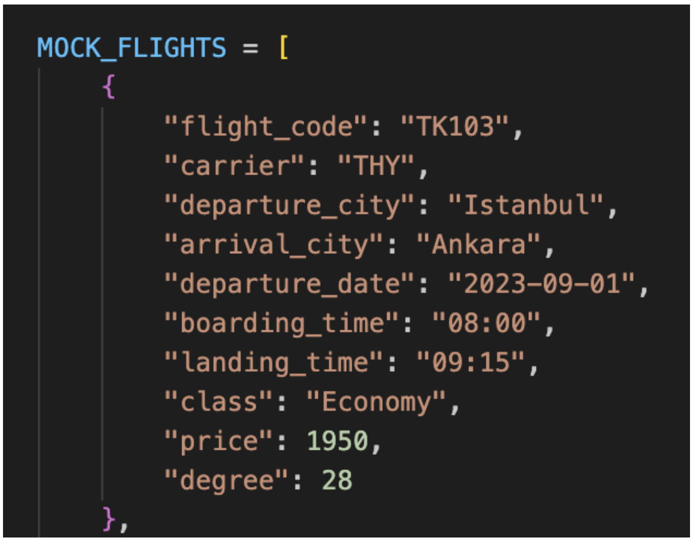
TRAVEL ASSISTANT CASE

I built the Travel Assistant (Flight) project as a demonstration of **agent-based AI** for flight searching and ticket purchasing. In this system, **ZeroShotAgent** from **LangChain** coordinates multiple tools—searching flights, checking corporate policies, and mocking a purchase flow—all through a user-friendly Streamlit application. I used the **gemini-2.0-flash** model for the agent.

Source code: https://github.com/xemeriusss/Travel-Asistant-Flight

1. DATASET

Mock dataset is used for this case:



```
MOCK_FLIGHTS = [
    {
        "flight_code": "TK103",
        "carrier": "THY",
        "departure_city": "Istanbul",
        "arrival_city": "Ankara",
        "departure_date": "2023-09-01",
        "boarding_time": "08:00",
        "landing_time": "09:15",
        "class": "Economy",
        "price": 1950,
        "degree": 28
    },
```

2. TOOLS

Why tools?

- Pure LLMs can only output text. They can’t natively perform real calculations or fetch real-world data. Tools fill this gap.
- By hooking an LLM up to, for example, a **database query**, a **payment service**, a **policy checker**, or an **API call**, the agent can do tasks that exceed simple text generation.
- Each tool is basically a **function** wrapped inside a LangChain `Tool` object so that the **LLM-based agent** can call it when needed.

For this project an example scenario can be:

- **SEARCH FLIGHT TOOL:** When the user asks for flights, the agent invokes the search flights tool, which filters a mock flight database and returns matching flights.
- **POLICY CHECK TOOL**: Then the agent calls the policy check tool to enforce constraints.

- **PURCHASE TOOL:** Finally, once the user selects a flight, the agent calls the purchase tool, which “confirms” the booking.
- **RETRIEVE PAST PURCHASE TOOL:** Also if the user wants to see past purchases, the agent calls the retrieve_past_purchases_tool**.**
- **WEATHER TOOL:** If the user wants to fly to hot or cold places, the agent recommends these places with recommend_destination_tool.

Each of these tools does one job—search, check, or purchase—and the agent orchestrates them based on the user’s requests.

3. FLOW and AGENT

1.1 Agent Creation and Initialization

1. `create_agent()` function in app.py
2. agent.py : Creating LLM -> Agent prompt setup -> Creating the agent (ZeroShotAgent)

The **`ZeroShotAgent`** is one of the agent types provided by LangChain, designed to work with LLMs.

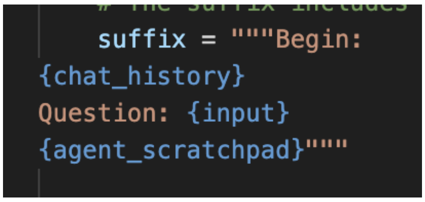
- Agents can make decisions about which tools to use based on the input.
- "Zero-shot" means that the model doesn't need to have seen any specific examples in the prompt. This makes it very flexible and suitable for a wide range of tasks, especially when you don’t have a specific dataset for the task.

