

## Arquivo: atas.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from datetime import datetime
from pathlib import Path
import pandas as pd
from num2words import num2words
from docxtpl import DocxTemplate
from docx import Document
from docx.shared import Pt
from openpyxl import Workbook, load_workbook
from docx.oxml.ns import nsdecls
from docx.oxml import parse_xml
from openpyxl.styles import Font, PatternFill
from paths import PRE_DEFINICOES_JSON, TEMPLATE_PATH, ORGANIZACOES_FILE, AGENTES_RESPONSAVEIS_FILE
import json
from modules.utils.add_button import add_button_func_vermelho
import os
import sys
import subprocess
from PyQt6.QtSql import QSqlQuery

from modules.utils.linha_layout import linha_divisoria_layout

DEFAULT_CONFIG = {
    "ultimo_cnpj": "0005055505050",
    "ultimo_ano": "2024",
    "ultimo_sequencial": "00290",
    "ordenador_despesas": ["CT IM Siqueira Campos", "CT MARCOS", "CT Romulo"],
    "cidades_combobox": ["Brasília", "Rio de Janeiro", "São Paulo"],
    "org_combobox": ["Ceimbra", "ceimspa", "ceimrj"],
    "initial_text": "Cabeçalho para teste"
}

class GerarAtaWidget(QWidget):
    def __init__(self, icons, database_ata_manager, main_window, parent=None):
        super().__init__(parent)
        self.dataframe_selecionado = None
        self.icons = icons
        self.database_ata_manager = database_ata_manager
        self.main_window = main_window

        # Carrega configurações do JSON
        self.config_data = self.carregar_configuracoes(PRE_DEFINICOES_JSON)
        self.organizacoes_file = ORGANIZACOES_FILE
        self.ordenador_despesas_file = AGENTES_RESPONSAVEIS_FILE

        self.setup_ui()

    def carregar_configuracoes(self, config_path):
```

```

config_path = Path(config_path)

# Verifica se o arquivo existe
if config_path.is_file():
    with open(config_path, 'r', encoding='utf-8') as file:
        config_data = json.load(file)

    # Verifica se todas as chaves necessárias estão presentes e adiciona as ausentes
    for key, value in DEFAULT_CONFIG.items():
        if key not in config_data:
            config_data[key] = value
            QMessageBox.information(
                self,
                "Configuração Atualizada",
                f"A chave ausente '{key}' foi adicionada com o valor padrão."
            )

    # Atualiza o arquivo com as novas chaves (se adicionadas)
    with open(config_path, 'w', encoding='utf-8') as file:
        json.dump(config_data, file, ensure_ascii=False, indent=4)

return config_data
else:
    # Cria o arquivo de configuração com valores padrão
    try:
        with open(config_path, 'w', encoding='utf-8') as file:
            json.dump(DEFAULT_CONFIG, file, ensure_ascii=False, indent=4)

        QMessageBox.information(self, "Informação", "Arquivo de configuração criado com
valores padrão.")
    except Exception as e:
        QMessageBox.critical(self, "Erro", f"Não foi possível criar o arquivo de
configuração: {e}")
    return DEFAULT_CONFIG

def setup_ui(self):
    layout = QVBoxLayout(self)

    # Título principal
    header_title = QLabel("Pré-Definições para Geração de Atas")
    header_title.setAlignment(Qt.AlignmentFlag.AlignCenter)
    header_title.setFont(QFont('Arial', 16, QFont.Weight.Bold))
    layout.addWidget(header_title)

    # Layout horizontal para a seleção
    selecao_layout = QHBoxLayout()

    # Adiciona um espaço flexível antes do QLabel para empurrá-lo para a direita
    spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
    selecao_layout.addItem(spacer)

    # Criação do QLabel
    selecao_label = QLabel("Selecione a Licitação:")

```

```

selecao_layout.addWidget(selecao_label)
selecao_label.setFont(QFont('Arial', 14))

# Criação do ComboBox com tamanho fixo de 200
self.selecao_combobox = QComboBox()
self.selecao_combobox.setFixedWidth(350)
self.selecao_combobox.setFont(QFont('Arial', 12))
selecao_layout.addWidget(self.selecao_combobox)

selecao_layout.addStretch()

# Adiciona o layout ao layout principal
layout.addLayout(selecao_layout)

# Carregar tabelas com "result" no nome para o ComboBox
self.carregar_tabelas_result()
linha_divisoria, spacer_baixo_linhal = linha_divisoria_layout()
layout.addWidget(linha_divisoria)
layout.addItem(spacer_baixo_linhal)

cabecalho_label = QLabel("Defina o cabeçalho:")
layout.addWidget(cabecalho_label)
cabecalho_label.setFont(QFont('Arial', 14))

# Editor de texto para o cabeçalho
self.header_editor = QTextEdit()
initial_text = self.config_data.get("initial_text", "")
self.header_editor.setFont(QFont('Arial', 12))
self.header_editor.setText(initial_text)
layout.addWidget(self.header_editor)

# Combobox de cidades com botão Alterar Pre-definições
cidade_layout = QHBoxLayout()
# Adiciona um espaço flexível antes do QLabel para empurrá-lo para a direita
spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
cidade_layout.addItem(spacer)

cidades_label = QLabel("Selecione a Cidade:")
cidade_layout.addWidget(cidades_label)
cidades_label.setFont(QFont('Arial', 14))

self.cidades_combobox = QComboBox()
self.cidades_combobox.setFixedWidth(350)
self.cidades_combobox.setFont(QFont('Arial', 12))
cidade_layout.addWidget(self.cidades_combobox)

layout.addLayout(cidade_layout)

