Arquivo: sicaf.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import pandas as pd
from modules.atas_api.widgets.worker_homologacao import TreeViewWindow, WorkerSICAF
import logging
from modules.utils.add_button import add_button_func_vermelho, add_button_copy, add_button_func
from modules.utils.linha_layout import linha_divisoria_layout

class RegistroSICAFDialog(QWidget):
    def __init__(self, pdf_dir, model, icons, database_ata_manager, main_window, parent=None):
        super().__init__(parent)
        self.sicaf_dir = pdf_dir / 'pasta_sicaf'
        self.model = model
        self.icons = icons
        self.database_ata_manager = database_ata_manager
        self.main_window = main_window
        self.homologacao_dataframe = None
        self.setWindowTitle("Registro SICAF")
        self.setup_ui()

    def setup_ui(self):
        """Configura a interface do diálogo."""
        main_layout = QVBoxLayout(self)

        # Carregar os dados iniciais, se disponíveis
        if self.homologacao_dataframe is None:
            self.homologacao_dataframe = pd.DataFrame()  # Inicializa com um DataFrame vazio

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

        # Criação do ComboBox para seleção de tabelas
        selecao_layout = QHBoxLayout()

        spacer = QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum)
        selecao_layout.addItem(spacer)

        selecao_label = QLabel("Selecione a Licitação:")
        selecao_label.setFont(QFont('Arial', 14))
        selecao_layout.addWidget(selecao_label)

        self.selecao_combobox = QComboBox()
        self.selecao_combobox.setFixedWidth(360)
        self.selecao_combobox.setFont(QFont('Arial', 12))
        selecao_layout.addWidget(self.selecao_combobox)

        selecao_layout.addStretch()
```

```
        main_layout.addLayout(selecao_layout)

        self.carregar_tabelas_result()

        linha_divisoria1, spacer_baixo_linha1 = linha_divisoria_layout()
        main_layout.addWidget(linha_divisoria1)
        main_layout.addSpacerItem(spacer_baixo_linha1)

        layout_conteudo = QHBoxLayout()

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

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

        main_layout.addLayout(layout_conteudo)

        linha_divisoria2, spacer_baixo_linha2 = linha_divisoria_layout()
        main_layout.addWidget(linha_divisoria2)
        main_layout.addSpacerItem(spacer_baixo_linha2)

        button_layout = QHBoxLayout()
        button_layout.addStretch()
              add_button_func_vermelho("Iniciar  Processamento",  self.iniciar_processamento_sicaf,
button_layout, "Iniciar processamento do SICAF", button_size=(300, 40))
        button_layout.addStretch()
        # Adiciona o layout de botão centralizado ao layout principal
        main_layout.addLayout(button_layout)

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

    def open_results_treeview(self):
        if self.homologacao_dataframe is not None and not self.homologacao_dataframe.empty:
            # Passa o database_ata_manager ao TreeViewWindow
            tree_view_window = TreeViewWindow(
                dataframe=self.homologacao_dataframe,
                icons_dir=self.icons,       # Diretório de ícones
                database_ata_manager=self.database_ata_manager,  # Gerenciador do 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 carregar_tabelas_result(self):
        # Obtém tabelas cujo nome começa com "result"
        tabelas_result = self.database_ata_manager.get_tables_with_keyword("result")
```

```
    print(f"Tabelas encontradas: {tabelas_result}")

    # Limpa o ComboBox antes de adicionar novos itens
    self.selecao_combobox.clear()

    # Adiciona a opção padrão "Selecione a Tabela"
    self.selecao_combobox.addItem("Selecione a Tabela")
    self.selecao_combobox.setCurrentIndex(0)  # Define como a opção inicial

    # Adiciona as tabelas encontradas
    for tabela in tabelas_result:
        if tabela.startswith("result"):
            self.selecao_combobox.addItem(tabela)

