**LANStreamer - Technical Specification**

**1. Executive Summary**

LANStreamer is a Node.js-based application that transforms a standard Windows PC into a multi-channel audio streaming server for local area networks. This document outlines the system architecture, technology stack, file structure, and implementation details for the project.

**Related Documentation:**

* [Product Requirements](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/LANStreamer-PRD.md) - Business requirements and user stories
* [UI Design Specification](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/Admin-Dashboard-UI-Design.md) - Visual design guidelines and frontend requirements
* [Audio Monitoring Feature](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/Audio-Monitoring-Feature-Specification.md) - Professional monitoring implementation details
* [Authentication & Security](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/Authentication-Security-Specification.md) - Security requirements and implementation
* [Environment Configuration](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/env-example.md) - Configuration variables and deployment settings

**Document Information**

* **Project**: LANStreamer
* **Version**: 1.0.0
* **Date**: 2025-09-01
* **Author**: LANStreamer Development Team

**2. Technology Stack**

**2.1 Backend Technologies**

* **Runtime:** Node.js 16+ with Express.js framework
* **WebSocket:** Socket.io for real-time communication
* **Process Management:** Node.js child\_process for FFmpeg and Icecast control
* **Validation:** Joi for schema validation
* **Logging:** Winston with daily log rotation
* **System Information:** systeminformation library for hardware monitoring

**2.2 Frontend Technologies**

* **Framework:** Vue.js 3 with Composition API
* **State Management:** Vuex 4 for centralised state management
* **Routing:** Vue Router 4 for single-page application navigation
* **UI Framework:** Bootstrap 5 with custom SCSS styling
* **HTTP Client:** Axios for API communication
* **Real-time:** Socket.io-client for WebSocket connections

**2.3 Browser APIs and Client-Side Security**

**2.3.1 Device Permission Management**

**Implementation:** navigator.mediaDevices.getUserMedia() JavaScript API **Purpose:** Request browser permissions for audio/video device access **Security:** Browser-enforced permission system with user consent

**Permission Request Flow:**

async function requestDevicePermissions() {

try {

const stream = await navigator.mediaDevices.getUserMedia({ audio: true });

*// Permission granted - cleanup and proceed*

stream.getTracks().forEach(track => track.stop());

return { status: 'granted' };

} catch (error) {

if (error.name === 'NotAllowedError') {

return { status: 'denied', error: 'User denied permission' };

}

return { status: 'error', error: error.message };

}

}

**Permission States:**

* granted - User allowed device access
* denied - User blocked device access
* prompt - Permission not yet requested

**2.3.2 Authentication Security Implementation**

**JWT Token Structure:**

{

"userId": "admin",

"role": "administrator",

"iat": 1640995200,

"exp": 1641081600,

"isDefaultCredentials": true

}

**Security Configuration:**

const JWT\_CONFIG = {

secret: process.env.JWT\_SECRET || 'change-this-secret',

expiresIn: process.env.JWT\_EXPIRES\_IN || '24h',

algorithm: 'HS256'

};

**Password Validation:**

function validatePassword(password) {

const requirements = {

minLength: password.length >= 8,

hasLetter: /[a-zA-Z]/.test(password),

hasNumber: /\d/.test(password),

hasSymbol: /[!@#$%^&\*(),.?":{}|<>]/.test(password),

notDefault: password !== 'admin'

};

return {

isValid: Object.values(requirements).every(req => req),

requirements

};

}

**2.4 External Dependencies**

* **Audio Processing:** FFmpeg for audio encoding and streaming
* **Streaming Server:** Icecast for audio stream distribution
* **Audio Hardware:** Support for multi-channel USB audio interfaces

**Audio Concepts:** For detailed understanding of audio pipeline concepts and hardware integration, see [Audio Pipeline Concepts](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/LANStreamer-Audio-Pipeline-Concepts.md).

**2.4 Development & Testing Tools**

* **E2E Testing:** Playwright
* **Unit/Integration Testing:** Vitest
* **API Client for Tests:** Supertest
* **Linting:** ESLint
* **Dev Server:** Nodemon for development
* **Version Control:** Git

**3. System Architecture**

**3.1 Current Implementation Architecture**

**Backend Services:**

* **Express Server:** Main HTTP server (src/server.js) running on port 3001
* **Route Handlers:** Modular API endpoints organized by functionality:
  + src/routes/system.js - System management (Icecast, FFmpeg, audio devices)
  + src/routes/streams.js - Audio streaming control
  + src/routes/settings.js - Configuration management
  + src/routes/auth.js - Authentication endpoints
* **Service Layer:** Business logic services:
  + IcecastService.js - Windows-specific Icecast management
  + AudioDeviceService.js - Audio hardware detection
  + StreamService.js - FFmpeg process management
* **Configuration:** Environment-based configuration with .env support
* **Logging:** Winston-based logging with file rotation

