import streamlit as st import pandas as pd import mysql.connector import os import plotly.express as px from datetime import datetime import io from price_updaters import update_insumo_prices, update_competitor_prices from snapshot_creator import create_financial_snapshot from db_connection import connect_db

Configuración de la página

st.set_page_config(page_title="Atomick - Sistema de Actualización y Snapshot Financiero", layout="wide")

Título y descripción

st.title("Sistema de Actualización y Snapshot Financiero") st.markdown("Actualiza precios de insumos y competencia, y crea snapshots financieros para análisis histórico.")

Crear conexión

@st.cache_resource def get_connection(): return connect_db()

Función para mostrar datos

def show_data_preview(df, title): st.subheader(title) st.dataframe(df.head(10))

Sidebar con opciones

st.sidebar.header("Opciones") option = st.sidebar.radio("Selecciona una opción:", ("Ver Datos", "Actualizar Precios", "Crear Snapshot", "Ver Historial"))

Variable para mensaje de estado

status_placeholder = st.empty()

Vista principal basada en la opción seleccionada

if option == "Ver Datos": st.header("Visualización de Datos")

```
# Conectar a la base de datos
conn = get_connection()
if conn and conn.is_connected():
    try:
        # Selector de tablas
        table_option = st.selectbox(
```

```
            "Selecciona una tabla para visualizar:",
            ("INSUMOS", "PLATOS", "V_PLATOS_FINANCIALS")
        )

        # Consulta según la tabla seleccionada
        cursor = conn.cursor(dictionary=True)
        if table_option == "INSUMOS":
            cursor.execute("SELECT * FROM INSUMOS LIMIT 100")
        elif table_option == "PLATOS":
            cursor.execute("SELECT * FROM PLATOS LIMIT 100")
        else:
            cursor.execute("SELECT * FROM V_PLATOS_FINANCIALS LIMIT 100")

        # Obtener resultados
        results = cursor.fetchall()
        cursor.close()

        # Convertir a DataFrame y mostrar
        if results:
            df = pd.DataFrame(results)
            st.dataframe(df)

            # Exportar a CSV
            csv = df.to_csv(index=False).encode('utf-8')
            st.download_button(
                label="Descargar como CSV",
                data=csv,
                file_name=f"{table_option.lower()}_export.csv",
                mime="text/csv",
            )
        else:
            st.warning(f"No hay datos disponibles en {table_option}")

    except Exception as e:
        st.error(f"Error al consultar datos: {str(e)}")
else:
    st.error("No se pudo conectar a la base de datos.")
```

