**TRIP PLAN AGENTIC AI SYSTEM (Google ADKs)**

**Documentation of Processes**

**A. FINAL TECHNOLOGY STACK:**

**1. Core Components**

| **Layer** | **Technology** | **Notes** |
| --- | --- | --- |
| **Frontend** | Not included **(Local ADK Web Console)** | Will use **Local ADK Web Console** during development/testing |
| **Backend** | **Google ADKs (Python)** | Agent logic implemented via ADK’s event → request → response format |
| **Database** | **Local JSON-based lightweight DB** | Stored locally via **Database Session Service** (no SQL/Mongo DBs needed) |
| **LLM Models** | **OpenAI** or **Ollama (Local)** | For conversation generation, fallback handling, and natural language flow |

**2. Agentic Architecture**

Main Agents:

| **Agent** | **Purpose** |
| --- | --- |
| **Supervisor Agent** | Orchestrates requests, manages flow, routes to sub-agents (Delegation agent – MAIN) |
| **Accommodation Agent** | Handles accommodation search/booking logic |
| **Travel Agent** | Manages travel (flights, trains, cabs) booking |
| **Sightseeing Agent** | Manages local activities, passes & booking |

Optional/Plug-in Agents:(Mainly for concurrency maintenance and conflict scheduling)

| **Agent** | **Purpose** |
| --- | --- |
| **Conflict Checker Agent** | Checks conflicts in date/location/price across users |
| **Conversation Agent** | Handles clarification conversations with users |

**3. Agent Workflow Type**

| **Type** | **Inclusion** | **Purpose** |
| --- | --- | --- |
| **Stateful Multi-Agent Workflow** | Yes | Stores user context using user-id, maintains conversation + session |
| **Sequential Agent Workflow** | Yes | Accommodation → Travel → Sightseeing booking logic |
| **Parallel Agent Workflow (Partial)** | Optional | Possible light parallel calls where dependencies are loose (ex: sightseeing) |
| **A2A Message-Passing (Lightweight)** | Yes | Agents pass state or context to each other as needed via Supervisor |

**4. Data Storage & Session**

| **Storage Type** | **Use Case** |
| --- | --- |
| **In-Session Store** | For short-term memory and active sessions per user (within session use) |
| **Local JSON Store** | For persistent state (user data, history, bookings) – MAIN USAGE DB |
| **No External DBs** | Simple and local setup for ease of use |

**5. Tools & Utilities**

| **Tool** | **Use Case** |
| --- | --- |
| **ADK Web Console** | For local dev & testing without frontend UI |
| **Simple CLI/Terminal UI** | Optional wrapper for ad-hoc testing |
| **Logging** (text or file) | Debugging agent flow and rollback scenarios |

**6. Rollbacks & Conflict Handling**

* Built into **Supervisor Agent**
* Supervisor queries **Conflict Checker Agent** to resolve booking issues
* If clash → initiates rollback to previous state (e.g., accommodation → retry)

**B. BASIC WORKFLOW:**

User wants to book for trip (accommodation/travel/sightseeing) – Types in the requirement in frontend window (for now adk web console – locally run, later on frontend can be developed) – Supervisor (main) agent looks at request and delegates task to the appropriate agent – Sub agent looks at the tools available and if the description of tool’s task and the requirement match it is given to the tool – Else it is passed to the LLM model which would generate a proper appropriate response with accordance to the requirement – Once the task is completed it throws the response back to the main agent (in appropriate output format which is described) which is then reflected as the response from the main agent in web console (later, frontend) – LLMs also have querying capabilities (Cue Google Search built-in tool, for answering based on general knowledge on trip information) through which users can ask for their information needed – Displaying their info before and after the particular state transaction (in proper output format)

**\*THINGS TO KEEP IN MIND HERE**

* Since many users can use this app at any time and they have a particular user-id through they are identified, it has to be a stateful multi agent workflow (maybe coupled along with sequential or parallel agent workflow).
* DB – User data has to be stored and updated over a period of time.  
  Operations possible on every agent – CRUD (Create/Read [or View]/Update/Delete)  
  MOST IMPORTANTLY FOCUS ON THESE X-FACTORS:
* System must be able to display other users’ data for clarification upon clash of dates/location/price etc.
* CONVERSATIONS & ROLLBACKS – This would be key in mostly parallel workflows, where the order would most probably be accommodation – travel – sightseeing; if any of these doesn’t match the other, i.e, if sightseeing plans don’t match the accommodation (some kind of clash with dates/location/price), it has to roll back to accommodation to look at a new/fresh combination. The user can either be specific with the price range/location/dates or have a conversation with the main agent (or sub agents) regarding the requirements.

