Arquivo: model.py

```
from modules.dispensa.database_manager.db_manager import DatabaseManager
from PyQt6.QtCore import QObject
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from PyQt6.QtSql import QSqlDatabase, QSqlTableModel, QSqlQuery
from functools import partial
import sqlite3
import re
from .dialogs.edit_data.apoio_data import VALID_SITUATIONS

class DispensaEletronicaModel(QObject):
    def __init__(self, database_path, parent=None):
        super().__init__(parent)
        self.database_manager = DatabaseManager(database_path)
        self.db = None  # Adiciona um atributo para o banco de dados
        self.model = None  # Atributo para o modelo SQL
        self.init_database()  # Inicializa a conexão e a estrutura do banco de dados

    def init_database(self):
        """Inicializa a conexão com o banco de dados e ajusta a estrutura da tabela."""
        if QSqlDatabase.contains("my_conn"):
            QSqlDatabase.removeDatabase("my_conn")
        self.db = QSqlDatabase.addDatabase('QSQLITE', "my_conn")
        self.db.setDatabaseName(str(self.database_manager.db_path))

        if not self.db.open():
            print("Não foi possível abrir a conexão com o banco de dados.")
        else:
            print("Conexão com o banco de dados aberta com sucesso.")
            self.adjust_table_structure()  # Ajusta a estrutura da tabela, se necessário

    def save_api_data(self, data_api):
        """Salva os dados da API no banco de dados com depuração aprimorada."""

        # Inspecionar `data_api`
        print("DEBUG: Conteúdo de `data_api`:", data_api)

        # Acessa `data_informacoes` e converte para dicionário, se for uma lista de tuplas
        data_informacoes = data_api['data_informacoes']
        if isinstance(data_informacoes, list):
            data_informacoes = dict(data_informacoes)

        numero_controle_pncp = data_informacoes.get('numeroControlePNCP')
        if not numero_controle_pncp:
            print("Erro: 'numeroControlePNCP' não encontrado.")
            return

        # Remover caracteres especiais do nome da tabela
        table_name = re.sub(r'[/-]', '_', numero_controle_pncp)
```

```
print(f"DEBUG: Nome da tabela convertido: {table_name}")

# SQL para criar a tabela com as colunas especificadas
create_table_sql = f"""
    CREATE TABLE IF NOT EXISTS '{table_name}' (
        numeroItem INTEGER PRIMARY KEY,
        descricao TEXT,
        materialOuServico TEXT,
        valorUnitarioEstimado REAL,
        valorTotal REAL,
        valorUnitarioHomologado REAL,
        valorTotalHomologado REAL,
        quantidadeHomologada REAL,
        unidadeMedida TEXT,
        situacaoCompraItemNome TEXT,
        dataAtualizacao TEXT,
        niFornecedor TEXT,
        nomeRazaoSocialFornecedor TEXT,
        situacaoCompraItemResultadoNome TEXT
    )
"""

# Criação da tabela, se não existir
with self.database_manager as conn:
    cursor = conn.cursor()
    cursor.execute(create_table_sql)
    conn.commit()
    print(f"Tabela '{table_name}' criada ou já existe.")

    # Inserir os dados de `resultados_completos` na tabela
    insert_sql = f"""
        INSERT OR REPLACE INTO '{table_name}' (
            numeroItem,
            descricao,
            materialOuServico,
            valorUnitarioEstimado,
            valorTotal,
            valorUnitarioHomologado,
            valorTotalHomologado,
            quantidadeHomologada,
            unidadeMedida,
            situacaoCompraItemNome,
            dataAtualizacao,
            niFornecedor,
            nomeRazaoSocialFornecedor,
            situacaoCompraItemResultadoNome
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """

    # Depuração para verificar o SQL de inserção e os valores
    print("DEBUG: SQL de inserção:", insert_sql)

    for resultado in data_api['resultados_completos']:
```

```
                # Preparando os valores para inserção
                valores = (
                    resultado.get("numeroItem"),
                    resultado.get("descricao"),
                    resultado.get("materialOuServico"),
                    resultado.get("valorUnitarioEstimado"),
                    resultado.get("valorTotal"),
                    resultado.get("valorUnitarioHomologado"),
                    resultado.get("valorTotalHomologado"),
                    resultado.get("quantidadeHomologada"),
                    resultado.get("unidadeMedida"),
                    resultado.get("situacaoCompraItemNome"),
                    resultado.get("dataAtualizacao"),
                    resultado.get("niFornecedor"),
                    resultado.get("nomeRazaoSocialFornecedor"),
                    resultado.get("situacaoCompraItemResultadoNome")
                )

                # Verificando o conteúdo dos valores antes de inserir
                print(f"DEBUG: Inserindo valores na tabela '{table_name}': {valores}")

                # Inserir os valores na tabela
                try:
                    cursor.execute(insert_sql, valores)
                except Exception as e:
                    print(f"Erro ao inserir dados na tabela '{table_name}': {e}")

            conn.commit()

        print(f"Dados inseridos com sucesso na tabela '{table_name}'.")

