

---

# Explorative Konzeption und Implementierung einer web-basierten Plattform zur musikalischen Echtzeit-Kollaboration an Modular-Synthesizern

Masterarbeit zur Erlangung des akademischen Grades  
*Master of Science*  
im Studiengang Medieninformatik  
an der Fakultät für Informatik und Ingenieurwissenschaften  
der Technischen Hochschule Köln

vorgelegt von: Sebastian Brock  
Matrikel-Nr.: 11139299  
Adresse: Zum Alten Weiher 13  
51674 Wiehl  
sebastian.brock@smail.th-koeln.de

eingereicht bei: Prof. Christian Noss  
Zweitgutachter\*in: Prof. Dr. Christian Faubel

Wiehl, 05.08.2024

## Kurzfassung/*Abstract*

Diese Arbeit untersucht die Konzeption und Implementierung einer webbasierten Plattform für die musikalische Echtzeit-Kollaboration an Modular-Synthesizern. Traditionelle musikalische Zusammenarbeit erfordert in der Regel die Anwesenheit aller Teilnehmer am gleichen Ort. Während bereits Softwarelösungen für Kontexte wie Remote-Proben von Bands existieren, lösen diese nicht die einzigartigen Herausforderungen, welche von hoch individualisierbaren Instrumenten wie Modular-Synthesizern ausgehen. Synthesizer dieser Art sind hochgradig anpassbar, wodurch jedes Setup einzigartig sein kann. Zusätzlich macht ihre analoge Bauweise die genaue Replikation von Einstellungen zwischen entfernten Systemen in der Praxis nahezu unmöglich.

Das Hauptziel dieses Projekts ist die Entwicklung eines Systems, welches eine kollaborative Klangerzeugung mit Modular-Synthesizern über das Web ermöglicht. Dies erfordert nicht nur technische Lösungen für die Klangerzeugung, sondern auch Strategien zur Synchronisierung mehrerer Sitzungen in Echtzeit unter Berücksichtigung der Herausforderungen von Web-Architekturen und Latenz.

Das Projekt verfolgt sowohl in der Konzeption als auch in der Implementierung einen explorativen Ansatz, sodass technische Möglichkeiten durch die praktische Anwendung bewertet werden können. Zentrale Aufgaben umfassen die Grundlagenforschung zur web-basierten Klangerzeugung, die Überprüfung bestehender Lösungen und die Entwicklung einer funktionalen Plattform, die eine virtuelle Zusammenarbeit an Modular-Synthesizern ermöglicht.

Herausforderungen wie die Minimierung der Latenz und die inhärenten Einschränkungen digitaler Simulationen im Vergleich zu ihren analogen Gegenstücken werden adressiert. Das Projekt zielt darauf ab, Erkenntnisse zu liefern, die nicht nur für die musikalische Kollaboration relevant sind, sondern auch auf andere Bereiche anwendbar sind, die eine Echtzeit-Interaktion über das Web erfordern.

Das praktische Ergebnis dieser Arbeit ist ein funktionsfähiges, webbasiertes System für die synchrone Zusammenarbeit an einem virtuellen Modular-Synthesizer. Die während des Projekts gewonnenen Erkenntnisse und entwickelten Methoden werden dokumentiert und bieten Beiträge zu den Bereichen Webentwicklung, digitale Klangerzeugung und Technologien für die Remote-Kollaboration.

# Inhaltsverzeichnis

|  |           |
|--|-----------|
| <b>Glossar</b>   | <b>IV</b> |
| <b>1 Einleitung</b>  | <b>1</b>  |
| 1.1 Funktionsweise von Modular-Synthesizern . . . . .            | 1         |
| 1.2 Problemstellung . . . . .                                    | 3         |
| 1.3 Lösungsansatz und Zielsetzung . . . . .                      | 4         |
| 1.4 Explorativer Forschungsansatz . . . . .                      | 5         |
| 1.5 Abgrenzung zu bestehenden Lösungen . . . . .                 | 6         |
| 1.6 Prozessdokumentation . . . . .                               | 7         |
| <b>2 Konzeption</b>  | <b>8</b>  |
| 2.1 Anforderungsanalyse . . . . .                                | 8         |
| 2.1.1 Technische Aspekte . . . . .                               | 9         |
| 2.1.2 Soziale Aspekte . . . . .                                  | 9         |
| 2.2 Interface- und Interaktionskonzeption . . . . .              | 10        |
| 2.2.1 Arbeitsfläche . . . . .                                    | 10        |
| 2.2.2 Navigationsleiste . . . . .                                | 11        |
| 2.2.3 Modul-Auswahl . . . . .                                    | 11        |
| <b>3 Implementierung</b>   | <b>13</b> |
| 3.1 Audio-Architektur . . . . .                                  | 13        |
| 3.2 Aufbau des Frontends . . . . .                               | 16        |
| 3.2.1 Zuständigkeitsverteilung der Komponenten . . . . .         | 16        |
| 3.2.2 Event-Bus . . . . .  | 18        |
| 3.2.3 State-Management . . . . .                                 | 18        |
| 3.2.4 Arbeitsfläche . . . . .                                    | 20        |
| 3.2.5 Verbindung zwischen Frontend und Audio-Kontext . . . . .   | 20        |
| 3.3 Ermöglichen von Kollaboration . . . . .                      | 23        |
| 3.3.1 Vermitteln von virtuellem Besitz und Territorium . . . . . | 24        |
| 3.3.2 Etablieren von Sessions . . . . .                          | 25        |
| <b>4 Herausforderungen</b>                                       | <b>27</b> |
| 4.1 Umsetzung von Trigger-Signalen . . . . .                     | 27        |
| 4.2 Nutzung von Audio- und CV-Signalen . . . . .                 | 28        |

|          |   |           |
|----------|---|-----------|
| 4.3      | Optimierung der Schnittstelle Frontend/Audio . . . . .              | 30        |
| 4.4      | Gestiegener Datenverbrauch des Websockets durch neue Module . . . . | 31        |
| <b>5</b> | <b>Diskussion</b>   | <b>32</b> |
| 5.1      | Audiogenerierung und Interaktion . . . . .                          | 32        |
| 5.2      | Kollaboration und Latenz . . . . .                                  | 33        |
| <b>6</b> | <b>Fazit und Ausblick</b>   | <b>35</b> |
|          | <b>Literatur</b>  | <b>36</b> |
|          | <b>Anhang</b>   | <b>38</b> |

## Glossar

**Debounce-Funktion** Debounce-Funktionen sind Funktionen, die dazu dienen, unerwünschte Mehrfachsignale zu verhindern, indem die Funktion in einem vorgegebenen Zeitraum nicht mehr als ein mal ausgeführt werden kann. . 24

**Hüllkurvengenerator** Auch Envelope Generator genannt - ein Modul in Synthesizern, das zur zeitlichen Steuerung von Parametern wie Lautstärke, Filterfrequenz oder Tonhöhe eines Audiosignals verwendet wird. Er erzeugt eine Steuerspannung, die über einen definierten Zeitraum verläuft und typische Phasen wie Attack (Anstieg), Decay (Abfall), Sustain (Haltephase) und Release (Abklingen) umfasst. . 27

**Mikrotonalität** Mikrotonalität beschreibt die Nutzung von mikrotonalen Intervallen in Musik - dies sind Intervalle, welche kleiner als ein Halbtonabstand sind.. 3

**Potentiometer** Ein Potentiometer ist ein elektrisches Bauelement, das zur variablen Einstellung von elektrischen Widerständen in einem Stromkreis dient. Es wird häufig zur Steuerung von Lautstärke, Helligkeit und anderen variablen Parametern in elektronischen Geräten verwendet. . 3

**Sampling** (Im Kontext der digitalen Signalverarbeitung) Prozess, bei dem ein kontinuierliches analoges Signal in diskrete digitale Werte umgewandelt wird. Dies geschieht durch regelmäßige Messung der Amplitude des analogen Signals zu bestimmten Zeitpunkten, den sogenannten Abtastzeitpunkten. Die Qualität der Digitalisierung hängt von der Samplingrate ab, also der Anzahl der Abtastungen pro Sekunde. Eine höhere Samplingrate führt zu einer genaueren Repräsentation des ursprünglichen Signals. . 15

**Trigger-Signal** Kurzes, impulsartiges elektrisches Signal, das eine bestimmte Aktion oder ein Ereignis in einem Modul auslöst. Trigger-Signale werden häufig verwendet, um bestimmte Ereignisse zu synchronisieren oder zu steuern, wie das Starten oder Stoppen von Hüllkurvengeneratoren, das Auslösen von Samples oder das Umschalten von Zuständen in Schaltmodulen. . 18

**VCA** Ein Voltage Controlled Amplifier (VCA) ist ein elektronisches Gerät oder Modul, das in der Audiotechnik verwendet wird, um die Lautstärke eines Audiosignals durch eine externe Steuerspannung zu regeln. Der VCA arbeitet als Verstärker, dessen Verstärkung (Gain) proportional zur Höhe der angelegten Steuerspannung ist. . 14

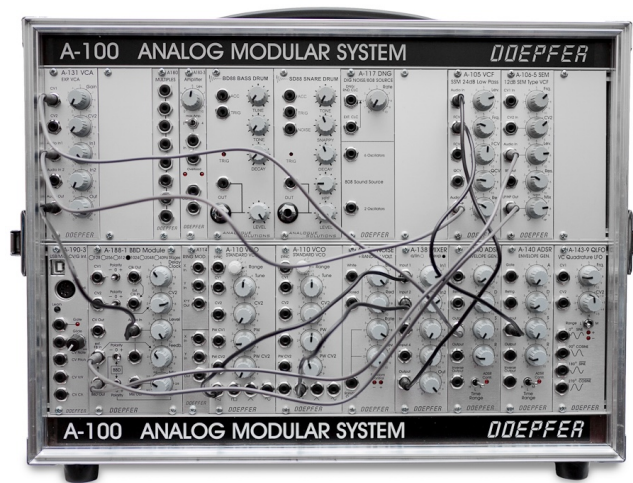
# 1 Einleitung

## 1.1 Funktionsweise von Modular-Synthesizern

In der Welt der Musikinstrumente beanspruchen Synthesizer und im speziellen Modular-Synthesizer in der Regel eine Kategorie für sich - denn sowohl der Aufbau als auch das Interaktionsprinzip weichen stark zu anderen Instrumenten ab. Ein Modular-Synthesizer gleicht dabei selten einem anderen - aus welchen Modulen der Synthesizer besteht, kann stark variieren und ist davon abhängig, nach welchen Kriterien die verbauten Module vom Anwender ausgesucht werden. Aus diesem Grund können Größe und Komplexität zwischen verschiedenen Synthesizern ebenfalls stark unterschiedlich sein (Enders, 1985).

Das zugrundeliegende Prinzip von Modular-Synthesizern bleibt jedoch stets gleich - anhand von in sich geschlossenen elektronischen Bauteilen (Modulen) können Klänge erzeugt und gestaltet werden. Diese Module bieten dem Anwender Bausteine zur Klangerzeugung und können beispielsweise Oszillatoren (zur Klangerzeugung durch elektrische Schwingungen) oder Filter (zur klanglichen Weiterverwertung und Gestaltung bereits vorhandener Töne) umfassen. Da diese Module grundsätzlich in sich geschlossene Elemente sind, müssen sie erst vom Anwender miteinander verbunden werden, um einen Klang nach außen transportieren zu können. Dafür bieten Module Ein- und Ausgänge, im Eurorack-Format zum Beispiel durch 3,5mm Mono-Klinkenstecker. Die Module können über diese Anschlüsse durch den Musiker mit Kabeln verbunden werden, sodass Signalketten entstehen.

Der modulare Aufbau bietet dem Anwender die Möglichkeit, das Instrument nach Belieben aufzubauen und zu verändern. Welche Module in welcher Anordnung verbaut werden, ist frei wählbar. Solange die verbauten Module innerhalb eines festgelegten Standards wie *Eurorack* oder *Moog Unit* sind, können sie miteinander verbunden werden, um Klang zu erzeugen (Jenkins, 2019). Die Kabel, mit welchen die Module untereinander verbunden werden, können dabei zwei Arten von Signalen übertragen: **Audiosignale** und **Steuerspannungssignale** (im folgenden aus dem englischen *Control Voltage* oder *CV* genannt).

Abbildung 1.1: Doepfer A-100 Modular-Synthesizer <sup>1</sup>

**Audiosignale** sind analoge Signale, in der Regel mit einer Amplitude von  $\pm 5V$ . Sie werden genutzt, um Klang von einem Modul in ein anderes zu übermitteln. Das entgegennehmende Modul kann dieses Signal anschließend nutzen und in der Regel in abgewandelter Form wieder herausgeben. Wenn ein solches Signal durch einen Lautsprecher oder Kopfhörer entgegen genommen wird, kann es (solange es sich innerhalb des menschlich wahrnehmbaren Frequenzbereichs bewegt) als für den Menschen hörbaren Klang wiedergegeben werden. Da das Signal analog vorhanden ist, muss es dafür nicht erst durch Komponenten wie einen Digital-Analog-Wandler verarbeitet werden. Durch die Analogität entsteht zusätzlich der Vorteil, dass die Klangauflösung theoretisch unbegrenzt ist - es gibt keine Limitierung durch Sample-Raten wie bei einem digitalen Signal (Veretekhina et al., 2018).

**Control Voltage Signale** ermöglichen es, Parameter eines Moduls zu verändern, ohne dass dafür beispielsweise ein Regler am Modul betätigt werden muss. So wird ein bestimmter Parameter an die Eingangsspannung durch ein CV-Signal geknüpft - verändert sich die Eingangsspannung, verändert sich auch der Wert des Parameters. Die Amplituden dieser Spannungen können generell variieren, es gibt jedoch Standards, beispielsweise bei Eurorack; hier ist für Tonhöhen eine Spannung von 1V pro Oktave definiert. Die Nutzung von Spannungssteuerung bildet ein zentrales Grundprinzip von Modulersynthesizern und ermöglicht die Zustandsveränderung des Synthesizers ohne Fremdeinwirkung des Nutzers. (Enders, 1985).

