Csound Web-IDE

Steven Yi
Rochester Institute of Technology
syyigm@rit.edu

Hlöðver Sigurðsson
Berlin, Germany
hlolli@gmail.com

Edward Costello
Music Department
Maynooth University, Ireland
edward.costello@mu.ie

ABSTRACT

Csound Web-IDE[1] is an open-source[2], browser-based integrated development environment (IDE) for sound and music computing using Csound[4]. The web application offers users the ability to edit and run standard, multi-file Csound projects in the same way they would do on the desktop, mobile, and embedded platforms. Enabled by modern web technologies, envisioned use cases for the Web-IDE include computer music education, music composition, and development of realtime interactive systems, as well future integration with other Web Audio-based systems.

Keywords

Csound, Web IDE, Web Audio, Web MIDI

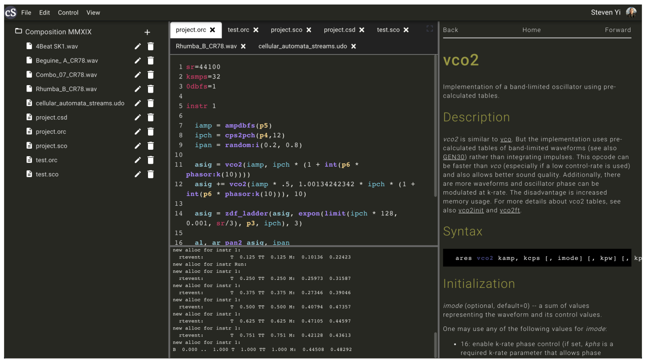
1. INTRODUCTION

As the capabilities of the modern web platform continue to grow, so too does the ability to develop more complex multimedia software applications once only possible on traditional native platforms. Modern web APIs such as the Web Audio API and the WebAssembly runtime make it possible to create, or port existing libraries to build complex audio processing and music composition software systems. Also, the increasing availability of mature UI frameworks built on top of existing web technologies, such as Angular and React, have also greatly streamlined the development of elaborate and responsive user interfaces.

Additionally, the web platform coupled with cloud based database/application platform services such as Google's Firebase[3] also offers a robust means of application distribution, document storage and online collaboration tools. This confluence of technologies has allowed us to develop a fully featured IDE–called *Csound Web-IDE*, shown in Figure 1–for the Csound language which in many ways extends the capabilities of previous Csound-based environments by leveraging the unique power of the web-based software stack.

[1]https://ide.csound.com
[2]https://github.com/csound/web-ide
[3]https://firebase.google.com/

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.



```
sr=44100
ksmps=32
0dbfs=1

instr 1

  iamp = ampdbfs(p5)
  ipch = cps2pch(p4,12)
  ipan = random:i(0.2, 0.8)

  asig = vco2(iamp, ipch * (1 + int(p6 *
phasor:k(10))))
  asig += vco2(iamp * .5, 1.00134242342 * ipch * (1 +
int(p6 * phasor:k(10))), 10)

  asig = zdf_ladder(asig, expon(limit(ipch * 128,
0.001, sr/3), p3, ipch), 3)

  al, ar pan2 asig, ipan
```

```
new alloc for instr 1:
  rtevent:        T  0.125 TT  0.125 M:  0.10136  0.22423
new alloc for instr Run:
new alloc for instr 1:
  rtevent:        T  0.250 TT  0.250 M:  0.25973  0.31587
new alloc for instr 1:
  rtevent:        T  0.375 TT  0.375 M:  0.27346  0.39046
new alloc for instr 1:
  rtevent:        T  0.500 TT  0.500 M:  0.40794  0.47357
new alloc for instr 1:
  rtevent:        T  0.625 TT  0.625 M:  0.47105  0.44597
new alloc for instr 1:
  rtevent:        T  0.751 TT  0.751 M:  0.42128  0.43613
new alloc for instr 1:
B  0.000 ..  1.000 T  1.000 TT  1.000 M:  0.44508  0.48292
```

vco2

Implementation of a band-limited oscillator using pre-calculated tables.

