Nodeaa: A Browser-Based Platform for Modular Sound Processing and Node-Based Audio Routing Using the WebAudio API

August Ross
aross3@oxy.edu
Occidental College

1 Introduction

Digital Audio Workstations (DAWs) have long been essential tools for musicians, sound designers, and audio engineers. However, traditional DAWs are often limited by their dependency on specific operating systems, high system resource requirements, price, and the need for installation and maintenance. With the increasing prevalence of browser-based applications and the capabilities of the WebAudio API [21], the browser has emerged as a viable platform for real-time audio synthesis and manipulation. *Nodeaa*, a modular sound processing and node-based dynamic audio routing tool, demonstrates the potential of the browser as a fully functional DAW. Inspired by established tools like MaxMSP and Pure Data, *Nodeaa* enables users to create, manipulate, and route audio signals through an intuitive node-based interface, all within a web browser.

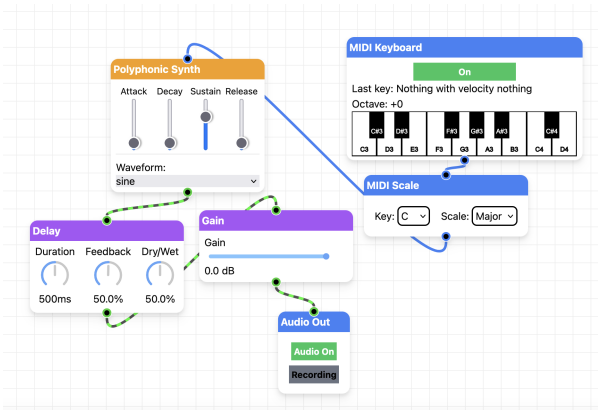

Figure 1: Nodeaa's interface and a basic signal flow.

2 Problem Statement

The primary problem *Nodeaa* addresses is the lack of accessible, cross-platform tools for real-time modular audio processing and dynamic routing. While traditional DAWs like Ableton Live and Logic Pro offer extensive features, they require significant system resources, installations, and are often prohibitively expensive. Similarly, tools like MaxMSP and Pure Data are powerful but can present steep learning curves, limiting their accessibility to users with advanced DSP programming knowledge. This creates a significant barrier for beginners, educators, and those without access to high-performance hardware or expensive software licenses. Table 1 compares the features of traditional DAWs, modular tools, and Nodeaa.

2.1 Accessibility Challenges in Audio Processing

The accessibility of audio tools is constrained by both technical and financial factors. High-performance DAWs often require powerful hardware and advanced knowledge, placing them out of reach for many users. Educational institutions and hobbyists in resource-limited regions face additional hurdles due to the cost and complexity of these tools. This disparity hinders creativity and innovation by excluding a large portion of potential users who lack the means to experiment with digital audio.

2.2 Democratizing Audio Tools Through the Web

By leveraging the browser as a platform, *Nodeaa* addresses these challenges head-on. Users can access *Nodeaa* from any modern desktop device with a web browser, eliminating the need for installations or specialized hardware. This approach significantly reduces entry barriers and allows musicians, educators, and hobbyists to explore modular audio processing without requiring advanced technical expertise [15]. The web-based nature of *Nodeaa* ensures cross-platform compatibility, reaching users regardless of their operating system or device specifications.

3 Technical Background

To understand the development and functionality of *Nodeaa*, it is essential to detail the underlying technologies and concepts that enable it. *Nodeaa* combines principles of modular audio processing, node-based dynamic routing, and modern web development frameworks. Below, we introduce the key technical foundations necessary for this project.

Table 1: Comparison of Traditional DAWs, Modular Tools, and Nodeaa

Feature/Aspect	**Traditional DAWs**	**Modular Tools (e.g., Max/MSP)**	**Nodeaa**
Platform Dependency	High	High	Low
Cost	Expensive	Expensive (or limited free options)	Free and Open Source
Installation Required	Yes	Yes	No (Browser-Based)
Hardware Requirements	High	Moderate to High	Low
Ease of Use for Beginners	Moderate	Low (Steep Learning Curve)	High
Flexibility and Customization	High	Very High	High
Cross-Platform Accessibility	Limited (OS-Specific)	Limited (OS-Specific)	Universal (Web-Based)
Real-Time Audio Processing	Yes	Yes	Yes
Community and Educational Use	Limited (Cost Barrier)	Limited (Complexity Barrier)	Broad
Extensibility via Code or Nodes	Limited (Plugins)	High	High

3.1 WebAudio API

The WebAudio API is a high-level JavaScript API for processing and synthesizing audio in web applications. It provides a graph-based model where audio processing is performed by connecting various *AudioNodes*, such as sources, effects, and output nodes. Each *AudioNode* represents a single processing module, and developers can dynamically create and route connections between nodes to build complex audio graphs [21]. The API also supports features like real-time audio processing, spatial audio, and playback control, making it a robust tool for browser-based audio applications.

