Arquivo: view.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
import pandas as pd
import os
from modules.utils.add_button import add_button_func
import sys
import subprocess
from pathlib import Path

class IndicadoresView(QWidget):
    def __init__(self, icons, db_manager, data_atas_path, data_atas_api_path, parent=None):
        super().__init__(parent)

        self.icon_cache = icons
        self.db_manager = db_manager
        self.data_atas_path = data_atas_path
        self.data_atas_api_path = data_atas_api_path

        self.setup_ui()

    def setup_ui(self):
        """Configura a interface da view."""
        self.layout = QVBoxLayout(self)
        self.layout.setContentsMargins(10, 10, 10, 10)
        self.layout.setSpacing(15)

        label_title = QLabel("Indicadores", self)
        label_title.setStyleSheet("font-size: 26px; font-weight: bold;")
        self.layout.addWidget(label_title)

        self.selector_layout = QHBoxLayout()

        self.select_db = QVBoxLayout()

        # Rótulo de instrução
        self.label = QLabel("Selecione a tabela para visualização:", self)
        self.label.setAlignment(Qt.AlignmentFlag.AlignLeft)
        self.label.setFont(QFont('Arial', 12))
        self.select_db.addWidget(self.label)

        self.db_selector = QComboBox(self)
        self.db_selector.addItem("Controle Atas", self.data_atas_path)
        self.db_selector.addItem("Controle Atas API", self.data_atas_api_path)
        self.db_selector.currentIndexChanged.connect(self.update_table_selector)
        self.db_selector.setFont(QFont('Arial', 12))
        self.db_selector.setFixedWidth(250)
        self.select_db.addWidget(self.db_selector)

        # Combobox para selecionar a tabela
```

```
        self.table_selector = QComboBox(self)
        self.table_selector.currentIndexChanged.connect(self.load_table_data)
        self.table_selector.setFont(QFont('Arial', 12))
        self.table_selector.setFixedWidth(250)
        self.select_db.addWidget(self.table_selector)

        self.selector_layout.addLayout(self.select_db)

        # Adicionar QLabel para contadores
        self.count_label = QLabel("", self)
        self.count_label.setFont(QFont('Arial', 14))
        self.count_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.selector_layout.addWidget(self.count_label)  # Adiciona no layout principal

        self.indicadores_layout = QVBoxLayout()
        # Indicador de Economicidade
        economicidade_label = QLabel("Indicador de Economicidade")
        economicidade_label.setFont(QFont('Arial', 12, QFont.Weight.Bold))
        economicidade_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.indicadores_layout.addWidget(economicidade_label)

        # Cálculo do indicador de economicidade
        self.economicidade_percentual = QLabel("0.00% de economia média")
        self.economicidade_percentual.setFont(QFont('Arial', 14))
        self.economicidade_percentual.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.indicadores_layout.addWidget(self.economicidade_percentual)

        button_layout = QHBoxLayout()  # Layout horizontal para o botão
            button_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)   # Centraliza o conteúdo do
layout

        add_button_func(
            text="Gerar Tabela",
            icon_name="excel",
            slot=self.gerar_tabela_excel,
            layout=button_layout,  # Adiciona o botão ao layout centralizado
            icons=self.icon_cache,
            tooltip="Clique para gerar a tabela com os dados",
            button_size=(150, 30)
        )

        self.indicadores_layout.addLayout(button_layout)

        self.selector_layout.addLayout(self.indicadores_layout)

        self.layout.addLayout(self.selector_layout)

        # Tabela para exibir os dados (usando QTableView com modelo dinâmico)
        self.table_view = QTableView(self)
        self.table_view.setStyleSheet(
            """
            QTableView {
                background-color: #F3F3F3;
```

```
                color: #333333;
                gridline-color: #CCCCCC;
                alternate-background-color: #FFFFFF;
                selection-background-color: #E0E0E0;
                selection-color: #000000; /* Cor do texto ao selecionar */
                font-size: 14px;
            }
            QHeaderView::section {
                background-color: #D6D6D6; /* Fundo do cabeçalho */
                color: #333333; /* Cor do texto do cabeçalho */
                font-weight: bold;
                font-size: 14px;
                padding: 4px;
                border: 1px solid #CCCCCC; /* Borda entre as seções */
            }
            """
        )
        self.layout.addWidget(self.table_view)

        # Atualiza as tabelas ao inicializar
        self.update_table_selector()

    def update_table_selector(self):
        """Atualiza o combobox de tabelas com base no banco de dados selecionado."""
        selected_db_path = self.db_selector.currentData()
        tables = self.db_manager.get_tables_with_keyword('result', selected_db_path)

        self.table_selector.clear()
        if tables:
            self.table_selector.addItems(tables)
        else:
                 QMessageBox.warning(self, "Aviso", "Não há tabelas disponíveis que comecem com
'result'.")


