Arquivo: controller.py

```
from PyQt6.QtCore import *
from modules.atas.widgets.worker_homologacao import Worker
from PyQt6.QtWidgets import QFileDialog

class GerarAtasController(QObject):
    def __init__(self, icons, view, model):
        super().__init__()
        self.icons = icons
        self.view = view
        self.model = model.setup_model("controle_atas")
        self.worker = None
        self.setup_connections()

    def setup_connections(self):
        # Conecta o sinal de instruções
        self.view.instructionSignal.connect(self.instrucoes)
        self.view.trSignal.connect(self.termo_referencia)
        self.view.homologSignal.connect(self.termo_homologacao)
        self.view.sicafSignal.connect(self.sicaf_widget)
        self.view.atasSignal.connect(self.gerar_atas)

        # Configura os botões usando GerarAtasModel
        self.view.tr_widget.abrirTabelaNova.connect(self.abrir_tabela_nova)
        self.view.tr_widget.configurarSqlModelSignal.connect(self.configurar_sql_model)

        # Conecta o sinal de carregar tabela ao método do modelo
        self.view.tr_widget.carregarTabela.connect(self.caregar_tabela_com_dados)

        # Conecta o sinal `tabelaCarregada` para configurar o modelo SQL após carregar a tabela
        self.model.tabelaCarregada.connect(self.configurar_sql_model)

        if hasattr(self.view.homolog_widget, 'gerarPlanilhaBaseClicked'):

self.view.homolog_widget.gerarPlanilhaBaseClicked.connect(self.iniciar_geracao_planilha_base)

    def iniciar_geracao_planilha_base(self):
        """
        Esta função é chamada quando o usuário clica no botão "Gerar Planilha Base".
        Ela configura e inicia o Worker no modo 'tabela_base'.
        """
        # Pega o diretório dos PDFs a partir do widget de homologação
        pdf_dir = self.view.homolog_widget.pdf_dir
        if not pdf_dir or not pdf_dir.exists():
            # Você pode mostrar um pop-up de erro aqui
            print("Erro: O diretório de PDFs não foi selecionado.")
            return

        # 1. Cria a instância do Worker no modo correto
        self.worker = Worker(pdf_dir=pdf_dir, modo='tabela_base')
```

```
        # 2. Conecta os sinais do worker aos slots (funções) da sua interface
        # Conecta o sinal de progresso à sua barra de progresso
        self.worker.progress_signal.connect(self.view.homolog_widget.update_progress)
        # Conecta as mensagens de log à sua área de texto de contexto
        self.worker.update_context_signal.connect(self.view.homolog_widget.update_context)
        # Conecta o sinal de conclusão da thread
        self.worker.finished.connect(self.finalizar_worker)
        # ** A conexão mais importante: Quando a tabela estiver pronta, chama a função para salvar
**
        self.worker.tabela_base_pronta.connect(self.salvar_planilha_base)

        # 3. Inicia a thread
        self.worker.start()
        # Opcional: Desabilitar o botão enquanto o worker estiver rodando
        self.view.homolog_widget.set_buttons_enabled(False)

    def salvar_planilha_base(self, df):
        """
        Este slot é ativado pelo sinal 'tabela_base_pronta' do worker.
        Ele recebe o DataFrame e abre uma janela para salvar o arquivo.
        """
        if df.empty:
            # Você pode mostrar uma mensagem de aviso aqui
            print("AVISO: O DataFrame está vazio. Nenhum arquivo será salvo.")
            return

        # Abre a caixa de diálogo "Salvar como..."
        # O padrão é o nome do arquivo que sugerimos
        caminho_inicial = "planilha_base_licitacao.xlsx"
        caminho_arquivo, _ = QFileDialog.getSaveFileName(
            self.view,
            "Salvar Planilha Base",
            caminho_inicial,
            "Arquivos Excel (*.xlsx);;Todos os Arquivos (*)"
        )

        # Se o usuário selecionou um caminho e clicou em "Salvar"
        if caminho_arquivo:
            try:
                # Usa o pandas para salvar o DataFrame em um arquivo .xlsx
                df.to_excel(caminho_arquivo, index=False)
                            self.view.homolog_widget.update_context(f"Sucesso!  Planilha  salva  em:
{caminho_arquivo}")
                # Você pode adicionar um pop-up de sucesso aqui se desejar
            except Exception as e:
                self.view.homolog_widget.update_context(f"ERRO ao salvar a planilha: {e}")
                # Você pode adicionar um pop-up de erro aqui

    def finalizar_worker(self):
        """
        Chamado quando a thread do worker termina.
        """
        self.view.homolog_widget.update_context("Processo finalizado.")
```

```
    # Reabilita os botões na interface
    self.view.homolog_widget.set_buttons_enabled(True)
    self.worker = None # Limpa a referência


def configurar_sql_model(self):
    # Aplica `self.model` ao `table_view` diretamente
    self.view.tr_widget.table_view.setModel(self.model)
    self.view.configurar_visualizacao_tabela_tr(self.view.tr_widget.table_view)

def abrir_tabela_nova(self):
    # Lógica para abrir uma nova tabela
    self.model.abrir_tabela_nova()

def caregar_tabela_com_dados(self):
    # Lógica para abrir uma nova tabela
    self.model.carregar_tabela()

def instrucoes(self):
    # Atualiza para o widget de instruções
    self.view.content_area.setCurrentWidget(self.view.instrucoes_widget)

def termo_referencia(self):
    # Define o widget atual para Termo de Referência
    self.view.content_area.setCurrentWidget(self.view.tr_widget)
    # Configura o QTableView com o modelo e ajusta a visualização
    self.view.tr_widget.table_view.setModel(self.model)
    self.view.configurar_visualizacao_tabela_tr(self.view.tr_widget.table_view)

def termo_homologacao(self):
    # Exibe o widget de Processamento para Termo de Homologação
    self.view.content_area.setCurrentWidget(self.view.homolog_widget)

def sicaf_widget(self):
    # Exibe o widget de Processamento para Termo de Homologação
    self.view.content_area.setCurrentWidget(self.view.sicaf_widget)

def gerar_atas(self):
    # Atualiza para o widget de geração de atas
    self.view.content_area.setCurrentWidget(self.view.atas_widget)
```