Ideas and Code for TTS for mkdocs

Native Browser API (no library needed)

The Web Speech API is built into modern browsers:

```
const utterance = new SpeechSynthesisUtterance("Hello world");
utterance.rate = 1.0;
utterance.pitch = 1.0;
utterance.voice = speechSynthesis.getVoices()[0];
speechSynthesis.speak(utterance);
```

This works in Chrome, Firefox, Safari, and Edge with no dependencies. Voice quality depends on the OS/browser's installed voices.

Libraries

- **meSpeak.js** — A pure JavaScript port of eSpeak. Works entirely offline, ~1MB, produces robotic but functional speech. Good for environments where you can't rely on browser voices.
- **speak.js** — Similar Emscripten-compiled eSpeak port.
- **ResponsiveVoice.js** — Commercial wrapper that falls back gracefully across browsers, but requires their service for some features.

For your Lecture Forge context, if you're generating audio versions of your lectures, the native `SpeechSynthesis` API would be the simplest starting point. Browser voices have improved significantly — macOS and Windows both include high-quality neural voices now. You'd just need to handle the async nature (voices load lazily) and potentially chunk long text to avoid browser limits.

Want me to show a more complete implementation pattern for longer-form content like lectures?

This is a great addition to your learning workflow! Here's a complete implementation for adding TTS to MkDocs:

Project Structure

```
docs/
├── javascripts/
│   └── tts-reader.js
├── stylesheets/
│   └── tts-reader.css
└── ...your markdown files...
mkdocs.yml
```