    def adjust_table_structure(self):
        """Verifica e cria a tabela 'controle_dispensas' se não existir."""
        query = QSqlQuery(self.db)
                  if  not  query.exec("SELECT  name  FROM  sqlite_master  WHERE  type='table'  AND
name='controle_dispensas'"):
            print("Erro ao verificar existência da tabela:", query.lastError().text())
        if not query.next():
            print("Tabela 'controle_dispensas' não existe. Criando tabela...")
            self.create_table_if_not_exists()
        else:
            pass
            # print("Tabela 'controle_dispensas' existe. Verificando estrutura da coluna...")

    def save_api_data_to_database(self, data_api):
        # Obtém o valor de 'numeroControlePNCP' para nome da tabela
        numero_controle_pncp = data_api['data_informacoes'].get('numeroControlePNCP')

        if not numero_controle_pncp:
            print("Erro: 'numeroControlePNCP' não encontrado nos dados da API.")
            return

        # Constrói a consulta de criação de tabela com o nome dinâmico
```

```
create_table_sql = f"""
    CREATE TABLE IF NOT EXISTS '{numero_controle_pncp}' (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        valorTotalEstimado REAL,
        valorTotalHomologado REAL,
        orcamentoSigilosoCodigo INTEGER,
        orcamentoSigilosoDescricao TEXT,
        numeroControlePNCP TEXT,
        linkSistemaOrigem TEXT,
        anoCompra INTEGER,
        sequencialCompra INTEGER,
        numeroCompra TEXT,
        processo TEXT
        -- Adicione outras colunas conforme necessário
    )
"""

# Executa a criação da tabela
with self.database_manager as conn:
    cursor = conn.cursor()
    cursor.execute(create_table_sql)

    # Insere os dados da API na tabela criada
    insert_sql = f"""
        INSERT INTO '{numero_controle_pncp}' (
            valorTotalEstimado,
            valorTotalHomologado,
            orcamentoSigilosoCodigo,
            orcamentoSigilosoDescricao,
            numeroControlePNCP,
            linkSistemaOrigem,
            anoCompra,
            sequencialCompra,
            numeroCompra,
            processo
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """

    # Extrai valores de 'data_informacoes' para inserir na tabela
    data_informacoes = data_api['data_informacoes']
    valores = (
        data_informacoes.get("valorTotalEstimado"),
        data_informacoes.get("valorTotalHomologado"),
        data_informacoes.get("orcamentoSigilosoCodigo"),
        data_informacoes.get("orcamentoSigilosoDescricao"),
        data_informacoes.get("numeroControlePNCP"),
        data_informacoes.get("linkSistemaOrigem"),
        data_informacoes.get("anoCompra"),
        data_informacoes.get("sequencialCompra"),
        data_informacoes.get("numeroCompra"),
        data_informacoes.get("processo")
    )
```

```
        cursor.execute(insert_sql, valores)
        conn.commit()

    print(f"Tabela '{numero_controle_pncp}' criada e dados inseridos com sucesso.")


def create_table_if_not_exists(self):
    """Cria a tabela 'controle_dispensas' com a estrutura definida, caso ainda não exista."""
    query = QSqlQuery(self.db)
    if not query.exec("""
        CREATE TABLE IF NOT EXISTS controle_dispensas (
            situacao TEXT,
            id_processo VARCHAR(100) PRIMARY KEY,
            tipo VARCHAR(100),
            numero VARCHAR(100),
            ano VARCHAR(100),
            nup VARCHAR(100),
            material_servico VARCHAR(30),
            objeto VARCHAR(100),
            vigencia TEXT,
            data_sessao DATE,
            operador text,
            criterio_julgamento TEXT,
            com_disputa TEXT,
            pesquisa_preco TEXT,
            previsao_contratacao TEXT,
            uasg VARCHAR(10),
            orgao_responsavel VARCHAR(250),
            sigla_om VARCHAR(100),
            setor_responsavel TEXT,
            responsavel_pela_demanda TEXT,
            ordenador_despesas TEXT,
            agente_fiscal TEXT,
            gerente_de_credito TEXT,
            cp TEXT,
            cod_par TEXT,
            prioridade_par TEXT,
            cep TEXT,
            endereco TEXT,
            email TEXT,
            telefone TEXT,
            dias_para_recebimento TEXT,
            horario_para_recebimento TEXT,
            valor_total TEXT,
            acao_interna TEXT,
            fonte_recursos TEXT,
            natureza_despesa TEXT,
            unidade_orcamentaria TEXT,
            ptres TEXT,
            atividade_custeio TEXT,
            paraseredito TEXT,
            comentarios TEXT,
            justificativa TEXT,
```

```
                cnpj_matriz TEXT,
                sequencial_pncp TEXT,
                link_pncp TEXT,
                comunicacao_padronizada TEXT,
                campo_do_cp TEXT,
                campo_ao_cp TEXT
            )
        """):
            print("Falha ao criar a tabela 'controle_dispensas':", query.lastError().text())
        else:
            print("Tabela 'controle_dispensas' criada com sucesso.")

