main.py

```
import os
import sys
import datetime # Importado para timestamp dos resumos
import uuid     # Importado para gerar IDs de resumo
import traceback
from utils import pdf_exporter

# --- Funções Auxiliares de Exibição no Terminal ---

def print_separator(char="=", length=60):
    """Imprime uma linha de separação no terminal."""
    print(char * length)


def colorize_text(text, color_code, bold=False):
    """
    Aplica cores e/ou negrito ao texto para exibição no terminal.
    Os códigos de cor são padrão ANSI.
    Ex: 31=vermelho, 32=verde, 33=amarelo, 34=azul, 35=magenta, 36=ciano, 37=branco.
    """
    bold_code = "\033[1m" if bold else ""
    color_start = f"\033[{color_code}m"
    color_end = "\033[0m" # Código para resetar a formatação
    return f"{bold_code}{color_start}{text}{color_end}"


# Adiciona o diretório raiz do projeto ao sys.path para garantir que as importações funcionem
# quando main.py é executado de qualquer subdiretório (embora geralmente seja da raiz).
project_root = os.path.abspath(os.path.dirname(__file__))
sys.path.insert(0, project_root)

# Importa módulos e configurações
from config.config import (
    DEFAULT_MODEL, TEMPERATURE, MAX_TOKENS_LIMIT,
HISTORY_MESSAGE_LIMIT,
    SYSTEM_MESSAGE, SUMMARY_INSTRUCTION_MESSAGE,
DEFAULT_SESSION_NAME,
    PDFS_DIR, COMMANDS, SUMMARY_MAX_TOKENS # SUMMARY_MAX_TOKENS
importado aqui
)
from utils import api_service, pdf_processor, session_manager, token_utils, pdf_exporter

# --- Variáveis de Estado Global ---
# Histórico de mensagens da sessão atual
chat_history = []
# Conteúdo do resumo do PDF atualmente ativo na API
active_api_summary_content = None
# Metadados do resumo do PDF atualmente ativo (ex: nome do arquivo original)
active_api_summary_metadata = None
# Nome da sessão de chat atualmente ativa
```

```
current_session_name = DEFAULT_SESSION_NAME

# --- Funções Auxiliares de Gerenciamento de Sessão ---

# --- Funções Auxiliares de Gerenciamento de Sessão ---

def load_session_state(session_name: str):
    """Carrega o estado de uma sessão."""
    global chat_history, active_api_summary_content, active_api_summary_metadata,
current_session_name

   # print(f"DEBUG: Tentando carregar sessão: {session_name}")
    loaded_data = session_manager.load_session(session_name)
    #print(f"DEBUG: Tipo de loaded_data: {type(loaded_data)}, Valor: {loaded_data}")

    if loaded_data: # Início do bloco principal 'if loaded_data'
        loaded_chat_history = loaded_data.get("chat_history", [])
       # print(f"DEBUG: Tipo de loaded_chat_history: {type(loaded_chat_history)}, Valor:
{loaded_chat_history}")

        loaded_active_api_summary_content = loaded_data.get("active_api_summary_content",
None)
        loaded_active_api_summary_metadata = loaded_data.get("active_api_summary_metadata",
None)

        # Início do bloco 'if' interno para ajustar o histórico de chat
        if (not loaded_chat_history or
            not isinstance(loaded_chat_history[0], dict) or # Esta é a linha onde o erro estava
            loaded_chat_history[0].get("role") != "system"):
            # Estas linhas são o corpo do 'if' interno. Devem estar indentadas.
            chat_history = [{"role": "system", "content": SYSTEM_MESSAGE}] + \
                           [m for m in loaded_chat_history if isinstance(m, dict) and m.get("role") !=
"system"]
        else: # 'else' do 'if' interno
            chat_history = loaded_chat_history
        # Fim do bloco 'if/else' interno

        # Estas linhas pertencem ao bloco principal 'if loaded_data'.
        # Elas devem estar na mesma indentação do 'if' interno acima.
        active_api_summary_content = loaded_active_api_summary_content
        active_api_summary_metadata = loaded_active_api_summary_metadata
        current_session_name = session_name
        #print(f"Sessão '{current_session_name}' carregada com sucesso.") # Mensagem de sucesso
aqui
        # Aprimoramento: Verifica se ambos, conteúdo E metadados, existem
        if active_api_summary_content and active_api_summary_metadata:
            print(f"Resumo ativo: {active_api_summary_metadata.get('original_filename', 'PDF
Desconhecido')}")
        else:
            print("Nenhum resumo de PDF ativo nesta sessão.")

    else: # 'else' do bloco principal 'if loaded_data', para quando a sessão não existe
```

```
        print(f"Sessão '{session_name}' não encontrada. Iniciando uma nova sessão.")
        chat_history = [{"role": "system", "content": SYSTEM_MESSAGE}]
        active_api_summary_content = None
        active_api_summary_metadata = None
        current_session_name = session_name
        session_manager.save_session(current_session_name, chat_history,
active_api_summary_content, active_api_summary_metadata)

# --- Funções Auxiliares de Gerenciamento de Sessão ---

def save_session_state():
    """Salva o estado atual da sessão."""
    session_manager.save_session(current_session_name, chat_history,
active_api_summary_content, active_api_summary_metadata)
    #print(f"Sessão '{current_session_name}' salva.")

# --- Funções de Comando ---

def handle_read_pdf(file_name: str):
    """Processa o comando /lerpdf."""
    global active_api_summary_content, active_api_summary_metadata, chat_history
    pdf_path = os.path.join(PDFS_DIR, file_name)
    print(f"Lendo PDF de '{pdf_path}'...")

    extracted_text = pdf_processor.extract_text_from_pdf(pdf_path)

    if not extracted_text:
        print("Erro: Não foi possível extrair texto do PDF ou o arquivo está vazio.")
        return

    print("Texto extraído. Gerando resumo do PDF via OpenAI API (isso pode levar um tempo)...")

    # Preparar mensagens para a API para resumir o PDF
    summary_prompt_messages = [
        {"role": "system", "content": "Você é um assistente especializado em criar resumos concisos e
objetivos de documentos. Seu objetivo é extrair as informações mais importantes e apresentá-las de
forma clara. Responda apenas com o resumo."},
        {"role": "user", "content": f"Resuma o seguinte texto de um documento PDF, focando nos
pontos chave e informações mais relevantes. Seja conciso e direto. Texto do PDF:\n\
n{extracted_text}"}
    ]

    # Contar tokens do prompt do resumo para evitar exceder o limite
    prompt_tokens = token_utils.count_tokens_in_messages(summary_prompt_messages)

    # Lógica de Truncamento Aprimorada:
    # Considera o limite de tokens do modelo menos uma margem para a resposta do resumo
    # e uma margem de segurança para o sistema/instrução (ex: 200 tokens).
    max_prompt_tokens_allowed = MAX_TOKENS_LIMIT - SUMMARY_MAX_TOKENS - 200

    if prompt_tokens > max_prompt_tokens_allowed:
```

```
        print(f"Aviso: O texto do PDF é muito longo ({prompt_tokens} tokens) para ser resumido no modelo atual.")
        print(f"Truncando o texto do PDF para caber no limite de {max_prompt_tokens_allowed} tokens...")

        # Truncar o texto do PDF
        encoded_text = token_utils.ENCODER.encode(extracted_text)

        # Garante que não tentamos fatiar mais do que o texto tem
        if len(encoded_text) > max_prompt_tokens_allowed:
            truncated_encoded_text = encoded_text[:max_prompt_tokens_allowed]
            extracted_text = token_utils.ENCODER.decode(truncated_encoded_text)
        else:
            # Caso o texto original já fosse menor que o limite permitido após margem,
            # mas ainda excedia o MAX_TOKENS_LIMIT total, isso não deve ocorrer com a lógica
            # de max_prompt_tokens_allowed, mas é uma salvaguarda.
            print("Erro inesperado durante o truncamento: texto não reduzido o suficiente.")
            return

        # Atualiza a mensagem do usuário com o texto truncado
        summary_prompt_messages[1]["content"] = f"Resuma o seguinte texto de um documento PDF, focando nos pontos chave e informações mais relevantes. Seja conciso e direto. Texto do PDF:\n\n{extracted_text}"

        # Recalcula prompt_tokens com o texto truncado
        prompt_tokens = token_utils.count_tokens_in_messages(summary_prompt_messages)
        print(f"Texto do PDF truncado. Novo tamanho do prompt: {prompt_tokens} tokens.")

