HTBLVA, Wien V, Spengergasse

Höhere Lehranstalt für Informatik
Ausbildungsschwerpunkt Informatik

DIPLOMARBEIT

Squavy
Browser Based Digital Audio Workstation

Implementation of audio and signal processing in modern web browsers
Fabian Mild 5AHIF Betreuer: Ing. Klaus Unger, MSc.

Leveraging RFC6455 for low-latency, concurrent collaboration
Fabian Hummel 5AHIF Betreuer: Ing. Klaus Unger, MSc.

Monitoring of custom front and backend software using tools like Prometheus and Grafana
Konrad Simlinger 5AHIF Betreuer: Ing. Klaus Unger, MSc.

Deployment and Scalability Strategies and their Implementation for Web-Applications
Alejandro Dario Tomeniuc 5AHIF Betreuer: Ing. Klaus Unger, MSc.

Schuljahr 2024/25

Abgabevermerk:

Datum: 04.04.2025 übernommen von:

SPENGERGASSE	HÖHERE TECHNISCHE BUNDESLEHRANSTALT SPENGERGASSE	
	Fachrichtung:	**Informatik**
	Ausbildungsschwerpunkt:	**Informatik (IOT, SOS, BAP)**

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benützt habe. Die Stellen, die anderen Werken (gilt ebenso für Werke aus elektronischen Datenbanken oder aus dem Internet) wörtlich oder sinngemäß entnommen sind, habe ich unter Angabe der Quelle und Einhaltung der Regeln wissenschaftlichen Zitierens kenntlich gemacht. Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen.
Sofern für die Erstellung der Arbeit generative KI-Tools eingesetzt wurden, so sind auch diese im Quellenverzeichnis vollständig und wahrheitsgetreu inkl. Produktversion und Prompt ausgewiesen.

Wien, am 04.04.2025

Verfasser / Verfasserinnen:

Fabian Mild

Fabian Hummel

Konrad Simlinger

Alejandro Dario Tomeniuc

SPENGERGASSE	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT SPENGERGASSE**
	Fachrichtung: **Informatik** Ausbildungsschwerpunkt: **Informatik (IOT, SOS, BAP)**

DIPLOMARBEIT

DOKUMENTATION

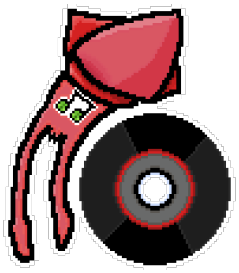
Namen der Verfasser/innen	Fabian Mild, Fabian Hummel, Konrad Simlinger, Alejandro Dario Tomeniuc
Jahrgang Schuljahr	2024/25
Thema der Diplomarbeit	Squavy - Browserbasierte Digital Audio Workstation
Kooperationspartner	Digimagical GmbH, Legstadtgasse 4-6/C25

Aufgabenstellung	**Mild:** Seit dem Start von Squavy haben wir viele verschiedene Bibliotheken getestet, die mit Browser-Audio und Signalverarbeitung arbeiten. Keine von ihnen entsprach den Anforderungen einer modernen DAW, also begannen wir, unsere eigene Lösung zu entwickeln. Ich werde diese verschiedenen Optionen erläutern und die Leistung, die Vorteile und die Mängel jeder einzelnen von ihnen hervorheben. **Hummel:** Effiziente, schnelle und robuste Kommunikation über das Netzwerk ist etwas, das wir jeden Tag als selbstverständlich ansehen, das aber leichter gesagt als getan ist. Im Laufe dieser Diplomarbeit analysiere ich verschiedene Architekturen und Technologien, die hinsichtlich Flexibilität, Wartbarkeit und Leistung punkten, um die Echtzeit-Zusammenarbeit nahtlos in Squavy zu integrieren. **Simlinger:** Mein Thema erforscht, wie benutzerdefinierte Metriken in Node- und Rust-Software verfügbar gemacht werden können. Ziel ist es, nützliche Metriken unserer Front- und Backend-Anwendung zu exportieren, die dann zum weiteren Verständnis des Serverstatus oder zur Skalierung von Diensten verwendet werden können. **Tomeniuc:** Ich werde untersuchen, wie moderne Webanwendungen hohe Verfügbarkeit, Flexibilität und Skalierbarkeit erreichen können. Der Schwerpunkt liegt dabei auf der effizienten Skalierung bei schwankenden Nutzerzahlen und den Herausforderungen, die durch On-Prem- und Cloud-Technologien sowie Container entstehen. Ziel ist es, Strategien zur Optimierung von Skalierbarkeit und Effizienz zu identifizieren.

Realisierung	Squavy schafft eine benutzerfreundliche Plattform für Musikbegeisterte und Studenten mit Zusammenarbeit im Vordergrund. Sie richtet sich an neue Nutzer, die in die Musikproduktion eintauchen möchten, ohne unzählige Stunden mit den Grundlagen zu verbringen. Damit das Komponieren von Musik noch mehr Spaß macht, können sich die Nutzer nahtlos zusammenschließen und gemeinsam an einem Projekt arbeiten.

SPENGERGASSE	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT SPENGERGASSE**
	Fachrichtung: **Informatik** Ausbildungsschwerpunkt: **Informatik (IOT, SOS, BAP)**

Ergebnisse	**Mild:** Die Audio-Engine ist eines der Kernsysteme von Squavy. Hier werden Klänge synthetisiert, gemischt, moduliert, usw. Das System muss robust und schnell sein, mit wenig bis keinem Raum für Fehler zur Laufzeit. Es muss auch Funktionen für die Wiedergabe und den Import/Export von Klängen enthalten. **Hummel:** Ziel dieses Teils der Diplomarbeit ist es, ein robustes System für die Online-Zusammenarbeit zu entwickeln, das geringe Latenzzeit und hohe Zuverlässigkeit gewährleistet. Wichtig ist die nahtlose Integrierung in das Hauptprogramm, so dass mehrere Benutzer in Echtzeit über den Browser und mit minimaler Einrichtung an der Musikbearbeitung zusammenarbeiten können. Darüber hinaus sollte die Software auf unser verteiltes System zugeschnitten sein, um eine einfache Bereitstellung und Skalierbarkeit zu ermöglichen und uns gleichzeitig ein Höchstmaß an Kontrolle über die Analysen und Vitaldaten des Servers zu geben. **Simlinger:** Bereitstellung einer Monitoring-Plattform mit verschiedenen Tools und Diensten als Basis für die weitere Bearbeitung (z.B. Skalierung von Diensten) und Schaffung einer Plattform zur Anzeige des aktuellen Serverstatus. **Tomeniuc:** Konzeption eines skalierbaren Deployments für die Webanwendung.

Typische Grafik, Foto etc. (mit Erläuterung)	 Logo von Squavy  Screenshot des Programms

Teilnahme an Wettbewerben, Auszeichnungen	Teilnahme beim „Jugend Innovativ“ Wettbewerb 2024/25

Möglichkeiten der Einsichtnahme in die Arbeit	Bibliothek Spengergasse; Online auf www.squavy.com

Approbation (Datum / Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungsvorständin

SPENGERGASSE	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT SPENGERGASSE**
	Fachrichtung: **Informatik** Ausbildungsschwerpunkt: **Informatik (IOT, SOS, BAP)**

DIPLOMA THESIS

Documentation

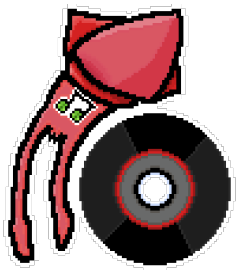
Author(s)	Fabian Mild, Fabian Hummel, Konrad Simlinger, Alejandro Dario Tomeniuc
Form Academic year	2024/25
Topic	Squavy - Browser Based Digital Audio Workstation
Co-operation partners	Digimagical GmbH, Legstadtgasse 4-6/C25

Assignment of tasks	**Mild**: Since the start of Squavy, we have been testing many different libraries that handle browser audio and signal processing. None of them sufficiently suit the needs of a modern DAW, so we started to develop our own solution. I will explain these different options and highlight the performance, benefits and shortcomings of each of them. **Hummel**: Efficient, fast and robust communication over the network is something we take for granted every day but is easier said than done. Over the course of this diploma, I will analyze and test several different architectures and technologies that score in flexibility, maintainability and performance to seamlessly integrate real-time collaboration into Squavy. **Simlinger**: My topic explores how custom metrics can be made available in node and rust software. The goal is to export useful metrics of our front and backend application which then can be used to further understand the server status or to scale services. **Tomeniuc**: I will explore how modern web applications can achieve high availability, flexibility, and scalability. The focus will be on efficient scaling with fluctuating user numbers and the challenges posed by on-prem and cloud technologies and containers. The goal is to identify strategies to optimize scalability and efficiency.

Realisation	Squavy aims to create a user-friendly platform for music enthusiasts and students to collaborate easily. It targets new users looking to start their music production journey without needing to spend countless hours learning the basics. To make music composition even more fun, users can choose to seamlessly get together and collaboratively work on a project.

