## Arquivo: view.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from modules.utils.search_bar import setup_search_bar, MultiColumnFilterProxyModel
from modules.utils.add_button import add_button

import pandas as pd

class PlanejamentoView(QMainWindow):
    # Sinais para comunicação com o controlador
    messageAlert = pyqtSignal()
    apiCheck = pyqtSignal()
    openTable = pyqtSignal()
    loadData = pyqtSignal(str)
    rowDoubleClicked = pyqtSignal(dict)
    request_consulta_api = pyqtSignal(str, str, str, str, str)

    def __init__(self, icons, model, database_path, parent=None):
        super().__init__(parent)
        self.icons = icons
        self.model = model
        self.database_path = database_path
        self.selected_row_data = None

        # Inicializa o proxy_model e configura o filtro
        self.proxy_model = MultiColumnFilterProxyModel(self)
        self.proxy_model.setSourceModel(self.model)
        self.proxy_model.setFilterCaseSensitivity(Qt.CaseSensitivity.CaseInsensitive)

        # Configura a interface de usuário
        self.setup_ui()

    def setup_ui(self):
        # Cria o widget principal e layout principal
        self.main_widget = QWidget(self)
        self.setCentralWidget(self.main_widget)
        self.main_layout = QVBoxLayout(self.main_widget)
        title_layout = QHBoxLayout()
        label_contratos = QLabel("Planejamento", self)
        label_contratos.setStyleSheet("font-size: 20px; font-weight: bold; color: #4E648B")
        title_layout.addWidget(label_contratos)
        title_layout.addStretch()

        label_select_om = QLabel("Selecione a OM:", self)
        title_layout.addWidget(label_select_om)
        combobox_select_om = QComboBox(self)
        title_layout.addWidget(combobox_select_om)

        self.main_layout.addLayout(title_layout)
```

```
        # Layout para a barra de ferramentas
        top_layout = QHBoxLayout()
        self.search_bar = setup_search_bar(self.icons, top_layout, self.proxy_model)
        self.setup_buttons(top_layout)
        self.main_layout.addLayout(top_layout)

        self.setup_table_view()
        self.configure_table_model()
        self.adjust_columns()

    def connect_editar_dados_window(self, editar_dados_window):
        # Conecta o sinal do EditarDadosWindow ao próprio widget
        editar_dados_window.request_consulta_api.connect(self.request_consulta_api.emit)

    def on_table_double_click(self, index):
        row = self.proxy_model.mapToSource(index).row()
        id_processo = self.model.index(row, self.model.fieldIndex("id_processo")).data()

        # Carrega os dados e redefine `selected_row_data` a cada clique duplo
        self.selected_row_data = self.carregar_dados_por_id(id_processo)
        print (self.selected_row_data)
        print (id_processo)
        if self.selected_row_data:
            self.rowDoubleClicked.emit(self.selected_row_data)
        else:
                QMessageBox.warning(self, "Erro", "Falha ao carregar dados para o ID do processo
selecionado.")

    def carregar_dados_por_id(self, id_processo):
                """Carrega os dados da linha selecionada a partir do banco de dados usando
`id_processo`."""
        query = f"SELECT * FROM controle_contratos WHERE id_processo = '{id_processo}'"
        try:
            # Obtenha os dados do banco de dados
            dados = self.model.database_manager.fetch_all(query)

            # Converte para DataFrame caso dados seja uma lista
            if isinstance(dados, list):
                        dados = pd.DataFrame(dados, columns=self.model.column_names)  # Substitua
`self.model.column_names` pela lista de nomes de colunas correta

            # Verifica se o DataFrame não está vazio
            return dados.iloc[0].to_dict() if not dados.empty else None
        except Exception as e:
            print(f"Erro ao carregar dados: {e}")
            return None

    def setup_buttons(self, layout):
                    add_button("Mensagem", "mensagem", self.messageAlert, layout, self.icons,
tooltip="Adicionar um novo item")  # Alteração aqui
            add_button("API", "api", self.apiCheck, layout, self.icons, tooltip="Excluir o item
selecionado")
             add_button("Abrir Tabela", "excel", self.openTable, layout, self.icons, tooltip="Salva o
```

```
dataframe em um arquivo Excel")

    def refresh_model(self):
        """Atualiza a tabela com os dados mais recentes do banco de dados."""
        self.model.select()

    def setup_table_view(self):
        self.table_view = QTableView(self)
        self.table_view.setModel(self.proxy_model)  # Usa o proxy_model corretamente
        self.table_view.verticalHeader().setVisible(False)
        self.table_view.doubleClicked.connect(self.on_table_double_click)