    # Se mesmo após a tentativa de truncar ainda for muito grande, ou se a lógica
    # de limite não funcionou como esperado.
    if prompt_tokens > MAX_TOKENS_LIMIT - 100: # Pequena margem de segurança final
        print("Erro: O prompt do resumo ainda é muito grande mesmo após truncamento. Não é possível gerar resumo.")
        return

    # Chamada à API da OpenAI com o max_tokens corrigido
    summary_response = api_service.get_openai_completion(
        messages=summary_prompt_messages,
        model=DEFAULT_MODEL,
        temperature=TEMPERATURE,
        max_tokens=SUMMARY_MAX_TOKENS  # Usando a constante definida em config.py
    )

    if summary_response:
        active_api_summary_content = summary_response
        active_api_summary_metadata = {
            "original_filename": file_name,
            "timestamp": datetime.datetime.now().isoformat(),
            "summary_id": str(uuid.uuid4()) # Gera um ID temporário que será atualizado após salvar
        }
        # Salva o resumo no sistema de arquivos
```

```
        actual_summary_id = session_manager.save_pdf_summary(active_api_summary_content, active_api_summary_metadata)
        active_api_summary_metadata["summary_id"] = actual_summary_id # Atualiza com o ID real

        print(f"\nResumo gerado e definido como contexto ativo para a sessão '{current_session_name}'.")
        print("Você pode fazer perguntas sobre o PDF agora.")
        # Opcional: Adicionar uma mensagem ao histórico indicando que um resumo foi carregado
        chat_history.append({"role": "system", "content": f"Resumo do PDF '{file_name}' carregado e pronto para consultas."})
    else:
        print("Não foi possível gerar o resumo do PDF.")
        active_api_summary_content = None
        active_api_summary_metadata = None

def handle_new_session(args: list):
    """Processa o comando /nova_sessao."""
    global current_session_name
    if len(args) < 1:
        print("Uso: /nova_sessao <nome_da_sessao>")
        return

    save_session_state() # Salva a sessão atual antes de mudar
    new_session_name = args[0]
    # Cria uma nova sessão com estado inicial
    session_manager.save_session(new_session_name, {
        "chat_history": [{"role": "system", "content": SYSTEM_MESSAGE}],
        "active_api_summary_content": None,
        "active_api_summary_metadata": None
    })
    load_session_state(new_session_name)
    print(f"Sessão '{new_session_name}' criada e ativada.")

def handle_load_session(args: list):
    """Processa o comando /carregar_sessao."""
    if len(args) < 1:
        print("Uso: /carregar_sessao <nome_da_sessao>")
        return

    session_to_load = args[0]
    if session_manager.session_exists(session_to_load):
        save_session_state() # Salva a sessão atual antes de carregar outra
        load_session_state(session_to_load)
    else:
        print(f"Sessão '{session_to_load}' não encontrada.")

def handle_list_sessions():
    """Processa o comando /listar_sessoes."""
    sessions = session_manager.list_sessions()
    if sessions:
        print("\nSessões salvas:")
        for session in sessions:
```

```
            status = "(Atual)" if session == current_session_name else ""
            print(f"- {session} {status}")
    else:
        print("Nenhuma sessão salva.")

def handle_delete_session(args: list):
    """Processa o comando /excluir_sessao."""
    if len(args) < 1:
        print("Uso: /excluir_sessao <nome_da_sessao>")
        return

    session_to_delete = args[0]
    if session_to_delete == current_session_name:
        print("Não é possível excluir a sessão ativa. Mude para outra sessão ou crie uma nova
primeiro.")
        return

    if session_manager.session_exists(session_to_delete):
        session_manager.delete_session(session_to_delete)
        print(f"Sessão '{session_to_delete}' excluída com sucesso.")
    else:
        print(f"Sessão '{session_to_delete}' não encontrada.")

def handle_delete_summary(args: list):
    """Processa o comando /excluir_resumo."""
    global active_api_summary_content, active_api_summary_metadata, chat_history

    if len(args) < 1:
        print("Uso: /excluir_resumo <ID_do_resumo_ou_numero>")
        return

    summaries = session_manager.list_summaries()
    target_summary_id = None
    target_summary_filename = "Resumo Desconhecido" # Para mensagens de feedback

    try:
        # Tenta excluir por número na lista (ex: /excluir_resumo 1)
        index = int(args[0]) - 1
        if 0 <= index < len(summaries):
            target_summary_data = summaries[index]
            target_summary_id = target_summary_data['id']
            target_summary_filename = target_summary_data['filename']
        else:
            print(f"Número '{args[0]}' fora do intervalo. Use /listar_resumos para ver os números
válidos.")
            return
    except ValueError:
        # Tenta excluir por ID direto (ex: /excluir_resumo 7ae7cf17-d96d-4640-872e-22ec38bbaf4c)
        input_id = args[0]
        found_by_id = False
        for s in summaries:
            if s['id'] == input_id:
```

```
                target_summary_id = s['id']
                target_summary_filename = s['filename']
                found_by_id = True
                break
        if not found_by_id:
            print(f"ID de resumo '{input_id}' não encontrado.")
            return

    if target_summary_id:
        # Se o resumo a ser excluído é o resumo ativo, descarrega-o primeiro
        if active_api_summary_metadata and active_api_summary_metadata.get('summary_id') ==
target_summary_id:
            active_api_summary_content = None
            active_api_summary_metadata = None
            print(f"O resumo ativo ('{target_summary_filename}') foi descarregado antes da exclusão.")

        if session_manager.delete_pdf_summary(target_summary_id):
            print(f"Resumo '{target_summary_filename}' (ID: {target_summary_id}) excluído com
sucesso.")
            save_session_state() # Salva o estado da sessão após a exclusão para refletir a remoção
        else:
            print(f"Não foi possível excluir o resumo com ID '{target_summary_id}'.")

# IMPORTANTE: Certifique-se de que 'os' esteja importado no topo do main.py
import os

# ... (suas funções handle_existentes, por exemplo, após handle_delete_summary ou
handle_clear_context)

def handle_export_chat(args: list):
    """
    Exporta o histórico do chat atual para um arquivo PDF.
    Uso: /exportar_chat <nome_do_export>
    """
    if not args:
        print(f"Uso: {COMMANDS['export_chat']} <nome_do_export>. Por favor, forneça um nome
para o arquivo PDF.")
        return

    export_name = args[0] # O nome do export é o primeiro argumento
    if not export_name:
        print("O nome do export não pode estar vazio. Por favor, forneça um nome válido.")
        return

    print(f"Exportando chat da sessão '{current_session_name}' para PDF '{export_name}.pdf'...")
    # A função export_chat_to_pdf espera o histórico completo da sessão
    # E o nome da sessão para organizar os arquivos
    result = pdf_exporter.export_chat_to_pdf(chat_history, current_session_name, export_name)
    print(result)

def handle_list_exports(args: list): # 'args' para possível filtro futuro
    """
```

```
    Lista os PDFs de chat exportados para a sessão atual.
    Uso: /listar_exports
    """
    session_to_list = current_session_name # Por padrão, lista da sessão atual
    if args:
        # Se um argumento for fornecido, tente listar para essa sessão
        session_to_list = args[0]
        print(f"Listando exports para a sessão: '{session_to_list}'...")
    else:
        print(f"Listando exports para a sessão atual: '{session_to_list}'...")

    exports = pdf_exporter.list_exported_pdfs(session_to_list)

    if exports:
        print(f"\nExports disponíveis para a sessão '{session_to_list}':")
        for i, exp in enumerate(exports):
            # O exp retornado é apenas o nome do arquivo, ex: "MeuEstudo.pdf"
            print(f"  {i+1}. {exp}")
    else:
        print(f"Não há exports de chat para a sessão '{session_to_list}'.")
        if args and not os.path.exists(os.path.join(EXPORTS_DIR, session_to_list)):
            print(f"O diretório para a sessão '{session_to_list}' não existe em {EXPORTS_DIR}.")


def handle_load_export(args: list):
    """
    'Carrega' (indica o caminho de) um PDF de chat exportado.
    Uso: /carregar_export <nome_do_export>
    """
    if not args:
        print(f"Uso: {COMMANDS['load_export']} <nome_do_export>. Por favor, forneça o nome do
arquivo PDF.")
        return

    export_name = args[0]
    # Adiciona a extensão .pdf se não estiver presente para a busca
    if not export_name.lower().endswith(".pdf"):
        export_name += ".pdf"

    # Tenta obter o caminho completo e verifica se existe
    full_path = pdf_exporter.get_exported_pdf_path(current_session_name, export_name)

    if full_path and os.path.exists(full_path):
        print(f"PDF de exportação '{export_name}' encontrado em: {full_path}")
        print("Você pode acessar este arquivo diretamente para visualização ou compartilhamento.")
    else:
        print(f"PDF de exportação '{export_name}' não encontrado para a sessão
'{current_session_name}'.")
        print("Verifique se o nome está correto e se ele foi exportado para esta sessão.")
        handle_list_exports([]) # Sugere listar os exports existentes
```

```
def handle_delete_export(args: list):
    """
    Exclui um PDF de chat exportado.