# Combobox de organizações com botão Alterar Pre-definições
org_layout = QHBoxLayout()
spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
org_layout.addItem(spacer)
org_label = QLabel("Selecione a Organização Gerenciadora:")

```

```

org_layout.addWidget(org_label)
org_label.setFont(QFont('Arial', 14))

self.org_combobox = QComboBox()
self.org_combobox.setFixedWidth(350)
self.org_combobox.setFont(QFont('Arial', 12))
org_layout.addWidget(self.org_combobox)

layout.addLayout(org_layout)

# Combobox de ordenador de despesas com botão Alterar Pre-definições
despesa_layout = QBoxLayout()
spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
despesa_layout.addItem(spacer)
despesa_label = QLabel("Selecione o Ordenador de Despesas:")
despesa_layout.addWidget(despesa_label)
despesa_label.setFont(QFont('Arial', 14))

self.ordenador_despesa_combobox = QComboBox()
self.ordenador_despesa_combobox.setFixedWidth(350)
self.ordenador_despesa_combobox.setFixedHeight(65)
self.ordenador_despesa_combobox.setFont(QFont('Arial', 12))
despesa_layout.addWidget(self.ordenador_despesa_combobox)

layout.addLayout(despesa_layout)

# Layout horizontal para número de controle
numero_layout = QBoxLayout()
linha_divisoria, spacer_baixo_linha = linha_divisoria_layout()
numero_layout.addWidget(linha_divisoria)
numero_layout.addSpacerItem(spacer_baixo_linha)
# Rótulo e campo para número de controle
rotulo = QLabel("Digite o próximo Número de Controle:")
despesa_layout.addWidget(rotulo)
rotulo.setFont(QFont('Arial', 14))

numero_layout.addWidget(rotulo)

self.numero_controle_lineedit = QLineEdit()
self.numero_controle_lineedit.setFixedWidth(100)
self.numero_controle_lineedit.setFont(QFont('Arial', 12))

numero_layout.addWidget(self.numero_controle_lineedit)

linha_divisoria2, spacer_baixo_linha2 = linha_divisoria_layout()
numero_layout.addWidget(linha_divisoria2)
numero_layout.addSpacerItem(spacer_baixo_linha2)

# Adiciona um espaço flexível antes do QLabel para empurrá-lo para a direita
spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
layout.addItem(spacer)

# Adiciona o layout horizontal ao layout principal

```

```

layout.addWidget(numero_layout)

spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
layout.addItem(spacer)

# Layout horizontal para centralizar o botão
button_layout = QHBoxLayout()
button_layout.addStretch()
add_button_func_vermelho("Gerar Ata", self.gerar_ata, button_layout, "Clique para gerar a ata", button_size=(300, 40))
button_layout.addStretch()
# Adiciona o layout de botão centralizado ao layout principal
layout.addLayout(button_layout)
layout.addStretch()

self.carregar_dados_comboboxes()
self.carregar_dados_ordenador_despesa_comboboxes()

def carregar_dados_comboboxes(self):
    try:
        # Verifica se o arquivo existe
        if not self.organizacoes_file.is_file():
            raise FileNotFoundError(f"Arquivo {self.organizacoes_file} não encontrado.")

        # Carrega o JSON
        with open(self.organizacoes_file, "r", encoding="utf-8") as file:
            data = json.load(file)

            # Extração de dados únicos
            organizacoes = data.get("organizacoes", [])
            cidades = {org.get("Cidade") for org in organizacoes if "Cidade" in org}
            organizacoes_formatadas = {f"{org.get('Nome')}, {org.get('Sigla')}" for org in organizacoes if "Nome" in org and "Sigla" in org}

            # Adiciona os valores aos comboboxes
            if cidades:
                self.cidades_combobox.addItems(sorted(cidades))
            else:
                self.cidades_combobox.addItem("Adicione a cidade nas configurações.")

            if organizacoes_formatadas:
                self.org_combobox.addItems(sorted(organizacoes_formatadas))
            else:
                self.org_combobox.addItem("Adicione a organização nas configurações.")

    except (json.JSONDecodeError, KeyError) as e:
        QMessageBox.critical(self, "Erro", f"Erro ao carregar dados do arquivo: {e}")
        self.cidades_combobox.addItem("Adicione a cidade nas configurações.")
        self.org_combobox.addItem("Adicione a organização nas configurações.")

    except FileNotFoundError as e:
        QMessageBox.warning(self, "Aviso", str(e))
        self.cidades_combobox.addItem("Adicione a cidade nas configurações.")

```

```

        self.org_combobox.addItem("Adicione a organização nas configurações.")

def carregar_dados_ordenador_despesa_comboboxes(self):
    try:
        # Verifica se o arquivo existe
        if not self.ordenador_despesas_file.is_file():
            raise FileNotFoundError(f"Arquivo {self.ordenador_despesas_file} não encontrado.")

