Architecting a Modern Web Radio Application: From Frontend Optimization to Backend Security

Introduction and Frontend Build Optimization

The "Radio Adamowo" project is a lightweight, single-page web radio application designed for simplicity and performance. It is built using Vanilla JavaScript for the frontend logic, Vite as the build tool, and a simple PHP backend for handling dynamic data like comments. During the initial development phase, a significant technical hurdle emerged in the form of a frontend build failure, which provided an opportunity to refine the project's asset handling strategy.

This section details the investigation and resolution of a critical build error originating from Vite's HTML parser, parse5. The error, Unable to parse HTML; parse5 error code missing-whitespace-between-attributes, occurred when an inline SVG was embedded directly into an HTML element's class attribute using Tailwind CSS's arbitrary value syntax . The underlying issue is that the parse5 parser can fail to correctly interpret the complex string of a data:image/svg+xml URI [1]. Characters commonly found in SVG code, such as quotes ("), hash symbols (#), and angle brackets (<, >), can be misinterpreted by the parser as the beginning of new, improperly formatted HTML attributes, triggering the error .

The most robust and recommended solution is to externalize the inline SVG, moving it from the HTML markup into a dedicated CSS file . This approach not only resolves the parsing error but also leads to cleaner HTML and better code maintainability [2]. The process involves creating a custom CSS class that applies the SVG as a background-image.

css .bg-custom-icon { background-image: url("your-svg-data-uri-here"); background-repeat: no-repeat; background-size: contain; }

For this method to work reliably across all browsers, the SVG data must be properly formatted and URL-encoded [3]. Unencoded SVGs might only render correctly in certain browsers like those based on WebKit [3]. Best practices for

encoding include ensuring the root

The following table summarizes the key aspects of the problem and the applied solution.

Topic	Summary of Best Practices	Reference
Error Cause	A complex inline data:image/svg+xml string in an HTML class attribute confuses the parse5 HTML parser used by Vite.	[1]
Solution	Externalize the SVG data URI into a dedicated class in a separate .css file.	[4]
CSS Syntax	Use the background-image: url property within a custom CSS class.	[5]
Encoding	SVG data must be URL-encoded for cross-browser compatibility, especially characters like #, <, and >.	[3]
SVG xmlns	The root	[3]
Quote Management	Use alternate quote types between the url wrapper and the SVG's internal attributes (e.g., url("...") with fill='...').	[5]

By adopting this externalization strategy, the build error was resolved, and the project's frontend architecture was made more scalable and robust. This initial optimization set a precedent for maintaining clean code and adhering to web standards throughout the development process.

Advanced Audio Playback: HLS Streaming and Media Session API

Following the optimization of the application's build process, this section details the implementation of its core audio streaming functionality and enhanced user experience features. The application employs a robust, dual-pronged strategy for HTTP Live Streaming (HLS) to ensure wide browser compatibility and integrates the Media Session API to provide a seamless, native-like control experience.

To deliver the HLS audio stream, the application first attempts to use the hls.js library. Hls.js is a JavaScript library that enables HLS playback in browsers that support Media Source Extensions (MSE) [6]. It functions by reading the HLS manifest, then transmuxing the audio streams into ISO BMFF (MP4) fragments that can be fed into a standard HTML5

The core of the implementation is a conditional check using Hls.isSupported [7]. If this method returns true, it confirms that MSE is available, and an Hls.js instance is created to load and manage the stream [8]. However, some browsers, most notably Safari, have native HLS support and do not require hls.js [9]. For these cases, Hls.isSupported returns false. The application then falls back to detecting this native capability by checking the return value of audio.canPlayType('application/vnd.apple.mpegurl') [7]. If native support is present, the audio element's src attribute is set directly to the .m3u8 stream URL for the browser to handle playback natively [9].