    Uso: /excluir_export <nome_do_export>
    """
    if not args:
        print(f"Uso: {COMMANDS['delete_export']} <nome_do_export>. Por favor, forneça o nome
do arquivo PDF a ser excluído.")
        return

    export_name = args[0]
    # Adiciona a extensão .pdf se não estiver presente para a busca e exclusão
    if not export_name.lower().endswith(".pdf"):
        export_name += ".pdf"

    print(f"Tentando excluir o export '{export_name}' da sessão '{current_session_name}'...")
    success = pdf_exporter.delete_exported_pdf(current_session_name, export_name)

    if success:
        print(f"Export '{export_name}' excluído com sucesso da sessão '{current_session_name}'.")
    else:
        print(f"Falha ao excluir o export '{export_name}'. O arquivo pode não existir ou houve um
erro.")
        handle_list_exports([]) # Sugere listar os exports existentes para ajudar

def handle_list_summaries():
    print("Resumos de PDF salvos:")
    summaries = session_manager.list_summaries() # Esta linha já está correta
    if summaries:
        # A linha abaixo precisa estar EXATAMENTE assim para formatar a saída
        for i, summary in enumerate(summaries, 1):
            print(f"{i}. ID: {summary['id']} | Arquivo Original: {summary['filename']} | Data:
{summary['timestamp']}")
    else:
        print("Nenhum resumo de PDF encontrado.")

def handle_load_summary(args: list):
    """Processa o comando /carregar_resumo."""
    global active_api_summary_content, active_api_summary_metadata, chat_history
    if len(args) < 1:
        print("Uso: /carregar_resumo <ID_do_resumo_ou_numero>")
        return

    # Chama a função CORRETA em session_manager e guarda os resumos disponíveis
    available_summaries = session_manager.list_summaries() # CORRIGIDO AQUI

    summary_to_load_id = None
    summary_filename_for_display = "PDF Desconhecido" # Usado para mensagens de feedback

    try:
        # Tenta carregar o resumo pelo número na lista exibida
        index = int(args[0]) - 1
```

```
        if 0 <= index < len(available_summaries):
            # Extrai o dicionário completo do resumo e obtém o ID e nome do arquivo
            summary_data_from_list = available_summaries[index]
            summary_to_load_id = summary_data_from_list['id']
            summary_filename_for_display = summary_data_from_list.get('filename',
summary_filename_for_display)
        else:
            print(f"Número '{args[0]}' fora do intervalo. Use /listar_resumos para ver os números
válidos.")
            return
    except ValueError:
        # Se o argumento não for um número, tenta carregar o resumo por ID (string)
        input_id_string = args[0]
        found_by_id = False
        for s_data in available_summaries: # Itera sobre os dicionários de resumos disponíveis
            if s_data['id'] == input_id_string: # Compara o ID fornecido com o ID de cada resumo
                summary_to_load_id = s_data['id']
                summary_filename_for_display = s_data.get('filename', summary_filename_for_display)
                found_by_id = True
                break

        if not found_by_id:
            print(f"ID de resumo '{input_id_string}' não encontrado.")
            return

    if summary_to_load_id:
        # Se um ID válido (seja por número ou string) foi encontrado, tenta carregar o resumo
        summary_data = session_manager.load_specific_pdf_summary(summary_to_load_id)
        if summary_data:
            active_api_summary_content = summary_data.get("content")
            active_api_summary_metadata = summary_data.get("metadata")

            # Usa o nome do arquivo dos metadados para exibição, se disponível
            display_filename = active_api_summary_metadata.get('original_filename',
summary_filename_for_display)

            print(f"\nResumo '{display_filename}' (ID: {summary_to_load_id}) carregado como
contexto ativo.")
            # Adiciona mensagem ao histórico do chat sobre o resumo carregado
            chat_history.append({"role": "system", "content": f"Resumo '{display_filename}' carregado
e pronto para consultas."})
            save_session_state() # Salva o estado da sessão com o novo resumo ativo
        else:
            print(f"Não foi possível carregar o resumo com ID '{summary_to_load_id}'.")

def handle_clear_context():
    """Processa o comando /limpar."""
    global chat_history # Agora, apenas chat_history é declarado como global para modificação
    #global chat_history, active_api_summary_content, active_api_summary_metadata
    chat_history = [{"role": "system", "content": SYSTEM_MESSAGE}] # Reseta histórico,
mantendo mensagem do sistema
    #active_api_summary_content = None
```

```
    #active_api_summary_metadata = None
    print("Histórico de chat limpo para a sessão atual. O resumo de PDF ativo foi mantido.")

def handle_help():
    """Exibe a lista de comandos disponíveis."""
    print("\nComandos disponíveis:")
    for cmd_name, cmd_value in COMMANDS.items():
        print(f"- {cmd_value}: {cmd_name.replace('_', ' ').capitalize()}")
    print("- /sair: Salva a sessão atual e encerra o chatbot.")
    print("\nPara fazer perguntas normais, basta digitar sua mensagem.")
    if active_api_summary_content:
        print("Lembre-se: Há um resumo de PDF ativo. Suas perguntas serão respondidas com base
nele.")

# --- Loop Principal do Chatbot ---

def run_chatbot():
    """Inicia e executa o loop principal do chatbot."""
    # Carrega o estado inicial da sessão padrão
    load_session_state(DEFAULT_SESSION_NAME)

    print_separator()
    print("Bem-vindo ao Chatbot de Consulta de PDFs!")
    print("Digite suas perguntas ou um comando (ex: /ajuda para ver os comandos).")

    while True:
        try:
            print_separator()
            user_input = input(f"\n[{current_session_name}] {colorize_text('VOCÊ', 34, bold=True)}:
") # Azul negrito
            if not user_input.strip(): # Não processa entrada vazia
                continue

            # Processar comandos
            if user_input.startswith('/'):
                parts = user_input.split(maxsplit=1)
                command = parts[0].lower()
                args = parts[1].split() if len(parts) > 1 else []

                if command == COMMANDS["read_pdf"]:
                    if args:
                        handle_read_pdf(args[0])
                    else:
                        print("Uso: /lerpdf <nome_do_arquivo.pdf>")
                elif command == COMMANDS["new_session"]:
                    handle_new_session(args)
                elif command == COMMANDS["load_session"]:
                    handle_load_session(args)
                elif command == COMMANDS["list_sessions"]:
                    handle_list_sessions()
                elif command == COMMANDS["delete_session"]:
                    handle_delete_session(args)
```

```
            elif command == COMMANDS["list_summaries"]:
                handle_list_summaries()
            elif command == COMMANDS["load_summary"]:
                handle_load_summary(args)
            elif command == COMMANDS["delete_summary"]:
                handle_delete_summary(args)
            elif command == COMMANDS["clear_context"]:
                handle_clear_context()
            elif command == COMMANDS["help"]:
                handle_help()
            elif command == COMMANDS["exit"]:
                # Inicia um loop para garantir uma resposta válida (s/n)
                export_decision_made = False
                while not export_decision_made:
                    user_choice = input("\nDeseja exportar o chat atual para PDF antes de sair? (s/n): ").lower().strip()

                    if user_choice == 's':
                        export_name = input("Por favor, digite um nome para o arquivo PDF (ex: MinhaConversaImportante): ").strip()
                        if export_name:
                            result = pdf_exporter.export_chat_to_pdf(chat_history, current_session_name, export_name)
                            print(result)
                        else:
                            print_separator()
                            print("Nome de exportação vazio. O chat não será exportado.")
                            print_separator()
                        export_decision_made = True # Sai do loop de decisão
                    elif user_choice == 'n':
                        print_separator()
                        print("Chat não exportado.")
                        export_decision_made = True # Sai do loop de decisão
                    else:
                        # Se a opção for inválida, exibe a mensagem e o loop continua
                        print_separator()
                        print("Opção inválida. Por favor, digite 's' para sim ou 'n' para não.")
