Arquivo: consultar_api.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import json
import requests
import time
from pathlib import Path
from modules.atas_api.widgets.progresso_homolog import TreeViewWindow
from paths import CONFIG_API_FILE
from modules.utils.add_button import add_button_func_vermelho
from modules.utils.linha_layout import linha_divisoria_layout

class PNCPConsultaThread(QThread):
    consulta_concluida = pyqtSignal(list, list)
    erro_consulta = pyqtSignal(str)
    progresso_consulta = pyqtSignal(str)

    def __init__(self, cnpj, ano, sequencial, db_manager, parent=None):
        super().__init__(parent)
        self.cnpj = cnpj
        self.ano = ano
        self.sequencial = sequencial
        self.db_manager = db_manager

    def run(self):
        try:
                                data_informacoes,   data_informacoes_lista,   resultados_completos   =
self.consultar_por_sequencial()
            self.db_manager.salvar_consulta_api_no_db(data_informacoes)
            self.db_manager.criar_tabela_itens_pregao(
                data_informacoes.get("numeroCompra"),
                data_informacoes.get("anoCompra"),
                data_informacoes["unidadeOrgao"].get("codigoUnidade")
            )

            # Carregar dados da tabela controle_atas_api
            controle_atas_api = self.db_manager.execute_query(
                "SELECT item, catalogo FROM controle_atas_api WHERE catalogo IS NOT NULL"
            )
            controle_atas_dict = {row[0]: row[1] for row in controle_atas_api}

            # Popular o banco com os resultados e o controle_atas_dict
            self.homologacao_dataframe = self.db_manager.popular_db_consulta_itens_api(
                resultados_completos,
                data_informacoes,
                data_informacoes.get("numeroCompra"),
                data_informacoes.get("anoCompra"),
                data_informacoes["unidadeOrgao"].get("codigoUnidade"),
                controle_atas_dict  # Passa o dicionário de controle_atas_api
            )
```

```
            self.consulta_concluida.emit(data_informacoes_lista, resultados_completos)
        except Exception as e:
            self.erro_consulta.emit(str(e))

    def consultar_por_sequencial(self):
                                                                        url_informacoes         =
f"https://pncp.gov.br/api/consulta/v1/orgaos/{self.cnpj}/compras/{self.ano}/{self.sequencial}"
        tentativas_maximas = 10

        for tentativa in range(1, tentativas_maximas + 1):
            try:
                        self.progresso_consulta.emit(f"Tentativa {tentativa}/{tentativas_maximas} -
Procurando sequencial '{self.sequencial}' no PNCP\n")
                response_informacoes = requests.get(url_informacoes, timeout=20)
                response_informacoes.raise_for_status()
                data_informacoes = response_informacoes.json()

                # Converte data_informacoes para uma lista e salva
                data_informacoes_lista = self.converter_para_lista(data_informacoes)
                qnt_itens = self.consultar_quantidade_de_itens()
                            resultados_completos  =  self.consultar_detalhes_dos_itens(qnt_itens,
self.progresso_consulta)

                return data_informacoes, data_informacoes_lista, resultados_completos
            except requests.exceptions.RequestException as e:
                 self.progresso_consulta.emit(f"Erro na tentativa {tentativa}/{tentativas_maximas}:
{str(e)}\n")
                if tentativa < tentativas_maximas:
                    time.sleep(2)
                else:
                    raise Exception(f"Falha após {tentativas_maximas} tentativas: {str(e)}")
        return {}, [], []

    def salvar_json(self, resultados_completos):
        # Salva `resultados_completos` em JSON
        file_path = f"resultados_{self.ano}_{self.sequencial}.json"
        with open(file_path, "w", encoding="utf-8") as f:
            json.dump(resultados_completos, f, ensure_ascii=False, indent=4)
        self.progresso_consulta.emit(f"Dados detalhados salvos em {file_path}\n")

    def consultar_detalhes_dos_itens(self, qnt_itens, progresso_callback):
        resultados_completos = []

        for i in range(1, qnt_itens + 1):
                                                                    url_item_info      =
f"https://pncp.gov.br/api/pncp/v1/orgaos/{self.cnpj}/compras/{self.ano}/{self.sequencial}/itens/{i
}"
            response_item_info = requests.get(url_item_info)
            response_item_info.raise_for_status()
            data_item_info = response_item_info.json()

