

Arquivo: progresso_homolog.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from PyQt6.QtSql import QSqlQuery
import re
import time
import pandas as pd
from pathlib import Path
from modules.atas.widgets.worker_homologacao import Worker, TreeViewWindow, WorkerSICAF,
extrair_dados_sicaf
import logging
from modules.utils.add_button import add_button_func, add_button_func_vermelho

class ConclusaoDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Processamento Concluído")
        self.setup_ui()

    def setup_ui(self):
        main_layout = QVBoxLayout(self)
        message_label = QLabel("O processamento foi concluído com sucesso!")
        message_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        font = message_label.font()
        font.setPointSize(14)
        message_label.setFont(font)
        main_layout.addWidget(message_label)

        close_button = QPushButton("Fechar")
        close_button.clicked.connect(self.close)
        main_layout.addWidget(close_button)

        self.setLayout(main_layout)

class ProcessamentoWidget(QWidget):
    resultadosHomologacao = pyqtSignal(object)

    def __init__(self, pdf_dir, icons, model, database_ata_manager, main_window, parent=None):
        super().__init__(parent)
        self.pdf_dir = pdf_dir
        self.icon_cache = icons
        self.model = model
        self.database_ata_manager = database_ata_manager
        self.main_window = main_window
        self.homologacao_dataframe = None # Inicializa o dataframe como None
        self.setWindowTitle("Processamento")
        self.setup_ui()
        self.timer = QTimer(self)
        self.start_time = 0
```

```

def on_pdf_dir_changed(self, new_pdf_dir):
    """Atualiza o diretório PDF em ProcessamentoDialog."""
    # Atualiza o pdf_dir no diálogo de processamento se estiver aberto
    if hasattr(self, 'termo_homologacao_widget'):
        self.termo_homologacao_widget.pdf_dir = new_pdf_dir

def setup_ui(self):
    main_layout = QVBoxLayout(self)
    title = QLabel("Processamento de Documentos")
    title.setAlignment(Qt.AlignmentFlag.AlignCenter)
    title.setFont(QFont('Arial', 16, QFont.Weight.Bold))
    main_layout.addWidget(title)

    layout_instruction = QHBoxLayout()
    layout_instruction.addStretch()
    label_pdf = QLabel("Insira os arquivos PDF na pasta")
    label_pdf.setAlignment(Qt.AlignmentFlag.AlignCenter)
    label_pdf.setFont(QFont('Arial', 12, QFont.Weight.Bold))
    layout_instruction.addWidget(label_pdf)

    add_button_func("Abrir Pasta", "open-folder", self.abrir_pasta_pdf, layout_instruction,
self.icon_cache, "Clique para ver instruções", button_size=(150, 30))

    label_contador = QLabel("e depois atualize o Contador")
    label_contador.setAlignment(Qt.AlignmentFlag.AlignCenter)
    label_contador.setFont(QFont('Arial', 12, QFont.Weight.Bold))
    layout_instruction.addWidget(label_contador)

    add_button_func("Atualizar", "loading-arrow", self.update_pdf_count, layout_instruction,
self.icon_cache, "Clique para ver instruções", button_size=(150, 30))
    # add_button_func("Redefinir Pasta PDF", "add-folder", self.definir_pasta_pdf_padrao,
button_layout, self.icon_cache, "Clique para ver instruções")
    layout_instruction.addStretch()
    main_layout.addLayout(layout_instruction)

    layout_progress = QHBoxLayout()

    self.time_label = QLabel("Tempo decorrido: 0s")
    self.time_label.setFont(QFont('Arial', 12))
    layout_progress.addWidget(self.time_label)

    self.progress_bar = QProgressBar()
    self.progress_bar.setValue(0)

    # Define a fonte diretamente no QProgressBar
    self.progress_bar.setFont(QFont('Arial', 12))

    layout_progress.addWidget(self.progress_bar)

    main_layout.addLayout(layout_progress)

    self.context_area = QTextEdit()

```

```

self.context_area.setReadOnly(True)

# Define a fonte e o tamanho do texto
self.context_area.setFont(QFont('Arial', 12))

main_layout.addWidget(self.context_area)

self.setup_button_layout(main_layout)
self.setLayout(main_layout)

self.update_pdf_count()

def setup_button_layout(self, main_layout):
    button_layout = QHBoxLayout()

    add_button_func_vermelho("Iniciar Processamento", self.start_processing, button_layout,
"Clique para ver instruções", button_size=(300, 40))
    main_layout.addLayout(button_layout)

def abrir_pasta_pdf(self):
    # Garante que a pasta exista
    if not self.pdf_dir.exists():
        try:
            self.pdf_dir.mkdir(parents=True, exist_ok=True)
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Falha ao criar o diretório PDF: {e}")
            return

    # Abre a pasta PDF
    if self.pdf_dir.is_dir():
        QDesktopServices.openUrl(QUrl.fromLocalFile(str(self.pdf_dir)))
    else:
        QMessageBox.warning(self, "Erro", "O diretório PDF não é válido.")

def definir_pasta_pdf_padrao(self):
    folder = QFileDialog.getExistingDirectory(self, "Defina a Pasta PDF Padrão")
    if folder:
        # Atualiza self.pdf_dir com o novo caminho
        self.pdf_dir = Path(folder)
        # Emite o sinal para notificar a mudança de diretório PDF
        self.main_window.pdf_dir_changed.emit(self.pdf_dir)
        QMessageBox.information(self, "Pasta Padrão Definida", f"A pasta padrão foi definida
como: {folder}")

def update_context(self, text):
    self.context_area.append(text)

def update_pdf_count(self):
    if self.pdf_dir.exists() and self.pdf_dir.is_dir():
        pdf_count = len(list(self.pdf_dir.glob("*.pdf")))
        self.update_context(f"Quantidade de arquivos PDF: {pdf_count}")
    else:
        self.update_context("Diretório PDF não encontrado.")

```

