Arquivo: gerar_documentos.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import pandas as pd
import re
from num2words import num2words
import os
from datetime import datetime
from PyQt6.QtWidgets import *
from PyQt6.QtCore import QSize
from fpdf import FPDF
from docxtpl import DocxTemplate
from PyPDF2 import PdfMerger
from paths import load_config_path_id, save_config, TEMPLATE_DISPENSA_DIR
from      modules.dispensa.dialogs.edit_data.widgets.gerenciador_anexos.pdf_add_dialog      import
ProgressDialog
#from merge_pdf.merge_anexos import merge_pdfs_anexos
import sys
import subprocess
from pathlib import Path

class ConsolidarDocumentos(QObject):
    status_atualizado = pyqtSignal(str, str)
    pastas_criadas = pyqtSignal(bool)

    def __init__(self, dados, icons, documentos_encontrados, status_label=None):
        super().__init__()
        self.dados = dados
        self.icons = icons
        self.config = load_config_path_id()
        self.relacao_documentos_encontrados = documentos_encontrados

        # Define o diretório raiz (ex: Desktop) a partir da configuração
        self.diretorio_raiz = Path(self.config.get('pasta_base', str(Path.home() / 'Desktop')))

        # O nome da pasta do processo será definido dinamicamente
        self.nome_pasta = ""
        self.pasta_processo = "" # Será o caminho completo (diretorio_raiz + nome_pasta)

        # Verifica se 'dados' é um DataFrame ou dict
        if isinstance(self.dados, pd.DataFrame) and not self.dados.empty:
            self.id_processo = self.dados['id_processo'].iloc[0]
            self.objeto = self.dados['objeto'].iloc[0]
            self.numero = self.dados.get('numero', self.dados.get('id_processo', '')) # Adicionado
            self.ano = self.dados.get('ano', self.dados.get('id_processo', '')) # Adicionado
        elif isinstance(self.dados, dict):
            self.id_processo = self.dados.get('id_processo', 'Desconhecido')
            self.objeto = self.dados.get('objeto', 'Desconhecido')
            self.numero = self.dados.get('numero', 'Desconhecido') # Adicionado
            self.ano = self.dados.get('ano', 'Desconhecido') # Adicionado
```

```
        # 2. Define os caminhos como objetos Path desde o início
        self.diretorio_raiz = Path(self.config.get('pasta_base', str(Path.home() / 'Desktop')))

        # Formata o nome da pasta do processo
        id_processo_formatado = str(self.id_processo).replace("/", "-")
        objeto_formatado = str(self.objeto).replace("/", "-")
        self.nome_pasta = f"{id_processo_formatado} - {objeto_formatado}"

        # 3. Garante que self.pasta_processo seja um objeto Path IMEDIATAMENTE
        self.pasta_processo = self.diretorio_raiz / self.nome_pasta

        # Exemplo de dados de índice (mantido)
        self.data = {
            'id_processo': 'DE 15/2024', 'tipo': 'DE', 'numero': '50', 'ano': '2024',
            'situacao': 'Planejamento, Sessão Pública, Concluído', 'nup': '62055.00055/2024-01',
            'material_servico': 'Material ou Serviço', 'objeto': 'Suprimentos de informática',
            'vigencia': '12 meses a partir da assinatura', 'data_sessao': '15/10/2024',
            'operador': 'João da Silva', 'criterio_julgamento': 'Menor preço, Técnica e preço',
            'com_disputa': 'Sim, Não',
        }

        # Chama a função para definir os nomes e caminhos pela primeira vez
        self.atualizar_nome_pasta()



    def atualizar_nome_pasta(self):
        def sanitize_filename(name):
            """Remove caracteres ilegais para nomes de pasta/arquivo."""
            # Remove caracteres ilegais do Windows E o apóstrofo
            return re.sub(r'[\\/*?:"<>|\']', "", name)

        if isinstance(self.dados, dict):
            objeto_bruto = self.dados.get('objeto') or 'objeto_desconhecido'
            numero = str(self.dados.get('numero', 'N_A'))
            ano = str(self.dados.get('ano', 'N_A'))
        else: # Assume DataFrame
            objeto_bruto = self.dados['objeto'].iloc[0]
            numero = str(self.dados['numero'].iloc[0])
            ano = str(self.dados['ano'].iloc[0])

