Arquivo: add_item.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from pathlib import Path
from datetime import datetime
from modules.utils.select_om import load_sigla_om
import sqlite3
from paths import ORGANIZACOES_FILE
import json

class AddItemDialog(QDialog):
    def __init__(self, icons, database_path, controle_om, parent=None):
        super().__init__(parent)
        self.icons = icons
        self.database_path = database_path
        self.controle_om = controle_om
        self.om_details = {}  # Será populado pelo método load_om_data
        self.setWindowTitle("Adicionar Item")
        self.setWindowIcon(self.icons["plus"])

        self.layout = QVBoxLayout(self)
        self.setStyleSheet("QWidget { font-size: 14px; }")
        self.setup_ui()

        # Carrega os dados das OMs (siglas e detalhes)
        self.load_om_data(ORGANIZACOES_FILE)

        self.load_next_numero()

    # Adicione este novo método inteiro dentro da classe AddItemDialog
    def load_om_data(self, json_path):
        """Lê o JSON das organizações, popula o combobox e o dicionário de detalhes."""
        try:
            with open(json_path, 'r', encoding='utf-8') as f:
                data = json.load(f)

            siglas = []
            # Itera sobre a lista de organizações no JSON
            for org in data.get('organizacoes', []):
                sigla = org.get('Sigla')
                if sigla:
                    siglas.append(sigla)
                    # Popula o dicionário self.om_details com os detalhes de cada OM
                    self.om_details[sigla] = {
                        'orgao_responsavel': org.get('Nome'),
                        'uasg': org.get('UASG')
                    }

            # Limpa e preenche o combobox com as siglas carregadas
            self.sigla_om_cb.clear()
```

```
            self.sigla_om_cb.addItems(sorted(siglas)) # Ordena as siglas em ordem alfabética

        except (FileNotFoundError, json.JSONDecodeError) as e:
            print(f"ERRO: Não foi possível carregar o arquivo de Organizações Militares: {e}")
             QMessageBox.warning(self, "Erro de Arquivo", "Não foi possível carregar o arquivo de
Organizações Militares (organizacoes.json).")

    def setup_ui(self):
        self.tipo_cb, self.numero_le, self.ano_le = self.setup_first_line()
        self.objeto_le = self.setup_third_line()
        self.nup_le, self.sigla_om_cb = self.setup_fourth_line()
        self.material_radio, self.servico_radio = self.setup_fifth_line()
        self.setup_save_button()

    def setup_first_line(self):
        hlayout = QHBoxLayout()
        tipo_cb = QComboBox()
        numero_le = QLineEdit()
        ano_le = QLineEdit()

             [tipo_cb.addItem(option[0]) for option in [("Dispensa Eletrônica (DE)", "Dispensa
Eletrônica")]]
        tipo_cb.setCurrentText("Dispensa Eletrônica (DE)")
        numero_le.setValidator(QIntValidator(1, 99999))
        ano_le.setValidator(QIntValidator(1000, 9999))
        ano_le.setText(str(datetime.now().year))

        hlayout.addWidget(QLabel("Tipo:"))
        hlayout.addWidget(tipo_cb)
        hlayout.addWidget(QLabel("Número:"))
        hlayout.addWidget(numero_le)
        hlayout.addWidget(QLabel("Ano:"))
        hlayout.addWidget(ano_le)
        self.layout.addLayout(hlayout)

        return tipo_cb, numero_le, ano_le

    def setup_third_line(self):
        hlayout = QHBoxLayout()
        objeto_le = QLineEdit()
            objeto_le.setPlaceholderText("Exemplo: 'Material de Limpeza' (Utilizar no máximo 3
palavras)")
        hlayout.addWidget(QLabel("Objeto:"))
        hlayout.addWidget(objeto_le)
        self.layout.addLayout(hlayout)
        return objeto_le

    def setup_fourth_line(self):
        hlayout = QHBoxLayout()
        nup_le = QLineEdit()
        sigla_om_cb = QComboBox()
        nup_le.setPlaceholderText("Exemplo: '00000.00000/0000-00'")
        hlayout.addWidget(QLabel("Nup:"))
```