SPENGERGASSE	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT SPENGERGASSE**
	Fachrichtung: **Informatik** Ausbildungsschwerpunkt: **Informatik (IOT, SOS, BAP)**

Results	**Mild:** The audio engine is one of the core systems of Squavy. Here, sounds are synthesized, mixed, modulated, etc. The system must be robust and fast, with little to no room for errors on runtime. It must also include playback-related features and sound import/export. **Hummel:** The aim for this part of the diploma project is to develop a robust communication protocol for real-time online collaboration, ensuring low-latency and high reliability. The application should be seamlessly integrated into the main program, allowing multiple users to collaborate on music editing in real-time all through within the browser and minimal setup. Additionally, the software should be tailored to our distributed system, facilitating easy deployment and scalability whilst granting us maximum control over the analytics and vitals of the server. **Simlinger:** Deliver a monitoring platform using different tools and services to provide a basis for further processing (e.g. scaling of services) and create a platform to view the current server status. **Tomeniuc:** Conception of a scalable deployment for the web application.

Illustrative graph, photo (incl. explanation)	 Logo of Squavy	 Screenshot of the application

Participation in competitions Awards	Participation in the „Jugend Innovativ“ competition in 2024/25

Accessibility of diploma thesis	Library Spengergasse; Online at www.squavy.com

Approval (date / signature)	Examiner	Head of College / Department

Table of contents

1. Preface

Music. It is a cultural tool that has persisted over the vast majoritiy of human development and is rooted deep into all cultures of the world. Something that has changed over this timespan however, is the way of creating music. Nowadays, numerous tools are used to ease the lives of musicians. A crucial stage for music production is the production phase, where different arrangements of instruments are tested to create the best outcome possible.

However, the more features these products contain, the more difficult it is to get the hang out of them. Beginners who are completely new to the field are daunted by the very sight of them. We think that there is a better solution, which is why we have created Squavy.

The aim of the project "Squavy – Browser Based Digital Audio Workstation" is to create a user-friendly and visually appealing platform for music enthusiasts and students, which also enables them to collaborate easily with one another. People who are looking to start their music production journey will not have to spend countless hours learning the basics on just how to handle the platform. They can easily start composing the music they always dreamt of. With a browser at hand, the experience of Squavy is just a click away.

2. Implementation of audio and signal processing in modern web browsers

2.1 Introduction

Since multimedia support was first introduced, the technologies available to web developers have become more and more sophisticated. One of these technological advances is the ability to play and modify audio signals. This offers many exciting possibilities, such as adding acoustic feedback to websites, creating interactive music applications, and enabling real-time voice modification. As browsers continue to evolve, their capabilities will only expand, creating even more opportunities for web developers.

2.2 Topic of Investigation

Since the start of Squavy, the team has been testing many different libraries that handle browser audio and signal processing. None of them sufficiently suit the needs of a modern DAW[1], so we started to develop our own solution. I will explain these options and highlight the performance, benefits and shortcomings of them. I will also talk about some of the development hurdles that occurred along the way, as well as their solutions.

2.3 Objective

Audio and signal processing is a mandatory part of Squavy. The purpose of this research is to find the best solution for audio processing in the browser, which runs stable, has a clear signal resolution and suits the high performance requirements of Squavy. It also functions as an opportunity to expand on the current ideas for implementation, so that the best outcome will be achieved.

[1]Digital Audio Workstation `https://www.steinberg.net/de/tutorials/what-is-a-daw`

2.4 Intended Result

This diploma includes a functioning and viable version of the SMSE[2] audio engine, which implements the acquired knowledge in a meaningful and efficient way and is a mandatory part of Squavy. The SMSE consists of an efficient and correctly implemented synthesizer with the source to sink architecture, while also following the guidelines of the midi standard.

2.5 Sound and the Browser

Before I go into the details of sound synthesis and signal processing in general, I want to mention that there are different browser implementations of audio, which caused some problems along the way.

This is because audio interaction is handled by one of the browsers Web-APIs, more specifically the Web Audio API. Since this is not part of JavaScript, but built on top of it[3], some differences might occur across different browsers. This includes the default sample rate, variations in latency and the autoplay policy.

These issues are often hard to spot and might take some time to fix. An example is the difference in sample rate on Chrome, when using macOS instead of Windows[4]. This confused the audio visualizer of Squavy, thus messing with the integrity of the application. All in all, it took me about a week to fix the issue, which is humbling when taking the amount of these small inconveniences into account.

2.6 The Web Audio API

The Web Audio API gives developers a way to play back files, process effects or simply synthesize sounds via the browsers internal audio thread.

The usage is straightforward and intuitive, as the whole signal processing chain is based on a source-to-sink architecture. This design choice is heavily inspired from technologies like PulseAudio or the JACK Audio Connection Kit, so it should not be a new idea to a general audio developer.

[2] https://github.com/Squavy-DAW/Synth

[3] Web APIs https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Client-side_APIs/Introduction

[4] Waveform Visualizer Problem https://www.reddit.com/r/webaudio/comments/1aprgbi/web_audio_api_issues_on_macos/

To illustrate this practically, some sort of oscillator might generate a sinusoidal wave in the form of $y(x,t) = A * sin(kx - \omega t + \phi)$, which is then simply routed into the browser's audio destination. The below code shows the simple implementation.

Creating a simple 330 Hz sine wave via the Web Audio API

```
let ctx = new AudioContext();
let oscillator = ctx.createOscillator();

oscillator.connect(ctx.destination);
oscillator.type = 'sine';
oscillator.frequency.value = 330;
oscillator.start();
```

In this example, the oscillator is connected to the context's destination, which results in audible sound through the device's speakers. The modular structure of this synthesizer allows for the insertion of additional nodes between the oscillator and the destination node and can heavily influence the resulting sound.

2.6.1 FM Synthesis

The first idea of Squavy was in fact using the Web Audio APIs source to sink capabilities for FM Synthesis, which generates sound via a fixed amount of carrier oscillators and a flexible modulation pool (Figure 1).

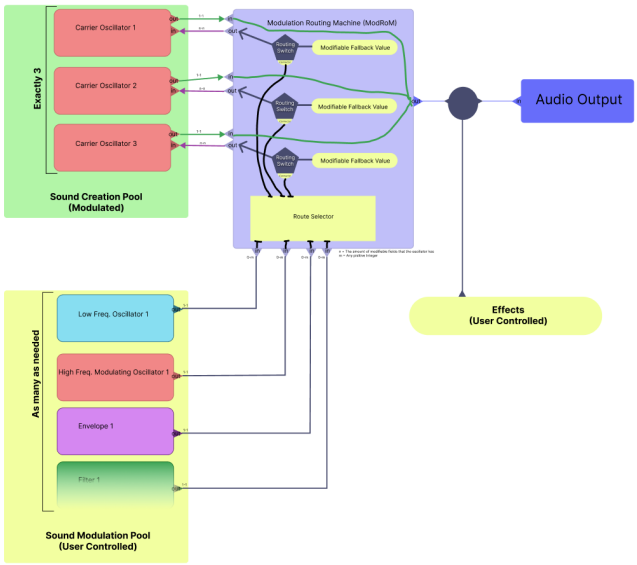


Figure 1: Squavys first Synthesizer

This approach however comes with a major downside, which is the bad scalability. That is because in order to extend on the synthesizer, which means, for example, to add time based value automation or external plugins, the architecture has to be somewhat modified at its core. Furthermore, additions like external plugin support and custom audio nodes just make this system more and more complex. This is why I did not stick with this approach in the long term and worked on a more modular system instead.

2.7 Shortcomings

Despite its simplicity and easy setup of simple audio chains, the numerous shortcomings of the Web Audio API lead to a lot of pain and suffering further into the development of Squavy. This chapter lists a few of these.

2.7.1 Monophony

Generative audio nodes such as the oscillator node are monophonic. This means that it is not possible to play multiple frequencies at once, which is a huge problem when composing more advanced songs.

The fact that the Web Audio API does not adhere to basic standards of audio synthesis makes development very tedious and unnecessarily difficult. Traditionally, analog synthesizers were in fact monophonic, which makes sense, since each physical oscillator contains a chip to convert electrical impulses to audible sound at a fixed frequency. To create harmonics, multiple oscillators would be bundled together, like the infamous 3x-Osc (Figure 2) from the digital audio workstation *FL-Studio*.

Figure 2: 3x-Osc from FL-Studio

In Squavy, this monophony led to tedious routing of the audio-node connections: The generator pool had to be split in an active and an inactive part, because oscillators cannot be restarted after being stopped, so they would have to be disconnected instead. Additionally, there had to be multiple oscillator copies, which would, as a whole, act as a polyphonic instrument. In general, this means that they would all have to be disconnected and reconnected every single time a note is played. This is incredibly inefficient, and eventually led to a search for alternatives.

Tedious routing because of Monophony

```
1 export function routeTree(trackId:string, freq:number, idx:number) {
2     // check if a specific oscillator at
3     //  a specific frequency is active
4     let active = activeAudioNodes[trackId][freq][idx];
5 }
```

2.7.2 Inconsistencies and missing Features

Something else that plays a big part in audio synthesis is property modulation, which is the process of changing a value of an audio node over a set time frame. For example, the gain of an oscillator could be changing from `0%` to `100%` for the first two seconds after starting playback. While the Web Audio API officially supports this ability, it is not implemented for some specific properties like an oscillator's type or phase, the model of a panner-node or the curve of a waveshaper node. While these are not necessary in most scenarios, their lack still restricts the freedom of the developer. Furthermore, the audio nodes that are available per default are rather simple. They do not provide extensive functionality like multiplying panner nodes or out of the box flanger nodes. Everything else has to be added via custom audio nodes.

2.7.3 Timing in JavaScript

Timing is a crucial aspect when working with signals that require high precision and accuracy. Not only modulations, but also events, the effect processing chain and even the sound synthesizing itself have to be handled in a suitable time frame. In the case of audio, it is set in the sub-millisecond range. But does JavaScript meet this requirement?