Description

vco2 is similar to vco. But the implementation uses pre-calculated tables of band-limited waveforms (see also GEN30) rather than integrating impulses. This opcode can be faster than *vco* (especially if a low control-rate is used) and also allows better sound quality. Additionally, there are more waveforms and oscillator phase can be modulated at k-rate. The disadvantage is increased memory usage. For more details about vco2 tables, see also vco2init and vco2ft.

Syntax

```
ares vco2 kamp, kcps [, imode] [, kpw] [, kp
```

Initialization

imode (optional, default=0) -- a sum of values representing the waveform and its control values.

One may use any of the following values for *imode*:

- 16: enable k-rate phase control (if set, *kphs* is a required k-rate parameter that allows phase

Figure 1: Primary Csound Web-IDE interface showing project file-tree and code editor.

Employing the Web Audio Csound library as the basis of the IDE enables users to be able to write, compile, and run Csound compositions on any computer with a modern web browser without installing any additional software or libraries. It also allows for the controlling of Csound-based synthesisers using the Web MIDI API and to continually recompile and update running Csound code via live coding. Users can also store their Csound instruments and compositions online giving them the ability to share and collaborate on documents with other participants. These capabilities greatly reduce any friction involved with using the Csound language and also potentially creates an environment which will allow users to easily distribute, incorporate into their own projects, and learn from an online library of Csound based instruments, effects and compositions.

2. RELATED WORK

Traditional desktop IDEs, such as Visual Studio[4] and XCode[5], operate within the context of their platforms where projects are generally made up of files that are organized into directories that all exist on a filesystem. IDEs like these provide development experiences where users may edit, compile, run, and debug projects all within a single application. Although the IDE may provide a unified interface to the user, they often depend upon secondary programs (e.g., compilers, linkers, debuggers) to perform operations.

Web IDEs operate much like their desktop counterparts but are developed using browser technologies. They fea-

[4]https://visualstudio.microsoft.com/
[5]https://developer.apple.com/xcode/

ture similar offerings of a unified interface for code-based projects and depend upon either server-side or client-side tools (e.g., compilers, audio engines) for executing those projects. Web IDEs offer additional features unique compared to desktop IDEs in that they are delivered over the network with zero-installation required, making them easy to deploy to end users (assuming they have a compatible browser). They also interoperate easily with servers to organize and persist projects for easy access by users across multiple systems. While recent developments of Web Audio and Web MIDI standards have allowed browser-based music applications to approach the level of features and performance found in desktop platforms, they are not quite as robust and performant as of yet. However, web applications on the browser platform offer solutions to satisfy unique use cases that traditional desktop IDEs can not do so easily or at all.

Within the Csound ecosystem, for Csound Web-IDE, we seek to create a comparable experience to the desktop CsoundQt[6] and Cabbage[7] applications. CsoundQt is a cross-platform IDE that works with projects on the filesystem, offers syntax-higlighting and code completion in its code editors for Csound code, uses Csound for rendering to disk and in realtime, and offers custom audio and MIDI I/O using audio and MIDI systems native to the desktop platforms. CsoundQt also offers tools such as audio scopes, UI widgets, and other features to provide a single working environment for Csound users. For Csound Web-IDE, the use cases and feature set of CsoundQt are the closest to what we are targetting for our initial phases of development.

Cabbage is a framework for audio software development that provides an IDE capable of running standalone as well as exporting binary audio plugins for VST and AU formats. Cabbage employs Csound as its audio engine via Csound's API and works within its target context, either generating audio directly to the desktop system's audio system or generating and processing signals within the host VST or AU API. Cabbage's primary standalone IDE features largely overlap with CsoundQt, though its support of industry plugin formats is unique in the Csound ecosystem. While supporting binary plugin generation is not a concern at this time for the Csound Web-IDE, we expect to have continued discussions with Cabbage's author in the future to see where compatibility with Cabbage's feature set and projects can be achieved.