    def load_table_data(self):
        """Carrega os dados da tabela selecionada e atualiza a exibição."""
        selected_table = self.table_selector.currentText()
        selected_db_path = self.db_selector.currentData()

        if not selected_table:
            return

        dataframe = self.db_manager.load_table_to_dataframe(selected_table, selected_db_path)
        if dataframe is None or dataframe.empty:
            QMessageBox.warning(self, "Aviso", "Falha ao carregar a tabela selecionada.")
            return

        self.update_table(dataframe)
        self.update_economicidade(dataframe)
        self.update_chart(dataframe)

    def update_chart(self, dataframe):
```

```
        """Atualiza os contadores com base nos dados e exibe na QLabel (self.count_label)."""
        if len(dataframe.columns) <= 14:
            QMessageBox.warning(self, "Aviso", "O DataFrame não possui o índice 14.")
            return

        # Obtém os valores da coluna pelo índice 14
        column_data = dataframe.iloc[:, 14]

        # Conta os tipos de itens encontrados
        counts = column_data.value_counts()
        total_count = column_data.count()

        if total_count == 0:
            text = "Nenhum item encontrado."
        else:
            # Formata os resultados
            text = ""
            for item, count in counts.items():
                percentage = (count / total_count) * 100
                text += f"{item}: {count} itens ({percentage:.2f}%)\n"

            # Adiciona o total de itens
            text += f"\nTotal de itens: {total_count}"

        # Exibe o texto na QLabel
        self.count_label.setText(text)
        self.count_label.setAlignment(Qt.AlignmentFlag.AlignLeft)
        self.count_label.setFont(QFont('Arial', 12))



    def update_table(self, dataframe):
        """Atualiza a exibição da tabela com os dados do DataFrame usando QTableView."""
        model = QStandardItemModel()

        # Adicionar cabeçalhos das colunas
        model.setHorizontalHeaderLabels(dataframe.columns.tolist())

        # Preencher o modelo com os dados
        for row in dataframe.itertuples(index=False):
            items = [QStandardItem(str(value)) for value in row]
            model.appendRow(items)

        self.table_view.setModel(model)
        self.table_view.resizeColumnsToContents()
        self.configurar_visualizacao_tabela_tr(self.table_view)
        self.ajustar_coluna_percentual(self.table_view, 9)
        self.ajustar_coluna_brl(self.table_view, 7)
        self.ajustar_coluna_brl(self.table_view, 8)

    def ajustar_coluna_brl(self, table_view, column_index):
            """Ajusta os valores da coluna especificada para serem apresentados em formato BRL
(R$)."""
```

```
        model = table_view.model()
        if model is None:
            return

        for row in range(model.rowCount()):
            value = model.data(model.index(row, column_index))
            try:
                value = float(value)
                            formatted_value  =  f"R$  {value:,.2f}".replace(",",  "X").replace(".",
",").replace("X", ".")
                model.setData(model.index(row, column_index), formatted_value)
            except ValueError:
                continue

    def ajustar_coluna_percentual(self, table_view, column_index):
        """Ajusta os valores da coluna especificada para serem apresentados em percentual com duas
casas decimais."""
        model = table_view.model()
        if model is None:
            return

        for row in range(model.rowCount()):
            value = model.data(model.index(row, column_index))
            try:
                value = float(value)
                formatted_value = f"{value:.2f}%"
                model.setData(model.index(row, column_index), formatted_value)
            except ValueError:
                continue

    def configurar_visualizacao_tabela_tr(self, table_view):
        """Configura colunas visíveis, redimensionamento e centralização na QTableView."""

        # Adiciona rótulos para as colunas
        table_view.model().setHeaderData(1, Qt.Orientation.Horizontal, 'Item')
        table_view.model().setHeaderData(3, Qt.Orientation.Horizontal, 'Descrição')
        table_view.model().setHeaderData(5, Qt.Orientation.Horizontal, 'UF')
        table_view.model().setHeaderData(7, Qt.Orientation.Horizontal, 'Valor\nEstimado')
        table_view.model().setHeaderData(8, Qt.Orientation.Horizontal, 'Valor\nHomologado')
        table_view.model().setHeaderData(9, Qt.Orientation.Horizontal, 'Percentual\nDesconto(%)')
        table_view.model().setHeaderData(14, Qt.Orientation.Horizontal, 'Situação')

        # Verifica se o modelo foi configurado antes de prosseguir
        if table_view.model() is None:
            print("O modelo de dados não foi configurado para table_view.")
            return  # Sai da função se o modelo não estiver configurado

        # Define colunas visíveis
        visible_columns = [1, 3, 5, 7, 8, 9, 14]  # Colunas visíveis
        for col in range(table_view.model().columnCount()):
            if col not in visible_columns:
                table_view.hideColumn(col)  # Oculta as colunas que não estão na lista
```

```
        # Oculta a coluna de índice
        table_view.verticalHeader().setVisible(False)

