Arquivo: controller.py

```
from modules.dispensa.dialogs.add_item import AddItemDialog
from modules.dispensa.dialogs.salvar_tabela import DataManager
from modules.dispensa.dialogs.gerar_tabela import TabelaResumidaManager
from modules.dispensa.dialogs.edit_data.edit_data import EditarDadosWindow
from modules.dispensa.database_manager.db_manager import DatabaseManager
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
import pandas as pd
from paths import CONTROLE_DADOS, TEMPLATE_DISPENSA_DIR
import sqlite3
from datetime import datetime
import os
from modules.dispensa.dados_api.api_consulta import ConsultaAPIDialog
import webbrowser
from urllib.parse import quote
from modules.utils.automacao_email import executar_automacao_email
from modules.utils.automacao_coordenadas import executar_automacao_email_coords

class DispensaEletronicaController(QObject):
    def __init__(self, icons, view, model):
        super().__init__()
        self.icons = icons
        self.view = view
        self.edit_data_dialog = None
        self.model_add = model
        self.model = model.setup_model("controle_dispensas")
        self.controle_om = CONTROLE_DADOS  # Atribui o caminho diretamente ao controle_om
        self.setup_year_filter() #filtro por ano
        self.setup_connections()

    def setup_connections(self):
        # Conecta os sinais da view aos métodos do controlador
        self.view.addItem.connect(self.handle_add_item)
        self.view.deleteItem.connect(self.handle_delete_item)
        self.view.dataManager.connect(self.handle_data_manager)
        self.view.salvar_print.connect(self.handle_save_print)
        self.view.rowDoubleClicked.connect(self.handle_edit_item)
        self.view.request_consulta_api.connect(self.consultar_api)
         self.view.filtro_ano_combo.currentTextChanged.connect(self.apply_year_filter) # Conecta o
sinal ao filtro de ano

    # Adicione estes dois novos métodos completos à sua classe de controller

    def setup_year_filter(self):
        """
        Busca os anos distintos no banco de dados e os adiciona à QComboBox na view.
        """
        # Acessa o db_manager através do model para buscar os anos
        anos = self.model.database_manager.get_distinct_years()
```

```
        # Limpa a caixa de seleção para evitar duplicatas ao recarregar
        self.view.filtro_ano_combo.clear()

        # Adiciona a opção "Todos" no topo da lista
        self.view.filtro_ano_combo.addItem("Todos")

        # Adiciona os anos buscados do banco
        if anos:
            self.view.filtro_ano_combo.addItems(anos)

    def apply_year_filter(self, year):
        """
        Aplica um filtro ao QSqlTableModel com base no ano selecionado.
        """
        # Adiciona um print para sabermos que a função foi chamada
        print(f"Filtro de ano acionado. Ano selecionado: '{year}'")

        if year == "Todos" or not year:
            # Se "Todos" for selecionado, o filtro é uma string vazia
            filter_str = ""
        else:
            # Cria a string de filtro para a cláusula WHERE do SQL.
            # Ex: id_processo LIKE '%/2024'
            filter_str = f"id_processo LIKE '%/{year}'"

        # Define o filtro no modelo. Até aqui, nada muda na tela.
        self.model.setFilter(filter_str)
        self.model.select()
        print(f"Filtro aplicado. A tabela agora exibe {self.model.rowCount()} linhas.")
        self.view.proxy_model.setYearFilter(year)

    def consultar_api(self, cnpj, ano, sequencial, uasg, numero):
        # Inicia o diálogo de consulta API com o parent `self.view`
        dialog = ConsultaAPIDialog(numero, cnpj, sequencial, ano, uasg, parent=self.view)

        # Conecta o sinal `consulta_concluida` para processar os dados após a consulta
        dialog.consulta_concluida.connect(self.handle_api_data)
        dialog.exec()

    def handle_api_data(self, data_informacoes_lista, resultados_completos):
        # Estrutura os dados da API para salvar no banco de dados
        data_api_to_save = {
            "data_informacoes": data_informacoes_lista,
            "resultados_completos": resultados_completos
        }

        # Solicita ao modelo que salve os dados no banco de dados
        self.model_add.save_api_data(data_api_to_save)

    def handle_add_item(self):
        """Trata a ação de adicionar item."""
         dialog = AddItemDialog(self.icons, self.model.database_manager.db_path, self.controle_om,
self.view)  # Passa o caminho do banco de dados
```

```
        if dialog.exec():
            item_data = dialog.get_data()
            # Adiciona a situação padrão 'Planejamento' antes de salvar
            item_data['situacao'] = 'Planejamento'
            self.model_add.insert_or_update_data(item_data)  # Salva no banco de dados
            self.view.refresh_model()   # Salva no banco de dados
            self.setup_year_filter()

    def handle_delete_item(self):
        """Trata a ação de exclusão de um item selecionado."""
        selection_model = self.view.table_view.selectionModel()
        print("Iniciando a exclusão do item...")

