SonicAI: Browser-Based AI Text-to-Song Generation System

A Web Audio API Approach to Real-Time Music Synthesis
with Formant Vocal Modeling and Speech Integration

Romi Nur Ismanto
rominur@gmail.com

April 2026

GitHub: github.com/romizone/sonic-ai
Live Demo: sonic-ai-dun.vercel.app

Abstract

This paper presents SonicAI, a browser-based text-to-song generation system that transforms textual lyrics into musical compositions in real-time using the Web Audio API. The system implements a multi-layered audio synthesis architecture combining procedural melody generation, genre-adaptive chord progressions, rhythm pattern synthesis, formant-based vocal modeling, and browser-native Speech Synthesis for lyric vocalization. SonicAI supports six musical genres (Pop, Rock, Jazz, Electronic, Classical, Lo-fi) with distinct tonal characteristics, and includes five preset Indonesian-language songs with English translations. The system operates entirely client-side without server dependencies, demonstrating that meaningful music generation is achievable within modern browser constraints. This paper details the system architecture, synthesis algorithms, genre modeling approach, and discusses the limitations of browser-based audio synthesis compared to deep learning approaches such as Suno AI.

Keywords: *Web Audio API, text-to-song, formant synthesis, procedural music generation, browser-based audio, speech synthesis*

1. Introduction

The field of AI-generated music has seen remarkable advances with systems like Suno AI, MusicLM, and MusicGen, which leverage large neural networks trained on millions of songs to produce human-quality vocal performances. However, these systems require substantial server-side GPU resources and are not accessible for local, offline, or educational use cases.

SonicAI explores an alternative approach: using the Web Audio API, a native browser technology, to create a fully client-side text-to-song system. While it cannot match the fidelity of deep learning models, it demonstrates key music synthesis concepts including melody generation from text, chord progression theory, drum pattern programming, and formant-based vocal synthesis in a zero-dependency, instantly accessible web application.

The contributions of this work include: (1) a character-to-note mapping algorithm that converts arbitrary text into melodic sequences following genre-specific scales, (2) a formant vocal synthesizer using multiple detuned oscillators with bandpass filter chains to simulate vowel-like vocal timbres, (3) a dual-vocal architecture combining synthesized vocal melodies with Speech Synthesis API for lyric articulation, and (4) a genre-adaptive music engine supporting six distinct genres with unique harmonic, rhythmic, and timbral characteristics.

2. System Architecture

SonicAI is implemented as a single HTML file (~1000 lines) containing all markup, styling, and JavaScript logic. The application follows a modular audio graph architecture built on the Web Audio API, with distinct processing chains for each musical component.

2.1 Audio Signal Flow

The audio routing follows a bus architecture where all instrument outputs feed into a master gain node, which then splits into dry and wet (reverb) paths before reaching the analyser node and final audio destination. The signal flow is as follows:

Component	Output	Destination
Melody Oscillators	Low-pass Filter	Master Gain
Formant Vocal Synth	Bandpass Filters + LP	Master Gain
Pad Chord Oscillators	Low-pass Filter	Master Gain
Bass Oscillator	Low-pass Filter (400Hz)	Master Gain
Drum Synthesizer	Drum Bus Gain	Master Gain
Master Gain (dry)	Dry Gain Node	Analyser + Output
Master Gain (wet)	Convolution Reverb	Analyser + Output

Table 1: Audio signal routing architecture

2.2 User Interface

The UI adopts a dark theme inspired by modern music production software, featuring a two-column layout with input controls on the left (lyrics textarea, genre selector, key selector, tempo slider, preset song dropdown) and output visualization on the right (frequency bar

visualizer, karaoke-style lyric display, playback controls with progress bar and volume). The design uses CSS custom properties for theming with a purple accent color scheme (#7c3aed).

3. Synthesis Algorithms

3.1 Text-to-Melody Conversion

The melody generation algorithm maps each character of the input text to a musical note using the following process: (1) Characters are converted to their ASCII code offset from 'a' (code - 97). (2) The resulting index is mapped to the current genre's scale using modulo arithmetic (index % scale.length). (3) Octave variation is applied based on the quotient (floor(index / scale.length) % 2), adding 12 semitones for alternating octave shifts. (4) Space characters produce rest notes (silence). This approach creates deterministic but musically coherent melodies that vary with different input texts while staying within the genre-appropriate scale.