<sup>1</sup>Foto von Nina Richards via Wikimedia Commons, [https://commons.wikimedia.org/wiki/File:Doepfer\\_A-100.jpg](https://commons.wikimedia.org/wiki/File:Doepfer_A-100.jpg)



## 1.2 Problemstellung

Die für Modular-Synthesizer einzigartigen Eigenschaften können in vielerlei Hinsicht Vorteile bieten, doch entstehen auch Limitierungen, vor allem im Rahmen der musikalischen Kollaboration mit anderen Personen. Würden beispielsweise zwei Personen an unterschiedlichen Standorten an einem Synthesizer arbeiten wollen, wäre dies nur schwer umzusetzen. Zum einen müsste der verwendete Modular-Synthesizer zwei mal in identischer Konfiguration an den Standorten vorhanden sein, zum anderen müssten beide Personen die Kabelverbindungen und Reglerpositionen des jeweils anderen Synthesizers stets nachstellen. Da die in den Modulen verbauten Potentiometer in der Regel stufenlos und rein analog sind, ist eine exakte Replikation der Reglerpositionen nahezu unmöglich. Durch die stufenlosen Einstellungsmöglichkeiten sind die erzeugten Klänge außerdem häufig nicht exakt Noten der herkömmlichen Notenskala zuzuordnen - Frequenzen können sich in Tonhöhenbereichen zwischen Noten befinden (dies wird *Mikrotonalität* genannt), was eine Replikation von Tonhöhen auf Basis des menschlichen Gehörs weiter erschwert.

Mit einer wachsenden Anzahl an Modulen und Kabelverbindungen steigt zusätzlich die Komplexität, was die Fehleranfälligkeit bei der Replikation von Aktionen erhöht und eine Übersicht über den Gesamtzustand des Synthesizers erschwert.

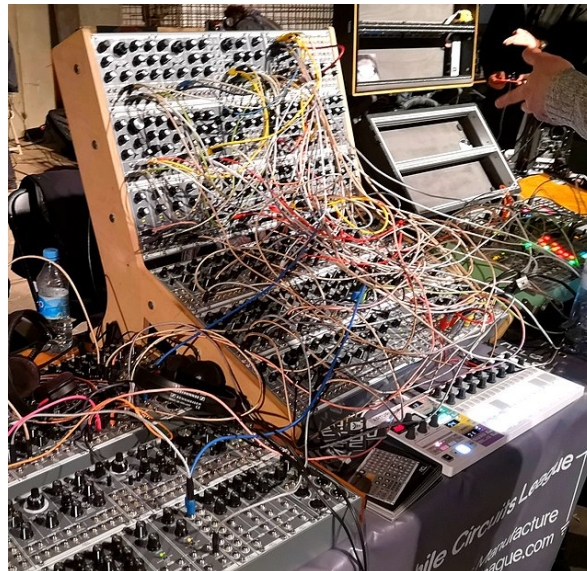


Abbildung 1.2: Modular-Synthesizer mit verbundenen Patchkabeln <sup>2</sup>

<sup>2</sup>Foto von Hans Rompel via Wikimedia Commons, [https://commons.wikimedia.org/wiki/File:Modern\\_handbuilt\\_analogue\\_synthesizer.jpg](https://commons.wikimedia.org/wiki/File:Modern_handbuilt_analogue_synthesizer.jpg)

Aus den genannten Gründen stellt sich die Kollaboration über Distanz an Modular-Synthesizern in der Praxis als unattraktiv und problembehaftet dar. In diesem Kontext besteht daher das Potenzial, die musikalische Kollaboration über Distanz durch Entwicklung eines Software-Systems zu ermöglichen.

### 1.3 Lösungsansatz und Zielsetzung

In dem gegebenen Problemkontext bietet sich die Entwicklung eines Web-basierten Systems als mögliche Lösung an. Durch die Verlagerung der Klangerzeugung und -Interaktion in eine webbasierte Umgebung können mehrere Personen unabhängig von ihrem geografischen Standort gemeinsam an einem virtuellen Modular-Synthesizer arbeiten.

Ein wesentlicher Vorteil eines webbasierten Systems liegt dabei in der Zugänglichkeit. Webanwendungen sind plattformunabhängig und erfordern keine spezielle Hardware oder Softwareinstallation, was die Teilnahme an dem Kollaborationsprozess erleichtert. Nutzer könnten über einen Browser auf das System zugreifen und sofort mit der gemeinsamen Klanggestaltung beginnen. Dies senkt die Einstiegshürden und ermöglicht einer breiteren Nutzerschaft die Teilnahme an der kollaborativen Musikproduktion. Durch die Digitalisierung und Virtualisierung des Modular-Synthesizers wären die Parameter der vorhandenen Module außerdem digital in konkreten Werten vorhanden, was eine präzisere Kontrolle und Reproduzierbarkeit von Einstellungen ermöglicht. Die Notwendigkeit, zwei identische Modular-Synthesizer an zwei Standorten zu nutzen, entfällt ebenfalls - außer einem Endgerät mit Web-Browser wären keine Hardware-Anforderungen vorhanden. Das Speichern und Laden von Patch-Konfigurationen würde es dabei ermöglichen, kollaborative Sessions über größere Zeiträume hinweg zu gestalten. Nutzer müssten den Zustand des Systems während Pausen nicht wie bei einem realen Modular-Synthesizer unangetastet lassen, sondern könnten Patch-Konfigurationen später wiederverwenden.

Aus diesen Gründen bietet sich die Umsetzung eines Web-basierten Modularsynthesizers zur Lösung der dargelegten Problemstellung an. Dies soll im Rahmen dieses Projektes umgesetzt werden. Dabei sollen folgende zentrale Forschungsfragen beantwortet werden:

- Wie funktioniert Audiogenerierung im Web, und welche technischen Möglichkeiten existieren?
- Ist es möglich, die musikalischen Möglichkeiten eines Modular-Synthesizers vollständig im Web abzubilden?
- Wie kann musikalische Echtzeit-Kollaboration im Web umgesetzt werden?

- Bietet das Web zu viele technische Limitierungen (beispielsweise im Kontext von Audioqualität oder Latenz), um eine für Nutzer effektive Kollaboration zu verhindern?

## 1.4 Explorativer Forschungsansatz

Der für diese Arbeit gegebene Problemkontext umfasst eine Kombination aus verschiedenen Thematiken, darunter zentral Mensch-Computer-Interaktion, Soziale Interaktion via der Nutzung von Web-Systemen, sowie kreatives Arbeiten und das Erzeugen von Inhalten im auditiven Kontext im Web. Zu dieser Kombination aus Themen existieren zum Zeitpunkt dieser Arbeit zwar bereits Forschungen und Software-Lösungen für Kontexte wie beispielsweise die Remote-Verbindung von Musikern einer Band für Probe-Situationen <sup>3</sup>, oder die Synchronisation von vollwertiger Studio-Software über Distanz <sup>4</sup>, doch in der Kombination mit den vorangegangen beschriebenen, für Modular-Synthesizer teils einzigartigen Interaktionsansätzen sind bisher keine etablierten Lösungen vorhanden. Aus diesem Grund wird in dieser Arbeit ein möglichst ganzheitlicher Ansatz verfolgt, die Problemstellung zu erforschen.

Für Kontexte wie diesen bietet sich ein explorativer Forschungsansatz an. Explorative Forschung kann vor allem eingesetzt werden, um grundsätzliche Erkenntnisse zu einem Kontext zu erhalten, in welchem wenige oder keine Vorarbeiten vorhanden sind - daher hat dieser Ansatz auch stets die Aufgabe, eine Basis für eventuell weiterführende Forschungen zu schaffen (Schüll, 2009). Da explorative Ansätze einen erkundenden Charakter besitzen, wird infolge dessen eine gewisse Ergebnisoffenheit vorausgesetzt.

Im Rahmen von Entwicklungsarbeiten wie in diesem Projekt erlaubt ein explorativer Entwicklungsansatz, flexibel und adaptiv auf unerwartete Probleme und neue Erkenntnisse zu reagieren, die im Laufe der Entwicklung auftreten können. Da der gegebene Kontext bislang nur begrenzt erforscht ist, bietet dieser Ansatz die Möglichkeit, verschiedene Technologien, Architekturen und Implementierungsmethoden systematisch zu testen und zu bewerten. Die Natur des Projekts erfordert ein tiefes Verständnis der theoretischen Grundlagen der Klanggenerierung im Web sowie der praktischen Umsetzung bestehender Lösungen. Ein explorativer Ansatz ermöglicht es, durch iterative Experimente und Prototyping verschiedene technische Möglichkeiten zu erproben und die vielversprechendsten Ansätze weiterzuverfolgen. Darüber hinaus erlaubt der explorative Entwicklungsansatz eine dynamische Anpassung an die spezifischen Bedürfnisse und Anforderungen, die sich im Verlauf der Entwicklung

---

<sup>3</sup>Software *Jamulus* von Volker Fischer <https://jamulus.io/de/>

<sup>4</sup>*VST Connect* von Steinberg <https://www.steinberg.net/de/vst-connect/>

herausstellen. Vor allem im Kontext von konkret messbaren Werten wie Latenz können neu- oder weiterentwickelte Methoden anhand dieser Werte bewertet, nahtlos implementiert und getestet werden.

## 1.5 Abgrenzung zu bestehenden Lösungen

Der Ansatz, einen Modular-Synthesizer virtuell nachzubilden, ist nicht grundsätzlich neu. Software wie die Open Source-Anwendung VCV Rack<sup>5</sup> ermöglicht die Simulation von Eurorack-Synthesizern und hat dabei zum Ziel, real existierende Module des Eurorack-Formats möglichst authentisch nachzubilden. Durch die Nutzung des LLVM-Compilers Emscripten existieren Versionen von VCV Rack, welche im Web benutzt werden können, beispielsweise miRack<sup>6</sup> oder Cardinal<sup>7</sup>. Durch Lösungen wie diese wird zwar die Nutzung eines Modular-Synthesizers im Web ermöglicht, doch es existiert zum Zeitpunkt dieser Arbeit keine verbreitete Webanwendung, welche eine Live-Kollaboration mehrerer Nutzer in diesem Kontext zulassen würde. Viele Anwendungen lassen den Nutzer zwar die Patchkonfiguration exportieren, sodass diese an einem anderen Standort von einem anderen Nutzer aufgerufen werden könnte, doch eine Echtzeit-Synchronisierung ist dabei nicht möglich.

Für einen gemeinschaftlichen, kreativen Prozess ist das Bewusstsein über andere Personen im Rahmen des Kontextes allerdings unabdinglich und wird auch als *workspace awareness* beschrieben (Fencott und Bryan-Kinns, 2013). Workspace Awareness bezieht sich dabei nicht ausschließlich auf Kollaborateure, sondern kann auch das Bewusstsein über die Zuhörer bei einer Live-Performance beschreiben. Desweiteren gilt Echtzeit-Kommunikation als besonders förderlich für kreative Improvisationsprozesse in Gruppen (Healey et al., 2005). Da die Arbeit an Modular-Synthesizern häufig einen experimentellen Charakter von Musikkomposition annehmen kann (White, 2022) und der Künstler eher die Rolle eines steuernden Dirigenten über ein elektronisches Orchester annimmt (Auricchio und Borg, 2016), ist das Ermöglichen von explorativen Improvisationsprozessen ein zentrales Ziel dieses Projekts. Um dieses Ziel zu erreichen, ist das Ermöglichen von Echtzeit-Kommunikation in der Anwendung von großer Relevanz und beschreibt einen einzigartigen, neuen Forschungsbereich, welcher in dieser Arbeit untersucht werden soll.

---

<sup>5</sup>VCV Rack-Webseite <https://vcvrack.com>

<sup>6</sup>miRack Webanwendung <https://assets.mifki.com/mirack/Rack.html>

<sup>7</sup>Cardinal-Webseite <https://cardinal.kx.studio>

## 1.6 Prozessdokumentation

Die Entwicklung der Anwendung wird vollständig über ein Git-Repository<sup>8</sup> verwaltet. So sind alle vorgenommenen Entwicklungsschritte ganzheitlich historisch nachvollziehbar. Gleichzeitig ermöglicht der Anbieter GitHub über den Dienst *GitHub Pages* das Deployment der Anwendung, um eine Zugänglichkeit des Systems über die Entwicklungsumgebung hinaus zu schaffen.

Um die zur Entwicklung eingesetzten Strategien und Technologien bewerten, und ihren Einsatz begründet nachvollziehen zu können, wird ein zweischrittiger Prozess angewandt. So werden alle Technologie- und Architekturentscheidungen anhand sogenannter *Architectural Decision Records* (ADRs)<sup>9</sup> im Projekteigenen Git-Repository festgehalten und anschließend kritisch eingeordnet. Ein ADR hält fest, welche Optionen zum Zeitpunkt einer technischen Entscheidung vorhanden sind, und welche Vor- und Nachteile die jeweiligen Optionen bieten. Aus den Optionen kann dann eine begründete Wahl für die Implementierung getroffen werden, sodass Entscheidungen auch später nachvollzogen werden können. Diese Methode wird in der Implementierungsphase dieser Arbeit angewandt, um trotz dem explorativen, agilen Entwicklungsansatz stets alle getroffenen Entscheidungen historisch nachvollziehen zu können.

Da die Entwicklungsphase die konzipierten Funktionen hinsichtlich des explorativen Forschungsansatzes nicht ausschließlich umsetzen, sondern ebenfalls im Prozess auftretende Herausforderungen dokumentieren soll, werden *Issues* des Git-Repositorys genutzt, um diese unmittelbar festzuhalten. Mit Referenz auf die jeweils umgesetzte Lösung innerhalb eines Issues können entstehende Herausforderungen so von der Identifikation bis zur Lösung vollständig dokumentiert werden. So kann der explorative Ansatz des Projekts maximal genutzt werden, um Informationen über den Forschungskontext zu sammeln.