**Frontend Architecture:**

* **Component-Based UI:** Dynamic loading with static fallbacks
* **Component Manager:** ComponentManager.js handles dynamic loading
* **Core Components:**
  + IcecastManager.js - Icecast server control panel
  + LobbyMusicPlayer.js - Audio playback interface
* **Static Templates:** HTML fallbacks when components fail to load
* **Responsive Design:** Tailwind CSS with Material Symbols icons

**Process Management:**

* **Windows Integration:** Native Windows tasklist/taskkill commands
* **Process Tracking:** PID monitoring and lifecycle management
* **File Validation:** Comprehensive installation verification
* **Error Recovery:** Graceful failure handling with user feedback

**4. File Structure**

The file structure outlined in the README.md is accurate and will be followed. Key development will occur within vue-app/src (backend) and vue-app/frontend/src (frontend).

**5. Current API Implementation**

**5.1 Implemented Endpoints**

**System Management (/api/system/\*):**

GET /api/health *// Server health check*

GET /api/system/audio-devices *// List available audio devices*

GET /api/system/ffmpeg-check *// Verify FFmpeg installation*

GET /api/system/ffmpeg-processes *// List running FFmpeg processes*

*// Icecast Management*

GET /api/system/icecast-status *// Basic Icecast status*

GET /api/system/icecast/detailed-status *// Detailed status with mountpoints*

POST /api/system/icecast/start *// Start Icecast server*

POST /api/system/icecast/stop *// Stop Icecast server*

POST /api/system/icecast/restart *// Restart Icecast server*

POST /api/system/icecast/check-installation *// Verify Icecast installation*

GET /api/system/icecast/search-installations *// Search for installations*

GET /api/system/icecast/validate-config *// Validate configuration*

**Stream Management (/api/streams/\*):**

GET /api/streams *// List active streams*

POST /api/streams/start *// Start audio stream*

POST /api/streams/stop *// Stop audio stream*

**Configuration (/api/settings/\*):**

GET /api/settings *// Get current settings*

POST /api/settings/update *// Update settings*

**5.2 Icecast Service Implementation**

**Windows-Specific Features:**

* **Installation Detection:** Multi-path search including Program Files directories
* **File Validation:** Comprehensive checking of icecast.exe, icecast.xml, batch files
* **Process Management:** Windows tasklist/taskkill integration with PID tracking
* **Configuration Validation:** XML parsing and path verification
* **Error Recovery:** Graceful handling of missing files and permissions

**Configuration Example:**

*// Environment variables for manual path configuration*

ICECAST\_EXE\_PATH="C:\Program Files (x86)\Icecast\bin\icecast.exe"

ICECAST\_CONFIG\_PATH="C:\Program Files (x86)\Icecast\icecast.xml"

ICECAST\_ACCESS\_LOG="C:\Program Files (x86)\Icecast\logs\access.log"

ICECAST\_ERROR\_LOG="C:\Program Files (x86)\Icecast\logs\error.log"

**5.3 Component Architecture Implementation**

**Dynamic Loading System:**

*// ComponentManager.js handles component lifecycle*

const componentRegistry = {

'icecast-server': IcecastManager,

'lobby-music-player': LobbyMusicPlayer

};

*// Fallback to static HTML if component fails*

if (component && componentClass) {

new componentClass(targetId);

} else {

container.innerHTML = staticTemplate;

}

**Error Handling Strategy:**

* Service-level error catching with detailed logging
* User-friendly error messages in the UI
* Graceful degradation when external processes fail
* Comprehensive installation troubleshooting

**5.4 Future Implementation Details**

* **Stream Management:** FFmpeg command generation for audio encoding
* **Real-time Communication:** WebSocket integration for live updates
* **Audio Device Integration:** Hardware detection and configuration

**6. Deployment**

The final deliverable will be a single .exe file created using a packager like **pkg** or **nexe**. This will bundle the Node.js server, Vue.js frontend, and the FFmpeg/Icecast binaries into one executable, eliminating external dependencies for the end-user.