Looking at web-based audio IDEs, the Faust Online Compiler[5][8] (FOC) provides a browser-based web application for editing, compiling, and running Faust code. Originally a server-based solution where the compiler would run on the server-side, the system evolved to use a WebAssembly Faust compiler to run completely client-side. The Web Audio backend for the Faust compiler allows for realtime testing of Faust code for signal processing. The Faust Online Compiler simplifies development for users who may find setting up the Faust toolchain difficult on the desktop by providing a zero-install solution that is capable of working with single-file Faust projects. However, it does not save projects to the server, nor supports multi-file projects.

SOUL Playground[9] is an online environment for developing and testing single-file projects using the SOUL[10] domain-specific language for audio programming. Similar in features to the Faust Online Compiler, the playground allows compiling SOUL projects in the browser and auditioning the results using Web Audio and Web MIDI. The SOUL playground does not have import/export features like FOC, but does have a global server-side saving system that is not tied to accounts. Users can press the "Compile and Run" option to audition results, then use the "Share this playground" button to provide a link to a saved project. The ability to share projects easily through web URLs promotes a socially-rich online culture for an ecosystem and is one that we plan to implement for the Csound Web-IDE.

Looking at non-audio web IDEs, a number of systems provide multi-file project support. CodePen[11] describes itself as a "social development environment" and provides a system for multi-file projects with a fixed-set of files (CodePen *Pens*, made up of a single HTML, JS, and CSS file) as well as an user-definable set of directories and files (CodePen *Projects*). Both systems provide editors for files of different types, though Projects allow the user flexibility to add as many files and directories as desired. Both systems also provide options to export the project from CodePen into a zip that can be expanded and further edited and used from a desktop system. In addition to the development environment, projects can be made either public or private as well as *forked* for modification by other users. CodePen's Projects is a fully-featured IDE that compares well with desktop IDEs and maintains projects using a virtual filesystem that compares well the desktop IDE development experience. The Csound Web-IDE seeks to provide a similar feature for social development set as CodePen Projects but targetting Csound audio and music work rather than client-side web development.

Glitch provides both client-side and server-side development using multi-file projects served from per-user virtualized containers. Glitch touts easy *remixing* of projects (similar to CodePen's *forking* system), saved project history backed by Git, as well as realtime collaboritive editing through its Team system. While we are not looking at these features for the current phase of Csound Web-IDE development, we do look to Glitch as a model of features that we would like to implement in future iterations of our project.

JSFiddle[12] is a fixed-set, multi-file web IDE, similar to CodePen's Pens system. JSFiddle provide easy, quick experimentation for small experiments that does not require logging in; boilerplates as starting points for projects; project history and forking; as well as easy inclusion of many popular third-party JS libraries via entering a URL or searching for them through a registry. JSFiddle's anonymous experimentation and importing of libraries are features we are interested to implement for Csound Web-IDE.

Finally, the p5.js Web Editor[13] offers a multi-file, filesystem-based IDE for p5.js projects. Projects are all public but only accessible if one is given a URL link from its author. The Web Editor provides an easy-to-use environment for development and teaching with p5.js that serves similar goals as what we envision for the Csound Web-IDE and the

[6]http://csoundqt.github.io/
[7]http://cabbageaudio.com/
[8]https://faust.grame.fr/onlinecompiler/
[9]https://soul.dev/playground/
[10]https://soul.dev/
[11]https://codepen.io/
[12]https://jsfiddle.net/
[13]https://editor.p5js.org/

Csound community.