        # Carrega o JSON
        with open(self.ordenador_despesas_file, "r", encoding="utf-8") as file:
            data = json.load(file)

        # Extração de dados únicos
        ordenador_despesa = data.get("ordenador_de_despesa", [])
        ordenador_despesas_formatado = [
            f"{od.get('Nome')}\n{od.get('Posto')}\n{od.get('Funcao')}"
            for od in ordenador_despesa if "Nome" in od and "Posto" in od and "Funcao" in od
        ]

        # Adiciona os valores aos comboboxes
        if ordenador_despesas_formatado:
            self.ordenador_despesa_combobox.addItems(sorted(ordenador_despesas_formatado))
        else:
            self.ordenador_despesa_combobox.addItem("Adicione o Ordenador de Despesa nas configurações.")

    except (json.JSONDecodeError, KeyError) as e:
        QMessageBox.critical(self, "Erro", f"Erro ao carregar dados do arquivo: {e}")
        self.ordenador_despesa_combobox.addItem("Adicione o Ordenador de Despesa nas configurações.")

    except FileNotFoundError as e:
        QMessageBox.warning(self, "Aviso", str(e))
        self.ordenador_despesa_combobox.addItem("Adicione o Ordenador de Despesa nas configurações.")

def carregar_tabelas_result(self):
    # Obtém tabelas cujo nome começa com "result"
    tabelas_result = self.database_ata_manager.get_tables_with_keyword("result")

    # Limpa o ComboBox antes de adicionar novos itens
    self.selecao_combobox.clear()

    # Verifica a quantidade de tabelas encontradas
    if len(tabelas_result) == 1:
        # Carregar automaticamente a tabela se apenas uma for encontrada
        tabela = tabelas_result[0]
        self.dataframe_selecionado = self.database_ata_manager.load_table_to_dataframe(tabela)

        # Adiciona o nome ao ComboBox e define como selecionado (opcional para visualização)
        self.selecao_combobox.addItem(tabela)
        self.selecao_combobox.setCurrentIndex(0)

    # Print para verificação

```

```

print(f"Tabela única '{tabela}' carregada automaticamente:")
print(self.dataframe_selecionado)

else:
    # Adiciona todas as tabelas ao ComboBox para seleção manual
    for tabela in tabelas_result:
        if tabela.startswith("result"):
            self.selecao_combobox.addItem(tabela)

    # Conecta a seleção do ComboBox para carregar a tabela selecionada

self.selecao_combobox.currentIndexChanged.connect(self.atualizar_dataframe_selecionado)

def atualizar_dataframe_selecionado(self):
    # Obter o nome da tabela selecionada
    tabela = self.selecao_combobox.currentText()

    if tabela:
        # Carregar a tabela selecionada como DataFrame
        self.dataframe_selecionado = self.database_ata_manager.load_table_to_dataframe(tabela)

        # Verifica se o DataFrame foi carregado corretamente e imprime o conteúdo
        if self.dataframe_selecionado is not None:
            print(f"Tabela '{tabela}' carregada com sucesso.")
            print(self.dataframe_selecionado) # Exibe o DataFrame carregado
        else:
            print(f"Erro ao carregar a tabela '{tabela}'. Verifique o banco de dados.")

        # Exibe o tipo e se está vazio ou não
        print(f"Tipo de self.dataframe_selecionado: {type(self.dataframe_selecionado)}")
        print(f"DataFrame está vazio: {self.dataframe_selecionado.empty} if self.dataframe_selecionado is not None else 'None'")

def salvar_configuracoes(self):
    with open(PRE_DEFINICOES_JSON, 'w', encoding='utf-8') as file:
        json.dump(self.config_data, file, ensure_ascii=False, indent=4)

def gerar_ata(self):
    # Obter os valores inseridos pelo usuário
    header_text = self.header_editor.toPlainText()
    cidade_selecionada = self.cidados_combobox.currentText()
    organizacao_selecionada = self.org_combobox.currentText()
    ordenador_despesas = self.ordenador_despesa_combobox.currentText()
    numero_controle = self.numero_controle_lineedit.text()

    # Mapear os campos aos seus respectivos rótulos
    campos = {
        "Cabeçalho": header_text,
        "Cidade": cidade_selecionada,
        "Organização Gerenciadora": organizacao_selecionada,
        "Ordenador de Despesas": ordenador_despesas,
        "Número de Controle": numero_controle,
    }

```

```

# Identificar os campos não preenchidos
campos_nao_preenchidos = [campo for campo, valor in campos.items() if not valor.strip()]

if campos_nao_preenchidos:
    # Exibir mensagem com os campos faltantes
    QMessageBox.warning(
        self,
        "Aviso",
        f"Por favor, preencha os seguintes campos: {', '.join(campos_nao_preenchidos)}."
    )
    return

# Verifica se self.dataframe_selecionado foi carregado corretamente antes de continuar
if self.dataframe_selecionado is None or self.dataframe_selecionado.empty:
    QMessageBox.warning(self, "Aviso", "Dados de homologação não disponíveis.")
    print("self.dataframe_selecionado está vazio ou não foi carregado corretamente.")
    return

# Confirmação final de que o DataFrame está pronto para ser usado
print("self.dataframe_selecionado antes de processar a ata:")
print(self.dataframe_selecionado)

# Passando para a próxima função
self.processar_ata_de_registro_de_precos(
    header_text, cidade_selecionada, organizacao_selecionada, ordenador_despesas,
numero_controle, self.dataframe_selecionado
)