Browser	Native HLS Playback Support
Safari (macOS/iOS)	Supported
Chrome (desktop)	Not Supported
Firefox	Not Supported
Edge	Not Supported
Android Chrome	Device-dependent

To improve the user experience, the application integrates the Media Session API. This API allows the web application to customize media notifications and respond to platform-level media controls, such as those on a device's lock screen, in the notification shade, or from hardware media keys [10]. This enables users to control playback without needing the application's browser tab to be active [10].

Integration involves two main tasks. First, the application provides the browser with information about the current media by assigning a MediaMetadata object to navigator.mediaSession.metadata [11]. This object contains dynamic details like the track's title, artist, and album (the station name, in this case) [10]. It also includes an artwork property, which is an array of image objects, each with a src, sizes, and type [12]. Providing multiple artwork sizes allows the user agent to select the most appropriate image for the specific UI context, such as a lock screen versus a compact notification [13].

Second, the application defines action handlers to respond to user interactions with the media controls. This is achieved with the navigator.mediaSession.setActionHandler method, which registers a callback function for a specific action [12]. By setting handlers for play and pause, the application signals to the browser that it supports these actions, which in turn makes the corresponding controls visible in the platform UI [13]. The associated callback functions execute the necessary logic, such as calling audio.play or audio.pause, and update the navigator.mediaSession.playbackState to keep the UI synchronized [13].

The following code demonstrates the combined logic for HLS streaming and Media Session API integration: javascript const audio = document.getElementById('radio-player'); const streamUrl = 'https://your-radio-stream.m3u8';

// 1. HLS Streaming Logic if (Hls.isSupported) { console.log("Hls.js is supported. Initializing HLS player."); const hls = new Hls; hls.loadSource(streamUrl); hls.attachMedia(audio); hls.on(Hls.Events.MANIFEST_PARSED, => audio.play); } else if (audio.canPlayType('application/vnd.apple.mpegurl')) { console.log("Using native HLS playback."); audio.src = streamUrl; audio.addEventListener('loadedmetadata', => audio.play); } else { console.error("HLS streaming is not supported in this browser."); }

// 2. Media Session API Integration if ('mediaSession' in navigator) { // Set dynamic metadata for the currently playing track navigator.mediaSession.metadata = new MediaMetadata({ title: "Current Song Title", artist: "Artist Name", album: "Radio Adamowo", artwork: [{ src: 'path/to/icon-96x96.png', sizes: '96x96', type: 'image/png' }, { src: 'path/to/icon-512x512.png', sizes: '512x512', type: 'image/png' },] });

// Define action handlers for play and pause navigator.mediaSession.setActionHandler('play', async => { await audio.play; navigator.mediaSession.playbackState = 'playing'; });

navigator.mediaSession.setActionHandler('pause', => { audio.pause; navigator.mediaSession.playbackState = 'paused'; }); }

Progressive Web App and Offline Support

To provide a reliable and app-like user experience, the application is implemented as a Progressive Web App (PWA). This is achieved using the vite-plugin-pwa, a Vite-specific plugin that simplifies PWA creation by leveraging Google's Workbox library to generate and manage a service worker . The service worker is a crucial component that intercepts network requests, enabling advanced caching strategies and offline functionality [14].

A key aspect of the implementation is a dual caching strategy designed to balance offline availability with the need for live data. The configuration ensures a robust offline-first experience for the application's core structure while guaranteeing that dynamic content is always current.

Precaching Core UI Assets

The primary goal of the PWA implementation is to make the application shell—the core user interface—load instantly, regardless of network connectivity. To accomplish this, the service worker is configured to precache all essential static assets during its installation phase. By default, the plugin precaches html, js, and css files, but this has been extended via the workbox.globPatterns option to include all UI-related assets such as icons, images, and fonts (ico, png, svg, woff, woff2) . This ensures that the entire user interface is stored on the user's device, providing an immediate and reliable loading experience.

Network-Only Strategy for Dynamic Content

For a web radio application, it is critical that live and dynamic content is never served from a cache. To enforce this, a NetworkOnly strategy is implemented for specific types of requests [15]. This strategy instructs the service worker to always fetch the resource directly from the network, bypassing the cache entirely. This is configured using the workbox.runtimeCaching option, which defines rules for handling requests that are not part of the precache manifest [16].