                        print_separator()

                # Este bloco só é executado após uma decisão válida (s ou n) ser tomada
                print_separator()
                save_session_state()
                print("Saindo do chatbot. Até mais!")
                break # Este 'break' sai do loop principal do chatbot
            # --- NOVOS COMANDOS DE EXPORTAÇÃO ---
            elif command == COMMANDS["export_chat"]:
                handle_export_chat(args)
            elif command == COMMANDS["list_exports"]:
                handle_list_exports(args)
            elif command == COMMANDS["load_export"]:
                handle_load_export(args)
            elif command == COMMANDS["delete_export"]:
```

```
                handle_delete_export(args)
            # --- FIM DOS NOVOS COMANDOS ---
            else:
                print(f"Comando '{command}' não reconhecido. Digite /ajuda para ver os comandos.")
        else:
            # Se não for um comando, é uma pergunta ao chatbot
            user_message = {"role": "user", "content": user_input}

            # Preparar mensagens para a API
            messages_for_api = []

            # Incluir mensagem de instrução de resumo se houver um resumo ativo
            if active_api_summary_content:
                summary_context_message = {
                    "role": "user",
                    "content": f"{SUMMARY_INSTRUCTION_MESSAGE}\n\nResumo do
Documento:\n{active_api_summary_content}"
                }
                messages_for_api.append(summary_context_message)

            # Adicionar histórico de chat limitado (exceto a mensagem de sistema inicial, que já está
em chat_history[0])
            # Limita para HISTORY_MESSAGE_LIMIT mensagens do usuário/assistente mais a
mensagem do sistema
            messages_for_api.append(chat_history[0]) # Mensagem do sistema
            messages_for_api.extend(chat_history[max(1, len(chat_history) -
HISTORY_MESSAGE_LIMIT):])

            # Adicionar a mensagem atual do usuário
            messages_for_api.append(user_message)

            # Contar tokens do prompt antes de enviar à API
            api_prompt_tokens = token_utils.count_tokens_in_messages(messages_for_api)

            # Se o prompt for muito longo, alertar e não enviar
            if api_prompt_tokens > MAX_TOKENS_LIMIT:
                print(f"Aviso: Sua pergunta e o contexto (histórico/resumo) excedem o limite de tokens
do modelo ({api_prompt_tokens} tokens). Por favor, limpe o contexto (/limpar) ou faça uma
pergunta mais curta.")
                continue
            print_separator()
            print("Gerando resposta (isso pode levar um tempo)...")
            # max_tokens para a resposta da IA em conversas normais pode ser
DEFAULT_MAX_TOKENS_RESPONSE
            # que podemos adicionar ao config.py, ou deixar a API definir.
            # Aqui, não estamos limitando explicitamente a resposta para conversas gerais,
            # a menos que o DEFAULT_MODEL já tenha um limite de saída implicito.
            bot_response = api_service.get_openai_completion(
                messages=messages_for_api,
                model=DEFAULT_MODEL,
                temperature=TEMPERATURE
            )
```

```
            if bot_response:
                print_separator()
                print(f"[{current_session_name}] {colorize_text('BOT', 36, bold=True)}:
{colorize_text(bot_response, 36, bold=False)}")
                chat_history.append(user_message) # Adiciona a pergunta do usuário
                chat_history.append({"role": "assistant", "content": bot_response}) # Adiciona a
resposta do bot
                save_session_state() # Salva a sessão após cada interação
            else:
                print_separator()
                print("Não foi possível obter uma resposta do chatbot.")

        except KeyboardInterrupt:
            print("\nEncerrando o chatbot.")
            save_session_state()
            break
        except Exception as e:
            print(f"Ocorreu um erro inesperado: {e}")
            traceback.print_exc()
            # Para depuração, você pode adicionar um traceback mais detalhado:
            # import traceback
            # traceback.print_exc()

if __name__ == "__main__":
    run_chatbot()
```

config.py

```
import os

# --- Configurações Gerais do Chatbot ---
# Modelo padrão da OpenAI a ser usado.
# Considere "gpt-4o-mini" para melhor custo-benefício ou "gpt-3.5-turbo" para mais velocidade.
# Modelos disponíveis em: https://platform.openai.com/docs/models/overview
DEFAULT_MODEL = "gpt-4o-mini"

# Criatividade da resposta do modelo (0.0 a 2.0).
# Valores mais baixos são mais focados e determinísticos, valores mais altos são mais criativos.
TEMPERATURE = 0.8

# --- Limites de Contexto e Tokens ---
# Limite máximo de tokens de entrada para o modelo (incluindo System, Histórico e Pergunta).
# Consulte a documentação da OpenAI para os limites específicos do modelo escolhido.
# gpt-3.5-turbo (16k) -> 16385 tokens
# gpt-4o-mini (128k) -> 128000 tokens
MAX_TOKENS_LIMIT = 100000 # Um pouco abaixo do limite do gpt-4o-mini para segurança

# Limite de tokens para a resposta gerada pela IA ao resumir PDFs
SUMMARY_MAX_TOKENS = 1500 # Um valor razoável para a maioria dos resumos. Ajuste conforme necessário.

# Número máximo de mensagens do histórico de chat a serem incluídas na requisição da API.
# Isso ajuda a controlar o uso de tokens e manter o contexto relevante.
# Cada mensagem é um par (user, assistant).
HISTORY_MESSAGE_LIMIT = 15 # Limita o histórico a 10 pares (20 mensagens individuais)

# --- Caminhos de Arquivo e Diretórios ---
# Caminho base para o diretório de dados, relativo ao diretório raiz do projeto.
BASE_DATA_DIR = os.path.join(os.path.dirname(os.path.dirname(__file__)), 'data')

# Caminho para o diretório de perfis de usuário.
PROFILES_DIR = os.path.join(BASE_DATA_DIR, 'profiles')

# Caminho para o diretório do perfil padrão (onde sessões e resumos serão salvos).
DEFAULT_PROFILE_DIR = os.path.join(PROFILES_DIR, 'default')

# Caminho para o diretório onde as sessões de chat são salvas.
SESSIONS_DIR = os.path.join(DEFAULT_PROFILE_DIR, 'sessions')

# Caminho para o diretório onde os resumos de PDF são salvos.
SUMMARIES_DIR = os.path.join(DEFAULT_PROFILE_DIR, 'summaries')

# Caminho para o diretório onde os PDFs de interações exportadas serão salvos.
EXPORTS_DIR = os.path.join(DEFAULT_PROFILE_DIR, 'exports')

# Caminho para o diretório onde os PDFs originais são armazenados.
PDFS_DIR = os.path.join(os.path.dirname(os.path.dirname(__file__)), 'pdfs')
```

```
# --- Mensagens Padrão do Sistema ---
# Mensagem de sistema que define o comportamento geral do chatbot.
# Ajuste para definir o "persona" do seu assistente.
SYSTEM_MESSAGE = (
    "Você é um assistente de IA prestativo e conciso, otimizado para responder perguntas "
    "com base em documentos PDF fornecidos. Seja objetivo, direto e sempre baseie suas "
    "respostas no contexto mais recente. Se não souber a resposta com base no contexto, "
    "indique isso. Priorize a eficiência e o uso mínimo de tokens."
)

# Mensagem de instrução que é anexada antes do conteúdo do resumo ativo.
# Isso orienta o modelo sobre como usar o resumo.
SUMMARY_INSTRUCTION_MESSAGE = (
    "O texto a seguir é um resumo de um documento PDF. Utilize-o como seu "
    "principal contexto para responder a perguntas futuras. Priorize "
    "as informações deste resumo ao gerar suas respostas. Se a pergunta "
    "não puder ser respondida com base neste resumo, indique isso. "
    "Resumo do PDF:\n\n"
)

# Nome da sessão padrão que será carregada ou criada ao iniciar o chatbot.
DEFAULT_SESSION_NAME = "default_session"

# --- Comandos do Chatbot ---
# Dicionário de comandos para facilitar a referência e futura expansão.
# Em config/config.py

# --- Comandos do Chatbot ---
# --- Comandos do Chatbot ---
# Dicionário de comandos para facilitar a referência e futura expansão.
COMMANDS = {
    "read_pdf": "/lerpdf",
    "new_session": "/nova_sessao",
    "load_session": "/carregar_sessao", # Mantido como você tem
    "list_sessions": "/listar_sessoes",
    "delete_session": "/excluir_sessao",
    "list_summaries": "/listar_resumos",
    "load_summary": "/carregar_resumo",
    "clear_context": "/limpar",       # Mantido como você tem
    "exit": "/sair",
    "help": "/ajuda",
    "delete_summary": "/excluir_resumo",
    # --- NOVOS COMANDOS DE EXPORTAÇÃO ---
    "export_chat": "/exportar_chat",
    "list_exports": "/listar_exports",
    "delete_export": "/excluir_export",
    # --- FIM DOS NOVOS COMANDOS ---
}
```

api_service.py

```
import os
import sys
from openai import OpenAI
from openai import RateLimitError, APIConnectionError, OpenAIError

# Adiciona o diretório raiz do projeto ao sys.path para permitir importações absolutas
# quando o módulo é executado diretamente.
project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir))
sys.path.insert(0, project_root)

from config.config import DEFAULT_MODEL, TEMPERATURE

# Carrega a chave da API do arquivo .env
# Verifica se a chave OPENAI_API_KEY está definida como variável de ambiente.
# Se não estiver, tenta carregá-la do arquivo .env.
if "OPENAI_API_KEY" not in os.environ:
    try:
        from dotenv import load_dotenv
        load_dotenv()
        print("Chave de API OpenAI carregada do arquivo .env")
    except ImportError:
        print("Aviso: python-dotenv não está instalado. Não foi possível carregar variáveis do .env.")
        print("Por favor, instale com 'pip install python-dotenv' ou defina OPENAI_API_KEY
manualmente.")