# Abrir o diretório principal onde as subpastas foram criadas
if hasattr(self, 'pasta_principal_criada') and self.pasta_principal_criada:
    path = str(self.pasta_principal_criada)

    if sys.platform == "win32":  # Windows
        os.startfile(path)
    elif sys.platform == "darwin":  # macOS
        subprocess.Popen(["open", path])
    else:  # Linux (Ubuntu, Debian, etc.)
        subprocess.Popen(["xdg-open", path])

def processar_ata_de_registro_de_precos(self, header_text, cidade, organizacao,
ordenador_despesas, numero_controle, dataframe):
    print(f"Tipo em processar_ata_de_registro_de_precos: {type(dataframe)}")

    if not all([header_text, cidade, organizacao, ordenador_despesas, numero_controle,
dataframe is not None]):
        QMessageBox.warning(self, "Erro", "Informações incompletas para processar a ata de
registro de preços.")
        return

    criar_pastas_com_subpastas(dataframe)
    self.processar_ata(header_text, cidade, organizacao, ordenador_despesas, numero_controle,
dataframe)

```

```

# Exibir mensagem de sucesso
QMessageBox.information(self, "Sucesso", "Ata de registro de preços processada com
sucesso!")

def processar_ata(self, header_text, cidade, organizacao, ordenador_despesas, numero_controle,
dataframe):
    print(f"Tipo em processar_ata: {type(dataframe)}")

    # Iniciar controle de número sequencial com o valor inicial passado
    numero_controle_atual = int(numero_controle)

    # 1. Obter a lista de empresas únicas
    empresas_unicas = dataframe['empresa'].dropna().unique()

    if 'numero_ata' not in dataframe.columns:
        dataframe['numero_ata'] = None

    # 2. Loop através de cada empresa única
    for empresa in empresas_unicas:
        # 3. Obter todos os registros (itens) para esta empresa
        registros_empresa = dataframe[dataframe['empresa'] == empresa]

        # Pega os dados do pregão do primeiro item (para nomear a pasta)
        # Assumimos que, para o propósito da pasta, qualquer um dos pregões serve.
        if not registros_empresa.empty:
            primeiro_registro = registros_empresa.iloc[0]
            num_pregao = primeiro_registro['num_pregao']
            ano_pregao = primeiro_registro['ano_pregao']

            if not pd.isna(num_pregao) and not pd.isna(ano_pregao):
                # Cria o diretório para a empresa
                path_subpasta = criar_diretorio(Path.cwd() / "relatorios", int(num_pregao),
int(ano_pregao), empresa)

                # Armazena o caminho do diretório principal
                self.pasta_principal_criada = path_subpasta.parent

                # 4. Processa a ata para a empresa com TODOS os seus itens
                numero_ata = self.processar_empresa(registros_empresa, empresa, path_subpasta,
numero_controle_atual)
                dataframe.loc[dataframe['empresa'] == empresa, 'numero_ata'] = numero_ata

                # 5. Incrementa o número de controle apenas uma vez por empresa
                numero_controle_atual += 1

    print(dataframe[['numero_ata', 'item', 'empresa']])

def processar_empresa(self, registros_empresa, empresa, path_subpasta, numero_controle):
    if registros_empresa.empty:
        print(f"Nenhum registro encontrado para a empresa: {empresa}")
        return None

```

```

# Obtem o primeiro registro como dicionário
registro = registros_empresa.iloc[0].to_dict()

# Verifica se o registro possui CNPJ
cnpj = registro.get("cnpj")
if cnpj:
    # Remove qualquer formatação existente (mantendo apenas números)
    cnpj = ''.join(filter(str.isdigit, cnpj))

    # Formata como CNPJ ou CPF com base na quantidade de dígitos
    if len(cnpj) == 14:
        cnpj = f"{cnpj[:2]}.{cnpj[2:5]}.{cnpj[5:8]}/{cnpj[8:12]}-{cnpj[12:]}"
    elif len(cnpj) == 11:
        cnpj = f"{cnpj[:3]}.{cnpj[3:6]}.{cnpj[6:9]}-{cnpj[9:]}"
    else:
        print(f"Formato de CNPJ/CPF inválido para o valor: {cnpj}")
        return None

    # Exibe o CNPJ formatado para depuração
    print(f"CNPJ formatado para consulta: {cnpj}")

    # Consulta a tabela registro_sicaf pelo CNPJ formatado
    registro_sicaf = self.database_ata_manager.consultar_registro("registro_sicaf",
"cnpj", cnpj)

    # Atualiza os valores de registro com os dados obtidos de registro_sicaf e imprime
atualizações
    if registro_sicaf:
        campos_atualizados = {}
        for campo in ["nome_fantasia", "endereco", "municipio", "uf", "cep", "telefone",
"email", "responsavel_legal"]:
            if campo in registro_sicaf:
                registro[campo] = registro_sicaf[campo]
                campos_atualizados[campo] = registro[campo]
            else:
                registro[campo] = None # Define valor padrão se o campo estiver ausente

        # Exibe quais campos foram atualizados
        print(f"Campos atualizados para CNPJ {cnpj}: {campos_atualizados}")
    else:
        print(f"Nenhum registro encontrado para CNPJ {cnpj} na tabela registro_sicaf.")

itens_relacionados = registros_empresa.to_dict('records')
num_contrato = self.gerar_numero_contrato(registro, numero_controle)

