Arquivo: msg_alert.py

```
from datetime import datetime
import logging
import os
import re
from pathlib import Path

import pandas as pd
from pandas.tseries.offsets import BDay

from PyQt6.QtCore import *
from PyQt6.QtGui import *
from PyQt6.QtWidgets import *

from database.utils.treeview_utils import load_images, create_button
from diretorios import MSG_CONTRATOS_DIR

class MensagemDialog(QDialog):
    def __init__(self, df_registro_selecionado, icons_dir, indice_linha, parent=None):
        super().__init__(parent)
        self.df_registro_selecionado = df_registro_selecionado
        self.icons_dir = Path(icons_dir)
        self.indice_linha = indice_linha

        # Atributo para armazenar o cabeçalho atual
        self.cabecalhoAtual = self.gerarCabecalho("alerta_prazo")



        self.templatePath = Path(MSG_CONTRATOS_DIR) / "msg_alerta_prazo.txt"
        self.setWindowTitle("Enviar Mensagem de Alerta")
        self.resize(1400, 700)

        self.image_cache = load_images(self.icons_dir, ["apply.png", "copy.png", "mensagem.png"])



        # Ajusta o estilo para definir a fonte como 14px
        self.setStyleSheet("""
            QDialog {
                font-size: 16px;
            }
            QTextEdit, QListWidget {
                font-size: 16px;
            }
            QGroupBox {
                font-size: 18px;
            }
            QPushButton {
                font-size: 14px;
            }
            QGroupBox::title {
                subcontrol-origin: margin;
```

```
            padding-top: -10px;
            color: #2196F3;
        }
    """)

    self.mainLayout = QVBoxLayout(self)

    # Configuração dos GroupBoxes e seus layouts
    self.setupGroupBoxes()

    # Configuração dos botões
    self.setupButtonsLayout()

    # Carregar o último template
    self.loadLastTemplate()

def setupGroupBoxes(self):
    groupBoxLayout = QHBoxLayout()  # Layout horizontal para os GroupBoxes

    self.variableListGroupBox = QGroupBox("Índice de Variáveis - Variável Atual")
    variableListLayout = QVBoxLayout()
    self.variableList = QListWidget()

    # Lista de variáveis a serem excluídas
    excluded_keys = {
        "assinatura_contrato", "atualizacao_comprasnet", "comentarios",
        "comprasnet_contratos", "cp", "msg", "registro_status"
    }

    # Adicionar variáveis com ou sem valores correspondentes
    for key, value in self.df_registro_selecionado.iloc[0].items():
        if key not in excluded_keys:
            item_text = f"{{{{{key}}}}} - {value}"  # Nome da variável com valor

            self.variableList.addItem(item_text)

    self.variableList.setMaximumWidth(400)  # Limita o tamanho do QListWidget

    variableListLayout.addWidget(self.variableList)
    self.variableListGroupBox.setLayout(variableListLayout)
    self.variableListGroupBox.setMaximumWidth(400)  # Limita o tamanho do QGroupBox

    groupBoxLayout.addWidget(self.variableListGroupBox)

    # Conectar o evento itemDoubleClicked ao método insertVariable
    self.variableList.itemDoubleClicked.connect(self.insertVariable)

    # Grupo para o editor de modelo
    self.modelEditorGroupBox = QGroupBox("Campo para Edição do Modelo")
    modelEditorLayout = QVBoxLayout()
    self.modelEditor = QTextEdit()
    modelEditorLayout.addWidget(self.modelEditor)
```

```
        # Botão "Aplicar Modelo" no canto inferior do editor
              applyButton = create_button(text="Aplicar Modelo", icon=self.image_cache['apply'],
callback=self.applyTemplate,
                                                  tooltip_text="Aplica o modelo atual", parent=self,
icon_size=QSize(40, 40))
        modelEditorLayout.addWidget(applyButton)

        self.modelEditorGroupBox.setLayout(modelEditorLayout)
        groupBoxLayout.addWidget(self.modelEditorGroupBox)

        # Grupo para o visualizador de texto
        self.textViewerGroupBox = QGroupBox("Campo para Visualização da Mensagem")
        textViewerLayout = QVBoxLayout()
        self.textViewer = QTextEdit()
        self.textViewer.setReadOnly(True)
        textViewerLayout.addWidget(self.textViewer)

        # Botão "Copiar Mensagem" no canto inferior do visualizador
              copyButton = create_button(text="Copiar Mensagem", icon=self.image_cache['copy'],
callback=self.copyTextToClipboard,
                                       tooltip_text="Copia a mensagem para a área de transferência",
parent=self, icon_size=QSize(40, 40))
        textViewerLayout.addWidget(copyButton)

        self.textViewerGroupBox.setLayout(textViewerLayout)
        groupBoxLayout.addWidget(self.textViewerGroupBox)

        self.mainLayout.addLayout(groupBoxLayout)

    def setupButtonsLayout(self):
        self.buttons_layout = QHBoxLayout()
        self.createButtons()
         self.mainLayout.addLayout(self.buttons_layout)  # Adiciona o layout dos botões ao layout
principal

    def createButtons(self):
        icon_size = QSize(40, 40)  # Tamanho do ícone para todos os botões
        self.button_specs = [
                ("Alerta Prazo", self.image_cache['mensagem'], self.alertaPrazo, "Envia alerta de
prazo", icon_size),
               ("Ata Assinada", self.image_cache['mensagem'], self.ataAssinada, "Marca a ata como
assinada", icon_size),
              ("Contrato Assinado", self.image_cache['mensagem'], self.contratoAssinado, "Marca o
contrato como assinado", icon_size),
             ("Autorização", self.image_cache['mensagem'], self.aprovacaoIntanciaGovernanca, "Marca
o contrato como assinado", icon_size),
              ("Outra", self.image_cache['mensagem'], self.msgNaoDefinida, "Marca o contrato como
assinado", icon_size),
        ]

        for text, icon, callback, tooltip, icon_size in self.button_specs:
               btn = create_button(text=text, icon=icon, callback=callback, tooltip_text=tooltip,
parent=self, icon_size=icon_size)
```

```
            self.buttons_layout.addWidget(btn)

    def loadTemplate(self, template_name):
        """Carrega o template específico e aplica ao editor."""