3. PROJECT GOALS

Our goals for the Csound Web-IDE are to provide a zero-install, easy to use development environment for Csound. We believe a browser-based web application will provide many of the same benefits as using a desktop-based application, such as CsoundQt, while also providing additional features that will allow it to reach a broader audience. Use cases we considered include:

- Music composition
- Procedural Sound Generation
- Realtime, interactive audio system development
- Computer music pedagogy
- Cross-platform Csound project development

To serve those use cases, the Csound Web-IDE is currently designed to include the following features:

- Multi-file projects with support for audio assets
- Code editor with syntax highlighting and other editing features
- Project rendering in realtime as well as to disk
- Live code evaluation
- Support audio and MIDI processing through Web Audio and Web MIDI APIs
- Signal scoping (i.e., waveform and spectrum views)*[14]
- User-defined graphical user interfaces*
- Project collections*
- Public project and collections links making projects easy to share*

By implementing the above, the Csound Web-IDE should be an application where users may create or import projects; develop projects within the IDE; share projects with others; and export projects to be used on other platforms that support Csound (i.e., desktop, mobile, and embedded systems). Also, by targeting the browser platform, we see potential for using the IDE in places where installation of desktop software is difficult, if not impossible, to manage. Scenarios include image-based lab computers (such as found at educational institutions) and Chromebooks[15] running Google's Chrome OS.

4. ARCHITECTURE

The following describes the architecture of the Csound Web-IDE application. It describes technologies used; project data representation; and interaction with Csound, Web Audio, and Web MIDI for realtime rendering.

[14]Items marked with asterisks are not yet implemented as of time of this writing and are planned for later phases of development.

[15]https://www.google.com/chromebook/

4.1 Technologies

4.1.1 Server

The Csound Web-IDE is built using Firebase as a backend. Rather than develop our own bespoke server application, we chose to follow a *serverless* design and chose Firebase to fulfill our requirements for authentication, document storage, binary file storage, computation, and web serving services. These requirements are met using Firebase's Authentication, Cloud Firestore, Storage, Cloud Functions, and Hosting modules respectively.

The choice to delegate server functionality to Firebase was made primarily due to ease of development and cost. In this early stage of development, we are currently working under the limits of the Spark Plan, a free-to-use tier meant for small projects. Usage and cost over time will factor into future decisions whether to continue using Firebase under paid plans or to migrate to a different solution.

4.1.2 Client

The Csound Web-IDE client is built with various Javascript libraries using Typescript. React[16] and Redux[17] handle the user-interface state and rendering logic. As with many modern Javascript projects, we depend on many libraries from the NPM package manager. MaterialUI[18] provides many pre-styled components like buttons, menus and lists. The code editor itself is based on CodeMirror[19], the tab system on GoldenLayout[20] and FirebaseUI[21] provides us with ready made components for login. We use Web Audio Csound[8], a C library and an interface compiled with Emscripten[9] into JavaScript and WebAssembly, as our audio engine. This version of Csound is compiled from the same source as all of the other versions of Csound (i.e., desktop, mobile, embedded) and comes with additional JS library code that manages connections between Web Audio and Web MIDI to Csound.

While our primary goal is to deploy the client using standard web browser loading of applications over the internet, we have seen that the Web-IDE client works equally well as an Electron[22] desktop application. By using Electron we have full access to a modern browser engine (Chromium) which can run Web Audio Csound. Electron also provides full access to NodeJS which gives us the possibility to use either Web Audio Csound or NodeJS native-bindings to desktop Csound. Further research of the pros and cons in using native bindings is necessary to determine the value of providing this option.

4.2 Project Data and File System

Historically, Csound projects were made up of files organized into directories. A primary .orc and .sco file or single unified .csd file was used as the entry point into a project. Additional code files may be included with Csound's preprocessor and projects may employ binary assets (e.g., audio files, text data tables). The project was loaded at the start

