Arquivo: formulario.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from pathlib import Path
import pandas as pd
import os
import subprocess
from pathlib import Path
from openpyxl import Workbook, load_workbook
from openpyxl.styles import Font, Border, Side, PatternFill, Alignment
from openpyxl.utils import get_column_letter
from modules.utils.add_button import add_button, add_button_func


class FormularioExcel(QObject):
    formulario_carregado = pyqtSignal(pd.DataFrame)  # Sinal para enviar os dados carregados

    def __init__(self, df_registro_selecionado, pasta_base, parent_dialog):
        super().__init__()  # Inicializa o QObject
                    self.df_registro_selecionado   =   pd.DataFrame([df_registro_selecionado])   if
isinstance(df_registro_selecionado, dict) else df_registro_selecionado
        self.pasta_base = Path(pasta_base)
        self.parent_dialog = parent_dialog
        self.icons = parent_dialog.icons

        self.colunas_legiveis = {
            'nup': 'NUP',
            'material_servico': 'Material (M) ou Serviço (S)',
            'vigencia': 'Vigência (Ex: 2 (dois) meses, 6 (seis) meses), etc.',
            'criterio_julgamento': 'Critério de Julgamento (Menor Preço ou Maior Desconto)',
            'com_disputa': 'Com disputa? Sim (S) ou Não (N)',
            'pesquisa_preco': 'Pesquisa Concomitante? Sim (S) ou Não (N)',
            'sigla_om': 'OM',
            'setor_responsavel': 'Setor Responsável',
            'cod_par': 'Código PAR',
            'prioridade_par': 'Prioridade PAR (Necessário, Urgente ou Desejável)',
            'cep': 'CEP',
            'endereco': 'Endereço',
            'email': 'Email',
            'telefone': 'Telefone',
            'dias_para_recebimento': 'Dias para Recebimento',
            'horario_para_recebimento': 'Horário para Recebimento',
            'valor_total': 'Valor Total',
            'acao_interna': 'Ação Interna',
            'fonte_recursos': 'Fonte de Recursos',
            'natureza_despesa': 'Natureza da Despesa',
            'unidade_orcamentaria': 'Unidade Orçamentária',
            'ptres': 'PTRES',
            'atividade_custeio': 'Atividade de Custeio',
            'justificativa': 'Justificativa',
```

```
    'comunicacao_padronizada': 'Comunicação Padronizada (CP), Ex: 60-25',
}

self.normalizacao_valores = {
    'material_servico': {
        'M': 'Material',
        'm': 'Material',
        'Material': 'Material',
        'material': 'Material',
        'S': 'Serviço',
        's': 'Serviço',
        'Serviço': 'Serviço',
        'serviço': 'Serviço',
        'Servico': 'Serviço',
        'servico': 'Serviço'
    },
    'com_disputa': {
        'S': 'Sim',
        's': 'Sim',
        'Sim': 'Sim',
        'sim': 'Sim',
        'N': 'Não',
        'n': 'Não',
        'Não': 'Não',
        'não': 'Não',
        'Nao': 'Não',
        'nao': 'Não'
    },
    'pesquisa_preco': {
        'S': 'Sim',
        's': 'Sim',
        'Sim': 'Sim',
        'sim': 'Sim',
        'N': 'Não',
        'n': 'Não',
        'Não': 'Não',
        'não': 'Não',
        'Nao': 'Não',
        'nao': 'Não'
    },
    'atividade_custeio': {
        'S': 'Sim',
        's': 'Sim',
        'Sim': 'Sim',
        'sim': 'Sim',
        'N': 'Não',
        'n': 'Não',
        'Não': 'Não',
        'não': 'Não',
        'Nao': 'Não',
        'nao': 'Não'
    }
}
```

```
        self.colunas_legiveis_inverso = {v: k for k, v in self.colunas_legiveis.items()}

    def setup_formularios(self):
        """Configura o layout para consulta à API com campos de CNPJ e Sequencial PNCP."""
        group_box = QGroupBox("Formulário", self.parent_dialog)
        layout = QVBoxLayout(group_box)