3.2 Node-Based Interfaces

Node-based interfaces represent data or processes as interconnected nodes in a visual graph [2]. Each node performs a specific function, such as generating sound, applying an effect, or routing data. Users can intuitively connect nodes to define the flow of data or signals, enabling modular and flexible workflows. This pattern is widely used in audio programming environments such as MaxMSP, Pure Data, and VCV Rack, and *Nodeaa* brings this approach to the browser.

3.3 WebAssembly (Wasm)

WebAssembly is a low-level binary instruction format designed for efficient execution in web browsers [9]. It enables near-native performance for computationally intensive tasks by compiling code written in languages like C++ and Faust into a format that the browser can execute. In *Nodeaa*, WebAssembly is used to load custom audio nodes compiled from Faust and RNBO code, allowing developers to extend the functionality of the WebAudio API with high-performance, user-defined modules.

3.4 React and React Flow

React is a popular JavaScript library for building user interfaces [7]. It enables developers to create reusable components that manage their own state, simplifying the development of complex web applications. React Flow is a library built on React for creating interactive node-based interfaces. It provides the core functionality needed for rendering, connecting, and managing nodes and edges in *Nodeaa*, allowing users to visually construct and manipulate audio graphs.

3.5 Audio Signal Flow

Understanding audio signal flow is critical for working with *Nodeaa*. Audio signals are continuous streams of data representing sound waves, which are processed in real time by connecting input, processing, and output nodes [19] [23]. For instance, in a simple setup, a microphone's audio input is routed to a delay effect node and then to the speakers. Signal processing involves modifying these streams through operations like filtering, amplification, and spatialization to achieve desired audio effects.

3.6 Faust and RNBO

Faust (Functional Audio Stream) and RNBO (a node-based language associated with MaxMSP) are key technologies enabling the features of *Nodeaa*.

Faust is an open-source, domain-specific functional programming language for real-time audio signal processing. It allows developers to create DSP algorithms with a high-level syntax that can be compiled into multiple formats, including WebAssembly, making it an ideal choice for integrating custom audio nodes into browser-based platforms like *Nodeaa* [16].

Similarly, RNBO provides a low-level, node-based environment for designing audio processing and synthesis workflows. It supports exporting these workflows to WebAssem-

bly, enabling seamless integration with web audio applications [10].

By combining these technologies, *Nodeaa* bridges the gap between traditional DAWs and web-based tools, offering a modular environment for sound processing directly in the browser. This integration highlights the potential of modern web technologies to deliver advanced audio synthesis and processing capabilities traditionally reserved for desktop applications.

4 Prior Work

The development of *Nodeaa* draws heavily on concepts and techniques from existing tools and frameworks in digital audio processing, each contributing unique approaches to sound synthesis and manipulation.

4.1 Faust in Digital Audio Processing

Faust has established itself as a powerful functional programming language for real-time audio signal processing and synthesis. Its ability to compile high-level functional code into efficient low-level implementations makes it highly versatile, supporting deployment on platforms ranging from embedded systems to web applications [16]. Faust's integration into *Nodeaa* demonstrates how its modular, high-level syntax can be leveraged to create advanced audio effects and instruments tailored for modern web environments [14, 13].

4.2 Max/MSP and Pure Data

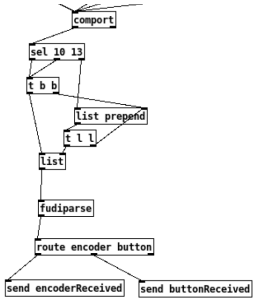

Figure 2: An example DSP chain in Pure Data

Max/MSP and Pure Data (Pd) are visual programming environments that promote the creation of interactive multimedia applications [18]. They employ a node-based interface where users connect objects representing different functions to design complex audio and visual processes. This modular approach provides significant flexibility and customization, enabling users to develop intricate signal processing chains and interactive installations. However, the depth of customization available can present a steep learning curve for newcomers, potentially limiting accessibility for those without prior experience in audio programming.

4.3 Bitwig Studio's The Grid

Figure 3: Another node based audio routing program, The Grid in Bitwig Studio

Bitwig Studio's The Grid is a modular sound design environment integrated within the Bitwig digital audio workstation [3]. It offers a high-level, guided interface for creating audio effects and instruments, emphasizing user-friendly interaction and a simple workflow. While The Grid simplifies the process of sound design, its high-level nature may impose certain limitations on the depth of customization and flexibility compared to more low-level programming environments like FAUST or Max/MSP.

