Real-Time Conversational Voice Agent Prototype

A full-stack, real-time voice agent that demonstrates a complete Speech-to-Speech (S2S) conversational pipeline. This prototype uses a third-party Automatic Speech Recognition (ASR) service (e.g., Deepgram or AssemblyAI) to transcribe user input and Murf Text-to-Speech (TTS) to generate spoken responses.

1. Project Overview

Core Features

Component	Technology	Requirement
ASR (Speech-to-Text)	Deepgram / AssemblyAI	Must enable conversational interaction. Transcribes user's spoken audio into text in real-time.
NLP (Language Processing)	*(To be defined: e.g., LLM/GPT API)*	Processes the transcribed text and generates a natural language text response.
TTS (Text-to-Speech)	Murf AI	Generates high-quality speech output from the LLM response in real-time.
Conversational Flow	Custom Backend (Node.js/Python)	The user must be able to speak to the agent and receive a spoken response seamlessly.
Security	Environment Variables	All sensitive API keys are handled securely outside the codebase.

Architecture

The voice stream flows through the following pipeline:

1. **User Speech** $\rightarrow$ Microphone captures audio input.
2. **ASR Service** $\rightarrow$ Real-time conversion of audio to text.
3. **Backend/LLM** $\rightarrow$ Processes text, determines intent, and generates a text response.
4. **TTS Service** $\rightarrow$ Generates audio (PCM/WAV data) from the text response (using Murf).
5. **Agent Speech** $\rightarrow$ Audio is streamed back to the user's speakers/headphones in real-time.

2. Setup and Installation

These instructions assume a standard Node.js or Python environment is used for the backend server that orchestrates the API calls.

Prerequisites

- Node.js (v18+) or Python (3.10+)
- A code editor (VS Code recommended)
- Account credentials for:
  - **ASR Provider** (e.g., Deepgram, AssemblyAI)
  - **TTS Provider** (Murf AI - leveraging the 1,000,000 free characters)
  - **LLM Provider** (e.g., Gemini API, OpenAI, or a custom LLM)

Installation Steps

1. **Clone the Repository (or create your project folder):**
   ```
   git clone [your-repo-url]
   cd [your-project-name]
   ```

2. **Install Dependencies:**
   - **Node.js:** npm install (or yarn install)
   - **Python:** pip install -r requirements.txt (or install specific packages like dotenv, websocket-client, etc.)
3. Configure Environment Variables (.env):
   This is the crucial step for securely handling your API keys.
   - Create a file named .env in the root of your project directory.
   - **DO NOT** commit this file to Git. Ensure .env is listed in your .gitignore.
   - Populate the file with your credentials:
     ```
     # --- ASR API Key (Example using Deepgram) ---
     ASR_API_KEY="dg_xxxxxxxxxxxxxxxxxxxxxxxxxxx"
     ASR_BASE_URL="wss://[api.deepgram.com/v1/listen](https://api.deepgram.com/v1/listen)"

     # --- Murf TTS API Key ---
     MURF_API_KEY="murf_xxxxxxxxxxxxxxxxxxxxxxxxx"
     # Optional: Murf Voice ID to ensure consistent output
     MURF_VOICE_ID="Voice-ID-of-your-choice"

     # --- LLM/Conversational Engine API Key (Example) ---
     LLM_API_KEY="ai_xxxxxxxxxxxxxxxxxxxxxxxxxx"
     ```

4. **Run the Prototype:**
   - **Node.js:** npm start (or your defined start script)
   - **Python:** python server.py (or your main entry file)

3. API Details and Authentication

The prototype relies on three distinct external APIs:

A. Automatic Speech Recognition (ASR)

- **Endpoint:** Typically a WebSocket connection for real-time, streaming audio.
  - *Example (Deepgram):* wss://api.deepgram.com/v1/listen
- **Authentication:** API Key passed either in a request header or as a WebSocket subprotocol/query parameter.
- **Key Requirement:** Must support real-time audio streaming for low-latency conversational interaction.

B. LLM / Conversational Engine

- **Endpoint:** Standard REST API, e.g., for generating text content.
- **Input:** Text transcript from the ASR service.
- **Output:** Text response for the user.

C. Murf Text-to-Speech (TTS)

- **Service Requirement:** Utilize the free character limit for new accounts.
- **Function:** Accepts the LLM's text response and returns audio data (e.g., PCM, WAV, or MP3 bytes).
- **Real-Time Output:** The implementation must stream the audio back to the user as soon as the first chunks are received, rather than waiting for the entire audio file to be generated. This is critical for achieving a low-latency, "real-time" spoken response.
- **Authentication:** Requires the MURF_API_KEY for secure access.

4. Security Notes

- **Environment Variables:** The use of a .env file ensures that API keys are never hardcoded into the source code (.js, .py, etc.).
- **.gitignore:** The .gitignore file **must** include the line .env to prevent accidental public disclosure of keys via version control.
- **Server-Side Access:** API keys should **only** be used on the server-side component of the application. Client-side (browser) code should proxy all API requests through the secure backend server.