tts-reader.js

```
// TTS Lecture Reader for MkDocs - VibeVoice Edition
// Uses VibeVoice neural TTS for high-quality speech synthesis

class LectureReader {
  constructor() {
    // VibeVoice WebSocket endpoint
    this.ttsEndpoint = 'wss://rfh.tinymachines.ai/tts/stream';

    // Audio playback
    this.audioCtx = null;
    this.audioQueue = [];
    this.isProcessingAudio = false;
    this.socket = null;

    // State
    this.chunks = [];
    this.currentChunk = 0;
    this.isPlaying = false;
    this.isPaused = false;

    // Settings - actual VibeVoice presets
    this.selectedVoice = 'en-Carter_man';
    this.voices = [
      { name: 'Carter (Male)', value: 'en-Carter_man' },
      { name: 'Davis (Male)', value: 'en-Davis_man' },
      { name: 'Frank (Male)', value: 'en-Frank_man' },
      { name: 'Mike (Male)', value: 'en-Mike_man' },
      { name: 'Emma (Female)', value: 'en-Emma_woman' },
      { name: 'Grace (Female)', value: 'en-Grace_woman' },
      { name: 'Samuel (Indian)', value: 'in-Samuel_man' },
    ];

    // Audio settings
    this.sampleRate = 24000;
    this.bufferSize = 4096;
    this.prebufferSize = 8192; // Prebuffer before starting playback

    this.init();
  }

  init() {
    this.createUI();
    this.bindEvents();
    this.populateVoiceSelector();
  }

  createUI() {
    const widget = document.createElement('div');
    widget.id = 'tts-reader';
    widget.innerHTML = `
      <div class="tts-collapsed" id="tts-toggle">
        <svg viewBox="0 0 24 24" width="24" height="24">
          <path fill="currentColor" d="M3 9v6h4l5 5V4L7 9H3zm13.5 3c0-1.77-1.02-
3.29-2.5-4.03v8.05c1.48-.73 2.5-2.25 2.5-4.02zM14 3.23v2.06c2.89.86 5 3.54 5
```

```
6.71s-2.11 5.85-5 6.71v2.06c4.01-.91 7-4.49 7-8.77s-2.99-7.86-7-8.77z"/>
        </svg>      </div>      <div class="tts-expanded" id="tts-panel">
<div class="tts-header">           <span>Lecture Reader</span>           <button
id="tts-minimize" title="Minimize">−</button>        </div>        <div
class="tts-progress">           <div class="tts-progress-bar" id="tts-progress-
bar"></div>        </div>        <div class="tts-controls">           <button
id="tts-prev" title="Previous section">⏮</button>           <button id="tts-play"
title="Play">▶</button>           <button id="tts-next" title="Next
section">⏭</button>        </div>        <div class="tts-settings">
<label>            Voice:            <select id="tts-voice"></select>
</label>        </div>        <div class="tts-status" id="tts-status">Ready</div>
</div>    `;    document.body.appendChild(widget);  }  populateVoiceSelector() {
const selector = document.getElementById('tts-voice');    if (!selector) return;
selector.innerHTML = '';    this.voices.forEach(voice => {      const option =
document.createElement('option');      option.value = voice.value;
option.textContent = voice.name;      if (voice.value === this.selectedVoice) {
option.selected = true;      }      selector.appendChild(option);    });  }
bindEvents() {    document.getElementById('tts-toggle').addEventListener('click',
() => this.togglePanel());    document.getElementById('tts-
minimize').addEventListener('click', () => this.togglePanel());
document.getElementById('tts-play').addEventListener('click', () =>
this.togglePlayback());    document.getElementById('tts-
prev').addEventListener('click', () => this.prevChunk());
document.getElementById('tts-next').addEventListener('click', () =>
this.nextChunk());    document.getElementById('tts-
voice').addEventListener('change', (e) => {      this.selectedVoice =
e.target.value;    });    // Keyboard shortcuts
document.addEventListener('keydown', (e) => {      if (e.target.tagName ===
'INPUT' || e.target.tagName === 'TEXTAREA') return;      if (e.key === ' ' &&
e.ctrlKey) {        e.preventDefault();        this.togglePlayback();      } else