API_KEY = os.getenv("OPENAI_API_KEY")

if not API_KEY:
    raise ValueError("A chave OPENAI_API_KEY não foi encontrada. Por favor, defina-a no seu
ambiente ou no arquivo .env.")

client = OpenAI(api_key=API_KEY)

def get_openai_completion(messages: list, model: str = DEFAULT_MODEL, temperature: float =
TEMPERATURE, max_tokens: int = None) -> str:
    """
    Obtém uma resposta do modelo de linguagem da OpenAI.

    Args:
        messages (list): Uma lista de dicionários de mensagens para enviar à API.
                         Ex: [{"role": "system", "content": "You are a helpful assistant."},
                              {"role": "user", "content": "Hello!"}]
        model (str): O nome do modelo da OpenAI a ser usado (ex: "gpt-3.5-turbo").
        temperature (float): A temperatura para controlar a criatividade da resposta (0.0 a 2.0).
        max_tokens (int, optional): O número máximo de tokens na resposta gerada.
                                    Se None, a API usará seu padrão.

    Returns:
        str: A resposta de texto do modelo, ou uma string vazia se houver um erro.
    """
```

```
    try:
        # Constrói o dicionário de argumentos para a chamada da API
        completion_args = {
            "model": model,
            "messages": messages,
            "temperature": temperature,
        }
        # Adiciona max_tokens apenas se for fornecido (não None)
        if max_tokens is not None:
            completion_args["max_tokens"] = max_tokens

        chat_completion = client.chat.completions.create(**completion_args)
        return chat_completion.choices[0].message.content
    except RateLimitError:
        print("Erro de limite de taxa da OpenAI: Muitas requisições. Por favor, espere um pouco.")
    except APIConnectionError as e:
        print(f"Erro de conexão com a API da OpenAI: {e}")
        print("Verifique sua conexão com a internet ou a URL da API.")
    except OpenAIError as e:
        print(f"Erro da API OpenAI: {e}")
        print("Verifique sua chave de API ou se há algum problema com o serviço da OpenAI.")
    except Exception as e:
        print(f"Ocorreu um erro inesperado ao chamar a API: {e}")
    return ""

if __name__ == "__main__":
    print("Testando api_service.py...")

    # Teste básico
    test_messages = [
        {"role": "system", "content": "Você é um assistente prestativo."},
        {"role": "user", "content": "Qual a capital do Brasil?"}
    ]

    print("\nTestando get_openai_completion sem max_tokens:")
    response = get_openai_completion(test_messages)
    print("Resposta da API:", response)

    # Teste com max_tokens (resposta curta esperada)
    test_messages_short = [
        {"role": "system", "content": "Responda de forma extremamente concisa."},
        {"role": "user", "content": "Quem descobriu o Brasil?"}
    ]
    print("\nTestando get_openai_completion com max_tokens=10:")
    response_short = get_openai_completion(test_messages_short, max_tokens=10)
    print("Resposta da API (curta):", response_short)

    # Teste de erro (ex: modelo inválido)
    print("\nTestando com modelo inválido (deve gerar erro):")
    error_response = get_openai_completion(test_messages, model="modelo_invalido_xyz")
    if not error_response:
```

```
        print("Teste de erro bem-sucedido: Não houve resposta (como esperado para modelo inválido).")
```

requirements.txt

```
python-dotenv
openai
pdfplumber
tiktoken
PyPDF2
fpdf2
```

pdf_exporter.py

```
import os
import sys
import json
import datetime
from fpdf import FPDF # Importa a classe FPDF
from typing import Union # Adicionado para 'Union' na delete_exported_pdf
# Adiciona o diretório raiz do projeto ao sys.path para permitir importações absolutas
# quando o módulo é executado diretamente ou como parte do projeto maior.
project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir))
sys.path.insert(0, project_root)

# Importa as configurações de caminhos do config.py
from config.config import EXPORTS_DIR, DEFAULT_SESSION_NAME # Importamos
EXPORTS_DIR e DEFAULT_SESSION_NAME

# --- Funções Auxiliares ---
def _ensure_dir_exists(directory_path: str):
    """Garante que um diretório exista. Se não existir, ele é criado."""
    os.makedirs(directory_path, exist_ok=True)

def _get_export_path(session_name: str, export_name: str) -> str:
    """
    Retorna o caminho completo para o arquivo PDF exportado dentro da sessão.
    Os exports são organizados em subdiretórios por nome de sessão.
    """
    session_export_dir = os.path.join(EXPORTS_DIR, session_name)
    _ensure_dir_exists(session_export_dir)
    return os.path.join(session_export_dir, export_name) # Removido o ".pdf" extra aqui

# --- Funções de Gerenciamento de Exportação de Chat ---
def export_chat_to_pdf(chat_history: list, session_name: str, export_name: str) -> str:
    """
    Exporta o histórico completo de um chat para um arquivo PDF.
    """
    #print(f"DEBUG: export_chat_to_pdf iniciado. session_name: {session_name}, export_name:
{export_name}") # Print de depuração

    if not chat_history:
        #print("DEBUG: Histórico do chat vazio.") # Print de depuração
        return "O histórico do chat está vazio. Nada para exportar."

    # ESTA É A LINHA CRUCIAL QUE PRECISA SER ADICIONADA OU VERIFICADA!
    # Ela limpa o nome do export e atribui à variável base_name_cleaned.
    base_name_cleaned = "".join(c for c in export_name if c.isalnum() or c in (' ', '_', '-')).strip()
    #print(f"DEBUG: base_name_cleaned após limpeza inicial: '{base_name_cleaned}'") # Print de
depuração

    # 1. Garante que o nome base não tenha .pdf extra
    # Agora, base_name_cleaned EXISTE ANTES DE SER USADA aqui.
    if base_name_cleaned.lower().endswith(".pdf"):
```

```
        #print(f"DEBUG: '{base_name_cleaned}' termina com .pdf, removendo a extensão.") # Print de depuração
        base_name_cleaned = base_name_cleaned[:-4] # Remove a última ".pdf"
        #print(f"DEBUG: base_name_cleaned após remover .pdf extra: '{base_name_cleaned}'") # Print de depuração

    # Adicione uma verificação para nome vazio após a limpeza
    if not base_name_cleaned:
        #print("DEBUG: base_name_cleaned está vazia após a limpeza. Retornando erro.") # Print de depuração
        return "Erro: O nome do arquivo exportado não pode ser vazio ou conter apenas caracteres inválidos após a limpeza."

    # 2. Adiciona o timestamp para garantir um nome de arquivo único
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    #print(f"DEBUG: Timestamp gerado: {timestamp}") # Print de depuração

    # 3. Constrói o nome de arquivo final, com o nome limpo, timestamp e UMA extensão .pdf
    final_export_filename = f"{base_name_cleaned}_{timestamp}.pdf"
    #print(f"DEBUG: Nome final do arquivo exportado: '{final_export_filename}'") # Print de depuração

    # 4. Chama _get_export_path com o nome de arquivo FINAL e COMPLETO
    pdf_path = _get_export_path(session_name, final_export_filename)
    #print(f"DEBUG: Caminho completo para o PDF: '{pdf_path}'") # Print de depuração

    try:
        pdf = FPDF()
        pdf.set_auto_page_break(auto=True, margin=15) # Adicionado para melhor quebra de página
        pdf.add_page()
        pdf.set_font("Arial", size=12)

        pdf.multi_cell(0, 10, f"Documentação da Interação - Sessão: {session_name}\n")
        pdf.multi_cell(0, 10, f"Nome do Export: {final_export_filename}\n\n")

        # Itera sobre o histórico do chat
        for message in chat_history:
            role = message.get("role", "unknown").capitalize()
            content = message.get("content", "")

            # Ignora a mensagem do sistema para a exportação, pois ela é fixa e não parte da interação dinâmica
            if role.lower() == "system":
                continue

            # Formatação melhorada para usuário e assistente
            if role == "User":
                pdf.set_font("Arial", "B", 12) # Negrito para usuário
                pdf.write(5, f"Você: ")
                pdf.set_font("Arial", size=12) # Volta à fonte normal
                pdf.multi_cell(0, 5, content)
            elif role == "Assistant":
```

```
                pdf.set_font("Arial", "B", 12) # Negrito para bot
                pdf.write(5, f"Bot: ")
                pdf.set_font("Arial", size=12) # Volta à fonte normal
                pdf.multi_cell(0, 5, content)
            else: # Para outros papéis desconhecidos
                pdf.multi_cell(0, 5, f"{role}: {content}")

            pdf.ln(2) # Pequena quebra de linha entre mensagens para melhor leitura

        pdf.output(pdf_path)
        #print(f"DEBUG: PDF gerado com sucesso em '{pdf_path}'") # Print de depuração
        return f"Histórico do chat exportado com sucesso para: {pdf_path}"
    except Exception as e:
        import traceback # Para depuração
        traceback.print_exc() # Mostra o erro completo no console para depuração
        #print(f"DEBUG: Erro na exportação do PDF: {e}") # Print de depuração
        return f"Erro ao exportar chat para PDF: {e}"

def list_exported_pdfs(session_name: str = None) -> list:
    """
    Lista todos os PDFs de interações exportados.