                        objeto_formatado   =   str(objeto_bruto   if   pd.notna(objeto_bruto)   else
'objeto_desconhecido').replace("/", "-")
        objeto_sanitizado = sanitize_filename(objeto_formatado) # Aplica a limpeza

        prefixo_pasta = f"DE-787010-{numero}-{ano}-"
        limite_objeto = 150 - len(prefixo_pasta)

        if len(objeto_sanitizado) > limite_objeto:
            objeto_final = objeto_sanitizado[:limite_objeto].strip()
        else:
            objeto_final = objeto_sanitizado.strip()
```

```
        self.nome_pasta = f"{prefixo_pasta}{objeto_final}"
        self.pasta_processo = self.diretorio_raiz / self.nome_pasta

    def validar_e_definir_pasta_base(self):
        """Verifica se o caminho base da configuração é válido. Se não, pede um novo."""
        caminho_salvo = self.config.get('pasta_base')

        # Se não há caminho salvo ou o caminho não existe, pede para o usuário selecionar um novo
        if not caminho_salvo or not Path(caminho_salvo).exists():
            QMessageBox.warning(None, "Diretório Base Inválido",
                                f"O diretório base ('{caminho_salvo}') não foi encontrado.\n\n"
                                      "Por favor, selecione um novo diretório base para salvar os
documentos.")

            # Chama a função que abre o diálogo para escolher pasta
            self.alterar_diretorio_base()
        else:
            # Se o caminho é válido, define-o
            self.pasta_base = Path(caminho_salvo)

    def atualizar_nome_pasta(self):
        id_processo = self.dados.get('id_processo', 'desconhecido').replace("/", "-")
        objeto = self.dados.get('objeto', 'objeto_desconhecido').replace("/", "-")
        self.nome_pasta = f"{id_processo} - {objeto}"

    def update_data(self, new_data):
        """Atualiza os dados e renova o nome da pasta."""
        self.dados.update(new_data)
        self.atualizar_nome_pasta()

    def criar_e_abrir_pasta(self):
        pastas_existentes = self.verificar_pastas(self.pasta_base, criar=True)
        status_text = "Pastas encontradas" if pastas_existentes else "Pastas não encontradas"
        icon_key = "folder_v" if pastas_existentes else "folder_x"
        icon = self.icons.get(icon_key)

        # Emitindo o sinal para atualizar o ícone e a mensagem
        self.status_atualizado.emit(status_text, icon)

        # Abre a pasta criada
        self.abrir_pasta(self.pasta_base / self.nome_pasta)

    def verificar_existencia_pastas(self):
        # Verifica a existência das pastas sem criá-las
        return self.verificar_pastas(self.pasta_base, criar=False)

    def verificar_pastas(self, criar=False):
        # USA self.pasta_processo para criar a estrutura DENTRO da pasta do processo
        pastas_necessarias = [
            self.pasta_processo / '1. Autorizacao',
            self.pasta_processo / '2. CP e anexos',
            self.pasta_processo / '3. Aviso',
            self.pasta_processo / '2. CP e anexos' / 'DFD',
```

```
            self.pasta_processo / '2. CP e anexos' / 'DFD' / 'Anexo A - Relatorio Safin',
                 self.pasta_processo / '2. CP e anexos' / 'DFD' / 'Anexo B - Especificações e
Quantidade',
              self.pasta_processo / '2. CP e anexos' / 'DFD' / 'Anexo C - PDF DFD',  # <-- NOVA
PASTA ADICIONADA AQUI
            self.pasta_processo / '2. CP e anexos' / 'TR',
            self.pasta_processo / '2. CP e anexos' / 'TR' / 'Pesquisa de Preços',
            self.pasta_processo / '2. CP e anexos' / 'Declaracao de Adequação Orçamentária',
               self.pasta_processo / '2. CP e anexos' / 'Declaracao de Adequação Orçamentária' /
'Relatório do PDM-Catser',
            self.pasta_processo / '2. CP e anexos' / 'ETP',
            self.pasta_processo / '2. CP e anexos' / 'MR',
            self.pasta_processo / '2. CP e anexos' / 'Justificativas Relevantes',
        ]

        # Criação das pastas, se necessário
        for pasta in pastas_necessarias:
            if not pasta.exists() and criar:
                pasta.mkdir(parents=True)