Agent types: https://medium.com/@csakash03/a-quick-guide-to-agent-types-in-langchain-for-csv-eb823be10799

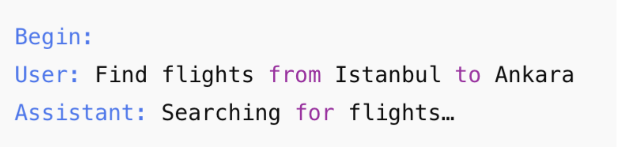
Components of agent prompt:

- **Prefix:** Defines the **context** and **instructions** for the agent at the beginning of the task.
- **Suffix:** Specifies the formatting of the agent's response.

- Example of suffix:



```
    suffix = """Begin:
{chat_history}
Question: {input}
{agent_scratchpad}"""
```



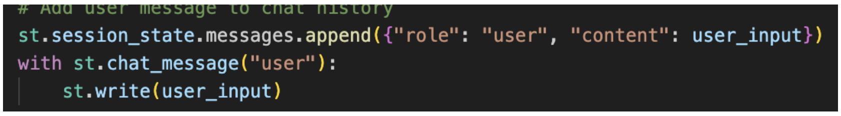
```
Begin:
User: Find flights from Istanbul to Ankara
Assistant: Searching for flights…
```

-

1.2 User Interaction Flow

Once the agent is created and returned, the following happens when the user interacts with the Streamlit UI:

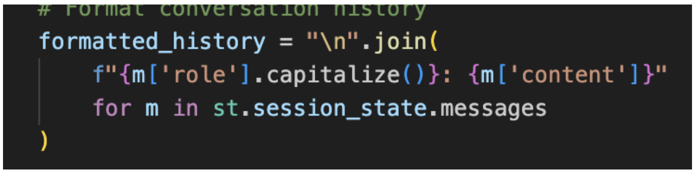
1. **Initializing Session State** -> to store agent instance, chat history, outbound and inbound flight lists, past purchases, process step
2. **Initializing sidebar to process tracking**
3. **User Input Handling** through the `st.chat_input()`
4. **Storing User Input** in chat history



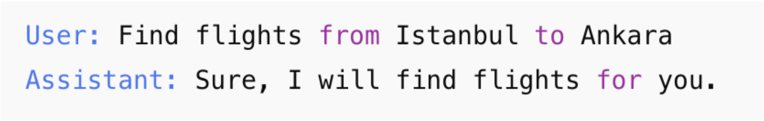
```
# Add user message to chat history
st.session_state.messages.append({"role": "user", "content": user_input})
with st.chat_message("user"):
    st.write(user_input)
```

5. **Formatting Chat History:**

- Before the agent processes the input, the entire conversation history is formatted into a string.



```
# Format conversation history
formatted_history = "\n".join(
    f"{m['role'].capitalize()}: {m['content']}"
    for m in st.session_state.messages
)
```



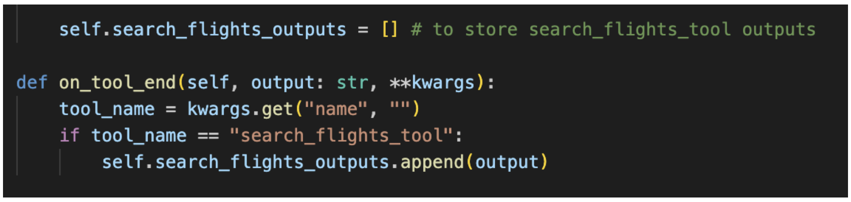
```
User: Find flights from Istanbul to Ankara
Assistant: Sure, I will find flights for you.
```

6. **Callback:**

- LangChain automatically calls the on_tool_end method **after** a tool finishes.

```
callback_handler = ToolOutputCatcher()
```

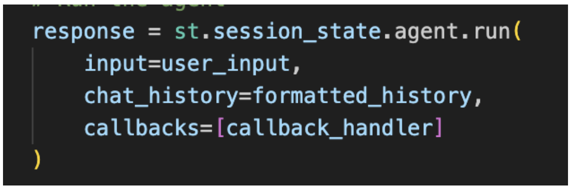
- Each time the **`search_flights_tool`** finishes, it stores its **output** in a list to efficiently capture the outputs.



```
    self.search_flights_outputs = [] # to store search_flights_tool outputs

def on_tool_end(self, output: str, **kwargs):
    tool_name = kwargs.get("name", "")
    if tool_name == "search_flights_tool":
        self.search_flights_outputs.append(output)
```

7. **Running the Agent to Process the User Input:**

- `agent.run()` is called. This is where the agent interacts with the tools .