4.4 Influence on *Nodeaa*

The design and development of *Nodeaa* are informed by the strengths and limitations of these existing tools. By integrating the flexibility of node-based audio routing, as exemplified by Max/MSP and Pure Data, with the accessibility of a browser-based platform, *Nodeaa* aims to provide a balanced environment for both novice and experienced users. Additionally, by compiling FAUST code into WebAssembly (Wasm), *Nodeaa* leverages efficient, low-level audio processing capabilities within the browser context, bridging the gap between high performance and user-friendly interaction.

5 Methods

5.1 Project Structure and Development Workflow

The development of *Nodeaa* follows a streamlined Continuous Integration/Continuous Deployment (CI/CD) workflow utilizing GitHub Actions [4]. This ensures that every change is automatically tested and deployed to the live

site, enabling rapid iteration and consistent delivery. Each change is subjected to automated checks for performance and building before deployment. By eliminating manual deployment steps, the workflow reduces human error and speeds up the development cycle. This approach ensures that users consistently experience a reliable and up-to-date application.

5.2 Integration of Faust and RNBO for Custom Audio Nodes

A key component of *Nodeaa*'s functionality is its ability to integrate custom audio nodes compiled from Faust and RNBO code [10] [14] [13]. These nodes implement custom algorithms tailored for specific audio processing tasks, such as delays, synthesizers, and filters. The compilation process transforms the Faust and RNBO scripts into WebAssembly (WASM) modules, which are then added to the public directory of the web app for direct access by the browser. To enable user interaction with these modules, a TypeScript interface is created to manage their parameters dynamically, allowing seamless control and real-time adjustments. Additionally, an entry is added to the master JSON file to define the node's metadata, including its name, default parameters, audio type, public URL, and version. This modular approach simplifies the addition of new nodes and ensures compatibility within the existing ecosystem.

5.3 Graphical Interface and Node Management

The node-based interface is built using React Flow, which is extensively customized to enhance its functionality and usability. React Flow provides a robust foundation for visualizing and managing node connections, but *Nodeaa* extends its capabilities by incorporating state persistence through Zustand. This allows users to save and reload projects seamlessly, preserving all node configurations and connections. Visual clarity is a priority, and connection wires are color-coded to indicate the type of data being transmitted: green-striped for audio, blue for MIDI, and grey for general data. Such visual distinctions help users quickly understand the routing within their projects. While alternative libraries were evaluated, they often relied on `<canvas>` elements, which would have shifted focus to graphics programming and increased complexity. By leveraging React Flow, *Nodeaa* maintains an intuitive interface that prioritizes functionality and accessibility.

5.4 Latency Management and Real-Time Performance

Low latency is critical for live audio performance, and *Nodeaa* is designed to achieve this without overburdening the system. The application introduces slight latencies

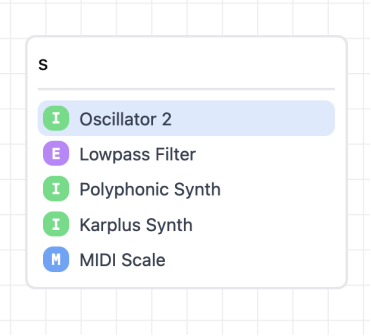

Figure 4: The node creator created by pressing "n" on the keyboard. Allows users to quickly create nodes by typing and getting instant results.

(1ms) to clear the audio thread and prevent overloading, ensuring smooth playback even under demanding conditions [1]. Each node operates independently in the audio thread, leveraging the WebAudio API to manage audio connections and processing [11]. This decentralized state management architecture minimizes inter-node communication overhead, allowing the system to scale efficiently. Additionally, the application prioritizes real-time audio performance by optimizing its scheduling, ensuring that user actions translate to immediate auditory feedback without noticeable delays.

5.5 Design Decisions

The design of *Nodeaa* draws inspiration from industry-standard audio software such as Ableton Live and MaxMSP. Key user interface elements, including knobs and sliders, are modeled to emulate the tactile experience of professional audio equipment [8]. These abstract components are designed for both aesthetic appeal and functional clarity, enhancing the overall user experience for users. The connection process between nodes mimics MaxMSP's intuitive drag-and-drop system, simplifying the creation of complex audio routing configurations. By adopting these proven design principles, *Nodeaa* lowers the learning curve for new users while maintaining the flexibility and power required by experienced professionals.

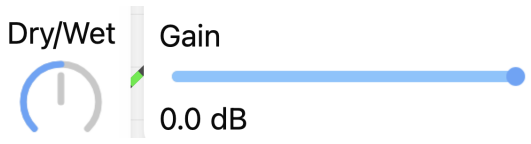

Figure 5: Example of two custom Nodeaa interface components, Knob and Slider.

