Arquivo: model.py

```
from modules.atas_api.database import DatabaseATASAPIManager
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import os
import pandas as pd
from PyQt6.QtSql import QSqlDatabase, QSqlTableModel, QSqlQuery
import logging
import sqlite3

class GerarAtasApiModel(QObject):
    tabelaCarregada = pyqtSignal()

    def __init__(self, database_path, parent=None):
        super().__init__(parent)
        self.database_ata_manager = DatabaseATASAPIManager(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):
        if QSqlDatabase.contains("my_conn"):
            QSqlDatabase.removeDatabase("my_conn")

        db_path = str(self.database_ata_manager.db_path)
        self.db = QSqlDatabase.addDatabase('QSQLITE', "my_conn")
        self.db.setDatabaseName(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_atas_structure()


    def setup_model(self, table_name, editable=False):
        """Configura o modelo SQL para a tabela especificada."""
        self.model = CustomSqlTableModel(
            parent=self, db=self.db, database_manager=self.database_ata_manager,
            non_editable_columns=[4, 8, 10, 13], gerar_atas_model=self
        )
        self.model.setTable(table_name)
        self.model.setEditStrategy(QSqlTableModel.EditStrategy.OnFieldChange)
        self.model.select()
        return self.model

    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_atas_api (
                grupo TEXT,
                item TEXT PRIMARY KEY,
                catalogo TEXT,
                descricao TEXT,
                descricao_detalhada TEXT,
                unidade TEXT,
                quantidade TEXT,
                valor_estimado TEXT,
                valor_homologado_item_unitario TEXT,
                percentual_desconto TEXT,
                valor_estimado_total_do_item TEXT,
                valor_homologado_total_item TEXT,
                marca_fabricante TEXT,
                modelo_versao TEXT,
                situacao TEXT,
                uasg TEXT,
                orgao_responsavel TEXT,
                num_pregao TEXT,ano_pregao TEXT,
                srp TEXT,
                objeto TEXT,
                melhor_lance TEXT,
                valor_negociado TEXT,
                ordenador_despesa TEXT,
                empresa TEXT,
                cnpj TEXT
            )
        """):
            print("Falha ao criar a tabela 'controle_atas_api':", query.lastError().text())
        else:
            print("Tabela 'controle_atas_api' criada com sucesso.")

    def adjust_table_atas_structure(self):
        query = QSqlQuery(self.db)
                  if  not  query.exec("SELECT  name  FROM  sqlite_master  WHERE  type='table'  AND
name='controle_atas_api'"):
            print("Erro ao verificar existência da tabela:", query.lastError().text())
        if not query.next():
            print("Tabela 'controle_atas_api' não existe. Criando tabela...")
            self.create_table_if_not_exists()
        else:
            print("Tabela 'controle_atas_api' existe. Verificando estrutura da coluna...")

    def configure_columns(self, table_view, visible_columns):
        for column in range(self.model.columnCount()):
            header = self.model.headerData(column, Qt.Orientation.Horizontal)
            if column not in visible_columns:
                table_view.hideColumn(column)
            else:
                self.model.setHeaderData(column, Qt.Orientation.Horizontal, header)

class CustomSqlTableModel(QSqlTableModel):
    tabelaCarregada = pyqtSignal()
```

```
      def __init__(self, parent=None, db=None, database_manager=None, non_editable_columns=None,
gerar_atas_model=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
[]
        self.gerar_atas_model = gerar_atas_model
        self.connection = None

        # Define os nomes das colunas
        self.column_names = [
            'grupo', 'item', 'catalogo', 'descricao', 'unidade', 'quantidade', 'valor_estimado',
                                        'valor_homologado_item_unitario',   'percentual_desconto',
'valor_estimado_total_do_item',
            'valor_homologado_total_item', 'marca_fabricante', 'modelo_versao', 'situacao',
            'descricao_detalhada', 'uasg', 'orgao_responsavel', 'num_pregao', 'ano_pregao',
            'srp', 'objeto', 'melhor_lance', 'valor_negociado', 'ordenador_despesa', 'empresa',
            'cnpj'
        ]

    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)

    def connect_to_database(self):
        if self.connection is None:
               self.connection = sqlite3.connect(self.database_manager.db_path)  # Corrige para o
atributo correto
                                  logging.info(f"Conexão   com   o   banco   de   dados   aberta   em
{self.database_manager.db_path}")
        return self.connection

    def close_connection(self):
        if self.connection:
            logging.info("Fechando conexão com o banco de dados...")
            self.connection.close()
            self.connection = None
            logging.info(f"Conexão com o banco de dados fechada em {self.database_manager}")

    def execute_query(self, query, params=None):
        conn = self.connect_to_database()
        try:
            cursor = conn.cursor()
```

```
            if params:
                cursor.execute(query, params)
            else:
                cursor.execute(query)
            conn.commit()
            return cursor.fetchall()
        except sqlite3.Error as e:
            logging.error(f"Erro ao executar consulta: {query}, Erro: {e}")
            return None
        finally:
            self.close_connection()

    def carregar_tabela(self):
        conn = None  # Inicializa conn como None
        try:
            # Seleciona o arquivo para carregar
                  caminho_arquivo, _ = QFileDialog.getOpenFileName(None, "Carregar Tabela", "",
"Arquivos Excel (*.xlsx);;Todos os Arquivos (*)")
            if not caminho_arquivo:
                return None  # Se o usuário cancelar o diálogo, sai da função

            # Carrega o arquivo Excel, filtrando apenas as colunas desejadas
                tabela = pd.read_excel(caminho_arquivo, usecols=['item', 'catalogo', 'descricao',
'descricao_detalhada'])

            # Conecta ao banco de dados
            conn = self.database_manager.connect_to_database()
            cursor = conn.cursor()

            # Exclui a tabela existente, se houver
            cursor.execute("DROP TABLE IF EXISTS controle_atas_api")
            conn.commit()
            print("Tabela 'controle_atas_api' excluída com sucesso.")

            # Recria a tabela usando o método do GerarAtasModel
            if self.gerar_atas_model:
                self.gerar_atas_model.create_table_if_not_exists()

            # Insere os novos dados no banco de dados
            tabela.to_sql("controle_atas_api", conn, if_exists='append', index=False)
            print("Dados inseridos no banco com sucesso.")

            # Atualiza o modelo para refletir as mudanças
            self.select()  # Recarrega os dados do banco
            self.tabelaCarregada.emit()  # Emite o sinal de que a tabela foi carregada

        except Exception as e:
            print(f"Erro ao carregar a tabela: {e}")
            QMessageBox.critical(None, "Erro", f"Erro ao carregar a tabela: {e}")

        finally:
            if conn:  # Verifica se conn foi atribuído antes de fechar
                conn.close()
```

```
    def abrir_tabela_nova(self):
        # Define o caminho do arquivo Excel
        file_path = os.path.join(os.getcwd(), "tabela_nova.xlsx")

        # Cria um DataFrame vazio com as colunas especificadas
        df = pd.DataFrame(columns=["item", "catalogo", "descricao", "descricao_detalhada"])

        try:
            # Tenta salvar o DataFrame no arquivo Excel
            df.to_excel(file_path, index=False)

            # Abre o arquivo Excel após a criação (opcional)
            os.startfile(file_path)
        except PermissionError:
            # Mostra uma mensagem para o usuário caso o arquivo já esteja aberto
              QMessageBox.warning(None, "Aviso", "O arquivo 'tabela_nova.xlsx' já está aberto. Por favor, feche-o e tente novamente.")
```