Overview / Introduction

🔹 *Title:*

VOCAHIRE – Real-Time Voice-Based Interview Coach using Agentic AI

🔹 *Objective:*

VOCAHIRE is an intelligent, real-time **voice-based interview simulator** designed to prepare users for technical and behavioral interviews. It uses **voice input**, **LLM-based question generation**, **tone analysis**, and **feedback systems** to coach candidates interactively.

🔹 *Problem Statement:*

Many candidates struggle with:

- Public speaking or hesitation during interviews.
- Lack of real-time feedback.
- Limited access to personalized mock interviewers.

There is a need for an **automated, personalized interview coach** that can:

- Ask questions based on resumes.
- Provide spoken and textual feedback.
- Improve candidate confidence using tone/hesitation analysis.

🔹 *Proposed Solution:*

- VOCAHIRE uses **voice input** for interview simulation.
- **LLMs** (like Claude, FLAN-T5, etc.) generate:
  - Resume-based questions
  - HR-style fallback questions
  - Smart feedback
- **Tone and pause detection** is integrated to assess hesitation.
- Results are saved in per-user interview logs for analysis.

🔹 *Target Users:*

- Students preparing for placements/internships.
- Job seekers.
- Career counseling platforms.

🔹 *Key Features:*

- Multi-user login/registration system.
- Resume upload and parsing.
- Voice-to-text (Deepgram / Whisper).
- Interview question generation (Resume + General).
- Real-time tone/hesitation scoring.
- Voice + Text feedback using LLM.
- Logs stored per user.

High-Level Architecture

Here's a **clear architecture overview** of your VOCAHIRE system, showing how all the components interact in real-time:

🔹 *User Interface (Terminal)*

- User registers or logs in
- Uploads resume
- Answers interview questions via mic
- Receives spoken feedback

🔹 *2. Resume Processing Module*

- Reads uploaded `Resume.pdf`
- Uses **PyMuPDF (fitz)** to extract text
- Parses and categorizes into:
  - 📘 Education
  - 💼 Projects
  - 🧠 Skills
- Stores structured data as: `parsed_resume.json`

🔹 *3. Real-time Interview Flow Engine*

- Drives the Q&A session dynamically
- Tracks current question, stores all interactions
- Pulls questions from:
  - Resume-based LLM prompts (Claude 3)
  - HR fallback prompts (Claude)
- Maintains memory and history during interview

🔹 *4. Question Generator (LLM Agent)*

- **Claude 3 Haiku** via **OpenRouter**
  - Resume-based technical questions
  - General HR-style fallback questions
- Prompts are customized with parsed resume info
- Also used for giving detailed **feedback**

🔹 *Voice Engine*

Speech-to-Text:

- User speaks using microphone
- Processed via:
  - **Deepgram API** (primary)
  - or **OpenAI Whisper** (offline)

Tone Detection:

- VAD (Voice Activity Detection)
- Hesitation score computed with silence gaps

Text-to-Speech:

- AI reads each question + feedback out loud using `pyttsx3`

🔹 *Logging & Memory Storage*

- Each user has their own folder: `users/{username}/`
- Inside:
  - `Resume.pdf`

    - `parsed_resume.json`
    - `logs/interview_timestamp.json`
- Logs include:
    - Question
    - Answer
    - Hesitation score
    - LLM feedback

🔹 *Technologies Used*

Layer	Tools / APIs
LLM	Claude 3 (OpenRouter), Flan-T5
STT	Deepgram, Whisper
Resume Parsing	PyMuPDF
Voice	WebRTC VAD, pyttsx3
Backend	Python (Modular Files)
Memory	JSON + Folder Storage

Component-Level Architecture

VocaHire is designed with a modular, agent-based architecture that promotes clarity, scalability, and easy debugging. Each component in the system is responsible for a specific function and communicates through well-defined interfaces.