    Se session_name for fornecido, lista apenas os exports daquela sessão.
    """
    _ensure_dir_exists(EXPORTS_DIR) # Garante que o diretório base exista

    exported_files = []
    if session_name:
        # Lista exports para uma sessão específica
        session_export_dir = os.path.join(EXPORTS_DIR, session_name)
        if os.path.exists(session_export_dir):
            for filename in os.listdir(session_export_dir):
                if filename.endswith(".pdf"):
                    exported_files.append(f"{session_name}/{filename}")
    else:
        # Lista todos os exports de todas as sessões
        for session_dir in os.listdir(EXPORTS_DIR):
            session_path = os.path.join(EXPORTS_DIR, session_dir)
            if os.path.isdir(session_path):
                for filename in os.listdir(session_path):
                    if filename.endswith(".pdf"):
                        exported_files.append(f"{session_dir}/{filename}")
    return exported_files

def get_exported_pdf_path(session_name: str, export_name: str) -> str:
    """
    Retorna o caminho completo de um PDF de interação exportado específico.
    """
    #print(f"DEBUG (get_exported_pdf_path): Tentando obter caminho para
sessão='{session_name}', export_name='{export_name}'")
    pdf_path = _get_export_path(session_name, export_name)
    #print(f"DEBUG (get_exported_pdf_path): Caminho construído: '{pdf_path}'")
    if os.path.exists(pdf_path):
```

```
        #print(f"DEBUG (get_exported_pdf_path): Arquivo encontrado em: '{pdf_path}'")
        return pdf_path
    #print(f"DEBUG (get_exported_pdf_path): Arquivo NÃO encontrado em: '{pdf_path}'")
    return None

def delete_exported_pdf(session_name: str, export_name: str) -> bool:
    """
    Exclui um PDF de interação exportado específico.
    """
    #print(f"DEBUG (delete_exported_pdf): Tentando excluir para sessão='{session_name}',
export_name='{export_name}'")
    pdf_path = _get_export_path(session_name, export_name)
    #print(f"DEBUG (delete_exported_pdf): Caminho construído para exclusão: '{pdf_path}'")
    if os.path.exists(pdf_path):
        #print(f"DEBUG (delete_exported_pdf): Arquivo encontrado para exclusão em: '{pdf_path}'")
        try:
            os.remove(pdf_path)
            #print(f"DEBUG (delete_exported_pdf): Arquivo removido: '{pdf_path}'")
            # Tenta remover o diretório da sessão se estiver vazio
            session_export_dir = os.path.join(EXPORTS_DIR, session_name)
            if not os.listdir(session_export_dir): # Verifica se o diretório está vazio
                #print(f"DEBUG (delete_exported_pdf): Diretório da sessão vazio, tentando remover:
'{session_export_dir}'")
                os.rmdir(session_export_dir)
                #print(f"DEBUG (delete_exported_pdf): Diretório da sessão removido:
'{session_export_dir}'")
            return True
        except Exception as e:
            #print(f"DEBUG (delete_exported_pdf): Erro ao excluir PDF de exportação: {e}")
            print(f"Erro ao excluir PDF de exportação: {e}")
            return False
        #print(f"DEBUG (delete_exported_pdf): Arquivo NÃO encontrado para exclusão em:
'{pdf_path}'")
    return False

# --- Bloco de Teste (apenas para depuração do módulo) ---
if __name__ == "__main__":
    print("Testando pdf_exporter.py...")

    # Certifica-se que o diretório de exports existe para o teste
    _ensure_dir_exists(EXPORTS_DIR)

    test_session = "TESTE_SESSAO_EXPORT"
    test_export_name = "Interacao_Exemplo"
    test_export_name_2 = "Outra_Interacao"

    # Simula um histórico de chat
    test_chat_history = [
        {"role": "system", "content": "Você é um assistente de teste."},
        {"role": "user", "content": "Olá, bot de teste! Como você está?"},
        {"role": "assistant", "content": "Olá! Estou funcionando perfeitamente."},
        {"role": "user", "content": "Pode me falar sobre a fpdf2?"},
```

```
        {"role": "assistant", "content": "A fpdf2 é uma biblioteca Python para geração de PDFs de forma simples e eficiente."},
    ]

    print("\n--- Teste de Exportação ---")
    export_result = export_chat_to_pdf(test_chat_history, test_session, test_export_name)
    print(export_result)

    export_result_2 = export_chat_to_pdf([{"role": "user", "content": "Mais um teste"}], test_session, test_export_name_2)
    print(export_result_2)

    print("\n--- Teste de Listagem (sessão específica) ---")
    listed_for_session = list_exported_pdfs(test_session)
    print(f"Exports para '{test_session}': {listed_for_session}")

    print("\n--- Teste de Listagem (todos os exports) ---")
    all_listed = list_exported_pdfs()
    print(f"Todos os Exports: {all_listed}")

    print("\n--- Teste de Obter Caminho ---")
    path_found = get_exported_pdf_path(test_session, test_export_name)
    print(f"Caminho do '{test_export_name}': {path_found}")

    print("\n--- Teste de Exclusão ---")
    if delete_exported_pdf(test_session, test_export_name):
        print(f"'{test_export_name}.pdf' excluído com sucesso.")
    else:
        print(f"Falha ao excluir '{test_export_name}.pdf' ou não encontrado.")

    if delete_exported_pdf(test_session, test_export_name_2):
        print(f"'{test_export_name_2}.pdf' excluído com sucesso.")
    else:
        print(f"Falha ao excluir '{test_export_name_2}.pdf' ou não encontrado.")

    print("\n--- Teste de Listagem Após Exclusão ---")
    all_listed_after_delete = list_exported_pdfs()
    print(f"Todos os Exports Após Exclusão: {all_listed_after_delete}")

    # Limpeza final do diretório de teste, se estiver vazio
    test_session_export_dir = os.path.join(EXPORTS_DIR, test_session)
    if os.path.exists(test_session_export_dir) and not os.listdir(test_session_export_dir):
        try:
            os.rmdir(test_session_export_dir)
            print(f"Diretório de teste '{test_session_export_dir}' removido.")
        except Exception as e:
            print(f"Erro ao remover diretório de teste: {e}")
```

pdf_processor.py

```
import sys
import os
import PyPDF2

# Adiciona o diretório raiz do projeto ao sys.path para permitir importações absolutas
# quando o módulo é executado diretamente.
project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir))
sys.path.insert(0, project_root)

# Importa as configurações do config.py (apenas para exemplo de uso de caminhos, não diretamente necessário para extração)
from config.config import PDFS_DIR

def extract_text_from_pdf(pdf_path: str) -> str:
    """
    Extrai todo o texto de um arquivo PDF.

    Args:
        pdf_path (str): O caminho completo para o arquivo PDF.

    Returns:
        str: O conteúdo de texto extraído do PDF, ou uma string vazia se houver um erro.
    """
    if not os.path.exists(pdf_path):
        print(f"Erro: O arquivo PDF não foi encontrado em '{pdf_path}'")
        return ""

    text = ""
    try:
        with open(pdf_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            for page_num in range(len(reader.pages)):
                page = reader.pages[page_num]
                text += page.extract_text() or "" # Adiciona o texto da página, garantindo que não seja None
        return text
    except Exception as e:
        print(f"Erro ao extrair texto do PDF '{pdf_path}': {e}")
        return ""

if __name__ == "__main__":
    print("Testando pdf_processor.py...")

    # Para testar, você precisará ter um arquivo PDF na pasta 'pdfs' do seu projeto.
    # Por exemplo, crie um arquivo chamado 'exemplo.pdf' ou use um dos PDFs do tutorial.
    test_pdf_name = "exemplo.pdf" # Altere para o nome de um PDF existente na sua pasta 'pdfs'
    full_pdf_path = os.path.join(PDFS_DIR, test_pdf_name)

    # Crie um arquivo PDF de exemplo se não existir para facilitar o teste
    # (Este é um snippet mínimo para criar um PDF vazio, PyPDF2 não cria conteúdo facilmente)
```

```
    # Se você não tem PDFs, pode pular esta parte e apenas observar o erro de arquivo não encontrado.
    if not os.path.exists(full_pdf_path):
        print(f"\nAVISO: O arquivo de teste '{test_pdf_name}' não foi encontrado em '{PDFS_DIR}'.")
        print("Por favor, coloque um PDF de teste na pasta 'pdfs/' ou crie um.")
        print("Você pode criar um PDF simples com texto usando um editor de texto e 'Salvar como PDF'.")
