Project Documentation: Agentic AI-Powered Hospitality & Travel Management System

1. Abstract

The traditional approach to travel planning requires users to manually navigate multiple platforms (Airbnb, MakeMyTrip, Skyscanner) to synthesize itineraries, budgets, and logistics. This project introduces a fully autonomous, multi-agent Hospitality Management System built on the CrewAI framework. By leveraging a dual-LLM architecture (Grok API for high-level reasoning/tool execution and a local Ollama model for natural language synthesis), the system autonomously scrapes real-time data, processes constraints (budget, group type), and generates highly optimized, day-by-day travel itineraries.

2. System Architecture

The platform follows a decoupled architecture, separating the agentic reasoning layer from the data persistence layer.

2.1 Technology Stack

- **Orchestration Framework:** CrewAI
- **Large Language Models (LLMs):** Grok API (Primary logic/Tool calling), Ollama (Local synthesis)
- **Search & Scraping Tools:** Tavily API, SerpAPI, Selenium (Dynamic JS scraping)
- **Vector Database (Semantic Memory):** FAISS / ChromaDB
- **Document Database (Unstructured Data):** MongoDB
- **Relational Database (Structured State):** MySQL

---

3. Multi-Agent System Design

The core of the application operates on a Sequential Process orchestrated by CrewAI, utilizing two specialized AI agents.

3.1 The Researcher Agent (Logistics & Data Gathering)

- **Role:** Lead Logistics & Hospitality Researcher
- **Assigned LLM:** Grok API
- **Responsibilities:**
  - Execute external searches using `TavilySearchTool` to identify top attractions, landmarks, and operational days.
  - Utilize `SerpApiTool` and a custom `SeleniumDynamicScraper` to extract real-time flight charges, cab availability, and hotel pricing from MakeMyTrip and Airbnb.
  - Filter gathered data based on user constraints: Budget, Location, and Travel Type (Friends, Family, Business).
- **Output:** A structured JSON object containing verified, raw travel logistics.

3.2 The Writer Agent (Synthesis & Formatting)

- **Role:** Itinerary Architect & Travel Writer
- **Assigned LLM:** Ollama (Local Model)
- **Responsibilities:**
  - Ingest the raw data JSON provided by the Researcher Agent.
  - Synthesize the data into a cohesive, user-friendly format highlighting what is "Special about that place".
  - Map out exact travel timings and calculate the cumulative daily budget.
- **Output:** A strict `Pydantic` schema representing the `FinalItinerary`, ready for database insertion.

---

4. Data Storage and Persistence Strategy

To optimize data retrieval and prevent redundant API calls, the system utilizes a tri-database architecture.

4.1 MySQL (Structured Application State)

Acts as the core relational database for the application.

- Stores User Profiles and historical trip preferences.
- Stores the final, generated `DayWiseSchedule` and calculated budgets.

4.2 MongoDB (Unstructured Staging Cache)

Acts as a high-speed dump for the scraping tools.

- Stores raw HTML/JSON payloads from Selenium and SerpAPI.
- Caches unstructured user reviews and hotel descriptions to minimize repetitive, costly scraping.

4.3 FAISS / ChromaDB (Vector Knowledge Base)

Acts as the semantic memory for the AI agents.

- Stores vector embeddings of landmark descriptions, restaurant menus, and cultural insights.
- Allows the Researcher Agent to perform similarity searches (e.g., finding "kid-friendly restaurants near Landmark A").

5. Execution Flow & Tool Integration

1. **User Input:** The system receives a prompt detailing the destination, budget, and group type.
2. **Context Retrieval:** System queries FAISS/Chroma to check if similar itineraries or landmark data already exist in the vector space.
3. **Data Acquisition:** The Researcher Agent invokes Tavily, SerpAPI, and Selenium to fetch missing real-time data (e.g., current Skyscanner flight prices).

4. **Data Staging:** Raw scraped data is pushed to MongoDB.
5. **Synthesis:** The Writer Agent processes the staged data into a final itinerary.
6. **State Persistence:** The completed itinerary is validated via Pydantic and saved to MySQL for user retrieval.