The TTS Reader Implementation

```javascript
// docs/javascripts/tts-reader.js

class LectureReader {
  constructor() {
    this.synth = window.speechSynthesis;
    this.utterance = null;
    this.chunks = [];
    this.currentChunk = 0;
    this.isPlaying = false;
    this.isPaused = false;
    this.voices = [];
    this.selectedVoice = null;
    this.rate = 1.0;

    // Browser limit safety - chunk at ~3000 chars
    this.chunkSize = 3000;

    this.init();
  }

  init() {
    // Voices load asynchronously in some browsers
    if (this.synth.onvoiceschanged !== undefined) {
      this.synth.onvoiceschanged = () => this.loadVoices();
    }
    this.loadVoices();
    this.createUI();
    this.bindEvents();
  }

  loadVoices() {
    this.voices = this.synth.getVoices();

    // Prefer high-quality voices
    const preferredVoices = [
      'Google US English',
      'Microsoft Zira',
      'Samantha',
      'Daniel',
      'Karen',
      'Alex'
    ];

    for (const name of preferredVoices) {
      const voice = this.voices.find(v => v.name.includes(name));
      if (voice) {
        this.selectedVoice = voice;
        break;
      }
    }

    // Fallback to first English voice
    if (!this.selectedVoice) {
      this.selectedVoice = this.voices.find(v => v.lang.startsWith('en')) ||
```

```
this.voices[0];    }    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>            Speed: <span id="tts-rate-display">1.0x</span>
<input type="range" id="tts-rate" min="0.5" max="2.5" step="0.1" value="1.0">
</label>          <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 = '';    // Group by
language    const englishVoices = this.voices.filter(v =>
v.lang.startsWith('en'));    englishVoices.forEach(voice => {      const option =
document.createElement('option');      option.value = voice.name;
option.textContent = `${voice.name} (${voice.lang})`;      if (voice ===
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-
rate').addEventListener('input', (e) => {      this.rate =
parseFloat(e.target.value);      document.getElementById('tts-rate-
display').textContent = `${this.rate.toFixed(1)}x`;      if (this.isPlaying) {
// Restart current chunk with new rate        const wasPlaying = this.isPlaying;
this.stop();        if (wasPlaying) this.play();      }    });
document.getElementById('tts-voice').addEventListener('change', (e) => {
this.selectedVoice = this.voices.find(v => v.name === e.target.value);      if
(this.isPlaying) {        this.stop();        this.play();      }    });    //
Keyboard shortcuts    document.addEventListener('keydown', (e) => {      // Only
if not typing in an input      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');    // Extract content on first open    if
(this.chunks.length === 0) {      this.extractContent();    }  }
```

```
extractContent() {    // Target MkDocs content area    const content =
document.querySelector('.md-content__inner') ||
document.querySelector('article') ||
document.querySelector('main');    if (!content) {      this.setStatus('No
content found');      return;    }    // Clone to avoid modifying the page
const clone = content.cloneNode(true);    // Remove elements we don't want read
const removeSelectors = [      'script', 'style', 'nav', '.headerlink', '.toc',
'pre', 'code', '.highlight', // Skip code blocks      '.admonition.note', //
Optional: skip admonitions      '#tts-reader'    ];
removeSelectors.forEach(sel => {      clone.querySelectorAll(sel).forEach(el =>
el.remove());    });    // Get text content    let text = clone.textContent ||
'';    // Clean up whitespace    text = text      .replace(/\s+/g, ' ')
.replace(/\n+/g, '\n')      .trim();    // Chunk by sentences, respecting the
size limit    this.chunks = this.chunkText(text);    this.currentChunk = 0;
this.setStatus(`${this.chunks.length} sections ready`);    this.updateProgress();
}  chunkText(text) {    const chunks = [];    // Split on sentence boundaries
const sentences = text.match(/[^.!?]+[.!?]+/g) || [text];    let currentChunk =
'';    for (const sentence of sentences) {      if ((currentChunk +
sentence).length > this.chunkSize) {        if (currentChunk) {
chunks.push(currentChunk.trim());        }        currentChunk = sentence;      }
else {        currentChunk += sentence;      }    }    if (currentChunk.trim()) {
chunks.push(currentChunk.trim());    }    return chunks;  }  play() {    if
(this.chunks.length === 0) {      this.extractContent();      if
(this.chunks.length === 0) return;    }    if (this.isPaused) {
this.synth.resume();      this.isPaused = false;      this.isPlaying = true;
this.updatePlayButton();      return;    }    this.speakChunk(this.currentChunk);
}  speakChunk(index) {    if (index >= this.chunks.length) {      this.stop();
this.setStatus('Finished');      return;    }    this.currentChunk = index;
const text = this.chunks[index];    this.utterance = new
SpeechSynthesisUtterance(text);    this.utterance.voice = this.selectedVoice;
this.utterance.rate = this.rate;    this.utterance.pitch = 1.0;
this.utterance.onstart = () => {      this.isPlaying = true;      this.isPaused =
false;      this.updatePlayButton();      this.setStatus(`Section ${index + 1} of
${this.chunks.length}`);      this.updateProgress();    };
this.utterance.onend = () => {      if (this.isPlaying && !this.isPaused) {
this.speakChunk(this.currentChunk + 1);      }    };    this.utterance.onerror =
(e) => {      if (e.error !== 'interrupted') {        console.error('TTS error:',
e);        this.setStatus(`Error: ${e.error}`);      }    };
this.synth.speak(this.utterance);  }  pause() {    this.synth.pause();
this.isPaused = true;    this.isPlaying = false;    this.updatePlayButton();
this.setStatus('Paused');  }  stop() {    this.synth.cancel();    this.isPlaying
= false;    this.isPaused = false;    this.updatePlayButton();  }
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', () => {  // Check for
Speech Synthesis support  if ('speechSynthesis' in window) {
window.lectureReader = new LectureReader();  } else {    console.warn('Speech
Synthesis not supported in this browser');  }});
```