```

def start_processing(self):
    self.current_dataframe = None

    if not self.verify_directories():
        return

    self.start_time = time.time()
    self.timer.timeout.connect(self.update_time)
    self.timer.start(1000)

    self.worker_thread = Worker(self.pdf_dir)
    self.worker_thread.processing_complete.connect(self.finalizar_processamento_homologacao)
    self.worker_thread.update_context_signal.connect(self.update_context)
    self.worker_thread.progress_signal.connect(self.progress_bar.setValue)

    self.worker_thread.start()

def finalizar_processamento_homologacao(self, extracted_data):
    self.update_context("Processamento concluído.")
    self.timer.stop() # Para o temporizador
    elapsed_time = int(time.time() - self.start_time)
    self.time_label.setText(f"Tempo total: {elapsed_time}s")

    # Processar dados e atualizar o treeView
    self.homologacao_dataframe = save_to_dataframe(extracted_data)
    # Salva os dados do DataFrame no modelo
    self.atualizar_ou_inserir_controle_homologacao()
    # # Atualiza o botão de registro SICAF
    # self.update_registro_sicaf_button()

def atualizar_ou_inserir_controle_homologacao(self):
    if self.homologacao_dataframe is None or self.homologacao_dataframe.empty:
        print("DataFrame de homologação está vazio ou não foi criado.")
        return

    unique_combinations = self.homologacao_dataframe[['uasg', 'num_pregao',
'ano_pregao']].dropna().drop_duplicates()

    if unique_combinations.empty:
        print("Não há combinações válidas de UASG e número do pregão.")
        return

    if len(unique_combinations) > 1:
        combinations = unique_combinations.apply(lambda row:
f"{row['uasg']}-{row['num_pregao']}-{row['ano_pregao']}", axis=1)
        QMessageBox.warning(None, "Cominações Múltiplas Encontradas",
f"As seguintes combinações únicas foram encontradas:\n" +
"\n".join(combinations))
        return

    uasg, num_pregao, ano_pregao = unique_combinations.iloc[0]
    table_name = f"result_{num_pregao}_{ano_pregao}_{uasg}"

```

```

# Deleta a tabela caso já exista e cria uma nova
if not self.recriar_tabela(table_name):
    return

# Copia os dados da tabela 'controle_atas' para a nova tabela
if not self.copiar_dados_controle_atas(table_name):
    return

# Processa o DataFrame para inserção ou atualização dos dados
self.processar_linhas_dataframe(table_name)
print(f"Dados do DataFrame inseridos/atualizados na tabela '{table_name}' com sucesso.")

def recriar_tabela(self, table_name):
    delete_table_query = QSqlQuery(self.model.database())
    if not delete_table_query.exec(f"DROP TABLE IF EXISTS {table_name}"):
        print(f"Erro ao deletar a tabela '{table_name}':",
delete_table_query.lastError().text())
        return False

    create_table_query = QSqlQuery(self.model.database())
    if not create_table_query.exec(f"""
        CREATE TABLE {table_name} AS
        SELECT * FROM controle_atas_api WHERE l=0
    """):
        print(f"Erro ao criar a tabela '{table_name}':",
create_table_query.lastError().text())
        return False
    return True

def copiar_dados_controle_atas(self, table_name):
    copy_query = QSqlQuery(self.model.database())
    if not copy_query.exec(f"""
        INSERT INTO {table_name} (item, catalogo, descricao, descricao_detalhada)
        SELECT item, catalogo, descricao, descricao_detalhada FROM controle_atas_api
    """):
        print(f"Erro ao copiar colunas específicas para '{table_name}':",
copy_query.lastError().text())
        return False
    return True

def processar_linhas_dataframe(self, table_name):
    for _, row in self.homologacao_dataframe.iterrows():
        item_value = row['item']
        if self.verificar_existencia_item(table_name, item_value):
            # Atualiza os valores nas colunas existentes para o 'item'
            self.atualizar_item_existente(table_name, item_value, row)
        else:
            # Insere um novo 'item' caso não exista
            self.inserir_novo_item(table_name, row)

def verificar_existencia_item(self, table_name, item_value):
    check_query = QSqlQuery(self.model.database())

```

```

        check_query.prepare(f"SELECT COUNT(*) FROM {table_name} WHERE item = :item")
        check_query.bindValue(":item", item_value)
        if not check_query.exec():
            print(f"Erro ao verificar existência do item {item_value}:",
                  check_query.lastError().text())
            return False
        check_query.next()
        return check_query.value(0) > 0

    def atualizar_item_existente(self, table_name, item_value, row):
        update_query = QSqlQuery(self.model.database())
        update_fields = ", ".join([f"{col} = :{col}" for col in self.homologacao_dataframe.columns
                                   if col not in ["item", "catalogo", "descricao",
                                                  "descricao_detalhada"]])
        update_query.prepare(f"UPDATE {table_name} SET {update_fields} WHERE item = :item")

        for col in self.homologacao_dataframe.columns:
            if col not in ["catalogo", "descricao", "descricao_detalhada"]:
                update_query.bindValue(f":{col}", row[col])

        update_query.bindValue(":item", item_value)
        if not update_query.exec():
            print(f"Erro ao atualizar o item {item_value} em '{table_name}':",
                  update_query.lastError().text())

    def inserir_novo_item(self, table_name, row):
        columns = ", ".join(self.homologacao_dataframe.columns)
        placeholders = ", ".join([f":{col}" for col in self.homologacao_dataframe.columns])
        insert_query = QSqlQuery(self.model.database())
        insert_query.prepare(f"INSERT INTO {table_name} ({columns}) VALUES ({placeholders})")

        for col in self.homologacao_dataframe.columns:
            insert_query.bindValue(f":{col}", row[col])

        if not insert_query.exec():
            print(f"Erro ao inserir o item {row['item']} em '{table_name}':",
                  insert_query.lastError().text())

    def save_data(self, table_name):
        if isinstance(self.current_dataframe, pd.DataFrame) and not self.current_dataframe.empty:
            # print("Salvando DataFrame com as colunas:", self.current_dataframe.columns)
            try:
                with self.model as conn:
                    self.current_dataframe.to_sql(table_name, conn, if_exists='replace',
                                                  index=False)
            except Exception as e:
                QMessageBox.critical(self, "Erro", f"Erro ao salvar os dados no banco de dados: {e}")
            else:
                QMessageBox.critical(self, "Erro", "Nenhum DataFrame válido disponível para salvar ou o objeto não é um DataFrame.")

```

