```
<!DOCTYPE html>
<html lang=”en”>
<head>
 <meta charset=”UTF-8” />
 <meta name=”viewport” content=”width=device-width,initial-scale=1.0” />
 <title>Microphone Rhythm Test</title>
 <style>
  Body {
   Font-family: Arial, sans-serif;
   Background: linear-gradient(135deg, #667eea, #764ba2);
   Min-height: 100vh;
   Display: flex;
   Justify-content: center;
   Align-items: center;
   Margin: 0;
  }
  .container {
   Background: white;
   Padding: 40px;
   Border-radius: 20px;
   Text-align: center;
   Box-shadow: 0 20px 40px rgba(0,0,0,0.1);
   Max-width: 600px;
   Width: 90%;
  }
  H1 { color: #333; margin-bottom: 30px; }
```

```
.mic-circle {
  Width: 150px; height: 150px; border-radius: 50%;
  Background: #667eea; display: flex; align-items: center;
  Justify-content: center; margin: 20px auto; cursor: pointer;
  Transition: all 0.3s ease; font-size: 4em;
}
.mic-circle:hover { transform: scale(1.1); }
.mic-circle.recording { background: #dc3545; animation: pulse 1s infinite; }
@keyframes pulse { 0%,100%{ transform: scale(1);} 50%{ transform: scale(1.2);} }
.btn {
  Background: #667eea; color: white; border: none; padding: 15px 30px;
  Border-radius: 25px; font-size: 16px; cursor: pointer; margin: 10px;
  Transition: all 0.3s ease;
}
.btn:hover { background: #5a67d8; transform: translateY(-2px); }
.btn.stop { background: #dc3545; }
.volume-bar { width: 100%; height: 30px; background: #e0e0e0; border-radius: 15px; margin: 20px 0; overflow: hidden; }
.volume-fill { height: 100%; background: linear-gradient(90deg,#28a745,#ffc107,#dc3545); width: 0%; transition: width 0.1s ease; }
.status { font-size: 18px; margin: 20px 0; padding: 15px; border-radius: 10px; }
.status.info { background: #e3f2fd; color: #1565c0; }
.status.success { background: #e8f5e9; color: #2e7d32; }
.status.error { background: #ffebee; color: #c62828; }
.debug { background: #f5f5f5; padding: 15px; border-radius: 10px; margin: 20px 0; font-family: monospace; text-align: left; max-height: 200px; overflow-y: auto; }
.hidden { display: none; }
```

```
   Canvas { width: 100%; height: 100px; background: #f8f9fa; border-radius: 10px; margin: 20px 0; display: block; }

   .tempo { font-size: 2em; font-weight: bold; color: #28a745; margin: 15px 0; }

   .beats { display:flex; justify-content:center; gap:10px; margin:20px 0; }

   .beat { width:40px; height:40px; border-radius:50%; background:#ccc; display:flex; align-items:center; justify-content:center; font-weight:bold; transition: all 0.3s ease; }

   .beat.active { background:#dc3545; color:white; transform: scale(1.3); }

  </style>

</head>

<body>

  <div class="container">

   <h1>🎤 Microphone Rhythm Test</h1>


   <div class="mic-circle" id="micCircle">🎤</div>


   <div>

    <button class="btn" id="startBtn">Start Microphone</button>

    <button class="btn stop hidden" id="stopBtn">Stop Microphone</button>

   </div>


   <div class="status info" id="status">Click "Start Microphone" to begin</div>


   <div class="volume-bar"><div class="volume-fill" id="volumeFill"></div></div>


   <div class="tempo" id="tempo">0 BPM</div>
```

```
  <div class=”beats” id=”beats”>
    <div class=”beat”>1</div>
    <div class=”beat”>2</div>
    <div class=”beat”>3</div>
    <div class=”beat”>4</div>
  </div>

  <canvas id=”canvas”></canvas>

  <div class=”debug” id=”debug”>Debug info will appear here...</div>
</div>

<script>
  Let audioContext = null;
  Let mediaStream = null;
  Let analyser = null;
  Let microphone = null;
  Let isRecording = false;
  Let animationId = null;

  // Beat detection
  Let lastBeatTime = 0;
  Let beatIntervals = [];
  Let currentBeat = 0;
  Let detectedTempo = 0;
```

```
// Rhythm
Let rhythmInterval = null;

// DOM
Const startBtn = document.getElementById('startBtn');
Const stopBtn  = document.getElementById('stopBtn');
Const micCircle= document.getElementById('micCircle');
Const status   = document.getElementById('status');
Const volumeFill = document.getElementById('volumeFill');
Const debug    = document.getElementById('debug');
Const canvas   = document.getElementById('canvas');
