

## Arquivo: controller.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
import pandas as pd
from paths import CONTROLE_DADOS
from modules.contratos.api.comprasnet_api import ConsultaAPIWindow
import sqlite3

class ContratosController(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_contratos")
        self.controle_om = CONTROLE_DADOS # Atribui o caminho diretamente ao controle_om

        self.setup_connections()

    def setup_connections(self):
        # Conecta os sinais da view aos métodos do controlador
        self.view.apiCheck.connect(self.abrir_consulta_api)

    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_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` existe para salvar a tabela como imagem
    self.view.salvar_tabela_resumida()
    QMessageBox.information(self.view, "Imagen 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_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 carregar_tabela(self):
    filepath, _ = QFileDialog.getOpenFileName(self.view, "Abrir arquivo de tabela", "",
    "Tabelas (*.xlsx *.xls *.ods)")
    if filepath:
        try:
            # Carrega o arquivo selecionado em um DataFrame
            df = pd.read_excel(filepath)
            self.validate_and_process_data(df)

            # Insere ou atualiza os dados no banco de dados
            for _, row in df.iterrows():
                data = row.to_dict()

```

```

        self.model_add.insert_or_update_data(data)
        self.view.refresh_model()
    # Atualiza o modelo para refletir as alterações
    self.model.select()
    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 excluir_database(self):
    reply = QMessageBox.question(self.view, "Confirmação de Exclusão",
                                 "Tem certeza de que deseja excluir todos os dados?", QMessageBox.StandardButton.Yes |
QMMessageBox.StandardButton.No,
QMMessageBox.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 abrir_consulta_api(self):
    dialog = ConsultaAPIWindow(self.icons, self.model_add, self.view) # Agora passando model
    view
    dialog.exec()

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
```