Arquivo: gerar_tabela.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import pandas as pd
from datetime import datetime
import os
import subprocess
import pandas as pd

class TabelaResumidaManager:
    def __init__(self, model):
        """
        Inicializa o gerenciador da tabela com o modelo PyQt.

        :param model: O modelo PyQt da tabela (QAbstractTableModel)
        """
        self.model = model
        self.data = []
        self.columns = []
        self.df_resumido = pd.DataFrame()

    def carregar_dados(self):
        """Carrega os dados do modelo e armazena em um DataFrame."""
        try:
            print("Iniciando o carregamento de dados...")
            print(f"Tipo de self.model: {type(self.model)}")

            # Exibe uma lista dos atributos e métodos disponíveis no modelo
            print("Atributos e métodos de self.model:")
            print(dir(self.model))

            # Verifique se `self.model` possui `headerData` e `columnCount`
            if hasattr(self.model, 'headerData') and hasattr(self.model, 'columnCount'):
                # Obter os cabeçalhos das colunas
                     self.columns = [self.model.headerData(i, Qt.Orientation.Horizontal) for i in
range(self.model.columnCount())]
                print(f"Colunas obtidas: {self.columns}")

                # Obter os dados do modelo linha por linha
                self.data = []  # Reinicia self.data para evitar acúmulo
                for row in range(self.model.rowCount()):
                    row_data = []
                    for column in range(self.model.columnCount()):
                        index = self.model.index(row, column)
                        value = self.model.data(index)
                        row_data.append(value if value is not None else "")
                    self.data.append(row_data)

                # Converte `self.data` para um DataFrame com as colunas capturadas
                self.df = pd.DataFrame(self.data, columns=self.columns)
```

```
                print("DataFrame criado com sucesso a partir de self.model.")

                # Mapeamento de nomes das colunas para os esperados
                df = self.df.rename(columns={
                    'situacao': 'Status',
                    'id_processo': 'ID Processo',
                    'nup': 'NUP',
                    'objeto': 'Objeto',
                    'sigla_om': 'OM'
                })

                # Filtrar e renomear as colunas para um DataFrame resumido
                self.df_resumido = df[['Status', 'ID Processo', 'NUP', 'Objeto', 'OM']].copy()

                # Ordenar e estruturar conforme necessário
                status_ordem = {
                    'Arquivado': 1, 'Fracassado': 2, 'Deserto': 3,
                    'Homologado': 4, 'Sessão Pública': 5, 'Republicado': 6, 'Planejamento': 7
                }
                                                                self.df_resumido['Status_Value']    =
self.df_resumido['Status'].map(status_ordem).fillna(0)
                                self.df_resumido  =  self.df_resumido.sort_values(by='Status_Value',
ascending=False).drop(columns=['Status_Value'])

                # Adicionar a coluna de ordenação simples
                self.df_resumido.insert(0, 'Nº', range(1, len(self.df_resumido) + 1))
                print("Dados carregados e DataFrame resumido criado com sucesso.")

        except Exception as e:
            print(f"Erro ao carregar dados: {e}")


    def exportar_para_excel(self, output_path):
        """
        Exporta o DataFrame resumido para um arquivo Excel.

        :param output_path: Caminho para salvar o arquivo Excel gerado.
        :return: Caminho do arquivo Excel gerado.
        """
        # Exportar para Excel
        writer = pd.ExcelWriter(output_path, engine='xlsxwriter')
        self.df_resumido.to_excel(writer, sheet_name='Resumo', startrow=4, index=False)

        # Obter workbook e worksheet
        workbook = writer.book
        worksheet = writer.sheets['Resumo']

        # Configurações do formato de página e margens
        worksheet.set_landscape()  # Define o layout de página para paisagem
        worksheet.set_margins(left=0.79, right=0.39, top=0.39, bottom=0.39)  # Margens

