Summary

You will design and implement an **Agentic Retrieval-Augmented Generation (RAG)** chatbot that serves as a travel guide by ingesting a travel-focused knowledge base and orchestrating multiple autonomous agents via the **CrewAI** framework. Your system will index and retrieve travel documents (e.g., hotel reviews, itineraries) using a vector database, then generate user-centric recommendations. You'll wrap the chatbot in a **Streamlit** UI and expose it securely over the internet using **ngrok**, enabling live demos. This project demonstrates end-to-end skills in data ingestion, vector search, multi-agent orchestration, and web deployment.

Description

- **Objective:** Build a multi-agent RAG travel guide chatbot that:

    1. Processes and indexes travel documents (e.g., hotel reviews, itineraries).
    2. Retrieves relevant information using vector search.
    3. Coordinates multiple CrewAI agents (retriever, summarizer, response-composer).
    4. Presents a conversational interface via Streamlit and ngrok.

- **Scope:**

    1. **Data Layer:** Ingest a travel dataset (e.g., Bitext Travel LLM corpus or TripAdvisor hotel reviews).
    2. **RAG Pipeline:** Implement vectorization, indexing (e.g., Qdrant), and retrieval logic.
    3. **Agentic Orchestration:** Configure CrewAI agents for retrieval, filtering, and response crafting.
    4. **Deployment:** Serve the app locally with Streamlit and tunnel it using ngrok.

Requirements

1. Environment Setup

- **Python ≥3.9** with virtual environment.

Install **CrewAI**:
Install **vector database client** (e.g., Qdrant or Pinecone):
Install **Streamlit** and **pyngrok**:

2. Data Ingestion

- **Dataset Selection:**
    - Option A: **Bitext Travel LLM Chatbot Training Dataset** (4.16M tokens) on Hugging Face
    - Hugging Face.
    - Option B: **TripAdvisor Hotel Reviews** (20K reviews) on KaggleKaggle.
- **Preprocessing:**
    - Clean and normalize text (remove HTML, special characters).
    - Split into document chunks (≈500 tokens each).
    - Serialize chunks with metadata (source, timestamp, category).

3. Vector Indexing & Retrieval

- **Vectorization:** Embed each chunk using an open-source embedding model (e.g., OpenAI embeddings or Cohere).
- **Indexing:** Ingest embeddings into Qdrant (or alternative) with semantic metadata filters (e.g., hotel vs. flight data).
- **Retrieval Agent:** Configure a CrewAI agent that accepts a user query and returns the top-k semantically similar chunks

  Qdrant.

4. Response Generation

- **Summarizer Agent:** Aggregate retrieved chunks, removing redundancy.
- **Composer Agent:** Leverage an LLM (OpenAI GPT-4 or similar) to generate the final user-facing response, formatted as a travel itinerary or recommendation.

5. Streamlit Frontend

- Build an interactive UI with:
    - **Text input** for user queries.
    - **Chat window** showing conversation history.
    - **Sidebar** for agent-chain status or logs.

Deliverables

Component	What to Submit
1. Environment Setup	- `requirements.txt` listing all dependencies - Setup instructions (`README.md`)
2. Data Ingestion	- Jupyter notebook (`data_ingest.ipynb`) showing dataset download, cleaning, chunking - Sample processed JSON or CSV files
3. Indexing & Retrieval	- Script/notebook (`index_and_retrieve.py`) demonstrating embedding and upload to Qdrant - Retrieval demo (unit tests or example queries)
4. Agentic Orchestration	- CrewAI configuration file (`crew_config.yml` or Python script) - Agent definitions with comments explaining each role
5. Response Generation	- Python module (`compose_response.py`) invoking the summarizer and composer agents - Example prompts and outputs

6. Streamlit App	- `app.py` with Streamlit code - Screenshot of UI layout - `ngrok` launch script or instructions
7. Documentation & Demo	- **Video walkthrough** (5–7 min) showing end-to-end flow - **Project report** (PDF) describing design decisions, architecture diagram, and future improvements