            # Usar o callback de progresso para atualizar o progresso
```

```
            progresso_callback.emit(f"Verificando item {i}/{qnt_itens}")

            if data_item_info.get('temResultado', False):
                # Se tem resultado, faz a consulta adicional
                                                                        url_item_resultados     =
f"https://pncp.gov.br/api/pncp/v1/orgaos/{self.cnpj}/compras/{self.ano}/{self.sequencial}/itens/{i
}/resultados"
                response_item_resultados = requests.get(url_item_resultados)
                response_item_resultados.raise_for_status()
                data_item_resultados = response_item_resultados.json()

                if isinstance(data_item_resultados, list):
                    for resultado in data_item_resultados:
                        for key, value in resultado.items():
                            data_item_info[key] = value
            else:
                # Se não há resultado, adicionar 'None' para as chaves esperadas
                    expected_keys = ['dataResultado', 'niFornecedor', 'nomeRazaoSocialFornecedor',
'numeroControlePNCPCompra', 'tipoBeneficioNome']
                for key in expected_keys:
                    data_item_info[key] = None

            # Adiciona o item, seja com resultado ou com valores 'None'
            resultados_completos.append(data_item_info)

        return resultados_completos


    def converter_para_lista(self, dados):
        lista_resultado = []

        def _achatar(sub_dados, chave_pai=""):
            if isinstance(sub_dados, dict):
                for chave, valor in sub_dados.items():
                    nova_chave = f"{chave_pai}.{chave}" if chave_pai else chave
                    if isinstance(valor, (dict, list)):
                        _achatar(valor, nova_chave)
                    else:
                        lista_resultado.append((nova_chave, valor))
            elif isinstance(sub_dados, list):
                for index, item in enumerate(sub_dados):
                    nova_chave = f"{chave_pai}[{index}]" if chave_pai else f"[{index}]"
                    _achatar(item, nova_chave)

        _achatar(dados)
        return lista_resultado

    def consultar_quantidade_de_itens(self):
                                                                    url_quantidade          =
f"https://pncp.gov.br/api/pncp/v1/orgaos/{self.cnpj}/compras/{self.ano}/{self.sequencial}/itens/qu
antidade"
        response_quantidade = requests.get(url_quantidade)
        response_quantidade.raise_for_status()
```

```
        data_quantidade = response_quantidade.json()

        if isinstance(data_quantidade, int):
            return data_quantidade
        else:
            raise Exception("Resposta inesperada da API para quantidade.")

DEFAULT_CONFIG = {
    "ultimo_cnpj": "",
    "ultimo_ano": "",
    "ultimo_sequencial": ""
}

class ConsultarAPI(QWidget):
    def __init__(self, icons, database_ata_manager, main_window, parent=None):
        super().__init__(parent)
        self.icons = icons
        self.db_manager = database_ata_manager
        self.main_window = main_window
        self.config_data = self.carregar_configuracoes(CONFIG_API_FILE)
        self.homologacao_dataframe = None
        self.setup_ui()
        self.carregar_valores_entrada()

    def carregar_configuracoes(self, config_path):
        config_path = Path(config_path)
        if config_path.is_file():
            # Carrega o arquivo de configuração
            with open(config_path, 'r', encoding='utf-8') as file:
                return json.load(file)
        else:
            # Cria o arquivo com valores padrão
            try:
                with open(config_path, 'w', encoding='utf-8') as file:
                    json.dump(DEFAULT_CONFIG, file, ensure_ascii=False, indent=4)
                  QMessageBox.information(self, "Informação", f"Arquivo de configuração criado em:
{config_path}")
                return DEFAULT_CONFIG
            except Exception as e:
                      QMessageBox.critical(self, "Erro", f"Não foi possível criar o arquivo de
configuração: {e}")
                return {}

    def carregar_valores_entrada(self):
        self.cnpj_input.setText(self.config_data.get("ultimo_cnpj", ""))
        self.ano_input.setText(self.config_data.get("ultimo_ano", ""))
        self.sequencial_input.setText(self.config_data.get("ultimo_sequencial", ""))

    def salvar_configuracoes(self):
        self.config_data["ultimo_cnpj"] = self.cnpj_input.text()
        self.config_data["ultimo_ano"] = self.ano_input.text()
        self.config_data["ultimo_sequencial"] = self.sequencial_input.text()
```

```
        try:
            with open(CONFIG_API_FILE, 'w', encoding='utf-8') as file:
                json.dump(self.config_data, file, ensure_ascii=False, indent=4)
            # QMessageBox.information(self, "Informação", "Configurações salvas com sucesso.")
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Erro ao salvar configurações: {e}")

    def setup_ui(self):
        # Layout principal vertical
        layout = QVBoxLayout(self)

        # Label principal "API de consulta"
        label_api = QLabel("API do Portal Nacional de Contratações Públicas (PNCP)")
        label_api.setAlignment(Qt.AlignmentFlag.AlignCenter)
        label_api.setFont(QFont('Arial', 16, QFont.Weight.Bold))
        layout.addWidget(label_api)

                orientacoes = QLabel("O padrão do ID contratação do PNCP é '[CNPJ da
Matriz]-1-[sequencial]/[ano]'.")
        orientacoes.setAlignment(Qt.AlignmentFlag.AlignLeft)
        orientacoes.setFont(QFont('Arial', 12))
        layout.addWidget(orientacoes)