[16]https://reactjs.org/

[17]https://redux.js.org/

[18]https://material-ui.com/

[19]https://codemirror.net/

[20]https://golden-layout.com/

[21]https://opensource.google.com/projects/firebaseui

[22]https://electronjs.org/

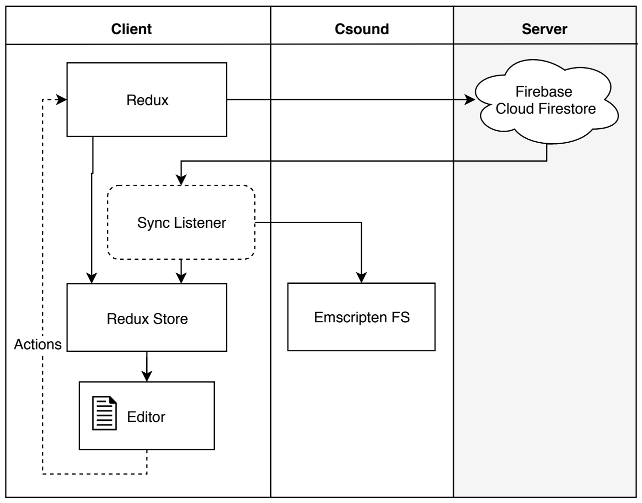


Figure 2: Diagram showing flow of project data. The Redux Store is the primary in-memory store for project data on the client. Redux Actions from the application or changes coming from Firebase mutate the Store which is synced to the client via React. Firestore changes, via a sync listener, update the Emscripten FS and Redux Store.

of Csound execution. With the introduction of the Csound API in Csound 5 and its further development in Csound 6, Csound evolved into a system that could be used as a server that begins execution without a project and where code would be evaluated live at runtime. This allowed the source of Csound project data to be determined by the host application which could, for example, store data in non-filesystem stores, such as databases, or use bespoke data file formats.

For Csound Web-IDE, we decided to follow the traditional model of project representation as a set of files and directories. Although the implementation uses in-memory data-stores as a virtual filesystem, the overall effect is that the project data model appears to the user in the same way as they would experience when working with Csound on other platforms and in working with other Csound IDEs. We believe this will simplify onboarding for experienced Csound users and that the filesystem model would be easily understood by others with programming experience in other languages. Using a filesystem representation was also essential to permit Web Audio Csound to operate using C file I/O functionality with Emscripten FS, meaning no platform-specific code would be necessary in Csound's source code for the browser platform.

Figure 2 shows an overview of how project data is stored within the Csound Web-IDE application. The primary source of truth for the client-side application is the Redux Store. Project data is loaded from Firebase (used for long-term persistence) into the store and also synced to the Emscripten FS via a synchronization listener. User edits to the project trigger Redux Actions which may directly modify the store or first write to Firebase which then triggers the synchronization flow. Once files are written to Emscripten FS, Csound can load and interpret those files.

For the user, the presentation of data appears much as it would on a desktop IDE. The project data files and organization is shown using a file-tree. Selecting files from the tree opens up filetype-specific editors in the main editing area for the Web-IDE. Users can use the file-tree to perform standard filesystem operations such as creating, renaming, and moving files and directories. Additionally, projects have permissions associated with them that limit accessibility only to the user (private) or permit visibility to the world (public). Using a filesystem representation in Csound Web-IDE offers a well-known paradigm for Csound users. It also permits easy exporting of projects for use with Csound on other platforms and importing into the Csound Web-IDE system from those platforms.

4.3 Csound, Web Audio, Web MIDI

Originally, Csound operated strictly by rendering to disk. Csound would load projects that define instrument definitions and score data that is driven by a scheduler to trigger events (e.g., activate an instrument instance at a given time for a given duration with x additional parameters). After interpreting the project data and loading data into memory, Csound would then run to completion and exit. Users audition the resulting audio file and iterate in this cycle of editing, rendering, and testing the project until they were satisfied with the results.

Later, when realtime support was added[6], Csound operation changed to render to DAC and allow auditioning in realtime. This change also allowed for realtime input and output through text, GUI interfaces, audio signals, and MIDI data. I/O with external systems was provided with Csound via plugins. Users would select input and output devices when executing Csound and, within their projects, use Csound opcodes to read and write data from external sources. They could write code generically that would work with any kind of audio and MIDI data that was routed to/from Csound through the plugin drivers. This allowed them to write a project that could function on a desktop system as well as–in the case of Cabbage–within the context of hosted VST and AU plugins. Additionally, in Csound 6[2], the API and engine were extended to support runtime evaluation of both Csound ORC and SCO code, opening up the possibilities of live coding with Csound.