As JavaScript runs in the browser, the timing of function calls is not very precise - in fact, they are actually quite off. Core functions like `setTimeout()` run the provided callback with rather poor accuracy and are not suited for precise timings, especially in combination with audio synthesis. Fortunately, the MDN documentation explicitly states that the actual delay for the timeout might be longer than specified[5], so it is not a point of uncertainty. But still, I am interested in the degree of this uncertainty. To better visualize the extent of this issue, I decided to run a quick test that outputs the average delay of `setTimeout()`, which iteratively runs the function, calculates the delay with the built-in performance tool and repeats the process a hundred times:

[5]Source: `https://developer.mozilla.org/en-US/docs/Web/API/Window/setTimeout` (March 29, 2025)

setTimout() accuracy test

```
let results = [];
let prev = performance.now();
setTimeout(function timeout() {
    results.push(performance.now()-prev-1);
    prev = performance.now();
    if (results.length > 100) {
        // evaluate minimum, maximum and average
        return;
    }
    setTimeout(timeout, 1);
}, 1);
```

```
Minimum: 0.19999992847442627
Maximum: 3.9000000953674316
Average: 3.508910889672761
```

The output is rather humbling. Although ∼3 milliseconds do not seem worrisome, delays like these are very audible in the final application, which kills the timing during playback. For reference, the smallest timing differences humans can perceive closely distribute around 10 microseconds[6]. In the case of Squavy, another approach must take the place of JavaScript timeouts.

2.7.4 Conclusion

Although the Web Audio API is hard to handle at times, it may find an optimal use in smaller applications that aim to play back audio files or simply play around with basic features.

At the end of the day, all of these drawbacks overshadow the positive aspects of the Web Audio Engine and lead to the decision of creating a custom in-house synthesizer - *Squavy's Modular Synthesizer Engine*.

[6]The smallest perceivable time differences https://pubs.aip.org/asa/jasa/article/145/1/458/638769/Smallest-perceivable-interaural-time-differences

2.8 Audio Worklets for the Rescue

2.8.1 Introduction

Thankfully, the Web Audio API supports creating custom audio nodes with user provided JavaScript code using Audio Worklet Processors. Formerly this feature was implemented through the `ScriptProcessorNode`, but it is now deprecated in favor of the new API. The code of the processor is executed in a separate JavaScript environment, the `AudioWorkletGlobalScope`, that has a reduced feature-set. This means that otherwise available functions and classes such as the document object model (DOM) and WebAssembly simply do not exist there, as it is fully focused around fast and accurate digital signal processing.

The AudioWorkletProcessor

```
1 class SmseProcessor extends AudioWorkletProcessor {
2     process(inputs, outputs, parameters) {
3         // synthesize audio
4     }
5 }
6 // register the processor to be available later
7 registerProcessor('smse-processor', SmseProcessor)
```

The `SmseProcessor` extends the browser's `AudioWorkletProcessor`, which enables direct access to the browsers precise timing APIs and provide helpful variables such as `currentTime` which tracks the time since the processor has started with very good precision, unlike the default JavaScript clock.

The heart of the processor lies within its `process` function, which is called whenever the audio context requires new audio data to play back. Within this function, the actual audio is being generated in steady, fixed 128 samples sized blocks, even though this is expected to change in the future[7].

To actually initialize the processor, the entire content needs to be in a separate file, which I have successfully found a workaround to work consistently with Squavys build tool. Essentially, the raw code is imported as a string, converted to a binary large object and finally assigned a unique URL that the browser can access with `URL.createObjectURL`.

[7]Source: `https://developer.mozilla.org/en-US/docs/Web/API/AudioWorkletProcessor/process` (March 29, 2025)

Create the Processor

```
import SmseProcessor from './smse-processor.js?raw';

const smseProcessorUrl = URL.createObjectURL(
    new Blob(SmseProcessor, {
        type: "application/javascript"
    })
);
```

Everything that is left now is to actually instantiate the processor and connect it to the audio context, just like with the oscillator from before. In fact, the processor is nothing different from an ordinary audio node:

Instantiation and Connection

```
await this._audioContext.audioWorklet.addModule(smseProcessorUrl);

this.node = new AudioWorkletNode(this._audioContext, 'smse-processor', {
    outputChannelCount: [2],
});

this.node.connect(this._audioContext.destination);
```

Now that the code is running, I could theoretically start writing my own synthesizer, but first, I would like to address the elephant in the room - *JavaScript itself.*

2.8.2 Going a step further

JavaScript is a great supportive tool for building interactive websites and plays an essential role in Squavy's development, but when it comes to something as bare bones and performance critical as digital audio synthesis, I would really love to avoid JavaScript in any way possible. To my advantage, WebAssembly is a common technology nowadays that helps port various performance intensive applications like Photoshop[8] to the web by allowing developers to program in native languages like C++ or Rust.

In the case of Squavy, WebAssembly would be the perfect addition to further optimize the program and to provide for a much better development experience. However, actually

[8]Source: `https://web.dev/articles/ps-on-the-web` (March 29, 2025)

integrating this technology into Squavy was more work than initially planned and required clever techniques to work around some problems:

Earlier, I mentioned that WebAssembly does not exist within the dedicated audio scope, even though this is not entirely true, as it is indeed available, but there is no obvious way to instantiate the compiled code without some neat tricks. Normally, the `.wasm` file can be imported straight into a JavaScript module using Vite's static imports, but this is not supported in external code like in the processor.

The only viable option would be to first fetch the individual bits and bytes of the binary and send them over to the audio thread using the audio node's message channel API for bidirectional communication.

Fetching the Assembly

```
1  import wasmUrl from '../pkg/synth_bg.wasm?url';
2
3  // initialize the processor
4
5  const smseWasmCode = await fetch(wasmUrl)
6      .then(response => response.arrayBuffer());
7
8  this.node.port.postMessage({
9      type: 'init-wasm',
10     data: smseWasmCode
11 });
```

Inside the processor, the message is handled via the `port.onmessage` event listener and contains the assembly that and is then compiled using `WebAssembly.compile`:

Loading the Assembly

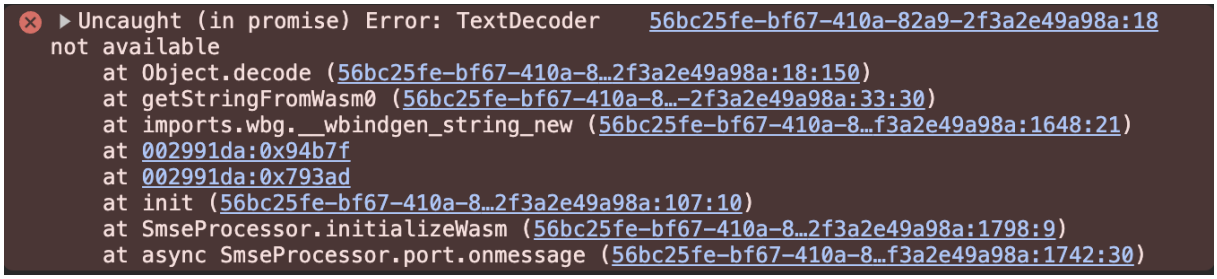
```
constructor(options) {
    super(options);

    this.port.onmessage = this.handleMessage;
}

function handleMessage(event) {
    if (event.data.type === 'init-wasm') {
        const module = await WebAssembly.compile(wasm);
        const m = await __wbg_load(module, __wbg_get_imports());
        __wbg_finalize_init(m.instance, m.module);
        init();
    }
}
```

But where do these weird looking functions `__wbg_load`, `__wbg_get_imports` and `__wbg_finalize_init` come from? Usually, these are invoked internally by the supplied `initSync` function when importing the WebAssembly module through Vite. To provide these functions to the processor, essentially the raw JavaScript code is extended by the WebAssembly's glue-code that is generated automatically by `wasm-bindgen`:

Concatenating the Glue-Code with the Processor

```
import bindgen from "../pkg/synth.js?raw";
import SmseProcessor from './smse-processor.js?raw';

const smseProcessorUrl = URL.createObjectURL(
    new Blob([bindgen + SmseProcessor], { // simple concatenation!
        type: "application/javascript"
    })
);
```

When I initially tried to run the code now, the console displayed an error that the `TextDecoder` class was not available in the `AudioWorkletGlobalScope`. The glue-code needs this functionality to convert between Rust's internal strings and JavaScript strings. The reason for why the worklet environment does not contain these classes is apparently due to bloat-reduction and the fact that *"There are no use-cases that require encoding and decoding character set in a real-time audio callback"*, which is what one of the projects maintainers stated over on GitHub.



Luckily, there is a workaround for this problem as well that includes downloading a small polyfill for the missing classes `TextEncoder` and `TextDecoder` written by *Viktor Mukhachev*[9]. The code also needs to be concatenated to the glue-code and the processor before creating the worklet. This finally resolves all issues and the dream of a native Rust synthesizer all within the browser lives on.

2.9 Synthesizer Architecture

2.9.1 SMSE Nodes

Squavy's modular synthesizer is inspired by the core concepts of JUCE[10], which helped a lot with making the right decisions when designing the architecture. Generally speaking, each node is either an audio node that processes actual samples or a MIDI node that takes in and modifies MIDI events. Initially, audio was synthesized sample per sample, but this led to unnecessary complications during integration with the audio worklet, so it was changed to be buffer based, the standardized way to process an audio node chain.

