DOCUMENTACION **VIRTUAL_ASSISTANT**

El Proyecto es un asistente virtual, puesto en una pagina web local, hace uso de tres APIS: el TTS (Eleven labs), OPENAI y MediaRecorder (en el JS)

Para esto tuve que hacer un pequeño estudio sobre que modelos de OPENAI es mejor para lo que se iba a emplear, usé dos modelos **Whisper** y **GPT-4o mini**

https://platform.openai.com/docs/models/whisper-1

https://platform.openai.com/docs/models/gpt-4o-mini

Documentación

JAVASCRIPT:

La clase AudioRecorder graba el audio desde el micrófono usando el api de MediaRecorder, esta esta conectada al DOM y maneja eventos y muestra resultados,

Variables del Constructor:

This.estado=0 usa 0 como inactivo,1 grabando, 2 procesando.

this.blobs =[] este almacena fragmentos (del audio)

this.stream, es el stream del audio el cual sale por getUserMedia

this.rec es la capturacion del audio.

This.mic/box es la selección de eso mismo dentro de mi css

outputSelector ID elemento del HTML es donde se mostrará la transcripción y la respuesta

setupListeners()

Espera a que DOM esté 100% cargado y asocia con el clic al botón del micrófono. Al hacer clic, se ejecuta handleMicClick().

async startRecording(, inicia la grabación de audio y...

1. Pide acceso al microfono.
2. Crea un MediaRecorder con el stream
3. Captura los datos disponibles y los almacena en blobs

4. Al detenerse, junta los datos como un Blob de tipo webm y lo envia al server con fetch usando FormData
5. Procesa la respuesta del servidor (espera un JSON con transcripcion, respuesta y opcionalmente audio[la respuesta del TTS])

Muestra el resultado en el campo output, y si hay audio generado por el asistente, lo reproduce.

stopRecording() detiene la grabación

1- Verifica si el MediaRecorder esta en grabando
2- Llama stop() (que pues para)
3- Detiene manualmente todas las pistas del stream de audio.

handleMicClick() si es 0, inicia grabacion y cambia el estado a 1, si es 1, detiene la grabacion muestra el cuadro de respuesta, cambia el estado a 2

Clases:

T_audio:

caudio = "temp_audio.webm" hace referencia al nombre

posteriormente se guarda el contenido del audio, seguido de la llamada de **Whisper**, el cual transcribe el audio dado en texto.

Llama a GPT, y después a el TTS, y se manda al front (la trasncripcion, la respuesta y el TTS)

GPT:

Recibe el texto de T_audio, lo manda al modelo **GPT-4o mini**, te asigna una respuesta, asignándole un rol, y también que responda según el texto mandando

Retorna la variable con el contenido.

TTS:

```
CHUNK_SIZE = 1024
url = https://api.elevenlabs.io/v1/text-to-speech/EXAVITQu4vr4xnSDxMaL
    headers = {
      "Accept": "audio/mpeg",#respuesta de mp3
      "Content-Type": "application/json",#se manda en un formato JSON
```

```
        "xi-api-key": self.key#esto es la autentificacion
    }
```

carga el api, acto seguido, se manda la información (texto), con el modelo multilengual, con algunas configuraciones extras como “estabilidad” y ”ajustar la voz” (para realismo), se carga el audio con la voz del TTS y retorna dicho audio (recordando que es un mp3)

html y complementos:

HTML: El icono, fue sacado de fontawesome

https://fontawesome.com/icons/microphone?f=classic&s=solid

complementos:

Pag_Web_Local:

Agrega dos rutas a propósito la primera renderiza el html y es un get por defecto, la otra se declara como post, y es la encargada de recibir y es la que procesa el audio, se llama desde el front como fetch() (cosa que se mencionó arriba)