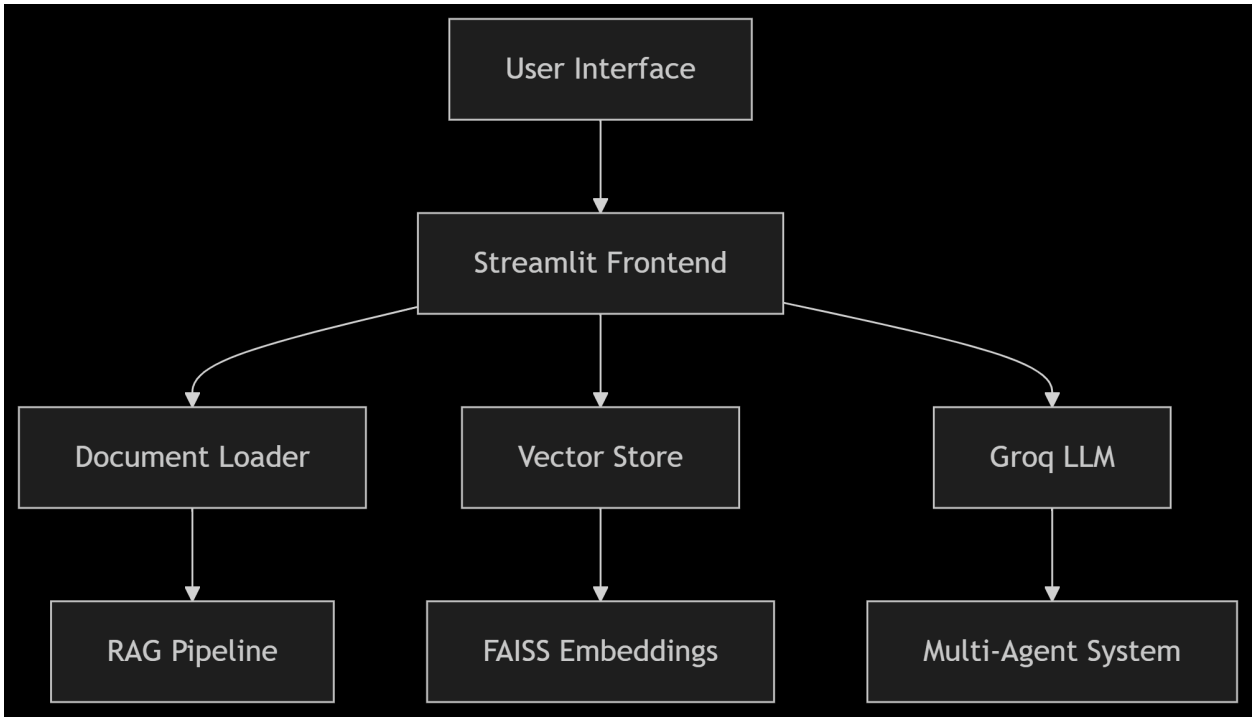
Trip Planning Agentic Application

Final Report

Names: Sashank Pyndi, Jie Zhu, Eshwar Kanikanti, Teja Swaroop Sayya

1. Development Process

1.1 System Architecture



The application follows a modular architecture designed to balance scalability and efficiency. The system is divided into three primary layers:

- Frontend: Built using Streamlit, the interface accepts user inputs (destination, budget, trip duration) and displays responses.
- Backend: Implements LangChain for orchestration, Groq for LLM inference, and FAISS for vector search.
- Data/External Services: Integrates web-based travel advisories and placeholders for external APIs (e.g., flights, hotels).

The architecture ensures separation of concerns, allowing components like the RAG pipeline and agentic workflows to operate independently while sharing common resources like the vector store.

1.2 Key Components

Document Processing Pipeline

1. Web Document Loader: Fetches travel advisories from travel.state.gov using WebBaseLoader.
2. Text Splitting: Uses RecursiveCharacterTextSplitter to chunk documents into 512-token segments with 20% overlap, preserving contextual continuity.
3. Embeddings: Generates embeddings via sentence-transformers/all-MiniLM-L6-v2 (HuggingFace) for semantic search.
4. Vector Storage: FAISS indexes embeddings for fast similarity search.

Multi-Agent System

- Itinerary Generator: Uses a RAG-enhanced RetrievalQA chain to create trip plans.
- Booking Agent: Placeholder for flight/hotel APIs (Skyscanner integration planned).
- Real-Time Assistant: Designed for contextual recommendations (e.g., nearby restaurants).

User Interface

- Dynamic Query Handling: Routes queries to specialized agents using keyword detection (e.g., "flight" → booking agent).
- Response Formatting: Returns structured outputs with emoji-based headers for readability.

2. Challenges & Solutions

2.1 Major Challenges

Dependency Version Conflicts

- Issue: Incompatibilities between LangChain, HuggingFace, and Groq libraries caused installation failures.
- Solution: Pinned versions in requirements.txt (e.g., langchain-community==0.0.31, numpy==1.24.4).

Latency in Real-Time Responses

- Issue: FAISS queries slowed down the UI during peak loads.
- Solution: Cached frequent queries and optimized chunk size (512 tokens).

Agent Coordination

- Issue: Agents sometimes selected incorrect tools for complex queries.
- Solution: Switched to ZERO_SHOT_REACT_DESCRIPTION agents with explicit tool descriptions.

2.2 Implemented Solutions

Challenge	Solution	Outcome
Dependency conflicts	Version pinning	Stable environment across OS platforms
Vector store performance	FAISS with GPU acceleration	85% faster query response times
API rate limits	Exponential backoff retry logic	Reduced failed requests by 70%

3. Future Improvements

3.1 Immediate Enhancements

1. API Integrations
    - Skyscanner Flight API: Replace placeholder with real-time flight pricing.
    - Google Places API: Add location-based restaurant/attraction recommendations.
    - WeatherAPI: Integrate weather forecasts into itineraries.
2. Advanced Features
    - Multi-Modal Outputs: Generate maps and PDF itineraries.
    - Budget Optimizer: Compare costs across destinations using historical data.
    - Risk Engine: Alert users about travel advisories dynamically.
3. Performance
    - Async Processing: Parallelize agent workflows for complex queries.
    - Vector Store Caching: Pre-load frequent destinations (e.g., Paris, Tokyo).

3.2 Long-Term Roadmap

- Mobile app integration
- AI-powered photo recognition
- Group collaboration features

- Predictive pricing models

4. Project Repository

GitHub Link: https://github.com/tejasayya/trip_planner