```

def update_time(self):
    elapsed_time = int(time.time() - self.start_time)
    self.time_label.setText(f"Tempo decorrido: {elapsed_time}s")
    self.time_label.setFont(QFont('Arial', 12))

def verify_directories(self):
    if not self.pdf_dir.exists() or not self.pdf_dir.is_dir():
        self.update_context("Diretório PDF não encontrado.")
        return False
    return True

def setup_treeview_styles(self):
    header = self.treeView.header()
    header.setSectionResizeMode(QHeaderView.ResizeMode.ResizeToContents)

def abrir_registro_sicaf(self):
    if self.homologacao_dataframe is None:
        QMessageBox.warning(self, "Erro", "Dados não disponíveis.")
        return

    # Instancia e exibe o diálogo de Registro SICAF com pdf_dir correto
    dialog = RegistroSICAFDialog (
        homologacao_dataframe=self.homologacao_dataframe,
        pdf_dir=self.pdf_dir,
        model=self.model,
        icons=self.icon_cache,
        database_ata_manager=self.database_ata_manager,
        update_context_callback=self.update_context,
        parent=self
    )
    dialog.exec()

# def update_registro_sicaf_button(self):
#     # Verifica a existência do DataFrame para habilitar o botão
#     self.registro_sicaf_button.setEnabled(self.homologacao_dataframe is not None)

class RegistroSICAFDialog(QDialog):
    def __init__(self, homologacao_dataframe, pdf_dir, model, icons, database_ata_manager,
parent=None, update_context_callback=None):
        super().__init__(parent)
        self.homologacao_dataframe = homologacao_dataframe
        self.sicaf_dir = pdf_dir / 'pasta_sicaf'
        self.model = model
        self.icon_cache = icons
        self.database_ata_manager = database_ata_manager #
        self.update_context = update_context_callback
        self.setWindowTitle("Registro SICAF")
        self.setFixedSize(1000, 600)
        self.setup_ui()

def setup_ui(self):

```

```

"""Configura a interface do diálogo."""
main_layout = QHBoxLayout(self)

# Cria e adiciona o layout esquerdo
left_layout = self.criar_layout_esquerdo()
main_layout.addLayout(left_layout)

# Cria e adiciona o layout direito
right_widget = self.criar_layout_direito()
main_layout.addWidget(right_widget)

self.setLayout(main_layout)

# Chama atualizar_lista após todos os elementos serem inicializados
self.atualizar_lista()

def criar_layout_esquerdo(self):
    left_layout = QVBoxLayout()

    # Adiciona o layout da legenda
    legenda_layout = self.criar_legenda_layout()
    left_layout.addLayout(legenda_layout)

    # Define o QListWidget para a lista de empresas e CNPJs
    self.left_list_widget = QListWidget()
    self.left_list_widget.setStyleSheet("QListWidget::item { text-align: left; }")

    # Adiciona o QListWidget ao layout esquerdo
    left_layout.addWidget(self.left_list_widget)

    # Botão "Processamento de SICAF" com ícone 'processing'
    processing_button = self.create_button(
        text="Processamento de SICAF",
        icon=self.icon_cache["processing"],
        callback=self.iniciar_processamento_sicaf, # Método de callback ao clicar no botão
        tooltip_text="Iniciar processamento do SICAF"
    )

    # Adiciona o botão ao layout e alinha ao centro
    left_layout.addWidget(processing_button, alignment=Qt.AlignmentFlag.AlignHCenter)

    return left_layout

def iniciar_processamento_sicaf(self):
    """Inicializa o processamento SICAF com atualizações de contexto para cada arquivo PDF
    analisado."""
    if not self.sicaf_dir.exists():
        QMessageBox.warning(self, "Erro", "A pasta SICAF não existe.")
        return

    if self.update_context: # Verifica se `update_context` não é None
        self.update_context("Iniciando o processamento dos arquivos SICAF...")

```



```

# Cria uma instância de WorkerSICAF e conecta os sinais
self.worker = WorkerSICAF(self.sicaf_dir)
self.worker.processing_complete.connect(self.on_processing_complete)

        self.worker.update_context_signal.connect(self.update_context)    # Conecta o sinal ao
método de atualização de contexto

# Inicia o Worker
self.worker.start()

def on_processing_complete(self, dataframes):
    """Salva os dados extraídos no banco de dados 'registro_sicaf'."""
    for df in dataframes:
        for _, row in df.iterrows():
            # Extrai as colunas para o registro
            empresa = row.get('empresa')
            cnpj = row.get('cnpj')
            nome_fantasia = row.get('nome_fantasia')
            endereco = row.get('endereco')
            cep = row.get('cep')
            municipio = row.get('municipio')
            telefone = row.get('telefone')
            email = row.get('email')
            responsavel_legal = row.get('nome')    # Assumindo que o CPF é do responsável legal

            # Verifica se o CNPJ já existe na tabela
            check_query = "SELECT 1 FROM registro_sicaf WHERE cnpj = ?"
            exists = self.database_ata_manager.execute_query(check_query, (cnpj,))

            # Se o CNPJ já existe, atualiza os dados, caso contrário, insere um novo registro
            if exists:
                update_query = """
                UPDATE registro_sicaf SET
                    empresa = ?,
                    nome_fantasia = ?,
                    endereco = ?,
                    cep = ?,
                    municipio = ?,
                    telefone = ?,
                    email = ?,
                    responsavel_legal = ?
                WHERE cnpj = ?
                """
                params = (empresa, nome_fantasia, endereco, cep, municipio, telefone, email,
responsavel_legal, cnpj)
                try:
                    self.database_ata_manager.execute_update(update_query, params)
                    print(f"Registro de {empresa} atualizado com sucesso.")
                except Exception as e:
                    logging.error(f"Erro ao atualizar o registro de {empresa}: {e}")
                    QMessageBox.warning(self, "Erro", f"Erro ao atualizar o registro de
{empresa}: {e}")
            else:

```