        if selection_model.hasSelection():
            index = selection_model.selectedRows(1)[0]  # Assumindo que a coluna 0 é 'id_processo'
            id_processo = index.data()
            print(f"ID do processo selecionado para exclusão: {id_processo}")

            if id_processo:
                confirmation = QMessageBox.question(
                    self.view, "Confirmação",
                    f"Tem certeza que deseja excluir o registro com ID '{id_processo}'?",
                    QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
                    QMessageBox.StandardButton.No
                )
                print("Confirmação de exclusão solicitada.")

                if confirmation == QMessageBox.StandardButton.Yes:
                    print(f"Confirmado. Excluindo o item com ID: {id_processo}")
                    # Usando o método delete_data corretamente
                    try:
                        self.model.database_manager.delete_data(id_processo)
                        print("Item excluído com sucesso.")
                        self.view.refresh_model()
                        print("Modelo atualizado após exclusão.")
                    except Exception as e:
                        print(f"Erro ao excluir o item: {e}")
                else:
                    print("Exclusão cancelada pelo usuário.")
        else:
            print("Nenhum item selecionado para exclusão.")
              QMessageBox.warning(self.view, "Nenhuma Seleção", "Por favor, selecione um item para
excluir.")

    def handle_save_print(self):
        """Trata a ação de salvar uma imagem da tabela."""
        # Implementação de salvar print da tabela
        output_image_path = "tabela_resumida.png"
        # Supondo que um método `tirar_print_da_tabela` exista para salvar a tabela como imagem
        self.view.salvar_tabela_resumida()
                 QMessageBox.information(self.view,  "Imagem  Salva",  f"A  tabela  foi  salva  em
{output_image_path}")
```

```
    def refresh_view(self):
        # Atualiza a visualização da tabela após alterações nos dados
        self.view.model.select()  # Recarrega os dados no modelo

    def handle_edit_item(self, data):
        # Extrai informações para compor o nome da tabela
        cnpj_matriz = data.get("cnpj_matriz") or ""
        sequencial_pncp = data.get("sequencial_pncp") or ""
        ano = str(data.get("ano", "0000"))[-4:]  # Captura apenas os 4 últimos dígitos do ano

        if cnpj_matriz and sequencial_pncp:
            table_name = f"{cnpj_matriz}_1_{sequencial_pncp}_{ano}"
        else:
            table_name = "" # Define como vazio se os dados não existirem

        print(f"DEBUG: Nome da tabela a ser verificada: {table_name}")

        # Conecta ao banco de dados para verificar se a tabela existe e realizar as consultas
        try:
            with self.model_add.database_manager as conn:
                cursor = conn.cursor()

                # Verifica se a tabela existe
                         cursor.execute(f'SELECT  name  FROM  sqlite_master  WHERE  type="table"  AND
name="{table_name}"')
                table_exists = cursor.fetchone() is not None

                if table_exists:
                    # print(f"Sucesso: A tabela '{table_name}' existe no banco de dados.")

                    # Calcula `total_homologado`
                       cursor.execute(f'SELECT SUM(valorTotalHomologado) FROM "{table_name}" WHERE
valorTotalHomologado IS NOT NULL')
                    total_homologado = cursor.fetchone()[0]
                    total_homologado = total_homologado if total_homologado is not None else None
                    print(f"Somatório de valorTotalHomologado: {total_homologado}")

                    # Conta `situacaoCompraItemNome` com valores específicos
                    cursor.execute(f'''
                        SELECT COUNT(*)
                        FROM "{table_name}"
                                    WHERE  situacaoCompraItemNome  IN  ("Anulado/Revogado/Cancelado",
"Fracassado")
                    ''')
                    count_anulado_fracassado = cursor.fetchone()[0]
                                        count_anulado_fracassado  =  count_anulado_fracassado  if
count_anulado_fracassado > 0 else None
                         print(f"Quantidade de itens 'Anulado/Revogado/Cancelado' ou 'Fracassado':
{count_anulado_fracassado}")

                    # Conta `situacaoCompraItemResultadoNome` com valor "Informado"
                    cursor.execute(f'''
                        SELECT COUNT(*)
```

```
                        FROM "{table_name}"
                        WHERE situacaoCompraItemResultadoNome = "Informado"
                    ''')
                    count_informado = cursor.fetchone()[0]
                    count_informado = count_informado if count_informado > 0 else None
                      print(f"Quantidade de itens com situacaoCompraItemResultadoNome 'Informado': {count_informado}")

                else:
                    print(f"Erro: A tabela '{table_name}' não foi encontrada no banco de dados.")