elif option == "Actualizar Precios": st.header("Actualización de Precios")

```
update_tab1, update_tab2 = st.tabs(["Actualizar Insumos", "Actualizar Precios
Competencia"])

with update_tab1:
    st.subheader("Actualizar Precios de Insumos")

    # Opción de cargar archivo o usar plantilla
    upload_method = st.radio(
        "Método de carga:",
```

```
        ("Cargar archivo CSV", "Completar manualmente")
    )

    if upload_method == "Cargar archivo CSV":
        uploaded_file = st.file_uploader("Cargar archivo CSV de insumos", type=
["csv"])

        if uploaded_file is not None:
            try:
                df_insumos = pd.read_csv(uploaded_file)
                if not all(col in df_insumos.columns for col in ['ID_Insumo',
'Nuevo_Costo_Compra', 'Nueva_Unidad_Compra']):
                    st.error("El archivo debe contener las columnas: ID_Insumo,
Nuevo_Costo_Compra, Nueva_Unidad_Compra")
                else:
                    show_data_preview(df_insumos, "Vista previa de datos")

                    # Guardar temporalmente para procesamiento
                    temp_path = "4-Brief/Gemini/nuevos_precios_insumos.csv"
                    os.makedirs(os.path.dirname(temp_path), exist_ok=True)
                    df_insumos.to_csv(temp_path, index=False)

                    if st.button("Actualizar Precios de Insumos"):
                        conn = get_connection()
                        if conn and conn.is_connected():
                            with st.spinner("Actualizando precios de insumos..."):
                                result = update_insumo_prices(conn)
                                if result:
                                    st.success("Precios de insumos actualizados
correctamente")
                                else:
                                    st.error("Error al actualizar precios de
insumos")
                        else:
                            st.error("No se pudo conectar a la base de datos")
            except Exception as e:
                st.error(f"Error al procesar el archivo: {str(e)}")
    else:
        # Interfaz para crear datos manualmente
        st.write("Ingresa los datos de los insumos a actualizar:")

        # Número de filas a crear
        num_rows = st.number_input("Número de insumos", min_value=1, max_value=20,
value=3)

        # Crear un dataframe vacío
        data = {
            "ID_Insumo": [""] * num_rows,
            "Nuevo_Costo_Compra": [0.0] * num_rows,
            "Nueva_Unidad_Compra": [""] * num_rows
        }
```

```
        # Interfaz para editar los datos
        edited_df = st.data_editor(
            pd.DataFrame(data),
            use_container_width=True,
            hide_index=True,
            num_rows=num_rows
        )

        if st.button("Actualizar con Datos Ingresados"):
            # Validar datos
            if edited_df["ID_Insumo"].isna().any() or (edited_df["ID_Insumo"] ==
"").any():
                st.error("Todos los ID_Insumo deben estar completos")
            else:
                # Guardar temporalmente para procesamiento
                temp_path = "4-Brief/Gemini/nuevos_precios_insumos.csv"
                os.makedirs(os.path.dirname(temp_path), exist_ok=True)
                edited_df.to_csv(temp_path, index=False)

                conn = get_connection()
                if conn and conn.is_connected():
                    with st.spinner("Actualizando precios de insumos..."):
                        result = update_insumo_prices(conn)
                        if result:
                            st.success("Precios de insumos actualizados
correctamente")
                        else:
                            st.error("Error al actualizar precios de insumos")
                else:
                    st.error("No se pudo conectar a la base de datos")

with update_tab2:
    st.subheader("Actualizar Precios de Competencia")

    # Opción de cargar archivo o usar plantilla
    upload_method = st.radio(
        "Método de carga:",
        ("Cargar archivo Excel", "Completar manualmente"),
        key="competitor_upload_method"
    )

    if upload_method == "Cargar archivo Excel":
        uploaded_file = st.file_uploader("Cargar archivo Excel de precios
competencia", type=["xlsx", "xls"])

        if uploaded_file is not None:
            try:
                df_competencia = pd.read_excel(uploaded_file)
                if not all(col in df_competencia.columns for col in ['ID_Plato',
'Precio_Competencia_Nuevo']):
```

```
                    st.error("El archivo debe contener las columnas: ID_Plato,
Precio_Competencia_Nuevo")
                else:
                    show_data_preview(df_competencia, "Vista previa de datos")

                    # Guardar temporalmente para procesamiento
                    temp_path = "4-Brief/Gemini/precios_competencia.xlsx"
                    os.makedirs(os.path.dirname(temp_path), exist_ok=True)
                    df_competencia.to_excel(temp_path, index=False)

                    if st.button("Actualizar Precios de Competencia"):
                        conn = get_connection()
                        if conn and conn.is_connected():
                            with st.spinner("Actualizando precios de
competencia..."):
                                result = update_competitor_prices(conn)
                                if result:
                                    st.success("Precios de competencia
actualizados correctamente")
                                else:
                                    st.error("Error al actualizar precios de
competencia")
                        else:
                            st.error("No se pudo conectar a la base de datos")
            except Exception as e:
                st.error(f"Error al procesar el archivo: {str(e)}")
    else:
        # Interfaz para crear datos manualmente
        st.write("Ingresa los datos de los platos a actualizar:")

        # Número de filas a crear
        num_rows = st.number_input("Número de platos", min_value=1, max_value=20,
value=3, key="num_competitor_rows")

        # Crear un dataframe vacío
        data = {
            "ID_Plato": [""] * num_rows,
            "Precio_Competencia_Nuevo": [0.0] * num_rows
        }

        # Interfaz para editar los datos
        edited_df = st.data_editor(
            pd.DataFrame(data),
            use_container_width=True,
            hide_index=True,
            num_rows=num_rows
        )

        if st.button("Actualizar con Datos Ingresados",
key="update_competitor_btn"):
            # Validar datos
```

```
            if edited_df["ID_Plato"].isna().any() or (edited_df["ID_Plato"] ==
"").any():
                st.error("Todos los ID_Plato deben estar completos")
            else:
                # Guardar temporalmente para procesamiento
                temp_dir = "4-Brief/Gemini"
                os.makedirs(temp_dir, exist_ok=True)
                temp_path = f"{temp_dir}/precios_competencia.xlsx"
                edited_df.to_excel(temp_path, index=False)

                conn = get_connection()
                if conn and conn.is_connected():
                    with st.spinner("Actualizando precios de competencia..."):
                        result = update_competitor_prices(conn)
                        if result:
                            st.success("Precios de competencia actualizados
correctamente")
                        else:
                            st.error("Error al actualizar precios de competencia")
                else:
                    st.error("No se pudo conectar a la base de datos")
```

