Kaan Kosesoy

Travel Assistant Documentation

1. Agent-based Architecture & Modularity

The system follows a modular agent-based architecture which comprises of the following agents, each responsible for a specific part of the user workflow:

- **Travel Agent (Entry Point):**

Interacts with the user initially to identify their intent (flight booking, car rental, or hotel reservation) regarding their trip and routes them to the corresponding specialized agent.

- **Flight Agent:**

A dedicated agent that collects user input related to flights (destinations, dates, trip type), searches flights from a local database, checks policy compliance, and simulates ticket purchasing or manager escalation processes. These functionalities of the flight agent are provided by individually implemented tools.

- **Policy Agent:**

Another dedicated validation agent operating under the flight agent, that checks selected flight options against organizational rules (class restrictions and price limits), and returns structured results for further decision-making.

- **Car Rental & Hotel Reservation Agents**

These agents are currently implemented as placeholders in the overall graph. While the Travel Agent can route users to them based on user intent, they do not perform any real functionality at this stage. They're included to demonstrate how new agents can be integrated into the existing architecture in a clean and forward-compatible manner.

Each agent operates independently and communicates with other agents through structured messages within a shared execution graph. The flight agent itself is implemented as a separate sub-graph embedded inside the main travel graph, where individual tools also operate and communicate via shared state variables. This decoupled graph and tool based approaches provide modularity at every level of the system, allowing both the extensibility of any single agent's capabilities via additional tools, and scalability of the overall system via integration of future agents like hotel reservation or car rental agents.

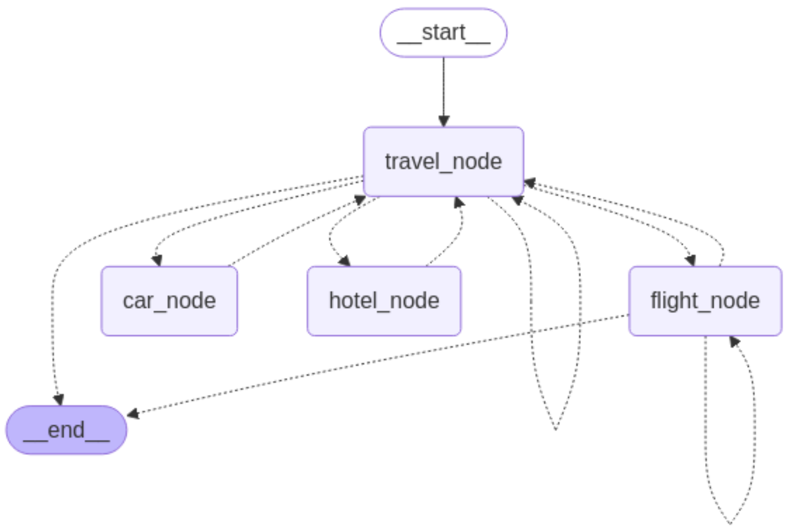


Figure 1. High-level graph structure of the overall travel assistant system

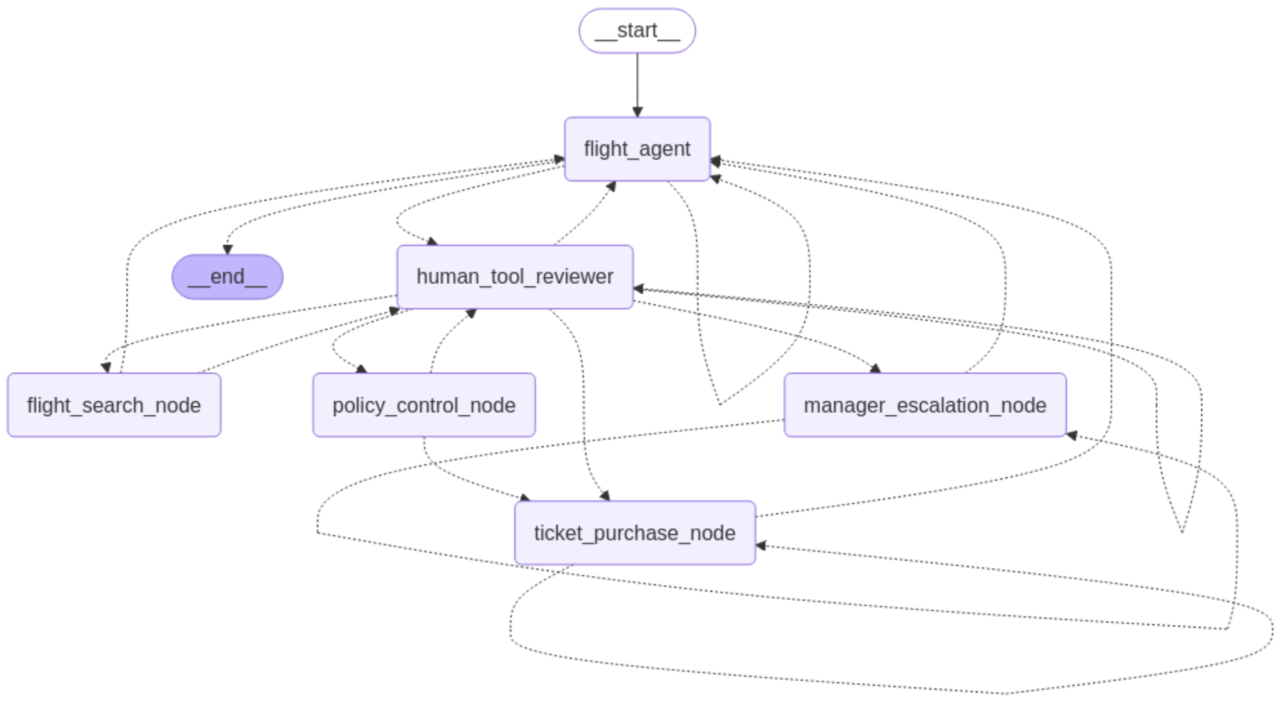


Figure 2. Structure of the flight assistant that resides as a sub-graph in the flight_node under the overall travel assistant graph