        # Configuração de redimensionamento das colunas
        table_view.setColumnWidth(1, 50)
        table_view.setColumnWidth(3, 200)
        table_view.setColumnWidth(5, 120)
        table_view.setColumnWidth(9, 100)

        table_view.horizontalHeader().setStretchLastSection(True)
        table_view.horizontalHeader().setSectionResizeMode(1, QHeaderView.ResizeMode.Fixed)
        table_view.horizontalHeader().setSectionResizeMode(3, QHeaderView.ResizeMode.Fixed)
        table_view.horizontalHeader().setSectionResizeMode(5, QHeaderView.ResizeMode.Fixed)
                                            table_view.horizontalHeader().setSectionResizeMode(7,
QHeaderView.ResizeMode.ResizeToContents)
                                            table_view.horizontalHeader().setSectionResizeMode(8,
QHeaderView.ResizeMode.ResizeToContents)
        table_view.horizontalHeader().setSectionResizeMode(9, QHeaderView.ResizeMode.Fixed)
        table_view.horizontalHeader().setSectionResizeMode(14, QHeaderView.ResizeMode.Stretch)

        # Centraliza o conteúdo das células
        delegate = QStyledItemDelegate()
        table_view.setItemDelegate(delegate)
        for row in range(table_view.model().rowCount()):
            for col in visible_columns:
                index = table_view.model().index(row, col)
                                    table_view.model().setData(index,  Qt.AlignmentFlag.AlignCenter,
Qt.ItemDataRole.TextAlignmentRole)

    def update_economicidade(self, dataframe):
        """Atualiza o indicador de economicidade com base no DataFrame."""
        economicidade_percentual = self.calcular_economicidade(dataframe)
              self.economicidade_percentual.setText(f"{economicidade_percentual:.2f}%  de  economia
média")

    def calcular_economicidade(self, dataframe):
        """Calcula a média dos percentuais de desconto."""
          if 'situacao' not in dataframe.columns or 'valor_estimado' not in dataframe.columns or
'valor_homologado_item_unitario' not in dataframe.columns:
            return 0.0

        # Filtrar dados homologados
        df_homologado = dataframe[dataframe['situacao'] == 'Adjudicado e Homologado'].copy()
        if df_homologado.empty:
            return 0.0

        # Converter colunas para tipo numérico, ignorando erros
                df_homologado['valor_estimado']  =  pd.to_numeric(df_homologado['valor_estimado'],
errors='coerce')
                                            df_homologado['valor_homologado_item_unitario']     =
pd.to_numeric(df_homologado['valor_homologado_item_unitario'], errors='coerce')

        # Remover linhas com valores nulos após a conversão
```

```
                            df_homologado    =    df_homologado.dropna(subset=['valor_estimado',
'valor_homologado_item_unitario'])
        if df_homologado.empty:
            return 0.0

        # Calcular percentual de desconto
                    df_homologado['percentual_desconto']   =   ((df_homologado['valor_estimado']   -
df_homologado['valor_homologado_item_unitario'])
                                                / df_homologado['valor_estimado']) * 100
        return df_homologado['percentual_desconto'].mean()

    def gerar_tabela_excel(self):
        """Gera e abre uma tabela Excel com valores de economicidade usando fórmulas."""

        selected_table = self.table_selector.currentText()
        selected_db_path = self.db_selector.currentData()

        if not selected_table:
            QMessageBox.warning(self, "Aviso", "Nenhuma tabela selecionada para exportação.")
            return

        dataframe = self.db_manager.load_table_to_dataframe(selected_table, selected_db_path)
        if dataframe is None or dataframe.empty:
            QMessageBox.warning(self, "Aviso", "Falha ao carregar a tabela selecionada.")
            return

        df_homologado = dataframe[dataframe['situacao'] == 'Adjudicado e Homologado'].copy()
                    df_homologado   =   df_homologado[['item',   'descricao',   'valor_estimado',
'valor_homologado_item_unitario']].copy()

        # Obtém o caminho da Área de Trabalho de acordo com o sistema operacional
        if sys.platform == "win32":
            desktop_path = Path.home() / "Desktop"
        elif sys.platform == "darwin":  # macOS
            desktop_path = Path.home() / "Desktop"
        else:  # Linux (Ubuntu, Debian, etc.)
            desktop_path = Path.home() / "Área de Trabalho"
            if not desktop_path.exists():  # Algumas distros usam "Desktop" em inglês
                desktop_path = Path.home() / "Desktop"

        file_path = desktop_path / "indicador.xlsx"

        try:
            with pd.ExcelWriter(file_path, engine='xlsxwriter') as writer:
                df_homologado.to_excel(writer, index=False, sheet_name='Indicador')

            QMessageBox.information(self, "Sucesso", f"O arquivo foi salvo em:\n{file_path}")

            # Abrir o arquivo no sistema operacional correspondente
            if sys.platform == "win32":
                os.startfile(file_path)
            elif sys.platform == "darwin":  # macOS
                subprocess.Popen(["open", file_path])
```

```
    else:  # Linux (Ubuntu, Debian, etc.)
        subprocess.Popen(["xdg-open", file_path])

except Exception as e:
    QMessageBox.critical(self, "Erro", f"Erro ao salvar o arquivo:\n{e}")
```