2.9.2 Types of modular Nodes

The synthesizer consists of different types of audio nodes that all have different responsibilities in the final node chain. The list below gives a rough overview about these characteristics:

[9]Source: `https://gist.github.com/Yaffle/5458286#file-textencodertextdecoder-js` (March 29, 2025)

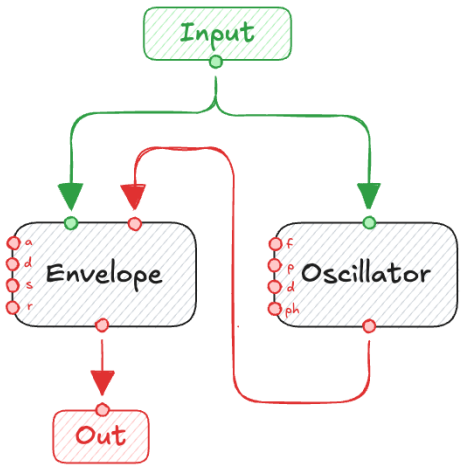
[10]JUCE (Framework) `https://juce.com/` is an open source C++ framework for digital signal processing including solutions for creating and loading a broad range of audio plugins

- **Generative Audio Nodes:** Nodes that primarily focus on the MIDI input and create sound based on the events within the received MIDI-buffer. Examples for generative audio nodes are Oscillators, Noise generators or Samplers, just to name a few. They sit at the start of the modular node tree, as they are the entry point for anything that is audible sound.
- **Effect Audio Nodes:** Special audio nodes that all consume at least one input from another audio node and apply an effect such as reverb, distortion, delay and many more.
- **MIDI Nodes:** Nodes that consume at least one MIDI buffer from another MIDI node and modify it to either add, remove or rebuild another MIDI buffer based on different inputs. Examples for these type of nodes are Pitchers or Arpeggiators.
- **Out Node:** Each instrument contains exactly one output node where the resulting audio buffer will be committed to the final, mixed output of the synthesizer.
- **Input Node:** A special kind of MIDI node that receives and forwards user events like key presses, but also actual timing information about which note should currently be played during playback. Similar to the out node, each instrument also has exactly one input node.

Though they might work differently, the fundamental concept of modifying input and providing output stays the same over each and every node. This flexibility is what allows the architecture to be as scalable as possible, even paving the way to work with external plugin software.

2.9.3 Instruments and Control Flow

Like an orchestra, Squavy songs consist of multiple individual instruments that, when played together, form one, great harmony. Each instrument groups specific nodes together to form a node chain where MIDI events flow from the very start (the input node) to the very end (the output node, respectively). Logically, the control flow moves from start to end, just like in analog synthesizers, where each output signal is passed to the next node, ultimately reaching the amplifier to produce audible



sound waves. From a technical perspective, the control flow is completely inverted due to reasons explained shortly after. This means that the very end of the chain asks the connected node for audio data, whereas the implementation details vary based on the node that is connected. For example, when the output is connected to an oscillator, the oscillator is going to produce data on behalf of the out node, not the other way around.

The reason for this has to do with the cardinality of the node graph. It is way more technically feasible to store which node is connected to a specific node's inputs, rather than to store what nodes are connected to a specific node's output:

For example, the input node does not store where its output is connected to, which would be required for the logical control flow, but instead the *oscillator* and *envelope* nodes store which node is connected to their MIDI parameter. This way, each node only needs to store `0..1` for each input, opposed to storing `0..*` connections for each output, allowing for the output node to traverse the tree into every possible branch until it reaches the leaf nodes, in our case the generative audio nodes or MIDI input node.

2.9.4 MIDI Buffer

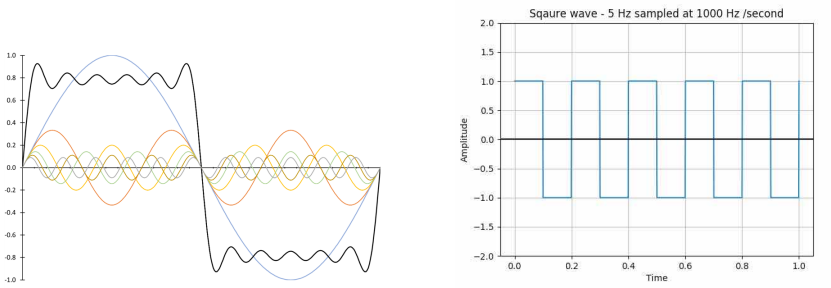
When playing back the song, the synthesizer wants to know which notes are subject to be played for the currently processed time frame. These notes are stored in a interval tree, a special type of binary tree that is optimized for filtering its elements by time. Afterward, the relevant notes are converted into a different format, similar to MIDI events, and placed in a dedicated buffer that audio nodes can make sense of.

Taken the pitcher node, it accepts a MIDI buffer as an input and mutates it to produce a new MIDI buffer, but with each note copied by a given harmonic interval up or down to easily create chords, even if the actual input was only one note at a fixed frequency.

The reason to use MIDI buffers is to adhere to JUCE's standardized audio node format, because it will become more relevant in the future as we prepare to integrate external plugins, and it seems like a fairly flexible way to manipulate note and timing data to influence the generated sound.

2.10 Problems of Digital Signal Processing

Designing a synthesizer from scratch is not a straightforward task by any means. During development, I have often come across hurdles that resulted in major setbacks. Thankfully, most of the logical issues were able to be resolved fairly quick after discovery to this day.



(a) Sinusoidal Components of a Square Wave

(b) Square Wave with Modulo Operator (%)

Figure 3: Mathematically correct vs. approximated Square Wave[11]

However, even when the program seems to be working flawlessly, there are always some invisible problems that leave me unsettled. A lot of these are not an issue when using an established framework like the Web Audio API, but only show themselves as fundamental inconveniences during development.

2.10.1 Aliasing

Aliasing for example, is an unwanted side effect of a mathematical function with jump points. These artifacts do not occur naturally and are more of an issue with digital signal processing. Taken a square wave, 50% of the time, the wave is up (1), and the other 50% of the time it is down (-1).

The mathematically correct way to calculate such a wave would be to harmonically overlay an infinite amount of different sine waves and therefore does not have any jump points. Unfortunately, computers have limited processing capabilities and need to take another route to calculate at least an approximation to this function. The naive way would be to overlay `x` amount of sine waves, which is still going to cost too much processing power to even get anything remote to a good square wave. Alternatively, I could accept my fate and try to solve the issue afterwards - at least it runs very fast this way.

In the case of our digital synthesizer, an oscillator may produce different wave types that include jump points such as pulse waves (the generic form of a square wave), triangle waves, sawtooth waves, et cetera. The most basic waveform, the simple sine wave, is not affected by this issue as it already is a single and steady wave. At least if its frequency is in the audible range.

[11]Source: `https://learnemc.com/qotw-221219` and `https://pythontic.com/visualization/waveforms/squarewave`

Generally speaking, aliasing is a common problem in digital signal processing, specifically as the sample rate leads to a projection of an infinite number of harmonics into a finite buffer.

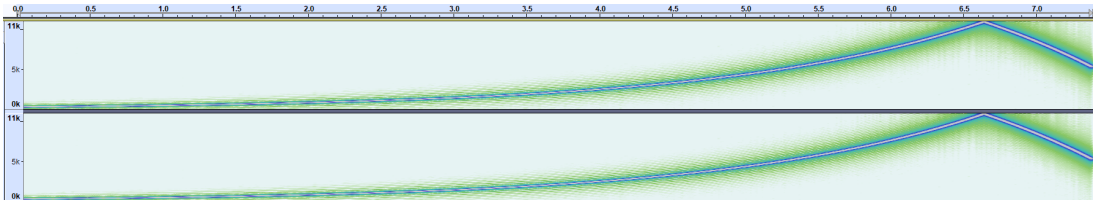


Figure 4: Aliasing with a sample rate of 22050

The visualization (Figure 4) emphasizes the issue that the frequency starts to wrap back down as soon as the pitch passes around 11 kHz, even though the pitch should rise continuously. This break takes place at exactly half the sample rate, and even has a special name - the Nyquist frequency[12].

2.10.2 The Nyquist Theorem

The Nyquist theorem claims that in order to accurately display an analog signal in digital form, the chosen sample rate must be double the highest occurring frequency. Everything that is higher in pitch than the Nyquist will be reflected into the audible spectrum again, ultimately distorting the result. Once this happens, there is no way to regain the original signal, as the projection is baked into the resulting audio now and it is hard to tell which frequencies to keep and what others to mute.

With a simple sine, this is not a problem, as the sample rate is set to a high value like 44.1 kHz or 48 kHz, so the Nyquist is above the highest frequency that humans can hear, which is about 20 kHz[13].

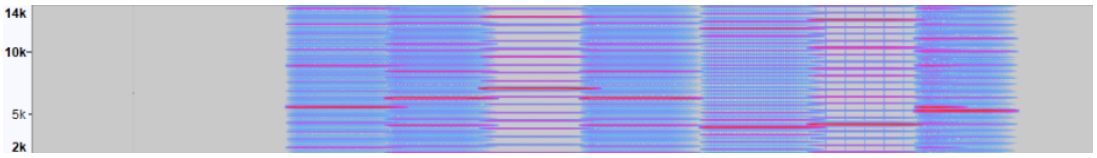


Figure 5: Aliasing of a Square Wave

However, the spectrum for a square wave looks a lot different, as seen in Figure 5, where the aliasing is obscured by a vast amount of unwanted frequencies to the point where the original note cannot be heard at all.

