```
import streamlit as st
from dotenv import load_dotenv
# Import the main LangGraph workflow entry point
from agentic_workflow import get_workflow
import asyncio
from langchain_core.tracers.context import collect_runs
# LangSmith Client enables feedback tracking and run tracing (for evaluation, debugging)
from langsmith import Client
from streamlit_feedback import streamlit_feedback
from functools import partial
# Traceback is used to print the full traceback of an error
import traceback

# Load environment variables
load_dotenv()
try:
    client = Client()
    print("LangSmith Client Initialized Successfully.")
except Exception as e:
    # Warn once during startup if client initialization fails
    st.warning(f"Could not initialize LangSmith client. Feedback submission may not work. Er
    print(f"LangSmith Client Initialization Failed: {e}")
    client = None # Set client to None if initialization fails

# Page config
st.set_page_config(
    page_title='Drucker Management Assistant',
    page_icon=" ",
    layout="centered"
)

# Main title
st.title("  Drucker's Management Assistant")

# Create the sidebar section
with st.sidebar:
    st.title("Drucker's Management Wisdom") # Sidebar title to anchor the app's purpose

    # Informational section about the assistant's data sources and usage examples
    st.markdown("""
    ##   Drucker's Knowledge Base
    This assistant is powered by the following Peter Drucker books:

    - **The Daily Drucker (2004)** - 366 daily insights on management, leadership, and inno
    - **The Effective Executive (2002)** - Habits and practices for executive effectiveness
    - **The Essential Drucker (2008)** - Curated collection of Drucker's foundational princi
    ##  Example Questions
    - What did Drucker say about knowledge workers?
    - What are Drucker's views on innovation?
    - Who is Peter Drucker?

    **The assistant can also search the web for additional information related to Peter Druc
    """)
    st.markdown("""----""")

    # Refresh button
    if st.button("New Conversation  ", use_container_width=True):

        # Clear chat history
        st.session_state.chat_history = []
        # Remove all feedback keys from session state
        # If these feedback keys weren't cleared, they could incorrectly map to new message
        keys_to_delete = [key for key in st.session_state.keys() if key.startswith("feedbac
        for key in keys_to_delete:
            del st.session_state[key]

        # Restart the app to show a fresh interface
        st.rerun()

# Initialize chat history
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []

# --- Feedback Submission Function ---
def submit_feedback(user_response, run_id, client):
    """
    Submits thumbs-up/down feedback to LangSmith for a given run ID.
    This helps evaluate the performance of the LLM workflow after user interaction.
    """

    # If client is None (init failed) or run_id is missing, skip silently.
    if not client or not run_id:
        print(f"Debug (submit_feedback skipped): Client available: {client is not None}, Run
        return

    try:
        # Map emoji score to numeric value expected by LangSmith
        score_map = {" ": 1, " ": 0}
        score = score_map.get(user_response.get("score"))
        comment = user_response.get("text")
        # Only submit feedback if there's a valid thumbs-up or thumbs-down
        # Currently, the client automatically associates the feedback with this LangSmith ac
        # If you want to associate feedback with a different LangSmith account, you can do s
        # to a different value (e.g., a user ID or email)
        if score is not None:
            feedback_result = client.create_feedback(
                run_id=run_id,
                key="user_thumb_feedback",       # Descriptive key
                score=score,                     # Numeric score (1 or 0)
                comment=comment,                 # Optional user explanation
                value=user_response.get("score") # Store the emoji
            )
            print(f"Feedback submitted: Run ID: {run_id}, Score: {score}, Comment: {comment}
        else:
            # Skip submission if user didn't click thumbs-up or down
             print(f"Feedback skipped (no score): Run ID: {run_id}, Response: {user_response

    except Exception as e:
        # If LangSmith API fails, show error in UI and log the exception
        st.error(f"Failed to submit feedback to LangSmith: {e}")
        print(f"Error submitting feedback: Run ID: {run_id}, Exception: {e}")

# --- Async function to get LLM response AND LangSmith run_id from the workflow ---
async def get_drucker_response_with_run_id(user_input):
    """
    Executes the LangGraph workflow with the user's input and captures:
    1. The AI-generated response.
    2. The LangSmith run ID (for feedback/tracing).

    This function supports streaming and error handling for feedback and traceability.
    """
    # Retrieve and compile the LangGraph workflow
    graph = get_workflow().compile()    # Cached if already built
    final_state = None                  # Holds the last emitted state from the graph
    run_id = None                        # LangSmith run ID (used for tracing/feedback)

    # Default response in case the graph doesn't yield any usable output
    ai_response_content = "I'm sorry, I couldn't find a good answer. Could you try rephrasin

    # Collect trace info from LangSmith
    with collect_runs() as cb:
        try:
            # Stream values emitted by the graph
            async for event in graph.astream(
                {"question": user_input},
                stream_mode="values",
                config={"tags": ["streamlit_app_call"]}
                ):
                final_state = event # Update the final_state as receiving new events

            # Extract generated response if present in the final state
            if final_state and "generation" in final_state:
                 ai_response_content = final_state["generation"]
            else:
                 print(f"Final state missing 'generation': {final_state}")

            # Extract LangSmith run ID for feedback tracking
            if cb.traced_runs:
                run_id = str(cb.traced_runs[-1].id) # LangSmith run ID will be created auto
            else:
                 print("Warning: No runs traced by collect_runs.")

