Arquivo: model.py

```
from modules.contratos.database_manager.db_manager import DatabaseManager
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from PyQt6.QtSql import QSqlDatabase, QSqlTableModel, QSqlQuery
import sqlite3
from datetime import datetime

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

    def create_table_if_not_exists(self):
        """Cria a tabela 'controle_contratos' com a estrutura definida, caso ainda não exista."""
        query = QSqlQuery(self.db)
        if not query.exec("""
            CREATE TABLE IF NOT EXISTS controle_contratos (
                status TEXT, dias TEXT, prorrogavel TEXT, custeio TEXT, numero_contrato TEXT,
                  tipo TEXT, id TEXT PRIMARY KEY, nome_fornecedor TEXT, objeto TEXT, valor_global
TEXT,
```

```
                codigo_uasg TEXT, processo_nup TEXT, cnpj_cpf_idgener TEXT, natureza_continuada
TEXT, orgao_contratante_resumido TEXT,
                orgao_contratante TEXT, material_servico TEXT, link_pncp TEXT, vigencia_inicial
TEXT, vigencia_final TEXT,
                    termo_aditivo TEXT, atualizacao_comprasnet TEXT, instancia_governanca TEXT,
comprasnet_contratos TEXT, licitacao_numero TEXT,
                     data_assinatura TEXT, data_publicacao TEXT, categoria TEXT, subtipo TEXT,
amparo_legal TEXT,
                modalidade TEXT, assinatura_contrato TEXT, situacao TEXT

            )
        """):
            print("Falha ao criar a tabela 'controle_contratos':", query.lastError().text())
        else:
            print("Tabela 'controle_contratos' 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)

        # Define 'Planejamento' como valor padrão para status se estiver vazio ou None
        data['status'] = data.get('status', 'Planejamento')
        if not data['status']:  # Se for string vazia, define 'Planejamento'
            data['status'] = 'Planejamento'

        upsert_sql = '''
        INSERT INTO controle_contratos (
            status, dias, prorrogavel, custeio, numero_contrato,
            tipo, id, nome_fornecedor, objeto, valor_global,
                            codigo_uasg,   processo_nup,   cnpj_cpf_idgener,   natureza_continuada,
orgao_contratante_resumido,
            orgao_contratante, material_servico, link_pncp, vigencia_inicial, vigencia_final,
                termo_aditivo, atualizacao_comprasnet, instancia_governanca, comprasnet_contratos,
licitacao_numero,
            data_assinatura, data_publicacao, categoria, subtipo, amparo_legal,
            modalidade, assinatura_contrato, situacao
        ) VALUES (
```

```
            ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?,
            ?, ?, ?)
        ON CONFLICT(id) DO UPDATE SET
                        status=excluded.status, dias=excluded.dias, prorrogavel=excluded.prorrogavel,
custeio=excluded.custeio, numero_contrato=excluded.numero_contrato,
                                    tipo=excluded.tipo,  nome_fornecedor=excluded.nome_fornecedor,
objeto=excluded.objeto, valor_global=excluded.valor_global,
                               codigo_uasg=excluded.codigo_uasg,  processo_nup=excluded.processo_nup,
cnpj_cpf_idgener=excluded.cnpj_cpf_idgener, natureza_continuada=excluded.natureza_continuada,
                                   orgao_contratante_resumido=excluded.orgao_contratante_resumido,
orgao_contratante=excluded.orgao_contratante,          material_servico=excluded.material_servico,
link_pncp=excluded.link_pncp,
                                                       vigencia_inicial=excluded.vigencia_inicial,
vigencia_final=excluded.vigencia_final,                      termo_aditivo=excluded.termo_aditivo,
atualizacao_comprasnet=excluded.atualizacao_comprasnet,
                                                 instancia_governanca=excluded.instancia_governanca,
comprasnet_contratos=excluded.comprasnet_contratos,     licitacao_numero=excluded.licitacao_numero,
data_assinatura=excluded.data_assinatura,
                            data_publicacao=excluded.data_publicacao,  categoria=excluded.categoria,
subtipo=excluded.subtipo, amparo_legal=excluded.amparo_legal,
                  modalidade=excluded.modalidade, assinatura_contrato=excluded.assinatura_contrato,
situacao=excluded.situacao
        '''

        # Verifica se 'situacao' está dentro dos valores válidos
              valid_situations  =  ["Planejamento",  "Aprovado",  "Sessão  Pública",  "Homologado",
"Empenhado", "Concluído", "Arquivado"]
        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('status'),  data.get('dias'),  data.get('prorrogavel'),
data.get('custeio'), data.get('numero_contrato'),
                                   data.get('tipo'),  data.get('id'),  data.get('nome_fornecedor'),
data.get('objeto'), data.get('valor_global'),
                                                 data.get('codigo_uasg'),  data.get('processo_nup'),
data.get('cnpj_cpf_idgener'),                                          data.get('natureza_continuada'),
data.get('orgao_contratante_resumido'),
                                     data.get('orgao_contratante'),  data.get('material_servico'),
data.get('link_pncp'), data.get('vigencia_inicial'), data.get('vigencia_final'),
                                    data.get('termo_aditivo'),  data.get('atualizacao_comprasnet'),
data.get('instancia_governanca'), data.get('comprasnet_contratos'), data.get('licitacao_numero'),