                     total_homologado, count_anulado_fracassado, count_informado = "Não Informado", "Não Informado", "Não Informado"

        except sqlite3.Error as e:
            print(f"Erro ao consultar a tabela no banco de dados: {e}")
               total_homologado, count_anulado_fracassado, count_informado = "Não Informado", "Não Informado", "Não Informado"

        # Passa os valores para a instância de EditarDadosWindow
        self.edit_data_dialog = EditarDadosWindow(
            data, self.icons, table_name, self.view
        )
        self.edit_data_dialog.save_data_signal.connect(self.handle_save_data)
        self.edit_data_dialog.disparar_email_signal.connect(self.handle_disparar_email)
        self.view.connect_editar_dados_window(self.edit_data_dialog)  # Conecta sinais
        self.edit_data_dialog.show()

    def handle_disparar_email(self, data):
        """
        Este método é o SLOT que recebe os dados da view.
        Ele verifica a situação e chama a função de e-mail apropriada.
        """
        situacao = data.get('situacao')
        print(f"Controller recebeu sinal para disparar e-mail. Situação: {situacao}")

        if situacao == 'Homologado':
            self._preparar_email_homologado(data)
        elif situacao == 'Sessão Pública':
            self._preparar_email_sessao_publica(data)
        else:
            # O aviso de "ação não permitida" é agora responsabilidade da View.
            # O controller apenas não faz nada se a situação não for uma das esperadas.
            print(f"Nenhuma ação de e-mail configurada para a situação: {situacao}")

    def handle_save_data(self, data):
        try:
            # Use `self.model_add` que se refere a uma instância de `DispensaEletronicaModel`
            self.model_add.insert_or_update_data(data)
            self.view.refresh_model()  # Atualiza a visualização da tabela
        except AttributeError as e:
                  QMessageBox.warning(self.view, "Erro", f"Ocorreu um erro ao salvar os dados: {str(e)}")
```

```
    def _preparar_email_sessao_publica(self, data):
        """
        Lê o template da mensagem de Sessão Pública e abre o cliente de e-mail.
        """
        destinatario_email = data.get("email", "")
        if not destinatario_email:
               QMessageBox.warning(self.view, "E-mail não encontrado", "O campo 'E-mail' não está
preenchido.")
            return

        try:
            caminho_template = TEMPLATE_DISPENSA_DIR / "mensagem_sessaopublica.txt"
            with open(caminho_template, 'r', encoding='utf-8') as f:
                mensagem_base = f.read()

            data_sessao_str = data.get("data_sessao", "")
            data_formatada = "[N/A]" # Valor padrão caso a data não seja válida
            if data_sessao_str:
                try:
                    # Converte a string de data para um objeto datetime
                    data_obj = datetime.strptime(data_sessao_str, '%Y-%m-%d')
                    # Formata o objeto datetime para o padrão DD/MM/YYYY
                    data_formatada = data_obj.strftime('%d/%m/%Y')
                except ValueError:
                    # Se o formato da data for inesperado, mantém o valor padrão
                    print(f"Aviso: Formato de data inválido para '{data_sessao_str}'.")

