Architecture Document

Dubai Travel Multi-Agent Assistant

1. Overview

The Dubai Travel Multi-Agent Assistant is a tool-driven AI system designed to provide activity information, handle bookings, and support human-in-the-loop escalation for complex requests.

The architecture separates:

- Conversational reasoning (LLM layer)
- Business logic (tool layer)
- Transport layer (REST API)
- Human escalation (Email + IMAP)
- Presentation layer (React frontend)

This separation ensures modularity, extensibility, and production-style control over AI behavior.

2. High-Level Architecture

The system consists of five primary layers:

1. Frontend (React + Vite)
2. FastAPI Backend
3. Agent Orchestration Layer
4. Business Logic Tools
5. Human-in-the-Loop Email Integration

User → Frontend → FastAPI → Agent → Tool → Database
Supervisor → Email → IMAP Poller → Backend Injection → Frontend

3. Component Breakdown

3.1 Frontend Layer

Technology: React + TypeScript

Responsibilities:

- Displays chat interface
- Polls backend every ~1 second for updated messages
- Renders structured responses
- Extracts and displays image URLs separately from text
- Shows supervisor replies seamlessly in conversation

The frontend is stateless and consumes only REST endpoints.

3.2 API Layer (FastAPI)

Responsibilities:

- Exposes `/chat/{conversation_id}` endpoints
- Stores and retrieves conversation messages
- Routes user input to agent layer
- Provides escalation injection endpoint:
  `/escalations/{conversation_id}/supervisor-reply`

The API layer acts as the boundary between UI and AI logic.

3.3 Agent Layer (LLM Orchestration)

The system uses a tool-driven LLM architecture rather than free-form generation.

The assistant can:

1. Call information tools
2. Call booking tools
3. Call escalation tool

The LLM never directly performs business operations.
Instead, it selects structured tools with arguments.

This ensures:

- Deterministic business logic
- Controlled side effects
- Reduced hallucination risk
- Clear auditability

3.4 Tool Layer (Business Logic)

Tools encapsulate all operational logic:

- `get_activity_tool`
- `book_activity_tool`
- `escalate_to_supervisor_tool`

Each tool:

- Validates inputs
- Enforces constraints (group size, availability)
- Returns structured results
- Updates conversation state

Escalation tool:

- Sends email via SMTP
- Returns structured success/failure response
- Prevents false escalation claims

3.5 Human-in-the-Loop System

This system enables real-world escalation.

Step 1: Escalation
The assistant calls `escalate_to_supervisor_tool`.

Step 2: Email Sent
SMTP sends structured escalation emails.

Step 3: IMAP Polling
A background IMAP process:

- Connects to Gmail
- Searches UNSEEN messages
- Extracts conversation_id
- Cleans reply body

Step 4: Backend Injection
Poller calls:

```
POST /api/escalations/{conversation_id}/supervisor-reply
```

Step 5: Frontend Sync
Front-end polling detects new supervisor messages and renders it.

This design simulates a webhook-based system without requiring public hosting.

4. Multi-Agent Design Principles

The system follows these architectural principles:

4.1 Tool-Based Control

The LLM does not directly mutate state.
All side effects occur through tools.

This ensures:

- Predictable state transitions
- Clear error handling
- Auditable operations

4.2 Separation of Concerns

Layer	Responsibility
Frontend	UI + polling
API	Routing + state storage
Agent	Intent reasoning
Tools	Business rules
IMAP	External reply ingestion

Each component is independently replaceable.

4.3 Human-in-the-Loop Safety

The system enforces:

- Escalation required for discounts
- Escalation required for manager requests
- No false escalation claims allowed

Supervisor responses are injected via backend endpoint rather than direct chat modification.

5. Data Flow Example

Information Request

User → Chat API
Chat API → Agent
Agent → get_activity_tool
Tool → Returns structured activity data
Agent → Formats response
Frontend → Renders text + images

Booking Request

User → Chat API
Agent → book_activity_tool
Tool validates:

- Variation exists
- Availability
- Group size constraints

If valid → Booking confirmed
If invalid → Escalation triggered

Escalation Flow

User → Agent
Agent → escalate_to_supervisor_tool
Tool → Sends SMTP email

Supervisor replies → IMAP detects
IMAP → Backend injection endpoint
Frontend → Displays supervisor message

6. Scalability Considerations

Current design uses:

- In-memory storage
- Polling-based updates
- IMAP polling

For production, improvements would include:

- PostgreSQL for persistence
- WebSocket for real-time updates
- SendGrid webhook instead of IMAP polling
- Background task queue (Celery / Redis)
- Auth & session management
- Rate limiting & logging

7. Why This Is a Multi-Agent System

The system demonstrates:

- Intent classification via LLM
- Tool routing
- Controlled execution environment
- External system integration
- Human override capability

It is not a simple chatbot — it is an orchestrated AI workflow engine.

8. Conclusion

This architecture demonstrates:

- Modular AI system design
- Tool-driven LLM orchestration
- Human-in-the-loop escalation
- Real-time UI synchronization
- Clean separation of AI reasoning and business logic

The system is designed to resemble a production-ready applied AI assistant rather than a prototype chatbot.