    def setup_model(self, table_name, editable=False):
        """Configura o modelo SQL para a tabela especificada."""
        # Passa o database_manager para o modelo personalizado
                                self.model    =    CustomSqlTableModel(parent=self,    db=self.db,
database_manager=self.database_manager, non_editable_columns=[4, 8, 10, 13])
        self.model.setTable(table_name)

        if editable:
            self.model.setEditStrategy(QSqlTableModel.EditStrategy.OnFieldChange)

        self.model.select()
        return self.model

    def get_data(self, table_name):
        """Retorna todos os dados da tabela especificada."""
        return self.database_manager.fetch_all(f"SELECT * FROM {table_name}")

    def insert_or_update_data(self, data):
        print("Dados recebidos para salvar:", data)
        upsert_sql = '''
        INSERT INTO controle_dispensas (
            situacao,
            id_processo,
            tipo,
            numero,
            ano,
            nup,
            material_servico,
            objeto,
            vigencia,
            uasg,
            orgao_responsavel,
            sigla_om,
            setor_responsavel,
            data_sessao,
            operador,
            criterio_julgamento,
            com_disputa,
            pesquisa_preco,
            atividade_custeio,
            parasereditado,
```

```
            previsao_contratacao,
            responsavel_pela_demanda,
            ordenador_despesas,
            agente_fiscal,
            gerente_de_credito,
            cp,
            cod_par,
            prioridade_par,
            justificativa,
            cep,
            endereco,
            email,
            telefone,
            dias_para_recebimento,
            horario_para_recebimento,
            valor_total,
            acao_interna,
            fonte_recursos,
            natureza_despesa,
            unidade_orcamentaria,
            ptres,
            cnpj_matriz,
            sequencial_pncp,
            link_pncp,
            comunicacao_padronizada,
            campo_do_cp,
            campo_ao_cp
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ON CONFLICT(id_processo) DO UPDATE SET
            situacao=excluded.situacao,
            tipo=excluded.tipo,
            numero=excluded.numero,
            ano=excluded.ano,
            nup=excluded.nup,
            material_servico=excluded.material_servico,
            objeto=excluded.objeto,
            vigencia=excluded.vigencia,
            uasg=excluded.uasg,
            orgao_responsavel=excluded.orgao_responsavel,
            sigla_om=excluded.sigla_om,
            setor_responsavel=excluded.setor_responsavel,
            data_sessao=excluded.data_sessao,
            operador=excluded.operador,
            criterio_julgamento=excluded.criterio_julgamento,
            com_disputa=excluded.com_disputa,
            pesquisa_preco=excluded.pesquisa_preco,
            atividade_custeio=excluded.atividade_custeio,
            parasereditado=excluded.parasereditado,
            previsao_contratacao=excluded.previsao_contratacao,
            responsavel_pela_demanda=excluded.responsavel_pela_demanda,
            ordenador_despesas=excluded.ordenador_despesas,
            agente_fiscal=excluded.agente_fiscal,
```

```
    gerente_de_credito=excluded.gerente_de_credito,
    cp=excluded.cp,
    cod_par=excluded.cod_par,
    prioridade_par=excluded.prioridade_par,
    justificativa=excluded.justificativa,
    cep=excluded.cep,
    endereco=excluded.endereco,
    email=excluded.email,
    telefone=excluded.telefone,
    dias_para_recebimento=excluded.dias_para_recebimento,
    horario_para_recebimento=excluded.horario_para_recebimento,
    valor_total=excluded.valor_total,
    acao_interna=excluded.acao_interna,
    fonte_recursos=excluded.fonte_recursos,
    natureza_despesa=excluded.natureza_despesa,
    unidade_orcamentaria=excluded.unidade_orcamentaria,
    ptres=excluded.ptres,
    cnpj_matriz=excluded.cnpj_matriz,
    sequencial_pncp=excluded.sequencial_pncp,
    link_pncp=excluded.link_pncp,
    comunicacao_padronizada=excluded.comunicacao_padronizada,
    campo_do_cp=excluded.campo_do_cp,
    campo_ao_cp=excluded.campo_ao_cp
'''

# Verifica se 'situacao' está dentro dos valores válidos
data['situacao'] = data.get('situacao', 'Planejamento')
if data['situacao'] not in VALID_SITUATIONS:
    data['situacao'] = 'Planejamento'

# Executa a inserção ou atualização
try:
    with self.database_manager as conn:
        cursor = conn.cursor()
        cursor.execute(upsert_sql, (
            data.get('situacao'),
            data.get('id_processo'),
            data.get('tipo'),
            data.get('numero'),
            data.get('ano'),
            data.get('nup'),
            data.get('material_servico'),
            data.get('objeto'),
            data.get('vigencia', '2 (dois) meses'),
            data.get('uasg'),
            data.get('orgao_responsavel'),
            data.get('sigla_om'),
            data.get('setor_responsavel', ''),
            data.get('data_sessao', ''),
            data.get('operador', ''),
            data.get('criterio_julgamento', ''),
            data.get('com_disputa'),
            data.get('pesquisa_preco'),
```

```
                    data.get('atividade_custeio'),
                    data.get('parasereditado'),
                    data.get('previsao_contratacao', ''),
                    data.get('responsavel_pela_demanda', ''),
                    data.get('ordenador_despesas', ''),
                    data.get('agente_fiscal', ''),
                    data.get('gerente_de_credito', ''),
                    data.get('cp', ''),
                    data.get('cod_par', ''),
                    data.get('prioridade_par', ''),
                    data.get('justificativa', ''),
                    data.get('cep', ''),
                    data.get('endereco', ''),
                    data.get('email', ''),
                    data.get('telefone', ''),
                    data.get('dias_recebimento', ''),
                    data.get('horario_recebimento', ''),
                    data.get('valor_total', ''),
                    data.get('acao_interna', ''),
                    data.get('fonte_recursos', ''),
                    data.get('natureza_despesa', ''),
                    data.get('unidade_orcamentaria', ''),
                    data.get('ptres', ''),
                    data.get('cnpj_matriz', '00394502000144'),
                    data.get('sequencial_pncp', ''),
                    data.get('link_pncp', ''),
                    data.get('comunicacao_padronizada', ''),
                    data.get('campo_do_cp', ''),
                    data.get('campo_ao_cp', '')
                ))
                conn.commit()

        except sqlite3.OperationalError as e:
            if "no such table" in str(e):
                  QMessageBox.warning(None, "Erro", "A tabela 'controle_dispensas' não existe. Por
favor, crie a tabela primeiro.")
                return
            else:
                   QMessageBox.warning(None, "Erro", f"Ocorreu um erro ao tentar salvar os dados:
{str(e)}")

class CustomSqlTableModel(QSqlTableModel):
    def __init__(self, parent=None, db=None, database_manager=None, non_editable_columns=None):
        super().__init__(parent, db)
        self.database_manager = database_manager
         self.non_editable_columns = non_editable_columns if non_editable_columns is not None else
[]

        # Define os nomes das colunas
        self.column_names = [
            "situacao", "id_processo", "tipo", "numero", "ano", "nup", "material_servico",
            "objeto", "vigencia", "data_sessao", "operador", "criterio_julgamento",
            "com_disputa", "pesquisa_preco", "previsao_contratacao", "uasg",
```

```
            "orgao_responsavel", "sigla_om", "setor_responsavel", "responsavel_pela_demanda",
            "ordenador_despesas", "agente_fiscal", "gerente_de_credito", "cp", "cod_par",
            "prioridade_par", "cep", "endereco", "email", "telefone",
            "dias_para_recebimento", "horario_para_recebimento", "valor_total",
            "acao_interna", "fonte_recursos", "natureza_despesa", "unidade_orcamentaria",
                  "ptres", "atividade_custeio", "parasereditado", "comentarios", "justificativa",
"cnpj_matriz","sequencial_pncp", "link_pncp",
            "comunicacao_padronizada", "campo_do_cp", "campo_ao_cp"
        ]


    def flags(self, index):
        if index.column() in self.non_editable_columns:
              return super().flags(index) & ~Qt.ItemFlag.ItemIsEditable  # Remove a permissão de
edição
        return super().flags(index)

    def data(self, index, role=Qt.ItemDataRole.DisplayRole):
        # Verifica se a coluna deve ser não editável e ajusta o retorno para DisplayRole
        if role == Qt.ItemDataRole.DisplayRole and index.column() in self.non_editable_columns:
            return super().data(index, role)

        return super().data(index, role)
```