    # Conecta o evento de alteração de índice à função de atualização
    self.selecao_combobox.currentIndexChanged.connect(self.atualizar_dataframe_selecionado)

def atualizar_dataframe_selecionado(self):
    tabela = self.selecao_combobox.currentText()
    if tabela == "Selecione a Tabela":
        print("Nenhuma tabela selecionada.")
        return

    print(f"Tabela selecionada no ComboBox: {tabela}")
    if tabela:
        self.homologacao_dataframe = self.database_ata_manager.load_table_to_dataframe(tabela)
        print(f"DataFrame carregado da tabela '{tabela}':")
        print(self.homologacao_dataframe.head())

        if 'situacao' not in self.homologacao_dataframe.columns:
            print("Coluna 'situacao' ausente. Adicionando...")
            self.homologacao_dataframe['situacao'] = None

        self.atualizar_lista()
    else:
        self.homologacao_dataframe = pd.DataFrame()

def criar_layout_esquerdo(self):
    left_layout = QVBoxLayout()

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

    linha_divisoria2, spacer_baixo_linha2 = linha_divisoria_layout()
    left_layout.addWidget(linha_divisoria2)
    left_layout.addSpacerItem(spacer_baixo_linha2)

    # 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)

        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

        # 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.start()

        # Atualiza a lista para refletir alterações imediatamente após iniciar o processamento
        self.atualizar_lista()

    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.")

        self.atualizar_lista()

        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 = QVBoxLayout()

        legenda_hlayout = QHBoxLayout()
        legenda_hlayout.setAlignment(Qt.AlignmentFlag.AlignLeft)

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

        # Ícone de confirmação
        confirm_icon = QLabel()
        confirm_icon.setPixmap(self.icons["check"].pixmap(16, 16))
        confirm_text = QLabel("SICAF encontrado")

        # Ícone de cancelamento
        cancel_icon = QLabel()
        cancel_icon.setPixmap(self.icons["cancel"].pixmap(16, 16))
        cancel_text = QLabel("SICAF não encontrado")

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

        legenda_layout.addLayout(legenda_hlayout)

        copy_hlayout = QHBoxLayout()
        copy_hlayout.setAlignment(Qt.AlignmentFlag.AlignLeft)

        copiar_text = QLabel("Clique no botão")
        copy_icon = QLabel()
        copy_icon.setPixmap(self.icons["copy"].pixmap(22, 22))
         copiar_text2 = QLabel("para copiar o CNPJ para área de transferência e facilitar a busca
do SICAF.")

        copy_hlayout.addWidget(copiar_text)
        copy_hlayout.addWidget(copy_icon)
        copy_hlayout.addWidget(copiar_text2)
        legenda_layout.addLayout(copy_hlayout)

         credenciais_text = QLabel("Consulte apenas o 'Nível 1 - Credenciamento' do SICAF e insira
o PDF na pasta de processamento")

        legenda_layout.addWidget(credenciais_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)
right_layout.setSpacing(1)
right_layout.setContentsMargins(0, 2, 0, 2)
right_layout.setAlignment(Qt.AlignmentFlag.AlignLeft)

# Adiciona os botões "Abrir Pasta" e "Atualizar" ao layout
top_right_layout = self.criar_botoes_direitos()
right_layout.addLayout(top_right_layout)
right_layout.setSpacing(5)
# Define o QListWidget para a lista de arquivos PDF
self.pdf_list_widget = QListWidget()
# Aplica o estilo ao QListWidget
list_widget_style = """
QListWidget {
    color: #333333; /* Texto escuro para contraste */
    background-color: #F3F3F3; /* Cor de fundo padrão */
    border: 1px solid #CCCCCC; /* Borda clara para harmonizar com o fundo */
    padding: 5px;
    border-radius: 5px;
}

QListWidget::item {
    padding: 8px;
    border: 1px solid transparent; /* Borda invisível por padrão */
    border-radius: 3px;
}

QListWidget::item:selected {
    background-color: #E0E0E0; /* Cor de fundo para seleção */
    border: 1px solid #A0A0A0; /* Borda para seleção */
    color: #000000; /* Texto preto para melhor legibilidade */
}

QListWidget::item:hover {
    background-color: #D6D6D6; /* Cor de fundo ao passar o mouse */
    border: 1px solid #AAAAAA; /* Borda ao passar o mouse */
}