```

        insert_query = """
            INSERT INTO registro_sicaf (empresa, cnpj, nome_fantasia, endereco, cep,
municipio, telefone, email, responsavel_legal)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
        """

        params = (empresa, cnpj, nome_fantasia, endereco, cep, municipio, telefone,
email, responsavel_legal)

        try:
            self.database_ata_manager.execute_query(insert_query, params)
            print(f"Registro de {empresa} inserido com sucesso.")
        except Exception as e:
            logging.error(f"Erro ao inserir o registro de {empresa}: {e}")
            QMessageBox.warning(self, "Erro", f"Erro ao inserir o registro de
{empresa}: {e}")

    # Mensagem final de conclusão do processamento
    QMessageBox.information(self, "Processamento Completo", "Todos os registros foram
processados e salvos no banco de dados com sucesso.")
    print("Processamento SICAF concluído e registros salvos no banco de dados.")

def create_button(self, text, icon, callback, tooltip_text, icon_size=QSize(30, 30),
button_size=QSize(120, 30)):
    """Cria um botão personalizado com texto, ícone, callback e tooltip."""
    btn = QPushButton(text)
    if icon:
        btn.setIcon(icon)
        btn.setIconSize(icon_size)
    btn.clicked.connect(callback)
    btn.setToolTip(tooltip_text)

    # Define o tamanho fixo do botão
    btn.setFixedSize(button_size.width(), button_size.height())

    btn.setStyleSheet("""
QPushButton {
    font-size: 12pt;
    padding: 5px;
}
""")

    return btn

def criar_legenda_layout(self):
    """Cria o layout da legenda com os ícones de confirmação e cancelamento."""
    legenda_layout = QHBoxLayout()
    legenda_layout.setAlignment(Qt.AlignmentFlag.AlignLeft)

    # Texto da legenda
    legenda_text = QLabel("Legenda: ")

    # Ícone de confirmação
    confirm_icon = QLabel()

```

```

confirm_icon.setPixmap(self.icon_cache["check"].pixmap(24, 24))
confirm_text = QLabel("SICAF encontrado")

# Ícone de cancelamento
cancel_icon = QLabel()
cancel_icon.setPixmap(self.icon_cache["cancel"].pixmap(24, 24))
cancel_text = QLabel("SICAF não encontrado")

# Adiciona elementos ao layout da legenda
legenda_layout.addWidget(legenda_text)
legenda_layout.addWidget(confirm_icon)
legenda_layout.addWidget(confirm_text)
legenda_layout.addWidget(cancel_icon)
legenda_layout.addWidget(cancel_text)

return legenda_layout

def criar_layout_direito(self):
    """Cria o layout direito com a lista de arquivos PDF e os botões."""
    right_widget = QWidget()
    right_widget.setFixedWidth(350)
    right_layout = QVBoxLayout(right_widget)

    # Adiciona os botões "Abrir Pasta" e "Atualizar" ao layout
    top_right_layout = self.criar_botoes_direitos()
    right_layout.addLayout(top_right_layout)

    # Conta e exibe a quantidade de arquivos PDF
    self.right_label = QLabel(self.obter_texto_arquivos_pdf())
    right_layout.addWidget(self.right_label)

    # Define o QListWidget para a lista de arquivos PDF
    self.pdf_list_widget = QListWidget()
    self.load_pdf_files()
    right_layout.addWidget(self.pdf_list_widget)

    return right_widget

def load_pdf_files(self):
    """Carrega arquivos .pdf da pasta sicaf_dir e os exibe na lista."""
    if self.sicaf_dir.exists() and self.sicaf_dir.is_dir():
        pdf_files = list(self.sicaf_dir.glob("*.pdf"))
        for pdf_file in pdf_files:
            self.pdf_list_widget.addItem(pdf_file.name)
    else:
        self.pdf_list_widget.addItem("Nenhum arquivo PDF encontrado.")

def criar_botoes_direitos(self):
    """Cria o layout horizontal com os botões 'Abrir Pasta' e 'Atualizar'."""
    top_right_layout = QHBoxLayout()

```

```

# Botão "Abrir Pasta"
abrir_pasta_button = QPushButton("Abrir Pasta")
abrir_pasta_button.clicked.connect(self.abrir_pasta_sicaf)

# Botão "Atualizar"
atualizar_button = QPushButton("Atualizar")
atualizar_button.clicked.connect(self.atualizar_lista)

# Adiciona os botões ao layout horizontal
top_right_layout.addWidget(abrir_pasta_button)
top_right_layout.addWidget(atualizar_button)

return top_right_layout

def obter_texto_arquivos_pdf(self):
    """Gera o texto exibido para a quantidade de arquivos PDF encontrados."""
    quantidade = self.count_pdf_files()
    if quantidade == 0:
        return "Nenhum arquivo PDF encontrado na pasta."
    elif quantidade == 1:
        return "1 arquivo PDF encontrado na pasta."
    else:
        return f"{quantidade} arquivos PDF encontrados na pasta."

def count_pdf_files(self):
    """Conta os arquivos PDF na pasta sicaf_dir."""
    if self.sicaf_dir.exists() and self.sicaf_dir.is_dir():
        return len(list(self.sicaf_dir.glob("*.pdf")))
    return 0

def atualizar_lista(self):
    """Atualiza o conteúdo de self.left_list_widget com dados de empresa e cnpj e o conteúdo
de self.pdf_list_widget com os arquivos PDF."""