        # Re-verifica a existência de todas as pastas
        pastas_existem = all(pasta.exists() for pasta in pastas_necessarias)

        # Emite o sinal indicando o status atualizado das pastas
        self.pastas_criadas.emit(pastas_existem)
        return pastas_existem

    def verificar_pdfs_existentes(self):
        base_path = self.pasta_processo
        print(f"Base path para verificação de PDFs: {base_path}")

        resultados = []

        # Define as verificações e os textos correspondentes
        pastas_para_verificar = [
            (
                [base_path / '2. CP e anexos' / 'DFD' / 'Anexo A - Relatorio Safin',
                base_path / '2. CP e anexos' / 'DFD' / 'Anexo B - Especificações e Quantidade'],
                "Documento de Formalização da Demanda (DFD) e seus anexos"
            ),
            (
                [base_path / '2. CP e anexos' / 'TR' / 'Pesquisa de Preços'],
                "Termo de Referência (TR) e seu anexo"
            ),
            (
                      [base_path / '2. CP e anexos' / 'Declaracao de Adequação Orçamentária' /
'Relatório do PDM-Catser'],
                "Declaração de Adequação Orçamentária e seu anexo"
            ),
            (
                [],  # Sem necessidade de verificação
                "Justificativas Relevantes"
            ),
```

```
            (
                [base_path / '2. CP e anexos' / 'ETP'],
                "Estudo Técnico Preliminar (ETP)"
            ),
            (
                [base_path / '2. CP e anexos' / 'MR'],
                "Matriz de Riscos"
            )
        ]

        # Verifica a existência de PDFs nas pastas especificadas e atribui letras dinamicamente
        letra_ordem = ord("A")
        for pastas, texto in pastas_para_verificar:
            if not pastas:  # Adiciona texto sem verificação de pastas
                resultados.append(f"{chr(letra_ordem)}) {texto}")
                letra_ordem += 1
            elif all(any(pasta.glob("*.pdf")) for pasta in pastas):  # Verifica se todas as pastas
têm PDFs
                resultados.append(f"{chr(letra_ordem)}) {texto}")
                letra_ordem += 1

        # Adiciona a pontuação final para cada item
        for i in range(len(resultados)):
            if i == len(resultados) - 1:
                resultados[i] += "."  # Último item termina com ponto final
            elif i == len(resultados) - 2:
                resultados[i] += "; e"  # Penúltimo item termina com "; e"
            else:
                resultados[i] += ";"  # Outros itens terminam com ponto e vírgula

        return resultados



    def editar_modelo(self, button_font_size=18, icon_size=QSize(40, 40)):
        dialog = QDialog()
        dialog.setWindowTitle("Editar Template")

        # Adicionar ícone ao título
        icon_confirm = QIcon(self.icons["confirm_green"])
        dialog.setWindowIcon(icon_confirm)

        # Ícones para os botões
        icon_index = QIcon(self.icons["pdf"])
        icon_open = QIcon(self.icons["open_icon"])  # Ícone para os demais botões "Abrir"

        # Layout principal do diálogo
        main_layout = QVBoxLayout(dialog)

        # Layout horizontal para o botão "Abrir índice"
        top_layout = QHBoxLayout()
        button_open_index = QPushButton("Índice")
        button_open_index.setIcon(icon_index)
```

```
        button_open_index.setFixedSize(110, 40)  # Definir tamanho fixo para uniformidade
        button_open_index.setIconSize(icon_size)  # Ajusta o tamanho do ícone
        button_open_index.setStyleSheet("font-size: 18px;")  # Ajusta o tamanho da fonte
        button_open_index.clicked.connect(self.abrir_indice)

        # Adicionar o texto ao lado do botão "Abrir índice"
        label_info = QLabel("Relação de Variáveis e exemplos de uso")
        label_info.setStyleSheet("font-size: 18px;")  # Definir tamanho da fonte para 14

        top_layout.addWidget(button_open_index, alignment=Qt.AlignmentFlag.AlignLeft)
        top_layout.addWidget(label_info, alignment=Qt.AlignmentFlag.AlignLeft)

        top_layout.addStretch()
        # Layout para os templates
        templates_layout = QVBoxLayout()

