Documentation

RAG-Voice-Agent

this is an AI voice Agent that answers according to the uploaded documents to it's database you may use it as a customer service agent

built using these technologies:
1-LiveKit - Real-time communication platform
2-LiveKit SDK (Python & JavaScript) - Client libraries for LiveKit
3-FastAPI - Python web framework for the token server
4-Google Gemini Live API (Realtime Model) - Voice AI model
5-FAISS - Vector database for similarity search
6-React - Frontend UI framework
7-Vite - Frontend build tool and dev server

if you want to test it, download the project, open it using any python support editor (i was using vscode),
create a virtual enviroment (better isolation) and download the requirments.txt file in your terminal
download nodejs to be able run the frontend after you satisfye all requirements for all imports then:

1-open three treminals go to the voice_agent folder then paste this command uvicorn token_server:app --reload --port 8000 to start server

2-in the second terminal go to the voice_agent folder then type python agent.py start (if you are using venv)

3-in the third terminal go to the frontend folder then type npm run dev and click on the local host that will appear and enjoy.

RAG Integration with Gemini Live API

Tool-Based RAG Integration

Instead of injecting retrieved documents directly into every prompt, this system integrates RAG using Gemini tool calling, enabling on-demand retrieval.
This tool exposes the HR knowledge base (Faiss) to Gemini.
Gemini decides when retrieval is needed, based on the user's spoken question.

Retrieval Pipeline

When Gemini determines that a question requires an HR answer:
The spoken query is transcribed by Gemini Live.
Gemini triggers the query_hr_policies tool.

The tool:

Embeds the query using HuggingFace embeddings
Retrieves top-30 candidate chunks from FAISS
Reranks results using a CrossEncoder
Returns the top-k most relevant document chunks
docs = retrieve_top_k(question, k=3)

Prompt Injection

The retrieved document chunks are merged into a single contextual string.
This context is returned to Gemini as the tool's output.
The final response is generated only using retrieved content, and if no answer is found in the Database Gemeni responds with i don't know sir.

Architecture Explanation

How LiveKit connects to Gemini Live API and RAG
This project is a real-time voice assistant that combines LiveKit for audio streaming, Google Gemini Live API for understanding and speaking, and a RAG system to give fact-based answers from HR documents(the example used for testing).

The system has four main parts:

1-Frontend (React + LiveKit Client)
The web interface connects to a LiveKit room using a short-lived token from the FastAPI server.
It sends the user's microphone audio to the room and plays back the assistant's voice responses.

2-Token Server (FastAPI)
Creates temporary tokens that let users join the room, speak, and listen.

3-Voice Agent Worker (LiveKit + Gemini Live)
Listens to the LiveKit room for incoming audio.
Streams the user's voice to Gemini Live for speech-to-text and response generation.
Sends back spoken answers directly into the room as audio.

3-RAG Knowledge Base (FAISS + HuggingFace Embeddings)
Stores documents in a vector database.
When the user asks a question, the assistant retrieves the most relevant info to give accurate answers.

Summary:
The user speaks into the microphone on the frontend, which sends the audio to Gemini Live via a LiveKit room using a token from the FastAPI server. Gemini transcribes and understands the question, then calls the RAG tool to perform a semantic search over the FAISS vector database built from uploaded documents. The retrieved information is used by Gemini to generate an answer, which is streamed back through LiveKit to the user in real time.