5.6 Developer Tools and Components

During development, *Nodeaa* introduced a selection of tools to support extensibility and customization. These tools include the aforementioned versatile and abstract UI components such as knobs, sliders, and number boxes, all created using p5.js canvas elements. By optimizing these elements to redraw only during user interaction, the project achieves efficient performance while maintaining visual fidelity. These modular components can be easily integrated into custom nodes or applications, providing developers with flexible building blocks for their projects.

The project also implemented foundational components for creating node shapes, such as a container and a draggable title bar, allowing for the design of cohesive and interactive nodes. Custom React hooks were developed to listen for specific event data, including MIDI input, enabling dynamic and responsive audio workflows with minimal boilerplate code. While the tools are limited due to time constraints, they establish a foundation for future expansion.

5.7 Context-Aware Information Display

Inspired by Ableton Live [20], *Nodeaa* incorporates a dynamic information box located in the bottom left corner of the interface. This feature provides real-time contextual documentation about the parameter or device that the user's mouse is currently hovering over. For example, hovering over a knob displays its function, parent device, and an explanation of its effect, while hovering over a device provides a brief description of its overall purpose and usage.

This design choice ensures that users have immediate access to relevant information without disrupting their workflow. Beginners benefit from reduced reliance on external documentation, while experienced users can quickly reference details about less frequently used features. By embedding this intuitive guidance directly into the interface, *Nodeaa* fosters a more seamless and efficient user experience, aligning with its goal of being accessible to users of all skill levels.

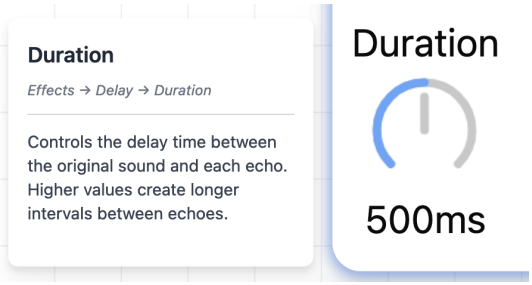

Figure 6: The info panel seen here showing the description for the delay duration knob on the delay effect.

5.8 Limitations and Alternatives

While the browser-based implementation of *Nodeaa* addresses accessibility and ease of use, it is not without limitations. JavaScript's single-threaded nature imposes constraints on performance [22], particularly when handling large projects with over 1,000 nodes [5]. Such scenarios can introduce graphics related paint lag, and tasks like audio parameter updates may require more time compared to native C++ desktop applications. A desktop application was considered as an alternative, but this approach would have negated the primary goal of eliminating installation barriers and providing a truly cross-platform solution. Despite these challenges, *Nodeaa* remains a highly capable tool for live audio performance and modular sound processing, offering a unique blend of accessibility and functionality.

6 Evaluation Metrics

The success of *Nodeaa* is evaluated based on its ability to deliver real-time, low-latency audio processing, an intuitive user experience for both beginners and advanced users, and compatibility across major desktop browsers. Metrics were chosen to measure the system's performance, usability, and creative potential, ensuring the site meets the needs of a broad user base.

6.1 Performance Metrics

Real-time performance was assessed by observing audio playback quality and responsiveness. Key indicators included the absence of audio glitches, such as clicks and pops, and smooth operation without perceptible lag, even under complex routing scenarios. These metrics were tested under varying levels of system load to ensure the tool remained stable and responsive in real-world use cases. Additionally, the performance of custom audio nodes, compiled from Faust and RNBO to WebAssembly, was evaluated to confirm that they integrated seamlessly into the broader audio graph. Browser compatibility was validated across major desktop browsers, including Chrome, Firefox, and Brave, ensuring users could reliably access the application regardless of their preferred platform.

6.2 Usability and Accessibility

Usability was evaluated through two rounds of user testing with two different versions of Nodeaa. Feedback was gathered via structured forms focusing on ease of use, clarity of the interface, and the tool's ability to inspire creativity. During testing, users were tasked with completing specific workflows, such as routing a microphone input through an

effect node and recording the output, to assess the intuitiveness of the interface. Metrics included the time required to understand and complete these tasks, the clarity of visual feedback provided by the node-based interface, and subjective ratings of the tool's overall usability. Additionally, the design was tested for accessibility features, ensuring that users with different levels of technical expertise could engage with the software effectively.

6.3 Creative Potential

To assess creative potential, users were asked whether *Nodeaa* supported their ability to create music or perform live. Specific feedback was requested on the extent to which the current feature set enabled meaningful exploration and experimentation. The metrics also included evaluations of the variety and flexibility of available nodes, as well as the ease of combining them to achieve unique audio effects. Insights were drawn from user feedback to identify potential areas for future feature development, such as additional node types or advanced customization options. Importantly, the evaluation aimed to determine whether users found the tool inspiring and versatile enough to include in their personal or professional projects.