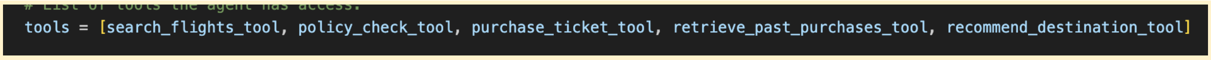


```
response = st.session_state.agent.run(
    input=user_input,
    chat_history=formatted_history,
    callbacks=[callback_handler]
)
```

- The input and chat history are passed to the agent.
- The response from the agent is appended to the chat history and displayed.

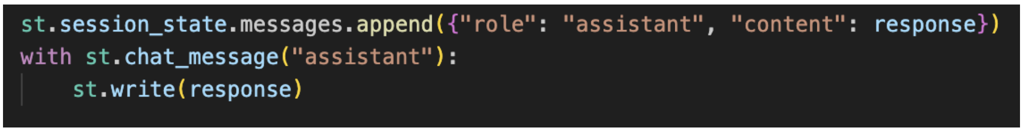
8. **Calling the Tools from `tools.py`:**

- The agent calls the necessary tools. The tools calls the `wrapper` functions
- Wrapper calls the functions -> search_flights ,check_policy…



```
tools = [search_flights_tool, policy_check_tool, purchase_ticket_tool, retrieve_past_purchases_tool, recommend_destination_tool]
```

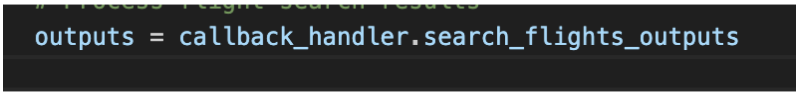
- The agent collects the results from the tools and constructs a response. This response is then returned to `app.py`, where it will be displayed in the UI.



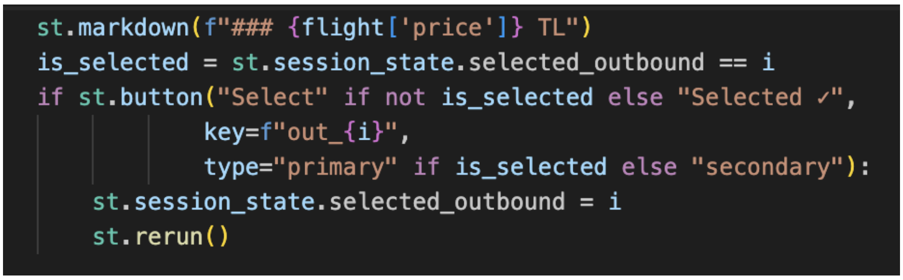
```
st.session_state.messages.append({"role": "assistant", "content": response})
with st.chat_message("assistant"):
    st.write(response)
```

1.3 Flight Selection and Purchase Flow

- The user can select flights by clicking the **"Select"** button with the separated outbound and inbound flights coming from callback next to each option.



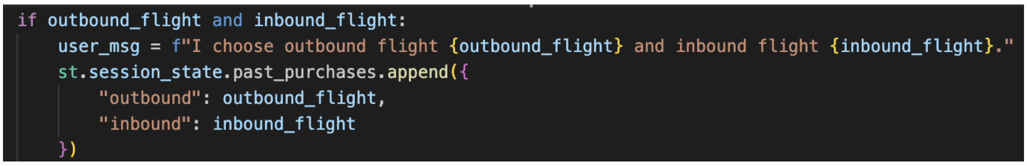
```
outputs = callback_handler.search_flights_outputs
```

- 

```
st.markdown(f"### {flight['price']} TL")
is_selected = st.session_state.selected_outbound == i
if st.button("Select" if not is_selected else "Selected ✓",
            key=f"out_{i}",
            type="primary" if is_selected else "secondary"):
    st.session_state.selected_outbound = i
    st.rerun()
```