```
        hlayout.addWidget(nup_le)
        hlayout.addWidget(QLabel("OM:"))
        hlayout.addWidget(sigla_om_cb)
        self.layout.addLayout(hlayout)
        return nup_le, sigla_om_cb

    def setup_fifth_line(self):
        hlayout = QHBoxLayout()
        material_radio = QRadioButton("Material")
        servico_radio = QRadioButton("Serviço")
        group = QButtonGroup(self)
        group.addButton(material_radio)
        group.addButton(servico_radio)
        material_radio.setChecked(True)

        hlayout.addWidget(QLabel("Material/Serviço:"))
        hlayout.addWidget(material_radio)
        hlayout.addWidget(servico_radio)
        self.layout.addLayout(hlayout)
        return material_radio, servico_radio

    def setup_save_button(self):
        btn = QPushButton("Adicionar Item")
        btn.clicked.connect(self.on_save)
        self.layout.addWidget(btn)

    def on_save(self):
        data = self.get_data()
        try:
            if self.check_id_exists(data['id_processo']):
                res = QMessageBox.question(
                    self, "Confirmação",
                    "ID do processo já existe. Deseja sobrescrever?",
                    QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
                    QMessageBox.StandardButton.No
                )
                if res == QMessageBox.StandardButton.Yes:
                    self.accept()  # Substitui o diálogo aceitar com a sobreposição
            else:
                self.accept()  # Aceita normalmente se o ID do processo não existir
        except sqlite3.OperationalError as e:
            if "no such table" in str(e):
                  QMessageBox.warning(self, "Erro", "A tabela 'controle_dispensas' não existe. Por
favor, atualize a interface gráfica do módulo.")
            else:
                QMessageBox.warning(self, "Erro", f"Ocorreu um erro: {str(e)}")

    def check_id_exists(self, id_processo):
        query = "SELECT 1 FROM controle_dispensas WHERE id_processo = ?"
        try:
            with sqlite3.connect(self.database_path) as conn:
                cursor = conn.cursor()
                cursor.execute(query, (id_processo,))
```

```
                return cursor.fetchone() is not None
        except sqlite3.OperationalError as e:
            if "no such table" in str(e):
                  QMessageBox.warning(self, "Erro", "A tabela 'controle_dispensas' não existe. Por favor, crie a tabela primeiro.")
            else:
                raise

    def load_next_numero(self):
        try:
            with sqlite3.connect(self.database_path) as conn:
                cursor = conn.cursor()
                # Converte o campo `numero` em um inteiro para garantir a comparação correta
                cursor.execute("SELECT MAX(CAST(numero AS INTEGER)) FROM controle_dispensas")
                max_number = cursor.fetchone()[0]
                next_number = 1 if max_number is None else int(max_number) + 1
                self.numero_le.setText(str(next_number))
        except Exception as e:
            print(f"Erro ao carregar o próximo número: {e}")

    def get_data(self):
        # Define o valor de 'sigla_om' como "CeIMBra" se a seleção atual estiver vazia
        sigla_selected = self.sigla_om_cb.currentText() or "CeIMBra"

        # Define valores padrão para 'orgao_responsavel' e 'uasg' caso 'sigla_selected' não esteja em 'self.om_details'
             orgao_responsavel  =  self.om_details.get(sigla_selected,  {}).get('orgao_responsavel', "Orgão Padrão")
        uasg = self.om_details.get(sigla_selected, {}).get('uasg', "000000")

        material_servico = "Material" if self.material_radio.isChecked() else "Serviço"
        tipo_de_processo = self.tipo_cb.currentText()
        com_disputa = "Sim"
        pesquisa_preco = "Não"
        atividade_custeio = "Não"

        data = {
            'tipo': tipo_de_processo,
            'numero': self.numero_le.text(),
            'ano': self.ano_le.text(),
            'nup': self.nup_le.text(),
            'objeto': self.objeto_le.text(),
            'sigla_om': sigla_selected,
            'orgao_responsavel': orgao_responsavel,
            'uasg': uasg,
            'material_servico': material_servico,
            'com_disputa': com_disputa,
            'pesquisa_preco': pesquisa_preco,
            'atividade_custeio': atividade_custeio
        }

        # Mapeamento do tipo de processo para o nome interno
        tipo_map = {
```

```
    "Dispensa Eletrônica (DE)": ("DE", "Dispensa Eletrônica"),
}

if tipo_de_processo in tipo_map:
    abreviatura, nome_interno = tipo_map[tipo_de_processo]
    data['tipo'] = nome_interno
    # Altera a formatação do id_processo para incluir a UASG
    data['id_processo'] = f"{abreviatura} 787010-{data['numero']}/{data['ano']}"
else:
    data['tipo'] = "Tipo Desconhecido"
    data['id_processo'] = f"Desconhecido {data['numero']}/{data['ano']}"

return data
```