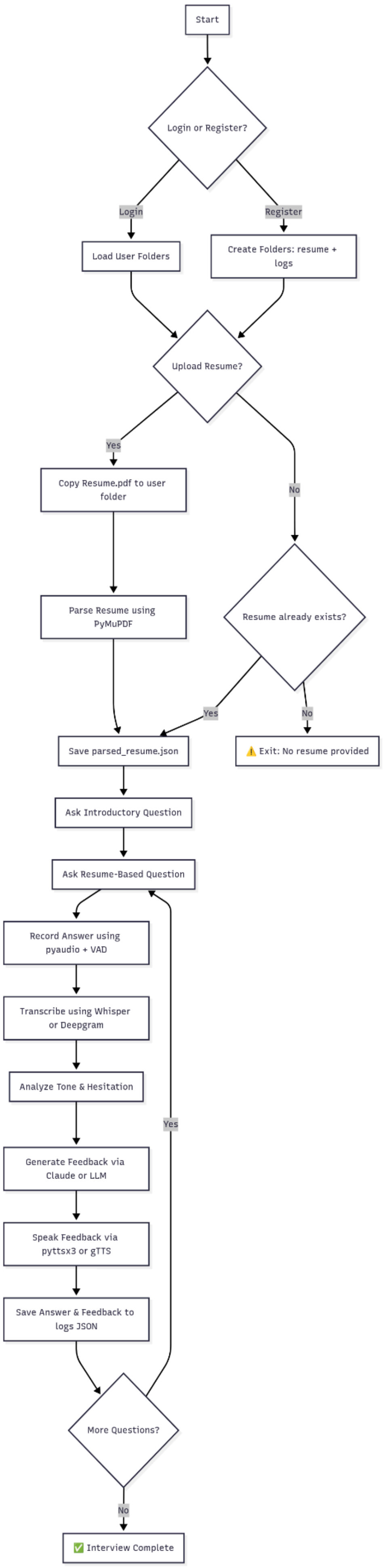
Core Components of VocaHire

Component	Role & Responsibility
1. User Manager	Manages login and registration. Creates per-user folders like `/users/{username}/resume` and `/users/{username}/logs`.
2. Resume Parser	Extracts structured data (education, skills, projects) from uploaded `Resume.pdf` using `PyMuPDF`. Saves output as `parsed_resume.json`.

3. Question Generator	Generates dynamic interview questions from three sources: • `resume_qa_generator.py` → Technical questions from resume • `general_qa_generator.py` → Introductory HR questions • `fallback_qa_generator.py` → Generic HR/behavioral questions
4. Interview Agent	Orchestrates the full interview flow: asking questions, collecting responses, tracking current state, and managing transitions.
5. Audio Input Handler	Records user's spoken response using `webrtcvad`. Applies voice activity detection for intelligent start/stop.
6. STT (Speech-to-Text)	Converts recorded `.wav` files to text using: • Local `Whisper` model or • `Deepgram API`
7. Tone Analyzer	Calculates hesitation/confidence score based on pauses and silence in the audio using a custom scoring algorithm.
8. Feedback Engine	Uses a Large Language Model (Claude 3 via OpenRouter) to evaluate answers and return helpful, structured feedback.
9. Logger	Stores the full interview session in a structured JSON format under `/users/{username}/logs/`, including question, answer, tone score, and feedback.

Interview Execution Pipeline

1. **User login/registration** creates or accesses personal workspace.
2. **Resume uploaded** and parsed into structured JSON.
3. **Interview starts**:
   - One **general intro** question first
   - Followed by **resume-aware** technical questions
   - Ends with **fallback HR/behavioral** questions
4. **Voice is recorded** and saved as `.wav`.
5. **STT engine transcribes** audio to text.
6. **Tone analysis computes** hesitation score.
7. **LLM generates feedback**.
8. **Session is saved** into a personal `.json` log file.


