# Travel Booking System Architecture Document

## 1. Overview

The Travel Booking System is a modular, AI-driven application designed to assist users in managing travel-related tasks, including flight bookings, hotel reservations, car rentals, excursion bookings, and calendar event management. Built using Python, LangChain, and SQLite, the system leverages a conversational AI interface powered by OpenAI's GPT-4o-mini model to handle user queries and delegate tasks to specialized assistants. The architecture is designed to process requests sequentially, ensuring robust handling of single intents and requiring user clarification for multiple simultaneous requests to maintain stability.

This document outlines the system's architecture, components, data flow, and key considerations for scalability, maintenance, and error handling.

---

## 2. System Architecture

The system follows a modular architecture with a primary assistant acting as the entry point for user interactions, delegating specialized tasks to sub-assistants. It integrates with a SQLite database for data storage, external APIs for embeddings and search, and a state management system to track conversation context.

### 2.1 High-Level Components

1. **Primary Assistant**:
    - Role: Handles general user queries, performs intent detection, and delegates specialized tasks.
    - Technologies: LangChain, OpenAI GPT-4o-mini, Tavily Search API.
    - Responsibilities: Answers general questions, searches flight information, checks policies, and routes tasks to specialized assistants.
2. **Specialized Assistants**:

- Types: Flight Booking Assistant, Hotel Booking Assistant, Car Rental Assistant, Excursion Assistant, Calendar Event Assistant.
- Technologies: LangChain, OpenAI GPT-4o-mini.
- Responsibilities: Execute specific tasks (e.g., booking a hotel, updating a flight) using dedicated tools and prompts.

3. **Database**:

   - Type: SQLite database ( `travel2.sqlite` ).
   - Purpose: Stores flight, hotel, car rental, and excursion data, including user tickets, bookings, and trip recommendations.

4. **External Services**:

   - OpenAI API: Provides embeddings for policy lookup and powers the LLM for conversational responses.
   - Tavily Search API: Supports web searches for general queries.
   - Google Calendar API: Manages calendar event creation (simulated in the provided code).

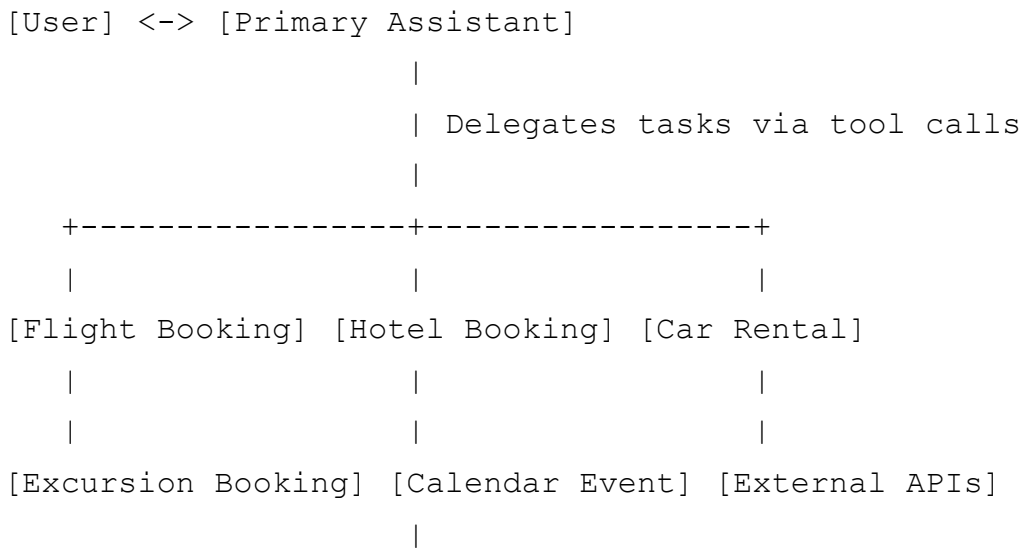
5. **State Management**:

   - Mechanism: LangGraph's `State` class with a `dialog_state` list and message history.
   - Purpose: Tracks conversation context and assistant delegation.

6. **Tools**:

   - Categories: Safe tools (e.g., `search_flights` , `search_hotels` ) for read-only operations and sensitive tools (e.g., `book_hotel` , `update_ticket_to_new_flight` ) for write operations.
   - Purpose: Execute specific actions like searching, booking, updating, or canceling.

