Travel Assistant

2025-12-09 03:32 Report

Agentic Travel Assistant – Requirements Document

1. Overview

The goal is to build an **agentic chat-based travel assistant** that can:

- Understand natural-language queries from users.
- Retrieve live flight information from an airline or aggregator API.
- Remember user travel preferences and history using **mem0**.
- Provide recommendations, comparisons, and clarifying questions.
- Enable users to book or plan itineraries through a conversational interface.

Backend language: **Python**
Frontend/UI: **React**

2. Core Capabilities

2.1 Conversational Understanding

The assistant should be capable of:

- Interpreting a user's travel intent (e.g., searching flights, modifying dates, comparing routes).
- Extracting necessary parameters: origin, destination, dates, passengers, cabin class.
- Asking follow-up questions if required details are missing.
- Distinguishing between general travel advice and actionable search requests.

2.2 Agentic Reasoning Loop

The assistant must:

- Evaluate whether a tool/action (e.g., flight search) is required.
- Plan steps before answering.

- Call the appropriate external API.
- Use results to generate user-understandable output.
- Persist user preferences when appropriate.

3. Integrations & External Systems

3.1 Airline / Flight Search API

The system must integrate with a real-time or near–real-time flight search service.
Key features required from the API:

- Query flights by origin, destination, departure date, optional return date.
- Specify number of passengers and cabin class.
- Return structured information: price, airline, departure/arrival times, stops, duration.
- Support filtering or sorting (if API allows).

3.2 mem0 Memory System

The assistant should use mem0 to:

- **Store preferences** such as seat type, preferred airlines, maximum number of stops, time-of-day preferences, cabin class, and aversion to red-eyes.
- **Store travel history** such as previously booked flights, frequently used routes, and duration preferences.
- **Retrieve memories** at the start of every conversation turn.
- **Update memory** whenever the user expresses a stable preference or confirms a travel pattern.

4. Functional Requirements

4.1 Flight Search

- Interpret user queries and extract route details.
- Validate dates and ask the user for clarification when needed.
- Call the flight API and handle errors gracefully.
- Present results in a structured and helpful format.
- Provide comparison categories:
  - Cheapest
  - Fastest
  - Best overall (weighted combination of factors)

4.2 Recommendation Logic

The agent should:

- Personalize results using stored preferences.
- Explain tradeoffs (e.g., price vs travel time).
- Recommend specific flights or routes when asked.
- Suggest alternative nearby airports or dates when relevant.

4.3 Memory-Aware Dialogue

- Use stored preferences automatically when searching.
- Mention remembered preferences ("Since you prefer morning flights…").
- Offer users the ability to revise or remove preferences.
- Log new preferences as they appear naturally in conversation.

4.4 Context-Aware Questions

When essential information is missing, the assistant should ask for:

- Travel dates
- Number of passengers
- Trip type (one-way or round-trip)
- Cabin class
- Preferred time of day (if relevant)

5. Non-Functional Requirements

Scale and reliability but, we'll define this later

6. Frontend (React UI) Requirements

6.1 Chat Interface

- Real-time chat component with streaming messages.
- Support for multi-turn conversation history.
- Display flight results with structured cards.
- Allow user to select options or refine filters interactively.

6.2 User Profile & Preferences (Optional for V2)

- View/edit stored preferences.
- View travel history.
- Provide preference toggles (direct flights only, avoid red-eye, preferred airlines).

6.3 Mobile-First Responsive Design

- Chat and results must work smoothly on mobile and desktop.

7. Backend (Python) Requirements

7.1 LLM Orchestrator

- Manage the agentic loop.
- Inject memories into context for each query.
- Handle function/tool call responses.
- Apply reasoning patterns like slot filling and intent classification.

7.2 Flight API Integration Layer

- Encapsulate requests to the external flight API.
- Normalize response formats for use by the agent.
- Provide filtering and scoring helpers.

7.3 Memory Layer (mem0)

- Fetch memories per user.
- Add new memory entries.
- Provide a consistent schema for travel preferences and history.

8. User Journey Examples (High-Level)

8.1 Searching for a Flight

1. User: "I need flights from Bangalore to Dubai next month for 4 days."
2. Agent retrieves stored preferences.
3. Agent asks: "Which week works best for you?"
4. User provides dates.
5. Agent searches flights and returns options.
6. Agent highlights best/cheapest/fastest.

8.2 Learning a Preference

1. User: "I hate red-eye flights."
2. Agent: "Got it, I'll avoid red-eye flights in future searches."
3. System stores this as a memory.

8.3 Using Memory

1. User: "Find me flights to San Francisco."

2. Agent automatically applies stored preferences (e.g., direct flights only).