        # Lista de templates e seus caminhos
        templates = [
                                            ("Template    Autorização",    TEMPLATE_DISPENSA_DIR    /
"template_autorizacao_dispensa.docx"),
            ("Template Comunicação Padronizada", TEMPLATE_DISPENSA_DIR / "template_cp.docx"),
            #("Template DFD", TEMPLATE_DISPENSA_DIR / "template_dfd.docx"),
            ("Template Termo de Referência", TEMPLATE_DISPENSA_DIR / "template_tr.docx"),
                        ("Template  Termo  de  Referência  (Serviço)",  TEMPLATE_DISPENSA_DIR  /
"template_tr_servico.docx"),
                    ("Template  Declaração  de  Adequação  Orçamentária",  TEMPLATE_DISPENSA_DIR  /
"template_dec_adeq.docx"),
            ("Template Aviso de Dispensa", TEMPLATE_DISPENSA_DIR / "template_aviso_dispensa.docx")
        ]

        # Adicionar os templates ao layout
        for template_name, template_path in templates:
            template_row = QHBoxLayout()

            label = QLabel(template_name)
            label.setStyleSheet("font-size: 18px;")  # Definir tamanho da fonte para 14
            button_open_template = QPushButton("Abrir")
            button_open_template.setIcon(icon_open)
            button_open_template.setFixedSize(110, 40)  # Definir tamanho fixo para uniformidade
            button_open_template.setIconSize(icon_size)  # Ajusta o tamanho do ícone
            button_open_template.setStyleSheet("font-size: 18px;")  # Ajusta o tamanho da fonte
                            button_open_template.clicked.connect(lambda  _,  path=template_path:
self.abrir_template(path))

            template_row.addWidget(button_open_template)
            template_row.addWidget(label)
            templates_layout.addLayout(template_row)

        # Adicionar layouts ao layout principal
        main_layout.addLayout(top_layout)
        main_layout.addLayout(templates_layout)

        dialog.setLayout(main_layout)
```

```
        dialog.exec()

    def abrir_template(self, path):
        # Verificar se o arquivo existe
        if path.exists() and path.is_file():
            print(f"Arquivo encontrado: {path}")
            try:
                # Abre o arquivo utilizando o caminho absoluto
                full_path = str(path.resolve())  # Resolve para o caminho absoluto
                subprocess.run(f'start "" "{full_path}"', shell=True)  # Windows
                # Para Linux ou macOS, use os comandos adequados
                # subprocess.run(['xdg-open', full_path])  # Linux
                # subprocess.run(['open', full_path])  # macOS
            except Exception as e:
                print(f"Erro ao abrir o template: {e}")
        else:
            print(f"Arquivo não encontrado: {path}")

    def abrir_indice(self):
        # Cria o PDF dos índices
        pdf = FPDF()
        pdf.add_page()
        pdf.set_font("Arial", size=12)

        # Adicionar título
        pdf.cell(200, 10, txt="Índices Utilizados nos Templates", ln=True, align='C')

        # Adicionar exemplos de cada índice com cores personalizadas
        for key, example in self.data.items():
            # Definir cor para o 'key' (azul marinho)
            pdf.set_text_color(0, 0, 128)  # Azul marinho (RGB: 0, 0, 128)
            pdf.write(10, f"{{{{{key}}}}}: ")

            # Definir cor para o 'example' (vermelho escuro)
            pdf.set_text_color(139, 0, 0)  # Vermelho escuro (RGB: 139, 0, 0)
            pdf.write(10, f"Exemplo: {example}\n")  # Adiciona texto com nova linha

        # Salva o PDF
        pdf_path = self.pasta_processo / "indice_templates.pdf"
        pdf.output(str(pdf_path))

        print(f"Arquivo PDF de índices criado: {pdf_path}")

        # Abrir o PDF criado
        if pdf_path.exists() and pdf_path.is_file():
            try:
                subprocess.run(f'start "" "{str(pdf_path)}"', shell=True)  # Windows
                # Para Linux ou macOS, use os comandos adequados
                # subprocess.run(['xdg-open', str(pdf_path)])  # Linux
                # subprocess.run(['open', str(pdf_path)])  # macOS
            except Exception as e:
                print(f"Erro ao abrir o PDF de índices: {e}")
        else:
```