# Verificar se os valores de `registro` foram atualizados antes de montar o contexto
context = self.montar_contexto_documento(registro, empresa, itens_relacionados,
num_contrato)

self.salvar_documento_empresa(path_subpasta, empresa, context, registro,
itens_relacionados)

return num_contrato

```

```

def gerar_numero_contrato(self, registro, numero_controle):
    ano_atual = datetime.now().year
    return f'{registro['uasg']} /{ano_atual}-{numero_controle:03}/00'

def montar_contexto_documento(self, registro, empresa, itens_relacionados, num_contrato):
    # Exibe o conteúdo de `registro` para verificar se os valores estão corretos
    # print("Conteúdo de registro em montar_contexto_documento:", registro)

    texto_substituto = f"Pregão Eletrônico nº
{registro['num_pregao']} /{registro['ano_pregao']} \n{num_contrato}"
    soma_valor_homologado = gerar_soma_valor_homologado(itens_relacionados)
    ordenador_despesa = self.ordenador_despesa_combobox.currentText()

    return {
        "num_pregao": registro['num_pregao'],
        "ano_pregao": registro['ano_pregao'],
        "empresa": empresa,
        "uasg": registro['uasg'],
        "numero_ata": self.numero_controle_lineedit.text(),
        "soma_valor_homologado": soma_valor_homologado,
        "cabecalho": texto_substituto,
        "dados_ug_contratante": self.header_editor.toPlainText(),
        "contrato": num_contrato,
        "cnpj": registro["cnpj"],
        "objeto": registro["objeto"],
        "ordenador_despesa": ordenador_despesa,
        "cidade": self.cidades_combobox.currentText(),
        "organizacao": self.org_combobox.currentText()
    }

def salvar_documento_empresa(self, path_subpasta, empresa, context, registro, itens_relacionados):
    self.salvar_documento(path_subpasta, empresa, context, registro, itens_relacionados, context["contrato"])

def limpar_nome_empresa(self, nome_empresa):
    # Substituir '/' e ':' por sublinhado
    caracteres_para_sublinhado = [ '/', ':' ]
    for char in caracteres_para_sublinhado:
        nome_empresa = nome_empresa.replace(char, '_')

    # Substituir '.' por nada (remover)
    nome_empresa = nome_empresa.replace('.', '')

    # Substituir outros caracteres inválidos por sublinhados
    caracteres_invalidos = [ '<', '>', '_', '"', '\\', '|', '?', '*' ]
    for char in caracteres_invalidos:
        nome_empresa = nome_empresa.replace(char, '_')

    # Remover espaços extras e sublinhados no final do nome da empresa
    nome_empresa = nome_empresa.rstrip(' _')

```

```

# Substituir múltiplos espaços ou sublinhados consecutivos por um único sublinhado
nome_empresa = '_'.join(filter(None, nome_empresa.split(' ')))

return nome_empresa

def salvar_documento(self, path_subpasta, empresa, context, registro, itens_relacionados,
num_contrato):
    max_len = 40
    contrato_limpo = self.limpar_nome_empresa(num_contrato)[:max_len].rstrip()

    template_path = Path(TEMPLATE_PATH)
    if not template_path.exists():
        QMessageBox.critical(self, "Erro", f"O template '{template_path}' não foi
encontrado.\nVerifique o caminho e tente novamente.")
        return

    try:
        tpl = DocxTemplate(template_path)
        tpl.render(context)

        nome_documento = f"{contrato_limpo}.docx"
        path_documento = path_subpasta / nome_documento

        # Salvar o documento e incluir informações detalhadas
        tpl.save(path_documento)
        self.alterar_documento_criado(path_documento, registro, registro["cnpj"], itens_relacionados)
        self.salvar_email(path_subpasta, context, registro)

    except Exception as e:
        QMessageBox.critical(self, "Erro", f"Ocorreu um erro ao salvar o documento:\n{e}")

def salvar_email(self, path_subpasta, context, registro):
    nome_arquivo_txt = "E-mail.txt"
    path_arquivo_txt = path_subpasta / nome_arquivo_txt
    with open(path_arquivo_txt, "w") as arquivo_txt:
        texto_email = (f"{{registro['email']} }\n\n"
                       f"Sr. Representante.\n\n"
                       f"Encaminho em anexo a Vossa Senhoria a ATA {context['contrato']} "
                       f"decorrente do Pregão Eletrônico (SRP) nº
{context['num_pregao']}/{context['ano_pregao']}, do Centro "
                       f"de Intendência da Marinha em Brasília (CeIMBra).\n\n"
                       f"Os documentos deverão ser conferidos, assinados e devolvidos a este
Comando.\n\n"
                       f"A empresa receberá uma via, devidamente assinada, após a
publicação.\n\n"
                       f"Respeitosamente,\n")
        arquivo_txt.write(texto_email)

def alterar_documento_criado(self, caminho_documento, registro, cnpj, itens):
    doc = Document(caminho_documento)

```

```

for paragraph in doc.paragraphs:
    if '{relacao_empresa}' in paragraph.text:
        paragraph.clear()
        self.inserir_relacao_empresa(paragraph, registro, cnpj)

    if '{relacao_item}' in paragraph.text:
        paragraph.clear()
        inserir_relacao_itens(paragraph, itens)

diretorio_documento = Path(caminho_documento).parent
caminho_arquivo_excel = diretorio_documento / 'relacao.xlsx'
gerar_excel_relacao_itens(itens, caminho_arquivo_excel)

inserir_tabela_no_documento(doc, caminho_arquivo_excel)
doc.save(caminho_documento)

def inserir_relacao_empresa(self, paragrafo, registro, cnpj):
    dados = {
        "Razão Social": registro.get("empresa", ""),
        "Nome Fantasia": registro.get("nome_fantasia", ""),
        "CNPJ": registro.get("cnpj", ""),
        "Endereço": registro.get("endereco", ""),
        "Município-UF": f'{registro.get("municipio", '')} - {registro.get("uf", '')}',
        "CEP": registro.get("cep", ""),
        "Telefone": registro.get("telefone", ""),
        "E-mail": registro.get("email", "")
    }

