     item_selecionado = treeview.selection()
464     if not item_selecionado:
465         messagebox.showerror("Erro", "Nenhum registro selecionado!")
466         return
467
468     valores_selecionados = treeview.item(item_selecionado, "values")
469
470     confirmar = messagebox.askyesno("Confirmar", "Tem certeza que deseja excluir este registro?")
471     if not confirmar:
472         return
473
474     conexao = self.conectar_banco()
475     if not conexao:
476         return
477     cursor = conexao.cursor()
478     try:
479         cursor.execute(f"DELETE FROM {tabela} WHERE {colunas[0]} = %s", (valores_selecionados[0],))
480         conexao.commit()
481         messagebox.showinfo("Sucesso", "Registro excluído com sucesso!")
482         treeview.delete(item_selecionado)
483     except Error as e:
484         messagebox.showerror("Erro", f"Erro ao excluir registro: {e}")
485     finally:
486         conexao.close()
487

```

Função para excluir um registro na tabela:

Verifica se o usuário selecionou o registro para excluir;

Exclui o registro do banco;

```

489 def criar_menu_cliente(self, id_cliente):
490     self.janela_cliente = tk.Tk()
491     self.janela_cliente.title("Menu Cliente")
492     self.janela_cliente.geometry("800x600")
493
494     frame_topo_cliente = tk.Frame(self.janela_cliente, bg="white", height=100)
495     frame_topo_cliente.pack(side="top", fill="x")
496
497     try:
498         logo_imagem = Image.open("icons/logo.png")
499         logo_imagem = logo_imagem.resize((100, 100), Image.Resampling.LANCZOS)
500         logo = ImageTk.PhotoImage(logo_imagem)
501
502         label_logo = tk.Label(frame_topo_cliente, image=logo, bg="white")
503         label_logo.image = logo
504         label_logo.pack(side="left", padx=25, pady=10)
505     except Exception as e:
506         messagebox.showerror("Erro", f"Erro ao carregar o logo: {e}")
507
508     titulo_cliente = tk.Label(
509         frame_topo_cliente,
510         text="Bem Vindo ao Sistema",
511         font=("Arial", 14, "bold"),
512         bg="white"
513     )
514     titulo_cliente.pack(pady=20)
515
516     frame_esquerdo_cliente = tk.Frame(self.janela_cliente, bg="white", width=200)
517     frame_esquerdo_cliente.pack(side="left", fill="y")
518
519     label_menu_cliente = tk.Label(
520         frame_esquerdo_cliente,
521         text="MENU CLIENTE",
522         font=("Arial", 14, "bold"),
523         bg="white",
524         fg="#151e70"
525     )
526     label_menu_cliente.pack(pady=10)
527
528     botoes_cliente = [
529         ("Visualizar Produtos", self.visualizar_produtos_cliente),
530         ("Alterar Informações", lambda: self.abrir_tela_alterar_info(id_cliente))
531     ]
532
533     for texto, comando in botoes_cliente:
534         botao_cliente = tk.Button(
535             frame_esquerdo_cliente,
536             text=texto,
537             bg="#151e70",
538             fg="white",
539             font=("Arial", 12, "bold"),
540             relief="raised",
541             width=15,
542             command=comando
543         )
544         botao_cliente.pack(pady=5)
545

```

```

545
546         botao_sair = tk.Button(
547             frame_esquerdo_cliente,
548             text="SAIR",
549             bg="#FF0000",
550             fg="white",
551             font=("Arial", 12, "bold"),
552             relief="raised",
553             width=15,
554             command=lambda: self.sair_cliente()
555         )
556         botao_sair.pack(pady=5)
557
558         self.janela_cliente = self.janela_cliente
559
560         self.frame_direita = tk.Frame(self.janela_cliente, bg="white")
561         self.frame_direita.pack(side="right", fill="both", expand=True)
562

```

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.

```

563     def sair_cliente(self):
564         self.janela_cliente.destroy()
565         self.__init__()
566

```

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