2. Selected Technologies

- **LangChain**

Used as the primary framework for building conversational agents and integrating tools. LangChain provided the building blocks to:
  - Create and manage structured chat models
  - Define and register tools (for actions like flight searching, ticket purchasing etc.)
  - Wrap chat models with tools and prompts to obtain agent-like behavior (for intent detection, policy enforcement etc.)

- **LangGraph**

Used to implement the actual graph-based control flow and inter-agent orchestration. Each agent is a node (or even a subgraph) in the overall travel assistant workflow. LangGraph enables:
  - Defining flexible, stateful workflows with transitions between agent nodes
  - Managing memory/state across user interactions
  - Modular nesting of agents (e.g., the flight assistant itself is implemented as a subgraph within the main graph)
  - Streaming outputs in real-time during conversation

- **SQLite**

The system uses a local mock flight database built with SQLite. This database was populated with ~5 million synthetic flights across cities in Turkey; generated with randomized airlines, times, prices, and durations. The SQL was employed for realistic querying and indexing for efficient tool-based flight search.

- **Chat Model Selection: GPT-4o-mini**

Initially, I tried to take an open-source approach by setting up a quantized local Llama model. However, due to system constraints (lack of GPU support and limited resources), the local model:
  - Had limited performance and response quality
  - Faced integration limitations with LangChain

As a result, I switched to a paid service with OpenAI's GPT-4o-mini model, accessed via LangChain OpenAI packages. This model was chosen for its familiarity (as I already use ChatGPT) and cost-efficiency (as the cheapest option among OpenAI's offerings) while still providing reliable structured output and realistic conversational flows.

3. Integration Points

Integration points are implemented using LangChain’s tool abstraction where each tool serves as a modular interface to an external system. Here are the integration points the system currently uses (all are mock implementations, but can be swapped with real APIs):

- **Flight Search Tool**
Queries a local SQLite database of mock flight data. Acts as the integration point for searching available flights, simulating the behavior of querying external flight provider APIs.

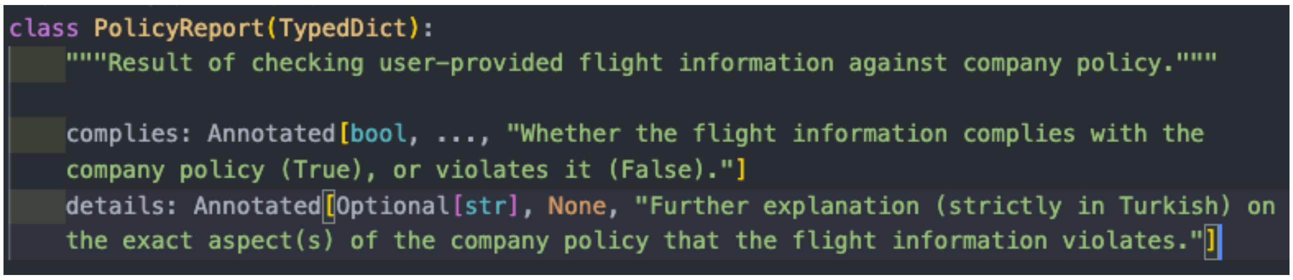
- **Ticket Purchase Tool**
Simulates the flight ticket reservation process, including seat selection and generating ticket details. Represents an integration point to a real-world airline booking or payment system.

- **Manager Escalation Tool**
Mocks the process of requesting managerial approval for policy-violating flights. In a real-world system, this could connect to an HR portal, email service etc.

4. How corporate travel rules are applied?

The system ensures policy compliance through a dedicated policy agent, which is invoked after the user selects their flights but before any purchase occurs.

The policy agent is implemented using LangChain’s structured output support. It wraps a GPT-4o-mini model using “with_structured_output()” and returns responses in a strict Python TypedDict schema called PolicyReport.



```python
class PolicyReport(TypedDict):
    """Result of checking user-provided flight information against company policy."""

    complies: Annotated[bool, ..., "Whether the flight information complies with the
    company policy (True), or violates it (False)."]
    details: Annotated[Optional[str], None, "Further explanation (strictly in Turkish) on
    the exact aspect(s) of the company policy that the flight information violates."]
```

Figure 3. Definition of the PolicyReport schema for the output of the policy agent.

This guarantees consistent and parsable output that downstream nodes can reliably act upon.

Prompt Engineering

The policy assistant uses a custom prompt template with a detailed system message that:
- Defines the current policy rules (e.g., price ≤ 2000 TL, only Economy class).
- Explains the required output format.
- Includes few-shot examples to demonstrate expected behavior for various flight inputs.

These examples cover both compliant and non-compliant flights to reinforce correct reasoning and formatting.