    total_itens = len(dados)
    contador = 1

    for chave, valor in dados.items():
        adicione_texto_formatado(paragrafo, f'{chave}: ', True)
        if contador == total_itens - 1:
            adicione_texto_formatado(paragrafo, f'{valor}; \n', False)
        elif contador == total_itens:
            adicione_texto_formatado(paragrafo, f'{valor}.\n', False)
        else:
            adicione_texto_formatado(paragrafo, f'{valor};\n', False)

        contador += 1

    adicione_texto_formatado(paragrafo, "Representada neste ato, por seu representante legal,\n"
o(a) Sr(a) ", False)
    adicione_texto_formatado(paragrafo, f'{registro.get("responsavel_legal", "")}.\n', False)

def abrir_pastas_criadas(self):
    # Caminho base onde as pastas foram criadas
    relatorio_path = Path.cwd() / "relatorios"

    if relatorio_path.exists():
        # Abre o diretório usando o explorador de arquivos padrão do sistema

```

```

    QDesktopServices.openUrl(QUrl.fromLocalFile(str(relatorio_path)))
else:
    QMessageBox.warning(self, "Aviso", "O diretório de relatórios não existe.")

def adicione_texto_formatado(paragraph, text, bold=False):
    run = paragraph.add_run(text)
    run.bold = bold
    font = run.font
    font.name = 'Calibri'
    font.size = Pt(12)

def criar_pastas_com_subpastas(dataframe) -> None:
    if dataframe is None or dataframe.empty:
        QMessageBox.warning(None, "Erro", "Padrão de pregão não encontrado. Por favor, carregue um banco de dados antes de continuar.")
    return

    # Definir o caminho base onde as pastas serão criadas
    # Por exemplo, você pode definir um diretório "relatorios" no diretório atual
    relatorio_path = Path.cwd() / "relatorios"
    relatorio_path.mkdir(parents=True, exist_ok=True)

    combinacoes = dataframe[['num_pregao', 'ano_pregao', 'empresa']].drop_duplicates().values

    pastas_criadas = set()

    for num_pregao, ano_pregao, empresa in combinacoes:
        if pd.isna(num_pregao) or pd.isna(ano_pregao) or pd.isna(empresa):
            continue

        chave_pasta = (int(num_pregao), int(ano_pregao), empresa)
        if chave_pasta not in pastas_criadas:
            # Criar o diretório para a empresa dentro do caminho de relatórios
            criar_diretorio(relatorio_path, int(num_pregao), int(ano_pregao), empresa)
            print(f"Criado 1 diretório para {empresa}")
            pastas_criadas.add(chave_pasta)

def gerar_soma_valor_homologado(itens):
    valor_total = sum(float(item["valor_homologado_total_item"]) or 0 for item in itens)
    valor_extenso = valor_por_extenso(valor_total)
    return f'R$ {formatar_brl(valor_total)} ({valor_extenso})'

def formatar_brl(valor):
    try:
        if valor is None:
            return "Não disponível" # Retorna uma string informativa caso o valor seja None
        # Formata o número no formato monetário brasileiro sem utilizar a biblioteca locale
        return f'R$ {float(valor):,.2f}'.replace(",", "X").replace(".", ",").replace("X", ".")
    except (ValueError, TypeError):
        return "Valor inválido" # Retorna isso se não puder converter para float

def valor_por_extenso(valor):
    extenso = num2words(valor, lang='pt_BR', to='currency')

```

```

return extenso.capitalize()

def criar_diretorio(base_path: Path, num_pregao: int, ano_pregao: int, nome_empresa: str) -> Path:
    nome_dir_principal = f"PE {num_pregao}-{ano_pregao}"
    path_dir_principal = base_path / nome_dir_principal

    if not path_dir_principal.exists():
        path_dir_principal.mkdir(parents=True)
        print(f"Criado diretório principal: {path_dir_principal}")

    nome_empresa_limpa = limpar_nome_empresa(nome_empresa)
    path_subpasta = path_dir_principal / nome_empresa_limpa

    if not path_subpasta.exists():
        path_subpasta.mkdir(parents=True)
        print(f"Criado subdiretório: {path_subpasta}")
    else:
        pass
        # print(f"O subdiretório já existe e não será recriado: {path_subpasta}")

    return path_subpasta

def limpar_nome_empresa(nome_empresa):
    # Substituir '/' e ':' por sublinhado
    nome_empresa = nome_empresa.replace('/', '_').replace(':', '_')

    # Substituir '.' por nada (remover)
    nome_empresa = nome_empresa.replace('.', '')

    # Substituir outros caracteres inválidos por sublinhados
    caracteres_invalidos = ['<', '>', '"', '\'', '|', '?', '*']
    for char in caracteres_invalidos:
        nome_empresa = nome_empresa.replace(char, '_')

    # Remover espaços extras e sublinhados no final do nome da empresa
    nome_empresa = nome_empresa.rstrip(' _')

    # Substituir múltiplos espaços ou sublinhados consecutivos por um único sublinhado
    nome_empresa = '_'.join(filter(None, nome_empresa.split(' ')))

    # Remover duplicatas de sublinhados causados por espaços ou caracteres inválidos
    while '__' in nome_empresa:
        nome_empresa = nome_empresa.replace('__', '_')

    return nome_empresa.upper()

def inserir_relacao_itens(paragrafo, itens):
    # Limpar parágrafo
    paragrafo.clear()