Const tempo    = document.getElementById('tempo');
Const beats    = document.getElementById('beats');
Const ctx      = canvas.getContext && canvas.getContext('2d');

Function log(message) {
  Console.log(message);
  Debug.innerHTML += new Date().toLocaleTimeString() + ': ' + message + '<br>';
  Debug.scrollTop = debug.scrollHeight;
}

Function updateStatus(message, type = 'info') {
  Status.textContent = message;
  Status.className = 'status ' + type;
  Log('STATUS: ' + message);
}
```

```
  startBtn.addEventListener('click', startMicrophone);
  stopBtn.addEventListener('click', stopMicrophone);
  micCircle.addEventListener('click', () => { if (!isRecording) startMicrophone(); else stopMicrophone(); });

  async function startMicrophone() {
   try {
    log('Starting microphone...');
    updateStatus('Requesting microphone permission...', 'info');

    audioContext = new (window.AudioContext || window.webkitAudioContext)();
    if (audioContext.state === 'suspended') await audioContext.resume();

    mediaStream = await navigator.mediaDevices.getUserMedia({ audio: true });
    microphone = audioContext.createMediaStreamSource(mediaStream);
    analyser = audioContext.createAnalyser();
    analyser.fftSize = 1024;
    analyser.smoothingTimeConstant = 0.3;
    microphone.connect(analyser);

    isRecording = true;
    startBtn.classList.add('hidden');
    stopBtn.classList.remove('hidden');
    micCircle.classList.add('recording');
```

```
    updateStatus('Microphone active! Make some noise!', 'success');

    lastBeatTime = 0;
    beatIntervals = [];
    detectedTempo = 0;
    tempo.textContent = '0 BPM';

    startAnalysis();
  } catch (error) {
    updateStatus('Mic error: ' + error.message, 'error');
  }
}

Function stopMicrophone() {
  isRecording = false;
  if (animationId) cancelAnimationFrame(animationId);
  if (mediaStream) { mediaStream.getTracks().forEach(t => t.stop()); }
  if (microphone) microphone.disconnect();
  analyser = null;
  if (audioContext) { audioContext.close(); audioContext = null; }
  stopRhythm();

  startBtn.classList.remove('hidden');
  stopBtn.classList.add('hidden');
  micCircle.classList.remove('recording');
  volumeFill.style.width = '0%';
```

```
  tempo.textContent = '0 BPM';
  beatIntervals = [];
  lastBeatTime = 0;
  currentBeat = 0;

  updateStatus('Microphone stopped', 'info');
}

Function startAnalysis() {
  If (!analyser) return;
  Const dataArray = new Uint8Array(analyser.frequencyBinCount);
  Const timeArray = new Uint8Array(analyser.fftSize);

  Function analyze() {
    If (!isRecording || !analyser) return;

    Analyser.getByteFrequencyData(dataArray);
    Analyser.getByteTimeDomainData(timeArray);

    // volume RMS
    Let sum = 0;
    For (let I = 0; I < timeArray.length; i++) {
      Const sample = (timeArray[i] – 128) / 128;
      Sum += sample * sample;
    }
    Const volume = Math.sqrt(sum / timeArray.length);
```

```
   Const volumePercent = Math.min(100, volume * 300);
   volumeFill.style.width = volumePercent + '%';

   detectBeat(volume);
   drawWaveform(dataArray, timeArray);

   animationId = requestAnimationFrame(analyze);
  }
  Analyze();
}

Function detectBeat(volume) {
  Const threshold = 0.05;
  Const currentTime = Date.now();
  If (volume > threshold) {
   Const timeSinceLastBeat = currentTime – lastBeatTime;
   If (timeSinceLastBeat > 300) {
    If (lastBeatTime > 0) {
     Const interval = timeSinceLastBeat / 1000;
     beatIntervals.push(interval);
     if (beatIntervals.length > 8) beatIntervals = beatIntervals.slice(-8);
     if (beatIntervals.length >= 3) updateTempo();
    }
    lastBeatTime = currentTime;
    animateBeat();
   }
```

```
  }
}

Function updateTempo() {
  Const sorted = [...beatIntervals].sort((a,b)=>a-b);
  Const median = sorted[Math.floor(sorted.length/2)];
  Const newTempo = Math.round(60/median);
  detectedTempo = Math.max(40, Math.min(240, newTempo));
  tempo.textContent = detectedTempo + ' BPM';
  log('Tempo: ' + detectedTempo);
  startRhythm(detectedTempo);
}

Function animateBeat() {
  Const beatElements = beats.querySelectorAll('.beat');