---

<sup>8</sup>GitHub-Repository des Projekts: <https://github.com/sebastianbroc/websynth>

<sup>9</sup>Homepage der ADR GitHub-Organisation: <https://adr.github.io>

## 2 Konzeption

Vor Beginn der Entwicklung des Systems müssen die Anforderungen und ein grundlegendes Konzept für den Web-Synthesizer erarbeitet werden. Dieser Schritt ist notwendig, um Rahmenanforderungen für die Entwicklung klar zu definieren und eine solide Basis für die nachfolgende Implementierung zu schaffen. Aufgrund des explorativen Charakters der Arbeit beschränkt sich die Konzeption dabei bewusst auf einen lediglich grundsätzlichen Rahmen - die Entwicklungsphase soll eine größtmögliche Flexibilität und Adaptivität auf im Verlauf der Implementierung entstehende Erkenntnisse bieten, und dadurch einen explorativen Forschungscharakter behalten. Wesentliche Anforderungen und Kernprinzipien sollen dabei festgelegt werden und lediglich als Leitlinien für die Entwicklungsphase dienen. So kann das Projekt dynamisch auf neue Erkenntnisse und Herausforderungen reagieren, ohne durch eine zu detaillierte Planung eingeschränkt zu sein. Der Entwicklungsprozess kann daher auch als Agil beschrieben werden.

### 2.1 Anforderungsanalyse

Um die Erfordernisse an das zu entwickelnde System zu ermitteln, muss zunächst ein grundsätzliches Verständnis über die Anforderungen von Musikern als Nutzergruppe, und im speziellen den Nutzern von Modular-Synthesizern geschaffen werden. Durch den gegebenen Kontext der Kollaboration und Zusammenarbeit zwischen Menschen umfassen die Anforderungen dabei mehrere Dimensionen - neben rein technischen Aspekten sind auch soziale Aspekte wie der zwischenmenschliche Austausch der Musiker via des Systems zentrale Punkte, welche von Anfang an in der Entwicklung berücksichtigt werden sollten. Aus diesem Grund wird eine Anforderungsanalyse durchgeführt. Die ermittelten Anforderungen werden im Folgenden in technische und soziale Aspekte unterteilt.

### 2.1.1 Technische Aspekte

- Das System muss die Möglichkeit bieten, alle grundsätzlichen Aktionen, welche an Modular-Synthesizern vorgenommen werden können, virtuell zu modulieren.
- Das System muss die Grundprinzipien von Modular-Synthesizern virtuell nachbilden - dies umfasst unter Anderem die freie Wahl und Anordnung von Modulen, und die gegenseitige Modulation zwischen Modulen über Kabelverbindungen.
- Das System muss die Synchronisierung und gleichzeitige Bearbeitung eines einzelnen Synthesizer-Patches von mehreren Nutzern gleichzeitig ermöglichen.
- Im System vorgenommene Patches müssen speicher- und reproduzierbar sein.
- Eventuelle Konfliktsituationen bei der Synchronisierung von Sessions müssen nach Möglichkeit automatisch vom System gelöst werden, um Fehlerfälle zu vermeiden.
- Aus dem System entspringende Audiowiedergabe muss Geräteunabhängig identisch sein.

### 2.1.2 Soziale Aspekte

- Aktionen anderer Nutzer, welche an dem selben Patch arbeiten, müssen unmittelbar bei dem jeweils anderen Nutzer reflektiert werden.
- Die *Position* (in diesem Kontext: virtuelle Position, z.B. des Mauszeigers) anderer Nutzer muss stets live angezeigt werden.
- Die Latenz zur Synchronisierung von Sessions muss sich in einem für die Nutzer akzeptablen Rahmen befinden.
- Die Interaktion mit dem System muss für den Nutzer intuitiv sein und die Interaktion an einem realen Modular-Synthesizer so eng wie möglich nachbilden.
- Die Qualität der Audioinhalte, welche dem System entspringen, dürfen keinen stark wahrnehmbaren Qualitätsunterschied im Vergleich zu einem realen Modular-Synthesizer aufweisen.

## 2.2 Interface- und Interaktionskonzeption

### 2.2.1 Arbeitsfläche

Grundsätzlich soll die Interaktion an dem System die Tätigkeit an einem realen Modular-Synthesizer möglichst eng nachstellen, und dies bereits im User Interface reflektieren. Da das Interface eines Modular-Synthesizers lediglich aus den Ein- und Ausgängen, Reglern und Displays der verbauten Module besteht, sollte das Interface des Web-Synthesizers den visuellen Fokus auf den Bereich setzen, in welchem sich die Module befinden werden. Hierfür soll ein eigener Bereich des UIs reserviert werden, welcher *Arbeitsfläche* genannt wird und stets möglichst viel Fläche des Viewports einnehmen sollte, damit möglichst der gesamte Patch auf einen Blick sichtbar ist und nicht von anderen Inhalten eingeschränkt oder abgeschnitten wird.

Innerhalb der Arbeitsfläche sollen Module beliebig angeordnet werden können, sodass der Nutzer den Synthesizer wie bei einem realen Modular-Synthesizer so aufbauen kann, dass er ideal für den spezifischen Anwendungsfall und die gewünschten Töne genutzt werden kann. Dabei sollte der Nutzer die Möglichkeit erhalten, die Ansicht auf die Arbeitsfläche beliebig zu vergrößern oder zu verkleinern. So kann bei variierender Display-Größe des Endgerätes dennoch stets gewährleistet werden, dass durch einen hohen Zoom-Grad genug Details der Module sichtbar bleiben, bzw. bei einem niedrigen Zoom-Grad eine Übersicht über den gesamten Patch geschaffen werden kann.

Die Module sollten alle für die Audiogenerierung relevanten Parameter stets klar anzeigen, und neben einer virtuellen Reglersteuerung die Möglichkeit bieten, konkrete Zahlenwerte einzugeben. Sollte der Nutzer beispielsweise einen Oszillator erfordern, welcher in einer spezifischen Hertz-Zahl schwingt, könnte dieser Parameter per Tastatur eingegeben werden. Dies bietet gegenüber realen Modular-Synthesizern zusätzlich den Vorteil, dass der Nutzer darüber entscheiden kann, ob generierte Töne im mikrotonalen Bereich liegen sollen oder nicht. Da reale Modul-Regler in der Regel stufenlos sind, kann es dort schwierig sein, genaue Tonhöhen zu konfigurieren, wodurch erzeugte Töne häufig gezwungenermaßen im mikrotonalen Bereich liegen. Diesem Umstand würde durch eine Zahleneingabe entgegengewirkt werden. Desweiteren sollten virtuelle Ein- und Ausgänge der Module klar gekennzeichnet und beschriftet sein, damit bei der Nutzung intuitiv ersichtlich ist, was eine Verbindung an dem jeweiligen Anschluss erwirkt. Virtuelle Kabelverbindungen sollten möglichst schnell und simpel ermöglicht werden, beispielsweise durch das aufeinanderfolgende Anklicken von einem Ausgang eines Moduls und einem Eingang eines anderen Moduls.



### 2.2.2 Navigationsleiste

Neben der Arbeitsfläche muss dem Nutzer die Möglichkeit gegeben werden, mit dem Synthesizer-Patch übergeordneten Funktionen zu interagieren (z.B. um einen Patch zu speichern oder zurückzusetzen, Tonwiedergabe zu deaktivieren etc.). Hierfür wird eine Navigationsleiste an der Oberseite platziert - dies ist ein Design Pattern, welches sich zum Zeitpunkt dieser Arbeit in einer Vielzahl von Software-Systemen etabliert hat. Die Nutzung von für Nutzer wiedererkennbaren Design Patterns kann nachweislich eine höhere Intuitivität und verminderte Fehleranfälligkeit bei der Benutzung des User Interface zur Folge haben (Maldonado und Jlesnick, 2002). Da der Fokus des Nutzers auf der Interaktion mit den Synthesizer-Modulen liegen soll, sollten andere Interaktionsbereiche wie die Navigation möglichst unintrusiv sein. Dieser Aspekt kann durch das Anwenden von etablierten Design Patterns optimiert werden.

### 2.2.3 Modul-Auswahl

Die letzte Komponente des UI ist die *Modul-Auswahl*. Da die in der Arbeitsfläche vorhandenen Module wie bei einem realen Modular-Synthesizer beliebig austauschbar und in ihrer Anordnung veränderbar sein sollen, wird dem Nutzer über die Modul-Auswahl die Möglichkeit gegeben, neue Module zur Arbeitsfläche hinzuzufügen. Da diese Auswahl perspektivisch nicht zu jedem Zeitpunkt gebraucht wird, sollte sie visuell ausgeblendet werden können. So wird gewährleistet, dass die Arbeitsfläche stets die größtmögliche Fläche einnehmen kann. Über die Darstellung des Titels eines Moduls sowie ein Icon, welches die Funktionalität beschreibt und einen Wiedererkennungswert bietet, kann das Hinzufügen von neuen Modulen für den Nutzer intuitiv gestaltet werden.

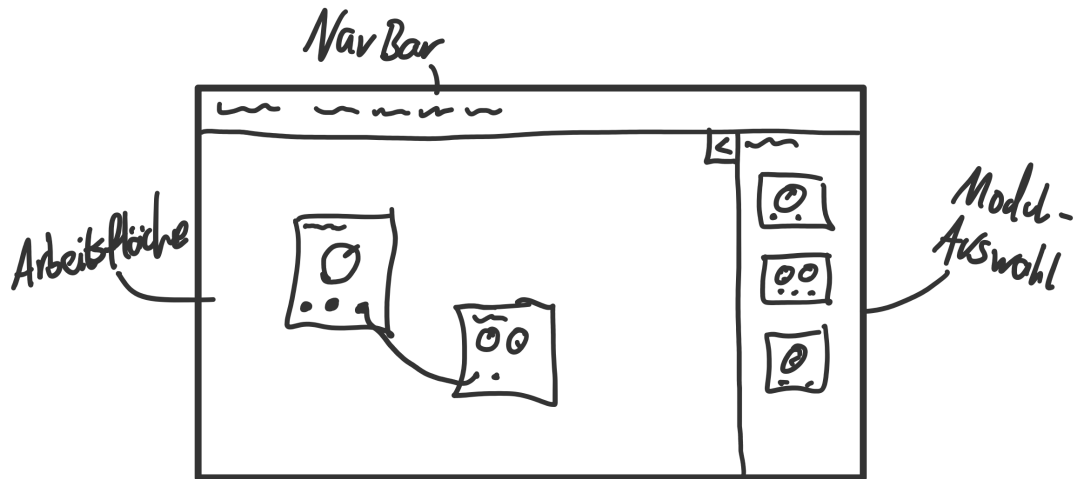


Abbildung 2.1: UI-Sketch für das Frontend bei der Benutzung an einem WIMP-Gerät.

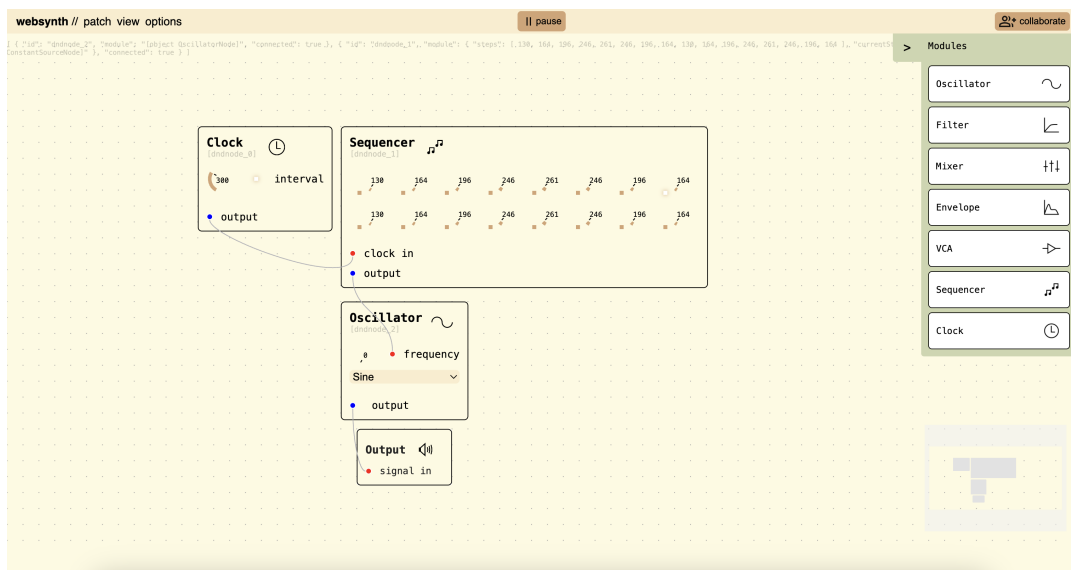


Abbildung 2.2: Abschließend implementiertes Frontend

## 3 Implementierung

Nachdem im vorangegangenen Abschnitt zentrale Erfordernisse und Interaktionsszenarien umrissen wurden, werden diese anschließend als Basis genutzt, das System zu Implementieren. Dabei ist es aufgrund des explorativen Charakters des Projekts möglich, dass erarbeitete Ansätze durch neu entstehende Erkenntnisse begründet angepasst oder ersetzt werden können. Die Implementierungsphase dient somit nicht lediglich als Umsetzung der definierten Systemaspekte, sondern ebenfalls als Quelle neuer Erkenntnisse bezüglich des Forschungsthemas.