# Verifica e cria a tabela 'registro_sicaf' no banco de dados, se necessário
create_table_query = """
CREATE TABLE IF NOT EXISTS registro_sicaf (
    empresa TEXT,
    cnpj TEXT PRIMARY KEY,
    nome_fantasia,
    endereco TEXT,
    cep TEXT,
    municipio TEXT,
    telefone TEXT,
    email TEXT,
    responsavel_legal TEXT
)
"""
try:
    self.database_atm.execute_query(create_table_query)
except Exception as e:
    QMessageBox.critical(self, "Erro no Banco de Dados", f"Erro ao criar a tabela
registro_sicaf: {e}")

```

```

        return

# Atualiza a lista de empresas e CNPJs no layout esquerdo
self.left_list_widget.clear()
unique_combinations = self.homologacao_dataframe[['empresa', 'cnpj']].drop_duplicates()

for _, row in unique_combinations.iterrows():
    empresa = row['empresa']
    cnpj = row['cnpj']

    # Ignora combinações onde 'empresa' ou 'cnpj' são nulos
    if pd.isnull(empresa) or pd.isnull(cnpj):
        continue

    # Verifica se o CNPJ já existe na tabela registro_sicaf
    check_cnpj_query = "SELECT 1 FROM registro_sicaf WHERE cnpj = ?"
    exists = self.database_atm_manager.execute_query(check_cnpj_query, (cnpj,))

    # Insere empresa e cnpj no banco de dados, se o CNPJ não existir
    if not exists:
        insert_query = """
        INSERT INTO registro_sicaf (empresa, cnpj)
        VALUES (?, ?)
        """
        try:
            self.database_atm_manager.execute_query(insert_query, (empresa, cnpj))
        except Exception as e:
            logging.error(f"Erro ao inserir empresa e CNPJ no banco de dados: {e}")

    # Criação de um item no QListWidget
    item_widget = QWidget()
    item_layout = QHBoxLayout(item_widget)
    item_layout.setAlignment(Qt.AlignmentFlag.AlignLeft)

    # Define o ícone de acordo com a existência do CNPJ no banco de dados
    icon_label = QLabel()
    icon_label.setPixmap(self.get_icon_for_cnpj(cnpj).pixmap(24, 24))

    empresa_label = QLabel(f"{cnpj} - {empresa}")
    copiar_button = QPushButton("Copiar CNPJ")

    # Conectar o botão de cópia ao método de cópia
    copiar_button.clicked.connect(lambda _, cnpj=cnpj:
self.copiar_para_area_de_transferencia(cnpj))

    # Adicionar ícone, botão e label ao layout do item
    item_layout.addWidget(icon_label)
    item_layout.addWidget(copiar_button)
    item_layout.addWidget(empresa_label)

    # Adiciona o item personalizado à QListWidget
    list_item = QListWidgetItem(self.left_list_widget)
    list_item.setSizeHint(item_widget.sizeHint())

```

```

        self.left_list_widget.addItem(list_item)
        self.left_list_widget.setItemWidget(list_item, item_widget)

# Atualiza a lista de arquivos PDF no layout direito
self.pdf_list_widget.clear()
pdf_files = list(self.sicaf_dir.glob("*.pdf"))

if pdf_files:
    for pdf_file in pdf_files:
        self.pdf_list_widget.addItem(pdf_file.name)
else:
    self.pdf_list_widget.addItem("Nenhum arquivo PDF encontrado.")

# Atualiza o texto do right_label com a contagem de arquivos PDF
quantidade = len(pdf_files)
if quantidade == 0:
    right_label_text = "Nenhum arquivo PDF encontrado na pasta."
elif quantidade == 1:
    right_label_text = "1 arquivo PDF encontrado na pasta."
else:
    right_label_text = f"{quantidade} arquivos PDF encontrados na pasta."

# Atualiza o texto do right_label dinamicamente
self.right_label.setText(right_label_text)

def abrir_pasta_sicaf(self):
    """Abre o diretório sicaf_dir no explorador de arquivos, criando-o se não existir."""
    if not self.sicaf_dir.exists():
        self.sicaf_dir.mkdir(parents=True, exist_ok=True)

    QDesktopServices.openUrl(QUrl.fromLocalFile(str(self.sicaf_dir)))

def get_icon_for_cnpj(self, cnpj):
    """Verifica se o CNPJ existe na tabela registro_sicaf usando db_manager e retorna o ícone
em cache correspondente."""
    try:
        query = "SELECT 1 FROM registro_sicaf WHERE cnpj = ?"
        result = self.database_ata_manager.execute_query(query, (cnpj,)) # Usa db_manager
        para a consulta
        return self.icon_cache["check"] if result else self.icon_cache["cancel"]
    except Exception as e:
        QMessageBox.critical(self, "Erro no Banco de Dados", f"Carregamento SICAF: Erro ao
acessar o banco de dados: {e}")
        return self.icon_cache["cancel"] # Ícone padrão em caso de erro

def load_icons(self):
    """Carrega ícones e os armazena em cache."""
    icon_cache = {}
    icon_paths = {
        "check": self.icons_dir / "check.png",
        "cancel": self.icons_dir / "cancel.png"
    }

```

```

for key, path in icon_paths.items():
    icon = QIcon(str(path)) if path.exists() else QIcon()
    icon_cache[key] = icon

```

```

return icon_cache

```

```

def copiar_para_area_de_transferencia(self, cnpj):
    clipboard = QApplication.clipboard()
    clipboard.setText(cnpj)

    # Cria a QMessageBox manualmente
    msg_box = QMessageBox(self)
    msg_box.setIcon(QMessageBox.Icon.Information)
    msg_box.setWindowTitle("CNPJ Copiado")
    msg_box.setText(f"O CNPJ {cnpj} foi copiado para a área de transferência.")
    msg_box.setStandardButtons(QMessageBox.StandardButton.Ok)

    # Cria um temporizador para fechar a mensagem automaticamente
    QTimer.singleShot(2000, msg_box.close) # 2000 milissegundos = 2 segundos

    # Exibe a mensagem
    msg_box.exec()

```

```

padrao_1 = (
    (r"UASG\s+(?P<uasg>\d+)\s+-\s+(?P<orgao_responsavel>.+?)\s+PREGÃO\s+(?P<num_pregao>\d+)/(?P<ano_pregao>\d+)")
    padrao_srp = r"(?P<srp>SRP - Registro de Preço|SISPP - Tradicional)"
    padrao_objeto = (r"Objeto da compra:\s*(?P<objeto>.*?)\s*Entrega de propostas:")

```