    # # Iterar sobre os itens e gerar campos
    # for item in itens:
    #     campos = gerar_campos_item(item)

```

```

#     if campos:
#         for texto, negrito in campos:
#             adicione_texto_formatado(paragrafo, texto + '\n', negrito)

# Calcular e adicionar o valor total homologado
valor_total = sum(float(item["valor_homologado_total_item"] or 0) for item in itens)
valor_extenso = valor_por_extenso(valor_total)
texto_soma_valor_homologado = f'{formatar_brl(valor_total)} ({valor_extenso})'
adicione_texto_formatado(paragrafo, 'Valor total homologado para a empresa:\n', False)
adicione_texto_formatado(paragrafo, texto_soma_valor_homologado + '\n', True)

# # Chamar a função para gerar o Excel
# gerar_excel_relacao_itens(itens)

return texto_soma_valor_homologado

def gerar_excel_relacao_itens(itens, caminho_arquivo_excel='relacao_itens.xlsx'):
    # Ordenar os itens, primeiro por 'grupo' (None será considerado menor) e depois por 'item'
    try:
        itens_ordenados = sorted(itens, key=lambda x: (
            x['grupo'] if x['grupo'] is not None else '',
            int(x['item']) # Convertendo o item para inteiro antes de ordenar
        ))
    except (ValueError, TypeError):
        # Fallback para o caso de algum item não ser um número válido
        itens_ordenados = sorted(itens, key=lambda x: (x['grupo'] if x['grupo'] is not None else '',
                                                       x['item']))

    # # Print do ordenamento
    # for item in itens_ordenados:
    #     print(f"Grupo: {item['grupo']} - Item Num: {item['item']}")

    wb = Workbook()
    ws = wb.active

    fonte_tamanho_12_cinza = Font(size=12, bold=True)
    fundo_cinza = PatternFill(start_color="D3D3D3", end_color="D3D3D3", fill_type="solid")
    fonte_tamanho_10 = Font(size=10)

    linha_atual = 1

    for item in itens_ordenados:
        ws.merge_cells(start_row=linha_atual, start_column=1, end_row=linha_atual, end_column=3)

        # Adicionando lógica para verificar o valor de grupo
        if isinstance(item['grupo'], (int, float)): # Verifica se 'grupo' é um número
            valor_cell_1 = f"Grupo {item['grupo']} - Item {item['item']} - {item['descricao']}"
        ({item['catalogo']})"
        else:
            valor_cell_1 = f"Item {item['item']} - {item['descricao']} ({item['catalogo']})"

    cell_1 = ws.cell(row=linha_atual, column=1, value=valor_cell_1)

```

```

cell_1.font = fonte_tamanho_12_cinza
cell_1.fill = fundo_cinza

linha_atual += 1
ws.merge_cells(start_row=linha_atual, start_column=1, end_row=linha_atual, end_column=3)
cell_2 = ws.cell(row=linha_atual, column=1, value=f"{item['descricao_detalhada']}") 
cell_2.font = fonte_tamanho_10

linha_atual += 1
ws.cell(row=linha_atual, column=1, value=f"UF: {str(item['unidade'])}").font =
fonte_tamanho_10
ws.cell(row=linha_atual, column=2, value=f"Marca: {item['marca_fabricante']}").font =
fonte_tamanho_10
ws.cell(row=linha_atual, column=3, value=f"Modelo: {item['modelo_versao']}").font =
fonte_tamanho_10

linha_atual += 1

# Removendo o sufixo ".0" de quantidade, se presente
quantidade_formatada = str(int(item['quantidade'])) if item['quantidade'] ==
int(item['quantidade']) else str(item['quantidade'])

# Aplicando a formatação corrigida na célula
ws.cell(row=linha_atual, column=1, value=f"Quantidade: {quantidade_formatada}").font =
fonte_tamanho_10

# Convertendo valores para número antes de formatar
valor_unitario = float(item['valor_homologado_item_unitario'])
valor_total = float(item['valor_homologado_total_item'])

# Valor Unitário
# print(f"Processando Valor Unitário para o item {item['item']}")
valor_unitario_formatado = format_currency(valor_unitario)
ws.cell(row=linha_atual, column=2, value=f"Valor Unitário: {valor_unitario_formatado}").font =
fonte_tamanho_10

# Valor Total
# print(f"Processando Valor Total para o item {item['item']}")
valor_total_formatado = format_currency(valor_total)
ws.cell(row=linha_atual, column=3, value=f"Valor Total: {valor_total_formatado}").font =
fonte_tamanho_10

linha_atual += 1

wb.save(caminho_arquivo_excel)

def format_currency(value):
    """ Função para formatar valores monetários no formato brasileiro. """
    # print(f"Valor original: {value}") # Print para depuração

    if isinstance(value, str):
        # Tentar converter string para float
        try:

```

```

        value = float(value.replace('. ', '').replace(' , ', '.'))
    except ValueError:
        return value # Se a conversão falhar, retorna o valor original
    elif not isinstance(value, (int, float)):
        return value # Retorna o valor original se não for um número

    # Formatar o valor como moeda brasileira
    formatted_value = f'R$ {value:.2f}'.replace(',', 'temp').replace('.', ',').replace('temp',
    ',')

    # print(f"Valor formatado: {formatted_value}") # Print para depuração
    return formatted_value

def inserir_tabela_no_documento(doc, caminho_arquivo_excel):
    # Carregar a planilha do Excel
    wb = load_workbook(caminho_arquivo_excel)
    ws = wb.active

    for paragraph in doc.paragraphs:
        if '<<tbl_itens>>' in paragraph.text:
            # Remover o texto do marcador
            paragraph.text = paragraph.text.replace('<<tbl_itens>>', '')