### 3.1 Audio-Architektur

Die Wahl der Audio-Architektur ist von zentraler Bedeutung, da alle weiterführenden Elemente des Systems als Schnittstelle und Interaktionsmöglichkeit mit den Audio-Möglichkeiten der Anwendung dienen und durch die Wahl der Audio-Architektur die auditiven Möglichkeiten, Datenstruktur und Funktionen zur Interaktion zwischen Frontend und Audio-Kontext grundlegend vorgegeben werden. Da viele weitere Elemente des Systems an diese Strukturen anknüpfen, wäre eine spätere Änderung der Audio-Architektur perspektivisch mit sehr großem Aufwand verbunden und könnte den Erfolg des Projekts gefährden. Daher sollte eine Entscheidung dieser Natur trotz des explorativen Ansatzes dieser Arbeit stets ausreichend begründet getätigt werden. In diesem Fall wurde die Entscheidung daher anhand des ADR-Formats getätigt und im Git-Repository des Projekts dokumentiert.

Für die Entwicklung der Anwendung fiel die Entscheidung auf die Nutzung der **Web Audio API**. Diese beschreibt zur Zeit dieser Arbeit aus den infrage kommenden Optionen die grundlegendste Möglichkeit, Audio im Web zu erzeugen - die Nutzung von Frameworks, welche auf der Web Audio API eigene Ansätze aufbauen, wurde abgelehnt. Die Entscheidung wurde unter anderem damit begründet, dass die Web Audio API eine Spezifikation der *W3C Audio Working Group* ist (Rogers, 2012), welche Teil des World Wide Web Consortium ist. Somit ist eine hohe Kompatibilität mit verschiedenen Browsern, sowie eine starke unterstützende Gruppierung aus Entwicklern und Anwendern der API zu erwarten. Die Web Audio API wurde darauf

ausgelegt, eine breite Menge an Anwendungsszenarien zu unterstützen, und den Funktionsumfang moderner Audio-Produktionssoftware nachzuempfinden. Somit ist die Web Audio API auf keinen spezifischen Kontext ausgelegt, sondern bietet lediglich grundsätzliche Werkzeuge, welche in Abhängigkeit zu dem Anwendungsfall von den implementierenden Entwicklern genutzt werden können.

Ein zentrales Prinzip der Web Audio API besteht in der Möglichkeit des modularen Routings von Signalketten. Jede Möglichkeit der Klangerzeugung oder -Verarbeitung wird dabei durch *Audio Nodes* umgesetzt, welche miteinander verbunden werden, um eine Signalkette herzustellen. Diese Verbindungen schaffen in der Summe einen *Audio Routing Graph*, welcher bei ständigem Bestehen des Audio-Kontextes flexibel verändert werden kann und Veränderungen der Signalketten unmittelbar in der Audiowiedergabe reflektiert. Dieses Prinzip ähnelt der Interaktion an einem realen Modular-Synthesizer sehr stark (siehe 1.1), wodurch die Web Audio API für die Nutzung in diesem Projekt sehr gut geeignet ist. Während die Routing-Möglichkeiten in einer Vielzahl von anderen Anwendungsfällen lediglich für die Nutzung eines Entwicklers für das Erzeugen von Klängen in einer Web-Anwendung oder einem Spiel vorgesehen sind und unsichtbar für den Nutzer ablaufen, soll der Nutzer des Web-Synthesizers selbst die Kontrolle über das Routing erhalten. Die von der API zur Verfügung gestellten Audio Nodes können in ihrem Funktionsumfang viele Module eines Modular-Synthesizers nachstellen - neben grundsätzlichen Elementen wie Oszillatoren zur Klangerzeugung existieren auch *FilterNodes* zur Klangverarbeitung oder *Gain Nodes*, welche die Funktion eines VCA nachstellen können.

Dabei kann das Ausgangssignal von Modulen nicht nur genutzt werden, um es auditiv wiederzugeben, sondern ebenfalls analog zu dem Prinzip von *Control Voltage* eingesetzt werden. Beispielsweise kann das Ausgangssignal eines Oszillators nicht nur auditiv wiedergegeben werden, sondern auch an den Frequenz-Parameter eines anderen Oszillators gekoppelt werden, sodass die Schwingungsrate des zweiten Oszillators in diesem Fall abhängig von dem Signal des ersten Oszillators ist. Eine gegenseitige Modulation von Parametern zwischen Modulen kann so ermöglicht werden - somit ist eines der zentralen Funktionsprinzipien von Modulersynthesizern ebenfalls nativ in der Web Audio API abgedeckt.

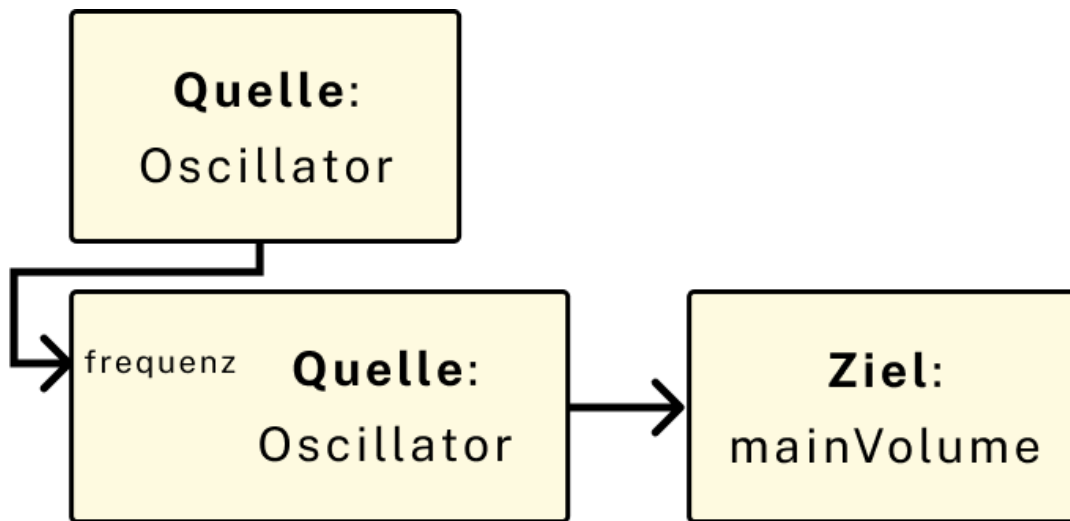


Abbildung 3.1: Beispiel für die Modulation der Frequenz eines Oszillators durch das Signal eines anderen Oszillators.

Wie bei Software für digitale Signalverarbeitung üblich, wird auch in der Web Audio API Sampling genutzt. Die Samplerate, in welcher Audio verarbeitet wird, ist dabei variabel, um die Prozessorauslastung bei schwächeren Prozessoren minimieren zu können und beträgt standardmäßig 44.1kHz (Rogers, 2012). Diese Samplerate hat sich mit der Einführung der CD etabliert und ist ein weitverbreiteter Standardwert (Pras und Gustavino, 2010), welcher gewählt wurde, da mit dieser Samplerate das gesamte Spektrum des menschlichen Gehörs (rund 20Hz-20kHz) abgedeckt werden kann. Daher bietet die Web Audio API in der Theorie keine maßgeblichen Einschränkungen hinsichtlich der Audioqualität, welche im Vergleich zu einem realen Modular-Synthesizer durch den Nutzer merklich wären. Verglichen mit dem analogen Signal eines realen Synthesizers besteht zwar bei digitaler Audioverarbeitung stets ein unvermeidbarer Nachteil in der Audioqualität, doch mit der Samplerate von 44.1kHz ist die Qualität dennoch hoch genug, sodass für den Nutzer in der Regel kein merklicher Unterschied identifizierbar ist. Hinsichtlich des Aspekts der Audioqualität kann die Web Audio API somit als passend zur Nutzung in diesem Projekt gewertet werden.

Zusammenfassend bieten die Werkzeuge der Web Audio API eine ideale Grundlage zur Umsetzung eines Modular-Synthesizers im Web. Viele der grundsätzlich vorhandenen Prinzipien ähneln der Funktionsweise eines realen Modular-Synthesizers sehr stark, sodass Funktionalitäten hinsichtlich des Audio-Kontextes mit wenig Entwicklungsaufwand digital nachgebildet werden können.

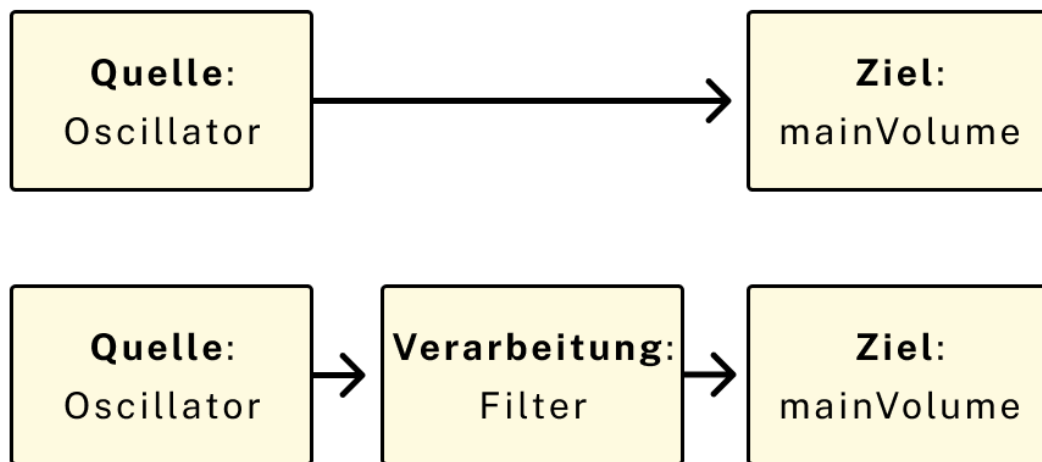


Abbildung 3.2: Zwei Beispiele für simple Signalketten, welche in der Web Audio API umgesetzt werden können.

## 3.2 Aufbau des Frontends

Das Frontend basiert auf einer single page application (SPA), welche in dem JavaScript Frontend-Framework Vue 3 umgesetzt wurde. Die Nutzung von Vue bietet dabei vor allem hinsichtlich des agilen, explorativen Entwicklungsansatzes Vorteile - durch den Komponenten-basierten modularen Aufbau sowie die Möglichkeit, Technologieerweiterungen (zum Beispiel für Zustandsverwaltung) mit geringem Aufwand anzuknüpfen, besteht eine hohe Flexibilität in der Entwicklung. Einzelne Komponenten der Anwendung können so in der Regel verändert werden, ohne dass weitreichende Änderungen in anderen Komponenten der Codebase nötig sind. Für den Aufbau des Frontends wurden diese Vue-spezifischen Paradigmen zum Vorteil des Projekts eingesetzt - so sind Elemente wie die Navigationsleiste oder die Modulauswahl separate Vue-Komponenten, welche in der zentralen Ansicht des Systems referenziert werden. Mehrere Ansichten sind nicht nötig, da alle Optionen und Einstellungen direkt über Menüoptionen der Navigationsleiste verfügbar sind.

### 3.2.1 Zuständigkeitsverteilung der Komponenten

Da in der Anwendung verschiedene Informationskanäle verwaltet werden müssen, ist es sinnvoll, die Logik für die Verwaltung auf verschiedene Komponenten aufzuteilen. Ähnlich zu einer Microservice-Architektur erhält so auch die Anwendungslogik einen

modularen Aufbau, was die Erweiterung oder den Austausch von einzelnen Komponenten in der Zukunft erleichtern kann. Hinsichtlich des explorativen Ansatzes der Implementierung muss damit gerechnet werden, dass solche Fälle eintreten können. Neben der Modularität sorgt die Aufteilung der Zuständigkeitsbereiche zusätzlich für eine bessere Verständlichkeit der Codebase, sollte das Projekt durch einen anderen Entwickler weitergeführt werden. Die Informationskanäle der Anwendung umfassen folgende Dimensionen:

- **Audiokontext:** Audiokomponenten und deren Verbindungen, sowohl untereinander als auch zur Gesamtaudioausgabe des Browsers. In diesem Kontext werden die Module auditiv abgebildet.
- **Arbeitsfläche:** Die verwendeten Module, ihre Einstellungen, Verbindungen und Positionen auf der Arbeitsfläche werden in diesem Kontext abgebildet.
- **Übergeordnete Einstellungen:** Dieser Kanal umfasst übergeordnete Informationen, wie die Zugehörigkeit zu einer Remote-Session, den Play/Pause Status der Wiedergabe, oder visuelle Einstellungen zur Darstellung der Anwendung.

Die Kommunikation zwischen Komponenten und Informationskontexten innerhalb der Applikation geschieht dabei über zwei zentrale Wege; Den Vue-eigenen **Event-Bus** sowie Vuex<sup>1</sup> zum **State-Management**. Welche der beiden Technologien zum Einsatz kommt, hängt dabei davon ab, ob eine Information einmalig übertragen werden muss, oder ob der Status einer veränderlichen Variable für eine andere Komponente dauerhaft zur Verfügung stehen muss. Die Einsatzzwecke waren dabei nicht von Beginn an festgelegt, sondern sind im Verlauf der Implementierung gewachsen und wurden nach Bedarf umgesetzt. So wurde im Verlauf der Implementierung beispielsweise erst durch Nutzung der prototypischen Anwendung deutlich, dass ein Knopf zur Wiedergabe beziehungsweise zum Pausieren der Audioausgabe in der Navigationsleiste sinnvoll ist, damit der Nutzer die Anwendung nicht schließen oder die gesamte System-Lautstärke einstellen muss, um die Audioausgabe zu pausieren. Für den Status der Wiedergabe wurde der neue Boolean-Parameter *playbackHalted* hinzugefügt, welcher durch Klick auf den Knopf seinen Wahrheitswert ändert. Der Audio-Kontext der Applikation kann via State-Management auf diesen Parameter zugreifen und die Audioausgabe pausieren, sollte *playbackHalted* true sein.