elif option == "Crear Snapshot": st.header("Creación de Snapshot Financiero")

```
st.markdown("""
Esta operación consulta la vista `V_PLATOS_FINANCIALS`, que contiene los cálculos
financieros actualizados,
y guarda un registro histórico en la tabla `PLATOS_FINANCIALS_HISTORY`.
""")

# Información de los últimos snapshots
conn = get_connection()
if conn and conn.is_connected():
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("""
            SELECT
                DATE(SnapshotTimestamp) as Fecha,
                COUNT(*) as TotalRegistros,
                MIN(Porcentaje_Margen_Bruto_PctMBA_Hist) as MinMargen,
                AVG(Porcentaje_Margen_Bruto_PctMBA_Hist) as PromMargen,
                MAX(Porcentaje_Margen_Bruto_PctMBA_Hist) as MaxMargen
            FROM PLATOS_FINANCIALS_HISTORY
            GROUP BY DATE(SnapshotTimestamp)
            ORDER BY Fecha DESC
            LIMIT 5
        """)
        results = cursor.fetchall()
        cursor.close()
```

```
        if results:
            st.subheader("Últimos 5 Snapshots")
            df_snapshots = pd.DataFrame(results)
            # Formatear columnas numéricas
            for col in ['MinMargen', 'PromMargen', 'MaxMargen']:
                df_snapshots[col] = df_snapshots[col].map('{:.2f}%'.format)
            st.dataframe(df_snapshots)

    except Exception as e:
        st.warning(f"No se pudieron recuperar datos de snapshots previos:
{str(e)}")

# Botón para crear snapshot
if st.button("Crear Nuevo Snapshot Financiero", type="primary"):
    conn = get_connection()
    if conn and conn.is_connected():
        with st.spinner("Creando snapshot financiero..."):
            result = create_financial_snapshot(conn)
            if result:
                st.success("Snapshot financiero creado correctamente")
                st.balloons()  # Efecto visual de éxito
            else:
                st.error("Error al crear snapshot financiero")
    else:
        st.error("No se pudo conectar a la base de datos")
```