[12]Nyquist Frequency https://sciencedirect.com/topics/computer-science/nyquist-frequency
[13]Highest audible frequency `https://www.ncbi.nlm.nih.gov/books/NBK10924`

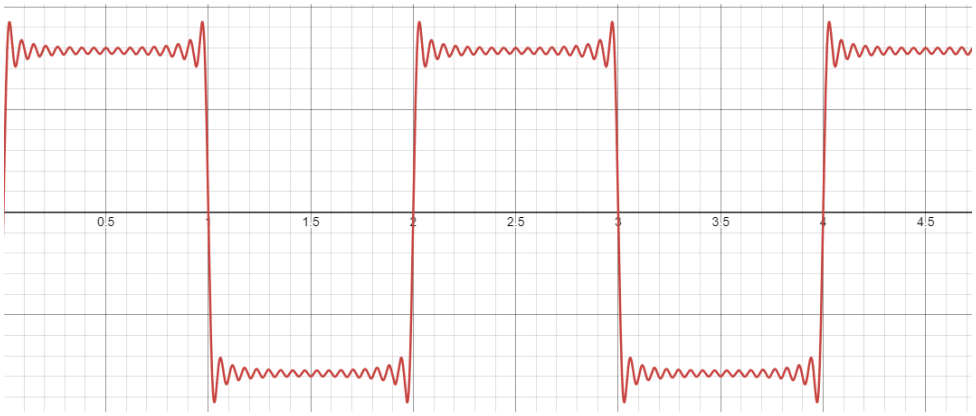


Figure 6: A Square Wave with 17 harmonics

This is because a square wave, unlike a sine wave, is made up of infinite harmonics, which gives it a more complex texture. In digital signal processing though, the amount of harmonics is finite, because otherwise the signal would be made out of an infinite number of sine waves, which is not computable.

2.10.3 Mitigating Aliasing

There is no single perfect solution to this problem, the effect can only be minimized. As explained earlier, the straightforward solution would be to simply add all harmonics under the Nyquist frequency, but this is extremely inefficient. Keep in mind that there are roughly 48.000 samples that have to be calculated every second. Even just adding ten harmonic waves would already go beyond the computational limits of the CPU. Furthermore, lower frequencies are more prone to this issue, as they can fit way more harmonics.

Using additive synthesis to generate a Square Wave

```
1  Waveform::Square => {
2      let mut value = 0.0;
3      let nyquist = sample_rate / 2.0;
4      let mut harmonic = 1.0;
5
6      while harmonic * frequency < nyquist {
7          value += (TAU * harmonic * frequency * time + phase)
8              .sin() / harmonic;
9          harmonic += 2;
10     }
11
12     value * 2.0 / TAU
13 }
```

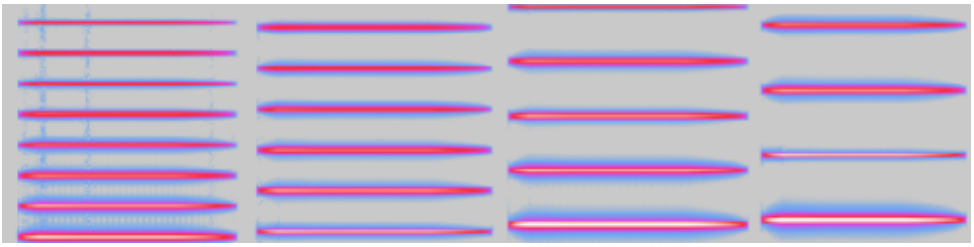


Figure 7: Anti Aliased Square Wave

To obtain such a clean result (Figure 7), a different technique is used. The oscillator may work on the original calculation that incorporates the modulo operation.

Typically, there are multiple approaches to anti-aliasing, which can be used alone or in combination[14]. The easiest solution is to use oversampling, which is computationally inexpensive and produces decent results, although it may not be sufficient by itself.

2.10.4 Oversampling

Oversampling is, in relation to the other techniques, a relatively basic concept for anti aliasing. It works by taking an input signal with a higher sample rate (up to 8 times the original) and then downsampling the signal later.

Simple Oversampling and Downsampling implementation

```
fn calculate_oversampled(
    &self,
    frequency: f32,
    phase: f32,
    time: f64) -> f32
{
    let mut sum = 0.0;
    let step = 1.0 / sample_rate * OVERSAMPLING_FACTOR as f64;

    for i in 0..OVERSAMPLING_FACTOR {
        let t = time + i as f64 * step;
        sum += self.calculate(frequency, phase, t);
    }

    sum / OVERSAMPLING_FACTOR as f32
}
```

[14] Anti Aliasing https://youtu.be/VVhXRzzzHa8?si=yxIU21AWL2-ylifL&t=299

As seen above, the sample rate is artificially increased by a predefined factor. This increases the amount of calculated samples, which are then averaged to reduce the sample rate to the original value. Although this idea is perfectly fine, using it on its own won't cut it. At least not, if the oversampling factor is computationally efficient, thus small. The minimum for Squavy is measured to be around 8x via experimentation. For contrast, the aliasing of high pitch sound only starts being acceptable with a factor of 64x or 128x, which is far from feasible in real time web applications, especially when there are many other things going on simultaneously.

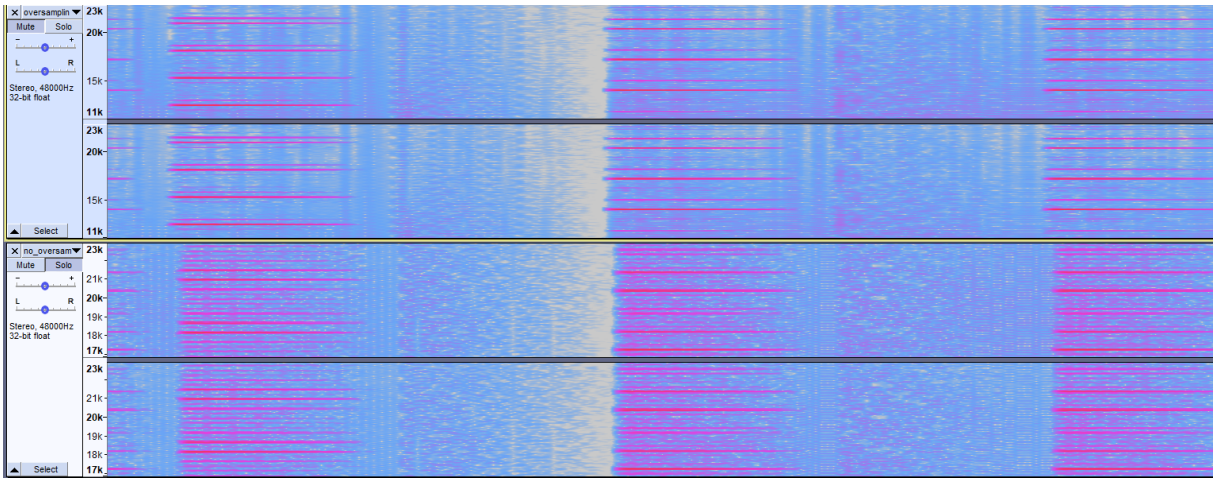


Figure 8: Oversampling (top) VS No Oversampling (bottom)

As shown in Figure 8, the aliasing is reduced a lot. Comparing the top channel with the bottom one, the pink lines, which show frequencies with lots of volume, experienced major reduction, thus eliminating unwanted noise.

2.10.5 IIR-Filters

Oversampling by itself is not enough to reduce aliasing to a satisfying amount. This is where an aliasing filter comes in. Before the oversampled signal gets downsampled, a low pass filter is placed on the signal, which then cuts the unnecessary frequencies out before they land in the final output.

Typically used filters are Butterworth, Chebyshev or Bessel. Figure 9 shows the output of both a butterworth filter (in the top half of the image) and an unprocessed aliased square wave (in the bottom half). The upper output looks a lot more acceptable in relation to the unprocessed one and is a lot more viable for Squavy. Both images are zoomed in quite a bit, which explains the moderate resolution.