```

padrao_grupo2 = (
    r"Item\s+(?P<item>\d+)(?:\s+do\s+Grupo\s+G(?P<grupo>\d+))?.*?"
    r"Valor\s+estimado:\s+R$\s+(?P<valor>[\d,\.]+).*?"
    r"(?:Critério\s+de\s+julgamento:\s+(?P<crit_julgamento>.*?))?\s*"
    r"Quantidade:\s+(?P<quantidade>\d+)\s+"
    r"Unidade\s+de\s+fornecimento:\s+(?P<unidade>.*?)\s+"
    r"Situação:\s+(?P<situacao>Adjudicado e Homologado|Deserto e Homologado|Fracassado e Homologado|Anulado e Homologado|Revogado e Homologado)"
)

```

```

padrao_item2 = (
    r"Item\s+(?P<item>\d+)\s+-\s+.*?"
    r"Quantidade:\s+(?P<quantidade>\d+)\s+"
    r"Valor\s+estimado:\s+R$\s+(?P<valor>[\d,\.]+)(?:\s+(\(unitário\)))?\s+"
    r"Unidade\s+de\s+fornecimento:\s+(?P<unidade>.*?)(?:\s+R$\s+|$)"
    r".*?Situação:\s+(?P<situacao>Adjudicado e Homologado|Deserto e Homologado|Fracassado e Homologado|Anulado e Homologado|Revogado e Homologado)"
)

```

```

padrao_4 = (
    r"Proposta\s+adjudicada.*?"

```

```

r"Marca/Fabricante\s*:\s*(?P<marca_fabricante>.*?)\s*"
r"Modelo/versão\s*:\s*(?P<modelo_versao>.*?)(?=\s*\d{2}/\d{2}/\d{4}|\s*Valor\s+proposta\s*:)"
)

padrao_3 = (

r"(Adjudicado|Adjudicado)\s+e\s+Homologado\s+por\s+CPF\s+(?P<cpf_od>*\*\*\.\d{3}.\*\*\*-*\d{1})\s+-\s+"
r"(?P<ordenador_despesa>[^\d,]+?)\s+para\s+"
r"(?P<empresa>.*?)(?=\s*,\s*CNPJ\s+)"
r"\s*,\s*CNPJ\s+(?P<cnnpj>\d{2}\.\d{3}\.\d{3}/\d{4}-\d{2}),\s+"
r"melhor\s+lance\s*:\s*R\$s*(?P<melhor_lance>[\d,.]+)\s*\(\unitário\)\s*"
r"/\s*R\$s*(?P<total_lance>[\d,.]+)\s*\(\total\)"
r"(?:,\s+valor\s+negociado\s*:\s*R\$s*(?P<valor_negociado>[\d,.]+))?\s+"
r"Propostas\s+do\s+Item"
)

def processar_item(match, conteudo: str, ultima_posicao_processada: int, padrao_3: str, padrao_4: str) -> dict:
    item = match.groupdict()
    item_data = {
        "item": int(item['item']) if 'item' in item and item['item'].isdigit() else 'N/A',
        "grupo": item.get('grupo', 'N/A'),
        "valor_estimado": item.get('valor', 'N/A'),
        "quantidade": item.get('quantidade', 'N/A'),
        "unidade": item.get('unidade', 'N/A'),
        "situacao": item.get('situacao', 'N/A')
    }

    # Somente tenta buscar correspondências de padrões adicionais se a situação não for 'Anulado e Homologado'
    if item_data['situacao'] not in ['Anulado e Homologado', 'Revogado e Homologado']:
        # Processamento do padrão 3
        match_3 = re.search(padrao_3, conteudo[ultima_posicao_processada:], re.DOTALL | re.IGNORECASE)
        if match_3:
            ultima_posicao_processada += match_3.end()
            empresa_name = match_3.group('empresa').replace('\n', ' ').strip() if match_3.group('empresa') else 'N/A'
            cnnpj_value = match_3.group('cnnpj').replace(" ", "").replace("\n", "") if match_3.group('cnnpj') else 'N/A'
            melhor_lance = match_3.group('melhor_lance').replace(" ", "").replace("\n", "") if match_3.group('melhor_lance') else 'N/A'
            valor_negociado = match_3.group('valor_negociado')

            # Remove espaços e quebras de linha se valor_negociado não for None
            valor_negociado = valor_negociado.replace(" ", "").replace("\n", "") if valor_negociado else 'N/A'

            item_data.update({
                "melhor_lance": melhor_lance,
                "valor_negociado": valor_negociado,
                "ordenador_despesa": match_3.group('ordenador_despesa') or 'N/A',

```



```

        "empresa": empresa_name,
        "cnpj": cnpj_value,
    })

    # Processamento do padrão 4
    match_4 = re.search(padrao_4, conteudo[ultima_posicao_processada:], re.DOTALL |
re.IGNORECASE)
    if match_4:
        item_data.update({
            "marca_fabricante": match_4.group('marca_fabricante') or 'N/A',
            "modelo-versao": match_4.group('modelo-versao') or 'N/A',
        })

    return item_data, ultima_posicao_processada

def create_dataframe_from_pdf_files(extracted_data):
    """
    Cria um DataFrame a partir de dados extraídos de arquivos de texto.

    Args:
        extracted_data (list): Lista de dicionários com os dados extraídos dos arquivos de texto.