        except Exception as e:
            # On error, return fallback message and log the full exception
            ai_response_content = f"An error occurred while processing your request. Please
            traceback.print_exc() # Log full traceback for backend errors

    # Return both the response text and the tracing run ID
    return ai_response_content, run_id

# --- Display Chathistory with Optional Feedback ---

# A helper function to create a standardized feedback widget with consistent configuration.
def create_feedback_widget(feedback_key, run_id, client, disable_with_score=None):
    """
    Creates a standardized feedback widget with consistent configuration.

    Args:
        feedback_key (str): Unique key for the widget instance.
        run_id (str): Identifier for the current run.
        client (Any): Client object used for submitting feedback.
        disable_with_score (bool, optional): If True, disables widget when a score is prese

    Returns:
        Feedback widget instance rendered via streamlit_feedback.
    """

    # Return a pre-configured feedback widget
    return streamlit_feedback(
        feedback_type="thumbs",                     # "thumbs" allows thumbs-up/down input
        optional_text_label="Provide feedback",     # Encourages free-text feedback
        key=feedback_key,
        on_submit=partial(submit_feedback, run_id=run_id, client=client),
        kwargs={"run_id": run_id, "client": client}, # asses extra metadata to the feedback
        disable_with_score=disable_with_score,
    )

# Chat Display and Feedback UI
# Loop over each message in the stored chat history (new format)
for i, message in enumerate(st.session_state.chat_history):
    role = message["role"]           # 'user' or 'ai'
    content = message["content"]     # Message text
    run_id = message.get("run_id")  # LangSmith run ID for this response (if any)

    # Use Streamlit's chat UI block to render the message
    with st.chat_message(role):
        st.markdown(content)

    # If this is an AI message and has a run ID, allow feedback as
    if role == "ai" and run_id:
        feedback_key = f"feedback_{i}" # Unique key per AI message

        # Initialize feedback state if not already present
        if feedback_key not in st.session_state:
            st.session_state[feedback_key] = None

        # Check if feedback has already been submitted (disable if so)
        current_feedback = st.session_state.get(feedback_key)
        score_to_disable_with = current_feedback.get("score") if current_feedback else None

        # Show thumbs-up/down feedback widget using helper function
        create_feedback_widget(feedback_key, run_id, client, score_to_disable_with)

    # Display warning only if run_id is missing for an AI message
    elif role == "ai" and not run_id:
         st.warning("Feedback not available for this message (missing run ID).", icon=" ")


# Initial greeting if chat history is empty
if not st.session_state.chat_history:
    with st.chat_message("ai"):
        st.write("Hello, I'm your Drucker Management Assistant. How can I help you today?  "

# User Input Processing
# Chat input
user_query = st.chat_input("Ask about Drucker's management philosophy...")
# Process user input
```

```
if user_query is not None and user_query != "":
    st.session_state.chat_history.append({"role": "human", "content": user_query})

    # Display User Input
    with st.chat_message("human"):
        st.markdown(user_query)

    # AI Response Generation
    with st.chat_message("ai"):
        message_placeholder = st.empty()
        run_id = None

        # Display Loading Indicator
        with message_placeholder.status("Consulting Drucker's wisdom..."):
            try:
                ai_response_content, run_id = asyncio.run(get_drucker_response_with_run_id(u
            except Exception as e:
                 st.error(f"Error generating response: {e}")
                 print(f"Error in asyncio.run(get_drucker_response_with_run_id): {e}") # De
                 ai_response_content = "Sorry, I encountered an error generating the respons

        # Display AI Response
        message_placeholder.markdown(ai_response_content)

        # Add AI message *with* run_id to history before attempting to render feedback
        st.session_state.chat_history.append({"role": "ai", "content": ai_response_content,

        # Render feedback widget only if run_id was successfully obtained
        if run_id:
            new_message_index = len(st.session_state.chat_history) - 1
            feedback_key = f"feedback_{new_message_index}"

            if feedback_key not in st.session_state:
                 st.session_state[feedback_key] = None

            # Display feedback widget using helper function
            create_feedback_widget(feedback_key, run_id, client)

        # Display a warning if feedback is not possible becasue run_id is missing
        elif not run_id:
             st.warning("Feedback not available for this message (missing run ID).", icon="

    print("--- Finished Processing User Query ---")
```