if (e.key === 'ArrowRight' && e.ctrlKey) {        e.preventDefault();
this.nextChunk();      } else if (e.key === 'ArrowLeft' && e.ctrlKey) {
e.preventDefault();        this.prevChunk();      }    });  }  togglePanel() {
const panel = document.getElementById('tts-panel');    const toggle =
document.getElementById('tts-toggle');    panel.classList.toggle('visible');
toggle.classList.toggle('hidden');    if (this.chunks.length === 0) {
this.extractContent();    }  }  extractContent() {    const content =
document.querySelector('.md-content__inner') ||
document.querySelector('article') ||
document.querySelector('main');    if (!content) {      this.setStatus('No
content found');      return;    }    const clone = content.cloneNode(true);
const removeSelectors = [      'script', 'style', 'nav', '.headerlink', '.toc',
'pre', 'code', '.highlight',      '.admonition.note',      '#tts-reader',
'[data-tts-skip]'    ];    removeSelectors.forEach(sel => {
clone.querySelectorAll(sel).forEach(el => el.remove());    });    let text =
clone.textContent || '';    text = text.replace(/\s+/g, ' ').replace(/\n+/g,
'\n').trim();    // Chunk into ~500 char segments for streaming    this.chunks =
this.chunkText(text, 500);    this.currentChunk = 0;
this.setStatus(`${this.chunks.length} sections ready`);    this.updateProgress();
}  chunkText(text, maxLen = 500) {    const chunks = [];    const sentences =
text.match(/[^.!?]+[.!?]+/g) || [text];    let currentChunk = '';    for (const
sentence of sentences) {      if ((currentChunk + sentence).length > maxLen) {
if (currentChunk) chunks.push(currentChunk.trim());        currentChunk =
sentence;      } else {        currentChunk += sentence;      }    }    if
```

```
(currentChunk.trim()) chunks.push(currentChunk.trim());    return chunks;  }
initAudio() {    if (!this.audioCtx) {      this.audioCtx = new
(window.AudioContext || window.webkitAudioContext)({        sampleRate:
this.sampleRate      });    }    if (this.audioCtx.state === 'suspended') {
this.audioCtx.resume();    }  }  async play() {    if (this.chunks.length === 0)
{      this.extractContent();      if (this.chunks.length === 0) return;    }
this.initAudio();    this.isPlaying = true;    this.isPaused = false;
this.updatePlayButton();    await this.speakChunk(this.currentChunk);  }  async
speakChunk(index) {    if (index >= this.chunks.length) {      this.stop();
this.setStatus('Finished');      return;    }    this.currentChunk = index;
const text = this.chunks[index];    this.setStatus(`Loading section ${index +
1}...`);    this.updateProgress();    try {      await this.streamTTS(text);
// Auto-advance to next chunk      if (this.isPlaying && !this.isPaused) {
this.currentChunk++;        if (this.currentChunk < this.chunks.length) {
await this.speakChunk(this.currentChunk);        } else {          this.stop();
this.setStatus('Finished');        }      }    } catch (err) {
console.error('TTS error:', err);      this.setStatus(`Error: ${err.message}`);
this.stop();    }  }  streamTTS(text) {    return new Promise((resolve, reject)
=> {      const params = new URLSearchParams({        text: text,        voice:
this.selectedVoice,        cfg: '1.5'      });      const wsUrl =
`${this.ttsEndpoint}?${params.toString()}`;      this.socket = new
WebSocket(wsUrl);      this.socket.binaryType = 'arraybuffer';
this.audioQueue = [];      let audioStarted = false;      let totalSamples = 0;
this.socket.onopen = () => {        this.setStatus(`Section ${this.currentChunk +
1} of ${this.chunks.length}`);      };      this.socket.onmessage = (event) => {
if (typeof event.data === 'string') {          // JSON log message          try {
const msg = JSON.parse(event.data);            if (msg.event ===