This NetworkOnly strategy is applied to two critical areas: 1. **HLS Audio Stream:** To prevent the service worker from caching segments of the live audio stream, a runtimeCaching rule targets HLS file types (.m3u8 and .ts). This ensures that users always receive the live broadcast without interruption from stale, cached audio segments. 2. **API Requests:** All API calls, such as those for fetching comments or other real-time data (identified by paths ending in .php), are also handled with a NetworkOnly strategy. This guarantees that the data displayed to the user is always the most up-to-date information available from the server [15].

To provide context on the available caching mechanisms, the table below outlines the common Workbox strategies.

Strategy	Description
NetworkOnly	Always attempts to fetch the request from the network. It will fail if the user is offline. Ideal for live data and non-GET requests [15].
CacheFirst	Serves the response from the cache first. If the resource is not in the cache, it is requested from the network, and the response is cached for future requests [15].
NetworkFirst	Attempts to get the resource from the network first. If the network request is successful, it updates the cache. If it fails, it falls back to the cached version [15].
CacheOnly	Only responds with assets that are already in the cache. It will never go to the network [15].

Finally, the PWA is configured with registerType: 'autoUpdate' to ensure that the service worker updates in the background without prompting the user, providing a seamless transition to new versions of the application as they are deployed [15]. This strategic combination of precaching and runtime caching rules results in a fast, reliable, and always-current PWA.

Backend Security and Data Integrity

Following the implementation of the Progressive Web App features, the focus shifts to the server-side architecture. The PHP backend is fortified with several critical security measures to ensure data integrity and protect against common web vulnerabilities. This section details the implementation of the Synchronized Token Pattern for Cross-Site Request Forgery (CSRF) mitigation, a database-driven rate-limiting mechanism to prevent comment spam, and the use of prepared statements to defend against SQL injection.

Cross-Site Request Forgery (CSRF) Protection

The primary defense against Cross-Site Request Forgery (CSRF)—an attack that tricks an authenticated user's browser into performing unwanted actions—is the Synchronized Token Pattern . This method establishes a secure communication channel by verifying that state-changing requests originate from the application's own frontend and not a malicious third-party site [17]. The implementation follows a four-step workflow, from token generation to server-side validation.

Step	Action	Component	Description
1. Generation	The server generates a secure 32-byte token.	get_csrf_token.php	The token is created using random_bytes and stored in the user's session ($_SESSION['csrf_token']) .
2. Retrieval	The client fetches the token upon initialization.	app.js	An asynchronous fetch call is made to get_csrf_token.php, and the token is stored in a JavaScript variable [18].
3. Submission	The client sends the token with state-changing requests.	app.js	The token is included in a custom X-CSRF-Token header for POST requests, such as adding a comment [18].
4. Validation	The server validates the submitted token.	add_comment.php	The server compares the header token with the session token using hash_equals. A mismatch results in an HTTP 403 error [18].

The workflow begins on the server, where the get_csrf_token.php script generates a unique, cryptographically secure, and unpredictable token for each user session [17]. This token, generated using PHP's random_bytes function to a length of 32 bytes (256 bits) for brute-force resistance, is then stored in the $_SESSION superglobal . Upon loading, the client-side JavaScript application fetches this token and stores it locally [18].

For any state-changing operation, such as submitting a new comment, the client must include this token in a custom X-CSRF-Token HTTP header with its POST request [18]. On the backend, the add_comment.php script validates the request by comparing the token received in the header against the one stored in the user's session. This comparison is performed using the timing-attack-safe hash_equals function [18]. If the tokens do not match, the server immediately rejects the request with an HTTP 403 Forbidden status code, effectively thwarting the CSRF attempt [18].

Spam Prevention via Rate Limiting

To further enhance data integrity and prevent comment spam, a database-driven rate-limiting mechanism was implemented in the add_comment.php endpoint. This system enforces a cool-down period between submissions from the same user, who is identified by a hash of their IP address.