**7. Development Status & Roadmap**

**7.1 Completed Implementation ✅**

**Backend Foundation:**

* ✅ **Express Server:** Core HTTP server with health check (src/server.js)
* ✅ **Route Structure:** Modular API endpoints with proper separation of concerns
* ✅ **Icecast Integration:** Complete Windows-specific management service
* ✅ **Configuration System:** Environment-based configuration with .env support
* ✅ **Error Handling:** Comprehensive logging and user-friendly error reporting

**System Management:**

* ✅ **Installation Detection:** Multi-path Icecast discovery with file validation
* ✅ **Process Management:** Windows tasklist/taskkill integration with PID tracking
* ✅ **Configuration Validation:** XML parsing and troubleshooting assistance
* ✅ **Audio Device Service:** Basic device detection infrastructure

**Frontend Architecture:**

* ✅ **Component System:** Dynamic loading with static fallbacks
* ✅ **Icecast Manager:** Complete UI for server control and monitoring
* ✅ **Installation Validation:** File status grid and troubleshooting interface
* ✅ **Responsive Design:** Tailwind CSS with Material Symbols

**7.2 Current Development Priorities 🚧**

**Audio Streaming Pipeline:**

* 🚧 **FFmpeg Integration:** Command generation and process management
* 🚧 **Stream Configuration:** Mount point setup and audio device mapping
* 🚧 **Real-time Monitoring:** Audio levels and stream health indicators

**User Interface Completion:**

* 🚧 **Stream Management UI:** Start/stop controls for individual streams
* 🚧 **Audio Device Selection:** Device picker with real-time detection
* 🚧 **Listener Interface:** Mobile-first web player for stream consumption

**7.3 Upcoming Features 📋**

**Core Functionality:**

* 📋 **WebSocket Integration:** Real-time status updates without polling
* 📋 **QR Code Generation:** Easy listener access to stream URLs
* 📋 **Stream Statistics:** Listener counts and connection monitoring
* 📋 **Advanced Error Recovery:** Automatic restart and health monitoring

**Deployment & Distribution:**

* 📋 **Executable Packaging:** Single .exe file with bundled dependencies
* 📋 **Installation Wizard:** Guided setup for Icecast and FFmpeg
* 📋 **Documentation:** Complete user manual and troubleshooting guide

**7.4 Technical Debt & Optimizations**

**Performance:**

* Address repetitive "Icecast installation verified" logging
* Implement caching for installation detection results
* Optimize component loading and error boundaries

**Testing Coverage:**

* Expand Playwright E2E tests for core workflows
* Add comprehensive unit tests for service classes
* Implement integration tests for process management

**Product Requirements Document: LANStreamer v1.0**

**1. Introduction**

This document outlines the requirements for **LANStreamer v1.0**, a desktop application that turns a standard Windows PC into a local, multi-channel audio streaming server. The primary goal is to provide a user-friendly, web-based interface for managing and broadcasting low-latency audio streams over a Local Area Network (LAN). This solution is designed for non-technical users in environments like conferences, houses of worship, and live events for purposes such as language interpretation or assistive listening.

**2. Core Objective & Philosophy**

**Simplicity is paramount.** The entire user experience is built around abstracting the technical complexity of audio streaming. The system must be operable by users with no command-line or audio engineering experience. The v1.0 product is exclusively a **local network solution**; it does not require and will not use an internet connection.

**3. User Stories (MVP)**

* **As an Event Admin,** I want the application to automatically detect all available audio input devices on my computer so I can easily select them for streaming.
* **As an Event Admin,** I want to name each audio stream (e.g., "French Interpretation," "Main Stage Audio") so they are clearly identifiable to listeners.
* **As an Event Admin,** I want to start and stop all streams with a single click and also be able to manage each stream individually.
* **As an Event Admin,** I want to see a clear, real-time status indicator (e.g., "Live," "Stopped," "Error") for each stream.
* **As an Event Admin,** I want to be able to monitor any live audio stream on a specific output device so I can check its quality before broadcasting to listeners. *(See*[*Audio Monitoring Feature Specification*](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/Audio-Monitoring-Feature-Specification.md)*for detailed implementation)*
* **As a Listener,** I want to connect to a simple web page on my phone, see a list of available audio streams, and click one to start listening immediately.
* **As a Listener,** I want the audio to play directly in my mobile web browser without needing to download a special app.