---

<sup>1</sup>Vuex-Guide: <https://vuex.vuejs.org>

### 3.2.2 Event-Bus

Der Event-Bus wird in Situationen, in welchen eine einmalige Information übergeben werden muss, genutzt. Dies geschieht beispielsweise, wenn ein Parameter eines Moduls (beispielsweise die Frequenz eines Oszillators) verändert wurde. Via Event-Bus wird der Arbeitsflächen-Komponente, welche die Elternkomponente aller virtuellen Synthesizer-Module ist, mitgeteilt, dass eine Veränderung stattgefunden hat. Die Arbeitsfläche gibt diese Information unmittelbar an den Audio-Kontext weiter, damit diese Änderung auditiv reflektiert werden kann. Diese Verbindung wird auch in die andere Richtung genutzt, beispielsweise wenn zwei Module untereinander eine Information austauschen müssen. Wenn etwa ein Clock-Modul, welches in einem vorgegebenen Intervall ein Trigger-Signal aussendet, an den Trigger-Input eines anderen Moduls angeschlossen ist, wird dem Ziel-Modul über den Event Bus mitgeteilt, dass die interne Logik bei Eingang eines Trigger-Signals nun ausgeführt werden muss.

```
103   eventBus.on( type: "triggerModule", handler: (id) => {  
104       if(id === props.id){  
105           trigger();  
106       }  
107   })
```

Abbildung 3.3: Code aus dem Envelope-Modul, welcher bei einem Trigger-Signal prüft, ob das eigene Modul das Ziel ist. Falls ja, wird die interne Trigger-Funktion ausgeführt.

### 3.2.3 State-Management

Der State-Speicher der Applikation wird für übergeordnete Informationen genutzt, welche sich nicht dauerhaft verändern und permanent relevant für andere Komponenten des Systems sind. Die gespeicherten Informationen umfassen beispielsweise die ID der kollaborativen Session, welcher der Nutzer aktuell angehört, oder die Information, ob die Eingaben der Module als virtueller Drehregler, oder als Textfeld angezeigt werden sollen. Generell werden die in State-Speicher vorhandenen Variablen in der Regel nur durch Nutzeraktionen verändert und dienen nicht zur Hintergrundkommunikation zwischen Komponenten. Eine Ausnahme besteht in dem Play/Pause Status, welcher automatisch bei Aufrufen der Applikation von der Hauptansicht auf „pausiert“ gesetzt wird.



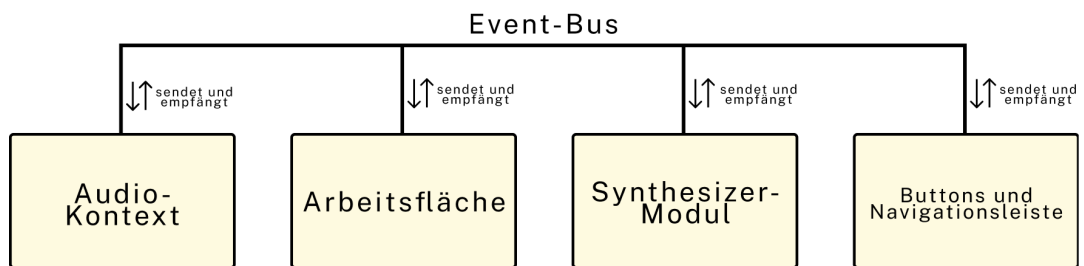


Abbildung 3.4: Übersicht über die Kommunikation via Event-Bus

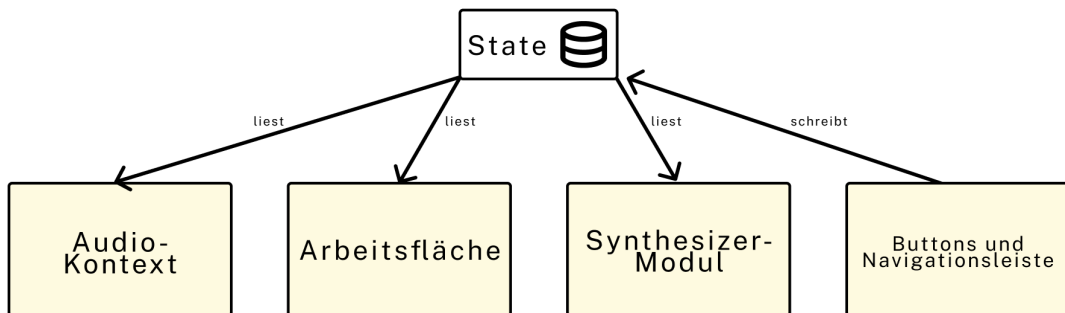


Abbildung 3.5: Übersicht über die Kommunikation via State-Speicher

### 3.2.4 Arbeitsfläche

Um die Arbeitsfläche mit dem konzipierten Funktionsumfang umsetzen zu können, ist eine Lösung notwendig, welche es ermöglicht, Elemente auf einer Leinwand (Canvas) anordnen und verbinden zu können. Dabei sollte die Leinwand die Möglichkeit bieten, Zoom-Level und Position der Ansicht auf die Leinwand zu verändern. Um dies umzusetzen, wurde die Vue-Komponente *Vue Flow*<sup>2</sup> eingebunden. Die Entscheidung, diese Komponente zu nutzen, ist eine der weitreichendsten Technologieentscheidungen, da alle zu implementierenden Module in die von Vue Flow vorgegebene Datenstruktur eingegliedert werden müssen. Somit entstehen Abhängigkeiten - der Funktionsumfang der Anwendung wird maßgeblich an die Möglichkeiten und Grenzen von Vue Flow geknüpft. Dieser Umstand ist hinsichtlich des Zeitrahmens des Projekts aber notwendig, da die eigene Neuentwicklung einer solchen Komponente perspektivisch mit sehr hohem Aufwand verbunden ist. Da der Fokus dieser Arbeit jedoch auf den Audio-Möglichkeiten des Webs liegt, und die Arbeitsfläche lediglich eine Schnittstelle zwischen visuellem und auditivem Kanal bieten soll, ist die Einbindung einer externen Komponente wie Vue Flow angemessen. Gleichzeitig bietet Vue Flow eine große Zahl an nativen Hilfsfunktionen und Erweiterungen, welche die Notwendigkeit eigener Implementierungen und damit verknüpftem Zeitaufwand minimieren können.

### 3.2.5 Verbindung zwischen Frontend und Audio-Kontext

Um Frontend und Audio-Kontext miteinander zu verbinden, wurden zwei Architekturmöglichkeiten identifiziert:

1. Da jedes der Module auf der Arbeitsfläche eine Vue-Komponente ist, könnte das zugehörige Audio-Modul direkt in der Komponente erzeugt und verwaltet werden.
2. Die Audio-Verwaltung geschieht vollständig getrennt, und es wird eine Schnittstelle zwischen Arbeitsfläche und Audio-Kontext implementiert. Die Komponenten der Arbeitsfläche dienen somit lediglich zum Modifizieren von Daten und interagieren nicht selbst auf einem auditiven Kanal.

Primär aufgrund von einer geringeren Komplexität und einfacherer Kommunikation zwischen Audio-Nodes wurde entschieden, dass die Zuständigkeiten strikt in visuellen und auditiven Signal getrennt werden. So stellt die Haupt-Ansicht der Anwendung (*MainView.vue*) alle Funktionen, welche für das Erzeugen von Audioinhalten benötigt

---

<sup>2</sup>Homepage von Vue Flow: <https://vueflow.dev>

sind, zur Verfügung und bindet die Web Audio API ein, während die Arbeitsfläche die visuelle Verwaltung der Module ermöglicht. Da der Nutzer über die Arbeitsfläche mit der Anwendung interagiert, muss ein Architekturansatz entwickelt werden, welcher vorgibt, wie Nutzereingaben verarbeitet und im Audio-Kontext reflektiert werden. Da Vue Flow die auf der Arbeitsfläche vorhandenen Module sowie ihre Verbindungen in einem übergeordneten Array speichert, wurde entschieden, die Module im Audio-Kontext ebenfalls in einem Array zu verwalten, welches die Vue Flow-Module und ihre Verbindungen spiegelt. So kann jedes Modul via einer ID sowohl im Audio-Kontext als auch im Array der Frontend-Komponenten identifiziert und modifiziert werden. Die Daten eines Moduls wie Typ (z.B. Oszillator/Filter/Mixer etc.) sowie vom Nutzer eingetragene Parameter (z.B. Frequenz/Cutoff etc.) können so genutzt werden, um die entsprechende AudioNode zu erzeugen und aus den Eingaben des Nutzers den angeforderten Klang zu erzeugen.

Der Prozess des Übertrags aus Informationen des Frontends in den Audio-Kontext besteht dabei aus zwei Schritten. Im ersten Schritt wird identifiziert, welche Module vorhanden sind, sodass diese im Array der Audio-Komponenten erzeugt werden können. Im zweiten Schritt werden dann die Verbindungen zwischen den Komponenten im Audio-Kontext umgesetzt. Da ein Modul über verschiedene Ein- und Ausgänge verfügen kann, wird die **id**-Eigenschaft der Aus- und Eingangs-Ports der Vue Flow-Module genutzt, um zu unterscheiden, welcher Ausgang eines Moduls an welchen Eingang eines anderen Moduls angeschlossen werden soll. Dieser Prozess ist aus dem Grund zweischrittig, da bei einem nicht vollständig abgeschlossenen Übertrag der auf der Arbeitsfläche vorhandenen Module versucht werden könnte, im Audio-Kontext eine Verbindung zu einem Modul zu erzeugen, welches dort noch nicht erzeugt wurde. Dieser Fehlerfall ist bei der ersten Implementierung des Codes aufgetreten und bietet ein Beispiel für ein Problem, welches bei der Konzeption der Architektur nicht identifiziert wurde. Es wurde erst durch die praktische Umsetzung sichtbar und stellt exemplarisch dar, wie durch den explorativen Ansatz der Umsetzung zwar ungeplante Probleme auftreten, dadurch aber auch neue Erkenntnisse entstehen können. Durch Fälle wie diesen wird deutlich, wie die Implementierungsphase als Werkzeug zum Sammeln von Informationen in einem explorativen Prozess genutzt werden kann.

```

241     case 'oscillator':
242       if(isNewNode){
243         Node = this.audioContext.createOscillator()
244         cvControls = this.buildCVControls(Node, {controls: ["frequency", "detune"]})
245       }
246       if(module.data.frequency && Node.frequency){
247         Node.frequency.value = module.data.frequency
248         Node.type = module.data.waveform
249         cvControls.frequency.gain.setValueAtTime(module.data.cv_in_level, this.audioContext.currentTime)
250         createdNewNode = true
251       }
252       break;

```

Abbildung 3.6: Code zur Erzeugung eines Oszillators im Audio-Kontext durch die Funktion *handleNodeList()*. *case('oscillator')*: bezieht sich dabei auf den Typ des Moduls aus der Vue Flow-Arbeitsfläche.

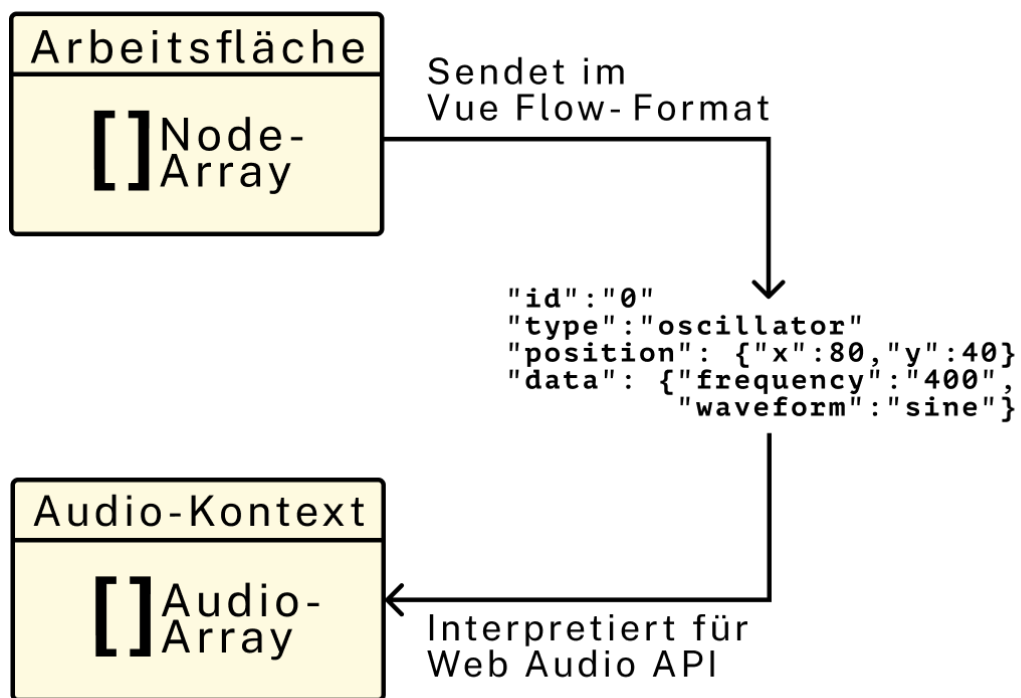


Abbildung 3.7: Übersicht über die Umwandlung von Daten der Arbeitsfläche in Daten des Audio-Kontextes der Web Audio API

### 3.3 Ermöglichen von Kollaboration

Ein zentraler Forschungsaspekt des Projekts liegt in der Frage, wie anhand des entwickelten Systems musikalische Kollaboration in Echtzeit umgesetzt werden kann. Dabei soll erforscht werden, ob es möglich ist, mit den Möglichkeiten des Webs eine Erfahrung zu schaffen, welche die Interaktion nachbildet, welche stattfindet, wenn mindestens zwei Nutzer gleichzeitig an einem realen Modularsynthesizer arbeiten. So könnte trotz der hohen Individualisierbarkeit eines Modularsynthesizers eine ortsunabhängige Kollaboration geschaffen werden, welche den Nachteilen einer physischen Zusammenarbeit an einem realen Synthesizer im Idealfall entgegenwirken kann.

