🎓 Voice FAQ Chatbot – Case Study Report

By

Ganesha moorthy k (ITvedant)

An in-depth explanation of the architecture, libraries, and workflow for voice-enabled Streamlit chatbot.

🔧 Project Overview

This app is a Streamlit-based voice-enabled chatbot that:
- Accepts voice or text input.
- Transcribes voice to text (STT).
- Retrieves an answer from a PDF-based FAQ using semantic search.
- Speaks the answer aloud (TTS).
- Maintains a transcript of the conversation.

🗂 Core Python Files

1. app.py – main Streamlit app
2. answer_engine.py – gets answers from FAQ
3. faq_processor.py – loads PDF and performs question matching
4. speech_utils.py – handles speech-to-text and text-to-speech
5. requirements.txt – all dependencies listed

📦 Libraries & Purpose

Library	Why It's Used
streamlit	To build the web app interface
streamlit-webrtc	For real-time audio capture using WebRTC
numpy	For audio array manipulation
soundfile	To save NumPy audio to `.wav` for STT
speechrecognition	To convert speech (audio) to text using Google
pyttsx3	Offline TTS engine (speaks text)
gtts	Google online TTS (returns MP3/base64)
pydub	To play audio files inside Python
PyMuPDF	To read and extract text from PDF FAQ
sentence-transformers	Semantic embedding for advanced Q&A

	(future-proof)
scikit-learn	For TfidfVectorizer and cosine similarity

🗀 Module Breakdown

🔷 app.py – Main Interface Logic

Responsibilities:
- Microphone input via streamlit-webrtc
- Voice-to-text transcription
- Calls ask_ai() with query (voice or text)
- Plays both user's question and AI's answer aloud
- Displays chat + transcript

Key Functions:
- AudioProcessor: Collects raw audio frames into a NumPy array
- ask_ai(): The central function that
  - Speaks user's input
  - Queries the AnswerEngine
  - Speaks the answer
  - Updates UI and transcript
- st.chat_input: Accepts fallback typed input

🔷 answer_engine.py – Query Router

Responsibilities:
- Initializes a FAQProcessor
- Provides the best matched answer to a question
- Handles exceptions and fallback responses

🔷 faq_processor.py – PDF-Based QA Model

Responsibilities:
- Extracts Q&A pairs from a PDF
- Uses TF-IDF vectorization + cosine similarity to find the closest match
- Returns matched answer if similarity is above a threshold

Why TF-IDF?
Efficient and fast for short text similarity tasks like FAQs. Ideal for lightweight models.

🔷 speech_utils.py – Voice Tools

Functions:

- tts_play(text): Uses pyttsx3 to speak text directly (offline)
- stt_transcribe_numpy(audio, rate): Saves NumPy audio → WAV → Google STT → text

- tts_play_to_bytes(text): Uses gTTS to generate MP3 in memory as base64

🔄 Workflow Overview

User → 🎤 Speak or 💬 Type
 → [STT via Google if voice]
 → Question → AnswerEngine
 → FAQProcessor → PDF Search (TF-IDF)
 → Answer → Speak (TTS)
 → Show in UI + Transcript

🎯 Notable Features

- ✅ Offline voice output using pyttsx3
- ✅ Online speech recognition using Google STT API
- ✅ Fallback text input
- ✅ PDF-driven FAQ model with semantic text matching
- ✅ Voice output for both user and bot
- ✅ Transcript logging
- ✅ WebRTC-based mic capture

🔒 Considerations for Deployment

Problem	Solution
Streamlit Cloud does not support full WebRTC	Use fallback via st_audiorec or deploy on Render/EC2
STT depends on internet (Google)	Can add Whisper offline STT later
TTS sometimes slow with gTTS	Use offline TTS (pyttsx3) for instant feedback

✅ Summary

Your app is a complete voice-enabled AI chatbot, combining:
- Real-time mic input (via WebRTC)
- STT using Google
- TTS via both pyttsx3 and gTTS
- PDF-based Q&A search via TF-IDF

- Streamlit UI with typed fallback and chat history

Link: https://voice-ai-chat-box-dbjh7yrn98yvut6pl2dv7p.streamlit.app/