        # Exemplo de criação de um PDF muito simples via pypdf (requer pypdf instalado)
        # from pypdf import PdfWriter
        # writer = PdfWriter()
        # writer.add_blank_page(width=72, height=72)
        # with open(full_pdf_path, "wb") as fp:
        #     writer.write(fp)
        # print(f"Um PDF vazio '{test_pdf_name}' foi criado para teste.")

    extracted_content = extract_text_from_pdf(full_pdf_path)

    if extracted_content:
        print(f"\nConteúdo extraído de '{test_pdf_name}' (primeiros 500 caracteres):\n")
        print(extracted_content[:500])
        if len(extracted_content) > 500:
            print("\n...")
        print(f"\nTotal de caracteres extraídos: {len(extracted_content)}")
    else:
        print(f"\nNão foi possível extrair conteúdo de '{test_pdf_name}'.")
```

session_manager.py

```
import json
import os
import sys
import datetime
import uuid # Para gerar IDs únicos para os resumos

# Adiciona o diretório raiz do projeto ao sys.path para permitir importações absolutas
# quando o módulo é executado diretamente.
project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir))
sys.path.insert(0, project_root)

# Importa as configurações de caminhos do config.py
from config.config import SESSIONS_DIR, SUMMARIES_DIR, DEFAULT_SESSION_NAME,
SYSTEM_MESSAGE

# --- Funções Auxiliares ---
def _ensure_dir_exists(directory_path: str):
    """Garante que um diretório exista. Se não existir, ele é criado."""
    os.makedirs(directory_path, exist_ok=True)

def _get_session_path(session_name: str) -> str:
    """Retorna o caminho completo para o arquivo de uma sessão."""
    _ensure_dir_exists(SESSIONS_DIR)
    return os.path.join(SESSIONS_DIR, f"{session_name}.json")

def _get_summary_path(summary_id: str) -> str:
    """Retorna o caminho completo para o arquivo de um resumo."""
    _ensure_dir_exists(SUMMARIES_DIR)
    return os.path.join(SUMMARIES_DIR, f"{summary_id}.json")


# --- Gerenciamento de Sessões ---
def load_session(session_name: str) -> dict:
    session_path = _get_session_path(session_name)
    if os.path.exists(session_path):
        try:
            with open(session_path, 'r', encoding='utf-8') as f:
                session_data = json.load(f) # <--- Linha existente

            # --- Adicione as validações de estrutura AQUI ---
            if not isinstance(session_data, dict):
                print(f"Aviso: Sessão '{session_name}' carregada, mas o conteúdo não é um dicionário.
Revertendo para padrão.")
                return {
                    "chat_history": [{"role": "system", "content": SYSTEM_MESSAGE}],
                    "active_api_summary_content": None,
                    "active_api_summary_metadata": None
                }

            # Garante que 'chat_history' é uma lista
```

```
            if "chat_history" not in session_data or not isinstance(session_data["chat_history"], list):
                print(f"Aviso: Sessão '{session_name}' tem 'chat_history' inválido. Revertendo para
histórico padrão.")
                session_data["chat_history"] = [{"role": "system", "content": SYSTEM_MESSAGE}]

            # Garante que 'active_api_summary_content' é uma string ou None
            if "active_api_summary_content" not in session_data or not
isinstance(session_data["active_api_summary_content"], (str, type(None))):
                session_data["active_api_summary_content"] = None

            # Garante que 'active_api_summary_metadata' é um dicionário ou None
            if "active_api_summary_metadata" not in session_data or not
isinstance(session_data["active_api_summary_metadata"], (dict, type(None))):
                session_data["active_api_summary_metadata"] = None
            # --- Fim das validações de estrutura ---

            print(f"Sessão '{session_name}' carregada com sucesso.")
            return session_data
        except json.JSONDecodeError as e:
            print(f"Erro ao decodificar a sessão '{session_name}': {e}. Criando uma nova sessão.")
            # Se o arquivo estiver corrompido, retorna uma sessão padrão
            return {
                "chat_history": [{"role": "system", "content": SYSTEM_MESSAGE}],
                "active_api_summary_content": None,
                "active_api_summary_metadata": None
            }
    else: # Este é o bloco para quando a sessão não existe no disco
        print(f"Sessão '{session_name}' não encontrada. Iniciando uma nova sessão.")
        return {
            "chat_history": [{"role": "system", "content": SYSTEM_MESSAGE}],
            "active_api_summary_content": None,
            "active_api_summary_metadata": None
        }

# ... (suas funções load_session, save_session, list_sessions, delete_session) ...

def session_exists(session_name: str) -> bool:
    """
    Verifica se uma sessão com o nome fornecido existe no disco.

    Args:
        session_name (str): O nome da sessão a ser verificada.

    Returns:
        bool: True se a sessão existir, False caso contrário.
    """
    return os.path.exists(_get_session_path(session_name))

# ... (suas funções load_session, save_session, list_sessions, delete_session) ...

def session_exists(session_name: str) -> bool:
    """
```

```
    Verifica se uma sessão com o nome fornecido existe no disco.

    Args:
        session_name (str): O nome da sessão a ser verificada.

    Returns:
        bool: True se a sessão existir, False caso contrário.
    """
    return os.path.exists(_get_session_path(session_name))

def save_session(session_name: str, chat_history: list, active_api_summary_content: str = None,
active_api_summary_metadata: dict = None):
    """
    Salva o estado atual da sessão de chat.

    Args:
        session_name (str): O nome da sessão a ser salva.
        chat_history (list): A lista de mensagens do histórico de chat.
        active_api_summary_content (str): O conteúdo do resumo ativo, se houver.
        active_api_summary_metadata (dict): Metadados do resumo ativo, se houver.
    """
    session_path = _get_session_path(session_name)
    session_data = {
        "chat_history": chat_history,
        "active_api_summary_content": active_api_summary_content,
        "active_api_summary_metadata": active_api_summary_metadata
    }
    try:
        with open(session_path, 'w', encoding='utf-8') as f:
            json.dump(session_data, f, indent=4, ensure_ascii=False)
        # print(f"Sessão '{session_name}' salva com sucesso.")
    except Exception as e:
        print(f"Erro ao salvar a sessão '{session_name}': {e}")

def list_sessions() -> list:
    """
    Lista todas as sessões salvas.

    Returns:
        list: Uma lista de nomes de sessões.
    """
    _ensure_dir_exists(SESSIONS_DIR)
    sessions = [
        f.replace('.json', '')
        for f in os.listdir(SESSIONS_DIR)
        if f.endswith('.json')
    ]
    return sorted(sessions)

def delete_session(session_name: str) -> bool:
    """
    Exclui uma sessão salva.
```

```
    Args:
        session_name (str): O nome da sessão a ser excluída.

    Returns:
        bool: True se a sessão foi excluída com sucesso, False caso contrário.
    """
    session_path = _get_session_path(session_name)
    if os.path.exists(session_path):
        try:
            os.remove(session_path)
            return True
        except Exception as e:
            return False
    else:
        return False

# --- Gerenciamento de Resumos de PDF ---
def save_pdf_summary(summary_content: str, metadata: dict) -> str:
    """
    Salva um resumo de PDF e seus metadados.

    Args:
        summary_content (str): O conteúdo de texto do resumo.
        metadata (dict): Dicionário com metadados do resumo (ex: nome do arquivo original, data).

    Returns:
        str: O ID único do resumo salvo.
    """
    _ensure_dir_exists(SUMMARIES_DIR)
    summary_id = str(uuid.uuid4()) # Gera um ID único
    summary_data = {
        "id": summary_id,
        "timestamp": datetime.datetime.now().isoformat(),
        "content": summary_content,
        "metadata": metadata
    }
    summary_path = _get_summary_path(summary_id)
    try:
        with open(summary_path, 'w', encoding='utf-8') as f:
            json.dump(summary_data, f, indent=4, ensure_ascii=False)
        print(f"Resumo do PDF '{metadata.get('original_filename', 'N/A')}' salvo com ID:
{summary_id}")
        return summary_id
    except Exception as e:
        print(f"Erro ao salvar o resumo do PDF: {e}")
        return ""

def load_specific_pdf_summary(summary_id: str) -> dict or None:
    """
    Carrega um resumo de PDF específico pelo seu ID.