Um die Zusammenarbeit über Distanz zu ermöglichen, wurde mit dem System eine Grundlage für weitere Forschung geschaffen - da alle vorhandenen Module durch das Vue Flow Node-Array der Arbeitsfläche repräsentiert werden können, würde eine Synchronisierung dieses Arrays an mehreren Endgeräten die gewünschte Funktionalität ermöglichen. Auf diese Weise müsste keine direkte Synchronisierung von Elementen des Audio-Kontextes stattfinden, die Verwaltung der Audio-Nodes der Web Audio API könnte direkt auf Basis der Frontend-Komponenten geschehen. So müsste die Schnittstelle zwischen Frontend und Audio-Kontext für die Einbindung von Kollaborationsfunktionen nicht verändert werden. Gleichzeitig bietet sich die Synchronisierung der Vue Flow-Elemente an, da Vue Flow Veränderungen von Elementen nativ als *Change*-Objekttyp verwaltet. Da Vue Flow in *Change*-Objekten ausschließlich Inhalte emittiert, welche eine Änderung erfahren haben, und diese *Change*-Objekte auch entgegen nehmen kann, um das Array der bestehenden Elemente zu aktualisieren, müssten Frontend-Module nicht in ihrer Gänze ausgetauscht werden. So reicht es, wenn bei der Aktion eines Nutzers lediglich das *Change*-Objekt an den Client des anderen Nutzers gesendet, und dort verarbeitet wird. So würden lediglich die Arrays der Frontend-Komponenten der Arbeitsfläche zwischen Sessions synchronisiert werden, die Verarbeitung der Frontend-Module in Module der Web Audio API würde jeder Client selbst übernehmen.

Um dies umzusetzen, wird ein WebSocket-Server genutzt. In diesem Kontext ist die Nutzung einer Peer-to-Peer-Verbindung unvorteilhaft, da es sinnvoll ist, eine *Single Source of Truth*, also im Fall dieser Anwendung das gesamte Array der Patch-Konfiguration persistent zu speichern. Da lediglich Veränderungen gesendet werden, müssen beide Clients auf dem gleichen Datenstand sein, bevor diese Art der Kommunikation funktionieren kann. Andernfalls könnte der zweite Client Veränderungsanweisungen für ein Modul erhalten, welches in dem lokalen Array an Elementen noch nicht vorhanden ist. Wenn der Server über einen Datenstand aller Elemente verfügt, kann dieser bei Etablieren der Verbindung zu einem neuen Client gesendet werden, sodass der Client

diesen übernimmt. Anschließend werden alle weiteren Aktionen lediglich über Vue Flow Change-Objekte abgewickelt, was die transportierten Datengrößen minimiert und dadurch perspektivisch insgesamt die Latenz minimieren kann. Eine niedrige Latenz bei der Synchronisierung zwischen Sessions ist besonders wichtig, um die Zusammenarbeit mit mehreren Nutzern effizient und problemlos zu ermöglichen.

### 3.3.1 Vermitteln von virtuellem Besitz und Territorium

(Fencott und Bryan-Kinns, 2013) beschreiben, dass Nutzer ein klares Verständnis über Besitz, Territorium und Privatsphäre erhalten müssen, sobald sie in einem Software-System auf die musikalischen Beiträge der jeweils anderen Nutzer zugreifen und diese theoretisch modifizieren können. Andernfalls können Konfliktsituationen entstehen, beispielsweise, wenn mehrere Nutzer zeitgleich auf eine Komponente zugreifen und Änderungen vornehmen wollen. Da das Entstehen solcher Konfliktsituationen in dem entwickelten System nicht unwahrscheinlich ist, bedarf es einer Lösung, den Nutzern bei der kollaborativen Nutzung ein Gefühl von Besitz über die Module zu vermitteln. Bei einem realen Synthesizer würde ein Regler, welcher gerade durch die Hand einer Person bedient wird, mit hoher Wahrscheinlichkeit nicht von einer zweiten Person genutzt werden, da visuell eindeutig ist, welche Person in diesem Moment „Besitz“ über den Regler ergreift. Da der Mauszeiger als virtuelle Repräsentation der Hand des Nutzers interpretiert werden kann, wurde der Ansatz gewählt, die Position der Mauszeiger aller Nutzer auf der Arbeitsfläche darzustellen. Sollte sich ein Mauszeiger über einem Regler befinden, wird visuell deutlich, dass der Regler gerade von einem anderen Nutzer verwendet wird. So können soziale und technische Konfliktsituationen vermieden werden.

Diese Funktion wurde mithilfe der Funktion *screenToFlowCoordinate* umgesetzt, welche nativ in Vue Flow genutzt werden kann, um die Position des Mauszeigers im Kontext der Arbeitsfläche auszugeben. Um die Darstellung von Mauszeigern nahtlos in die bisherigen Datenstrukturen einzubinden, wurde der Mauszeiger anderer Nutzer als eine Vue-Komponente umgesetzt, welche als Vue Flow-Node auf der Arbeitsfläche existiert. So kann die bisherige Logik zum Aktualisieren von Node-Daten und -Positionen für das Verwalten von Cursor-Positionen genutzt werden. Dabei wurde eine Debounce-Funktion mit einer Dauer von 10ms genutzt, um die Anzahl der ausgehenden Moduländerungen zu reduzieren und keinen unnötig hohen Datenverkehr an den Websocket-Server zu verursachen. Das Cursor-Objekt des eigenen Nutzers existiert aufgrund dieser Architekturentscheidung zwar auch stets im lokalen Frontend, wird aber über ein *v-if*-Direktiv ausgeblendet, damit Nutzer ihren eigenen Mauszeiger nicht doppelt sehen.

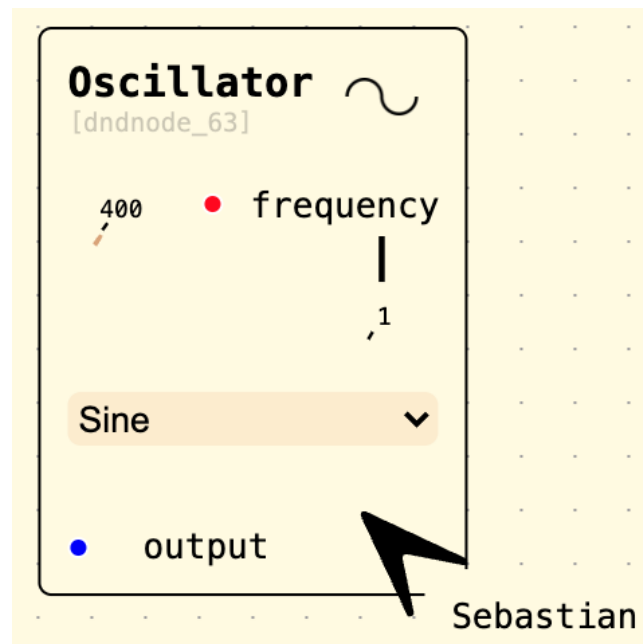


Abbildung 3.8: Der Cursor eines Kollaborateurs signalisiert beim Verschieben eines Moduls, welcher Nutzer aktuell Besitz über das Modul ergreift

### 3.3.2 Etablieren von Sessions

Durch den Umstand, dass ein Client, welcher einer bestehenden Session beitrifft, zuerst das gesamte Array erhalten muss, bevor die Kommunikation lediglich über Change-Objekte funktionieren kann, stellt sich eine zentrale Frage für den Ablauf der Websocket-Kommunikation; Wie und wann erhält der Websocket-Server das gesamte Array, um es an den neuen Client weiterzugeben? Würde das gesamte Array bei jeder Änderung durch einen Client mitgesendet werden, würde die Menge an gesendeten Daten immens steigen, und die Vorteile der Datenminimierung durch das Senden von Change-Objekten würden verloren gehen. Dieses Problem konnte mithilfe der vergleichsweise großen Freiräume, welche durch die Anwendung eines explorativen Ansatzes entstehen, behoben werden. Durch das Einbinden einer vollständig neuen Funktionalität, welche zuvor nicht konzipiert wurde, kann das Senden des gesamten Patches auf einen spezifischen Zeitpunkt festgelegt werden. So wurde aufgrund der Problemstellung der Beitritt zu einer bestehenden Session erweitert, sodass bereits beigetretene Clients im Frontend eine Aufforderung erhalten, den neuen Client anzunehmen oder abzulehnen. In dem Moment, in welchem der neue Client angenommen wird, wird zeitgleich mit der Entscheidung des Nutzers das gesamte Array

der Arbeitsflächen-Module im JSON-Format mitgesendet. Der Server kann dieses Array an den beitretenden Client weitersenden, sodass dessen Patch mit diesen Daten überschrieben wird. Auf diese Weise ist garantiert, dass beide Clients auf dem gleichen Stand sind, bevor die zukünftige Kommunikation über Change-Objekte abgewickelt wird.

Anhand dieses Problems und der zugehörigen Lösung kann exemplarisch aufgezeigt werden, welche Vor- und Nachteile ein explorativer Ansatz bieten kann. Da das Problem erst mit der Entwicklung und daraus entstehenden Technologieentscheidungen aufgetreten ist, war ungeplanter Konzeptions- und Entwicklungsaufwand erforderlich, um das Problem zu lösen. Durch eine hohe Handlungsfreiheit in der Umsetzung und die Möglichkeit, infrage kommende Ansätze durch prototypische Einbindung zu testen, konnte für das Problem jedoch eine Lösung gefunden werden, welche durch den unmittelbaren, praktischen Einsatz im System ihre Effektivität mit einer hohen Sicherheit beweisen kann.

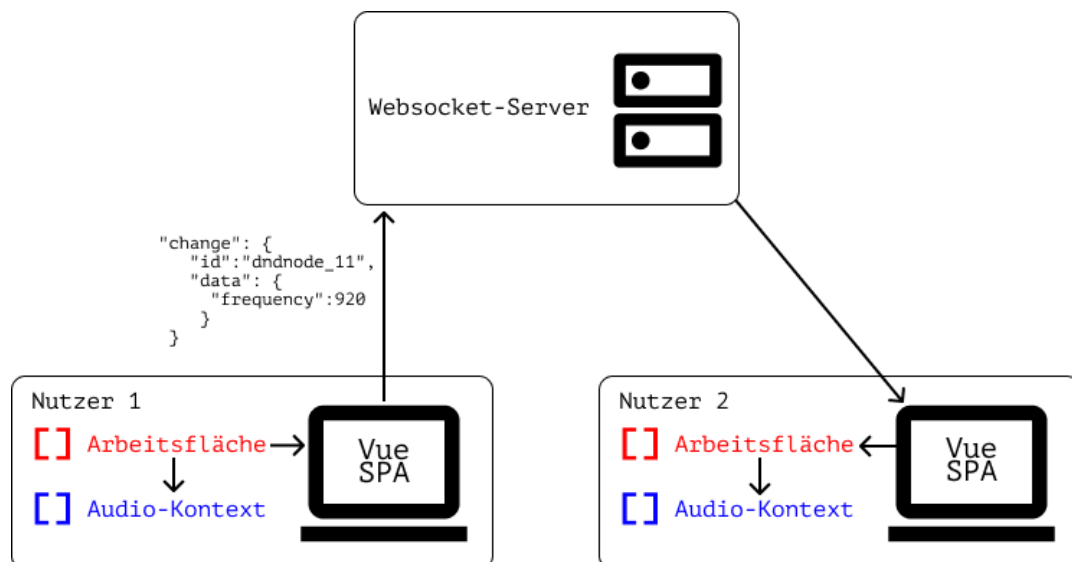


Abbildung 3.9: Übersicht über den Kommunikationsweg einer einzelnen Veränderung, ausgehend von einem Change-Objekt der Arbeitsfläche von Nutzer 1



## 4 Herausforderungen

Im Verlauf der Umsetzung des Systems sind Herausforderungen durch Umstände aufgetreten, welche in der ursprünglichen Konzeption oder der ersten Version der Implementation nicht berücksichtigt wurden, und beispielsweise durch Fehlerfälle bei bestimmten Anwendungssituationen sichtbar wurden. Fehlerfälle wurden dabei nach dem Auftreten stets via Issues im Projekteigenen Git-Repository dokumentiert, welche mit der Angabe des Commits, durch welchen der Fehler behoben wurde, geschlossen wurden. Somit ist auch bei unerwartet auftretenden Herausforderungen eine lückenlose Dokumentation vom Auftreten bis zur Lösung eines Problems möglich, sodass das Potenzial zur Sammlung von Erkenntnissen in der Implementierungsphase durch den explorativen Ansatz der Arbeit größtmöglich genutzt werden kann. Im Folgenden wird eine Auswahl der aufgetretenen Herausforderungen diskutiert.

### 4.1 Umsetzung von Trigger-Signalen

Module wie ein Sequencer oder ein Hüllkurvengenerator (ADSR/Envelope) benötigen ein Eingangssignal, welches bei Auftreten eine interne Aktion initiiert, zum Beispiel das Iterieren des aktiven Schrittes eines Sequencers. Diese Signale werden auch Trigger-Signale genannt und sind bei realen Synthesizern als kurze elektrische Impulse vorhanden. Die Web Audio API bietet keine native Möglichkeit, Trigger-Signale zu erzeugen oder zu identifizieren. Audiosignale könnten zwar genutzt werden, um eine solche Funktionalität umzusetzen, doch durch diesen Ansatz würde zusätzlicher Verarbeitungsaufwand für ein Audiosignal entstehen, welches lediglich zur Kommunikation einer punktuell auftretenden Information genutzt werden würde. Bei der Umsetzung des Clock-, Sequencer und Envelope-Moduls wurde daher bewusst ein Kommunikationskanal außerhalb der Web Audio API gewählt, um Trigger-Signale umzusetzen. So wird der Event-Bus der Vue-Applikation genutzt, um Trigger-Signale zu übermitteln (siehe Abbildung 3.3). Diese Signale werden von den Frontend-Komponenten der Arbeitsfläche entgegen genommen - entsprechende Änderungen werden dann wie bei einer regulären Änderung durch den Nutzer zur Verarbeitung im Audio-Kontext weitergegeben. Der Umstand, dass die Web Audio API keine eigene Lösung für Trigger-Signale bereitstellt, ist zu Beginn der Implementierung noch nicht bekannt gewesen, sodass der Aufwand zur Umsetzung dieser Lösung unerwartet hoch war.