                    mensagem_final  =  mensagem_base.replace("{{numero-da-cp}}",  str(data.get("cp",
"[N/A]")))
            mensagem_final = mensagem_final.replace("{{NUP}}", str(data.get("nup", "[N/A]")))
                            mensagem_final  =  mensagem_final.replace("{{numero  da  dispensa}}",
str(data.get("id_processo", "[N/A]")))
             mensagem_final = mensagem_final.replace("{{email_responsavel}}", str(data.get("email",
"[E-mail não informado]")))
            mensagem_final = mensagem_final.replace("{{sessao_publica}}", str(data_formatada))

            assunto = f"Início da Sessão Pública - Dispensa Eletrônica: {data.get('id_processo')}"

            # 1. Abre o site do webmail em uma nova aba
            webbrowser.open_new_tab("https://webmail.marinha.mil.br/")

            # 2. Copia o corpo da mensagem para a área de transferência
            clipboard = QApplication.clipboard()
            clipboard.setText(mensagem_final)

            # 3. Exibe uma notificação para o usuário
            QMessageBox.information(self.view, "Ação Necessária",
                                    f"O webmail foi aberto.\n\n"
                                       f"A mensagem para '{destinatario_email}' foi copiada para a
sua área de transferência.\n\n"
                                         f"Por favor, crie um novo e-mail, cole o destinatário e a
mensagem.")
```

```
            # --- CÓDIGO REFINADO AQUI ---
            # A variável 'corpo' recebe diretamente 'mensagem_final' sem replaces redundantes.
            corpo = mensagem_final

            executar_automacao_email_coords(destinatario_email, assunto, corpo)

        except FileNotFoundError:
                            QMessageBox.critical(self.view,  "Erro  de  Template",  "O  arquivo
'mensagem_sessaopublica.txt' não foi encontrado.")
        except Exception as e:
                QMessageBox.critical(self.view, "Erro", f"Ocorreu um erro ao preparar a mensagem:
{e}")

    def _preparar_email_homologado(self, data):
        """
        Abre o webmail, copia a mensagem para a área de transferência
        e notifica o usuário.
        """
        destinatario_email = data.get("email", "")
        if not destinatario_email:
            QMessageBox.warning(self.view, "E-mail não encontrado",
                                "O campo 'E-mail' do responsável não está preenchido.")
            return

        try:
            caminho_template = TEMPLATE_DISPENSA_DIR / "mensagem_homologado.txt"
            with open(caminho_template, 'r', encoding='utf-8') as f:
                mensagem_base = f.read()

            # Preenche o template
                    mensagem_final  =  mensagem_base.replace("{{numero-da-cp}}",  str(data.get("cp",
"[N/A]")))
            mensagem_final = mensagem_final.replace("{{NUP}}", str(data.get("nup", "[N/A]")))
                            mensagem_final  =  mensagem_final.replace("{{numero  da  dispensa}}",
str(data.get("id_processo", "[N/A]")))
             mensagem_final = mensagem_final.replace("{{email_responsavel}}", str(data.get("email",
"[E-mail não informado]")))

            assunto = f"Homologação da Dispensa Eletrônica: {data.get('id_processo')}"

            # 1. Abre o site do webmail em uma nova aba
            webbrowser.open_new_tab("https://webmail.marinha.mil.br/")

            # 2. Copia o corpo da mensagem para a área de transferência
            clipboard = QApplication.clipboard()
            clipboard.setText(mensagem_final)

            # 3. Exibe uma notificação para o usuário
            QMessageBox.information(self.view, "Ação Necessária",
                                    f"O webmail foi aberto.\n\n"
                                       f"A mensagem para '{destinatario_email}' foi copiada para a
sua área de transferência.\n\n"
                                        f"Por favor, crie um novo e-mail, cole o destinatário e a
```

```
mensagem.")

            # --- CORREÇÃO APLICADA AQUI ---
            # A variável 'corpo' agora recebe o valor de 'mensagem_final'
            corpo = mensagem_final
            # O assunto já foi definido, não precisa repetir.

            # Chama a nova função de automação baseada em coordenadas
            executar_automacao_email_coords(destinatario_email, assunto, corpo)

        except Exception as e:
               QMessageBox.critical(self.view, "Erro", f"Ocorreu um erro ao preparar a mensagem:
{e}")

    def carregar_tabela(self):
        """
        Abre uma janela para o usuário selecionar um arquivo de tabela (Excel)
        e inicia o processo de validação e importação dos dados.
        """
        filepath, _ = QFileDialog.getOpenFileName(
            self.view,
            "Abrir arquivo de tabela",
            "",
            "Tabelas (*.xlsx *.xls *.ods)"
        )

        if filepath:
            try:
                # Carrega o arquivo selecionado em um DataFrame do pandas
                df = pd.read_excel(filepath)

                # Chama a função que você já tem para validar e processar os dados
                self.validate_and_process_data(df)

                # Insere ou atualiza cada linha do DataFrame no banco de dados
                for _, row in df.iterrows():
                    data = row.to_dict()
                    self.model_add.insert_or_update_data(data)

                # Atualiza a tabela na tela para mostrar os novos dados
                self.view.refresh_model()
                 QMessageBox.information(self.view, "Carregamento Concluído", "Dados carregados com
sucesso.")