```
            print(f"Arquivo PDF de índices não encontrado: {pdf_path}")

    def alterar_diretorio_base(self):
            new_dir = QFileDialog.getExistingDirectory(None, "Selecione o Novo Diretório Base",
str(Path.home()))
        if new_dir:
            self.pasta_base = Path(new_dir)  # Garante que o atributo seja atualizado
            self.config['pasta_base'] = str(self.pasta_base)
            save_config("pasta_base", str(self.pasta_base))
              QMessageBox.information(None, "Diretório Base Alterado", f"O novo diretório base foi
definido como: {self.pasta_base}")
        else:
            # Caso o usuário cancele, usa a pasta Desktop como padrão para evitar erros
            self.pasta_base = Path.home() / 'Desktop'
                QMessageBox.warning(None, "Nenhum Diretório Selecionado", f"Nenhum diretório foi
selecionado. Usando a Área de Trabalho como padrão: {self.pasta_base}")

    def abrir_pasta_base(self):
        try:
            os.startfile(self.pasta_base)
        except Exception as e:
            print(f"Erro ao abrir a pasta base: {e}")
            QMessageBox.warning(None, "Erro", f"Erro ao abrir a pasta base: {e}")

    def abrirDocumento(self, docx_path):
        try:
            pdf_path = self.convert_to_pdf(docx_path)
            QDesktopServices.openUrl(QUrl.fromLocalFile(str(pdf_path)))
            print(f"Documento PDF aberto: {pdf_path}")
        except Exception as e:
            print(f"Erro ao abrir ou converter o documento: {e}")
            QMessageBox.warning(None, "Erro", f"Erro ao abrir ou converter o documento: {e}")

    def salvarPDF(self, docx_path):
        try:
            pdf_path = self.convert_to_pdf(docx_path)
            print(f"Documento PDF salvo: {pdf_path}")
            return pdf_path
        except Exception as e:
            print(f"Erro ao converter o documento: {e}")
            QMessageBox.warning(None, "Erro", f"Erro ao converter o documento: {e}")
            return None

    def convert_to_pdf(self, docx_path):
        docx_path = Path(docx_path) if not isinstance(docx_path, Path) else docx_path
        pdf_path = docx_path.with_suffix('.pdf')

        # Lógica para Windows
        if sys.platform.startswith("win"):
            word = None
            doc = None
            try:
                import win32com.client
```

```
            import pythoncom

            # Inicializa a comunicação COM (essencial em apps com interface gráfica)
            pythoncom.CoInitialize()

            word = win32com.client.Dispatch("Word.Application")
            word.Visible = False # Mantém o Word invisível

            # Abre o documento usando o caminho absoluto
            doc = word.Documents.Open(str(docx_path.resolve()))

            # Salva como PDF
            doc.SaveAs(str(pdf_path.resolve()), FileFormat=17)

            # COMANDO CRÍTICO: Fecha o documento SEM salvar alterações no modelo.
            # Isso evita as caixas de diálogo ocultas que travam o Word.
            doc.Close(SaveChanges=0)

        except Exception as e:
            error_message = f"Falha na conversão para PDF: {e}\nDocumento: {docx_path}"
            print(error_message)
            raise Exception(error_message) # Propaga o erro

        finally:
            # Este bloco SEMPRE é executado, garantindo que o Word feche.
            if word is not None:
                word.Quit()

            doc = None
            word = None

            pythoncom.CoUninitialize() # Finaliza a comunicação COM

    # Lógica para outros sistemas (Linux/macOS)
    else:
        try:
            comando = ["libreoffice", "--headless", "--convert-to", "pdf",
                       "--outdir", str(docx_path.parent), str(docx_path)]
            subprocess.run(comando, check=True)
        except Exception as e:
            raise e

    if not pdf_path.exists():
        raise FileNotFoundError(f"O arquivo PDF não foi criado após a conversão: {pdf_path}")

    return pdf_path

# def convert_to_pdf(self, docx_path):
#     docx_path = Path(docx_path) if not isinstance(docx_path, Path) else docx_path
#     pdf_path = docx_path.with_suffix('.pdf')
#     word = win32com.client.Dispatch("Word.Application")
#     doc = None
#     try:
```

