# **EchoAI Documentation**

**EchoAI** is a full-stack web application that processes customer feedback by analyzing its sentiment and generating an appropriate response. The project leverages Azure AI Services for sentiment analysis and text-to-speech (TTS) conversion, and uses OpenAI’s GPT model for creating professional, natural language responses. The application is built using a Flask backend and a Streamlit frontend.

## **Overview**

**EchoAI** is designed to provide an intelligent, responsive, and interactive experience for analyzing user feedback. Given a piece of text feedback, the application:

* Uses Azure Text Analytics to determine if the feedback is positive, neutral, or negative, along with corresponding confidence scores.
* Uses OpenAI’s GPT (e.g., gpt-3.5-turbo) to generate a professional and contextual response tailored to the sentiment of the feedback.
* Converts the AI-generated text response into speech using Azure Speech Services, with voice characteristics selected according to the detected sentiment.
* Provides a user-friendly interface built with Streamlit that visualizes sentiment distributions and allows users to play the synthesized speech response.

## **Technologies Used**

1. **Backend:** Flask, Python
2. **Frontend:** Streamlit
3. **APIs and Services:**
   1. **Azure AI Services:**
      1. Text Analytics for sentiment analysis
      2. Speech Services for text-to-speech synthesis
   2. **OpenAI API:** For generating AI-based responses
4. **Environment Management:** python-dotenv
5. **Other Libraries:** Requests, Logging, and various visualization and utility libraries.

## **Architecture and Components**

The application comprises two main components:

1. **Flask Backend API:**
   1. **Endpoints:**
      1. analyze\_sentiment (POST): Accepts feedback, performs sentiment analysis, generates an AI response, and converts it to speech.
      2. get\_audio (GET): Serves the generated audio file.
   2. **Integrations:**
      1. **Azure AI Services:** For both text analytics and speech synthesis.
      2. **OpenAI:** For generating natural language responses.
2. **Streamlit Frontend:**
   1. **User Interface:**
      1. Accepts user input via a text area.
      2. Displays sentiment results and AI responses.
      3. Visualizes confidence scores with an interactive pie chart.
      4. Provides functionality to play the synthesized audio response.
   2. **HTTP Requests:**
      1. Connects to the Flask backend to submit feedback and retrieve results.

## **Project Structure**

EchoAI/

├ app.py # Flask backend API

├ app\_frontend.py # Streamlit frontend

├azure\_sentiment.py # Azure sentiment analysis

├azure\_tts.py # Azure text-to-speech with emotion

├ openai\_response.py # OpenAI GPT responses

├config.py # Environment variables & API creds

├requirement.txt # Project dependencies

├README.md # Project documentation

## **API Endpoints**

#### **analyze\_sentiment (POST)**

**Purpose:**This endpoint is designed to analyze the sentiment of user-provided feedback, generate an AI-based response, and synthesize the response into speech. It leverages Azure AI Services for sentiment analysis and text-to-speech conversion, as well as the OpenAI API for generating natural language responses.

**Response:**Upon successful processing, the endpoint returns a JSON response that includes:

* The overall sentiment (e.g., "positive", "neutral", or "negative"),
* Detailed confidence scores for each sentiment category, and
* The AI-generated response tailored to the analyzed sentiment.

#### **get\_audio (GET)**

**Purpose:**This endpoint is responsible for serving the audio file (typically named response.wav) that is generated by the Azure Speech Service. The file contains the speech-synthesized version of the AI response.

**Response:**

* If the audio file is available, the endpoint returns the file in the appropriate audio format.

## **Scripts**

#### **app.py**

**Role:**The app.py module hosts the Flask backend API. It defines the endpoints required for processing user feedback, including sentiment analysis and audio file delivery.

**Key Functions:**

* **analyze\_sentiment:**Manages the complete processing workflow by receiving user feedback, performing sentiment analysis, generating an AI response, and triggering text-to-speech conversion.
* **get\_audio:**Facilitates the delivery of the generated audio file to the frontend.

#### **app\_frontend.py**

**Role:**The app\_frontend.py module implements the Streamlit-based user interface. It serves as the interactive layer where users input their feedback and view the corresponding analysis and response.

**Key Features:**

* Provides an input text area for user feedback.
* Submits the feedback to the backend and retrieves sentiment analysis and AI-generated responses.
* Visualizes the sentiment confidence scores using an interactive Plotly pie chart.
* Offers functionality to play the synthesized audio response.

#### **azure\_sentiment.py**

**Role:**The azure\_sentiment.py module interfaces with the Azure Text Analytics API to analyze the sentiment of the provided feedback.

**Key Function:**

* **analyze\_sentiment (feedback):**Processes the input text to determine the overall sentiment and returns associated confidence scores.

#### **azure\_tts.py**

**Role:**This module handles the conversion of text to speech using the Azure Speech Service. It incorporates emotion-based voice selection to enhance the responsiveness of the output.

**Key Functions:**

* **text\_to\_speech (text, sentiment, filename="response.wav"):**Synthesizes speech from the provided text and saves it as an audio file.
* **select\_voice (sentiment):**Determines the appropriate voice for speech synthesis based on the sentiment of the feedback.
* **generate\_ssml (text, sentiment):**Constructs the SSML (Speech Synthesis Markup Language) required for producing emotion-infused speech output.

#### **openai\_response.py**

**Role:**The openai\_response.py module is responsible for generating an empathetic and concise AI response based on the analyzed sentiment and the original feedback.

**Key Function:**

* **get\_ai\_response (sentiment, feedback):**Constructs a prompt using the provided feedback and sentiment, and interacts with the OpenAI API to produce a tailored response.

#### **config.py**

**Role:**The config.py module is tasked with loading environment variables and setting the necessary API credentials for both Azure and OpenAI services.

**Key Function:**

* Utilizes the python-dotenv library to manage configuration from a .env file.
* Logs detailed error messages if any required API keys or endpoints are missing, ensuring proper configuration for the application.