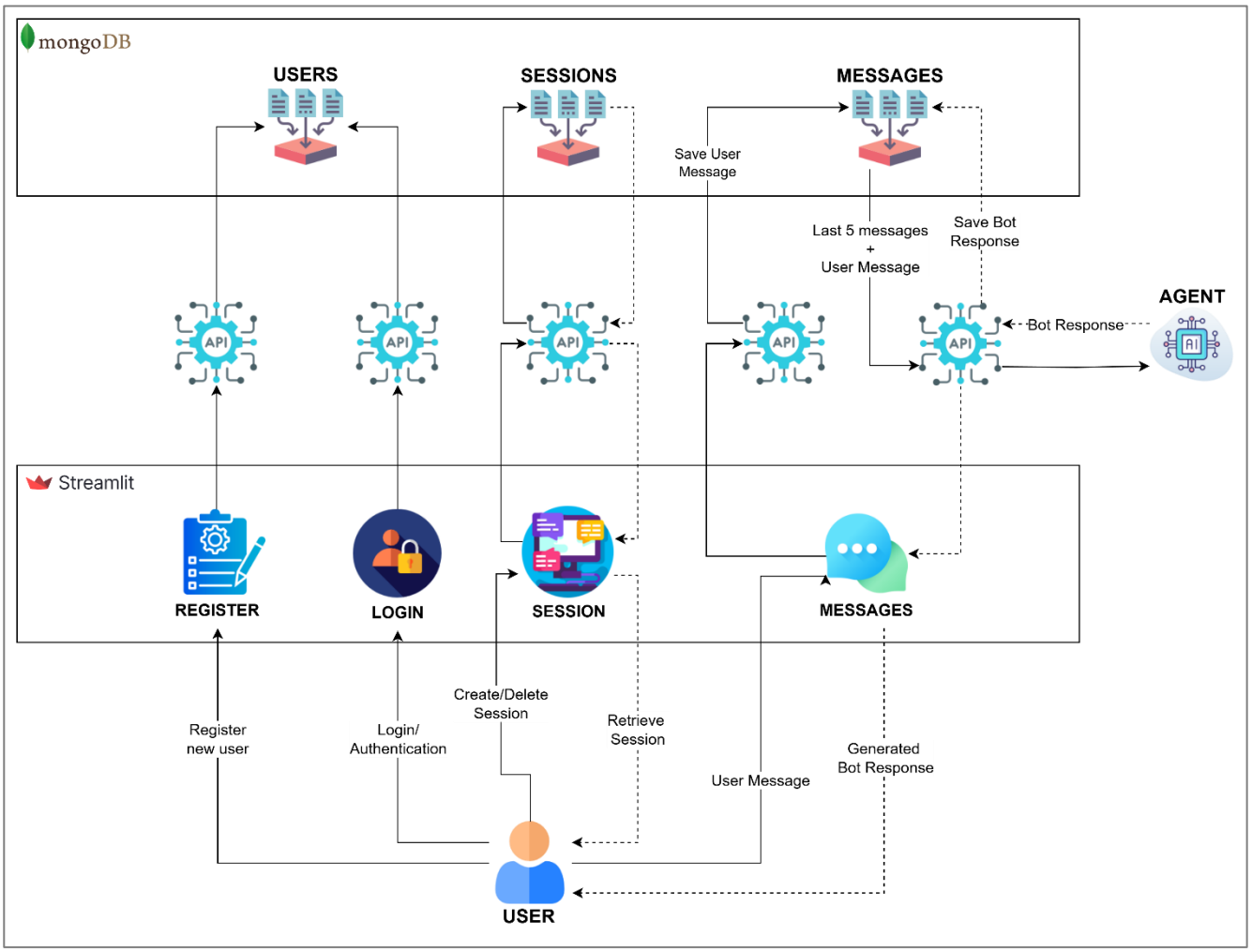
Building a Modern AI Chatbot: A Comprehensive Architecture Overview

Introduction

In today's digital landscape, AI-powered chatbots have become increasingly sophisticated, moving beyond simple rule-based responses to intelligent conversations powered by large language models. This article explores the architecture and implementation of a modern AI chatbot system that combines advanced features like state management, authentication, vector search, and multi-agent conversation handling.