        # Aplicando o estilo CSS específico ao GroupBox
        group_box.setStyleSheet("""
            QGroupBox {
                border: 1px solid #3C3C5A;
                border-radius: 10px;
                font-size: 16px;
                font-weight: bold;
                color: white;
                margin-top: 13px;
            }
            QGroupBox:title {
                subcontrol-origin: margin;
                padding: 0 3px;
            }
        """)

        # Layout vertical para os botões
        vlayout_botoes = QVBoxLayout()

        # Conectar o botão `Novo Formulário` à função `criar_formulario` usando uma função lambda
          add_button_func("    Novo Formulário    ", "excel_down", lambda: self.criar_formulario(),
vlayout_botoes, self.icons, tooltip="Gerar o Formulário")
                 add_button_func("Importar  Formulário",  "excel_up",  self.carregar_formulario(),
vlayout_botoes, self.icons, tooltip="Carregar o Formulário")

        # Adicionar o layout de botões ao layout principal
        layout.addLayout(vlayout_botoes)
        return group_box

    def criar_formulario(self):
        try:
            wb = Workbook()
            ws = wb.active
            ws.title = "Formulário"

            # Filtra o DataFrame e preenche o conteúdo da planilha
            df_filtrado = self._filtrar_dataframe()
            self._adicionar_titulo(ws)
            self._definir_cabecalhos(ws)
            self._preencher_dados(ws, df_filtrado)
            self._aplicar_bordas(ws)

            # Salva e abre o arquivo criado
            file_path = self._salvar_arquivo(wb)
```

```
            self._abrir_arquivo(file_path)

            QMessageBox.information(None, "Sucesso", "Formulário criado e aberto com sucesso.")
        except Exception as e:
            print(f"Erro ao criar formulário: {str(e)}")
            QMessageBox.critical(None, "Erro", f"Falha ao criar formulário: {str(e)}")

    def carregar_formulario(self):
        try:
            # Adicione uma mensagem de depuração para verificar se o método está sendo executado
            print("Abrindo o diálogo de seleção de arquivo")

            # Remova o parâmetro `None` para abrir o diálogo sem definir um parent específico
                    file_path,  _  =  QFileDialog.getOpenFileName(caption="Selecione  o  formulário",
filter="Excel Files (*.xlsx *.ods);;All Files (*)")
            if not file_path:
                print("Nenhum arquivo foi selecionado.")
                return

            print(f"Arquivo selecionado: {file_path}")

            if file_path.endswith('.xlsx'):
                wb = load_workbook(file_path)
                ws = wb.active

                if ws['A2'].value != "Índice" or ws['B2'].value != "Valor":
                    QMessageBox.critical(None, "Erro", "O formulário selecionado está incorreto.")
                    return

                for row in ws.iter_rows(min_row=3, max_col=2, values_only=True):
                    coluna_legivel = row[0]
                    valor = row[1]
                    coluna = self.colunas_legiveis_inverso.get(coluna_legivel, coluna_legivel)
                    if coluna in self.normalizacao_valores:
                        valor = self.normalizacao_valores[coluna].get(valor, valor)
                    if coluna in self.df_registro_selecionado.columns:
                        self.df_registro_selecionado.at[0, coluna] = valor

            elif file_path.endswith('.ods'):
                df = pd.read_excel(file_path, engine='odf')

                if df.iloc[0, 0] != "Índice" or df.iloc[0, 1] != "Valor":
                    QMessageBox.critical(None, "Erro", "O formulário selecionado está incorreto.")
                    return

                for _, row in df.iloc[1:].iterrows():
                    coluna_legivel = row[0]
                    valor = row[1]
                    coluna = self.colunas_legiveis_inverso.get(coluna_legivel, coluna_legivel)
                    if coluna in self.normalizacao_valores:
                        valor = self.normalizacao_valores[coluna].get(valor, valor)
                    if coluna in self.df_registro_selecionado.columns:
                        self.df_registro_selecionado.at[0, coluna] = valor
```

