```python
import streamlit as st
from dotenv import load_dotenv
import os
from langchain_core.messages import HumanMessage, AIMessage
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_community.tools import TavilySearchResults
from langchain_core.tools import Tool
from langchain import hub
from langchain.agents import AgentExecutor, create_openai_functions_agent

# Load environment variables
load_dotenv()

# Initialize LLM
openai_api_key = os.getenv("OPENAI_API_KEY")
llm_gpt = ChatOpenAI(model_name="gpt-4o", temperature=0.7)

# Initialize Tavily Search
tavily_search = TavilySearchResults(
    max_results=3,
    search_depth="advanced",
    include_answer=True,
    api_key=os.getenv("TAVILY_API_KEY")
)

# Create the search tool
tools = [
    Tool(
        name="SearchTravel",
        func=tavily_search.invoke,
        description="Search for up-to-date travel information including: current weather co
    )
]

# Page config: Sets up basic parameters for the Streamlit page.
# - page_title: Appears in the browser tab.
# - page_icon: Displays a custom icon in the browser tab.
# - layout: "centered" or "wide" affects the overall page layout.
st.set_page_config(page_title='TravelPlanner with Web Search',
                   page_icon="lab4/img/cgu_icon.png",
                   layout="centered")

# Main title displayed at the top of the web page.
```

```python
st.title("   Aviana - your AI Travel Assistant")

# Create the sidebar section
with st.sidebar:
    st.title("Travel Guide")      #Sidebar title
    st.subheader("Meet Aviana ")  # Subheader for the sidebar
    st.write("""
    Your friendly AI travel companion! I am here to help you plan the perfect trip.
    Simply share:
    -   Where you want to go
    -   Travel dates
    -   Any specific interests or preferences

    I 'll even search the web for the most up-to-date travel information!

    """)
    st.subheader("Popular Destinations")
    st.write("""
    -   New York City, USA
    -   Tokyo, Japan
    -   Paris, France
    -   London, UK
    -   Bali, Indonesia
    """)

    # Refresh button to "restart" the trip planning conversation.
    # use_container_width=True makes the button span the full sidebar width.
    if st.button("Plan New Trip  ", use_container_width=True):
        st.session_state.chat_history = []
        st.rerun()

# Initialize chat history
# If the 'chat_history' key does not exist in the session state,
# create an empty list for it. This list will store the conversation.
if "chat_history" not in st.session_state:
    st.session_state.chat_history =[]

# Travel agent with search capability
def get_travel_response(user_input, chat_history):
    # Custom travel planner prompt
    prompt = ChatPromptTemplate.from_template("""
    **You are a professional travel planner with web search capabilities. **
    Your goal is to create detailed, engaging, and practical travel itineraries using the mo

    ---
```

CORE REQUIREMENTS:
1. You specialize **exclusively** in travel planning.
2. You **must** use the **SearchTravel** tool for current information (up to **4** searc
3. All itineraries must be **comprehensive** and **practical**.

SCOPE LIMITATION:
If the user's input is unrelated to travel planning, respond only with: "I'm sorry, I ca

---

MANDATORY WEB SEARCH:
For EVERY travel request, you MUST use the **SearchTravel** tool to find current (and ci
1. Exact driving distances and estimated drive times between destinations.
2. Attraction details (opening hours, entrance fees, reservation requirements).
3. Weather patterns and seasonal considerations for the planned dates.
4. Local events, festivals, and temporary closures during the travel period.
5. Current restaurant recommendations with operating hours.

INFORMATION GATHERING:
Before creating an itinerary, **always** collect or confirm the following:
• Destination(s)
• Travel dates/duration
• Number of travelers (age, special needs)
• Transportation method and preferences
• Accommodation preferences
• Activity interests
• Budget constraints

If critical information is missing, request clarifications from the user before proceedi

---

ITINERARY REQUIREMENTS:
Each itinerary must include:
• Structured **day-by-day plans** with specific timings.
• **Realistic travel times** between locations (based on your web search).
• At least **two accommodation options** per location with:
    - Name, address, price range, key amenities.
• **Multiple meal recommendations** for each day (including operating hours).
• **Detailed activity descriptions**, including:
    - Duration, cost, reservation requirement.
    - At least one alternative option for each time block.

```
    • A **Travel Tips & Considerations** section with:
        - Seasonal advice
        - Packing recommendations
        - Local customs
        - Safety information
        - Money-saving tips

    ---

    **SEARCH WORKFLOW:**
    1. Analyze the user's request; identify all destinations, dates, and user requirements.
    2. Use the **SearchTravel** tool to gather **current** information for each destination
    3. Compile and reference relevant data from your search results when building the itiner
    4. Cite specific details from the search (e.g., opening hours, fees) in your recommendat
    5. If more information is needed (dates, budget, travelers' details), ask clarifying que

    **Important:** Your knowledge may be outdated. ALWAYS reference **current** information

    Chat History: {chat_history}
    User Input: {input}

    {agent_scratchpad}
    """)

    # Create the agent
    agent = create_openai_functions_agent(llm_gpt, tools, prompt)
    agent_executor = AgentExecutor(
        agent=agent,
        tools=tools,
        verbose=True,
        max_iterations=10
    )

    # Get response
    response = agent_executor.invoke({"input": user_input, "chat_history": chat_history})
    return response["output"]

# # Display the existing conversation by iterating through the messages.
for message in st.session_state.chat_history:
    # Check if the message is from the user (HumanMessage).
    if isinstance(message, HumanMessage):
        with st.chat_message("Human"):
            st.markdown(message.content) # Display the user's message in the chat interface
    else:
        with st.chat_message("AI"):      # Otherwise, it's from the AI (AIMessage).
            st.markdown(message.content)
```

```python
# Create an input box for the user to type their query.
# The label "Where would you like to go?" appears inside the chat box.
user_query = st.chat_input("Where would you like to go?")

# Only show an initial greeting if the chat history is empty.
if not st.session_state.chat_history:
    with st.chat_message("AI"):
        st.write("""Hello, I'm Aviana, your personal Travel Assistant. How can I assist you

# Once the user enters a query, process it.
if user_query is not None and user_query != "":
    # Save the user query as a HumanMessage in the session's chat history.
    st.session_state.chat_history.append(HumanMessage(user_query))

    # Display the user query in the chat interface.
    with st.chat_message("Human"):
        st.markdown(user_query)

    # Generate AI response using get_travel_response.
    with st.chat_message("AI"):
        with st.spinner("Aviana is processing your request..."):
            ai_response = get_travel_response(user_query, st.session_state.chat_history)

        # Display the AI's response in the chat.
        st.markdown(ai_response)

    # Add the AI's message to the chat history for continuity.
    st.session_state.chat_history.append(AIMessage(ai_response))
```