```python
import gradio as gr
from openai import OpenAI
from PIL import Image
import requests
from dotenv import load_dotenv
from io import BytesIO
import os
import tempfile
import numpy as np

# Load environment variables
load_dotenv()

# Initialize OpenAI client
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))


# Function to generate an image using DALL-E 3
def generate_edited_image(image_path, prompt):
    try:
        with open(image_path, "rb") as image_file:
            response = client.images.generate(
                model="dall-e-3",
                prompt=prompt,
                size="1024x1024",
                n=1
                )
        print("DALL·E API Full Response:", response) # Print the full API response for debugging
        image_url = response.data[0].url
        return image_url
    except Exception as e:
        print(f"Image generation failed: {e}") # Print to console for debugging
        return None



# Function to transcribe audio using OpenAI Whisper API
def transcribe_audio(audio_file_path):
    try:
        with open(audio_file_path, "rb") as audio_file:
            transcript = client.audio.transcriptions.create(
                model="whisper-1",
                file=audio_file
            )
        return transcript.text

    except Exception as e:
        print(f"Transcription failed: {e}")  # Print to console
        return None


def process_inputs(audio_filepath, image_pil):
    if audio_filepath is None or image_pil is None:
        return "Please upload both audio and image.", None

    if audio_filepath:
        audio_extension = os.path.splitext(audio_filepath)[1]

        with tempfile.NamedTemporaryFile(delete=False, suffix=audio_extension) as temp_audio_file:
            with open(audio_filepath, "rb") as audio_file:
                temp_audio_file.write(audio_file.read())

            temp_audio_path = temp_audio_file.name

        transcription = transcribe_audio(temp_audio_path)

        if transcription:
            prompt = f"{transcription}"

            with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as temp_image_file:
                image_rgba = image_pil.convert("RGBA")  # Convert to RGBA
                image_rgba.save(temp_image_file, format="PNG")
                temp_image_path = temp_image_file.name

            image_url = generate_edited_image(temp_image_path, prompt)
            if image_url:
                try:
                    response = requests.get(image_url, stream=True)
                    response.raise_for_status()
                    image_bytes = BytesIO()
                    for chunk in response.iter_content(chunk_size=8192):
                        image_bytes.write(chunk)
                    image_bytes.seek(0)

                    generated_image = Image.open (image_bytes)
                    generated_image = generated_image.resize((512, 512), Image.LANCZOS)
                    generated_image_np = np.array(generated_image) # Convert to numpy array

                    os.remove(temp_image_path)  # Remove initial image after edit
                    os.remove(temp_audio_path)


                    return "Success!", generated_image_np

                except Exception as e:
                    return f"Error displaying image: {e}", None
            else:
                return "Image edit failed.", None


        os.remove(temp_audio_path)

    return "", None


# Define the Gradio interface
with gr.Blocks() as demo:
    audio_input = gr.Audio(type="filepath", label="Upload Audio")
    image_input = gr.Image(type="pil", label="Upload Initial Image (<4MB)") # Changed to type="pil"
    output_message = gr.Textbox(label="Message") # For displaying messages
    image_output = gr.Image(type= "numpy", label="Generated Image")
    btn = gr.Button("Generate")
    btn.click(
        fn=process_inputs,
        inputs=[audio_input, image_input], # Pass both inputs to the function
        outputs=[output_message, image_output] # Output the message and the image
    )

demo.launch(share=True, server_port=7860)
```