Start
Login or Register?
Login
Register
Load User Folders
Create Folders: resume + logs
Upload Resume?
Yes
No
Copy Resume.pdf to user folder
Resume already exists?
Parse Resume using PyMuPDF
Yes
No
Save parsed_resume.json
⚠️ Exit: No resume provided
Ask Introductory Question
Ask Resume-Based Question
Record Answer using pyaudio + VAD
Transcribe using Whisper or Deepgram
Analyze Tone & Hesitation
Yes
Generate Feedback via Claude or LLM
Speak Feedback via pyttsx3 or gTTS
Save Answer & Feedback to logs JSON
More Questions?
No
✅ Interview Complete

Component-Wise Explanation (Modular Design)

---

1. Login & User Manager

- **File:** `login.py` / main script
- **Purpose:** Handles user registration, login, folder creation (resume, logs)
- **Input:** Username
- **Output:** Creates `users/<username>/resume`, `users/<username>/logs`
- **Special Handling:** Prevents duplicates, prompts for resume upload

---

2. Resume Parser

- **File:** `resume/resume_parser.py`
- **Library:** `PyMuPDF (fitz)`
- **Purpose:** Extracts `education`, `skills`, `projects` from Resume.pdf
- **Output:** `parsed_resume.json` saved to user folder
- **Used In:** `resume_qa_generator.py`

---

3. Resume QA Generator

- **File:** `agents/resume_qa_generator.py`
- **Model Used:** Claude via OpenRouter
- **Purpose:** Generates technical interview questions from parsed resume
- **Input:** `parsed_resume.json`
- **Output:** 1 resume-based question per turn (customizable)

---

4. General QA Generator

- **File:** `agents/general_qa_generator.py`
- **Model:** Claude via OpenRouter
- **Purpose:** Generates intro/HR-style fallback questions like “Tell me about yourself”, “What are your strengths?”
- **Flow Control:** Comes **before** resume questions

5. Voice Capture & VAD

- **File:** `audio/step2_vad_listener.py`
- **Library:** `webrtcvad`, `pyaudio`, `numpy`
- **Purpose:** Records only voice segments using VAD
- **Output:** `.wav` audio saved in user folder

---

6. Speech-to-Text

- **File:** `audio/stt_whisper_local.py` or `deepgram_transcriber.py`
- **Backends Supported:** Whisper (local) / Deepgram (API)
- **Purpose:** Converts voice to text for answering questions

---

7. Tone & Hesitation Analyzer

- **File:** `audio/tone_analysis.py`
- **Purpose:** Computes hesitation % using silence gaps
- **Logic:** High hesitation → Suggests improvement via feedback

---

8. Feedback Engine

- **File:** `agents/feedback_engine.py`
- **LLM:** Claude (OpenRouter)
- **Purpose:** Generates helpful, kind, actionable feedback
- **Input:** question + answer + hesitation_score
- **Output:** clear feedback string

---

9. TTS Speaker

- **File:** `audio/tts_speaker.py`
- **Engines Supported:** `pyttsx3`, `gTTS`
- **Purpose:** Speaks both questions and feedback aloud
- **Customization:** Can set gender/voice rate

---

10. Interview Orchestration

- **File:** `audio/step4_voice_to_langgraph.py`
- **Acts As:** Main loop controller
- **Responsibilities:**
  - Asks questions in order (intro → resume → fallback)
  - Captures audio
  - Calls STT, analysis, feedback, and logging

---

11. Logging & Memory

- **Output:** `.json` logs stored in `users/<username>/logs/`
- **Fields Stored:** question, answer, hesitation %, feedback
- **Future Plan:** Can be queried by LangChain for memory-based interviews

Agent-Oriented Architecture (LangGraph-style)

Why Agent-Based?

Your system follows a **modular agent pattern**, where each module (question generation, STT, TTS, feedback, etc.) is an autonomous "agent" handling a single responsibility. This makes it easier to integrate with LangGraph in future or simulate its behavior manually.

Agents :