'generation_error') {              reject(new Error(msg.data?.error ||
'Generation failed'));            }          } catch (e) {            // Ignore
parse errors          }          return;        }        // Binary PCM16 audio
data        const pcm16 = new Int16Array(event.data);        const float32 = new
Float32Array(pcm16.length);        for (let i = 0; i < pcm16.length; i++) {
float32[i] = pcm16[i] / 32768;        }        this.audioQueue.push(float32);
totalSamples += float32.length;        // Start playback after prebuffering
if (!audioStarted && totalSamples >= this.prebufferSize) {          audioStarted
= true;          this.playAudioQueue();        }      };      this.socket.onclose
= () => {        // Play remaining audio        if (this.audioQueue.length > 0 &&
!audioStarted) {          this.playAudioQueue();        }        // Wait for
audio to finish        const checkComplete = () => {          if
(this.audioQueue.length === 0 && !this.isProcessingAudio) {            resolve();
} else if (this.isPlaying) {            setTimeout(checkComplete, 100);
} else {            resolve();          }        };
setTimeout(checkComplete, 100);      };      this.socket.onerror = (err) => {
reject(new Error('WebSocket connection failed'));      };    });  }
playAudioQueue() {    if (this.isProcessingAudio || this.audioQueue.length === 0
|| !this.isPlaying) {      return;    }    this.isProcessingAudio = true;    //
Combine all queued audio    const totalLength = this.audioQueue.reduce((sum, arr)
=> sum + arr.length, 0);    const combined = new Float32Array(totalLength);
let offset = 0;    while (this.audioQueue.length > 0) {      const chunk =
this.audioQueue.shift();      combined.set(chunk, offset);      offset +=
chunk.length;    }    // Create and play audio buffer    const buffer =
this.audioCtx.createBuffer(1, combined.length, this.sampleRate);
buffer.getChannelData(0).set(combined);    const source =
this.audioCtx.createBufferSource();    source.buffer = buffer;
```

```
source.connect(this.audioCtx.destination);    source.onended = () => {
this.isProcessingAudio = false;      // Check for more audio      if
(this.audioQueue.length > 0) {        this.playAudioQueue();      }    };
source.start();  }  pause() {    this.isPaused = true;    this.isPlaying = false;
this.updatePlayButton();    this.setStatus('Paused');    if (this.socket &&
this.socket.readyState === WebSocket.OPEN) {      this.socket.close();    }  }
stop() {    this.isPlaying = false;    this.isPaused = false;
this.updatePlayButton();    this.audioQueue = [];    if (this.socket &&
this.socket.readyState === WebSocket.OPEN) {      this.socket.close();    }  }
togglePlayback() {    if (this.isPlaying) {      this.pause();    } else {
this.play();    }  }  prevChunk() {    this.stop();    this.currentChunk =
Math.max(0, this.currentChunk - 1);    this.updateProgress();
this.setStatus(`Section ${this.currentChunk + 1} of ${this.chunks.length}`);
this.play();  }  nextChunk() {    this.stop();    this.currentChunk =
Math.min(this.chunks.length - 1, this.currentChunk + 1);
this.updateProgress();    this.setStatus(`Section ${this.currentChunk + 1} of
${this.chunks.length}`);    this.play();  }  updatePlayButton() {    const btn =
document.getElementById('tts-play');    btn.textContent = this.isPlaying ? '⏸' :
'▶';    btn.title = this.isPlaying ? 'Pause' : 'Play';  }  updateProgress() {
const bar = document.getElementById('tts-progress-bar');    const percent =
this.chunks.length > 0      ? ((this.currentChunk + 1) / this.chunks.length) *
100      : 0;    bar.style.width = `${percent}%`;  }  setStatus(msg) {
document.getElementById('tts-status').textContent = msg;  }}// Initialize when
DOM is readydocument.addEventListener('DOMContentLoaded', () => {
window.lectureReader = new LectureReader();});
```