```
            # Emite o sinal com o DataFrame atualizado
            self.formulario_carregado.emit(self.df_registro_selecionado)
            # Adicione o print para exibir o DataFrame carregado
            print("DataFrame carregado:")
            print(self.df_registro_selecionado)

            QMessageBox.information(None, "Sucesso", "Formulário carregado com sucesso.")
        except Exception as e:
            print(f"Erro ao carregar formulário: {str(e)}")
            QMessageBox.critical(None, "Erro", f"Falha ao carregar formulário: {str(e)}")

    def _filtrar_dataframe(self):
        colunas_incluir = list(self.colunas_legiveis.keys())
                    colunas_disponiveis = [col for col in colunas_incluir if col in
self.df_registro_selecionado.columns]
                                                                                df_filtrado =
self.df_registro_selecionado[colunas_disponiveis].rename(columns=self.colunas_legiveis)
        return df_filtrado

    def _adicionar_titulo(self, ws):
        tipo = self.df_registro_selecionado['tipo'].iloc[0]
        numero = self.df_registro_selecionado['numero'].iloc[0]
        ano = self.df_registro_selecionado['ano'].iloc[0]
        objeto = self.df_registro_selecionado['objeto'].iloc[0]
        titulo = f"{tipo} nº {numero}/{ano} ({objeto})"
        ws.merge_cells('A1:B1')
        ws['A1'] = titulo
        ws['A1'].font = Font(size=20, bold=True)
        ws['A1'].alignment = Alignment(horizontal='center', vertical='center')
        ws.row_dimensions[1].height = 40

    def _definir_cabecalhos(self, ws):
        ws['A2'] = "Índice"
        ws['B2'] = "Valor"
        ws['A2'].font = Font(size=14, bold=True)
        ws['B2'].font = Font(size=14, bold=True)
        ws['A2'].alignment = Alignment(horizontal='center', vertical='center')
        ws['B2'].alignment = Alignment(horizontal='center', vertical='center')
                    thin_border = Border(left=Side(style='thin'), right=Side(style='thin'),
top=Side(style='thin'), bottom=Side(style='thin'))
        ws['A2'].border = thin_border
        ws['B2'].border = thin_border
        ws.column_dimensions[get_column_letter(1)].width = 50
        ws.column_dimensions[get_column_letter(2)].width = 80

    def _preencher_dados(self, ws, df_filtrado):
        for i, (col_name, value) in enumerate(df_filtrado.iloc[0].items(), start=3):
            ws[f'A{i}'] = col_name
            ws[f'B{i}'] = str(value)
            ws[f'B{i}'].alignment = Alignment(wrap_text=True)
            fill_color = "F2F2F2" if i % 2 == 0 else "FFFFFF"
            fill = PatternFill(start_color=fill_color, end_color=fill_color, fill_type="solid")
```

```
            ws[f'A{i}'].fill = fill
            ws[f'B{i}'].fill = fill
                      ws[f'A{i}'].alignment  =  Alignment(horizontal='center',  vertical='center',
wrap_text=True)
            ws.row_dimensions[i].height = 60 if i == 28 else 15

    def _aplicar_bordas(self, ws):
                        thin_border   =   Border(left=Side(style='thin'),   right=Side(style='thin'),
top=Side(style='thin'), bottom=Side(style='thin'))
        for row in ws.iter_rows(min_row=1, max_row=ws.max_row, min_col=1, max_col=2):
            for cell in row:
                cell.border = thin_border

    def _salvar_arquivo(self, wb):
        file_path = self.pasta_base / "formulario.xlsx"
        wb.save(file_path)
        return file_path