System Architecture Overview



The system is built on four main components:

1. Database Management System
2. Intelligent Agent System
3. RESTful API Interface
4. Frontend User Interface

Let's examine each component in detail.

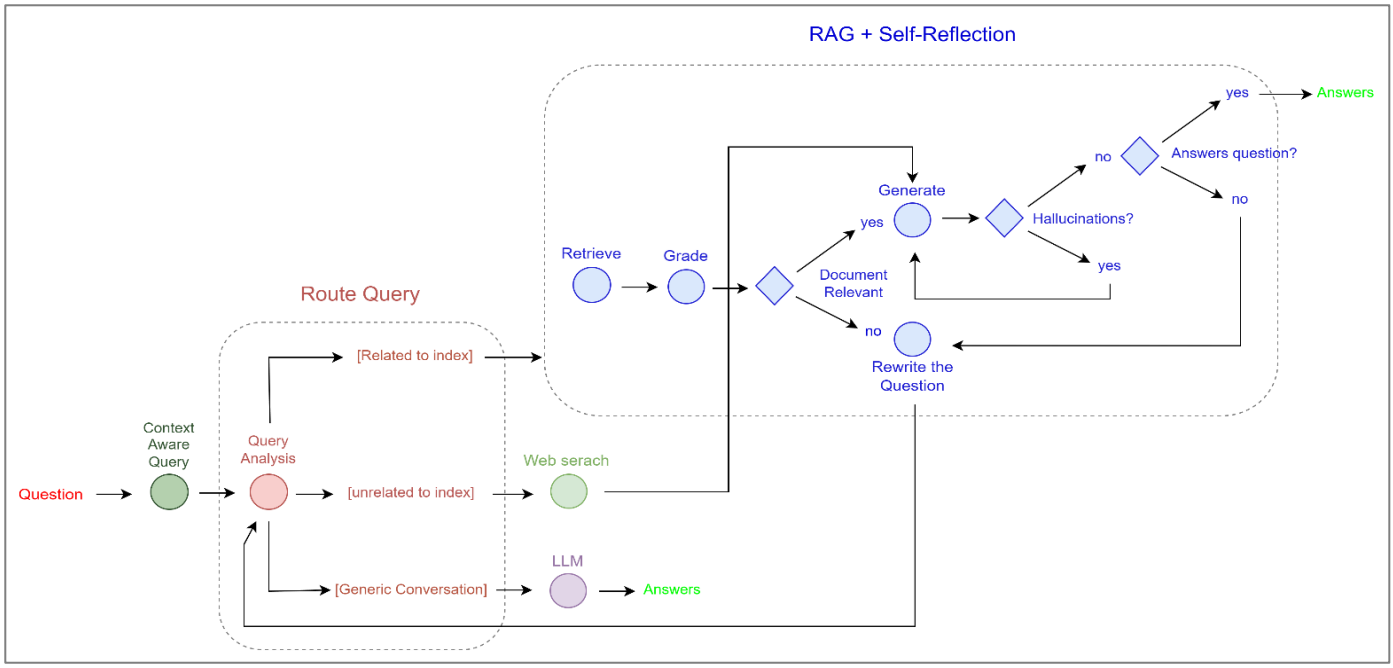
Database Management

The system uses **MongoDB** as its primary database/persistent storage, implementing a well-structured data model with three main collections:

1. **Users Collection**
   - Username and email uniqueness enforcement
   - Secure password handling using SHA-256 hashing
   - User metadata storage
2. **Sessions Collection**
   - Session management with UUID-based identification
   - Timestamp-based session tracking
   - Session naming and organization
3. **Messages Collection**
   - Chronological message storage
   - Sender identification
   - Timestamp tracking