3.2 Formant Vocal Synthesis

The vocal synthesizer creates vowel-like timbres using a subtractive synthesis approach. Each vocal note consists of three detuned sawtooth/triangle oscillators (+8 cents, 0, -6 cents) fed through a bank of three bandpass filters tuned to genre-specific formant frequencies. The formant frequencies determine the vowel character of the sound:

Genre	F1 (Hz)	F2 (Hz)	F3 (Hz)	Vocal Gain	Character
Pop	800	1200	2500	0.18	Bright, open
Rock	600	1000	2800	0.16	Darker, gritty
Jazz	700	1100	2600	0.16	Warm, smooth
Electronic	500	1500	3000	0.14	Synthetic, airy
Classical	900	1300	2400	0.20	Rich, resonant
Lo-fi	650	1050	2200	0.17	Muted, intimate

Table 2: Formant frequency configuration per genre

Each vocal note applies a smooth ADSR (Attack-Decay-Sustain-Release) envelope with attack time proportional to 15% of note duration and release at 25%. A natural vibrato effect is achieved using a 5.5Hz LFO modulating the fundamental frequency by 0.8%, with delayed onset matching the attack phase for realistic vocal expression.

3.3 Speech Synthesis Integration

In addition to the formant synthesizer, SonicAI uses the browser's native Speech Synthesis API to vocalize the actual lyric text. All speech utterances are queued within the user gesture context (button click handler) to comply with Chrome's autoplay policy. The speech parameters are tuned for a sing-like quality: rate is reduced to 0.6-0.95x (slower than natural speech), pitch is elevated to 1.35 (higher than default), and volume is set to 0.7 to blend behind the formant synth layer. The system attempts to select an Indonesian voice for Indonesian lyrics and a female English voice for English translations.

3.4 Chord Progression Engine

Each genre defines a four-chord progression mapped to semitone intervals from the root note. Chords are rendered as pad sounds using genre-specific waveforms (sine, triangle, sawtooth) with a detuned second oscillator (+5 cents) for stereo width. Each chord sustains for one bar (4 beats) with smooth crossfade envelopes to prevent clicks. Jazz and Lo-fi genres use extended chords (7ths) for harmonic richness.

Genre	Progression	Scale Type
Pop	I - IV - V - I (Major)	Ionian (Major)
Rock	i - iv - III - i (Minor)	Natural Minor
Jazz	Imaj7 - IVmaj7 - ii7 - I7	Blues + Chromatic
Electronic	I - iii - IV - V (Penta)	Pentatonic
Classical	I - iii - IV - I	Ionian (Major)
Lo-fi	i7 - iv7 - III7 - i7	Dorian

Table 3: Chord progressions and scale types per genre

3.5 Drum Pattern Synthesis

Drum sounds are synthesized procedurally without samples: Kick drums use a sine oscillator with rapid frequency sweep from 120Hz to 25Hz. Snare drums combine bandpass-filtered noise bursts (2000Hz center) with exponential decay. Hi-hats use high-pass filtered noise (7000Hz+) with very short decay (40ms). Each genre applies a distinct rhythmic pattern:

Genre	Kick Pattern	Snare Pattern	Hi-hat Pattern
Pop	Beat 1, 3	Beat 2, 4	Every beat + offbeat
Rock	Beat 1, 3	Beat 2, 4	Every beat + offbeat
Electronic	Every 2 beats	Beats 2, 4	16th notes (4 per beat)
Jazz	Beat 1	Beat 5 (of 8)	Swing pattern (triplet feel)
Lo-fi	Beat 1	Beat 3 (slightly late)	Every 2 beats
Classical	-	-	-

Table 4: Drum patterns per genre

3.6 Reverb and Effects

Spatial depth is achieved through convolution reverb using procedurally generated impulse responses. The impulse buffer is created by multiplying white noise with an exponential decay function modulated by a low-frequency sinusoid for early reflection simulation. Each genre specifies a decay time (1.2s for Rock to 4.0s for Classical) and wet/dry mix ratio (0.2 to 0.5). Additional per-instrument low-pass filtering provides frequency-dependent warmth, with cutoff frequencies ranging from 1800Hz (Lo-fi) to 5000Hz (Classical).

4. Preset Song Collection

SonicAI includes five original Indonesian-language songs with English translations, each assigned to a different genre to showcase the system's versatility:

#	Title	Artist	Genre	BPM	Key
1	Senja di Sudirman	Dian Sastro	Pop	110	C Major
2	Hujan di Senopati	Wulan	Jazz	95	A Minor
3	Kereta Terakhir	Davina	Rock	125	E Minor
4	Kopi dan Janji	Titi Kamal	Lo-fi	85	F Major
5	Lampu Kota	Davina	Electronic	128	G Major

Table 5: Preset song collection

All songs are themed around Jakarta city life, exploring themes of love, memory, and urban experience. The bilingual support (Indonesian/English) allows users to experience the same composition with different phonetic characteristics affecting the Speech Synthesis vocal layer.

5. Technical Implementation Details

5.1 Chrome Autoplay Policy Compliance

Modern browsers require AudioContext creation and Speech Synthesis initiation within a user-activated event handler. SonicAI addresses this by performing all audio initialization synchronously within the Generate button's click handler: the AudioContext is created, all audio nodes are scheduled, and all speech utterances are queued before any asynchronous operations. This ensures the browser recognizes the audio as user-initiated.