    def _abrir_arquivo(self, file_path):
        if os.name == 'nt':
            os.startfile(file_path)
        elif os.name == 'posix':
            subprocess.call(['open', file_path])
        else:
            subprocess.call(['xdg-open', file_path])


class TableCreationWorker(QThread):
    # Sinal para indicar quando o arquivo foi salvo
    file_saved = pyqtSignal(str)

    def __init__(self, dados, colunas_legiveis, pasta_base):
        super().__init__()
        self.dados = dados
        self.colunas_legiveis = colunas_legiveis
        self.pasta_base = pasta_base

    def run(self):
        # Criando e salvando a tabela
        wb = Workbook()
        ws = wb.active
        ws.title = "Formulário"

        # --- NOVO: Define o estilo para células obrigatórias ---
                      obrigatorio_fill   =   PatternFill(start_color="FFFFCC",   end_color="FFFFCC",
fill_type="solid")
        obrigatorio_font = Font(color="FF0000", bold=True)
        # --- FIM NOVO ---

        # Adicionando título
        tipo = self.dados.get('tipo', '')
        numero = self.dados.get('numero', '')
        ano = self.dados.get('ano', '')
```

```
        objeto = self.dados.get('objeto', '')
        titulo = f"{tipo} nº {numero}/{ano} ({objeto})"
        ws.merge_cells('A1:B1')
        ws['A1'] = titulo
        ws['A1'].font = Font(size=20, bold=True)
        ws['A1'].alignment = Alignment(horizontal='center', vertical='center')
        ws.row_dimensions[1].height = 40

        # Cabeçalhos
        ws['A2'] = "Índice"
        ws['B2'] = "Valor"
        ws['A2'].font = Font(size=14, bold=True)
        ws['B2'].font = Font(size=14, bold=True)
        ws['A2'].alignment = Alignment(horizontal='center', vertical='center')
        ws['B2'].alignment = Alignment(horizontal='center', vertical='center')
                    thin_border   =   Border(left=Side(style='thin'),   right=Side(style='thin'),
top=Side(style='thin'), bottom=Side(style='thin'))
        ws['A2'].border = thin_border
        ws['B2'].border = thin_border

        # Ajuste de largura das colunas
        ws.column_dimensions['A'].width = 50
        ws.column_dimensions['B'].width = 80

        # Preenchendo dados filtrados
        for i, (key, label) in enumerate(self.colunas_legiveis.items(), start=3):
            value = self.dados.get(key, '')  # Obtém o valor correspondente em `self.dados`

            # --- LÓGICA ATUALIZADA ---
            cell_a = ws[f'A{i}']
            cell_b = ws[f'B{i}']

            cell_a.value = label

            # Se o valor for vazio ou None, usa a mensagem padrão
            if not value:
                cell_b.value = "LOCAL DE PREENCHIMENTO OBRIGATORIO"
                cell_b.fill = obrigatorio_fill
                cell_b.font = obrigatorio_font
            else:
                cell_b.value = str(value)

            # Aplica estilos gerais
            cell_b.alignment = Alignment(wrap_text=True)
            fill_color = "F2F2F2" if i % 2 == 0 else "FFFFFF"
            fill = PatternFill(start_color=fill_color, end_color=fill_color, fill_type="solid")
            cell_a.fill = fill
            # Só aplica o preenchimento cinza se não for um campo obrigatório
            if not cell_b.font.color or cell_b.font.color.rgb != "FFFF0000":
                 cell_b.fill = fill

            cell_a.alignment = Alignment(horizontal='center', vertical='center', wrap_text=True)
            ws.row_dimensions[i].height = 15
```

```
    for cell in [cell_a, cell_b]:
        cell.border = thin_border
    # --- FIM DA LÓGICA ATUALIZADA ---

# Salvando arquivo e emitindo o sinal com o caminho
file_path = self.pasta_base / "formulario.xlsx"
wb.save(file_path)
self.file_saved.emit(str(file_path))  # Emite o sinal com o caminho do arquivo salvo
```