AI Interviewer Assignment

For the Generative AI Engineer Role

---

1. Overview

You are tasked with creating a **full-stack application** that simulates a voice-based AI interviewer. The interviewer should use **Text-to-Speech (TTS)** to ask questions, and **Speech-to-Text (STT)** to transcribe the candidate's responses. An **LLM** (e.g., GPT-4) will provide context-driven follow-up questions and ultimately rate the candidate.

---

2. Key Requirements

1. **Real-Time Video Meeting (LiveKit)**

- Integrate **LiveKit** (or an equivalent) for a live video/audio session.
- The candidate joins the session through a web interface.
- The AI Interviewer's video feed is optional, but **TTS audio** output is mandatory.

2. **Speech-to-Text (STT) & Text-to-Speech (TTS)**

- Use **Deepgram** or another STT/TTS provider to:

1. **Transcribe** candidate responses.
2. **Speak** the AI Interviewer's questions out loud.

3. **Language Model Integration**

- Incorporate **GPT-4** (or another LLM) to:
- Generate interview questions based on the candidate's responses.
- Provide a **final rating (1–10)** and a **verdict** on candidate suitability.
- Ensure the **system prompt** can be tailored to different interviewer "personalities."

4. **Local-Only Storage**

- **No external database** is required.
- You may store data in memory or a simple local file (e.g., JSON).

- Keep the CV, Job Description, system prompt, transcripts, and results locally.

5. **Frontend & Backend**

- **Frontend** (Admin + Candidate views):
- **Admin**: Upload or paste the CV, Job Description, and system prompt.
- **Candidate**: Joins the LiveKit session, speaks, and hears questions via TTS.
- **Backend**:
- Orchestrates calls to GPT-4, STT/TTS, and LiveKit.
- Stores interview data and final results.

6. **End-to-End Flow**

- **Admin** configures the interview by providing the candidate's details (CV, JD, system prompt).
- **Candidate** joins the interview URL.
- AI Interviewer greets them with **TTS**, listens via **STT**, and uses **GPT-4** to decide subsequent questions.
- At the end, the AI Interviewer generates a **rating** and **verdict**, stored locally.

---

3. Deliverables

1. **Source Code**

- A Git repository (or ZIP) with your entire solution.
- Both **frontend** (admin and candidate views) and **backend** (orchestration logic) should be included.

2. **README**

- **Setup Instructions**: How to install dependencies, run the application locally, and set environment variables (e.g., OPENAI_API_KEY, DEEPGRAM_API_KEY).
- **Usage**: Step-by-step guide to simulate an interview (admin side + candidate side).
- **Notes**: Any additional comments, known issues, or limitations.

3. **Brief Demo/Explanation**

- If possible, a short video or written walkthrough demonstrating a sample interview session (optional but appreciated).

---

4. Evaluation Criteria

1. **Code Quality**

- Readable, well-structured code.
- Proper separation of concerns and use of clear naming conventions.

2. **Implementation Accuracy**

- Correct usage of STT/TTS and GPT-4.
- Minimal dependencies beyond what is necessary.

3. **User Experience**

- Candidates can easily join the LiveKit room, speak, and hear TTS responses.
- Admin interface is straightforward for uploading CV, JD, prompt, and viewing results.

4. **Completeness**

- The system should produce a **final rating** and short **verdict**.
- All data (CV, JD, transcripts, results) should be stored and retrievable locally.

5. **Documentation**

- A clear README that allows others to run and test your solution without confusion.

---

5. Submission

- **Submission**:
- Send us a link to your Git repository or a ZIP archive.
- Include the README file at the root level.
- **Questions**: If you have any clarifications, email **akash@growhut.in**