Arquivo: edit_responsaveis.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from paths import *
from modules.config.config_widget import linha_divisoria_sem_spacer_layout
import sqlite3
from pathlib import Path
import pandas as pd
import os
import json
from functools import partial

def show_agentes_responsaveis_widget(content_layout, icons, parent):
    """Exibe o widget para Alteração dos Agentes Responsáveis."""
    # Limpa o layout de conteúdo
    while content_layout.count():
        item = content_layout.takeAt(0)
        widget = item.widget()
        if widget:
            widget.deleteLater()
        elif item.layout():
            clear_layout(item.layout)

    def clear_layout(layout):
        """Recursivamente limpa um layout."""
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            if widget:
                widget.deleteLater()
            elif item.layout():
                clear_layout(item.layout)

    # Widget principal para o conteúdo
    main_widget = QWidget()
    layout = QVBoxLayout(main_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]:
                item_label = QLabel(
                    f"{item['Nome']} - {item['Posto']} - {item['Abreviacao']} - {item['Funcao']}"
                )
                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 = QPushButton(agente)
        button.setIcon(icons.get("edit"))  # Ajuste para o ícone correto
        button.setStyleSheet("font-size: 14px; padding: 5px;")
        button.clicked.connect(partial(edit_agent, parent, agente))  # Passa o nome do agente para
a função
        button_layout.addWidget(button)

        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 layout principal
    content_layout.addWidget(main_widget)

def edit_agent(parent, 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 Path(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, parent)
    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
        show_agentes_responsaveis_widget(parent.content_layout, parent.icons, parent)


class ComboBoxDelegate(QStyledItemDelegate):
    def __init__(self, options, parent=None):
        super().__init__(parent)
        self.options = options

    def createEditor(self, parent, option, index):
        combo_box = QComboBox(parent)
        combo_box.setEditable(True)
        combo_box.addItems(self.options)
        combo_box.setFont(QFont('Arial', 12))

        # Definir um QListView para o QComboBox para controlar o estilo dos itens da lista
        list_view = QListView()
        list_view.setFont(QFont('Arial', 12))
        combo_box.setView(list_view)

        # Definir a fonte do line_edit diretamente
        line_edit = combo_box.lineEdit()
        line_edit.setFont(QFont('Arial', 12))

        return combo_box

    def setEditorData(self, editor, index):
        value = index.model().data(index, Qt.ItemDataRole.EditRole)
        if value:
            editor.setCurrentText(value)  # Chame setCurrentText no QComboBox

    def setModelData(self, editor, model, index):
```

```
        value = editor.currentText()  # Obtenha o texto atual do QComboBox
        model.setData(index, value, Qt.ItemDataRole.EditRole)

    def updateEditorGeometry(self, editor, option, index):
        editor.setGeometry(option.rect)

class AgentesResponsaveisTableModel(QAbstractTableModel):
    def __init__(self, data, database_path):
        super().__init__()
        self._data = data
        self._headers = ["Nome", "Posto", "Função"]
        self.database_path = database_path

    def rowCount(self, index):
        return len(self._data)

    def columnCount(self, index):
        return len(self._headers)

    def data(self, index, role):
        if role == Qt.ItemDataRole.DisplayRole or role == Qt.ItemDataRole.EditRole:
            return self._data[index.row()][index.column()]

    def headerData(self, section, orientation, role):
        if role == Qt.ItemDataRole.DisplayRole:
            if orientation == Qt.Orientation.Horizontal:
                return self._headers[section]
            if orientation == Qt.Orientation.Vertical:
                return section + 1

    def setData(self, index, value, role):
        if role == Qt.ItemDataRole.EditRole:
            self._data[index.row()][index.column()] = value
            self.updateDatabase(index.row(), index.column(), value)
            self.dataChanged.emit(index, index, (Qt.ItemDataRole.EditRole,))
            return True
        return False

    def flags(self, index):
                        return   Qt.ItemFlag.ItemIsSelectable   |   Qt.ItemFlag.ItemIsEnabled   |
Qt.ItemFlag.ItemIsEditable

    def updateDatabase(self, row, column, value):
        try:
            with sqlite3.connect(self.database_path) as conn:
                cursor = conn.cursor()
                headers = ['nome', 'posto', 'funcao']
                   query = f"UPDATE controle_agentes_responsaveis SET {headers[column]} = ? WHERE
rowid = ?"
                cursor.execute(query, (value, row + 1))  # rowid é 1-indexado
                conn.commit()
        except Exception as e:
            QMessageBox.critical(None, "Erro", f"Erro ao atualizar o banco de dados: {e}")
```