"""
self.pdf_list_widget.setStyleSheet(list_widget_style)

self.load_pdf_files()
right_layout.addWidget(self.pdf_list_widget)
layout_contador_pdf = QHBoxLayout()
layout_contador_pdf.addStretch()
# Conta e exibe a quantidade de arquivos PDF
self.right_label = QLabel(self.obter_texto_arquivos_pdf())
self.right_label.setFont(QFont('Arial', 12, QFont.Weight.Bold))
layout_contador_pdf.addWidget(self.right_label)
layout_contador_pdf.addStretch()
right_layout.addLayout(layout_contador_pdf)
```

```
        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"
          add_button_func("Abrir Pasta", "open-folder", self.abrir_pasta_sicaf, top_right_layout,
self.icons, "Clique para abrir a pasta de processamento do SICAF",  button_size=(150, 30))

        # Botão "Atualizar"
            add_button_func("Atualizar", "loading-arrow", self.atualizar_lista, top_right_layout,
self.icons, "Clique para atualizar os arquivos PDF",  button_size=(150, 30))

        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_ata_manager.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

        # Garante que o DataFrame contém as colunas necessárias
        if self.homologacao_dataframe is None or self.homologacao_dataframe.empty:
            self.homologacao_dataframe = pd.DataFrame(columns=["empresa", "cnpj"])
        else:
            # Adiciona colunas faltantes, se necessário
            for col in ["empresa", "cnpj"]:
                if col not in self.homologacao_dataframe.columns:
                    self.homologacao_dataframe[col] = None

        # 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_ata_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_ata_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.setSpacing(1)
            item_layout.setContentsMargins(0, 2, 0, 2)
```

```
            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(16, 16))

            empresa_label = QLabel(f"{cnpj} - {empresa}")
            copiar_button = add_button_copy(
                text="",  # Sem texto
                icon_name="copy",  # Nome do ícone no dicionário de ícones
                slot=lambda _, cnpj=cnpj: self.copiar_para_area_de_transferencia(cnpj),
                layout=item_layout,  # Layout onde o botão será inserido
                icons=self.icons,
                tooltip="Copiar para área de transferência"
            )

            # Adicionar ícone, botão e label ao layout do item
            item_layout.addWidget(icon_label)
            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 e se a coluna endereco possui um valor
válido.
        Retorna o ícone correspondente em cache.
        """
        try:
            # Consulta para verificar se o CNPJ existe e o endereço não é nulo ou vazio
            query = "SELECT endereco FROM registro_sicaf WHERE cnpj = ?"
            result = self.database_ata_manager.execute_query(query, (cnpj,))

            # Verifica o resultado da consulta
            if result:
                endereco = result[0][0]  # Supondo que o método execute_query retorna uma lista de
tuplas
                if endereco and endereco.strip():  # Confirma que 'endereco' não é vazio ou apenas
espaços em branco
                    return self.icons["check"]

            # Retorna o ícone 'cancel' se o endereço não for válido ou o CNPJ não existir
            return self.icons["cancel"]
        except Exception as e:
               QMessageBox.critical(self, "Erro no Banco de Dados", f"Erro ao acessar o banco de
dados: {e}")
            return self.icons["cancel"]  # Ícone padrão em caso de erro


    def load_icons(self):
        """Carrega ícones e os armazena em cache."""
        icons = {}
        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()
            icons[key] = icon

        return icons

    def copiar_para_area_de_transferencia(self, cnpj):
        """
        Copia o CNPJ para a área de transferência e exibe uma mensagem temporária de confirmação.
        """
        clipboard = QApplication.clipboard()
        clipboard.setText(cnpj)

        # Criação da mensagem de confirmação
          confirmation_label = QLabel(f"O CNPJ {cnpj} foi copiado para a área de transferência.",
self)
```

```
confirmation_label.setStyleSheet("""
    QLabel {
        background-color: #009B3A;
        color: white;
        font-size: 16px;
        border-radius: 5px;
        padding: 10px;
    }
""")
confirmation_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
confirmation_label.setFixedSize(550, 50)
confirmation_label.move(
    self.width() // 2 - confirmation_label.width() // 2,
    self.height() // 2 - confirmation_label.height() // 2
)
confirmation_label.show()

# Fecha a mensagem automaticamente após 700 ms
QTimer.singleShot(700, confirmation_label.close)
```