                                        data.get('data_assinatura'),  data.get('data_publicacao'),
```

```
data.get('categoria'), data.get('subtipo'), data.get('amparo_legal'),
                    data.get('modalidade'), data.get('assinatura_contrato'), data.get('situacao')
                    ))
                conn.commit()

        except sqlite3.OperationalError as e:
            if "no such table" in str(e):
                  QMessageBox.warning(None, "Erro", "A tabela 'controle_contratos' 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 = [
            "status", "dias", "prorrogavel", "custeio", "numero_contrato",
            "tipo", "id", "nome_fornecedor", "objeto", "valor_global",
                       "codigo_uasg",  "processo_nup",  "cnpj_cpf_idgener",  "natureza_continuada",
"orgao_contratante_resumido",
                       "orgao_contratante",  "material_servico",  "link_pncp",  "vigencia_inicial",
"vigencia_final",
                                "termo_aditivo",   "atualizacao_comprasnet",   "instancia_governanca",
"comprasnet_contratos", "licitacao_numero",
            "data_assinatura", "data_publicacao", "categoria", "subtipo", "amparo_legal",
            "modalidade", "assinatura_contrato", "situacao"
        ]

    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 estamos na coluna 'status'
        if index.column() == self.fieldIndex("status"):
            if role == Qt.ItemDataRole.DisplayRole:
                 status_value = super().data(index, Qt.ItemDataRole.DisplayRole)  # Obtém valor do
banco de dados
                if not status_value:  # Se estiver vazio ou None, substitui por "Planejamento"
                      # print(f"[DEBUG] Status vazio encontrado na linha {index.row()}. Definindo
'Planejamento'.")
                    return "Planejamento"
                return status_value  # Caso contrário, retorna o valor correto
```

```
        # Verifica se estamos na coluna 'dias'
        if index.column() == self.fieldIndex("dias"):
            if role == Qt.ItemDataRole.DisplayRole:
                # Obtém o índice e valor da coluna "vigencia_final"
                vigencia_final_index = self.fieldIndex("vigencia_final")
                vigencia_final = self.index(index.row(), vigencia_final_index).data()

                            #  print(f"[DEBUG] Vigência final obtida para a linha {index.row()}:
{vigencia_final}")  # Debug print

                if vigencia_final:
                    try:
                        # Tenta converter 'DD/MM/YYYY'
                        vigencia_final_date = datetime.strptime(vigencia_final, '%d/%m/%Y')
                                        #  print(f"[DEBUG] Conversão bem-sucedida (DD/MM/YYYY):
{vigencia_final_date}")  # Debug print
                    except ValueError:
                        try:
                            # Tenta converter 'YYYY-MM-DD'
                            vigencia_final_date = datetime.strptime(vigencia_final, '%Y-%m-%d')
                                         #  print(f"[DEBUG] Conversão bem-sucedida (YYYY-MM-DD):
{vigencia_final_date}")  # Debug print
                        except ValueError:
                               # print(f"[ERRO] Falha ao converter data: {vigencia_final}. Formato
inválido.")  # Debug print
                            return "Data Inválida"

                    # Calcula os dias restantes
                    hoje = datetime.today()
                    dias = (vigencia_final_date - hoje).days
                       # print(f"[DEBUG] Hoje: {hoje}, Vigência final: {vigencia_final_date}, Dias
restantes: {dias}")  # Debug print
                    return dias  # Retorna os dias restantes

                else:
                    print(f"[ERRO] Sem data para calcular na linha {index.row()}")  # Debug print
                    return "Erro"

            elif role == Qt.ItemDataRole.ForegroundRole:
                # Obtém o valor da coluna 'dias' calculado anteriormente
                value = self.data(index, Qt.ItemDataRole.DisplayRole)
                if isinstance(value, int):  # Certifica-se de que é numérico
                    if value < 0:
                        return QColor(195, 195, 195)  # Cinza
                    elif 0 <= value < 30:
                        return QColor(255, 0, 0)  # Vermelho vivo
                    elif 30 <= value < 60:
                        return QColor(255, 140, 0)  # Laranja forte
                    elif 60 <= value < 90:
                        return QColor(255, 200, 0)  # Amarelo alaranjado
                    elif 90 <= value < 120:
                        return QColor(255, 255, 0)  # Amarelo vivo
                    elif 120 <= value < 180:
```

```
                return QColor(173, 255, 47)  # Verde amarelado
            elif 180 <= value < 360:
                return QColor(50, 205, 50)  # Verde médio
            elif value > 360:
                return QColor(0, 150, 255)  # Azul vivo para valores maiores que 360

# Coluna 'prorrogável' (Exemplo de outra coloração)
if index.column() == self.fieldIndex("prorrogavel"):
    value = super().data(index, Qt.ItemDataRole.DisplayRole)
    if role == Qt.ItemDataRole.ForegroundRole:
        if value == "Sim":
            return QColor(50, 205, 50)  # Verde
        elif value == "Não":
            return QColor(255, 0, 0)  # Vermelho

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