When a user submits a comment, the backend first creates a SHA256 hash of their IP address ($_SERVER['REMOTE_ADDR']) for privacy and consistent identification. It then executes the following SQL query to find the timestamp of the most recent comment associated with that IP hash: sql SELECT created_at FROM comments WHERE ip_hash = ? ORDER BY created_at DESC LIMIT 1;

If a previous comment exists, the system calculates the time elapsed since its submission. If this duration is less than the defined 60-second cool-down period, the request is denied with an HTTP 429 Too Many Requests status code and a JSON response indicating the remaining wait time. If no prior comment is found for the IP hash, the user is treated as a first-time commenter, and the rate-limiting check is bypassed, allowing the comment to be processed.

SQL Injection Prevention

Finally, to protect against SQL injection attacks, all database interactions are handled using PHP Data Objects (PDO) with prepared statements. This practice is a fundamental security measure that separates the SQL query logic from the user-supplied data. By parameterizing the inputs, the database engine treats the data as literal values rather than executable code, which effectively neutralizes the threat of malicious SQL being injected into the queries. This layered approach, combining CSRF protection, rate limiting, and secure database queries, creates a robust security posture for the application's backend.

Playlist Integration and Educational Content Management

Building upon the secure backend infrastructure, the Radio Adamowo application incorporates a sophisticated playlist management system designed to deliver educational content with psychological themes. The playlist structure has been specifically crafted to address various aspects of psychological awareness, manipulation recognition, and emotional intelligence across different age groups and content formats.

Enhanced Playlist Structure

The application's playlist system is organized around six distinct categories, each serving a specific educational purpose while maintaining the core radio functionality. The playlist.json file follows a hierarchical structure that enables easy content management and dynamic loading:

json { "ambient": [{"title": "Whispers of Manipulation", "artist": "Silent Control", "url": "/public/music/hiphop/Utwor (1).mp3"}, {"title": "Echoes of a Toxic Mind", "artist": "Ethereal Shadows", "url": "/public/music/hiphop/Utwor (2).mp3"}, {"title": "Infinite Narcissistic Calm", "artist": "Mind Drift", "url": "/public/music/hiphop/Utwor (3).mp3"}], "disco": [{"title": "Dance of Deceptive Rhythms", "artist": "Disco Manipulation", "url": "/public/music/disco/track1.mp3"}, {"title": "Groove of Psychological Control", "artist": "Rhythmic Power", "url": "/public/music/disco/track2.mp3"}], "hiphop": [{"title": "Rap of Manipulative Power", "artist": "Urban Control", "url": "/public/music/hiphop/Utwor (6).mp3"}, {"title": "Rhymes of Toxic Cycles", "artist": "Street Mind Games", "url": "/public/music/hiphop/Utwor (7).mp3"}], "kids": [{"title": "Story of a Manipulative Friend", "artist": "Childhood Lessons", "url": "/public/music/kids/Utwor (1).mp3"}, {"title": "Learning to Spot Lies", "artist": "Young Minds", "url": "/public/music/kids/Utwor (2).mp3"}], "full": [{"title": "Symphony of Toxic Bonds", "artist": "Orchestra of Mind", "url": "/public/music/hiphop/Utwor (11).mp3"}, {"title": "Cycle of Manipulation Suite", "artist": "Eternal Control", "url": "/public/music/hiphop/Utwor (12).mp3"}], "podcasts": [{"id": "caseStudy", "title": "Studium Manipulacji: Adamowo", "url": "/public/music/kids/Utwor (6).mp3"}, {"id": "toxicCycle", "title": "Cykl Toksycznych Relacji", "url": "/public/music/kids/Utwor (7).mp3"}] }

Category-Specific Implementation

Each playlist category serves a distinct educational purpose within the application's psychological awareness framework:

Category	Purpose	Target Audience	Content Focus
ambient	Subtle psychological awareness through atmospheric music	General audience	Manipulation recognition, toxic relationship patterns
disco	Engaging rhythm-based learning	Young adults	Psychological control awareness through danceable content
hiphop	Urban storytelling for psychological education	Teens and young adults	Street-smart psychological awareness, power dynamics
kids	Age-appropriate psychological safety education	Children	Basic manipulation recognition, emotional safety
full	Comprehensive psychological education	All audiences	Complete psychological awareness programs
podcasts	In-depth educational content	Polish-speaking audience	Detailed case studies and educational materials

Dynamic Playlist Loading and Management

The application implements a robust playlist management system that dynamically loads content based on user selection and maintains state across sessions. The JavaScript implementation handles category switching, track progression, and educational content delivery:

javascript class PlaylistManager { constructor { this.currentPlaylist = null; this.currentTrack = 0; this.playlists = {}; this.loadPlaylists; }

async loadPlaylists { try { const response = await fetch('/playlist.json'); this.playlists = await response.json; this.initializeUI; } catch (error) { console.error('Failed to load playlists:', error); } }

switchCategory(category) { if (this.playlists[category]) { this.currentPlaylist = category; this.currentTrack = 0; this.updateMediaSession; this.loadTrack; } }

updateMediaSession { if ('mediaSession' in navigator && this.getCurrentTrack) { const track = this.getCurrentTrack; navigator.mediaSession.metadata = new MediaMetadata({ title: track.title, artist: track.artist, album: Radio Adamowo - ${this.currentPlaylist.toUpperCase}, artwork: [{ src: '/icons/playlist-96x96.png', sizes: '96x96', type: 'image/png' }, { src: '/icons/playlist-512x512.png', sizes: '512x512', type: 'image/png' }] }); } } }

Educational Content Integration

The playlist system is designed to seamlessly integrate educational content with entertainment value. Each track title and artist name has been carefully crafted to reinforce psychological awareness concepts while maintaining musical appeal. The "kids" category specifically addresses age-appropriate psychological safety education, using simple language and concepts that children can understand and apply in their daily lives.

The "podcasts" category represents a unique addition to the platform, providing Polish-language educational content that delves deeper into psychological manipulation patterns, toxic relationship cycles, and practical escape strategies. These podcasts are identified by unique IDs for easy reference and can be integrated with the application's comment system to facilitate educational discussions.

File Path Management and Organization

The playlist structure follows a logical file organization pattern that supports both development and production environments:

/public/music/ ├── ambient/ # Atmospheric psychological awareness content

├── disco/ # Rhythm-based educational content

├── hiphop/ # Urban psychological education

├── kids/ # Child-appropriate safety education

└── podcasts/ # In-depth Polish educational content

This organization enables efficient content delivery through the PWA's caching strategy while maintaining clear separation between different educational content types. The file naming convention supports automatic series recognition, allowing for easy expansion of content within each category as new educational materials are developed.

The integration of this playlist system with the existing HLS streaming infrastructure ensures that educational content is delivered with the same reliability and performance as traditional radio content, while the Media Session API integration provides users with familiar playback controls that enhance the learning experience across all supported devices and platforms.

References

[1] Unable to parse HTML; parse5 error code missing-wh...

[2] 09: SVG with Data URIs - CSS-Tricks

[3] URL-encoder for SVG

[4] Styling with utility classes - Core concepts

[5] Using an Data URI SVG as a CSS background image

[6] video-dev/hls.js - GitHub

[7] HLS.js in 2025: The Complete Guide to Adaptive Str...

[8] Home | hls.js

[9] Native HLS Playback: The Complete Guide for Develo...

[10] Media Session API - Web APIs - MDN - Mozilla

[11] MediaSession - Web APIs | MDN - Mozilla

[12] MediaSession: setActionHandler() method - Web APIs...

[13] Customize media notifications and playback control...

[14] Making a Site Work Offline Using the VitePWA Plugi...

[15] Vite and Progressive Web Apps - CODE Magazine

[16] Using Vite's Plugin for Progressive Web Apps (PWAs...

[17] How Synchronized Token Pattern Works? | by Chamo W...

[18] CSRF Protection in PHP - DEV Community