```
import pandas as pd
import streamlit as st

import modules.db_manager as db_manager  # Importar el gestor de base de datos
from modules.config import Config

# Identificador de usuario (temporalmente genérico, idealmente vendría de un login)
CURRENT_USER = "default_user"


def create_sidebar(gdf_stations, df_long):
    with st.sidebar:
        # --- LOGO Y TÍTULO ---
        if hasattr(Config, "LOGO_PATH"):
            try:
                st.image(Config.LOGO_PATH, width=150)
            except:
                pass

        st.title("Panel de Control")

        # --- 1. FILTROS DE PROCESAMIENTO ---
        with st.expander(" 🛠️ Procesamiento y Calidad", expanded=False):
            run_complete_series = st.checkbox(
                "Interpolación (Rellenar huecos)", value=False
            )
            exclude_nulls = st.checkbox("Excluir datos nulos (NaN)", value=False)
            exclude_zeros = st.checkbox("Excluir valores cero (0)", value=False)

            st.markdown("---")

            # Filtro por % de datos
            min_pct = st.slider(
                "Mínimo % de Datos Disponibles:",
                0,
                100,
                0,
                help="Filtra estaciones que tengan al menos este porcentaje de datos en el histórico.",
            )

            # Manejo de estado para interpolación
            if run_complete_series != st.session_state.get("apply_interpolation"):
                st.session_state["apply_interpolation"] = run_complete_series
                st.rerun()

        st.divider()
```

```
# --- 2. FILTROS DE UBICACIÓN ---
st.markdown("### 📍 Filtros de Ubicación")

# A. Lógica de Filtrado por Porcentaje de Datos
valid_stations_by_pct = gdf_stations[Config.STATION_NAME_COL].unique()

if min_pct > 0 and df_long is not None:
    # Calcular porcentaje real basado en los datos cargados
    counts = df_long.groupby(Config.STATION_NAME_COL)[
        Config.PRECIPITATION_COL
    ].count()

    # Calcular total teórico de meses (Años * 12)
    if not df_long.empty:
        n_years = (
            df_long[Config.YEAR_COL].max() - df_long[Config.YEAR_COL].min() + 1
        )
        total_months = n_years * 12
        pcts = (counts / total_months) * 100
        valid_stations_by_pct = pcts[pcts >= min_pct].index.tolist()

# B. Filtro por Altitud
altitude_options = [
    "Todos",
    "0-500",
    "500-1000",
    "1000-1500",
    "1500-2000",
    "2000-3000",
    ">3000",
]
selected_alt_range = st.selectbox("Filtrar por Altitud (m):", altitude_options)

# Aplicar filtros base (Estaciones válidas por % y luego Altitud)
gdf_filtered_base = gdf_stations[
    gdf_stations[Config.STATION_NAME_COL].isin(valid_stations_by_pct)
].copy()

if selected_alt_range != "Todos":
    if ">" in selected_alt_range:
        min_alt = int(selected_alt_range.replace(">", ""))
        gdf_filtered_base = gdf_filtered_base[
            gdf_filtered_base[Config.ALTITUDE_COL] >= min_alt
        ]
    else:
```

```
        try:
            min_alt, max_alt = map(int, selected_alt_range.split("-"))
            gdf_filtered_base = gdf_filtered_base[
                (gdf_filtered_base[Config.ALTITUDE_COL] >= min_alt)
                & (gdf_filtered_base[Config.ALTITUDE_COL] < max_alt)
            ]
        except:
            pass  # Manejo de errores si el formato del string falla

# C. Región (CON PERSISTENCIA)
selected_regions = []
if Config.REGION_COL in gdf_filtered_base.columns:
    all_regions = sorted(
        gdf_filtered_base[Config.REGION_COL].astype(str).unique()
    )

    # Recuperar preferencia guardada
    saved_regions = db_manager.get_user_preference(
        CURRENT_USER, "selected_regions", []
    )

    # Validar que las regiones guardadas sigan existiendo en los datos actuales
    if isinstance(saved_regions, list):
        valid_saved = [r for r in saved_regions if r in all_regions]
    else:
        valid_saved = []

    # Widget Multiselect con valor por defecto recuperado
    selected_regions = st.multiselect(
        "Región:", all_regions, default=valid_saved
    )

    # Guardar si hay cambios (comparando conjuntos para ignorar orden)
    if set(selected_regions) != set(valid_saved):
        db_manager.save_user_preference(
            CURRENT_USER, "selected_regions", selected_regions
        )

    if selected_regions:
        gdf_filtered_base = gdf_filtered_base[
            gdf_filtered_base[Config.REGION_COL].isin(selected_regions)
        ]

# D. Municipio
selected_municipios = []
if Config.MUNICIPALITY_COL in gdf_filtered_base.columns:
```

```
    all_munis = sorted(
        gdf_filtered_base[Config.MUNICIPALITY_COL].astype(str).unique()
    )
    selected_municipios = st.multiselect("Municipio:", all_munis)
    if selected_municipios:
        gdf_filtered_base = gdf_filtered_base[
            gdf_filtered_base[Config.MUNICIPALITY_COL].isin(selected_municipios)
        ]

# E. Selección Final de Estaciones
available_stations = sorted(
    gdf_filtered_base[Config.STATION_NAME_COL].astype(str).unique()
)

with st.expander(
    f"Estaciones ({len(available_stations)} disp.)", expanded=True
):
    select_all_stations = st.checkbox(
        "Seleccionar Todas las visibles", key="sel_all_st"
    )

    if select_all_stations:
        default_stations = available_stations
        if len(available_stations) > 50:
            st.caption(
                "⚠️ Muchas estaciones seleccionadas. El rendimiento puede variar."