```
    #         doc = word.Documents.Open(str(docx_path))
    #         doc.SaveAs(str(pdf_path), FileFormat=17)
    #     except Exception as e:
    #         raise e
    #     finally:
    #         if doc is not None:
    #             doc.Close()
    #         word.Quit()
    #     if not pdf_path.exists():
    #         raise FileNotFoundError(f"O arquivo PDF não foi criado: {pdf_path}")
    #     return pdf_path

    def valor_por_extenso(self, valor):
        if not valor or valor.strip() == '':  # Verifica se o valor está vazio ou None
            return None  # Retorna None se o valor não for válido

        try:
            valor = valor.replace('R$', '').replace('.', '').replace(',', '.').strip()
            valor_float = float(valor)
            parte_inteira = int(valor_float)
            parte_decimal = int(round((valor_float - parte_inteira) * 100))

            if parte_decimal > 0:
                            valor_extenso = f"{num2words(parte_inteira, lang='pt_BR')} reais e
{num2words(parte_decimal, lang='pt_BR')} centavos"
            else:
                valor_extenso = f"{num2words(parte_inteira, lang='pt_BR')} reais"

            # Corrige "um reais" para "um real"
            valor_extenso = valor_extenso.replace("um reais", "um real")

            return valor_extenso

        except ValueError:
            return None

    def alterar_posto(self, posto):
        # Define um dicionário de mapeamento de postos e suas respectivas abreviações
        mapeamento_postos = {
            r'Capitão[\s\-]de[\s\-]Corveta': 'CC',
            r'Capitão[\s\-]de[\s\-]Fragata': 'CF',
            r'Capitão[\s\-]de[\s\-]Mar[\s\-]e[\s\-]Guerra': 'CMG',
            r'Capitão[\s\-]Tenente': 'CT',
            r'Primeiro[\s\-]Tenente': '1ºTen',
            r'Segundo[\s\-]Tenente': '2ºTen',
            r'Primeiro[\s\-]Sargento': '1ºSG',
            r'Segundo[\s\-]Sargento': '2ºSG',
            r'Terceiro[\s\-]Sargento': '3ºSG',
            r'Cabo': 'CB',
            r'Sub[\s\-]oficial': 'SO',
        }

        # Itera sobre o dicionário de mapeamento e aplica a substituição
```

```
        for padrao, substituicao in mapeamento_postos.items():
            if re.search(padrao, posto, re.IGNORECASE):
                return re.sub(padrao, substituicao, posto, flags=re.IGNORECASE)

        # Retorna o posto original se nenhuma substituição for aplicada
        return posto

    def formatar_responsavel(self, chave, data, context):
        responsavel = data.get(chave)
        if responsavel and isinstance(responsavel, str):
            try:
                nome, posto, funcao = responsavel.split('\n')
                posto_alterado = self.alterar_posto(posto)
                responsavel_dict = {
                    'nome': nome,
                    'posto': posto_alterado,
                }
                                responsavel_extenso  =  f"{responsavel_dict.get('posto',  '')}
{responsavel_dict.get('nome', '')}"
                context.update({f'{chave}_formatado': responsavel_extenso})
            except ValueError:
                context.update({f'{chave}_formatado': 'Não especificado\nNão especificado'})
        else:
            context.update({f'{chave}_formatado': 'Não especificado\nNão especificado'})

    def prepare_context(self, data, documentos_encontrados):
        # Verifica e imprime documentos_encontrados para depuração
        print("Documentos encontrados para inclusão no contexto:", documentos_encontrados)
        # Garante que `documentos_encontrados` é uma lista ou usa uma lista vazia como padrão
        if not isinstance(documentos_encontrados, list):
            documentos_encontrados = []
            print(documentos_encontrados)
        # Cria o contexto com os dados, convertendo valores None para 'Não especificado'
         context = {key: (str(value) if value is not None else 'Não especificado') for key, value
in data.items()}
        # Descrição do serviço, dependendo do tipo de material
             descricao_servico  =  "aquisição  de"  if  data['material_servico']  ==  "Material"  else
"contratação de empresa especializada em"
                  descricao_servico_primeira_letra_maiuscula  =  descricao_servico[0].upper()  +
descricao_servico[1:]
        context.update({'descricao_servico': descricao_servico})
                                    context.update({'descricao_servico_primeira_letra_maiuscula':
descricao_servico_primeira_letra_maiuscula})

        # Adiciona a lista de anexos ao contexto para ser usada no template
        # A lista de anexos é passada como uma string única com quebras de linha entre os itens
        context.update({'lista_anexos': "\n".join(documentos_encontrados)})

        # Processar responsável pela demanda e operador
        self.formatar_responsavel('responsavel_pela_demanda', data, context)
        self.formatar_responsavel('operador', data, context)