    Returns:
        pd.DataFrame: DataFrame criado com os dados extraídos.
    """
    if not isinstance(extracted_data, list):
        raise TypeError("extracted_data deve ser uma lista de dicionários.")

    all_data = []
    print("\nIniciando processamento de extracted_data...\n")

    for idx, item in enumerate(extracted_data):
        # Verifica se 'item' é um dicionário e contém a chave 'text'
        if isinstance(item, dict) and 'text' in item and isinstance(item['text'], str):
            content = item['text']
            print(f"\nProcessando item {idx + 1} de {len(extracted_data)}:")
            print(f"Conteúdo: {content[:100]}..." # Exibe os primeiros 100 caracteres para
verificar o conteúdo

            uasg_pregao_data = extrair_uasg_e_pregao(content, padrao_1, padrao_srp, padrao_objeto)
            print(f"Dados UASG e Pregão extraídos: {uasg_pregao_data}")

            compra_data = extrair_objeto_da_compra(content)
            print(f"Dados de Objeto da Compra extraídos: {compra_data}")

            items_data = identificar_itens_e_grupos(content, padrao_grupo2, padrao_item2,
padrao_3, padrao_4, pd.DataFrame())
            print(f"Dados dos Itens e Grupos extraídos: {items_data}")

            for item_data in items_data:
                # Verifica se item_data é um dicionário antes de adicionar a all_data
                if isinstance(item_data, dict):
                    all_data.append({

```

```

        **uasg_pregao_data,
        "objeto": compra_data, # Ajusta para ter "objeto" como string do campo
extraído

        **item_data
    })
    else:
        print(f"Formato inesperado em item_data: {item_data}")
    else:
        print(f"Formato inválido em extracted_data: {item} (Tipo: {type(item)})")

# Verifica se o all_data não está vazio antes de criar o DataFrame
if not all_data:
    raise ValueError("Nenhum dado válido foi encontrado para criar o DataFrame.")

dataframe_licitacao = pd.DataFrame(all_data)
print("\nDataFrame criado com os dados extraídos:")
print(dataframe_licitacao.head()) # Verifica os primeiros registros do DataFrame

if "item" not in dataframe_licitacao.columns:
    raise ValueError("A coluna 'item' não foi encontrada no DataFrame.")

return dataframe_licitacao.sort_values(by="item")

# Verificações adicionais nos pontos críticos da função de extração
def identificar_itens_e_grupos(conteudo: str, padrao_grupo2: str, padrao_item2: str, padrao_3:
str, padrao_4: str, df: pd.DataFrame) -> list:
    conteudo = re.sub(r'\s+', ' ', conteudo).strip()
    itens_data = []
    itens = buscar_itens(conteudo, padrao_grupo2, padrao_item2)

    print(f"Total de itens encontrados: {len(itens)}")
    if len(itens) == 0:
        print("Nenhuma correspondência encontrada para os padrões fornecidos.")
    else:
        print(f"Itens encontrados: {[match.groupdict() for match in itens]}")

    ultima_posicao_processada = 0

    for idx, match in enumerate(itens):
        item_data, ultima_posicao_processada = processar_item(match, conteudo,
ultima_posicao_processada, padrao_3, padrao_4)

        print(f"\nProcessando item {idx + 1}: {item_data}")

        item_data = process_cnpj_data(item_data)
        print(f"Item {idx + 1} após processar dados CNPJ: {item_data}")

        itens_data.append(item_data)

    print(f"\nItens finais processados: {itens_data}")
    return itens_data

```

```

def process_cnpj_data(cnpj_dict):
    """Converter "valor_estimado", "melhor_lance", e "valor_negociado" para float se não for
    possível deverá pular"""
    for field in ["valor_estimado", "melhor_lance", "valor_negociado"]:
        valor = cnpj_dict.get(field, 'N/A') # Usa 'N/A' como valor padrão se a chave não existir
        if isinstance(valor, str):
            try:
                cnpj_dict[field] = float(valor.replace(".", "").replace(",", "."))
            except ValueError:
                cnpj_dict[field] = 'N/A'

    # Convert "quantidade" to integer if possible, otherwise keep as is
    quantidade = cnpj_dict.get("quantidade", 'N/A')
    try:
        cnpj_dict["quantidade"] = int(quantidade)
    except ValueError:
        pass

    # Ensure valor_homologado_item_unitario is defined
    valor_negociado = cnpj_dict.get("valor_negociado", 'N/A')
    if valor_negociado in [None, "N/A", "", "none", "null"]:
        cnpj_dict["valor_homologado_item_unitario"] = cnpj_dict.get("melhor_lance", 'N/A')
    else:
        cnpj_dict["valor_homologado_item_unitario"] = valor_negociado

    # Now perform the other calculations
    valor_estimado = cnpj_dict.get("valor_estimado", 'N/A')
    valor_homologado_item_unitario = cnpj_dict.get("valor_homologado_item_unitario", 'N/A')
    if valor_estimado != 'N/A' and valor_homologado_item_unitario != 'N/A':
        try:
            cnpj_dict["valor_estimado_total_do_item"] = cnpj_dict["quantidade"] *
float(valor_estimado)
            cnpj_dict["valor_homologado_total_item"] = cnpj_dict["quantidade"] *
float(valor_homologado_item_unitario)
            cnpj_dict["percentual_desconto"] = (1 - (float(valor_homologado_item_unitario) /
float(valor_estimado))) * 100
        except ValueError:
            pass

    return cnpj_dict

def buscar_itens(conteudo: str, padrao_grupo2: str, padrao_item2: str) -> list:
    """
    Busca itens no conteúdo fornecido utilizando os padrões especificados.

    Args:
        conteudo (str): O texto onde os itens serão buscados.
        padrao_grupo2 (str): O padrão regex para capturar itens com estrutura 'Item do Grupo'.
        padrao_item2 (str): O padrão regex para capturar itens com estrutura 'Item -'.