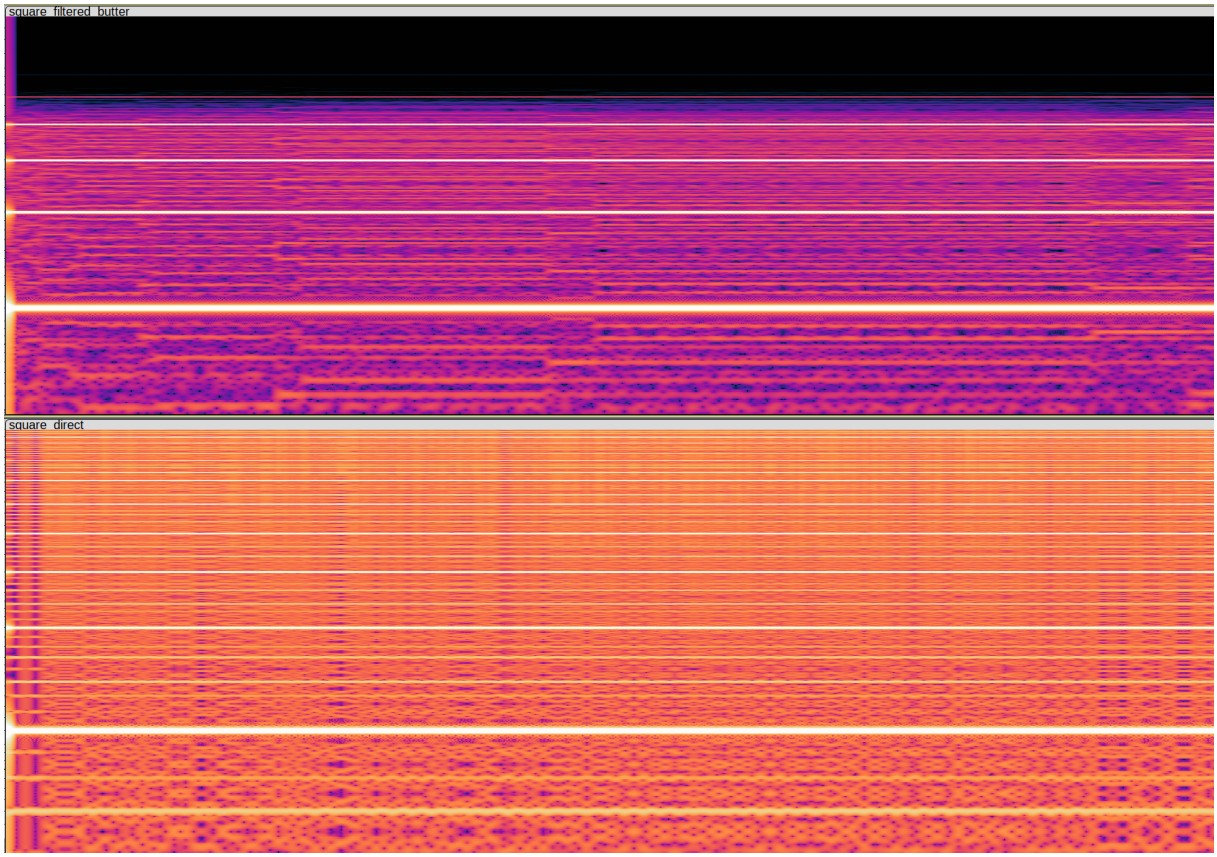


Figure 9: Anti Aliasing through a Filter paired with Oversampling

2.10.6 Usage in Squavy

Squavy uses a simple Butterworth filter of order 10, to maximize the effect of the anti aliasing.

Precalculated Coefficients for IIR

```
1 const ACOEFFS: [f32; 10] = [0.831585910854, 1.481784128538, /*...*/];
2 const BCOEFFS: [f32; 10] = [1.0,2.0,1.0,2.0,1.0,2.0,1.0,2.0,1.0,2.0];
3 const GAIN: f32 = 7.789799228640309;
```

The filter functions coefficients are calculated with the help of Jagged Planets IIR Filter Explorer[15]. The resulting values are then parsed into Rust code and finally used as an addition to the previous oversampling function. The resulting output is much better than before and also not too costly performance wise.

[15]IRR Filter Explorer `http://jaggedplanet.com/iir/iir-explorer.asp`

2.11 Wavetable Synthesis

The future of Squavy is still undecided, but as far as signal generation goes, I see a light at the end of the tunnel. The final step will be centered around wavetable synthesis. This means that instead of generating waveforms from scratch every time, a pregenerated table is used to look up the sound at the desired pitch (Figure 10).

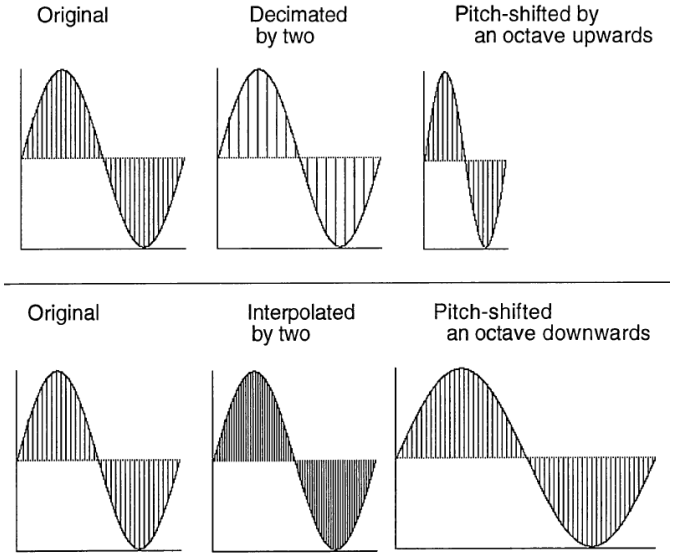


Figure 10: The Principle of Wavetable Synthesis[16]

2.11.1 Mipmapping

To further enhance the quality of the produced sound, an additional technique that is used in practice is Mipmapping. For this, a large set of tables with different frequencies is used to generate even more accurate sound via interpolating between the nearest frequencies of the desired pitch.

2.11.2 The best of all worlds

The great advantage of this approach is the flexibility of the underlying wavetables. For reference, instead of just generating a simple square wave via the modulo operation, a finite filter is baked into the waveform beforehand. This drastically improves performance, as additional antialiasing is not needed afterwards.

[16]Source: `https://www.researchgate.net/figure/The-principle-of-wavetable-synthesis-Reading-out-values-from-an-array-with-a-certain_fig1_340952649`

2.12 Lessons Learned

2.12.1 Aliasing and The Web Audio API

If I had used the Web Audio API, many of the issues that I had to resolve would have never occurred. Additionally, the implementation of more complex audio-nodes would have been fairly quick because the Web Audio APIs predefined nodes and obviously the excellent browser integration. Nonetheless, I would say that moving away from it did more good than harm to Squavy and the end product is much more sophisticated and cleanly built than what it could have been if the team still used the old approach.

2.12.2 Trial and Error

Developing a Digital Audio Workstation is hard. In the past year of developing Squavy's synthesizer, I learned that some things are better done via trial and error, as learning experiences are generally better when failing is a part of the journey. Constant redesigns and iteration after iteration is not something to be afraid of, but an opportunity to visualize progress and to emphasize what is working or what is not. In addition, it is always rewarding when looking back at past milestones and strengthen one's knowledge.

2.12.3 Further improvements

Doing audio generation the right way is a real challenge. Wavetable synthesis with mipmapping capabilities will play a major role for Squavy's audio processor, and is still currently being developed. However, the implementation details are not entirely fixed at the time of writing this paper, as more ideas are constantly researched to this date. Like most of the time, development comes with a lot of trial and error and totally exceeds the scope of my diploma.

The latter may even be subject to change depending on the future architecture of Squavy's audio pipeline, which is continuously altered to optimally support the complexity of external plugins and sophisticated synthesizer elements. As always, planning is key when it comes to technical decisions, and software projects are no exception. The obtained experiences will surely contribute to even better planning in the future. Nonetheless, it will definitely contribute to my skills in a positive way.

3. Using WebSockets for low-latency, concurrent collaboration

3.1 Topic of Investigation

Efficient, fast and robust communication over the network is something we take for granted every day, but is easier said than done. The objective is to analyze and compare various communication protocols to choose the most suitable one for Squavy's online collaboration feature. This involves examining factors such as performance, scalability, latency, and security. Besides that, the research aims to explore technical advancements that simplify web development in regard to real-time communication.

This study will take a heavily technical approach with deeply explained architectural patterns, but also sets the focus on explaining why different protocols suit particular applications best.

3.2 Intended Result

After the research phase, the most suitable communication protocol for Squavy's online collaboration feature will be identified based on the aforementioned factors and needs to be implemented and integrated into the main product, ready to be used by potentially thousands of people.

3.3 Boundaries of this Study

Internet communication is a complex and broadly diversified topic, which makes it a difficult task to cover every aspect within a single study. Therefore, I will not bother to explain the visual implementation such as frontend controls for the multiplayer collaboration and rather focus on the far more interesting technical side of things.

It is important to note that this study is **not** a one-size-fits-all solution for choosing the right communication protocol or implementing the best security measurements, but solely gives a broad overview of different technologies and the reasoning behind creating a product like Squavy's collaboration server.

3.4 Product Scope

The final software needs to meet specific criteria in order to satisfy the intended result. Precisely said, the product must fulfil the following conditions to such an extent that I am confident to claim that the software is ready to be publicly deployed.

- **Session based collaboration:** Allow users to create sessions and be given an invitation code that they share with other people. When a user joins a session, they receive the entire shared project data from a certain user or central server.
- **Low latency synchronization:** Share project updates on the fly with the lowest possible latency for an immersive experience. This means that I have to compare different communication protocols and carefully choose the best one for Squavy's collaboration feature. See subsection 3.5
- **Encryption:** Securely encrypt messages that are sent over the network. Only clients that need to know about specific data have the ability to read the data.

If the decision is made in favor of a classic server-client architecture, these additional points must be considered:

- **End-to-end encryption:** The server must not be able to make sense of the encrypted content. It solely serves as a communication relay between connected clients. This strategy is often called *pseudo peer to peer*. See subsection 3.13
- **Session management:** Sessions should be managed in a fully automated way, in the sense of disposing sessions when there are no users left to save server resources. See subsubsection 3.9.4
- **Scalability:** Options for flexible scalability should be kept in mind while developing the server. This means that multiple server instances should be deployed in different locations of the world with a way to share session data with as little delay as possible to make cross-country collaboration possible.