  beatElements.forEach(b => b.classList.remove('active'));
  const beatIndex = currentBeat % beatElements.length;
  beatElements[beatIndex].classList.add('active');
  setTimeout(() => beatElements[beatIndex].classList.remove('active'), 200);
  currentBeat++;
}

Function drawWaveform(freqArray, timeArray) {
  If (!ctx) return;
  Const dpr = window.devicePixelRatio || 1;
  Const displayWidth = canvas.offsetWidth;
```

```
Const displayHeight = canvas.offsetHeight;
Canvas.width = Math.floor(displayWidth * dpr);
Canvas.height = Math.floor(displayHeight * dpr);
Ctx.scale(dpr, dpr);
Ctx.clearRect(0, 0, displayWidth, displayHeight);
Const barCount = Math.floor(freqArray.length / 2);
Const barWidth = displayWidth / barCount;
Let x = 0;
For (let i=0; i<barCount; i++) {
  Const val = freqArray[i];
  Const barHeight = (val/255)*displayHeight*0.8;
  Const hue = Math.floor((i/barCount)*240);
  Ctx.fillStyle = 'hsl('+hue+',70%,50%)';
  Ctx.fillRect(x, displayHeight-barHeight, barWidth-1, barHeight);
  X += barWidth;
}
Ctx.beginPath();
Const step = Math.max(1, Math.floor(timeArray.length/displayWidth));
Const mid = displayHeight/2;
Ctx.lineWidth = 1;
For (let i=0, px=0; i<timeArray.length; i+=step, px++) {
  Const sample = (timeArray[i]-128)/128;
  Const y = mid + sample*mid*0.9;
  If (px===0) ctx.moveTo(px,y); else ctx.lineTo(px,y);
}
Ctx.strokeStyle='rgba(0,0,0,0.2)';
```

```
  Ctx.stroke();
  Ctx.setTransform(1,0,0,1,0,0);
}

// -------- Rhythm Generator ----------
Function startRhythm(bpm) {
  stopRhythm();
  if (!audioContext) return;
  const intervalMs = (60/bpm)*1000;
  let beatCount = 0;
  rhythmInterval = setInterval(() => {
    playBeat(beatCount);
    beatCount++;
  }, intervalMs);
}

Function stopRhythm() {
  If (rhythmInterval) { clearInterval(rhythmInterval); rhythmInterval = null; }
}

Function playBeat(beat) {
  If (!audioContext) return;
  If (beat % 4 === 0) playKick();
  Else if (beat % 4 === 2) playSnare();
  playHiHat();
}
```

```
Function playKick() {
  Const osc = audioContext.createOscillator();
  Const gain = audioContext.createGain();
  Osc.type='sine';
  Osc.frequency.setValueAtTime(150,audioContext.currentTime);
  Osc.frequency.exponentialRampToValueAtTime(50,audioContext.currentTime+0.5);
  Gain.gain.setValueAtTime(1,audioContext.currentTime);
  Gain.gain.exponentialRampToValueAtTime(0.001,audioContext.currentTime+0.5);
  Osc.connect(gain).connect(audioContext.destination);
  Osc.start(); osc.stop(audioContext.currentTime+0.5);
}

Function playSnare() {
  Const bufferSize=audioContext.sampleRate*0.2;
  Const buffer=audioContext.createBuffer(1,bufferSize,audioContext.sampleRate);
  Const data=buffer.getChannelData(0);
  For(let i=0;i<bufferSize;i++){ data[i]=Math.random()*2-1; }
  Const noise=audioContext.createBufferSource(); noise.buffer=buffer;
  Const filter=audioContext.createBiquadFilter(); filter.type='highpass';
filter.frequency.value=1000;
  Const gain=audioContext.createGain();
  Gain.gain.setValueAtTime(1,audioContext.currentTime);
  Gain.gain.exponentialRampToValueAtTime(0.01,audioContext.currentTime+0.2);
  Noise.connect(filter).connect(gain).connect(audioContext.destination);
  Noise.start(); noise.stop(audioContext.currentTime+0.2);
```

```
  }

  Function playHiHat() {
    Const bufferSize=audioContext.sampleRate*0.05;
    Const buffer=audioContext.createBuffer(1,bufferSize,audioContext.sampleRate);
    Const data=buffer.getChannelData(0);
    For(let i=0;i<bufferSize;i++){ data[i]=Math.random()*2-1; }
    Const noise=audioContext.createBufferSource(); noise.buffer=buffer;
    Const filter=audioContext.createBiquadFilter(); filter.type='highpass';
filter.frequency.value=5000;
    Const gain=audioContext.createGain();
    Gain.gain.setValueAtTime(0.5,audioContext.currentTime);
    Gain.gain.exponentialRampToValueAtTime(0.01,audioContext.currentTime+0.05);
    Noise.connect(filter).connect(gain).connect(audioContext.destination);
    Noise.start(); noise.stop(audioContext.currentTime+0.05);
  }

  Log('Page loaded, ready to test microphone rhythm');
 </script>
</body>
</html>
```