        # HLayout CNPJ
        h_layout = QHBoxLayout()
        label_cnpj = QLabel("CNPJ:")
        label_cnpj.setFont(QFont('Arial', 12))
        self.cnpj_input = QLineEdit()
        self.cnpj_input.setFont(QFont('Arial', 12))
        h_layout.addWidget(label_cnpj)
        h_layout.addWidget(self.cnpj_input)

        # HLayout Ano
        label_ano = QLabel("Ano:")
        label_ano.setFont(QFont('Arial', 12))
        self.ano_input = QLineEdit()
        self.ano_input.setFont(QFont('Arial', 12))
        h_layout.addWidget(label_ano)
        h_layout.addWidget(self.ano_input)

        # HLayout Sequencial
        label_sequencial = QLabel("Sequencial:")
        label_sequencial.setFont(QFont('Arial', 12))
        self.sequencial_input = QLineEdit()
        self.sequencial_input.setFont(QFont('Arial', 12))
        h_layout.addWidget(label_sequencial)
        h_layout.addWidget(self.sequencial_input)
        layout.addLayout(h_layout)

        linha_divisoria1, spacer_baixo_linha1 = linha_divisoria_layout()
        layout.addWidget(linha_divisoria1)
        layout.addSpacerItem(spacer_baixo_linha1)
```

```
        # Área de progresso
        self.progress_api_area = QTextEdit()
        self.progress_api_area.setReadOnly(True)
        layout.addWidget(self.progress_api_area)

        # Botão "Consultar PNCP"
        button_layout = QHBoxLayout()
        button_layout.addStretch()  # Espaço flexível à esquerda
          add_button_func_vermelho("Consultar PNCP", self.iniciar_consulta, button_layout, "Clique
para consultar o sequencial da contratação no PNCP", button_size=(300, 40))
        button_layout.addStretch()  # Espaço flexível à direita
        layout.addLayout(button_layout)

        self.setLayout(layout)

    def open_results_treeview(self):
        if self.homologacao_dataframe is not None and not self.homologacao_dataframe.empty:
            # Certifique-se de passar os argumentos na ordem correta
            tree_view_window = TreeViewWindow(
                dataframe=self.homologacao_dataframe,
                icons_dir=self.icons_dir,       # Diretório de ícones
                db_manager=self.db_manager,     # Gerenciador de banco de dados
                parent=self
            )
            tree_view_window.exec()  # Abre a janela como modal
        else:
                  QMessageBox.warning(self, "Erro", "Não há dados disponíveis para mostrar no
TreeView.")

    def iniciar_consulta(self):
        cnpj = self.cnpj_input.text()
        ano = self.ano_input.text()
        sequencial = self.sequencial_input.text()

        self.progress_api_area.clear()
        self.progress_api_area.append("Iniciando consulta ao PNCP...")

        # Salvar os valores atuais no JSON
        self.salvar_configuracoes()

        # Iniciar a thread de consulta
        self.thread = PNCPConsultaThread(cnpj, ano, sequencial, self.db_manager)
        self.thread.consulta_concluida.connect(self.consulta_concluida)
        self.thread.erro_consulta.connect(self.exibir_erro)
        self.thread.progresso_consulta.connect(self.atualizar_progresso)

        self.thread.start()

    def consulta_concluida(self, data_informacoes_lista, resultados_completos):
        self.progress_api_area.append("Consulta concluída com sucesso.")

        # Carregar dados da tabela controle_atas_api
        controle_atas_api = self.db_manager.execute_query(
```

```
            "SELECT item, catalogo FROM controle_atas_api WHERE catalogo IS NOT NULL"
        )
        controle_atas_dict = {row[0]: row[1] for row in controle_atas_api}

        # Preenche o DataFrame com os dados da consulta
        self.homologacao_dataframe = self.db_manager.popular_db_consulta_itens_api(
            resultados_completos,
                  data_informacoes_lista[0],   # Assume que data_informacoes_lista tem os dados
principais
            data_informacoes_lista[0].get("numeroCompra"),
            data_informacoes_lista[0].get("anoCompra"),
            data_informacoes_lista[0]["unidadeOrgao"].get("codigoUnidade"),
                        controle_atas_dict    # Passa o dicionário de controle_atas_api para
popular_db_consulta_itens_api
        )

    def atualizar_progresso(self, mensagem):
        self.progress_api_area.append(mensagem)

    def consulta_concluida(self, data_informacoes_lista, resultados_completos):
        self.progress_api_area.append("Consulta concluída com sucesso.")

    def exibir_erro(self, mensagem_erro):
        self.progress_api_area.append(f"Erro: {mensagem_erro}")
```