5.2 Real-Time Visualization

The frequency visualizer renders 80 vertical bars mapped to the AnalyserNode's frequency bin data (FFT size 512, smoothing constant 0.85). Each bar uses a vertical linear gradient colored according to the active genre. The canvas is rendered at 2x resolution for Retina display support and updates at the browser's native refresh rate via requestAnimationFrame.

5.3 Performance Considerations

A typical song generation schedules approximately 1000+ Web Audio nodes (oscillators, gains, filters). The Web Audio API handles scheduling efficiently through its internal audio thread, separate from the main JavaScript thread. Memory is managed by stopping and dereferencing all nodes when playback ends or a new song is generated.

6. Comparison with Deep Learning Approaches

Aspect	SonicAI (Browser)	Suno AI (Deep Learning)
Voice Quality	Formant synth + TTS	Neural vocal synthesis
Musicality	Rule-based, deterministic	Learned from millions of songs
Latency	Instant (client-side)	Seconds-minutes (server)
Dependencies	None (browser only)	GPU servers, API access
Offline Support	Full offline capability	Requires internet
Cost	Free, open source	Subscription-based
Customization	Genre, key, tempo, lyrics	Text prompt, style tags
Output Quality	Synthetic/robotic	Near-human quality
File Size	~40KB single HTML	Multi-GB models
Privacy	100% local processing	Data sent to servers

Table 6: Feature comparison between SonicAI and deep learning approaches

7. Limitations and Future Work

7.1 Current Limitations

The primary limitation is vocal quality: the formant synthesizer produces vowel-like sounds but cannot articulate consonants or words, while the Speech Synthesis API provides word articulation but with a distinctly robotic quality. The text-to-melody mapping, while deterministic and scale-appropriate, does not consider musical phrasing, dynamics, or emotional contour. Additionally, the system cannot learn from user preferences or improve over time, as it uses fixed algorithmic rules rather than trained models.

7.2 Future Directions

Several avenues exist for improvement: (1) Integration of WebAssembly-based neural vocoders (e.g., LPCNet) for more natural vocal synthesis. (2) Implementation of Markov chain or LSTM-based melody generation for more musical phrase structures. (3) Addition of AudioWorklet-based effects processing for real-time pitch shifting and time stretching. (4) Export functionality to WAV/MP3 using the MediaRecorder API. (5) MIDI input/output support via the Web MIDI API for integration with external instruments.

8. Technology Stack

Technology	Purpose	Specification
HTML5	Document structure	Single-page application
CSS3	Styling and animations	Custom properties, flexbox, grid
JavaScript (ES6+)	Application logic	Async/await, modules
Web Audio API	Audio synthesis and processing	AudioContext, OscillatorNode
Speech Synthesis API	Text-to-speech vocalization	SpeechSynthesisUtterance

Canvas API	Real-time visualization	2D rendering context
GitHub	Source code hosting	github.com/romizone/sonic-ai
Vercel	Production deployment	sonic-ai-dun.vercel.app

Table 7: Technology stack overview

9. Conclusion

SonicAI demonstrates that meaningful text-to-song generation is achievable entirely within the browser using standard Web APIs. While the output quality is fundamentally limited compared to deep learning approaches, the system serves as an educational tool for understanding music synthesis concepts, a rapid prototyping platform for musical ideas, and a proof of concept for client-side audio processing capabilities. The zero-dependency, single-file architecture ensures maximum accessibility and portability, requiring only a modern web browser to operate.

The dual-vocal architecture, combining formant synthesis for melodic expression with Speech Synthesis for lyric articulation, represents a novel approach to browser-based vocal modeling that balances musical quality with word intelligibility. As browser audio capabilities continue to evolve with technologies like AudioWorklet and WebAssembly, the gap between client-side and server-side music generation will continue to narrow.

References

[1] Web Audio API Specification, W3C Recommendation, 2021. https://www.w3.org/TR/webaudio/

[2] Speech Synthesis API, W3C Community Draft. https://wicg.github.io/speech-api/

[3] Copet, J. et al., "Simple and Controllable Music Generation," NeurIPS 2023.

[4] Agostinelli, A. et al., "MusicLM: Generating Music From Text," arXiv:2301.11325, 2023.

[5] Fant, G., "Acoustic Theory of Speech Production," Mouton, The Hague, 1960.

[6] Roads, C., "The Computer Music Tutorial," MIT Press, 1996.

[7] Smith, J.O., "Physical Audio Signal Processing," W3K Publishing, 2010.

[8] MDN Web Docs, "Web Audio API," Mozilla Developer Network, 2024.