6.4 Exclusions

While mobile compatibility was considered, it was ultimately excluded as a primary metric due to the project's focus on desktop environments. Mobile platforms often present unique challenges [6], such as reduced processing power and limited screen real estate, which were beyond the scope of this project. Similarly, analysis with existing DAWs was de-prioritized in favor of metrics that evaluate *Nodeaa*'s standalone utility. This decision reflects the focus on the unique contributions of *Nodeaa*, particularly its browser-based architecture and node-based interface, rather than positioning it as a direct competitor to traditional DAWs.

These metrics provide a comprehensive framework for determining the success of *Nodeaa* in meeting its goals and addressing the identified problem.

7 Results and Discussion

The results of the evaluation indicate that *Nodeaa* has successfully addressed many of its intended goals. By analyzing user feedback and metrics, the tool has demonstrated strong performance, usability, and creative potential. Below, the key findings are summarized and discussed in the context of the project's objectives.

7.1 Performance Metrics

One of the primary objectives of *Nodeaa* was to ensure real-time, low-latency audio performance suitable for live use. According to user responses, *Nodeaa* met this goal effectively:

- 100% of users reported no audio glitches such as clicks or pops during their sessions.
- 89.5% of users found audio playback to be consistently responsive, with low latency. Only 10.5% noted occasional lag, primarily when interacting with the interface.
- The application remained stable in complex routing scenarios, as confirmed by 100% of users.

While some lag was reported during interface interactions [5], the system's core audio performance was robust. Issues in Safari were noted by 15.8% of users, suggesting the need for further optimization for non-Chromium-based browsers.

7.2 Usability and Accessibility

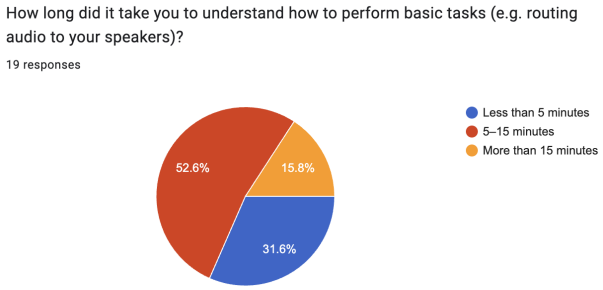

Figure 7: A graph showing the responses to time spent learning Nodeaa for the first time.

Another critical goal was to create a tool that is accessible to both beginners and advanced users. Feedback on the interface highlighted both strengths and areas for improvement:

- 42.1% of users rated the interface as a 3 out of 5 for intuitiveness, while 42.1% gave it a 4, and 15.8% rated it a 5. This indicates a generally positive reception but also room for refinement.
- Most users (52.6%) were able to understand basic tasks within 5–15 minutes, demonstrating the tool's approachability. However, 15.8% required more than 15 minutes.
- 94.7% of users agreed that the interface provided clear feedback, though minor issues were reported, such as a bug requiring multiple clicks to delete a node and the ability to create feedback loops by connecting nodes to themselves.

- 57.9% of users encountered barriers related to their expertise. Common challenges included unfamiliar terms (e.g., ”karplus”) and confusion about specific concepts like MIDI devices. Suggestions for tutorial videos or in-app guidance were frequently mentioned.

These findings emphasize the need for enhanced onboarding resources and clearer terminology to improve accessibility for beginners.

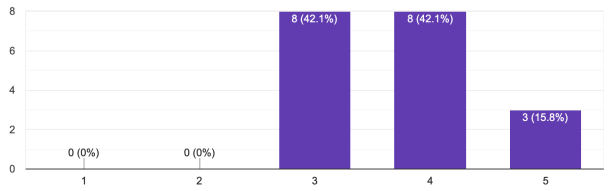

Figure 8: A graph showing the responses to ”How intuitive did you find the node-based interface?” on a scale of 1-5 (5 being very intuitive).

7.3 Creative Potential

Nodeaa excelled in enabling users to experiment and create:

- 100% of users felt they could experiment easily with the current feature set.
- 68.4% of users found the available nodes to somewhat support their creative goals, while 31.6% felt they fully supported them. No users reported dissatisfaction.
- 94.7% of users envisioned using *Nodeaa* in future projects, underscoring its potential as a practical and inspiring tool.

User feedback frequently highlighted the desire for additional effects, instruments, and advanced nodes to expand the creative possibilities.

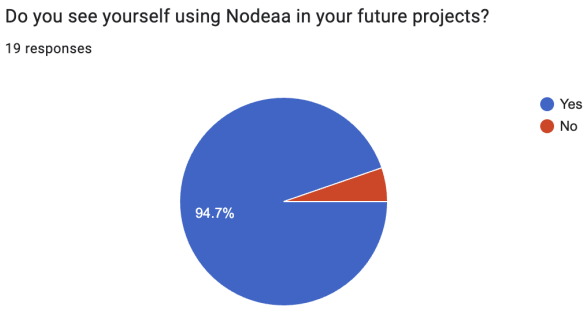

Figure 9: A graph showing the responses to using Nodeaa in future projects.