Die umgesetzte Lösung mithilfe des Event-Bus bildet zwar ein Abweichen zu der bisher verwendeten Architektur, da die Verbindung eines Trigger-Signal-Kanals durch diese Lösung lediglich im Frontend-Kontext vorhanden ist, es bietet jedoch ebenfalls den Vorteil, dass für das Ändern eines Status, welcher im Frontend reflektiert werden soll (z.B. die Anzeige, welcher Schritt des Sequencers aktuell aktiv ist), kein Weg vom Audio-Kontext zurück in die Frontend-Komponente des Moduls geschaffen werden muss, wodurch die erforderlichen Schritte und auszuführenden Funktionen insgesamt reduziert sind. Die so umgesetzte Lösung funktioniert in der Praxis und erreicht die geplante Funktionalität.

## 4.2 Nutzung von Audio- und CV-Signalen

Mit einer wachsenden Zahl von Modulen sind auch die Möglichkeiten, Module untereinander anzuschließen, gewachsen. Einige Module haben Eingänge zum Steuern von Parametern via Control Voltage-Signalen erhalten. Die Nutzung solcher Verbindungen, beispielsweise zur Kontrolle der Frequenz eines Oszillators, wird dabei nativ von der Web Audio API unterstützt, sodass der Audio-Output eines anderen Oszillators als CV-Signal dienen kann. Diese Verbindung kann hergestellt werden, indem die *connect()*-Funktion der Web Audio API angewandt wird, und dabei als Ziel kein anderes Modul, sondern der Parameter des zu steuernden Moduls angegeben wird. Grundsätzlich funktioniert diese Einbindung, doch sind durch die Implementierung in zwei Aspekten Probleme entstanden.

Durch die direkte Verbindung eines Oszillator-Outputs an den Frequenz-Parameter eines anderen Oszillators wird das Signal zwar beeinflusst, jedoch nur so schwach, dass ein Unterschied kaum merklich ist. Aus diesem Grund verfügen reale Modular-Synthesizer in der Regel über einen Regler, welcher kontrolliert, wie stark ein CV-Signal auf den Parameter einwirkt. Um diesen Regler in der Anwendung umzusetzen, wurde ein für den Nutzer unsichtbares Gain-Modul hinzugefügt, welches Permanent mit dem Frequenz-Parameter des Zielmoduls verbunden ist - so wird das CV-Eingangssignal eines anderen Moduls an dieses Gain-Modul angeschlossen und kann anschließend durch dieses in der Intensität gesteuert werden, bevor das Signal die Parameter-Steuerung des Moduls erreicht. Da die Intensität über die Kontrolle des Gain-Moduls durch diesen Ansatz ausreichend hoch eingestellt werden kann, konnte das Problem durch die Einbindung der Gain-Module behoben werden.

Ein weiteres Problem entstand, da nicht alle umgesetzten Module zur virtuellen Realisierung von CV-Signalen Signale aussenden, welche vergleichbar mit dem Aus-

gang eines Oszillators sind, sondern direkt auf den Parameter des Zielmoduls zugreifen. Das Envelope-Modul etwa gibt kein Audiosignal aus, sondern greift auf den Zielparameter zu, an welches es angeschlossen ist, und verändert den Wert über die Funktion *linearRampToValueAtTime()* direkt. Würde das Envelope-Modul auf das dem CV-Eingang zwischengeschalteten Gain-Modul zugreifen, würde sich an der Frequenzeinstellung des Ziel-Oszillators nichts ändern. Aus diesem Grund wird jedem Modulausgang, welches ein Audiosignal aussendet diese Eigenschaft via dem **id**-Feld des Vue Flow-Ausgangsports zugewiesen. Wenn die Verbindung zu dem Parameter eines Moduls dann im Audiokontext hergestellt werden soll, kann in diesem Fall an das Gain-Modul verbunden werden. Handelt es sich andernfalls um ein CV-Signal, wird der Parameter direkt verbunden.

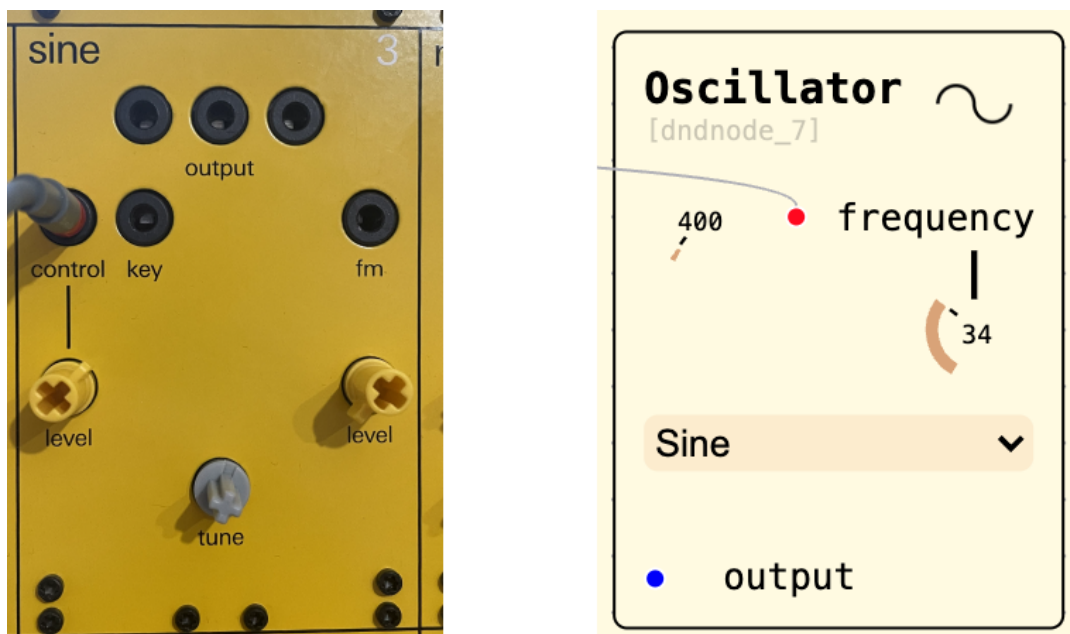


Abbildung 4.1: Sinus-Oszillator-Modul eines Teenage Engineering PO Modular 400 neben dem Oszillator-Modul des Web-Systems, jeweils mit eingehendem CV-Signal zur Steuerung der Frequenz und angepasster Intensität des CV-Signals.

### 4.3 Optimierung der Schnittstelle Frontend/Audio

Wie in Abschnitt 3.2.5 beschrieben, interagiert der Nutzer stets auf der visuellen Ebene mit dem Synthesizer, sodass die dort vorgenommenen Aktionen anschließend im Audio-Kontext übertragen werden können. In der ersten Implementation wurde dabei das vollständige Array aus Vue Flow-Komponenten übertragen. Anschließend konnte für jedes der Elemente das entsprechende Modul im Audio-Kontext erstellt oder geändert werden. Dabei wurde bei jeder Änderung eines Moduls der Arbeitsfläche stets das gesamte Array weitergegeben und verarbeitet. Neben dem Fakt, dass durch diese Umsetzung Module verarbeitet werden, welche keine Veränderung erfahren haben und dadurch vermeidbarer Rechenaufwand entstand, sind durch diesen Ansatz weitere Fehlerfälle entstanden. So etwa, dass durch das Bearbeiten eines Wertes eines beliebigen Moduls die Frequenz von einem im Patch vorhandenen Oszillator kurzzeitig auf den im Frontend-Modul konfigurierten Wert springt, obwohl ein CV-Eingangssignal eine andere Frequenz definiert. Dieser Fehler entstand, da der Oszillator erneut verarbeitet wurde, obwohl keine Änderung in diesem Modul vorlag - so wurde die im Frontend-Modul definierte Frequenz im Oszillator übernommen, bis die Frequenz unmittelbar im Anschluss durch das CV-Eingangssignal wieder einen anderen Wert erhielt.

Aufgrund des vermeidbaren Rechenaufwands und den beschriebenen Fehlerfällen wurde die Struktur grundsätzlich angepasst. So wird bei einer Änderung in einem Modul durch die neue Implementierung definiert, wie die ID des Moduls lautet, welches bearbeitet wurde. Anschließend wird nur dieses Modul im Audio-Kontext verarbeitet. Das Ergebnis ist eine Logik, welche die bei der Nutzung des Systems ausgeführten Anweisungen erheblich reduziert - die eingesparte Menge an Rechenleistung steigt dabei linear zur Anzahl der vorhandenen Module auf der Arbeitsfläche.

Das Filtern nach Modul-ID bei der Verarbeitung der Module im Audio-Kontext beschreibt den auf der ursprünglichen Methode logisch aufbauenden Schritt und beschreibt eine Erweiterung des Codes, nicht aber eine Neufassung. So konnte durch die anfangs implementierte, weniger performante Lösung mit minimalem Entwicklungsaufwand getestet werden, ob der Ansatz zum Übertragen der Module in den Audio-Kontext grundsätzlich funktioniert, bevor diese Logik zwecks der beschriebenen Optimierungen angepasst werden konnte. Diese Situation kann als geeignetes Beispiel für die Vor- und Nachteile eines explorativen Ansatzes dienen, da der Problemfall bei einer vorherigen vollständigen Konzeption der Logik mit einer hohen Wahrscheinlichkeit nicht aufgetreten wäre. Andererseits konnte durch die praktische Implementierung der ersten Version der Logik faktisch bewiesen werden, dass der gewählte Ansatz grundsätzlich funktioniert - eine Tatsache, welche auch bei einer vollständigen theoretischen Konzeption nicht mit einer vergleichbar hohen Sicherheit planbar wäre.

## 4.4 Gestiegener Datenverbrauch des Websockets durch neue Module

Teilweise sind im Entwicklungsprozess durch die Einbindung der Websocket-Verbindung zum kollaborativen Arbeiten Probleme entstanden, welche bei alleiniger Nutzung durch eine Person nicht aufgetreten wären. Diese Probleme beschreiben eine neue Dimension an Herausforderungen, welche teils Veränderungen der bisherigen Daten- und Kommunikationsstruktur nach sich zogen. So ist durch die Einbindung des Clock- und Sequencer-Moduls ein neues Problem aufgetreten, welches durch die vorangegangenen entwickelten Module nicht entstand. Da der aktuell aktive Schritt des Sequencers in dem Datenobjekt des Frontend-Elements (*node.data*) gespeichert wird, und bei Veränderung des Datenobjekts ein Change-Objekt des Moduls an den Websocket-Server gesendet wird, wurde bei jedem Schritt des Sequencers eine Nachricht an andere Clients der zugehörigen Websocket-Session gesendet. Da der Taktgeber dieser Änderungen, das Clock-Modul, allerdings auch im Frontend der anderen Clients vorhanden ist, stellen diese Nachrichten einen vermeidbaren Datenverbrauch dar. Der bisherige Mechanismus, um Veränderungen an einem Modul zu identifizieren, war für diesen spezifischen Fall daher ungeeignet.

Dieses Problem konnte gelöst werden, indem innerhalb des Sequencer-Moduls bei einer Änderung unterschieden wird, welche Eigenschaft der Moduldaten sich verändert hat. Sollte sich eine der für die Schritte definierten Frequenzen verändert haben, wird dabei unverändert das *moduleChanged*-Event emittiert, welches die Änderung bei einer aktiven kollaborativen Session an den Websocket-Server sendet. Bei Inkrementieren des aktuellen Schrittes wird dagegen das neue Event *advanceStep* emittiert, welches eine Verarbeitung der Änderungen im Audio-Kontext anstößt, diese Änderung aber nicht an den Websocket-Server sendet.

## 5 Diskussion

Im Folgenden wird das abschließend entwickelte System hinsichtlich der definierten Anforderungen kritisch eingeordnet, sowie betrachtet, inwieweit die Forschungsfragen der Arbeit durch die Entwicklung des Systems beantwortet werden konnten.

### 5.1 Audiogenerierung und Interaktion

Mit Einbindung der Web Audio API wurde ein etablierter Standard des Webs aufgezeigt, welcher annähernd universell genutzt werden kann, um in einem Web-Browser Klänge zu erzeugen. Dabei konnten durch die Nutzung dieser Technologie folgende Funktionalitäten eines physischen Modular-Synthesizers erfolgreich nachgebildet werden:

- In ihrer Funktion logisch abgetrennte Module, welche Ein- und Ausgänge bereitstellen, um Signalketten zu bilden.
- Freie Wahl der Anzahl und Anordnung von Modulen in einem Patch
- Signalerstellung und -Verarbeitung durch für Modular-Synthesizer maßgeblich wichtige Module, darunter Oszillatoren, Filter oder Mixer
- Nutzung von Audio- und Control-Voltage-Signalen
- Freie Verbindung von Modulen, um beliebige Signalketten zu erschaffen und eine gegenseitige Modulation der Module zu ermöglichen
- Taktgebung und Personenunabhängiger Zustandswechsel durch Nutzung von beispielsweise Clock- und Sequencer-Modulen möglich
- Möglichkeit, durch MIDI-Modul externe Hardware anzuschließen und in den Patch einzubinden

Da die aufgeführten Funktionen die grundsätzlichen Interaktionsmöglichkeiten eines Modulerssynthesizers erfolgreich abbilden können, und diese durch die Einbindung in die in dieser Arbeit entwickelte Anwendung in der Praxis getestet sind, kann die Schlussfolgerung gezogen werden, dass die Nachbildung eines Modulerssynthesizers im Web realisierbar ist. Durch die Synchronisierung von Sessions anhand Websockets

konnte dabei eine mögliche Architektur aufgezeigt werden, welche eine Kollaboration über Distanz auf Basis der Web Audio API ermöglicht. Hinsichtlich der Audioqualität wird durch den Unterschied zwischen analoger und digitaler Audiogenerierung stets ein unvermeidbarer Qualitätsunterschied zum Nachteil von digitalen Systemen bestehen, doch durch die Samplerate von 44.1kHz, welche theoretisch angehoben werden könnte, ist dennoch eine Qualität gegeben, welche dem Branchenstandard entspricht und somit als ausreichend bewertet werden kann.