            )
    else:
        # Por defecto seleccionar las primeras 3 si no se eligen todas
        default_stations = (
            available_stations[:3] if len(available_stations) > 0 else []
        )

    stations_for_analysis = st.multiselect(
        "Seleccione específicas:",
        options=available_stations,
        default=default_stations,
        label_visibility="collapsed",
    )

# Crear GDF final basado en la selección
gdf_final = gdf_stations[
    gdf_stations[Config.STATION_NAME_COL].isin(stations_for_analysis)
]

st.divider()
```

```
# ==============================================================================
# INICIO DEL CAMBIO: Encerramos el tiempo en un FORMULARIO para evitar bloqueos
# ==============================================================================
with st.sidebar.form(key="formulario_tiempo"):

    # --- 3. FILTRO DE TIEMPO ---
    st.markdown("### 🗓️ Periodo Temporal")
    try:
        min_y = int(df_long[Config.YEAR_COL].min())
        max_y = int(df_long[Config.YEAR_COL].max())
        # El slider ahora vive dentro del form, no recargará la página al moverlo
        year_range = st.slider("Años:", min_y, max_y, (max_y - 10, max_y))
    except:
        year_range = (1980, 2020)  # Fallback por seguridad

    # --- 4. FILTRO DE MESES ---
    st.markdown("### 🗓️ Análisis Estacional")

    meses_nombres = [
        "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
        "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre",
    ]

    # Checkbox para seleccionar todos
    sel_all_months = st.checkbox("Seleccionar todos los meses", value=True, key="sel_all_months")

    if sel_all_months:
        default_months = meses_nombres
    else:
        default_months = []

    selected_months = st.multiselect(
        "Meses a incluir:",
        options=meses_nombres,
        default=default_months,
        help="Seleccione los meses que desea incluir en el análisis.",
    )

    st.markdown("---")
    # ESTE ES EL BOTÓN CLAVE: Nada se procesa hasta que hagas clic aquí
    boton_aplicar = st.form_submit_button("🔄 Actualizar Gráficos y Mapa", type="primary")

# ==============================================================================
# FIN DEL FORMULARIO - A partir de aquí el código sigue normal
```

```
# ==============================================================================

# Mapear nombres a números (1-12) fuera del form para el procesamiento
mapa_meses = {m: i + 1 for i, m in enumerate(meses_nombres)}
selected_months_nums = [mapa_meses[m] for m in selected_months]

# --- 5. APLICACIÓN DE FILTROS AL DATAFRAME ---

# A. Filtro de Años y Estaciones
mask = (
    (df_long[Config.YEAR_COL] >= year_range[0])
    & (df_long[Config.YEAR_COL] <= year_range[1])
    & (df_long[Config.STATION_NAME_COL].isin(stations_for_analysis))
)
df_temp = df_long.loc[mask].copy()

# B. Filtro de Meses
if selected_months_nums:
    mask_mes = df_temp[Config.MONTH_COL].isin(selected_months_nums)
    df_monthly_filtered = df_temp.loc[mask_mes].copy()
else:
    # Si no hay meses seleccionados, DataFrame vacío con estructura correcta
    df_monthly_filtered = pd.DataFrame(columns=df_long.columns)

# C. Filtros de Calidad (Nulos y Ceros)
if exclude_nulls:
    df_monthly_filtered = df_monthly_filtered.dropna(
        subset=[Config.PRECIPITATION_COL]
    )
if exclude_zeros:
    df_monthly_filtered = df_monthly_filtered[
        df_monthly_filtered[Config.PRECIPITATION_COL] != 0
    ]

# D. Agrupación Anual
if not df_monthly_filtered.empty:
    df_anual_melted = (
        df_monthly_filtered.groupby([Config.STATION_NAME_COL, Config.YEAR_COL])[
            Config.PRECIPITATION_COL
        ]
        .sum()
        .reset_index()
    )
else:
    df_anual_melted = pd.DataFrame(
        columns=[
```

```
            Config.STATION_NAME_COL,
            Config.YEAR_COL,
            Config.PRECIPITATION_COL,
        ]
    )

# --- BOTÓN LIMPIAR CACHÉ ---
st.divider()
if st.button("🧹 Limpiar Caché y Recargar"):
    st.cache_data.clear()
    st.rerun()

# Determinamos el string de interpolación para el resumen
str_interpolacion = "Si" if run_complete_series else "No"

# RETORNO
return (
    stations_for_analysis,
    df_anual_melted,
    df_monthly_filtered,
    gdf_final,
    str_interpolacion,
    selected_regions,
    selected_municipios,
    selected_months_nums,
    year_range,
)
```