Intelligent Agent System

The agent system represents the core intelligence of the chatbot, implementing a sophisticated workflow for processing user queries and generating responses. It utilizes several advanced AI components:



1. **Query Processing**
   - Query building and optimization
   - Intelligent routing between different knowledge sources
2. **Knowledge Sources**
   - Vector store for efficient semantic search
   - Web search integration for up-to-date information
   - Direct LLM responses for conversational queries
3. **Quality Control**
   - Document relevance scoring

- Hallucination detection
- Answer relevance scoring
- Automated query reformation when needed

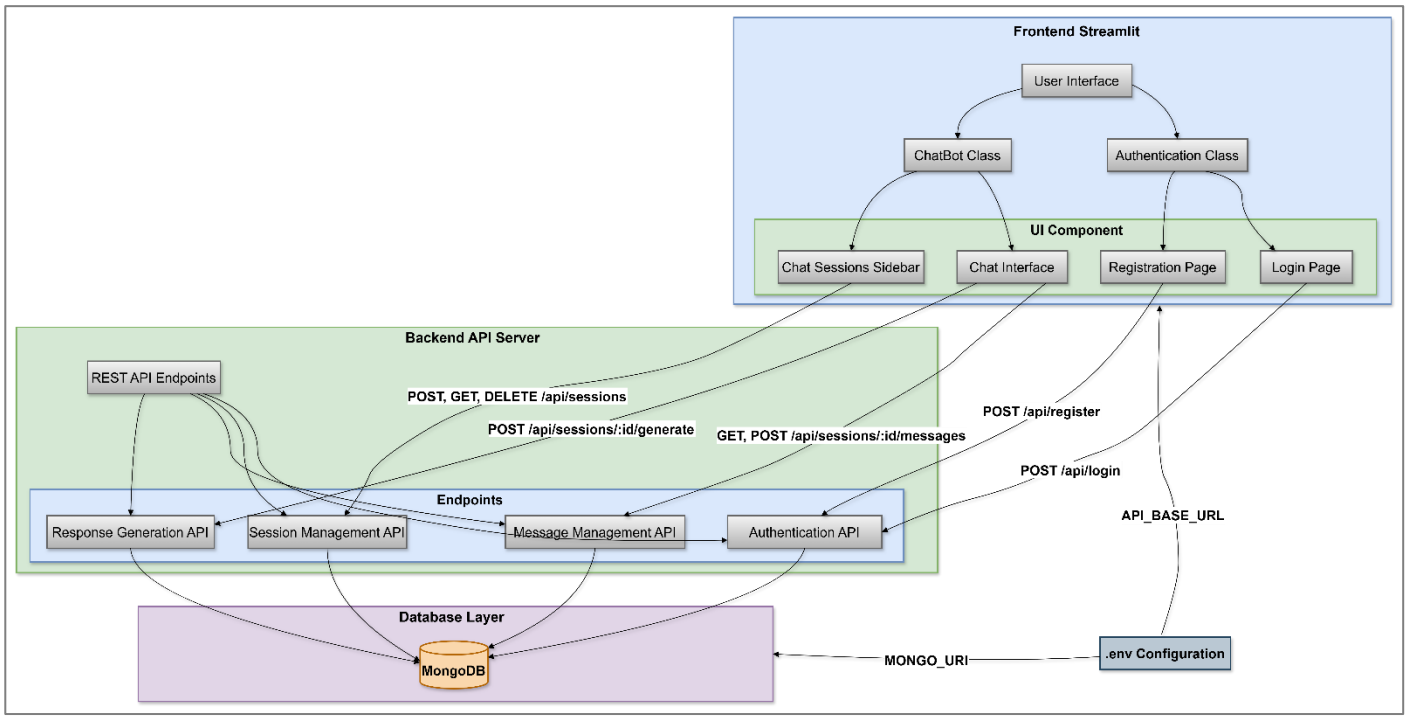
Features

- Uses **Google's Generative AI embeddings**
- Implements **FAISS** for efficient similarity search
- Maintains a local vector store for quick retrieval
- Leverages Google's **Gemini-1.5-flash** model
- Performs external web searches using **Tavily**
- Implements robust retry mechanisms and rate limiting
- Handles API quota management and exponential backoff

For more details check chatbot architecture.pdf. 