Die Realisierung der Arbeitsfläche durch Vue Flow konnte die an einem realen Synthesizer möglichen Interaktionen erfolgreich nachstellen - Module können frei angeordnet werden, die Eingabe von Parametern ist über virtuelle Regler möglich, und virtuelle Kabelverbindungen können zwischen Ein- und Ausgängen von Modulen hergestellt werden. Daher kann bewertet werden, dass die Interaktion an einem Modular-Synthesizer erfolgreich im Web nachgebildet werden kann.

Der Ansatz zur Synchronisierung von Frontend-Aktionen mit dem lokalen Audio-Kontext kann ebenfalls als erfolgreich gewertet werden, da in der Nutzung des Systems alle Aktionen auf der Arbeitsfläche des Systems unmittelbar auditiv umgesetzt werden. Die klare Abgrenzung der Zuständigkeiten in Frontend- und Audio-Kontext konnte dabei hinsichtlich der Entwicklung die erwarteten Vorteile bieten und eine hohe Übersichtlichkeit über die Verwaltung von virtuellen Synthesizer-Modulen in der Anwendungslogik gewährleisten.

## 5.2 Kollaboration und Latenz

Experimente mit Versuchspersonen haben belegt, dass es ungefähr ab einer Verzögerung des Audiosignals von 25ms schwierig für Kollaborateure wird, in einem musikalischen Kontext das vorherrschende Tempo beizubehalten und ihre Aktionen zu synchronisieren (Rottondi et al., 2016). Die Höhe der Verzögerung, bis bei kollaborativen Prozessen die Änderung eines Nutzers in der Session eines anderen Nutzers auftritt, wurde mit der umgesetzten Software unter verschiedenen Bedingungen getestet, um die praktische Anwendbarkeit hinsichtlich der kritischen Grenze von 25ms zu untersuchen. So bezog ein Client die Internetverbindung über einen VPN-Dienst in Atlanta, USA, während der zweite Client die Internetverbindung ohne VPN-Anbindung bezog. So konnte ein Signalweg geschaffen werden, welcher das Testen eines theoretischen worst-case-Szenarios ermöglicht, in welchem eine hohe örtliche Trennung zwischen den Nutzern vorliegt. Dabei wurde ermittelt, dass die Latenz durchschnittlich 444ms betrug (siehe Tabelle 1 des Anhangs). In einem Test mit identischem Versuchsaufbau, ohne, dass einer der beiden Computer über eine VPN-Verbindung mit dem Internet

verbunden ist, ergab sich eine durchschnittliche Latenz von 40ms (siehe Tabelle 2).

Da die Ergebnisse eine Differenz von einem Faktor von rund 10 aufweisen, zeigen sie auf, dass Herausforderungen durch die geographischen Standorte der Nutzer in dem umgesetzten System nach wie vor bestehen und nicht vollständig behoben werden konnten. Desweiteren überschreitet die Latenz von 40ms selbst in einem Idealszenario die kritische Grenze von rund 25ms deutlich. Da modulare Synthesizer allerdings von herkömmlichen Instrumenten in dem Sinne abzugrenzen sind, dass in der Regel keine synchronen Aktionen von mehreren Nutzern nötig sind (Die zeitliche Synchronisation von Aktionen kann beispielsweise durch ein Clock-Modul umgesetzt werden), sondern lediglich ein System zur Erstellung von Klängen angeordnet und „programmiert“ wird, ist fraglich, ob die vergleichsweise hohen Latenzwerte eine Zusammenarbeit maßgeblich erschweren.



## 6 Fazit und Ausblick

Die vorliegenden Ergebnisse zeigen auf, dass die grundlegenden Funktionen eines physischen Modularsynthesizers erfolgreich in ein webbasiertes System übertragen werden können. Die durchgeführten Experimente haben jedoch auch auf technische Herausforderungen hingewiesen, insbesondere hinsichtlich der Latenzzeiten bei kollaborativen Prozessen über große Distanzen.

Für zukünftige Arbeiten eröffnet dies mehrere Forschungsrichtungen: Zum einen könnte die Optimierung der Latenzzeiten durch minimierte Nachrichtengrößen, verbesserte Netzwerkprotokolle und Server-Architekturen weitergeführt werden. Dabei könnte auf einer sozialen Ebene anhand des Systems erforscht werden, inwieweit sich die von Nutzern wahrgenommene kritische Latenzzeit von herkömmlichen Instrumenten unterscheidet. Zusätzlich könnte die Nutzererfahrung und Akzeptanz in verschiedenen Anwendungsszenarien detaillierter analysiert werden, um die Bedienbarkeit und den praktischen Nutzen des Systems zu maximieren. Eine weitere Möglichkeit besteht darin, die Plattform für andere Formen musikalischer Kollaboration zu adaptieren und die Ergebnisse auf weitere Bereiche der digitalen Musikproduktion zu übertragen.

Zusammenfassend lässt sich sagen, dass die entwickelte Lösung ein vielversprechender Ansatz zur webbasierten Nachbildung von Modularsynthesizern ist. Durch den explorativen Forschungsansatz des Projekts konnten Ergebnisse und Potenziale aufgezeigt werden, welche eine solide Basis für weiterführende Forschung und Entwicklung in diesem Bereich darstellen.

## Literatur

- Auricchio, N., & Borg, P. (2016). New modular synthesizers and performance practice. <https://repository.uwl.ac.uk/id/eprint/2963/>
- Enders, B. (1985). *Die Klangwelt des Musiksynthesizers - Die Einführung in die Funktions- und Wirkungsweise eines Modulsynthesizers*. epOs-Verlag Osnabrück.
- Fencott, R., & Bryan-Kinns, N. (2013). Computer Musicking: HCI, CSCW and Collaborative Digital Musical Interaction. In S. Holland, K. Wilkie, P. Mulholland & A. Seago (Hrsg.), *Music and Human-Computer Interaction* (S. 189–205). Springer London. [https://doi.org/10.1007/978-1-4471-2990-5\\_11](https://doi.org/10.1007/978-1-4471-2990-5_11)
- Healey, P., Leach, J., & Bryan-Kinns, N. (2005). InterPlay: Understanding Group Music Improvisation as a Form of Everyday Interaction.
- Jenkins, M. (2019). *Analog Synthesizers: Understanding, Performing, Buying: From the Legacy of Moog to Software Synthesis*. Routledge. <https://doi.org/10.4324/9780429453991>
- Maldonado, C. A., & Jlesnick, M. L. (2002). Do Common User Interface Design Patterns Improve Navigation? *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 46(14), 1315–1319. <https://doi.org/10.1177/154193120204601416>
- Pras, A., & Gustavino, C. (2010). sampling rate discrimination: 44.1 khz vs. 88.2 khz. *journal of the audio engineering society*, (8101).
- Rogers, C. (2012). *Web Audio Api W3C Editor's Draft*. <https://web.archive.org/web/20120720115514/https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html>
- Rottondi, C., Chafe, C., Allocchio, C., & Sarti, A. (2016). An Overview on Networked Music Performance Technologies. *IEEE Access*, 4, 8823–8843. <https://doi.org/10.1109/ACCESS.2016.2628440>
- Schüll, E. (2009). Zur Forschungslogik explorativer und normativer Zukunftsforschung. In R. Popp & E. Schüll (Hrsg.), *Zukunftsforschung und Zukunftsgestaltung: Beiträge aus Wissenschaft und Praxis* (S. 223–234). Springer Berlin Heidelberg. [https://doi.org/10.1007/978-3-540-78564-4\\_16](https://doi.org/10.1007/978-3-540-78564-4_16)
- Veretekhina, S. V., Zhuravlyov, M. S., Shmakova, E. G., Soldatov, A. A., Kotenev, A. V., Kashirin, S. V., & Medvedeva, A. V. (2018). Analog sound signals digitalization and processing. *www. mjltm. com info@ mjltm. org*, 40.

White, A. (2022). Unstable Structure: The improvising modular synthesiser. *Organised Sound*, 27(2), 182–192. <https://doi.org/10.1017/S1355771821000595>

## Anhang

| A: Timestamp Senden | B: Timestamp Empfang | Latenz |
|---------------------|----------------------|--------|
| 13:50:26.512        | 13:50:26.950         | 438ms  |
| 13:50:26.764        | 13:50:27.192         | 428ms  |
| 13:50:26.818        | 13:50:27.282         | 464ms  |
| 13:50:26.897        | 13:50:27.345         | 448ms  |
| 13:50:27.040        | 13:50:27.482         | 442ms  |
| 13:50:27.213        | 13:50:27.653         | 440ms  |
| 13:50:27.331        | 13:50:27.751         | 420ms  |
| 13:50:27.434        | 13:50:27.875         | 441ms  |
| 13:50:27.584        | 13:50:28.016         | 432ms  |
| 13:50:27.702        | 13:50:28.142         | 440ms  |
| 13:50:27.774        | 13:50:28.200         | 426ms  |
| 13:50:27.813        | 13:50:28.259         | 446ms  |
| 13:50:27.860        | 13:50:28.299         | 439ms  |
| 13:50:27.908        | 13:50:28.344         | 436ms  |
| 13:50:27.994        | 13:50:28.426         | 432ms  |
| 13:50:28.081        | 13:50:28.533         | 452ms  |
| 13:50:28.144        | 13:50:28.579         | 435ms  |
| 13:50:28.263        | 13:50:28.710         | 447ms  |
| 13:50:28.351        | 13:50:28.774         | 423ms  |
| 13:50:28.429        | 13:50:28.866         | 437ms  |
| 13:50:28.476        | 13:50:28.914         | 438ms  |
| 13:50:28.579        | 13:50:29.026         | 447ms  |
| 13:50:28.713        | 13:50:29.154         | 441ms  |
| 13:50:28.855        | 13:50:29.291         | 436ms  |
| 13:50:28.947        | 13:50:29.389         | 442ms  |
| 13:50:33.043        | 13:50:33.490         | 447ms  |
| 13:50:33.091        | 13:50:33.542         | 451ms  |
| 13:50:33.104        | 13:50:33.584         | 480ms  |
| 13:50:33.121        | 13:50:33.602         | 481ms  |
| 13:50:33.138        | 13:50:33.616         | 478ms  |

Durchschnittliche Latenz: **444ms**

Tabelle 1: Messung der Latenz von 30 Websocket-Nachrichten zwischen zwei Clients während einer kollaborativen Session. Computer A bezieht die Internetverbindung über einen VPN-Dienst in Atlanta, USA. Computer B ist ohne VPN-Verbindung in Deutschland mit dem Internet verbunden. Alle Nachrichten enthalten Vue Flow Change-Objekte mit einer Zeichenlänge zwischen 148 und 194 Zeichen. Das Hosting des Frontends geschieht via GitHub Pages, der Websocket-Server wird via der Cloud-Plattform Heroku bereitgestellt.

| A: Timestamp Senden | B: Timestamp Empfang | Latenz |
|---------------------|----------------------|--------|
| 12:18:56.668        | 12:18:56.701         | 33ms   |
| 12:18:56.685        | 12:18:56.746         | 61ms   |
| 12:18:56.711        | 12:18:56.747         | 36ms   |
| 12:19:01.390        | 12:19:01.424         | 34ms   |
| 12:19:01.635        | 12:19:01.672         | 37ms   |
| 12:19:01.704        | 12:19:01.738         | 34ms   |
| 12:19:01.990        | 12:19:02.023         | 33ms   |
| 12:19:02.155        | 12:19:02.191         | 36ms   |
| 12:19:02.194        | 12:19:02.228         | 34ms   |
| 12:19:02.205        | 12:19:02.239         | 34ms   |
| 12:19:02.408        | 12:19:02.442         | 34ms   |
| 12:19:02.589        | 12:19:02.623         | 34ms   |
| 12:19:02.850        | 12:19:02.884         | 34ms   |
| 12:19:02.868        | 12:19:02.902         | 34ms   |
| 12:19:03.778        | 12:19:03.812         | 34ms   |
| 12:19:03.799        | 12:19:03.835         | 36ms   |
| 12:19:03.813        | 12:19:03.869         | 56ms   |
| 12:19:03.869        | 12:19:03.906         | 37ms   |
| 12:19:03.879        | 12:19:03.913         | 34ms   |
| 12:19:04.588        | 12:19:04.622         | 34ms   |
| 12:19:04.597        | 12:19:04.631         | 34ms   |
| 12:19:05.750        | 12:19:05.861         | 111ms  |
| 12:19:05.787        | 12:19:05.864         | 77ms   |
| 12:19:05.937        | 12:19:05.972         | 35ms   |
| 12:19:06.184        | 12:19:06.219         | 35ms   |
| 12:19:07.309        | 12:19:07.347         | 38ms   |
| 12:19:07.629        | 12:19:07.664         | 35ms   |
| 12:19:08.508        | 12:19:08.543         | 35ms   |
| 12:19:08.531        | 12:19:08.564         | 33ms   |
| 12:19:08.603        | 12:19:08.636         | 33ms   |

Durchschnittliche Latenz: **40ms**

Tabelle 2: Messung der Latenz von 30 Websocket-Nachrichten zwischen zwei Clients während einer kollaborativen Session. Beide Computer beziehen das Internet ohne VPN. Alle Nachrichten enthalten Vue Flow Change-Objekte mit einer Zeichenlänge zwischen 148 und 194 Zeichen. Das Hosting des Frontends geschieht via GitHub Pages, der Websocket-Server wird via der Cloud-Plattform Heroku bereitgestellt.


## Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Wiehl, 7.8.2024

---

Ort, Datum

A handwritten signature in black ink, appearing to be 'S. B.' with a stylized flourish.

---

Unterschrift