        # Formatos para as células
        cabecalho_format = workbook.add_format({
```

```
            'align': 'center',
            'valign': 'vcenter',
            'bold': True,
            'font_size': 14
        })
        cabecalho2_format = workbook.add_format({
            'align': 'center',
            'valign': 'vcenter',
            'italic': True,
            'font_size': 12
        })
        date_format = workbook.add_format({
            'italic': True,
            'font_size': 10,
            'align': 'right'
        })

        # Formatos com cores intercaladas
             light_gray_format = workbook.add_format({'bg_color': '#F2F2F2', 'align': 'center',
'valign': 'vcenter'})
          white_format = workbook.add_format({'bg_color': '#FFFFFF', 'align': 'center', 'valign':
'vcenter'})

        # Configurações do cabeçalho e data
              worksheet.merge_range('A1:F1', 'Centro de Intendência da Marinha em Brasília',
cabecalho_format)
             worksheet.merge_range('A2:F2', '"Prontidão e Efetividade no Coração do Brasil"',
cabecalho2_format)
        worksheet.merge_range('A3:F3', 'Controle de Dispensas Eletrônica', cabecalho_format)
        data_atual = datetime.now().strftime("%d/%m/%Y")
        worksheet.merge_range('A4:F4', f"Atualizado em: {data_atual}", date_format)

        # Configurações de altura das linhas para o cabeçalho
        worksheet.set_row(0, 20)
        worksheet.set_row(2, 30)
        worksheet.set_row(3, 20)  # Ajuste de altura para a linha da data

        # Ajustar a largura das colunas
        col_widths = [10, 15, 15, 20, 40, 15]  # Largura ajustada para 6 colunas
        for i, width in enumerate(col_widths):
            worksheet.set_column(i, i, width)

        # Aplicar formatação de conteúdo centralizado a partir da linha 6
        for row_num in range(len(self.df_resumido)):
            for col_num in range(len(self.df_resumido.columns)):
                cell_format = light_gray_format if (row_num % 2 == 0) else white_format
                    worksheet.write(row_num + 5, col_num, self.df_resumido.iloc[row_num, col_num],
cell_format)



        # Salvar o arquivo
        writer.close()
```

```
        return output_path

    def exportar_df_completo_para_excel(self, output_path):
        writer = pd.ExcelWriter(output_path, engine='xlsxwriter')
        self.df.to_excel(writer, sheet_name='Tabela Completa', startrow=0, index=False)
        writer.close()

        return output_path

    def abrir_arquivo_excel(self, filepath):
        """
        Abre o arquivo Excel gerado usando o programa padrão do sistema operacional.

        :param filepath: Caminho do arquivo a ser aberto.
        """
        try:
            if os.name == 'nt':  # Para Windows
                os.startfile(filepath)
            else:
                subprocess.call(('xdg-open', filepath))  # Para MacOS ou Linux
        except Exception as e:
            print(f"Erro ao tentar abrir o arquivo: {e}")

    # def tirar_print_da_tabela(self, output_image_path):
    #     """
    #     Gera prints da tabela resumida com os valores filtrados da coluna 'Status' e salva como
várias imagens
    #     se houver mais de 25 itens.

    #     :param output_image_path: Caminho base para salvar as imagens geradas.
    #     """

    #     # Filtrar os dados apenas para os status desejados
    #     status_permitidos = ['sessão pública', 'planejamento', 'aprovado', 'homologado']
                                    #                                         df_filtrado        =
self.df_resumido[self.df_resumido['Status'].str.strip().str.lower().isin(status_permitidos)].copy(
)

    #     # Resetar a coluna de ordenação
    #     if 'Nº' in df_filtrado.columns:
    #         df_filtrado.drop(columns=['Nº'], inplace=True)
    #     df_filtrado.insert(0, 'Nº', range(1, len(df_filtrado) + 1))

    #     # Criar uma coluna temporária que mapeia os status para a ordem desejada
    #     status_ordenacao = {status: index for index, status in enumerate(status_permitidos)}
    #     df_filtrado['Ordenacao'] = df_filtrado['Status'].map(status_ordenacao)

