**Arquivo: search_bar.py**

```
from PyQt6.QtWidgets import QLabel, QLineEdit
from PyQt6.QtCore import QSortFilterProxyModel, Qt, QRegularExpression
from datetime import datetime

class MultiColumnFilterProxyModel(QSortFilterProxyModel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.filter_regular_expression = QRegularExpression()
        self.year_filter = ""

    def setFilterRegularExpression(self, regex):
        self.filter_regular_expression = regex
        self.invalidateFilter()  # Revalida o filtro sempre que o regex é atualizado

    def setYearFilter(self, year):
        """Novo método para definir o filtro de ano."""
        # Se o ano for "Todos", o filtro é vazio, senão é o ano selecionado
        self.year_filter = "" if year == "Todos" else year
        self.invalidateFilter() # Força a reavaliação do filtro

    def filterAcceptsRow(self, source_row, source_parent):
        """
        Verifica se uma linha deve ser exibida, checando AMBOS os filtros (ano e busca).
        """
        # --- 1. Lógica para o filtro de ANO ---
        # Busca o dado da coluna 'id_processo' (coluna 1, conforme seu modelo)
        year_index = self.sourceModel().index(source_row, 1, source_parent)
        year_data = self.sourceModel().data(year_index, Qt.ItemDataRole.DisplayRole)

        # A linha passa no teste de ano se o filtro estiver vazio ("Todos")
        # ou se o id_processo terminar com o ano selecionado.
        year_match = not self.year_filter or (year_data and year_data.endswith(self.year_filter))

        # Logica para o filtro de busca
        self.search_regex = self.filter_regular_expression

        if not self.search_regex.pattern():
            search_match = True
        else:
            # Começa assumindo que não há correspondência com a busca
            search_match = False
            # Itera sobre todas as colunas da linha
            for column in range(self.sourceModel().columnCount()):
                index = self.sourceModel().index(source_row, column, source_parent)
                data = self.sourceModel().data(index, Qt.ItemDataRole.DisplayRole)

                # Se encontrar uma correspondência, define como True e para o loop
                if data and self.search_regex.match(str(data)).hasMatch():
                    search_match = True
                            break   # Otimização: para o loop assim que encontrar a primeira
```

correspondência

```
        # --- 3. Resultado Final ---
        # A linha só é exibida se passar nos DOIS testes (year_match E search_match).
        return year_match and search_match

class ContratosMultiColumnFilterProxyModel(QSortFilterProxyModel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.filter_regular_expression = QRegularExpression()

    def setFilterRegularExpression(self, regex):
        self.filter_regular_expression = regex
        self.invalidateFilter()  # Revalida o filtro sempre que o regex é atualizado

    def filterAcceptsRow(self, source_row, source_parent):
        # Verifica o valor do filtro em cada coluna da linha
        for column in range(self.sourceModel().columnCount()):
            index = self.sourceModel().index(source_row, column, source_parent)
            data = self.sourceModel().data(index, Qt.ItemDataRole.DisplayRole)
            if data is not None:
                data_str = str(data)
                if self.filter_regular_expression.match(data_str).hasMatch():
                    return True  # Mostra a linha se houver correspondência em qualquer coluna
        return False  # Oculta a linha se não houver correspondência em nenhuma coluna

    def lessThan(self, left, right):
        """Sobrescreve a comparação padrão para a coluna `vigencia_final`."""
        left_data = self.sourceModel().data(left, Qt.ItemDataRole.DisplayRole)
        right_data = self.sourceModel().data(right, Qt.ItemDataRole.DisplayRole)

        # Verifica se estamos na coluna `vigencia_final`
        column = left.column()
        if column == self.sourceModel().fieldIndex("vigencia_final"):
            left_date = self._parse_date(left_data)
            right_date = self._parse_date(right_data)

            # Coloca valores inválidos ou NULL no final
            if left_date is None and right_date is None:
                return False
            if left_date is None:
                return False
            if right_date is None:
                return True

            # Compara datas válidas em ordem DESCENDENTE
            return left_date > right_date  # Note que invertemos o operador para ordem descendente

        # Comparação padrão para outras colunas
        return super().lessThan(left, right)

    @staticmethod
    def _parse_date(date_str):
```

```
        """Tenta converter uma string de data no formato YYYY-MM-DD."""
        if not date_str:
            return None
        try:
            return datetime.strptime(date_str, "%Y-%m-%d")
        except ValueError:
            return None



def on_search_text_changed(text, proxy_model):
    """
    Atualiza o filtro do proxy_model com base no texto da barra de pesquisa.

    :param text: Texto inserido na barra de pesquisa.
    :param proxy_model: O modelo proxy que será filtrado com base no texto.
    """
    regex = QRegularExpression(text, QRegularExpression.PatternOption.CaseInsensitiveOption)
    proxy_model.setFilterRegularExpression(regex)

def setup_search_bar(icons, layout, proxy_model):
    search_label = QLabel()
    search_label.setPixmap(icons["magnifying-glass"].pixmap(30, 30))  # Ícone de tamanho 20x20
    layout.addWidget(search_label)
    # search_label = QLabel("Localizar:")
    # search_label.setStyleSheet("""
    #     color: #8AB4F7;
    #     font-size: 14px;
    #     font-weight: bold;
    #     margin-right: 10px;
    # """)
    # layout.addWidget(search_label)

    search_bar = QLineEdit()
    search_bar.setPlaceholderText("Digite para buscar...")
    # search_bar.setStyleSheet("""
    #     QLineEdit {
    #         background-color: #13141F;
    #         color: #8AB4F7;
    #         font-size: 14px;
    #         font-weight: bold;
    #         padding: 8px;
    #         border: 1px solid #8AB4F7;
    #         border-radius: 5px;
    #     }
    #     QLineEdit:focus {
    #         border: 1px solid #8AB4F7;
    #         background-color: #181928;
    #         color: #FFFFFF;
    #         border-radius: 5px;
    #     }
    # """)
    search_bar.textChanged.connect(lambda text: on_search_text_changed(text, proxy_model))
```

```
layout.addWidget(search_bar)

return search_bar
```