7.4 Caveats and Alternate Explanations

While the results are largely positive, some caveats should be considered:

- The sample size of advanced users was limited (5.3%), which may underrepresent the needs of highly experienced users.
- Browser compatibility issues in Safari indicate that further testing and optimization are necessary to ensure cross-browser functionality.
- The reported usability barriers, while expected for a modular tool, suggest a need for improved documentation and in-app tutorials to bridge the gap for novice users.

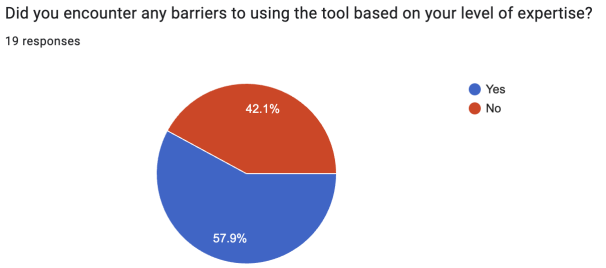

Figure 10: A graph showing the responses to barrier based on skill level.

7.5 Summary

Overall, *Nodeaa* successfully meets its primary objectives of delivering real-time, low-latency audio performance and fostering creativity through a browser-based platform. By addressing the identified usability challenges and expanding the feature set, *Nodeaa* has the potential to further solidify its role as an innovative and accessible digital audio workstation.

The raw data and user feedback collected during testing are available in the project repository for further reference: `https://github.com/a200xeaf/nodeaa2`.

8 Ethical Considerations

Developing and deploying *Nodeaa* involves several ethical considerations, both technological and biological. These issues highlight the responsibility of developers to create tools that are safe and equitable.

8.1 Hearing Safety and Equipment Protection

One of the primary ethical concerns in audio software development is ensuring user safety by preventing excessively

loud sounds, clicks, or pops, which can damage both hearing and audio equipment [24] [17]. *Nodeaa* mitigates these risks by:

- Ensuring that audio nodes are designed with safeguards against extreme volume levels, such as default gain limits.
- Feedback loop prevention in the audio graph.
- Testing to ensure that the custom nodes real-time processing does not introduce unintended audio artifacts under normal operation.
- A clipping limiter placed on the output to prevent audio that exceeds -1.0 - 1.0 amplitude range.

Future iterations of *Nodeaa* will explore real-time peak monitoring to further enhance safety measures and smoother output limiting options.

8.2 Intellectual Property and Copyright Concerns

As *Nodeaa* evolves, particularly with the potential for future versions to allow sharing, uploading, or recording of user-generated projects, intellectual property (IP) and copyright issues may arise. For example:

- Users may inadvertently or intentionally include copyrighted audio samples or MIDI sequences in their projects.
- Distributed projects could be used to bypass copyright protections, raising potential legal and ethical challenges.

To address these concerns, the development roadmap includes plans for:

- Implementing basic content analysis tools [12] that identify copyrighted material before projects are shared or distributed.
- Providing clear terms of use that emphasize users' responsibility for ensuring compliance with copyright laws.

These measures aim to balance the creative freedom *Nodeaa* offers with a commitment to respecting the rights of content creators.

8.3 Accessibility and Inclusivity

While *Nodeaa* strives to be an accessible tool for users across skill levels, ethical considerations regarding diversity and inclusivity remain:

- The interface and terminology may pose barriers to non-technical users or individuals unfamiliar with modular audio systems. To address this, future updates will begin work on localized language.
- Browser-based tools require reliable internet and modern hardware, which may inadvertently exclude users in regions with limited technological access. This highlights the need for further exploration of offline functionality and optimization for lower-spec devices.

8.4 Conclusion

Ethical considerations are important to the development of *Nodeaa*. By addressing safety, IP concerns, inclusivity, and safety, the site aims to create a responsible and impactful tool that is useful to users without compromising ethical standards. Future development will continue to prioritize these concerns, ensuring that *Nodeaa* remains a safe platform for creative expression.

A Replication Instructions

To replicate and use *Nodeaa*, follow these instructions carefully. These steps ensure compatibility and reproducibility of the project setup.

A.1 Prerequisites

Before starting, ensure that the following software is installed:

- Node.js (version 16.0.0 or higher)
- npm (version 7.0.0 or higher)
- A modern web browser (such as Google Chrome or Mozilla Firefox)

For consistent results, it is recommended to use the specified versions or later.

A.2 Cloning the Repository

Begin by cloning the project repository: `git clone https://github.com/a200xeaf/nodeaa2.git` and navigate to the directory using `cd nodeaa2`.