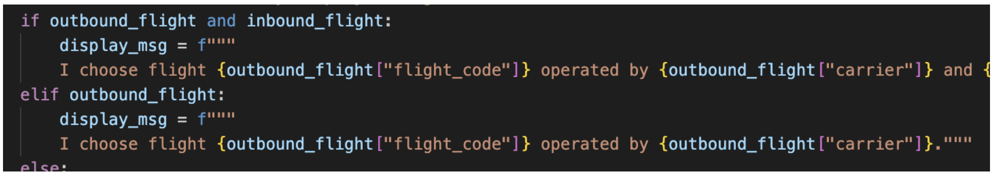
- Displays an order summary.
- When the user clicks **"Confirm Purchase"**, the selection is processed:

1. Constructs a user message for the agent and adds the purchase to the past purchases.



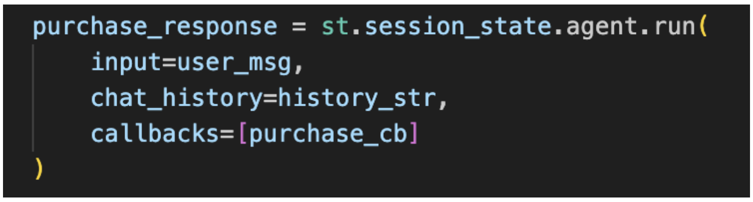
```
if outbound_flight and inbound_flight:
    user_msg = f"I choose outbound flight {outbound_flight} and inbound flight {inbound_flight}."
    st.session_state.past_purchases.append({
        "outbound": outbound_flight,
        "inbound": inbound_flight
    })
```

2. Creates a display message for the chat history.



```
if outbound_flight and inbound_flight:
    display_msg = f"""
    I choose flight {outbound_flight["flight_code"]} operated by {outbound_flight["carrier"]} and {
elif outbound_flight:
    display_msg = f"""
    I choose flight {outbound_flight["flight_code"]} operated by {outbound_flight["carrier"]}."""
else:
```

3. Calls the agent to finalize the purchase. The final part displays the LLM's **confirmation** or result in the chat, completing the purchase flow.



```
purchase_response = st.session_state.agent.run(
    input=user_msg,
    chat_history=history_str,
    callbacks=[purchase_cb]
)
```

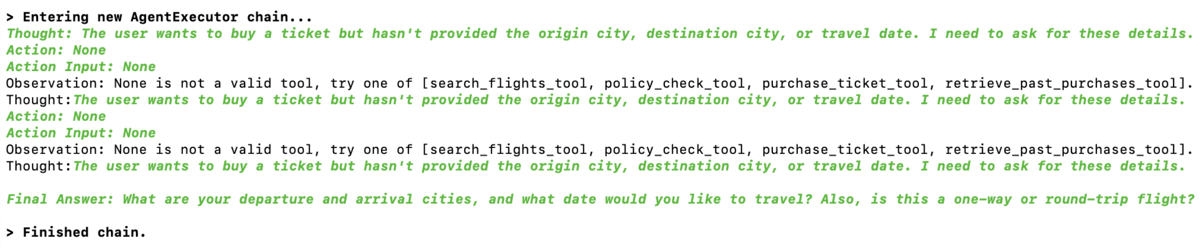
4. EXAMPLE OF CHAIN

Thought:

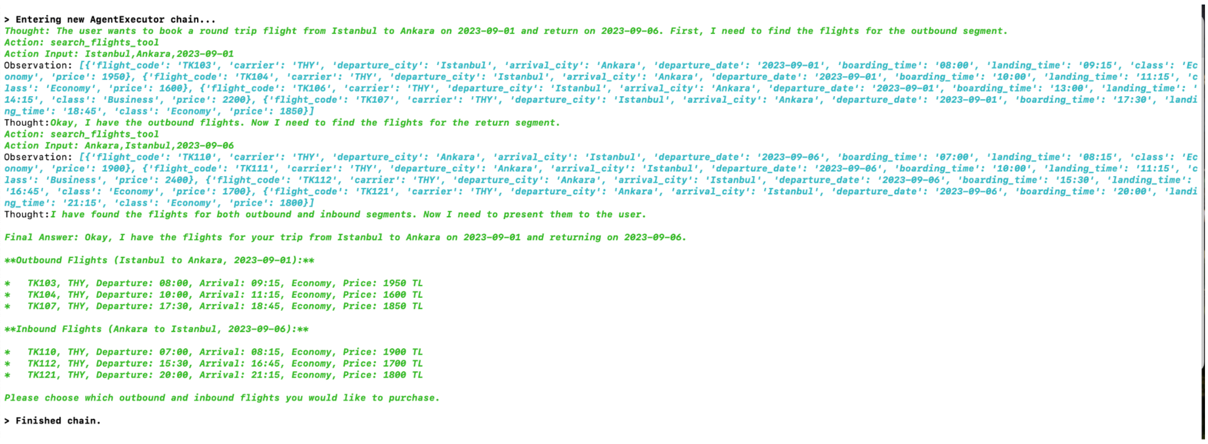
- Represents the agent's internal reasoning or decision-making process.

Observation:

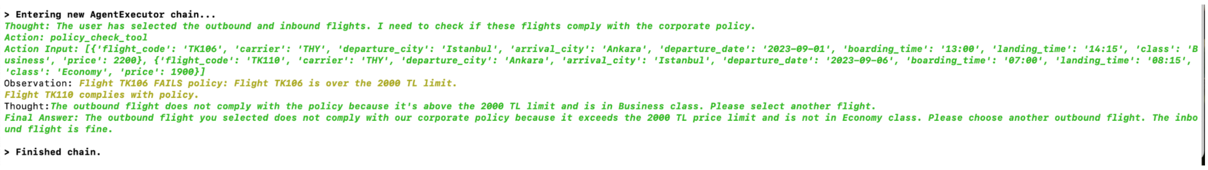
- Represents the information or data gathered from an external tool or system.