**C. BASIC HIERARCHY & WORKFLOW:**

1. Session Service → Sessions → [States + Events]

2. Runner = Agents + Sessions

3. Whole Workflow (3 Components):

EVENT – User Message (Input)

REQUEST – User Message → Runner → (MAIN) Agent → Sub Agents → Tools (or) LLM

RESPONSE – Tools (or) LLM → Sub Agents → (MAIN) Agent → Runner (Update Session) → Agent Response

**D. CHECK-LIST:**

**I. Environment Setup**

| **Task** | **Description** |  |
| --- | --- | --- |
| ☐ Create Python virtual environment | Use venv for isolation |  |
| ☐ Install dependencies | adk, openai/ollama, json, uuid, logging, etc. |  |
| ☐ Setup project structure | Folder scaffolding: agents/, data/, utils/, logs/, sessions/ |  |
| ☐ Setup ADK Web Console | Verify CLI + session runner works |  |

**II. Design & Architecture**

**A. Agentic System Design**

| **Task** | **Description** |  |
| --- | --- | --- |
| ☐ Implement **Supervisor Agent** | Central control agent for routing, rollback and orchestration |  |
| ☐ Implement **Accommodation Agent** | Handles stay bookings and CRUD |  |
| ☐ Implement **Travel Agent** | Manages transport bookings and CRUD |  |
| ☐ Implement **Sightseeing Agent** | Handles local attractions and activity planning |  |
| ☐ (Optional) Implement **Conflict Checker Agent** | Identifies clashes in date, price or location |  |
| ☐ (Optional) Implement **Conversation Agent** | Clarifies ambiguous user queries |  |
| ☐ Setup agent-to-agent (A2A) message passing | Via Supervisor routing logic |  |
| ☐ Design state schema | Define agent input/output contract:  event → request → response |  |

**B. Session + Storage**

| **Task** | **Description** |  |
| --- | --- | --- |
| ☐ Create session service handler | In-memory + JSON-backed per-user sessions |  |
| ☐ Design local DB schema | JSON file per user or collective structure for all bookings |  |
| ☐ Implement session save/load logic | Session persistence to disk |  |
| ☐ Handle in-session (temp) memory | Use for live chat handling without writing immediately to disk |  |

**III. Agent Functionality (CRUD + Other Functions)**

| **Agent** | **CRUD Tasks** |
| --- | --- |
| Accommodation Agent | ☐ Create booking ☐ View/search stay ☐ Update stay ☐ Cancel booking |
| Travel Agent | ☐ Create trip ☐ View routes ☐ Modify ticket ☐ Delete reservation |
| Sightseeing Agent | ☐ Plan activities ☐ Browse listings ☐ Reschedule ☐ Cancel |
| Supervisor Agent | ☐ Session management ☐ Delegation |
| Conflict Checker Agent | ☐ Fetch other users’ trips ☐ Compare/validate trip data |
| Conversation Agent | ☐ Rephrase unclear requests ☐ Ask for more info ☐ Handle fallback |

**IV. Workflow Pipeline**

| **Component** | **Task** |
| --- | --- |
| ☐ Event | Accept user message through ADK Web Console |
| ☐ Request | Parse → Session + Runner → Supervisor Agent |
| ☐ Delegation | Supervisor routes request to correct sub-agent |
| ☐ Sub-Agent Tasking | Calls internal logic/tool or escalates to LLM |
| ☐ LLM Use | Query OpenAI/Ollama only if no tool matches |
| ☐ Response | Returned through Sub-Agent → Supervisor → Session Update |

**V. Intelligence + Rollback Logic**

| **Task** | **Description** |
| --- | --- |
| ☐ Maintain rollback states | Supervisor retains last valid snapshot of booking state |
| ☐ Conflict detection logic | Conflict Checker compares dates/locations/costs with other users |
| ☐ Implement “retry” fallback | Supervisor rolls back to earlier agent state and re-executes |
| ☐ LLM clarification flow | Triggered via Conversation Agent if user input is vague |

**VI. Data & Mock API Design**

| **Task** | **Description** |
| --- | --- |
| ☐ Create mock JSON data | accommodations.json, travels.json, activities.json |
| ☐ Dummy user bookings | Add sample users to simulate conflict checks |
| ☐ Tools/logic for mock querying | Simple filter/search on mock data for agents |

**VII. Testing Plan (LATER – FINAL PHASES)**

| **Phase** | **Tests** |
| --- | --- |
| Phase 1 | ☐ One-agent flow (Accommodation only) ☐ Session tracking |
| Phase 2 | ☐ Full sequential flow (Accommodations → Travel → Sightseeing) |
| Phase 3 | ☐ Conflict rollback tests |
| Phase 4 | ☐ A2A passing test ☐ LLM fallback test |
| Phase 5 | ☐ Multi-user session tests (2–3 users concurrently) |

**E. PROJECT STRUCTURE:**

trip\_planner\_agent/

│

├── \_\_init\_\_.py # Required for ADK discovery

├── agent.py # Root 'supervisor' agent definition

│

├── sub\_agents/ # All trip-specific sub-agents

│ ├── \_\_init\_\_.py

│ ├── accommodation\_agent/

│ │ ├── \_\_init\_\_.py

│ │ └── agent.py

│ │

│ ├── travel\_agent/

│ │ ├── \_\_init\_\_.py

│ │ └── agent.py

│ │

│ ├── sightseeing\_agent/

│ │ ├── \_\_init\_\_.py

│ │ └── agent.py

│ │

│ ├── conflict\_checker\_agent/

│ │ ├── \_\_init\_\_.py

│ │ └── agent.py

│ │

│ └── conversation\_agent/

│ ├── \_\_init\_\_.py

│ └── agent.py

│

├── tools/ # General tools used by agents

│ ├── \_\_init\_\_.py

│ ├── llm\_interface.py # OpenAI/Gemini abstraction

│ └── search\_tool.py # Web search / info lookup

│

├── core/ # Infra components like session + DB

│ ├── \_\_init\_\_.py

│ ├── session\_service.py # Persistent session storage

│ ├── db.py # Local JSON-based DB

│ └── utils.py # Shared helpers

│

├── .env # API keys / config vars

├── main.py # Entry point – runs in console

├── requirements.txt # All dependencies