        self.templatePath = Path(MSG_CONTRATOS_DIR) / template_name
        self.loadLastTemplate()

           def   showAutoCloseMessageBox(self,   title,   text,   icon=QMessageBox.Icon.Information,
timeout=300):
        msg_box = QMessageBox(self)
        msg_box.setIcon(icon)
        msg_box.setWindowTitle(title)
        msg_box.setText(text)
        msg_box.show()
        QTimer.singleShot(timeout, msg_box.close)

    def alertaPrazo(self):
        self.cabecalhoAtual = self.gerarCabecalho("alerta_prazo")
        self.loadTemplate("msg_alerta_prazo.txt")
        self.showAutoCloseMessageBox("Alerta Prazo", "Template de alerta de prazo carregado.")

    def ataAssinada(self):
        self.cabecalhoAtual = self.gerarCabecalho("ata_assinada")
        self.loadTemplate("msg_ata_assinada.txt")
        self.showAutoCloseMessageBox("Ata Assinada", "Template de ata assinada carregado.")

    def contratoAssinado(self):
        self.cabecalhoAtual = self.gerarCabecalho("contrato_assinado")
        self.loadTemplate("msg_contrato_assinado.txt")
              self.showAutoCloseMessageBox("Contrato  Assinado",  "Template  de  contrato  assinado
carregado.")

    def aprovacaoIntanciaGovernanca(self):
        self.cabecalhoAtual = self.gerarCabecalho("autorizacao")
        self.loadTemplate("msg_autorizacao.txt")
             self.showAutoCloseMessageBox("Instância  de  Governança",  "Template  de  Instância  de
Governança carregado.")

    def msgNaoDefinida(self):
        self.cabecalhoAtual = self.gerarCabecalho("outra")
        self.loadTemplate("msg_nao_definida.txt")
        self.showAutoCloseMessageBox("Contrato Assinado", "Template de outra msg carregado.")

    def gerarCabecalho(self, tipo):
        mes_atual = datetime.now().strftime("%b").upper()
        ano_atual = datetime.now().strftime('%Y')

        tipos = {
            "alerta_prazo": "Renovação de Acordos Administrativos",
            "ata_assinada": "Ata de Assinatura",
            "contrato_assinado": "Assinatura de Contratos Administrativos",
            "autorizacao": "Autorização de Contratos Administrativos",
            "outra": ""
```

```
        }

        descricao = tipos.get(tipo, "Descrição não encontrada")

        return (
            "<p>ROTINA<br>"
            f"R-MINUTAZ/{mes_atual}/{ano_atual}<br>"
            "DE CITBRA<br>PARA SETDIS<br>GRNC<br>BT<br><br>"
            f"{descricao}<br><br>"
        )

    def applyTemplate(self):
        # Renderiza o template inicial sem a substituição de 'prazo_final'
        user_template = self.modelEditor.toPlainText()
                    texto_inicial   =   self.cabecalhoAtual   +   self.renderTemplate(user_template,
self.df_registro_selecionado.iloc[0])

        # Exibe o template inicial no editor
        self.textViewer.setHtml(texto_inicial)

        # Verifica se o índice está dentro dos limites antes de calcular o prazo
                        if   not   self.df_registro_selecionado.empty   and   self.indice_linha   <
len(self.df_registro_selecionado):
            # Depois da edição pelo usuário, calcula o prazo_envio
                                                                                vigencia_final      =
self.df_registro_selecionado.iloc[self.indice_linha]['vigencia_final']
            print(f"Vigência final obtida: {vigencia_final}")  # Print para depuração

            prazo_envio = self.calcular_prazo_envio(vigencia_final)
            print(f"Prazo de envio calculado: {prazo_envio}")  # Print para depuração

            # Renderiza novamente o template substituindo 'prazo_final'
            texto_completo = self.renderPrazoFinal(texto_inicial, prazo_envio)
            self.textViewer.setHtml(texto_completo)
        else:
            print("DataFrame está vazio ou índice fora dos limites")
            self.close()
            return

    def renderTemplate(self, template, data):
             # Converte o template em texto formatado substituindo as variáveis pelos valores
correspondentes
        template = template.replace('\n', '<br>')

        rendered_text = template
        for key, value in data.items():
            if key == "material_servico":
                if value.strip() == "Serviço":
                                substitution = "<span style='font-weight: bold; text-decoration:
underline;'>contratação de empresa especializada em</span>"