```
    def addRow(self):
        self.beginInsertRows(QModelIndex(), self.rowCount(None), self.rowCount(None))
        self._data.append(["", "", ""])  # Adiciona uma linha vazia
        self.endInsertRows()

        try:
            with sqlite3.connect(self.database_path) as conn:
                cursor = conn.cursor()
                   cursor.execute("INSERT INTO controle_agentes_responsaveis (nome, posto, funcao)
VALUES (?, ?, ?)", ("", "", ""))
                conn.commit()
        except Exception as e:
            QMessageBox.critical(None, "Erro", f"Erro ao adicionar ao banco de dados: {e}")

    def removeRow(self, row, parent=QModelIndex()):
        self.beginRemoveRows(QModelIndex(), row, row)
        self._data.pop(row)
        self.endRemoveRows()

        try:
            with sqlite3.connect(self.database_path) as conn:
                cursor = conn.cursor()
                 cursor.execute("DELETE FROM controle_agentes_responsaveis WHERE rowid = ?", (row +
1,))
                conn.commit()
        except Exception as e:
            QMessageBox.critical(None, "Erro", f"Erro ao remover do banco de dados: {e}")

class EditPredefinicoesDialog(QDialog):
    def __init__(self, categoria, config_data, parent=None):
        super().__init__(parent)
        self.categoria = categoria
        self.config_data = config_data
        self.setWindowTitle(f"Editar {categoria.replace('_', ' ').capitalize()}")

        layout = QVBoxLayout(self)

        # Título
        title = QLabel(f"Edição de {categoria.replace('_', ' ').capitalize()}")
        title.setStyleSheet("font-size: 18px; font-weight: bold;")
        layout.addWidget(title)

        # Lista de itens
        self.list_widget = QListWidget()
        if categoria in config_data:
            for item in config_data[categoria]:
                if isinstance(item, dict):  # Garantir que o item é um dicionário
                          item_text = f"{item.get('Nome', 'N/A')} - {item.get('Posto', 'N/A')} -
{item.get('Abreviacao', 'N/A')} - {item.get('Funcao', 'N/A')}"
                    self.list_widget.addItem(item_text)
        layout.addWidget(self.list_widget)
```

```
# Nome
layout.addWidget(QLabel("Nome:"))
self.nome_input = QLineEdit()
self.nome_input.setPlaceholderText("Digite o nome")
self.nome_input.textChanged.connect(self.forcar_caixa_alta)
layout.addWidget(self.nome_input)

# Posto
layout.addWidget(QLabel("Posto:"))
self.posto_input = QComboBox()
self.posto_input.setEditable(True)
self.posto_input.currentTextChanged.connect(self.atualizar_abreviacao)
layout.addWidget(self.posto_input)

# Abreviação do Posto
layout.addWidget(QLabel("Abreviação do Posto:"))
self.abrev_posto_input = QComboBox()
self.abrev_posto_input.setEditable(True)
layout.addWidget(self.abrev_posto_input)

# Preencher o ComboBox de posto dinamicamente
self.atualizar_posto()

# Conectar a função de inicialização da abreviação
self.atualizar_abreviacao(self.posto_input.currentText())

# Função
layout.addWidget(QLabel("Função:"))
self.funcao_input = QComboBox()
self.funcao_input.setEditable(True)
layout.addWidget(self.funcao_input)

# Inicializa os valores da função com base na categoria
self.inicializar_funcoes()

# Botões para adicionar e remover itens
button_layout = QHBoxLayout()
add_btn = QPushButton("Adicionar")
add_btn.clicked.connect(self.adicionar_item)
button_layout.addWidget(add_btn)

remove_btn = QPushButton("Remover")
remove_btn.clicked.connect(self.remover_item)
button_layout.addWidget(remove_btn)

layout.addLayout(button_layout)

# Botão de salvar
save_btn = QPushButton("Salvar")
save_btn.clicked.connect(self.salvar_e_fechar)
layout.addWidget(save_btn)