**4. Scope & Features (v1.0)**

**In Scope**

* **Packaged Application:** A single, standalone executable for Windows (.exe) that bundles the Node.js server and the web frontend. No installation of external dependencies (like Node.js) is required for the end-user.
* **Web-Based Dashboard (Admin):**
  + Runs locally, accessible at http://localhost:3001.
  + **Icecast Server Management:** Automatic detection, installation validation, and process control with Windows-specific optimizations.
  + **Audio device detection and selection:** Real-time discovery of available audio input devices.
  + **Component-Based Architecture:** Dynamic loading of UI components with static fallbacks for enhanced reliability.
  + Ability to name/label stream mount points (e.g., /french, /spanish).
  + Start/Stop controls for individual and all streams with process ID tracking.
  + **Real-time status display** for each FFmpeg/Icecast process with file validation indicators.
  + **Configuration validation** and troubleshooting assistance.
  + *(UI Design: See*[*Admin Dashboard UI Design*](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/Admin-Dashboard-UI-Design.md)*for complete visual specifications)*
* **Web-Based Player (Listener):**
  + A simple, mobile-first webpage served on the local network (e.g., http://[SERVER\_IP]:PORT/listen).
  + Dynamically updated list of active streams.
  + Simple "Play" button for each stream.
  + QR code on the admin dashboard for easy listener access to the player page.
* **Backend (Node.js/Express):**
  + Manages the lifecycle of Icecast and FFmpeg processes.
  + Serves the admin dashboard and listener player page.
  + Provides a simple API for the frontend to interact with.

**Out of Scope (for v1.0)**

* Video streaming.
* Internet broadcasting.
* User authentication or security roles.
* Recording, analytics, or statistics.
* Support for any operating system other than Windows.
* Advanced audio mixing, effects, or processing.
* The "Companion/Helper App" for Zoom/Teams integration.

**5. Technical Requirements**

* **Stack:** Node.js with Express for the backend, Vue.js for the frontend.
* **Core Dependencies:** The application will manage bundled, portable versions of **FFmpeg** and **Icecast**.
* **Deployment:** The final deliverable will be a single .exe file created using a packager like pkg or nexe.
* **Audio Device Discovery:** The backend must be able to programmatically list available audio input devices on the Windows host.
* **Error Handling:** The application must gracefully handle errors (e.g., FFmpeg process failure, device disconnection) and display a clear message to the admin.
* **Audio Assets:** Sample audio files are stored in the /assets directory for testing and demonstration purposes.

**Detailed Technical Implementation:** See [Technical Specification](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/LANStreamer-Technical-Specification.md) for complete API documentation and system architecture.

**5.1 Current Implementation Status (v1.0-alpha)**

**✅ Completed Features**

* **Backend Infrastructure:** Node.js/Express server with comprehensive API endpoints
* **Icecast Integration:** Complete Windows-specific detection, validation, and process management
* **Component Architecture:** Dynamic loading system with static fallbacks
* **Configuration Management:** Environment-based configuration with .env support
* **Installation Detection:** Multi-path search with file validation for Icecast installations
* **Process Management:** Windows tasklist/taskkill integration with PID tracking
* **Error Handling:** Comprehensive logging and user-friendly error reporting
* **Admin Dashboard Core:** Icecast server management panel with real-time status

**🚧 In Development**

* Audio device detection and selection interface
* Stream configuration and mount point management
* FFmpeg integration for audio streaming
* Listener web player interface

**📋 Planned**

* QR code generation for listener access
* Real-time stream monitoring and statistics
* Final packaging as standalone .exe

**Installation Guides:** Complete setup documentation available:

* [Icecast Installation Guide](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/guides/icecast-installation.md)
* [FFmpeg Installation Guide](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/guides/ffmpeg-installation.md)
* [Environment Configuration](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Username/Documents/github/LANStreamer/docs/env-example.md)

**6. Future Vision & Ecosystem**

This section captures potential future enhancements and companion applications that build upon the core LANStreamer solution. These are explicitly **out of scope for v1.0** but inform the architectural design.

**Core Application Enhancements (vNext)**

* **Video Integration:** Add a video player to the listener page, with the backend capable of receiving an RTMP stream from OBS to create a full, self-hosted, multi-language video platform.
* **Recording & Analytics:** Add features to record streams and provide basic analytics (e.g., listener count, uptime).
* **Cross-Platform Support:** Expand support to macOS and Linux.
* **User Roles & Security:** Introduce authentication for the admin dashboard.

**Ecosystem & Integrations**

* **Internet Gateway:** A feature to easily re-stream a local channel to an external RTMP service like YouTube, Twitch, or Castr.io.
* **"Virtual Audio Router" Helper App:** A separate, installable companion application for listeners. It would receive a stream from the LANStreamer server and create a virtual audio device on the user's machine. This would allow them to select an interpretation channel as their audio source in applications like Zoom, Teams, or Google Meet.
* **OBS Plugin:** A plugin for OBS Studio that would display the LANStreamer audio channels directly within the OBS interface, allowing for easier integration into complex broadcast productions.
* **AI Live Translation:** Integration with services like ElevenLabs or Azure AI Speech to provide real-time, AI-powered translation for a given audio source, creating a new, automated language stream.