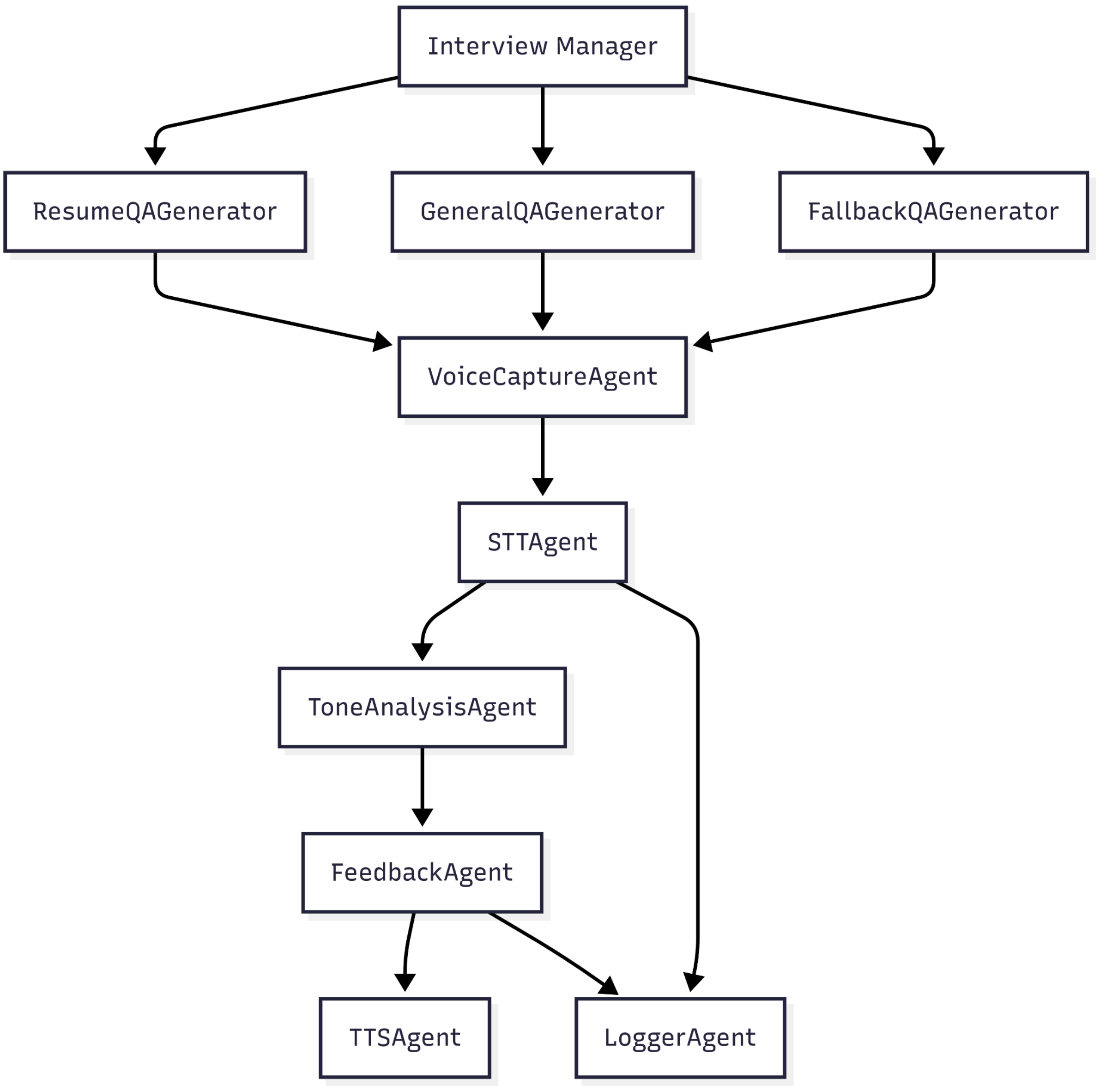
Agent Name	Responsibility	Input	Output
`ResumeParserAgent`	Extract resume data (skills, projects, etc.)	PDF resume	`parsed_resume.json`
`ResumeQAGenerator`	Ask technical questions based on resume	Parsed resume	Resume-based question

`VoiceCapture Agent`	Record user audio using VAD	Microphone input	`.wav` file
`STTAgent`	Convert voice to text	Audio file	Answer text
`ToneAnalysis Agent`	Analyze pauses & compute hesitation score	Audio file	Hesitation score (%)
`Feedback Agent`	Generate feedback based on response tone & content	Answer + hesitation	Feedback string
`TTSAgent`	Speak question and feedback to the user	Text	Spoken audio output
`Logger Agent`	Store question, answer, score, and feedback	All outputs above	JSON log entry
`Interview Manager`	Coordinates flow: question → record → feedback	State tracker	Runs all agents in correct sequence