Over time, Csound usage has grown to cover many use cases including music composition, sound design, development of realtime synthesizers, interactive audio applications, and more. Since the Csound library used in this project is compiled using the same C code as desktop, mobile, and embedded versions of Csound, it supports the same use cases as found on those other platforms.

The following discusses the usage of Csound in the Csound Web-IDE and interaction with Web Audio and Web MIDI APIs for realtime rendering. (Ahead-of-time rendering to disk and user-defined GUI I/O support is planned for the Web-IDE, but not yet implemented; see Section 5 for further discussion.)

4.3.1 Realtime Render

The Csound audio engine begins rendering when a user presses the "Play" button. At this time, messages are sent to Web Audio Csound to begin rendering using the file marked as the "main" project file. Users may additionally click on a file on the file-tree to change what is the "main" project or right-click on a different Csound file to execute as the starting point of the project.

During the initialization phase, the CsoundObj API–provided by Web Audio Csound–will create a Web Audio AudioNode that is either an AudioWorkletNode or ScriptProcessorNode, depending upon what the browser supports. The node wraps a running Csound instance, reads incoming samples from Web Audio and transfers them to the instance, executes Csound enough times to fill the Web Audio buffer size, then writes outgoing samples from Csound back to the Web Audio node graph.

Csound then starts by reading files from the Emscripten FS, starting from the main file and moving through #include'd code files and binary assets. Once the code is read and interpreted, realtime rendering begins and continues until completion. If a project is designed in the traditional way, completion occurs when the event scheduler is clear of any pending events and Csound's audio graph is clear. If a project is designed for realtime processing, the project may run continuously until a user action explicitly turns off Csound.

4.3.2 MIDI Input

While a Csound project is rendering, users may route MIDI input from WebMIDI devices into Csound. The MIDI data may be used for instantiate notes or sending controller messages for interpretation by Csound user-code. This is done using CsoundObj's provided utility methods for instantiating and routing Web MIDI to Csound.

4.3.3 Live Coding

In addition to realtime audio and MIDI input, Csound Web-IDE's editors support live coding by realtime evaluation of both ORC and SCO code. This is done by sending selected code from the editor as text to CsoundObj which in turn routes text as arguments to one of the appropriate Csound API functions for code evaluation. This action of evaluation will bypass triggering changes to the persistent storage in both Firestore and Emscripten FS.

Evaluating Csound code at runtime permits users to modify definitions of instruments and user-defined opcodes as well as auditioning score statements. Updating a running system often yields a faster development experience compared to the classic edit, render to completion, and test cycle when composing music with Csound. Runtime evaluation also permits live coding performances with Csound Web-IDE. For performance, users may create a basic project that contains library code and audio assets they wish to use, start Csound, then use a blank editor as a starting point for live coding their performance.

Using the Csound Web-IDE for live coding, together with audio and MIDI input, covers a large number of use cases where Csound has traditionally been used in realtime on the desktop. It is a testament to the state of browser technologies that a system like Csound could be supported via WebAssembly, Web Audio, and Web MIDI, to provide a cross-platform, zero-install Web-IDE that can function much like its desktop counterparts.

5. FUTURE WORK

Csound Web-IDE is currently in an alpha-state of development. We plan to finish implementing the features mentioned in Section 3 before opening a public beta for testing and feedback. For the first release, the feature set is primariy focused on individual usage of the IDE. Social development features, history tracking, and user-defined graphical user interface development are planned for future rounds of development.

Current work with binary assets has been limited in scope to sizes smaller than 1 megabyte. Since Web Audio Csound (as well as other versions of Csound) works with compressed files in OGG and FLAC formats, we believe the system should be able to handle any pedagogical requirements and a large number of use cases for creative purposes in the short-term. Further research is required for both maximum number of assets and maximum size of assets, both in terms of memory usage on the client-side as well as cost to support on the server-side. We believe this area of research will be one applicable to others developing Web Audio applications, particularly those using Emscripten FS.