# Conecta a seleção na lista para preencher os campos
```

```
        self.list_widget.itemClicked.connect(self.preencher_campos)

    def forcar_caixa_alta(self):
        """Garante que o nome seja sempre em caixa alta."""
        self.nome_input.setText(self.nome_input.text().upper())

    def atualizar_posto(self):
        """Atualiza as opções de posto com base na categoria."""
        postos_ordenador_agente_fiscal = [
            "Capitão de Mar e Guerra (IM)", "Capitão de Fragata (IM)", "Capitão de Corveta (IM)",
            "Capitão Tenente (IM)", "Outro"
        ]
        postos_geral = [
            "Primeiro-Tenente", "Segundo-Tenente", "Suboficial", "Primeiro-Sargento",
            "Segundo-Sargento", "Terceiro-Sargento", "Cabo", "Outro"
        ]
           postos = postos_ordenador_agente_fiscal if self.categoria in ["ordenador_de_despesa",
"agente_fiscal"] else postos_geral
        self.posto_input.clear()
        self.posto_input.addItems(postos)

    def atualizar_abreviacao(self, posto):
        """Atualiza as opções de abreviação de posto com base no posto selecionado e categoria."""
        abrev_map_ordenador_agente_fiscal = {
            "Capitão de Mar e Guerra (IM)": ["CMG (IM)", "Outro"],
            "Capitão de Fragata (IM)": ["CF (IM)", "Outro"],
            "Capitão de Corveta (IM)": ["CC (IM)", "Outro"],
            "Capitão Tenente (IM)": ["CT (IM)", "Outro"],
            "Outro": ["Outro"]
        }
        abrev_map_geral = {
            "Primeiro-Tenente": ["1ºTEN", "Outro"],
            "Segundo-Tenente": ["2ºTEN", "Outro"],
            "Suboficial": ["SO", "Outro"],
            "Primeiro-Sargento": ["1º SG", "Outro"],
            "Segundo-Sargento": ["2º SG", "Outro"],
            "Terceiro-Sargento": ["3º SG", "Outro"],
            "Cabo": ["CB", "Outro"],
            "Outro": ["Outro"]
        }
                        abrev_map   =   abrev_map_ordenador_agente_fiscal   if   self.categoria   in
["ordenador_de_despesa", "agente_fiscal"] else abrev_map_geral
        self.abrev_posto_input.clear()
        self.abrev_posto_input.addItems(abrev_map.get(posto, ["Outro"]))

    def inicializar_funcoes(self):
        """Define as funções disponíveis com base na categoria."""
        self.funcao_input.clear()
        if self.categoria == "ordenador_de_despesa":
                      self.funcao_input.addItems(["Ordenador  de  Despesa",  "Ordenador  de  Despesa
Substituto"])
        elif self.categoria == "agente_fiscal":
            self.funcao_input.addItems(["Agente Fiscal", "Agente Fiscal Substituto"])
```

```
        else:
            self.funcao_input.addItems([
                "Gerente de Crédito", "Responsável pela Demanda",
                "Operador da Contratação", "Pregoeiro"
            ])

    def preencher_campos(self, item):
        """Preenche os campos de edição com os valores do item selecionado."""
        partes = item.text().split(" - ")
        if len(partes) == 4:
            self.nome_input.setText(partes[0].strip())
            self.posto_input.setCurrentText(partes[1].strip())
            self.abrev_posto_input.setCurrentText(partes[2].strip())
            self.funcao_input.setCurrentText(partes[3].strip())

    def adicionar_item(self):
        """Adiciona um novo item à lista."""
        nome = self.nome_input.text().strip()
        posto = self.posto_input.currentText().strip()
        abreviacao = self.abrev_posto_input.currentText().strip()
        funcao = self.funcao_input.currentText().strip()

        if not nome or not posto or not abreviacao or not funcao:
            QMessageBox.warning(self, "Aviso", "Todos os campos devem ser preenchidos.")
            return

        item_text = f"{nome} - {posto} - {abreviacao} - {funcao}"
        self.list_widget.addItem(item_text)

        # Limpa os campos após adicionar
        self.nome_input.clear()
        self.posto_input.setCurrentIndex(-1)
        self.abrev_posto_input.setCurrentIndex(-1)
        self.funcao_input.setCurrentIndex(-1)

    def remover_item(self):
        """Remove o item selecionado da lista."""
        selected_item = self.list_widget.currentItem()
        if selected_item:
            self.list_widget.takeItem(self.list_widget.row(selected_item))

    def salvar_e_fechar(self):
        """Salva as alterações na configuração e fecha o diálogo."""
        items = []
        for i in range(self.list_widget.count()):
            partes = self.list_widget.item(i).text().split(" - ")
            if len(partes) == 4:
                items.append({
                    "Nome": partes[0].strip(),
                    "Posto": partes[1].strip(),
                    "Abreviacao": partes[2].strip(),
                    "Funcao": partes[3].strip()
                })
```

```
self.config_data[self.categoria] = items
self.accept()
```