The Stylesheet

```css
/* docs/stylesheets/tts-reader.css */

#tts-reader {
  position: fixed;
  bottom: 20px;
  right: 20px;
  z-index: 9999;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
}

.tts-collapsed {
  width: 50px;
  height: 50px;
  background: #2094f3;
  border-radius: 50%;
  display: flex;
  align-items: center;
  justify-content: center;
  cursor: pointer;
  color: white;
  box-shadow: 0 2px 10px rgba(0,0,0,0.2);
  transition: transform 0.2s, background 0.2s;
}

.tts-collapsed:hover {
  transform: scale(1.1);
  background: #1976d2;
}

.tts-collapsed.hidden {
  display: none;
}

.tts-expanded {
  display: none;
  width: 280px;
  background: white;
  border-radius: 12px;
  box-shadow: 0 4px 20px rgba(0,0,0,0.15);
  overflow: hidden;
}

.tts-expanded.visible {
  display: block;
}

.tts-header {
  background: #2094f3;
  color: white;
  padding: 12px 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-weight: 500;
```

```
}.tts-header button {  background: none;  border: none;  color: white;  font-
size: 18px;  cursor: pointer;  padding: 0 5px;}.tts-progress {  height: 4px;
background: #e0e0e0;}.tts-progress-bar {  height: 100%;  background: #2094f3;
width: 0%;  transition: width 0.3s;}.tts-controls {  display: flex;  justify-
content: center;  gap: 15px;  padding: 15px;}.tts-controls button {  width: 44px;
height: 44px;  border-radius: 50%;  border: none;  background: #f5f5f5;  font-
size: 16px;  cursor: pointer;  transition: background 0.2s;}.tts-controls
button:hover {  background: #e0e0e0;}#tts-play {  width: 54px;  height: 54px;
background: #2094f3;  color: white;  font-size: 20px;}#tts-play:hover {
background: #1976d2;}.tts-settings {  padding: 0 15px 15px;}.tts-settings label {
display: block;  margin-bottom: 10px;  font-size: 13px;  color: #666;}.tts-
settings input[type="range"] {  width: 100%;  margin-top: 5px;}.tts-settings
select {  width: 100%;  padding: 6px;  border: 1px solid #ddd;  border-radius:
4px;  margin-top: 5px;  font-size: 12px;}.tts-status {  padding: 10px 15px;
background: #f9f9f9;  font-size: 12px;  color: #888;  text-align: center;
border-top: 1px solid #eee;}/* Dark mode support for Material theme */[data-md-
color-scheme="slate"] .tts-expanded {  background: #2d2d2d;}[data-md-color-
scheme="slate"] .tts-controls button {  background: #404040;  color: #fff;}[data-
md-color-scheme="slate"] .tts-controls button:hover {  background: #505050;}
[data-md-color-scheme="slate"] .tts-settings label {  color: #aaa;}[data-md-
color-scheme="slate"] .tts-settings select {  background: #404040;  color: #fff;
border-color: #555;}[data-md-color-scheme="slate"] .tts-status {  background:
#252525;  color: #888;  border-color: #404040;}
```

MkDocs Configuration

```yaml
# mkdocs.yml

site_name: Rust Learning Lectures
theme:
  name: material
  palette:
    - scheme: default
      toggle:
        icon: material/brightness-7
        name: Switch to dark mode
    - scheme: slate
      toggle:
        icon: material/brightness-4
        name: Switch to light mode

extra_javascript:
  - javascripts/tts-reader.js

extra_css:
  - stylesheets/tts-reader.css
```

Usage

Once integrated, a speaker icon appears in the bottom-right corner. Click it to expand the reader panel, then hit play. The reader will work through the lecture content section by section, skipping code blocks automatically.

Keyboard shortcuts:

- `Ctrl+Space` — Play/Pause
- `Ctrl+←` — Previous section
- `Ctrl+→` — Next section

Tips for your lectures:

The reader automatically skips `<pre>` and `<code>` elements, which is perfect for your two-document format. The prose lecture plays as audio while you can glance at the code companion separately.

If you want certain sections read differently, you could add data attributes:

```
<div data-tts-skip>
This won't be read aloud.
</div>
```

Then modify `extractContent()` to honor those attributes.

Want me to extend this with features like bookmarking position, localStorage persistence for settings, or integration with the code companion documents?