elif option == "Ver Historial": st.header("Historial de Snapshots Financieros")

```
# Conectar a la base de datos
conn = get_connection()
if conn and conn.is_connected():
    cursor = conn.cursor(dictionary=True)

    # Obtener fechas disponibles para filtrar
    try:
        cursor.execute("SELECT DISTINCT DATE(SnapshotTimestamp) as Fecha FROM
PLATOS_FINANCIALS_HISTORY ORDER BY Fecha DESC")
        fechas = [row["Fecha"] for row in cursor.fetchall()]

        if not fechas:
            st.warning("No hay snapshots históricos disponibles")
        else:
            # Crear filtros
            col1, col2 = st.columns(2)
            with col1:
                fecha_seleccionada = st.selectbox("Selecciona una fecha:", fechas)

            with col2:
```

```
                # Obtener platos disponibles para esa fecha
                cursor.execute("""
                    SELECT DISTINCT ID_Plato
                    FROM PLATOS_FINANCIALS_HISTORY
                    WHERE DATE(SnapshotTimestamp) = %s
                """, (fecha_seleccionada,))
                platos = [row["ID_Plato"] for row in cursor.fetchall()]
                plato_seleccionado = st.selectbox("Filtrar por plato (opcional):",
["Todos"] + platos)

            # Cargar datos según filtro
            if plato_seleccionado == "Todos":
                cursor.execute("""
                    SELECT
                        h.SnapshotTimestamp, h.ID_Plato,
                        p.Nombre_Plato,
                        h.Costo_Plato_Hist, h.Precio_Competencia_Hist,
                        h.PBA_Hist, h.PNA_Hist, h.COGS_Partner_Actual_Hist,
                        h.Costo_Total_CT_Hist, h.Margen_Bruto_Actual_MBA_Hist,
                        h.Porcentaje_Margen_Bruto_PctMBA_Hist,
                        h.Market_Discount_Used, h.IVA_Rate_Used,
h.Commission_Rate_Used
                    FROM PLATOS_FINANCIALS_HISTORY h
                    LEFT JOIN PLATOS p ON h.ID_Plato = p.ID_Plato
                    WHERE DATE(h.SnapshotTimestamp) = %s
                    ORDER BY h.Porcentaje_Margen_Bruto_PctMBA_Hist DESC
                """, (fecha_seleccionada,))
            else:
                cursor.execute("""
                    SELECT
                        h.SnapshotTimestamp, h.ID_Plato,
                        p.Nombre_Plato,
                        h.Costo_Plato_Hist, h.Precio_Competencia_Hist,
                        h.PBA_Hist, h.PNA_Hist, h.COGS_Partner_Actual_Hist,
                        h.Costo_Total_CT_Hist, h.Margen_Bruto_Actual_MBA_Hist,
                        h.Porcentaje_Margen_Bruto_PctMBA_Hist,
                        h.Market_Discount_Used, h.IVA_Rate_Used,
h.Commission_Rate_Used
                    FROM PLATOS_FINANCIALS_HISTORY h
                    LEFT JOIN PLATOS p ON h.ID_Plato = p.ID_Plato
                    WHERE DATE(h.SnapshotTimestamp) = %s AND h.ID_Plato = %s
                """, (fecha_seleccionada, plato_seleccionado))

            results = cursor.fetchall()

            if results:
                # Convertir a DataFrame
                df_history = pd.DataFrame(results)

                # Mostrar datos en tabla
                st.subheader("Datos del Snapshot")
```

```
                st.dataframe(df_history)

                # Descargar como Excel
                output = io.BytesIO()
                with pd.ExcelWriter(output, engine='xlsxwriter') as writer:
                    df_history.to_excel(writer, sheet_name='Snapshot',
index=False)
                    writer.close()

                binary_data = output.getvalue()
                st.download_button(
                    label="Descargar como Excel",
                    data=binary_data,
                    file_name=f"snapshot_{fecha_seleccionada}.xlsx",
                    mime="application/vnd.ms-excel"
                )

                # Visualizaciones
                st.subheader("Visualizaciones")
                tab1, tab2 = st.tabs(["Margen Bruto", "Costos vs Precios"])

                with tab1:
                    # Gráfico de margen bruto por plato
                    fig = px.bar(

df_history.sort_values('Porcentaje_Margen_Bruto_PctMBA_Hist',
ascending=False).head(10),
                        x='ID_Plato', y='Porcentaje_Margen_Bruto_PctMBA_Hist',
                        title="Top 10 Platos por Margen Bruto (%)",
                        labels={'Porcentaje_Margen_Bruto_PctMBA_Hist': 'Margen
Bruto (%)', 'ID_Plato': 'Plato'},
                        color='Porcentaje_Margen_Bruto_PctMBA_Hist',
                        color_continuous_scale=px.colors.sequential.Blues,
                        text_auto='.1f'
                    )
                    st.plotly_chart(fig, use_container_width=True)

                with tab2:
                    # Gráfico de costos vs precios
                    fig = px.scatter(
                        df_history,
                        x='Costo_Plato_Hist', y='Precio_Competencia_Hist',
                        hover_name='ID_Plato',
                        size='Margen_Bruto_Actual_MBA_Hist',
                        color='Porcentaje_Margen_Bruto_PctMBA_Hist',
                        color_continuous_scale=px.colors.sequential.Viridis,
                        title="Relación Costo vs Precio de Competencia",
                        labels={
                            'Costo_Plato_Hist': 'Costo del Plato',
                            'Precio_Competencia_Hist': 'Precio de Competencia'
                        }
```

```
                    )
                    st.plotly_chart(fig, use_container_width=True)
            else:
                st.info("No hay datos disponibles para los filtros seleccionados")

    except Exception as e:
        st.error(f"Error al consultar historial: {str(e)}")

    cursor.close()
else:
    st.error("No se pudo conectar a la base de datos")
```

Footer

st.markdown("---") st.markdown("

", unsafe_allow_html=True)