            # Criar uma nova tabela no documento Word
            table = doc.add_table(rows=0, cols=3)

            for i in range(0, ws.max_row, 4):
                # Primeira linha (mesclada e pintada de cinza claro)
                row_cells = table.add_row().cells
                row_cells[0].merge(row_cells[2])
                run = row_cells[0].paragraphs[0].add_run(str(ws.cell(row=i+1, column=1).value))
                run.font.size = Pt(12)
                run.font.bold = True
                run.font.name = "Calibri"
                shading_elm = parse_xml(r'<w:shd {} w:fill="E3E3E3"/>'.format(nsdecls('w')))
                row_cells[0]._element.get_or_add_tcPr().append(shading_elm)

                # Segunda linha (mesclada com "Descrição Detalhada:" em negrito e quebras de
                linha)
                row_cells = table.add_row().cells
                row_cells[0].merge(row_cells[2])

                # Adiciona quebra de linha antes de "Descrição Detalhada:"
                run = row_cells[0].paragraphs[0].add_run("\n")
                run.font.name = "Calibri"

                # Adiciona "Descrição Detalhada:" em negrito
                run = row_cells[0].paragraphs[0].add_run("Descrição Detalhada:")
                run.font.size = Pt(12)
                run.font.bold = True
                run.font.name = "Calibri"

                # Adicionando o texto restante após "Descrição Detalhada:"

```

```

    texto_segunda_linha = str(ws.cell(row=i+2, column=1).value)
    run = row_cells[0].paragraphs[0].add_run(f" {texto_segunda_linha}")
    run.font.size = Pt(12)
    run.font.name = "Calibri"

    # Adiciona quebra de linha após o texto
    row_cells[0].paragraphs[0].add_run("\n")

    # Terceira linha (manter formatação padrão)
    row_cells = table.add_row().cells
    for j in range(3):
        value = ws.cell(row=i+3, column=j+1).value
        if value is not None:
            texto = str(value)
            if j == 0 and texto.startswith("UF:"):
                run = row_cells[j].paragraphs[0].add_run("UF:")
                run.font.size = Pt(12)
                run.font.bold = True
                run.font.name = "Calibri"
                run = row_cells[j].paragraphs[0].add_run(texto[3:])
                run.font.size = Pt(12)
                run.font.name = "Calibri"
            elif j == 1 and texto.startswith("Marca:"):
                run = row_cells[j].paragraphs[0].add_run("Marca:")
                run.font.size = Pt(12)
                run.font.bold = True
                run.font.name = "Calibri"
                run = row_cells[j].paragraphs[0].add_run(texto[6:])
                run.font.size = Pt(12)
                run.font.name = "Calibri"
            elif j == 2 and texto.startswith("Modelo:"):
                run = row_cells[j].paragraphs[0].add_run("Modelo:")
                run.font.size = Pt(12)
                run.font.bold = True
                run.font.name = "Calibri"
                run = row_cells[j].paragraphs[0].add_run(texto[7:])
                run.font.size = Pt(12)
                run.font.name = "Calibri"
            else:
                row_cells[j].text = texto
                run = row_cells[j].paragraphs[0].runs[0]
                run.font.size = Pt(12)
                run.font.name = "Calibri"

    # Quarta linha (manter formatação padrão)
    row_cells = table.add_row().cells
    for j in range(3):
        value = ws.cell(row=i+4, column=j+1).value
        if value is not None:
            texto = str(value)
            if j == 0 and texto.startswith("Quantidade:"):
                run = row_cells[j].paragraphs[0].add_run("Quantidade:")
                run.font.size = Pt(12)

```

```

        run.font.bold = True
        run.font.name = "Calibri"
        run = row_cells[j].paragraphs[0].add_run(texto[11:])
        run.font.size = Pt(12)
        run.font.name = "Calibri"
        row_cells[j].paragraphs[0].add_run("\n")
    elif j == 1 and texto.startswith("Valor Unitário:"):
        run = row_cells[j].paragraphs[0].add_run("Valor Unitário:")
        run.font.size = Pt(12)
        run.font.bold = True
        run.font.name = "Calibri"
        run = row_cells[j].paragraphs[0].add_run(texto[15:])
        run.font.size = Pt(12)
        run.font.name = "Calibri"
        row_cells[j].paragraphs[0].add_run("\n")
    elif j == 2 and texto.startswith("Valor Total:"):
        run = row_cells[j].paragraphs[0].add_run("Valor Total:")
        run.font.size = Pt(12)
        run.font.bold = True
        run.font.name = "Calibri"
        run = row_cells[j].paragraphs[0].add_run(texto[12:])
        run.font.size = Pt(12)
        run.font.name = "Calibri"
        row_cells[j].paragraphs[0].add_run("\n")
    else:
        row_cells[j].text = texto
        run = row_cells[j].paragraphs[0].runs[0]
        run.font.size = Pt(12)
        run.font.name = "Calibri"
        row_cells[j].paragraphs[0].add_run("\n")
# Mover a tabela para logo após o parágrafo atual
move_table_after_paragraph(paragraph, table)
break

def move_table_after_paragraph(paragraph, table):
    # Move a tabela para ficar logo após o parágrafo atual
    tbl, p = table._tbl, paragraph._element
    p.addnext(tbl)

```