    #     # Ordenar pelo valor da coluna de ordenação
    #     df_filtrado.sort_values(by='Ordenacao', inplace=True)

    #     # Remover a coluna de ordenação, se não for mais necessária
    #     df_filtrado.drop(columns=['Ordenacao'], inplace=True)
```

```
    #     # Definir o número máximo de itens por imagem
    #     max_itens_por_imagem = 25

    #     # Fragmentar o DataFrame em pedaços de até 25 itens
    #     num_imagens = (len(df_filtrado) // max_itens_por_imagem) + 1
        #          pedacos_df  =  [df_filtrado[i:i  +  max_itens_por_imagem]  for  i  in  range(0,
len(df_filtrado), max_itens_por_imagem)]

    #     # Definir as larguras das colunas (proporcional à escala)
    #     col_widths = [3, 8, 9, 12, 25, 8]

    #     # Gerar uma imagem para cada pedaço
    #     for idx, df_pedaco in enumerate(pedacos_df):
    #         # Criar figura e eixos para renderizar a tabela
     #           fig, ax = plt.subplots(figsize=(sum(col_widths) / 5, 4))  # Ajustar o tamanho da
figura conforme a soma das larguras

    #         # Esconder o eixo
    #         ax.xaxis.set_visible(False)
    #         ax.yaxis.set_visible(False)
    #         ax.set_frame_on(False)  # Sem bordas

    #         # Renderizar o DataFrame como uma tabela no gráfico
    #         tabela = ax.table(cellText=df_pedaco.values,
    #                           colLabels=df_pedaco.columns,
    #                           cellLoc='center',  # Centraliza os valores
    #                           loc='center')

    #         # Estilizar a tabela
    #         tabela.auto_set_font_size(False)
    #         tabela.set_fontsize(10)
    #         tabela.scale(1.5, 1.5)

    #         # Aplicar um efeito especial à linha de título
    #         for key, cell in tabela.get_celld().items():
    #             if key[0] == 0:  # Linha de cabeçalho
    #                 cell.set_fontsize(12)
    #                 cell.set_text_props(weight='bold')
    #                 cell.set_facecolor('#C5D9F1')  # Cor de fundo verde
    #                 cell.set_text_props(color='Black')  # Texto branco

    #         # Ajustar a largura das colunas conforme o array col_widths
    #         for i, width in enumerate(col_widths):
    #             for key, cell in tabela.get_celld().items():
    #                 if key[1] == i:  # Verificar se estamos na coluna i
    #                     cell.set_width(width / sum(col_widths))

    #                 # Intercalar as cores das linhas (ignora a linha de título)
    #                 if key[0] > 0:
    #                     if key[0] % 2 == 0:
    #                         cell.set_facecolor('#D0D0D0')  # Cor cinza clara
    #                     else:
    #                         cell.set_facecolor('#FFFFFF')  # Cor branca
```

```
#                 # Remover as bordas das células
#                 cell.set_edgecolor('none')

#         # Definir o nome da imagem com base no índice do pedaço
#         output_image_fragment_path = f"{output_image_path}_parte_{idx + 1}.png"

#         # Salvar a imagem
#         plt.savefig(output_image_fragment_path, bbox_inches='tight', dpi=600)

#         # Fechar a figura
#         plt.close(fig)

#         # Abrir a imagem automaticamente
#         self.abrir_imagem(output_image_fragment_path)

#     return output_image_path


def abrir_imagem(self, filepath):
    """
    Abre o arquivo de imagem gerado usando o programa padrão do sistema operacional.

    :param filepath: Caminho do arquivo a ser aberto.
    """
    try:
        if os.name == 'nt':  # Para Windows
            os.startfile(filepath)
        else:
            subprocess.call(('xdg-open', filepath))  # Para MacOS ou Linux
    except Exception as e:
        print(f"Erro ao tentar abrir o arquivo: {e}")
```