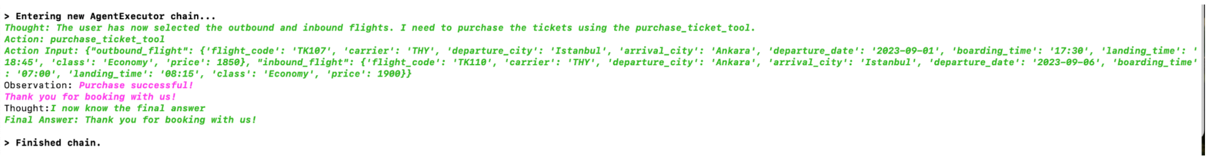
```
> Entering new AgentExecutor chain...
Thought: The user wants to buy a ticket but hasn't provided the origin city, destination city, or travel date. I need to ask for these details.
Action: None
Action Input: None
Observation: None is not a valid tool, try one of [search_flights_tool, policy_check_tool, purchase_ticket_tool, retrieve_past_purchases_tool].
Thought:The user wants to buy a ticket but hasn't provided the origin city, destination city, or travel date. I need to ask for these details.
Action: None
Action Input: None
Observation: None is not a valid tool, try one of [search_flights_tool, policy_check_tool, purchase_ticket_tool, retrieve_past_purchases_tool].
Thought:The user wants to buy a ticket but hasn't provided the origin city, destination city, or travel date. I need to ask for these details.

Final Answer: What are your departure and arrival cities, and what date would you like to travel? Also, is this a one-way or round-trip flight?

> Finished chain.
```



```
> Entering new AgentExecutor chain...
Thought: The user wants to book a round trip flight from Istanbul to Ankara on 2023-09-01 and return on 2023-09-06. First, I need to find the flights for the outbound segment.
Action: search_flights_tool
Action Input: Istanbul,Ankara,2023-09-01
Observation: [{'flight_code': 'TK103', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '08:00', 'landing_time': '09:15', 'class': 'Ec
onomy', 'price': 1950}, {'flight_code': 'TK104', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '10:00', 'landing_time': '11:15', 'c
lass': 'Economy', 'price': 1600}, {'flight_code': 'TK106', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '13:00', 'landing_time': '
14:15', 'class': 'Business', 'price': 2200}, {'flight_code': 'TK107', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '17:30', 'landi
ng_time': '18:45', 'class': 'Economy', 'price': 1850}]
Thought:Okay, I have the outbound flights. Now I need to find the flights for the return segment.
Action: search_flights_tool
Action Input: Ankara,Istanbul,2023-09-06
Observation: [{'flight_code': 'TK110', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time': '07:00', 'landing_time': '08:15', 'class': 'Ec
onomy', 'price': 1900}, {'flight_code': 'TK111', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time': '10:00', 'landing_time': '11:15', 'c
lass': 'Business', 'price': 2400}, {'flight_code': 'TK112', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time': '15:30', 'landing_time':
'16:45', 'class': 'Economy', 'price': 1700}, {'flight_code': 'TK121', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time': '20:00', 'landi
ng_time': '21:15', 'class': 'Economy', 'price': 1800}]
Thought:I have found the flights for both outbound and inbound segments. Now I need to present them to the user.

Final Answer: Okay, I have the flights for your trip from Istanbul to Ankara on 2023-09-01 and returning on 2023-09-06.

**Outbound Flights (Istanbul to Ankara, 2023-09-01):**

*   TK103, THY, Departure: 08:00, Arrival: 09:15, Economy, Price: 1950 TL
*   TK104, THY, Departure: 10:00, Arrival: 11:15, Economy, Price: 1600 TL
*   TK107, THY, Departure: 17:30, Arrival: 18:45, Economy, Price: 1850 TL

**Inbound Flights (Ankara to Istanbul, 2023-09-06):**

*   TK110, THY, Departure: 07:00, Arrival: 08:15, Economy, Price: 1900 TL
*   TK112, THY, Departure: 15:30, Arrival: 16:45, Economy, Price: 1700 TL
*   TK121, THY, Departure: 20:00, Arrival: 21:15, Economy, Price: 1800 TL

Please choose which outbound and inbound flights you would like to purchase.

> Finished chain.
```



```
> Entering new AgentExecutor chain...
Thought: The user has selected the outbound and inbound flights. I need to check if these flights comply with the corporate policy.
Action: policy_check_tool
Action Input: [{'flight_code': 'TK106', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '13:00', 'landing_time': '14:15', 'class': 'B
usiness', 'price': 2200}, {'flight_code': 'TK110', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time': '07:00', 'landing_time': '08:15',
'class': 'Economy', 'price': 1900}]
Observation: Flight TK106 FAILS policy: Flight TK106 is over the 2000 TL limit.
Flight TK110 complies with policy.
Thought:The outbound flight does not comply with the policy because it's above the 2000 TL limit and is in Business class. Please select another flight.
Final Answer: The outbound flight you selected does not comply with our corporate policy because it exceeds the 2000 TL price limit and is not in Economy class. Please choose another outbound flight. The inbo
und flight is fine.

> Finished chain.
```



```
> Entering new AgentExecutor chain...
Thought: The user has now selected the outbound and inbound flights. I need to purchase the tickets using the purchase_ticket_tool.
Action: purchase_ticket_tool
Action Input: {"outbound_flight": {'flight_code': 'TK107', 'carrier': 'THY', 'departure_city': 'Istanbul', 'arrival_city': 'Ankara', 'departure_date': '2023-09-01', 'boarding_time': '17:30', 'landing_time': '
18:45', 'class': 'Economy', 'price': 1850}, "inbound_flight": {'flight_code': 'TK110', 'carrier': 'THY', 'departure_city': 'Ankara', 'arrival_city': 'Istanbul', 'departure_date': '2023-09-06', 'boarding_time'
: '07:00', 'landing_time': '08:15', 'class': 'Economy', 'price': 1900}}
Observation: Purchase successful!
Thank you for booking with us!
Thought:I now know the final answer
Final Answer: Thank you for booking with us!

> Finished chain.
```

5. DEMO

🎬 streamlit-app-2025-03-21-13-03-86.webm

6. FUTURE WORK

- ☑ ~~Sidebar update~~
- ☐ Make a bigger database and using SQL
- ☐ New agents:
  - ☐ Notifications Tool
  - ☑ ~~Weather Information Tool~~
- ☑ ~~Improve the UI~~
- ☐ The bug for past purchases
- ☐ Update on selection of ticket part in UI
- ☐ Turkish language support