```

567     def visualizar_produtos_cliente(self):
568         conexao = self.conectar_banco()
569         if not conexao:
570             return
571
572         for widget in self.frame_direita.winfo_children():
573             widget.destroy()
574
575         style = ttk.Style()
576         style.configure(
577             "Custom.Treeview",
578             background="#e1f5fe",
579             foreground="black",
580             rowheight=25,
581             fieldbackground="#e1f5fe",
582             font=("Arial", 12)
583         )
584         style.configure("Custom.Treeview.Heading", font=("Arial", 14, "bold"))
585
586         frame_pesquisa = tk.Frame(self.frame_direita, bg="white")
587         frame_pesquisa.pack(fill="x", padx=10, pady=5)
588
589         tk.Label(frame_pesquisa, text="Digite para pesquisar:", font=("Arial", 12, "bold"), bg="white").pack(side="left", padx=5)
590
591         self.entry_pesquisa_produtos = tk.Entry(frame_pesquisa, font=("Arial", 12))
592         self.entry_pesquisa_produtos.pack(side="left", padx=5)
593
594         botao_pesquisa = tk.Button(
595             frame_pesquisa,
596             text="Pesquisar",
597             bg="#151e70",
598             fg="white",
599             font=("Arial", 12, "bold"),
600             relief="raised",
601             command=lambda: self.pesquisar_produtos()
602         )
603         botao_pesquisa.pack(side="left", padx=5)
604
605         scrollbar_y = ttk.Scrollbar(self.frame_direita, orient="vertical")
606         scrollbar_x = ttk.Scrollbar(self.frame_direita, orient="horizontal")
607
608         treeview = ttk.Treeview(
609             self.frame_direita,
610             columns=("Produto", "Preço"),
611             show="headings",
612             style="Custom.Treeview",
613             yscrollcommand=scrollbar_y.set,
614             xscrollcommand=scrollbar_x.set
615         )
616
617         treeview.heading("Produto", text="Produto")
618         treeview.heading("Preço", text="Preço")
619         treeview.column("Produto", width=250)
620         treeview.column("Preço", width=100)

```

```

622         try:
623             cursor = conexao.cursor()
624             cursor.execute("SELECT nome, valor FROM produtos")
625             produtos = cursor.fetchall()
626             for produto in produtos:
627                 treeview.insert("", "end", values=produto)
628         except Exception as e:
629             messagebox.showerror("Erro", f"Erro ao buscar produtos: {e}")
630         finally:
631             conexao.close()
632
633         scrollbar_y.config(command=treeview.yview)
634         scrollbar_x.config(command=treeview.xview)
635
636         treeview.pack(side="top", fill="both", expand=True)
637         scrollbar_y.pack(side="right", fill="y")
638         scrollbar_x.pack(side="bottom", fill="x")
639
640         self.treeview_produtos = treeview
641

```

Função para visualizar a tabela de produto:
 usa o treeview para mostrar a tabela;
 cria uma barra de pesquisa com botão;

```

642
643     def pesquisar_produtos(self):
644         for widget in self.treeview_produtos.get_children():
645             self.treeview_produtos.delete(widget)
646
647         filtro = self.entry_pesquisa_produtos.get()
648
649         conexao = self.conectar_banco()
650         if not conexao:
651             return
652
653         cursor = conexao.cursor()
654         try:
655             query = "SELECT nome, valor FROM produtos WHERE nome LIKE %s"
656             cursor.execute(query, (f"%{filtro}%",))
657             produtos = cursor.fetchall()
658
659             for produto in produtos:
660                 self.treeview_produtos.insert("", "end", values=produto)
661         except Exception as e:
662             messagebox.showerror("Erro", f"Erro ao buscar produtos: {e}")
663         finally:
664             conexao.close()
665

```

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

```

666     def abrir_tela_alterar_info(self, id_cliente):
667         for widget in self.frame_direita.winfo_children():
668             widget.destroy()
669
670         tk.Label(self.frame_direita, text="Alterar Informações", bg="white", fg="#151e70", font=("Arial", 14, "bold")).pack(pady=10)
671
672         tk.Label(self.frame_direita, text="Novo Nome:").pack(pady=5)
673         entrada_nome = tk.Entry(self.frame_direita)
674         entrada_nome.pack(pady=5)
675
676         tk.Label(self.frame_direita, text="Nova Senha:").pack(pady=5)
677         entrada_senha = tk.Entry(self.frame_direita, show="*")
678         entrada_senha.pack(pady=5)
679
680     def processar_alteracao():
681         nome = entrada_nome.get()
682         senha = entrada_senha.get()
683
684         conexao = self.conectar_banco()
685         if not conexao:
686             return
687
688         try:
689             cursor = conexao.cursor()
690             if nome:
691                 cursor.execute("UPDATE clientes SET nome = %s WHERE id = %s", (nome, id_cliente))
692             if senha:
693                 cursor.execute("UPDATE usuarios SET senha = %s WHERE id = %s", (senha, id_cliente))
694             conexao.commit()
695             messagebox.showinfo("Sucesso", "Informações alteradas com sucesso!")
696         except Exception as e:
697             messagebox.showerror("Erro", f"Erro ao atualizar informações: {e}")
698         finally:
699             conexao.close()
700
701         tk.Button(self.frame_direita, text="Alterar", command=processar_alteracao, bg="#151e70", fg="white").pack(pady=20)
702

```

Função para alterar as informações do usuário cliente:

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

```

702
703     def run(self):
704         self.janela.mainloop()
705

```

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

```

706     if __name__ == "__main__":
707         sistema = SistemaLogin()
708         sistema.run()

```

Inicia o Sistema.