        # Processa o valor total e o converte para extenso
```

```
        valor_total = data.get('valor_total')
        if valor_total and isinstance(valor_total, str):
            valor_extenso = self.valor_por_extenso(valor_total)
            valor_total_e_extenso = f"{valor_total} ({valor_extenso})"
            context.update({'valor_total_e_extenso': valor_total_e_extenso})
        else:
            context.update({'valor_total_e_extenso': 'Não especificado'})

        # Lógica para atividade_custeio
        if data.get('atividade_custeio') == 'Sim':
            texto_custeio = (
                 "A presente contratação por dispensa de licitação está enquadrada como atividade de custeio, "
                  "conforme mencionado no artigo 2º da Portaria ME nº 7.828, de 30 de agosto de 2022. "
                "Conforme previsão do art. 3º do Decreto nº 10.193, de 27 de dezembro de 2019, e as normas "
                  "infralegais de delegação de competência no âmbito da Marinha, que estabelecem limites e instâncias "
                     "de governança, essa responsabilidade é delegada ao ordenador de despesas, respeitando os valores "
                "estipulados no decreto."
            )
        else:
            texto_custeio = (
                 "A presente contratação por dispensa de licitação não se enquadra nas hipóteses de atividades de "
                 "custeio previstas no Decreto nº 10.193, de 27 de dezembro de 2019, pois o objeto contratado não se "
                  "relaciona diretamente às atividades comuns de suporte administrativo mencionadas no artigo 2º da "
                "Portaria ME nº 7.828, de 30 de agosto de 2022."
            )
        context.update({'texto_custeio': texto_custeio})

        # Alterar formato de data_sessao
        data_sessao = data.get('data_sessao')
        if data_sessao:
            try:
                data_obj = datetime.strptime(data_sessao, '%Y-%m-%d')
                dia_semana = data_obj.strftime('%A')
                data_formatada = data_obj.strftime('%d/%m/%Y') + f" ({dia_semana})"
                context.update({'data_sessao_formatada': data_formatada})
            except ValueError as e:
                context.update({'data_sessao_formatada': 'Data inválida'})
                print("Erro ao processar data da sessão:", e)
        else:
            context.update({'data_sessao_formatada': 'Não especificado'})
            print("Data da sessão não especificada")

        return context

    def gerarDocumento(self, template, subfolder, desc):
```

```
        # Caminhos dos templates e do arquivo a ser salvo
        template_filename = f"template_{template}.docx"
        template_path, save_path = self.setup_document_paths(template_filename, subfolder, desc)

        # Verifica se o template existe
        if not template_path.exists():
                QMessageBox.warning(None, "Erro de Template", f"O arquivo de template não foi encontrado: {template_path}")
            return

        # Carregar e renderizar o template
        with open(str(template_path), 'rb') as template_file:
            doc = DocxTemplate(template_file)
            context = self.prepare_context(self.dados, self.relacao_documentos_encontrados)
            doc.render(context)
            doc.save(str(save_path))

        return save_path

    def setup_document_paths(self, template_filename, subfolder_name, file_description):
        template_path = TEMPLATE_DISPENSA_DIR / template_filename

        # Verifique o tipo de self.dados e extraia id_processo e objeto conforme o tipo
        if isinstance(self.dados, pd.DataFrame):
            id_processo = self.dados['id_processo'].iloc[0].replace('/', '-')
            objeto = self.dados['objeto'].iloc[0]
        elif isinstance(self.dados, dict):
            id_processo = self.dados.get('id_processo', 'desconhecido').replace('/', '-')
            objeto = self.dados.get('objeto', 'objeto_desconhecido')
        else:
            raise ValueError("O tipo de 'dados' não é suportado. Esperado DataFrame ou dict.")

        self.nome_pasta = f"{id_processo} - {objeto}"

        pasta_destino = self.pasta_processo / subfolder_name
        pasta_destino.mkdir(parents=True, exist_ok=True)

        save_path = pasta_destino / f"{id_processo} - {file_description}.docx"
        return template_path, save_path

    def gerar_e_abrir_documento(self, template_type, subfolder_name, file_description):
        docx_path = self.gerarDocumento(template_type, subfolder_name, file_description)
        if docx_path:
            self.abrirDocumento(docx_path)

    def gerar_autorizacao(self):
          self.gerar_e_abrir_documento("autorizacao_dispensa", "1. Autorizacao", "Autorizacao para abertura de Processo Administrativo")

    def gerar_comunicacao_padronizada(self, documentos_encontrados):
        # Verifique se o material_servico é 'Serviço' e escolha o template adequado
             termo_referencia_template = "tr_servico" if self.dados.get("material_servico") == "Serviço" else "tr"
```