Ahead-of-time rendering to disk is planned using a WebWorker. Csound will write to Emscripten FS and the user will be able to download the rendered audio file as a Blob. We plan to include this functionality in the initial release of the Web-IDE.

The current editor supports syntax highlighting but does not yet support other commonly found features in IDEs. Initial work on documentation browsing and lookup of individual opcode entries of the Canonical Csound Reference Manual[7] is currently implemented in the Web-IDE and serves as the basis for future work on code completion for opcodes. Example, template, and tutorial projects are also on the roadmap for development.

In addition to the Electron-based desktop client, we plan to explore offering Csound Web-IDE as a Progressive Web Application[23]. This option has already been explored with success for another Web Audio Csound-based application, csound-live-code[24], where it is installable as a desktop and Android application using the Chrome browser. We hope to show that browser-based audio applications can serve in many places traditionally handled by native desktop and mobile applications.

Once user-defined GUI components and the GUI editor are implemented, we plan to explore making Csound Web-IDE projects be publishable as Web Audio Modules (WAM)[3] and Web Audio Plugins (WAP)[1]. Offering Csound-based plugins that interoperate with other Web Audio-based applications would open up new areas where Csound may be used for musicians and developers.

Exploring interoperability with other Csound-based applications and platforms will require collaboration with developers of CsoundQt and Cabbage as well as the larger Csound community. We are excited to see where our web-based system can compliment the capabilities of existing desktop IDEs.

Finally, we are excited to see how the Csound Web-IDE can serve education for both Csound and computer music. We are hopeful that public collections of projects, together with social development features, can be a vehicle for development of content suitable for education using an active learning approach.

6. CONCLUSIONS

[23]https://developers.google.com/web/progressive-web-apps/

[24]https://live.csound.com/

The Csound Web-IDE provides a web-based development environment for Csound sound and music computing projects. This was implemented using Firebase for the server-side and numerous well-known libraries and frameworks for the client-side application. Developing a full-featured IDE for Csound on the browser platform was made possible due to Emscripten's FS filesystem–employed by Web Audio Csound–as well as using Web Audio and Web MIDI APIs. Social development features found in other web-based IDEs are planned to encourage networking, education, sharing, and discovery of Csound work for users.

We hope the IDE proves useful for both creative and pedagogical purposes. We look forward to expanding upon the system over time to serve both computer music creators and educators in their work.

7. REFERENCES

[1] M. Buffa, J. Lebrun, J. Kleimola, O. Larkin, G. Pellerin, and S. Letz. WAP: Ideas for a Web Audio Plug-in Standard. In *4th Web Audio Conference, TU Berlin, Berlin*, 2018.

[2] J. P. ffitch, V. Lazzarini, S. Yi, M. Gogins, and A. Cabrera. The New Developments in Csound 6. In *International Computer Music Conference 2015*, 2015.

[3] J. Kleimola and O. Larkin. Web Audio Modules. SMC, Maynooth University, 2015.

[4] V. Lazzarini, J. ffitch, S. Yi, Ø. Brandtsegg, J. Heintz, and I. McCurdy. *Csound: A Sound and Music Programming System*. Springer, Berlin, 2016.

[5] R. Michon and Y. Orlarey. The Faust online compiler: a web-based IDE for the Faust programming language. In *Proceedings of the Linux Audio Conference (LAC-12)*, pages 111–116, 2012.

[6] B. Vercoe and D. Ellis. Real-time CSound: Software Synthesis with Sensing and Control. In *International Computer Music Conference 1990*, 1990.

[7] B. Vercoe et al. The Canonical Csound Reference Manual, 2019.

[8] S. Yi, V. Lazzarini, and E. Costello. WebAssembly AudioWorklet Csound. In *4th Web Audio Conference, TU Berlin, Berlin*, 2018.

[9] A. Zakai. Emscripten: An LLVM-to-JavaScript Compiler. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '11, pages 301–312, New York, NY, USA, 2011. ACM.