            except Exception as e:
                 QMessageBox.warning(self.view, "Erro ao Carregar", f"Ocorreu um erro ao carregar a
tabela: {str(e)}")

    def validate_and_process_data(self, df):
        required_columns = ['ID Processo', 'NUP', 'Objeto', 'uasg']
        if not all(col in df.columns for col in required_columns):
            missing_columns = [col for col in required_columns if col not in df.columns]
            raise ValueError(f"As seguintes colunas estão ausentes: {', '.join(missing_columns)}")
```

```
            df.rename(columns={'ID  Processo':  'id_processo',  'NUP':  'nup',  'Objeto':  'objeto'},
inplace=True)
        self.desmembramento_id_processo(df)
        self.salvar_detalhes_uasg_sigla_nome(df)

    def desmembramento_id_processo(self, df):
             df[['tipo',  'numero',  'ano']]  =  df['id_processo'].str.extract(r'(\D+)(\d+)/(\d+)',
expand=True)
        df['tipo'] = df['tipo'].map({'DE ': 'Dispensa Eletrônica'}).fillna('Tipo Desconhecido')

    def salvar_detalhes_uasg_sigla_nome(self, df):
        print(f"[DEBUG] Conectando a {self.controle_om} para detalhes de UASG e Sigla")
        with sqlite3.connect(self.controle_om) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT uasg, sigla_om, orgao_responsavel FROM controle_om")
               om_details = {row[0]: {'sigla_om': row[1], 'orgao_responsavel': row[2]} for row in
cursor.fetchall()}

        df['sigla_om'] = df['uasg'].map(lambda x: om_details.get(x, {}).get('sigla_om', ''))
                        df['orgao_responsavel']   =   df['uasg'].map(lambda   x:   om_details.get(x,
{}).get('orgao_responsavel', ''))

    def salvar_tabela_completa(self):
        try:
            self.model.select()
            tabela_manager = TabelaResumidaManager(self.model)
            tabela_manager.carregar_dados()
            output_path = os.path.join(os.getcwd(), "tabela_completa.xlsx")
            tabela_manager.exportar_df_completo_para_excel(output_path)
            tabela_manager.abrir_arquivo_excel(output_path)
        except PermissionError:
            QMessageBox.warning(self.view, "Erro de Permissão",
                                "Feche o arquivo 'tabela_completa.xlsx' se ele estiver aberto e
tente novamente.")

    def salvar_tabela_resumida(self):
        try:
            self.model.select()
            tabela_manager = TabelaResumidaManager(self.model)
            tabela_manager.carregar_dados()
            output_path = os.path.join(os.getcwd(), "tabela_resumida.xlsx")
            tabela_manager.exportar_para_excel(output_path)
            tabela_manager.abrir_arquivo_excel(output_path)
        except PermissionError:
            QMessageBox.warning(self.view, "Erro de Permissão",
                                "Feche o arquivo 'tabela_resumida.xlsx' se ele estiver aberto e
tente novamente.")

    def excluir_database(self):
        reply = QMessageBox.question(self.view, "Confirmação de Exclusão",
                                     "Tem certeza de que deseja excluir todos os dados?",
                                                          QMessageBox.StandardButton.Yes  |
```

```
QMessageBox.StandardButton.No,
                                    QMessageBox.StandardButton.No)
        if reply == QMessageBox.StandardButton.Yes:
            with self.model.database_manager as conn:
                cursor = conn.cursor()
                cursor.execute("DROP TABLE IF EXISTS controle_dispensas")
                conn.commit()
            QMessageBox.information(self.view, "Sucesso", "Tabela excluída com sucesso.")
            self.view.refresh_model()
            # self.model.select()  # Atualiza o modelo para refletir a exclusão

    def handle_data_manager(self):
        """Trata a ação de salvar a tabela e gerenciar as ações de dados."""
        dialog = DataManager(self.icons, self.model, self, parent=self.view)
        dialog.exec()
        # self.model.select()

def show_warning_if_view_exists(view, title, message):
    if view is not None:
        QMessageBox.warning(view, title, message)
    else:
        print(message)  # Mensagem para o log, caso `view` esteja indisponível
```