## 2.2 System Diagram



```
                    |
            [SQLite Database]
```

- **User**: Interacts via text input, sending queries to the primary assistant.
- **Primary Assistant**: Processes queries, uses tools for general tasks, or delegates to specialized assistants.
- **Specialized Assistants**: Handle specific tasks using dedicated tools and prompts.
- **SQLite Database**: Stores and retrieves data for flights, hotels, car rentals, and excursions.
- **External APIs**: Provide embeddings (OpenAI), search results ( Tavily), and calendar functionality (Google Calendar).

---

# 3. Component Details

## 3.1 Primary Assistant

- **Prompt**: Defined in `primary_assistant_prompt` , instructs the assistant to handle general queries, search flight information, check policies, and delegate tasks. It includes logic to ask for user clarification when multiple intents are detected (e.g., "Book a hotel and rent a car").
- **Tools**:
  - `TavilySearch` : For general web searches.
  - `search_flights` : Queries flight data from the database.
  - `lookup_policy` : Retrieves company policies using a vector store retriever.
  - `add_event_to_calendar` : Adds events to Google Calendar.
  - Delegation tools: `ToFlightBookingAssistant` , `ToHotelBookingAssistant` , `ToBookCarRental` , `ToBookExcursion` .
- **Behavior**: Uses intent detection to route tasks. For multiple intents, it prompts the user to prioritize (e.g., "Which would you like to do first?").

## 3.2 Specialized Assistants

Each specialized assistant has a dedicated prompt and toolset, ensuring focused task execution.

1. **Flight Booking Assistant**:
   - Prompt: `flight_booking_prompt` focuses on flight updates and cancellations, confirming details and checking fees.
   - Tools: `search_flights` , `update_ticket_to_new_flight` , `cancel_ticket` , `add_event_to_calendar` , `CompleteOrEscalate` .
   - Behavior: Searches for flights, updates or cancels tickets, and escalates if user needs change.

2. **Hotel Booking Assistant**:
    - Prompt: `book_hotel_prompt` handles hotel searches and bookings.
    - Tools: `search_hotels`, `book_hotel`, `update_hotel`, `cancel_hotel`, `add_event_to_calendar`, `CompleteOrEscalate`.
    - Behavior: Searches hotels based on user preferences, confirms bookings, and allows updates or cancellations.
3. **Car Rental Assistant**:
    - Prompt: `book_car_rental_prompt` manages car rental bookings.
    - Tools: `search_car_rentals`, `book_car_rental`, `update_car_rental`, `cancel_car_rental`, `add_event_to_calendar`, `CompleteOrEscalate`.
    - Behavior: Searches for available cars, books rentals, and supports updates or cancellations.
4. **Excursion Assistant**:
    - Prompt: `book_excursion_prompt` handles trip recommendations and bookings.
    - Tools: `search_trip_recommendations`, `book_excursion`, `update_excursion`, `cancel_excursion`, `add_event_to_calendar`, `CompleteOrEscalate`.
    - Behavior: Searches for excursions based on location or keywords, books them, and allows modifications.
5. **Calendar Event Assistant**:
    - Tool: `add_event_to_calendar`.
    - Behavior: Adds events to Google Calendar with user-specified details (title, location, start/end times).

## 3.3 Database Schema

The SQLite database (`travel2.sqlite`) contains the following key tables:

- **flights**: Stores flight details (e.g., `flight_id`, `departure_airport`, `scheduled_departure`).
- **tickets**: Contains user ticket information (e.g., `ticket_no`, `passenger_id`).
- **ticket_flights**: Links tickets to flights (e.g., `ticket_no`, `flight_id`, `fare_conditions`).
- **boarding_passes**: Stores seat assignments (e.g., `ticket_no`, `flight_id`, `seat_no`).
- **hotels**: Stores hotel data (e.g., `id`, `name`, `location`, `booked`).
- **car_rentals**: Contains car rental data (e.g., `id`, `name`, `location`, `booked`).
- **trip_recommendations**: Stores excursion data (e.g., `id`, `name`, `location`, `keywords`).

The `update_dates` function ensures all datetime fields are adjusted to the current time, maintaining data relevance.

### 3.4 External Services

- **OpenAI API**:
  - Used for text embeddings (`text-embedding-3-small`) in the `VectorStoreRetriever` for policy lookup.
  - Powers the LLM (GPT-4o-mini) for conversational responses and intent detection.
- **Tavily Search API**:
  - Provides web search capabilities for general queries.
- **Google Calendar API**:
  - Simulated in the code for adding events; assumes a service client (`calendar_service`) for integration.