3.5 Comparing Communication Protocols

There are several options for immediate communication over the internet. However, some strategies are superior to others in regard of stability, security, throughput and development comfort.

3.5.1 Long Polling

Long Polling is an easy way of exchanging messages both from and to the server, but is very limited and should only be used in an absolute emergency, or when messages don't need to arrive at an instant. This protocol misuses HTTP in the sense by using GET and POST requests to exchange data. Sending data from the client to the server is fairly straightforward by simply sending a post request with the packed data in the frame body. However, the server does not have a direct way of talking to the client, unlike with WebSockets. For that to work, the client needs to explicitly ask the server for new information in short intervals, which is pretty resource consuming and slow.

In the case of Squavy, long polling is not a suitable option because the collaboration feature heavily relies on rapid message exchange.

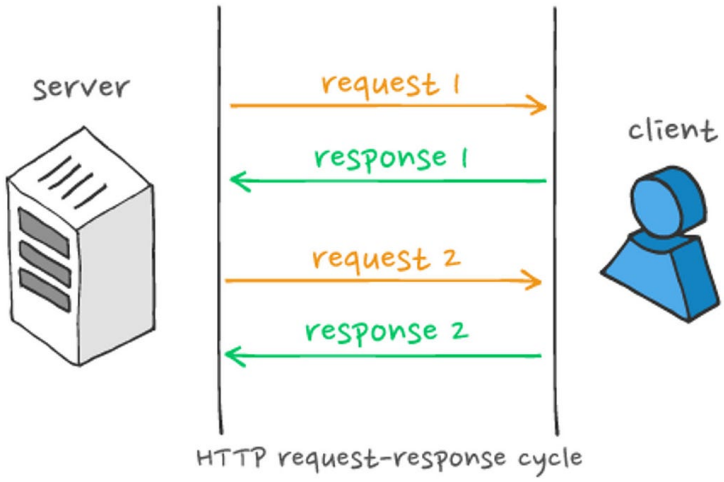


Figure 11: HTTP Long Polling[17]

[17]Image Source: https://medium.com/@ignatovich.dm/implementing-long-polling-with-express-and-react-2cb965203128 (March 26, 2025)

3.5.2 Peer to Peer

Peer to peer in contrast is the complete opposite of a traditional server-client architecture. It abuses the fact that a specific port on the client can be temporarily opened when making a request to a server (a signalling server) to allow for a response to be sent and received. However, during the small time the port is opened, we can intercept the connection and instead let the other client to connect instead. This procedure is called NAT punch-through[18] in technical terms, but has its limitations.

Foremost, it is important that at least one of the clients explicitly sends keep-alive packets[19] to all its peers, as this is the only thing that keeps the connection alive.

Secondly, NAT uses the port number of a client to map the response's destination address back to the original client behind the firewall, which can be ambitious when the computer is situated in a company or school. This may not be an issue for all public buildings or corporations, but when I tried it with Wi-Fi from within HTL Spengergasse, I was unable to join two clients together. Ultimately, I scrapped this idea because the implementation was terribly unstable and not supported everywhere, which I consider to be a major drawback.

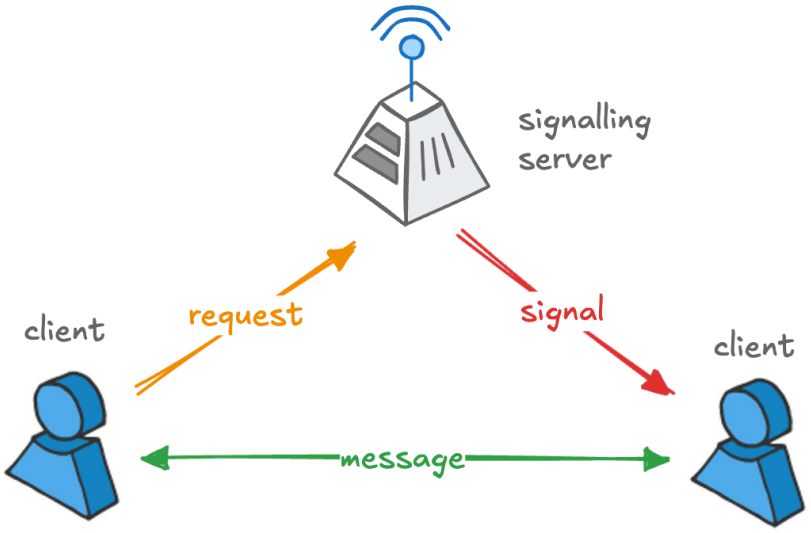


Figure 12: Peer to Peer Networking

[18]Hole punching (networking) https://en.wikipedia.org/wiki/Hole_punching_(networking)

[19]Keep-alive packets are special network messages that tell the destination that the sender is still available.

3.5.3 WebSockets

WebSockets solve all these issues by extending the base socket protocol[20] to a standardized JavaScript API that can be used in every major browser since July 2015[21]. As the title of my individual topic already points out, I decided to use the WebSocket protocol (RFC6455) for the collaboration feature of Squavy. The specific messaging "features" or capabilities of our collaboration server are explained extensively in subsubsection 3.9.3.

However, the server will not serve as a single source of truth (SSOT), but solely as a small relay that knows about each client and exchanges encrypted messages back and forth. In subsection 3.12 I further explain theoretical concepts and implementation details regarding this topic.

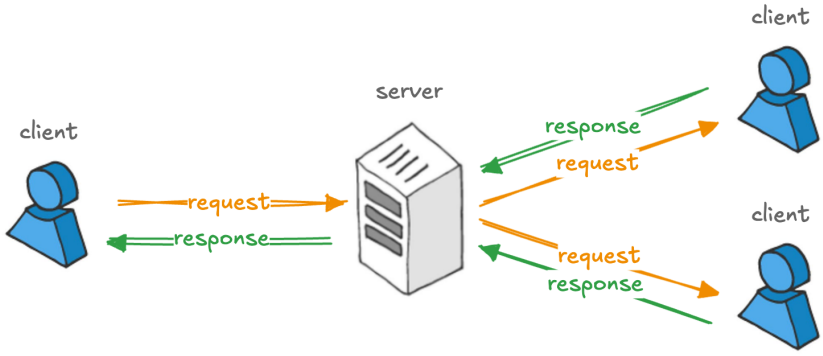


Figure 13: WebSocket Networking

3.5.4 HTTP/2

HTTP/2 is an upgraded version of the standard HTTP/1.1 protocol and was initially, amongst other things, designed to (partly) replace WebSockets. Why is it that I have never heard of this protocol before researching? Well, HTTP/2 does indeed closely follow the paradigm of classic WebSockets, but with one major exception: It is not possible to - quote - *"push binary data from the server to a JS web client"* because the browser does not have an API to receive and consume binary data payloads *and* **such an API is not planned**[22], which is absolutely hilarious in my opinion. This naturally makes it a dealbreaker, especially for binary-heavy communication, such as in Squavy's collaboration.

[20] A socket is defined to be the unique identification to or from which information is transmitted in the network. Source: `https://datatracker.ietf.org/doc/html/rfc147` (March 26, 2025)

[21] Source: `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket` (March 26, 2025)

[22] Source: `https://stackoverflow.com/a/42465368/13948619` (March 26, 2025)

3.6 Comparing Server Architectures

Now that the communication protocol is chosen, I am now able to decide which server architecture suits Squavy best. I picked out three good candidates that could theoretically satisfy the requirements:

3.6.1 Socket.io[23]

A modern JavaScript wrapper for the existing WebSocket implementation provided by Node.js with additional features like session management, state recovery when a client is forcefully disconnected, and even provides a dynamic router for grouping the application into independent sections. Although this library is considered industry standard when working with WebSockets, I decided against it because JavaScript's backend performance is slower than compiled languages due to its single-threaded and interpreted nature, leading to potential bottlenecks and ultimately making it less reliable for high performance backend applications[24].

3.6.2 uWebSockets[25]

C++ has limited options in the area of WebSocket programming. Our best bet is uWebSockets, a single-threaded library that support serving HTTP and WebSockets in a simple, modern fashion. However, it does not conform to the Socket.io standard, which means it does not support sessions, namespaces and all those nice-to-have features. Even though this library did not cut it for the collaborative feature, it does play an essential role in a different location of Squavy, specifically the Squidge.

[23] Socket.io (Software) `https://socket.io/` allows for real-time communication between clients and servers via WebSockets.

[24] Source: `https://wirekat.com/why-javascript-on-the-backend-is-not-that-great/` (March 26, 2025)

[25] uWebSockets (Software) `https://github.com/uNetworking/uWebSockets` is a C++ server WebSocket implementation.

3.6.3 Socketioxide[26]

This library is basically Socket.io, but entirely rewritten in Rust[27] from scratch by a few hobbyist maintainers. It's built to be compatible with Tokio[28], which enables the server to also use HTTP handlers, just like with uWebSockets. At first glance, this seems like the perfect solution for a server architecture, but does not account for the fact that learning Rust is not an easy task and can take dozens of hours to master, especially when required to work with asynchronous code.

3.7 Chosen Architecture

At the time of writing this paper, a decision has already been made which architecture to use to integrate real-time collaboration into Squavy. Actually, it has already undergone numerous rewrites of previous iterations, but the current implementation is written in Rust and uses Socketioxide to support online collaboration in Squavy. The repository is publicly available on GitHub at `https://github.com/Squavy-DAW/Server` if the team did not decide to keep it private for marketing reasons.