---

6. Future Scope and Optimizations

- **Dynamic Cost Re-calculation:** Implementing a cron job to periodically update cached flight and hotel prices in MongoDB.
- **Agentic UI/UX:** Connecting the MySQL backend to a React/Next.js frontend where users can chat with the itinerary to make micro-adjustments in real-time.
- **Rate Limit Management:** Enhancing the Selenium scraper with rotating proxies to bypass anti-bot mechanisms on major travel portals.

Appendix A: Database Schemas

A.1 MySQL Schema (Structured Relational Data)

This relational structure acts as the application's source of truth, maintaining strict referential integrity for users, their trips, and the generated day-to-day schedules.

```
-- 1. Users Table: Stores user profiles and preferences
CREATE TABLE Users (
    user_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    default_currency VARCHAR(3) DEFAULT 'INR',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Itineraries Table: Stores the high-level trip details generated by the system
CREATE TABLE Itineraries (
    itinerary_id VARCHAR(50) PRIMARY KEY,
    user_id VARCHAR(50) NOT NULL,
    destination VARCHAR(100) NOT NULL,
    group_type ENUM('Family', 'Friend', 'Business') NOT NULL,
    total_budget_allocated DECIMAL(10, 2),
    total_transport_cost DECIMAL(10, 2),
    hotel_name VARCHAR(255),
    special_about_place TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE
);
```

```
-- 3. Daily Schedules Table: Stores the day-by-day breakdown written by the Writer Agent
CREATE TABLE Daily_Schedules (
    schedule_id INT AUTO_INCREMENT PRIMARY KEY,
    itinerary_id VARCHAR(50) NOT NULL,
    day_number INT NOT NULL,
    attractions_to_visit JSON, -- Stores list of landmarks
    nearest_restaurants JSON,  -- Stores list of restaurants
    travel_timing VARCHAR(255),
    daily_budget_estimate DECIMAL(10, 2),
    FOREIGN KEY (itinerary_id) REFERENCES Itineraries(itinerary_id) ON DELETE CASCADE
);
```

A.2 MongoDB Schema (Unstructured Staging Data)

MongoDB is utilized as a flexible staging area. Because the Researcher Agent pulls dynamic data from APIs (SerpAPI) and web scrapers (Selenium), the schema must be schema-less to accommodate varying JSON structures.

Collection 1: `ScrapedContext` Stores the raw dumps from your scrapers. This acts as a cache so you don't have to re-scrape Airbnb or Skyscanner if another user searches for the same route on the same day.

JASON

```
{
  "_id": "ObjectId('...')",
  "search_parameters": {
    "destination": "Goa, India",
    "source_platform": "MakeMyTrip",
    "data_type": "Hotel Pricing"
  },
  "scraped_at": "ISODate('2026-02-15T10:00:00Z')",
  "raw_payload": {
    "hotels": [
      {
        "name": "Taj Exotica",
        "price_per_night": 15000,
        "amenities": ["Pool", "Beachfront", "Spa"],
        "dynamic_pricing_flag": true
      }
    ],
    "page_html_dump": "<raw html string if needed for debugging>"
  },
  "ttl_expiration": "ISODate('2026-02-16T10:00:00Z')" // Auto-deletes old prices
}
```

Collection 2: `LandmarkInsights` Stores the heavy text blocks retrieved by the Tavily Search Tool, which will later be vectorized into FAISS/Chroma.

```
{
  "_id": "ObjectId('...')",
  "landmark_name": "Gateway of India",
  "location": "Mumbai",
  "operational_days": "Monday - Sunday",
  "special_about_place": "A monumental arch built in the 20th century...",
  "transport_options": [
    {"mode": "Cab", "avg_cost": 300, "currency": "INR"},
    {"mode": "Local Train", "avg_cost": 20, "currency": "INR"}
  ],
  "raw_reviews": [
    "Great place for family visits, but very crowded on weekends.",
    "Best visited during early morning."
  ]
}
```