A.3 Installing Dependencies

Install the necessary packages using npm:

```
npm install
```

This command will fetch and install all dependencies specified in the `package.json` file. Ensure that your internet connection is stable during this step.

A.4 Running the Development Server

Start the development server to run *Nodeaa* locally:

```
npm run dev
```

Open your browser and navigate to `http://localhost:5173` to access the application.

A.5 Building for Production

To create an optimized build for production, execute the following command:

```
npm run build
```

The production-ready files will be placed in the `dist` directory.

A.6 Previewing the Production Build

To locally preview the production build, use the command:

```
npm run preview
```

This will serve the application using the optimized build.

A.7 Dependencies and Future-Proofing

The project uses the following major dependencies:

- React (version 18.0 or higher)
- TypeScript (version 5.0 or higher)
- Vite (version 5.0 or higher)
- RNBO (version 1.3.2 or higher)
- faustwasm (version 0.6.4 or higher)

For future-proofing, regularly update dependencies using `npm update`, but ensure that changes do not introduce breaking issues. Locking dependency versions in the `package.json` file helps maintain consistency.

A.8 Survey Results and Documentation

The repository includes the following additional files for reference:

- `LATEXSOURCE/document.tex`: The full LaTeX source of this project paper.
- `SURVEYRESULTS/`: A folder containing raw survey data.
- `LICENSE`: The project license (AGPL v3).

A.9 Contact for Issues

For any issues or questions, consult the project repository's `README.md` or open an issue on the GitHub repository `https://github.com/a200xeaf/nodeaa2`.

B Code Architecture Overview

This section provides an overview of *Nodeaa*'s code architecture and instructions for extending the project, with a focus on adding new nodes and utilizing core features such as the store and audio engine.

B.1 Folder Structure

The project is organized as follows:

- `src/engine`: Contains the core logic for the application, including `audio.ts`, which serves as the main engine for audio processing, and `store.ts`, which manages global application state using a reactive store architecture.
- `src/nodes`: Contains all node types grouped by their category (e.g., MIDI, Audio, Data).
- `public/nodes`: Stores precompiled WebAssembly (Wasm) and metadata files for Faust and RNBO nodes.
- `src/ui`: Houses reusable UI components such as sliders, knobs, and containers.

These folders work together to support the modular and extensible nature of *Nodeaa*.

B.2 Adding a New Node

Adding a new node to *Nodeaa* involves the following steps:

B.2.1 Step 1: Compile to Wasm (If Applicable)

If your node uses Faust or RNBO, compile your source to WebAssembly (Wasm):

- For Faust, ensure the output includes `dsp-module.wasm` and `dsp-module.json`.
- For RNBO, include `(name).export.json` and `dependencies.json`.
- Place these files into a new folder under `public/nodes/`. Do not rename the files.
- Skip this step if creating non-audio nodes, such as MIDI devices.

B.2.2 Step 2: Create the Node Component

Create a new folder under `src/nodes/(type)/` and add a `.tsx` file representing the visual and functional implementation of the node.

- Use `FC<NodeProps>` or `FC<NodeProps<YourNodeType>>` to define the component if it stores data.
- Define the node's data structure. For example:

```
type MidiInNodeData = {
    midiin_device: string;
};

type MidiInNodeType = Node<MidiInNodeData, '
    midiInNode'>;
```

- Use hooks like `useNodeStore` to update node parameters:

```
const updateNode = useNodeStore((state) =>
    state.updateNode);
// Usage: updateNode(id, { key: value });
```

- Define inlets and outlets with `<Handle />`, referencing React Flow documentation for options.

B.2.3 Step 3: Update the Master JSON File

Add an entry for the new node to `nodes.json` using the following interface (Note: `"type": "name"` inside `"audioNodeParams"` should match the folder name in the public/nodes directory):

```
export interface NodeConfig {
    nodeName: string;
    idPrefix: string;
    realName: string;
    defaultData: Record<string, unknown>;
    audioNodeParams?: {
        engine: "faust" | "rnbo";
        type: string;
        voices?: number;
    };
    hasAudio: boolean;
    audioType?: "instrument" | "effect";
    version: string;
};
```

For example:

```
"faustGainNode": {
    "nodeName": "faustGainNode",
    "idPrefix": "",
    "realName": "Gain",
    "defaultData": { "faustgain_Gain": 1 },
    "audioNodeParams": { "engine": "faust", "type
        ": "faustgain" },
    "hasAudio": true,
    "audioType": "effect",
    "version": "1.0.0"
},
```

B.2.4 Step 4: Register the Node and Add Documentation

- Add the node to `nodeTypes` in `App.tsx`.
- Optionally, add hover-over documentation by updating `info-map.ts`:

```
["midiInNode-check", {
    name: "Check Midi",
    parent: "MIDI In",
    type: "MIDI",
    description: "Checks for connected MIDI
        devices. Requires browser permission
        to work."
}]
```

Use the `data-info-panel-id` attribute in your component for integration.