    Returns:
        list: Uma lista de correspondências encontradas ou uma mensagem de falha.
    """

```

```

# Normaliza os espaços em branco para tratar quebras de linha e múltiplos espaços
conteudo = re.sub(r'\s+', ' ', conteudo).strip()

# Primeiro, busca correspondências para padrao_item2 com re.DOTALL e re.IGNORECASE
matches_item2 = list(re.finditer(padrao_item2, conteudo, re.DOTALL | re.IGNORECASE))
if matches_item2:
    print(f"Total de correspondências para padrao_item2: {len(matches_item2)}")
    for idx, match in enumerate(matches_item2, start=1):
        print(f"Correspondência {idx} para padrao_item2: {match.groupdict()}")
    return matches_item2 # Retorna se encontrar correspondências para padrao_item2

# Se não encontrou padrao_item2, tenta buscar correspondências para padrao_grupo2 com
re.DOTALL e re.IGNORECASE
matches_grupo2 = list(re.finditer(padrao_grupo2, conteudo, re.DOTALL | re.IGNORECASE))
if matches_grupo2:
    print(f"Total de correspondências para padrao_grupo2: {len(matches_grupo2)}")
    for idx, match in enumerate(matches_grupo2, start=1):
        print(f"Correspondência {idx} para padrao_grupo2: {match.groupdict()}")
    return matches_grupo2 # Retorna se encontrar correspondências para padrao_grupo2

# Se não encontrou correspondências para nenhum dos padrões, indica a falha
print("Nenhuma correspondência encontrada para padrao_item2 ou padrao_grupo2.")
return [] # Retorna uma lista vazia para indicar que não houve correspondências

def extrair_objeto_da_compra(conteudo: str) -> str:
    """
    Extraí o valor do 'Objeto da compra' usando uma regex mais robusta.

    Args:
        conteudo (str): O texto que contém o termo 'Objeto da compra'.

    Returns:
        str: O valor do 'Objeto da compra' se encontrado, caso contrário 'N/A'.
    """
    # Regex mais robusta para capturar o objeto da compra entre "Objeto da compra:" e "Entrega de
    propostas:"
    padrao_objeto_forte =
r"Objeto\s+da\s+compra\s*:\s*(?P<objeto>.*?)\s*Entrega\s+de\s+propostas\s*:"

    # Tentar encontrar a correspondência usando o padrão fornecido
    match = re.search(padrao_objeto_forte, conteudo, re.DOTALL | re.IGNORECASE)

    # Verificar se a correspondência foi encontrada
    if match:
        print("Padrão Objeto encontrado:")
        print(f"Valor do Objeto: {match.group('objeto')}")
        return match.group("objeto").strip()
    else:
        print("Padrão Objeto não encontrado.")
        return "N/A"

def extrair_uasg_e_pregao(conteudo: str, padrao_l: str, padrao_srp: str, padrao_objeto: str) ->
dict:

```

```

match = re.search(padrao_1, conteudo)
match2 = re.search(padrao_srp, conteudo)
match3 = re.search(padrao_objeto, conteudo)

srp_valor = match2.group("srp") if match2 else "N/A"
objeto_valor = match3.group("objeto") if match3 else "N/A"

# Caso o padrao_objeto não encontre, tente com o padrao_objeto_forte
if not match3:
    # Regex mais robusta para capturar o objeto da compra entre "Objeto da compra:" e "Entrega
    de propostas:"
    padrao_objeto_forte =
r"Objeto\s+da\s+compra\s*:\s*(?P<objeto>.*?)\s*Entrega\s+de\s+propostas\s*:"
    match_forte = re.search(padrao_objeto_forte, conteudo, re.DOTALL | re.IGNORECASE)
    if match_forte:
        print("Padrão Objeto Forte encontrado:")
        print(f"Valor do Objeto: {match_forte.group('objeto')}")
        objeto_valor = match_forte.group("objeto").strip()
    else:
        print("Padrão Objeto Forte não encontrado.")

# Adicionando prints para verificar se os padrões foram encontrados
if match:
    print("Padrão 1 encontrado:")
else:
    print("Padrão 1 não encontrado.")

if match2:
    print("Padrão SRP encontrado")
else:
    print("Padrão SRP não encontrado.")

if match3 or match_forte:
    print("Padrão Objeto encontrado")
else:
    print("Padrão Objeto não encontrado.")

if match:
    return {
        "uasg": match.group("uasg"),
        "orgao_responsavel": match.group("orgao_responsavel"),
        "num_pregao": match.group("num_pregao"),
        "ano_pregao": match.group("ano_pregao"),
        "srp": srp_valor,
        "objeto": objeto_valor
    }
return {}

def save_to_dataframe(extracted_data):
    df_extracted = create_dataframe_from_pdf_files(extracted_data)
    df_extracted['item'] = pd.to_numeric(df_extracted['item'], errors='coerce').astype('Int64')
    return df_extracted

```

```

# if tr_variavel_df_carregado is not None:
#     tr_variavel_df_carregado['item'] = pd.to_numeric(tr_variavel_df_carregado['item'],
errors='coerce').astype('Int64')
#     merged_df = pd.merge(tr_variavel_df_carregado, df_extracted, on='item', how='outer',
suffixes=('_x', '_y'))

#     for column in merged_df.columns:
#         if column.endswith('_y'):
#             col_x = column[:-2] + '_x'
#             if col_x in merged_df.columns:
#                 merged_df[col_x] = merged_df[col_x].combine_first(merged_df[column])
#                 merged_df.drop(columns=[column], inplace=True)
#                 merged_df.rename(columns={col_x: col_x[:-2]}, inplace=True)

#     # Reordenando as colunas
#     column_order = ['grupo', 'item', 'catalogo', 'descricao', 'unidade', 'quantidade',
'valor_estimado',
#                     'valor_homologado_item_unitario', 'percentual_desconto',
'valor_estimado_total_do_item', 'valor_homologado_total_item',
#                     'marca_fabricante', 'modelo_versao', 'situacao', 'descricao_detalhada',
'uasg', 'orgao_responsavel', 'num_pregao', 'ano_pregao',
#                     'srp', 'objeto', 'melhor_lance', 'valor_negociado', 'ordenador_despesa',
'empresa', 'cnpj',
#                     ]
#     merged_df = merged_df.reindex(columns=column_order)

#     if existing_dataframe is not None:
#         final_df = pd.concat([existing_dataframe,
merged_df]).drop_duplicates(subset='item').reset_index(drop=True)
#     else:
#         final_df = merged_df

#     return final_df
# else:
#     QMessageBox.warning(None, "Aviso", "Nenhum DataFrame de termo de referência carregado.")
#     return None

```