```

```
    Args:
        summary_id (str): O ID único do resumo a ser carregado.

    Returns:
        dict or None: Um dicionário contendo 'content' e 'metadata' do resumo,
                      ou None se o resumo não for encontrado.
    """
    summary_path = _get_summary_path(summary_id)
    if os.path.exists(summary_path):
        try:
            with open(summary_path, 'r', encoding='utf-8') as f:
                summary_data = json.load(f)
            return summary_data
        except json.JSONDecodeError as e:
            print(f"Erro ao decodificar o resumo '{summary_id}': {e}")
            return None
    else:
        print(f"Resumo com ID '{summary_id}' não encontrado.")
        return None

# Em utils/session_manager.py

# ... (código existente, incluindo _get_summary_path) ...

def delete_pdf_summary(summary_id: str) -> bool:
    """
    Exclui um arquivo de resumo de PDF existente.

    Args:
        summary_id (str): O ID do resumo a ser excluído.

    Returns:
        bool: True se o resumo foi excluído com sucesso, False caso contrário.
    """
    summary_path = _get_summary_path(summary_id)
    if os.path.exists(summary_path):
        try:
            os.remove(summary_path)
            return True
        except OSError as e:
            print(f"Erro ao excluir o resumo '{summary_id}': {e}")
            return False
    else:
        return False # Resumo não encontrado

def list_summaries() -> list:
    """
    Lista todos os resumos de PDF salvos.

    Returns:
        list: Uma lista de dicionários, cada um contendo 'id', 'filename' e 'timestamp' do resumo.
    """
```

```
    _ensure_dir_exists(SUMMARIES_DIR)
    summaries_info = []
    for f_name in os.listdir(SUMMARIES_DIR):
        if f_name.endswith('.json'):
            summary_id = f_name.replace('.json', '')
            summary_path = _get_summary_path(summary_id)
            try:
                with open(summary_path, 'r', encoding='utf-8') as f:
                    summary_data = json.load(f)
                metadata = summary_data.get('metadata', {})
                summaries_info.append({
                    "id": summary_data.get('id', 'N/A'),
                    "filename": metadata.get('original_filename', 'N/A'),
                    "timestamp": summary_data.get('timestamp', 'N/A')
                })
            except json.JSONDecodeError as e:
                print(f"Aviso: Arquivo de resumo corrompido ou inválido: {f_name} - {e}")
            except Exception as e:
                print(f"Aviso: Erro ao ler resumo: {f_name} - {e}")
    # Ordena os resumos pelo timestamp para que os mais recentes apareçam primeiro
    return sorted(summaries_info, key=lambda x: x.get('timestamp', ''), reverse=True)


if __name__ == "__main__":
    print("Testando session_manager.py...")

    # --- Teste de Sessões ---
    test_session_name = "test_session_123"
    print(f"\n--- Gerenciamento de Sessões ({test_session_name}) ---")

    # 1. Carregar sessão (deve ser nova)
    loaded_session = load_session(test_session_name)
    print(f"Histórico carregado (nova): {loaded_session['chat_history']}")

    # 2. Salvar uma sessão
    test_history = loaded_session['chat_history']
    test_history.append({"role": "user", "content": "Olá, esta é uma mensagem de teste."})
    test_history.append({"role": "assistant", "content": "Entendido. Salvando sessão."})
    save_session(test_session_name, test_history, None, None)

    # 3. Carregar novamente a sessão (agora deve ter histórico)
    loaded_session_again = load_session(test_session_name)
    print(f"Histórico carregado (existente): {loaded_session_again['chat_history']}")

    # 4. Listar sessões
    print("\nSessões Atuais:", list_sessions())

    # 5. Criar outra sessão para listar
    save_session("outra_sessao", [{"role": "user", "content": "Segunda sessão."}], None, None)
    print("Sessões Após criar 'outra_sessao':", list_sessions())

    # --- Teste de Resumos de PDF ---
```

```
    print("\n--- Gerenciamento de Resumos de PDF ---")
    test_summary_content = "Este é um resumo de teste de um documento sobre IA."
    test_summary_metadata = {
        "original_filename": "documento_ia.pdf",
        "pages": 5,
        "topic": "Inteligência Artificial"
    }

    # 1. Salvar um resumo
    saved_summary_id = save_pdf_summary(test_summary_content, test_summary_metadata)
    if saved_summary_id:
        print(f"Resumo de PDF salvo com ID: {saved_summary_id}")

        # 2. Carregar o resumo salvo
        loaded_summary = load_specific_pdf_summary(saved_summary_id)
        if loaded_summary:
            print(f"Conteúdo do resumo carregado: '{loaded_summary['content'][:50]}...'")
            print(f"Metadados do resumo carregado: {loaded_summary['metadata']}")

    # 3. Listar resumos
    print("\nResumos Atuais:")
    for summary in list_summaries():
        print(f"- ID: {summary['id']}, Arquivo: {summary['filename']}, Data:
{summary['timestamp']}")

    # --- Limpeza ---
    print("\n--- Limpeza ---")
    # Deletar sessões de teste
    delete_session(test_session_name)
    delete_session("outra_sessao")
    print("Sessões Após limpeza:", list_sessions())

    # (Não há função para deletar resumo individualmente no teste, mas a sessão o gerencia)
    # Em um cenário real, você poderia adicionar uma função de exclusão de resumo se necessário.
```

token_utils.py

```
import sys
import os
current_dir = os.path.dirname(os.path.abspath(__file__))
project_root = os.path.abspath(os.path.join(current_dir, '..'))
if project_root not in sys.path:
    sys.path.append(project_root)
import tiktoken
from config.config import DEFAULT_MODEL

# Carrega o codificador de tokens para o modelo padrão.
# O encoding "cl100k_base" é comumente usado por modelos como gpt-3.5-turbo e gpt-4.
ENCODER = tiktoken.get_encoding("cl100k_base")

def count_tokens_in_string(text: str) -> int:
    """
    Conta o número de tokens em uma string de texto usando o codificador padrão.

    Args:
        text (str): A string de texto a ser tokenizada.

    Returns:
        int: O número de tokens na string.
    """
    return len(ENCODER.encode(text))

def count_tokens_in_messages(messages: list) -> int:
    """
    Conta o número de tokens em uma lista de mensagens formatadas para a API do OpenAI.
    Esta função é baseada na documentação da OpenAI para contagem de tokens de mensagens
    e leva em consideração a estrutura de role/content.

    Args:
        messages (list): Uma lista de dicionários de mensagens, como:
                         [{"role": "system", "content": "Seu nome é Bot."},
                          {"role": "user", "content": "Olá!"}]

    Returns:
        int: O número total de tokens nas mensagens.
    """
    # Adaptação para gpt-3.5-turbo e gpt-4 conforme documentação da OpenAI
    # Cada mensagem tem um custo base de 4 tokens (role + content).
    # Algumas versões de modelo podem ter um custo extra de 1 token para respostas.
    # Estamos usando uma abordagem mais segura que se alinha com exemplos da OpenAI.
    tokens_per_message = 3 # Cada mensagem geralmente custa 3 tokens (role, content, e o final do
turno)
    tokens_per_name = 1    # Se um nome é fornecido, ele custa 1 token extra.

    total_tokens = 0
    for message in messages:
        total_tokens += tokens_per_message
```

```
        for key, value in message.items():
            total_tokens += count_tokens_in_string(value)
            if key == "name":
                total_tokens += tokens_per_name
    total_tokens += 3 # Cada resposta geralmente começa com 'assistant', o que custa 3 tokens.
                      # Esta é uma estimativa, pode variar ligeiramente.
    return total_tokens

if __name__ == "__main__":
    print("Testando token_utils.py...")

    # Teste de contagem de tokens em string
    text_example = "Olá, como você está hoje? Espero que esteja tudo bem!"
    tokens_string = count_tokens_in_string(text_example)
    print(f"\nTexto: '{text_example}'")
    print(f"Tokens na string: {tokens_string}")

    # Teste de contagem de tokens em mensagens
    messages_example = [
        {"role": "system", "content": "Você é um assistente de IA útil."},
        {"role": "user", "content": "Qual a capital do Brasil?"},
        {"role": "assistant", "content": "A capital do Brasil é Brasília."}
    ]
    tokens_messages = count_tokens_in_messages(messages_example)
    print(f"\nMensagens de exemplo: {messages_example}")
    print(f"Tokens nas mensagens (estimado): {tokens_messages}")

    # Exemplo com uma string longa
    long_text = "Isso é um texto muito longo para testar a contagem de tokens. " * 50
    print(f"\nTexto longo (primeiros 50 chars): '{long_text[:50]}...'")
    print(f"Tokens no texto longo: {count_tokens_in_string(long_text)}")
```