B.3 UI Components and Hooks

Nodeaa provides several reusable components to streamline development:

- `<Knob />`, `<Numbox />`, `<Slider />`, `<PianoKeyboard />`: Prebuilt UI elements for user interaction.
- `<NodeaaContainer />`, `<NodeaaHeader />`: Structural components for nodes.

Custom hooks available include:

- `useMidiNoteTracker`: Tracks active MIDI notes and processes MIDI events.
- `useEmitterSubscriptions`: Handles event subscriptions for connections.

These tools simplify extending the application and integrating new features seamlessly.

References

[1] Buffa, Michel and Demont, Samuel. "Can you DAW it Online?" In: *IEEE International Symposium on the Internet of Sounds 2024, 1st IEEE International Workshop on the Musical Metaverse (IEEE IWMM)*. 2024.

[2] Burnett, Margaret M and McIntyre, David W. "Visual programming". In: *COmputer-Los Alamitos-* 28 (1995), pp. 14–14.

[3] Burt, Warren. "A new ecosystem for microtonal computer music exploration and composition". In: *Chroma: Journal of the Australasian Computer Music Association* 39.2 (2023).

[4] Chandrasekara, Chaminda et al. "Introduction to github actions". In: *Hands-on GitHub actions: implement CI/CD with GitHub action workflows for your applications* (2021), pp. 1–8.

[5] Chec, Dariusz and Nowak, Ziemowit. "The performance analysis of web applications based on virtual DOM and reactive user interfaces". In: *Engineering Software Systems: Research and Praxis*. Springer. 2019, pp. 119–134.

[6] Clauhs, Matthew and Dozoretz, Brian. "The DAW revolution". In: *The Routledge companion to creativities in music education* (2022), pp. 217–227.

[7] Fedosejev, Artemij. *React. js essentials*. Packt Publishing Ltd, 2015.

[8] Gómez, Dániel and Sánchez, David. "Usage of adaptive features in sound synthesis GUIs". In: *2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS)*. IEEE. 2011, pp. 1–4.

[9] Haas, Andreas et al. "Bringing the web up to speed with WebAssembly". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, pp. 185–200.

[10] Kilius, Sarunas. "Web Integration of a granular synthesizer using RNBO in Max/MSP". In: (2015).

[11] Kleimola, Jari and Larkin, Oliver. "Web audio modules". In: *Proc. 12th Sound and Music Computing Conference*. 2015.

[12] Lalinský, Lukáš. *AcoustID: Open Source Audio Identification*. Accessed: 2024-12-11. 2024. URL: https://acoustid.org/.

[13] Letz, Stephane et al. "Faust audio DSP language in the Web". In: *Linux Audio Conference*. 2015, pp. 29–36.

[14] Letz, Stéphane, Orlarey, Yann, and Fober, Dominique. "Compiling Faust audio DSP code to WebAssembly". In: *Web Audio Conference*. 2017.

[15] Leyshon, Andrew. *Reformatted: Code, networks, and the transformation of the music industry*. Oxford University Press, 2014.

[16] Orlarey, Yann, Fober, Dominique, and Letz, Stéphane. "FAUST: an efficient functional approach to DSP programming". In: *New Computational Paradigms for Computer Music* (2009), pp. 65–96.

[17] Pienkowski, Martin. "Loud music and leisure noise is a common cause of chronic hearing loss, tinnitus and hyperacusis". In: *International journal of environmental research and public health* 18.8 (2021), p. 4236.

[18] Puckette, Miller et al. "Pure Data: another integrated computer music environment". In: *Proceedings of the second intercollege computer music concerts* (1996), pp. 37–41.

[19] Roads, Curtis. *Computer Music Tutorial*. Vol. 170. MIT Press, 1996.

[20] Robinson, Keith. *Ableton Live 9*. Routledge, 2014.

[21] Smus, Boris. *Web audio API: advanced sound for games and interactive apps*. " O'Reilly Media, Inc.", 2013.

[22] Verdú, Javier and Pajuelo, Alex. "Performance scalability analysis of javascript applications with web workers". In: *IEEE Computer Architecture Letters* 15.2 (2015), pp. 105–108.

[23] Wakefield, G. and Taylor, G. *Generating Sound & Organizing Time: Thinking with Gen˜ Book 1*. bk. 1. Cycling '74, 2022. ISBN: 9781732590311. URL: https://books.google.com/books?id=yvV4zwEACAAJ.

[24] Zhao, Fei et al. "Music exposure and hearing disorders: an overview". In: *International journal of audiology* 49.1 (2010), pp. 54–64.