                elif value.strip() == "Material":
                    substitution = "<span style='font-weight: bold; text-decoration: underline;'>a
aquisição de</span>"
```

```
                else:
                                substitution = f"<span style='font-weight: bold; text-decoration:
underline;'>{value}</span>"
            else:
                               substitution = f"<span style='font-weight: bold; text-decoration:
underline;'>{value}</span>"

            rendered_text = re.sub(rf"{{{{\s*{key}\s*}}}}", substitution, rendered_text)

        # Deixa a substituição de {{prazo_final}} para depois da edição pelo usuário
        return rendered_text

    def renderPrazoFinal(self, rendered_text, prazo_envio):
        # Substitui '{{prazo_final}}' usando o valor de 'prazo_envio' fornecido como argumento
        if "{{prazo_final}}" in rendered_text:
                            substitution = f"<span style='font-weight: bold; text-decoration:
underline;'>{prazo_envio}</span>"
            rendered_text = rendered_text.replace("{{prazo_final}}", substitution)

        return rendered_text

    def loadTemplate(self, template_name):
        """Carrega o template específico e aplica ao editor. Cria o arquivo se não existir."""
        # Salva o template atual antes de mudar
        self.saveCurrentTemplate()

        self.templatePath = Path(MSG_CONTRATOS_DIR) / template_name

        # Verifica se o arquivo existe; se não, cria um novo com conteúdo padrão
        if not self.templatePath.exists():
            with open(self.templatePath, 'w', encoding='utf-8') as file:
                file.write("Digite o texto da mensagem aqui...")  # Conteúdo padrão

        self.loadLastTemplate()

    def loadLastTemplate(self):
        try:
            with open(self.templatePath, 'r', encoding='utf-8') as file:
                last_template = file.read()
            self.modelEditor.setPlainText(last_template)
            self.applyTemplate()  # Aplica o modelo após carregar
        except Exception as e:
            QMessageBox.warning(self, "Erro ao carregar template", str(e))

    def saveCurrentTemplate(self):
        """Salva o template atual no arquivo atual."""
        try:
            current_template = self.modelEditor.toPlainText()
            with open(self.templatePath, 'w', encoding='utf-8') as file:
                file.write(current_template)
        except Exception as e:
            QMessageBox.warning(self, "Erro ao salvar template", str(e))
```

```
    def closeEvent(self, event):
        """Garante que o template atual seja salvo ao fechar a janela."""
        self.saveCurrentTemplate()
        super().closeEvent(event)


    def copyTextToClipboard(self):
        text = self.textViewer.toPlainText()
        QApplication.clipboard().setText(text)
        self.showAutoCloseMessageBox("Copiado", "Texto copiado para a área de transferência.")


    def insertVariable(self, item):
        cursor = self.modelEditor.textCursor()  # Obtém o cursor atual do QTextEdit

        if not cursor or cursor.position() == 0:
              cursor.movePosition(QTextCursor.MoveOperation.End)  # Move o cursor para o final do
texto

        # Extrai apenas o nome da variável, sem o valor
        variable_name = item.text().split(' - ')[0]
        cursor.insertText(variable_name)  # Insere o nome da variável na posição atual do cursor

        self.modelEditor.setTextCursor(cursor)  # Define o cursor de volta no QTextEdit
        self.modelEditor.setFocus()  # Foca no editor após inserção

    def calcular_prazo_envio(self, vigencia_final):
        try:
            # Converte a string de data para um tipo datetime
            data_final = pd.to_datetime(vigencia_final, format='%d/%m/%Y')
            print(f"Data final convertida: {data_final}")  # Print para depuração

            # Subtrai 45 dias úteis, considerando apenas os dias úteis
            data_prazo = data_final - BDay(45)

            # Formata a data calculada para o formato DD/MM/AAAA
            prazo_envio = data_prazo.strftime('%d/%m/%Y')
            print(f"Prazo de envio formatado: {prazo_envio}")  # Print para depuração

            return prazo_envio
        except Exception as e:
            print(f"Erro ao calcular prazo de envio: {e}")
            return "Data não definida"
```