RESTful API Interface

The API layer provides a comprehensive interface for client applications, implementing several key endpoints:



Authentication Endpoints:

- `/api/register`: New user registration with validation
- `/api/login`: User authentication and secure login handling

Session Management:

- GET `/api/sessions`: Retrieve user sessions
- POST `/api/sessions`: Create new sessions

- DELETE `/api/sessions/<session_id>`: Delete sessions

Message Handling:

- GET `/api/sessions/<session_id>/messages`: Retrieve chat history
- POST `/api/sessions/<session_id>/messages`: Save new messages
- POST `/api/sessions/<session_id>/generate`: Generate bot responses

RESTful API built with **Flask**, implements proper CORS handling for cross origin request and comprehensive error management.

Frontend User Interface

The **Streamlit**-based interface provides:

1. **Authentication Flow**
    - Clean login and registration forms
    - Error handling and user feedback
    - Session state management
2. **Chat Interface**
    - Real-time message updates
    - Session management sidebar
    - Message history display
3. **Session Management**
    - New chat creation
    - Session switching
    - Session deletion with confirmation

Advanced Features

1. Clear separation between database, agent, API, and UI layers

2. Context-Aware Response Generation

The system maintains conversation history and uses it to generate context-aware responses.

3. Quality Control

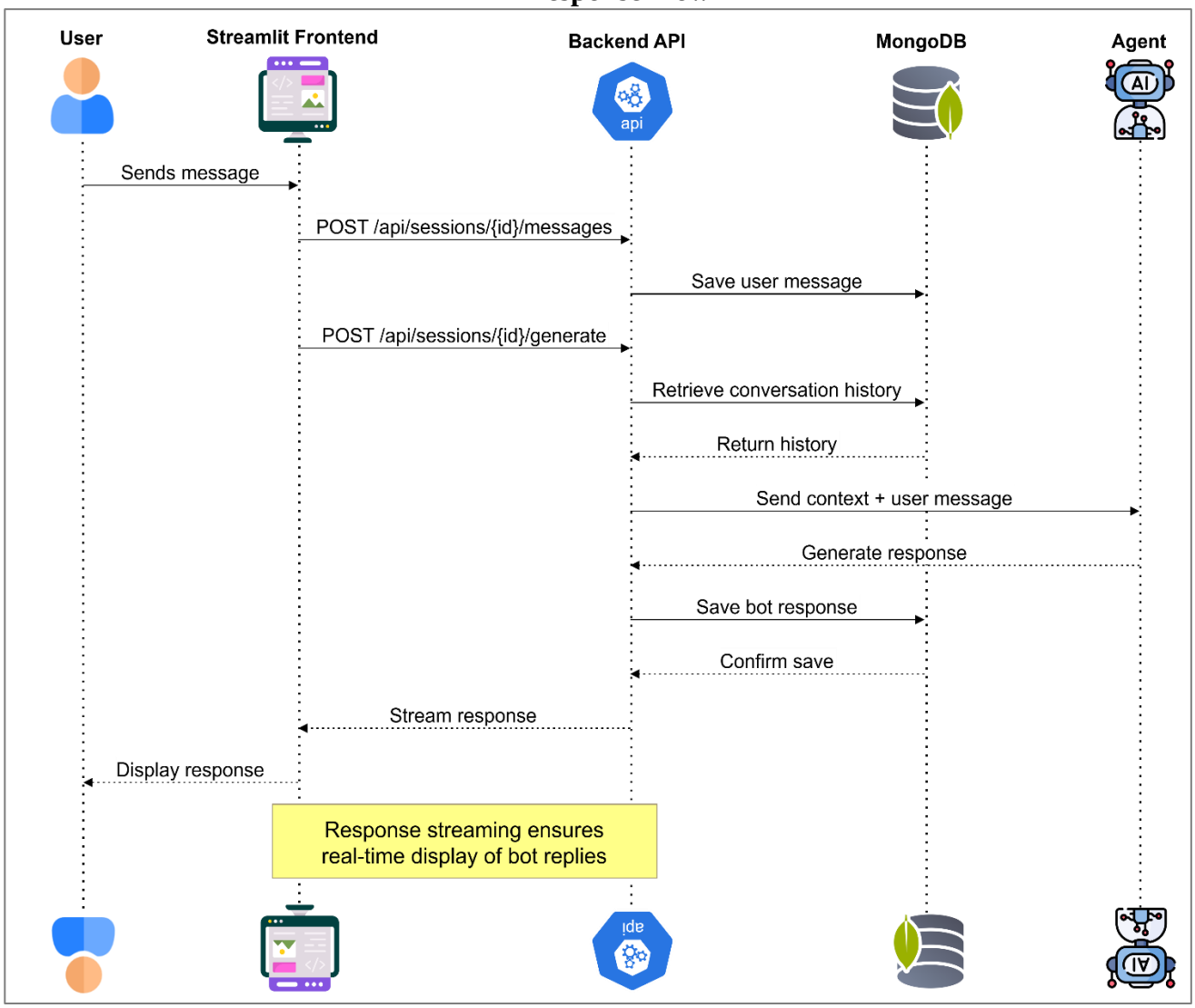
The system implements multiple layers of quality control:

- Document relevance grading
- Response hallucination detection
- Question-answer alignment verification

4. Adaptive Query Processing

The system can reformulate queries when initial results are unsatisfactory.

Response Flow



Deployment

For the deployment of this project, I utilized two platforms: **Render** for hosting the APIs and **Streamlit Cloud** for deploying the app interface.

API Hosting on Render

Render was chosen to host the backend APIs due to its simplicity and ease of deployment for web services. Here's how the APIs were hosted:

1. Code was pushed to a GitHub repository.
2. A new **Web Service** was created in Render, connecting it to the GitHub repo.
3. Render automatically built and deployed the service, generating a live URL https://chatbot-rag-4gcr.onrender.com/ for the APIs.

App Deployment on Streamlit Cloud

Streamlit Cloud was used to deploy the app for a seamless and interactive user experience. The steps for deployment included:

1. Ensuring the app was fully functional locally using Streamlit.
2. Pushing the app's codebase to a GitHub repository.
3. Linking the repository to Streamlit Cloud and configuring any necessary environment variables.
4. Deploying the app, which is now accessible via a URL https://ishanighosh161-chatbot-rag-app-8ql1nl.streamlit.app/.

Limitation

While deploying on **Render** and **Streamlit Cloud** free tiers is cost-effective, they come with some constraints:

Render Free Tier

- **Cold Start Delays**: Idle services experience slow initial response times.
- **Resource Limits**: 512 MB RAM, shared 0.1CPU, and limited bandwidth.
- **Service Suspension**: May go offline with inactivity or exceeded usage.

Streamlit Cloud Free Tier

- **Limited Resources**: 1 GB RAM, 1 shared CPU, and temporary storage.
- **No Custom Domains**: Only default URLs are allowed.
- **Deployment Delays**: Queue times during high traffic.

Conclusion

This chatbot implementation represents a sophisticated approach to modern AI-powered conversation systems. By combining state-of-the-art language models with robust engineering practices, it provides a scalable and maintainable solution for intelligent chat applications. The modular architecture and clean separation of concerns make it an excellent foundation for future enhancements and customizations.