let mediaRecorder;

let audioChunks = [];

let audioContext, analyser, micStream;

let dataArray;

let isRecording = false;

let silenceTimeout;

document.getElementById("startRecording").addEventListener("click", async () => {

if (isRecording) {

stopRecording();

return;

}

resetAudioPlayer();

try {

const stream = await navigator.mediaDevices.getUserMedia({ audio: true });

if (mediaRecorder && mediaRecorder.state !== "inactive") {

mediaRecorder.stop();

}

if (micStream) {

micStream.getTracks().forEach(track => track.stop());

}

if (audioContext && audioContext.state !== "closed") {

audioContext.close();

}

mediaRecorder = new MediaRecorder(stream, { mimeType: "audio/webm" });

audioChunks = [];

micStream = stream;

isRecording = true;

mediaRecorder.ondataavailable = (event) => {

if (event.data.size > 0) {

audioChunks.push(event.data);

}

};

mediaRecorder.onstop = async () => {

isRecording = false;

const audioBlob = new Blob(audioChunks, { type: "audio/webm" });

const formData = new FormData();

formData.append("file", audioBlob, "recording.webm");

addMessage("🎤 Processing your request...", "user-message");

try {

const response = await fetch("/api/VoiceProcessor", {

method: "POST",

body: formData

});

if (response.ok) {

const audioBlob = await response.blob();

const audioUrl = URL.createObjectURL(audioBlob);

addMessage("✅ AI Response Ready. Playing Audio!", "ai-message");

const audioPlayer = document.getElementById("audioPlayer");

audioPlayer.src = audioUrl;

audioPlayer.classList.remove("hidden");

audioPlayer.play();

} else {

addMessage("❌ Error processing request.", "ai-message");

}

} catch (error) {

console.error("Error:", error);

addMessage("❌ Failed to process audio.", "ai-message");

}

};

mediaRecorder.start();

monitorForSpeech();

toggleRecordingUI(true);

} catch (error) {

console.error("Error accessing microphone:", error);

addMessage("❌ Failed to access microphone.", "ai-message");

}

});

document.getElementById("stopRecording").addEventListener("click", () => {

stopRecording();

});

function stopRecording() {

if (mediaRecorder && mediaRecorder.state !== "inactive") {

mediaRecorder.stop();

}

if (micStream) {

micStream.getTracks().forEach(track => track.stop());

}

if (audioContext) {

audioContext.close();

}

isRecording = false;

toggleRecordingUI(false);

}

function monitorForSpeech() {

if (!micStream) return;

audioContext = new AudioContext();

analyser = audioContext.createAnalyser();

const source = audioContext.createMediaStreamSource(micStream);

source.connect(analyser);

analyser.fftSize = 512;

dataArray = new Uint8Array(analyser.fftSize);

function checkForSpeech() {

if (!isRecording) return;

analyser.getByteTimeDomainData(dataArray);

let sum = 0;

for (let i = 0; i < dataArray.length; i++) {

let amplitude = (dataArray[i] - 128) / 128;

sum += amplitude \* amplitude;

}

let volume = Math.sqrt(sum / dataArray.length);

let speechThreshold = 0.02;

if (volume > speechThreshold) {

console.log("Speech detected. Starting silence monitoring...");

monitorSilence();

return;

}

requestAnimationFrame(checkForSpeech);

}

checkForSpeech();

}

function monitorSilence() {

let silenceStart = null;

function checkSilence() {

if (!isRecording) return;

analyser.getByteTimeDomainData(dataArray);

let sum = 0;

for (let i = 0; i < dataArray.length; i++) {

let amplitude = (dataArray[i] - 128) / 128;

sum += amplitude \* amplitude;

}

let volume = Math.sqrt(sum / dataArray.length);

let silenceThreshold = 0.01;

if (volume > silenceThreshold) {

clearTimeout(silenceTimeout);

silenceStart = null;

} else {

if (silenceStart === null) {

silenceStart = performance.now();

}

let silenceDuration = performance.now() - silenceStart;

if (silenceDuration > 1000) {

stopRecording();

return;

}

}

requestAnimationFrame(checkSilence);

}

checkSilence();

}

function resetAudioPlayer() {

const audioPlayer = document.getElementById("audioPlayer");

if (!audioPlayer.paused) {

audioPlayer.pause();

}

audioPlayer.currentTime = 0;

audioPlayer.src = "";

audioPlayer.classList.add("hidden");

}

function toggleRecordingUI(isRecording) {

document.getElementById("startRecording").classList.toggle("hidden", isRecording);

document.getElementById("stopRecording").classList.toggle("hidden", !isRecording);

}

function addMessage(text, className) {

const chatBox = document.getElementById("chatBox");

const message = document.createElement("div");

message.classList.add("message", className);

message.innerText = text;

chatBox.appendChild(message);

chatBox.scrollTop = chatBox.scrollHeight;

}