Arquivo: config_widget.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from modules.utils.add_button import add_button_func
from modules.utils.linha_layout import linha_divisoria_sem_spacer_layout
from .config_Responsaveis.edit_responsaveis import EditPredefinicoesDialog
from paths import *
import json
from functools import partial
from .config_OM.edit_OM import show_organizacoes_widget
from .config_Setores.edit_Setores import show_setores_widget

class ConfigManager(QWidget):
    def __init__(self, icons, parent=None):
        super().__init__(parent)
        self.icons = icons
        self.setup_ui()
        self.current_content_layout = None
        # self.contratos_manager = ContratosManager(self.icons, dados={})

    def setup_ui(self):
        """Configura o layout de configuração com um menu lateral."""
        main_layout = QHBoxLayout(self)
        main_layout.setSpacing(0)
        main_layout.setContentsMargins(0, 0, 0, 0)

        # Menu lateral
        menu_layout = QVBoxLayout()
        menu_layout.setContentsMargins(0, 0, 0, 0)
        menu_layout.setSpacing(0)

        # Botões do menu lateral
        button_style = """
            QPushButton {
                border: none;
                color: white;
                font-size: 16px;
                text-align: center;
                padding: 15px;
            }
            QPushButton:hover {
                background-color: #3A3B47;
            }
            QPushButton:checked {
                background-color: #202124;
                font-weight: bold;
            }
        """
        buttons = [
            ("Agentes Responsáveis", self.show_agentes_responsaveis_widget),
```

```
        ("Organizações Militares", self.show_organizacoes_widget),
        # ("Setores Requisitantes", self.show_setores_requisitantes_widget),
        # ("Database", self.gerenciar_inclusao_exclusao_contratos_widget),
    ]

    self.config_menu_buttons = []

    for text, callback in buttons:
        button = QPushButton(text)
        button.setCheckable(True)
        button.setStyleSheet(button_style)
        button.setCursor(Qt.CursorShape.PointingHandCursor)
        button.clicked.connect(callback)
        button.clicked.connect(lambda _, b=button: self.set_selected_button(b))
        menu_layout.addWidget(button)
        self.config_menu_buttons.append(button)

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

    menu_widget = QWidget()
    menu_widget.setLayout(menu_layout)
    menu_widget.setStyleSheet("background-color: #181928;")
    main_layout.addWidget(menu_widget, stretch=1)

    # Área de conteúdo
    self.content_widget = QWidget()
    self.content_layout = QVBoxLayout(self.content_widget)

    main_layout.addWidget(self.content_widget, stretch=4)

def set_selected_button(self, selected_button):
    """Define o botão selecionado no menu lateral."""
    for button in self.config_menu_buttons:
        button.setChecked(False)
    selected_button.setChecked(True)

def clear_content(self):
    """Limpa o layout atual do content_widget."""
    while self.content_layout.count():
        item = self.content_layout.takeAt(0)
        widget = item.widget()
        if widget:
            widget.deleteLater()
        elif item.layout():
            self.clear_layout(item.layout())

def clear_layout(self, layout):
    """Recursivamente limpa um layout."""
    while layout.count():
        item = layout.takeAt(0)
        widget = item.widget()
        if widget:
```

```
            widget.deleteLater()
        elif item.layout():
            self.clear_layout(item.layout())

def show_agentes_responsaveis_widget(self):
    """Exibe o widget para Alteração dos Agentes Responsáveis."""
    self.clear_content()

    # Scroll Area
    scroll_area = QScrollArea()
    scroll_area.setWidgetResizable(True)

    # Widget principal para o conteúdo da scroll area
    scroll_widget = QWidget()
    layout = QVBoxLayout(scroll_widget)

    # Título
    title = QLabel("Alteração dos Agentes Responsáveis")
    title.setStyleSheet("font-size: 20px; font-weight: bold; color: #4E648B")
    layout.addWidget(title)

    # Carregar dados do arquivo JSON
    try:
        with open(AGENTES_RESPONSAVEIS_FILE, 'r', encoding='utf-8') as file:
            config_data = json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        config_data = {}

    # Lista de agentes
    agentes = [
        "Ordenador de Despesa",
        "Agente Fiscal",
        "Gerente de Crédito",
        "Responsável pela Demanda",
        "Operador da Contratação",
        "Pregoeiro",
    ]

    # Criando botões para cada agente
    for agente in agentes:
        categoria = agente.lower().replace(" ", "_")
        item_layout = QVBoxLayout()

        linha_divisoria = linha_divisoria_sem_spacer_layout()
        item_layout.addWidget(linha_divisoria)

        # Exibir valores existentes no JSON acima do botão
        if categoria in config_data:
            for item in config_data[categoria]:
                if isinstance(item, dict):  # Verifica se o item é um dicionário
                    item_label = QLabel(
                        f"{item.get('Nome', 'N/A')} - {item.get('Posto', 'N/A')} - "
                        f"{item.get('Abreviacao', 'N/A')} - {item.get('Funcao', 'N/A')}"
```

```
                    )
                else:
                    # Tratamento para itens inválidos no JSON
                    item_label = QLabel(f"Item inválido: {item}")
                item_label.setStyleSheet("font-size: 14px; color: #000;")
                item_layout.addWidget(item_label)

        # Layout horizontal para o botão com espaçadores laterais
        button_layout = QHBoxLayout()
        button_layout.addStretch()  # Espaçador à esquerda

        # Botão com texto do agente
        button = add_button_func(
            text=agente,
            icon_name="edit",  # Substituir pelo nome do ícone correto
            slot=partial(self.edit_agent, agente),  # Passa o nome do agente para a função
            layout=button_layout,
            icons=self.icons,
            tooltip=f"Editar {agente}",
            button_size=(300, 40)
        )

        button_layout.addStretch()  # Espaçador à direita
        item_layout.addLayout(button_layout)

        layout.addLayout(item_layout)

    # Adiciona espaçador para empurrar o conteúdo para o topo
    layout.addStretch()

    # Configura o widget no scroll area
    scroll_area.setWidget(scroll_widget)

    # Adiciona o scroll area ao layout principal
    self.content_layout.addWidget(scroll_area)

def edit_agent(self, agente):
    """Função chamada ao clicar em 'Editar'."""
    categoria = agente.lower().replace(" ", "_")  # Transformar em formato adequado para JSON
    try:
        # Carregar os dados do JSON
        if not AGENTES_RESPONSAVEIS_FILE.exists():
            with open(AGENTES_RESPONSAVEIS_FILE, 'w', encoding='utf-8') as file:
                json.dump({}, file, ensure_ascii=False, indent=4)

        with open(AGENTES_RESPONSAVEIS_FILE, 'r', encoding='utf-8') as file:
            config_data = json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        config_data = {}

    # Garantir que a categoria exista no dicionário
    if categoria not in config_data:
        config_data[categoria] = []
```

```
    # Abrir o diálogo de edição
    dialog = EditPredefinicoesDialog(categoria, config_data, self)
    if dialog.exec():
        # Salvar alterações no JSON
        with open(AGENTES_RESPONSAVEIS_FILE, 'w', encoding='utf-8') as file:
            json.dump(config_data, file, ensure_ascii=False, indent=4)

        # Atualizar o widget após salvar as alterações
        self.show_agentes_responsaveis_widget()

def show_organizacoes_widget(self):
    show_organizacoes_widget(self.content_layout, self.icons, self)

def show_setores_requisitantes_widget(self):
    show_setores_widget(self.content_layout, self.icons, self)
```