```
        documentos = [
            {"template": "cp", "subfolder": "2. CP e anexos", "desc": "Comunicacao Padronizada"},
                  {"subfolder": "2. CP e anexos/DFD/Anexo C - PDF DFD", "desc": "Documento de
Formalizacao de Demanda", "cover": "dfd.pdf"},
                     {"subfolder": "2. CP e anexos/DFD/Anexo A - Relatorio Safin", "cover":
"anexo-a-dfd.pdf"},
               {"subfolder": "2. CP e anexos/DFD/Anexo B - Especificações e Quantidade", "cover":
"anexo-b-dfd.pdf"},
              {"template": "tr", "subfolder": "2. CP e anexos/TR", "desc": "Termo de Referencia",
"cover": "tr.pdf"},
            {"subfolder": "2. CP e anexos/TR/Pesquisa de Preços", "cover": "anexo-tr.pdf"},
                  {"template": "dec_adeq", "subfolder": "2. CP e anexos/Declaracao de Adequação
Orçamentária", "desc": "Declaracao de Adequação Orçamentária", "cover": "dec_adeq.pdf"},
                 {"subfolder": "2. CP e anexos/Declaracao de Adequação Orçamentária/Relatório do
PDM-Catser", "cover": "anexo-dec-adeq.pdf"},
            {"subfolder": "2. CP e anexos/ETP", "cover": "etp.pdf"},
            {"subfolder": "2. CP e anexos/MR", "cover": "mr.pdf"},
                    {"template": "justificativas", "subfolder": "2. CP e anexos/Justificativas
Relevantes", "desc": "Justificativas Relevantes", "cover": "justificativas.pdf"},
        ]

        # Certifique-se de passar o df_registro_selecionado aqui
        dialog = ProgressDialog(documentos_encontrados, documentos, self.icons, self.dados)
        dialog.exec()

    def concatenar_e_abrir_pdfs(self, pdf_paths):
        if not pdf_paths:
            QMessageBox.warning(None, "Erro", "Nenhum PDF foi gerado para concatenar.")
            return

        output_pdf_path = self.pasta_base / self.nome_pasta / "2. CP e anexos" / "CP_e_anexos.pdf"
        merger = PdfMerger()

        try:
            for pdf in pdf_paths:
                if "cover_path" in pdf:
                    merger.append(str(pdf["cover_path"]))
                merger.append(str(pdf["pdf_path"]))

            merger.write(str(output_pdf_path))
            merger.close()

            os.startfile(output_pdf_path)
                print(f"(no gerar_documentos no gerenciador_anexo, gerar_dumentos)PDF concatenado
salvo e aberto: {output_pdf_path}")
        except Exception as e:
            print(f"Erro ao concatenar os PDFs: {e}")
            QMessageBox.warning(None, "Erro", f"Erro ao concatenar os PDFs: {e}")

    def get_latest_pdf(self, directory):
        pdf_files = list(directory.glob("*.pdf"))
        if not pdf_files:
```

```
            return None
        latest_pdf = max(pdf_files, key=os.path.getmtime)
        return latest_pdf

    """def gerar_documento_de_formalizacao_de_demanda(self):
            self.gerarDocumento("dfd", "2. CP e anexos/DFD", "Documento de Formalizacao de Demanda")"""

    def gerar_declaracao_orcamentaria(self):
          self.gerarDocumento("declaracao_orcamentaria", "2. CP e anexos/Declaracao de Adequação Orçamentária", "Declaracao Orcamentaria")

    def gerar_termo_de_referencia(self):
        self.gerarDocumento("tr", "2. CP e anexos/TR", "Termo de Referencia")

    def gerar_aviso_dispensa(self):
        self.gerar_e_abrir_documento("aviso_dispensa", "3. Aviso", "Aviso de Dispensa")

    def abrir_pasta(self, caminho):
        if os.path.exists(caminho):
            os.startfile(caminho)  # Abre o caminho no Windows
```