### 3.5 State Management

- **State Structure** (`State` class):
  - `messages`: A list of conversation messages, updated using `add_messages`.
  - `user_info`: Stores user flight information for context.
  - `dialog_state`: A list tracking the current assistant (e.g., `["assistant", "book_hotel"]`).
- **Dialog State Updates**: Managed by `update_dialog_stack`, which appends new states or pops the last state for escalation.
- **Purpose**: Ensures conversation context is preserved and tasks are routed correctly.

---

## 4. Data Flow

1. **User Input**:
   - The user submits a query (e.g., "Book a hotel in Zurich").
   - The input is received by the primary assistant.
2. **Intent Detection**:
   - The primary assistant processes the query using `assistant_runnable` (prompt + LLM).
   - If a single intent is detected, it invokes the relevant tool or delegates to a specialized assistant.
   - For multiple intents (e.g., "Book a hotel and rent a car"), it prompts for clarification (e.g., "Which would you like to do first?").

3. **Task Delegation**:
    - The primary assistant invokes a tool like `ToHotelBookingAssistant` with parameters (e.g., `location`, `checkin_date`).
    - The `create_entry_node` function transitions the `dialog_state` to the specialized assistant (e.g., `["book_hotel"]`).
4. **Task Execution**:
    - The specialized assistant uses its tools (e.g., `ceeded
    - `search_hotels`, `book_hotel`) to perform the task.
    - Results are returned to the user or escalated via `CompleteOrEscalate`.
5. **Database Interaction**:
    - Tools query or update the SQLite database (e.g., `UPDATE hotels SET booked = 1`).
    - Changes are committed to ensure data consistency.
6. **Response**:
    - The assistant returns a response to the user, either confirming the task or escalating if needed.

---

# 5. Key Features and Considerations

## 5.1 Sequential Task Processing

- The system is designed for sequential task execution, ensuring stability but limiting concurrent task handling.
- Multiple intents are managed by prompting the user to prioritize tasks, avoiding conflicts in `dialog_state`.

## 5.2 Error Handling

- The `handle_tool_error` function provides fallback responses for tool failures, informing users of errors and suggesting fixes.
- Example: If a tool call fails, a `ToolMessage` is generated (e.g., "Error: Invalid flight ID. Please try again.").

## 5.3 Scalability

- **Database**: SQLite is suitable for small-scale applications but may need to be replaced with a more robust database (e.g., PostgreSQL) for large-scale deployments.
- **API Limits**: OpenAI and Tavily API usage must be monitored to avoid rate limits.
- **Concurrency**: The system's sequential design avoids concurrency issues but may need queue-based processing for high user volumes.

### 5.4 Security

- **API Keys**: Stored in environment variables (`OPENAI_API_KEY`, `TAVILY_API_KEY`) to prevent exposure.
- **User Authentication**: The system uses `passenger_id` to verify user ownership of tickets, ensuring secure modifications.
- **Data Validation**: Tools validate inputs (e.g., flight IDs, dates) to prevent invalid operations.

### 5.5 Limitations

- **Multiple Intents**: Currently requires user clarification for simultaneous requests, which may be enhanced with a task queue.
- **Calendar Integration**: Assumes a Google Calendar service; actual implementation requires OAuth setup.
- **Dataset Size**: The toy dataset limits search results; a larger dataset would improve realism.

---

## 6. Future Improvements

1. **Task Queue for Multiple Intents**:
    - Implement a task queue in the `State` class to process multiple requests sequentially without user intervention.
    - Example: Store requests in a list and process them one by one, updating `dialog_state` accordingly.
2. **Enhanced Intent Detection**:
    - Use advanced NLP techniques to prioritize intents automatically based on context or user history.
3. **Database Optimization**:
    - Migrate to a distributed database for better scalability and performance.
    - Implement caching for frequent queries to reduce database load.

4. **User Interface**:
    - Develop a web or mobile interface using React or Flask to enhance user interaction.
    - Integrate with a frontend framework for visual feedback on bookings.
5. **Concurrent Task Handling**:
    - Explore asynchronous task processing using Python's `asyncio` or a workflow engine for parallel task execution.

---

# 7. Conclusion

The Travel Booking System is a robust, AI-driven solution for managing travel-related tasks. Its modular architecture, with a primary assistant and specialized sub-assistants, ensures efficient task delegation and execution. By leveraging LangChain, SQLite, and external APIs, the system provides a seamless user experience for single-task requests. The addition of user clarification for multiple intents addresses the current limitation of concurrent task handling, ensuring reliability. Future enhancements, such as a task queue and database optimization, can further improve scalability and usability.