        # Configurações adicionais de estilo e comportamento
        self.table_view.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)
        self.table_view.setSelectionMode(QTableView.SelectionMode.SingleSelection)

        self.table_view.setStyleSheet("""
            QTableView {
                font-size: 14px;
                padding: 4px;
                border: 1px solid #8AB4F7;
                border-radius: 6px;
                gridline-color: #3C3C5A;
            }
        """)

        # Define CenterAlignDelegate para centralizar o conteúdo em todas as colunas
        center_delegate = CenterAlignDelegate(self.table_view)
        for column in range(self.model.columnCount()):
            self.table_view.setItemDelegateForColumn(column, center_delegate)

        # # Configuração do delegate para a coluna 'dias'
        # dias_index = self.model.fieldIndex('dias')
                 #  self.table_view.setItemDelegateForColumn(dias_index,  DiasDelegate(self.icons,
self.table_view))

        # Configuração do delegate para a coluna 'situação' (ícones)
        status_icons = self.model.fieldIndex('status')
                                            self.table_view.setItemDelegateForColumn(status_icons,
CustomStatusItemDelegate(self.icons, self.table_view, self.model))

        self.main_layout.addWidget(self.table_view)

    def configure_table_model(self):
        self.proxy_model.setSortRole(Qt.ItemDataRole.UserRole)
        self.update_column_headers()
        self.hide_unwanted_columns()

    def update_column_headers(self):
        titles = {
            0: "Status", 1: "Dias", 2: "Renova?",
            4: "Contrato", 5: "Tipo", 7: "Empresa",
            8: "Objeto", 9: "Valor", 14: "OM",
```

```
            }
        for column, title in titles.items():
            self.model.setHeaderData(column, Qt.Orientation.Horizontal, title)

    def hide_unwanted_columns(self):
        visible_columns = {0, 1, 2, 4, 5, 7, 8, 9, 14}
        for column in range(self.model.columnCount()):
            if column not in visible_columns:
                self.table_view.hideColumn(column)

    def adjust_columns(self):
        # Ajustar automaticamente as larguras das colunas ao conteúdo
        self.table_view.resizeColumnsToContents()
        QTimer.singleShot(1, self.apply_custom_column_sizes)

    def apply_custom_column_sizes(self):
        header = self.table_view.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Fixed)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.Fixed)
        header.setSectionResizeMode(5, QHeaderView.ResizeMode.Fixed)
        header.setSectionResizeMode(7, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(8, QHeaderView.ResizeMode.Fixed)
        header.setSectionResizeMode(17, QHeaderView.ResizeMode.Fixed)

        header.resizeSection(0, 150)
        header.resizeSection(1, 50)
        header.resizeSection(5, 70)
        header.resizeSection(8, 200)
        header.resizeSection(17, 100)

class CenterAlignDelegate(QStyledItemDelegate):
    def initStyleOption(self, option, index):
        super().initStyleOption(option, index)
        option.displayAlignment = Qt.AlignmentFlag.AlignCenter

class CustomStatusItemDelegate(QStyledItemDelegate):
    def __init__(self, icons, parent=None, model=None):
        super().__init__(parent)
        self.icons = icons
        self.model = model

    def paint(self, painter, option, index):
        # Verifica se estamos na coluna de status
        if index.column() == self.model.fieldIndex('status'):
            status = index.data(Qt.ItemDataRole.DisplayRole)

            # Define o mapeamento de ícones
            icon_key = {
                'Planejamento': 'assinatura',
                'Reajuste': 'economy',
                'Prioritário': 'prioridade',
                'MSG Enviada': 'delivered',
                'Tá Safo!': 'like',
```

```
                'Vai garrar!': 'head_skull',
                'Nota Técnica': 'deal',
            }.get(status)

            # Desenha o ícone, se disponível
            if icon_key and icon_key in self.icons:
                icon = self.icons[icon_key]
                icon_size = 24
                icon_rect = QRect(option.rect.left() + 5,
                                  option.rect.top() + (option.rect.height() - icon_size) // 2,
                                  icon_size, icon_size)
                painter.drawPixmap(icon_rect, icon.pixmap(icon_size, icon_size))

                # Ajusta o retângulo de texto para não sobrepor o ícone
                text_rect = QRect(icon_rect.right() + 5, option.rect.top(),
                                  option.rect.width() - icon_size - 10, option.rect.height())
            else:
                # Usa o espaço completo se não houver ícone
                text_rect = option.rect

            # Configura o pincel para o texto
            # painter.setPen(text_color)
                                      painter.drawText(text_rect,   Qt.AlignmentFlag.AlignLeft   |
Qt.AlignmentFlag.AlignVCenter, status)

        else:
            # Desenha normalmente nas outras colunas
            super().paint(painter, option, index)
```