Future LangGraph Upgrade

You can replace `InterviewManager` with a real LangGraph flow using `langgraph.graph` and assign:

- `Stateful Mode` for resume memory
- `ConditionalEdge` to route based on score
- LangChain tools for each agent



Memory Strategy in VOCAHIRE

Memory Type	Method Used	Format	Where Stored
Interview Session	In-memory `InterviewState`	Python class	Tracks question index, state
Answers & Feedback	JSON file logs	`.json`	Stored per-user in `users/{name}/logs/`
Resume Parsing	Cached resume data	`.json`	`users/{name}/resume/parsed_resume.json`

Audio Files	Raw `.wav` recordings	`.wav`	`users/{name}/recordings/`
Multi-user Support	Folder structure	Folders	`users/{username}/...`

Core Components for Flow:

File	Purpose
`interview_agent.py`	Controls question flow
`resume_qa_generator.py`	Generates technical/resume questions
`fallback_qa_generator.py`	Generates HR-style questions
`submit_answer()`	Stores question-answer-feedback
`get_next_question()`	Selects next question logic

Components Used:

Type	Library	Description
Voice Activity Detection (VAD)	`webrtcvad`	Detects when the user starts/stops speaking
Audio Recording	`pyaudio`, `soundfile`	Captures voice and saves it as `.wav`
Speech-to-Text (STT)	`Deepgram API`	Transcribes spoken answers to text
Text-to-Speech (TTS)	`pyttsx3`	Speaks both questions and feedback

Files Involved:

File	Role
`stt_whisper_local.py` *(or Deepgram)*	Converts `.wav` to text
`step2_vad_listener.py`	Records voice based on VAD
`tone_analysis.py`	Calculates hesitation/pauses

`tts_speaker.py`	Converts text to speech using `pyttsx3`

Sample Voice Flow:

```
🧠 AI: Tell me about yourself.
🔊 Speaking: Tell me about yourself.
🎙️ Speak now...
🟢 Start Recording
🔴 Stop Recording
💾 Saved: recordings/recording_xxx.wav
🧠 Transcribing...
📝 Your Answer: I'm a student at IIT Patna...
🎯 Hesitation Score: 21.7%
```

Summary

VOCAHIRE is an AI-powered **real-time voice interview coach** designed to help users practice technical and behavioral interview questions in an interactive and personalized way. The system uses a combination of **local and API-based speech recognition, tone analysis, LLM-driven question generation, and smart feedback engines** to simulate the real interview experience.

As the developer, my goal was to create an intelligent, multi-user platform where candidates can:

- Speak naturally and be transcribed accurately
- Receive questions based on their **resume and HR scenarios**
- Get **real-time feedback** on tone, hesitation, and answer quality
- Maintain a record of their interviews and performance

This solution combines modular agents, natural language models (Claude via OpenRouter), resume parsing logic, voice activity detection, and a personalized feedback loop to deliver a full-stack interview simulation experience. The project was developed using **Python 3.11+, Deepgram/OpenAI Whisper, PyMuPDF, WebRTC VAD, and pyttsx3**, with flexible JSON storage and a folder-based multi-user system.

Key Achievements:

- Real-time voice conversation with VAD (webrtcvad)

- Accurate transcription using Whisper / Deepgram
- Resume parsing & dynamic question generation
- HR fallback & tone-aware feedback via Claude
- Multi-user support with session logging

Future Scope:

- Cloud-based deployment (e.g., using Streamlit, FastAPI)
- Interview dashboard with analytics
- Integration with ATS platforms or EdTech products
- Support for group or panel interviews

Conclusion

VOCAHIRE is a modular, AI-driven voice interview platform designed to simulate real-time interview experiences. Through the integration of advanced speech-to-text (STT), large language models (LLMs), and emotion-aware feedback systems, it provides users with personalized coaching and actionable feedback.

By supporting resume parsing, memory, and multi-user capabilities, VOCAHIRE mimics the structure and flow of real interviews — making it a powerful tool for students and professionals preparing for technical roles.

This system is extensible, scalable, and can easily integrate future upgrades like better UI, database-backed memory, or multimodal feedback systems.