For the client side of things, I have always kept the native JavaScript client implementation of the Socket.io protocol and is used together with a custom wrapper capable of simple end-to-end encryption. Additionally, Squavy uses SolidJS[29] Stores (reactive data structures) for change detection, a very important part of the online collaboration, which will be explained in detail in subsection 3.11.

3.8 Early Implementation Stages

The development of the server has come a long way since the start of creating the first iteration of such software in late 2023. Long before the final software was decided to be developed in early 2024, there were several previous attempts that all have had their drawbacks and were eventually rewritten from scratch.

[26]Socketioxide (Software) `https://github.com/Totodore/socketioxide` is a Rust based server implementation of the Socket.io protocol.

[27]Rust (Language) is a general-purpose programming language focused on performance, efficiency and memory safety.

[28]Tokio (Software) `https://tokio.rs/` is a runtime for writing reliable asynchronous applications with Rust. It provides async I/O, networking, scheduling, timers, and more.

[29]SolidJS (Software) `https://www.solidjs.com/` is a simple and performant frontend JavaScript framework.

October 28, 2023: The initial implementation of a multiplayer server using Express[30], uWebSockets.js and Socket.io. Development was very rapid, and the server worked quite well during the test phase, but had its major pitfall during deployment, as JavaScript based programs are not really intended to be run on the server side. Eventually, the decision was made to move away from this implementation due to concerns of reliability and performance of an interpreted language.

December 18, 2023: I quickly ported over the old code to a new server implementation in Rust using socketioxide. During development, I experienced several issues due to Rust's steep learning curve compared to lenient languages. However, there has been made progress and the completion of a somewhat working prototype in early January 2024.

April 9, 2024: Together with the rebranding of the old product's name *HarmonyHub* to *Squavy*, I decided to rewrite the server once more, especially to learn from previous mistakes and to rework many parts of the program that have become obsolete or broken in the past, mostly due to continuous evolution of the software's biggest dependency, socketioxide. This implementation is the newest version that's also deployed to the public and receives smaller updates for further improvements from time to time, but has all major features one would expect from a scalable backend server for real-time collaboration.

3.9 Backend Implementation

3.9.1 Querying Available Sessions

A collaborative session starts at the client, which initially asks the server for an available namespace to connect to. This is handled through a basic HTTP listener that is placed inside the Tokio tower structure. Furthermore, the session store is also added to the service to gain access to it within the handler, which is required to read and modify related data.

Setting up an HTTP handler for requesting an available session

```
let app = axum::Router::new()
    .route_service("/session", ServiceBuilder::new()
        .service(get(handle_query_available_session))
        .with_state(session_store.clone()));
```

[30]Express (Software) `https://expressjs.com` is a simple framework for creating RESTful APIs with Node.js.

The route handler then returns a randomly generated string that is not currently in use by other collaborators. Another option would be to use an ever-increasing number to use as the session's namespace, but sessions could be easily discovered by simply brute forcing various numbers and disrupt the service.

```
pub async fn handle_query_available_session(
    sessions: State<SessionStore>) -> Response<Body>
{
    let available = loop {
        let s = /* generate a random string */
        if !sessions.get_all().contains_key(&s) {
            return s;
        }
    }

    Response::builder()
        .status(200)
        .header("Content-Type", "application/json")
        .body(Body::from(available))
        .unwrap()
}
```

3.9.2 Connecting to a Session

Sessions can be thought of individual, independent rooms with people that freely share all sorts of messages amongst them. Thankfully, the Socket.io protocol supports this exact feature and allows easy integration into the server.

First, it's important to add the Socketioxide layer to the tower, just like the HTTP handler from before.

Setting up Socketioxide to use Websockets

```
let (socket_io_layer, io) = SocketIo::builder()
    .with_state(session_store.clone())
    .build_layer();

let app = axum::Router::new()
    .layer(socket_io_layer);
    // [...]
```

Subsequently, a dynamic namespace (or route) is configured on the layer. This allows the client to establish a connection to any arbitrary sub-namespace like `/abc` or `/mt5xbr1`. The actual handler then decides what to do with that information.

```
io.dyn_ns("/{id}", on_connect.with(middleware));
```

However, the actual session instance has not been created yet, which would connect the client to an empty, nonexistent room. To resolve this issue, it's important to check if the session has not been created yet, in which case it will be created with a supplied name and passphrase.

Optimally, this code is handled inside a middleware instead of the actual route handler due to the fact that the middleware runs slightly earlier and the session is then ready to be used by the time the handler is called.

```
pub fn middleware(
    socket: SocketRef,
    Data(data): Data<ConnectSession>,
    sessions: State<SessionStore>) -> Result<(), Error>
{
    let session = socket.ns().to_string();
    /**
     * if the session does not exist, add it and Ok()
     * if the password matches, retain the session and Ok()
     *  | auto session disposal is described later
     * otherwise, disconnect the socket with Err()
     */
}
```

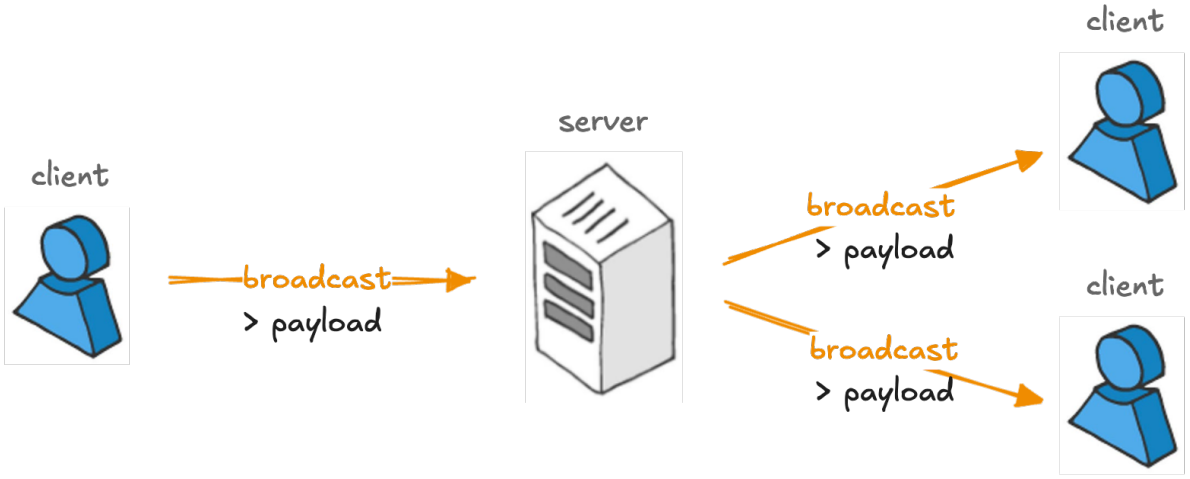
The handler code is only responsible for setting up event handlers for specific actions that the client is able to perform during its lifecycle like sending messages, requesting data, etc... and ultimately also handling disconnection.

```
pub async fn on_connect(socket: SocketRef) {
    socket.on("sqw:broadcast", on_broadcast);
    socket.on("sqw:unicast", on_unicast);
    socket.on("sqw:request", on_request);
    socket.on_disconnect(on_disconnect);
}
```

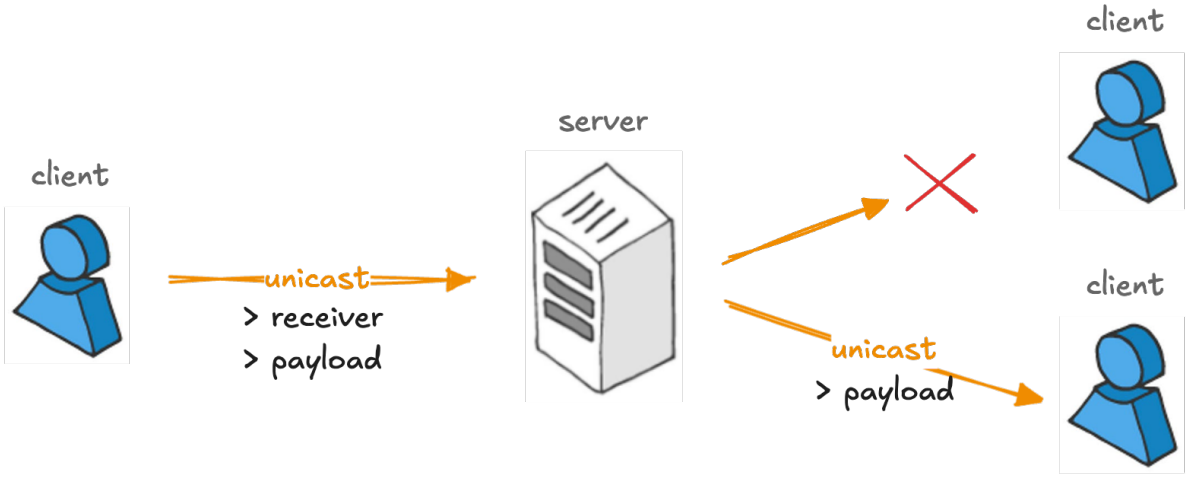
3.9.3 Bidirectional Communication

Now comes the fun part - sending and receiving messages to and from collaborators. Generally speaking, messages mainly contain two parts. First, the actual data which is ordinarily in a binary format due to end-to-end encryption and secondly a transport mode that defines how the message is processed on the server. Traditionally, the mode can be one of the following options, similar to other transport protocols like email:

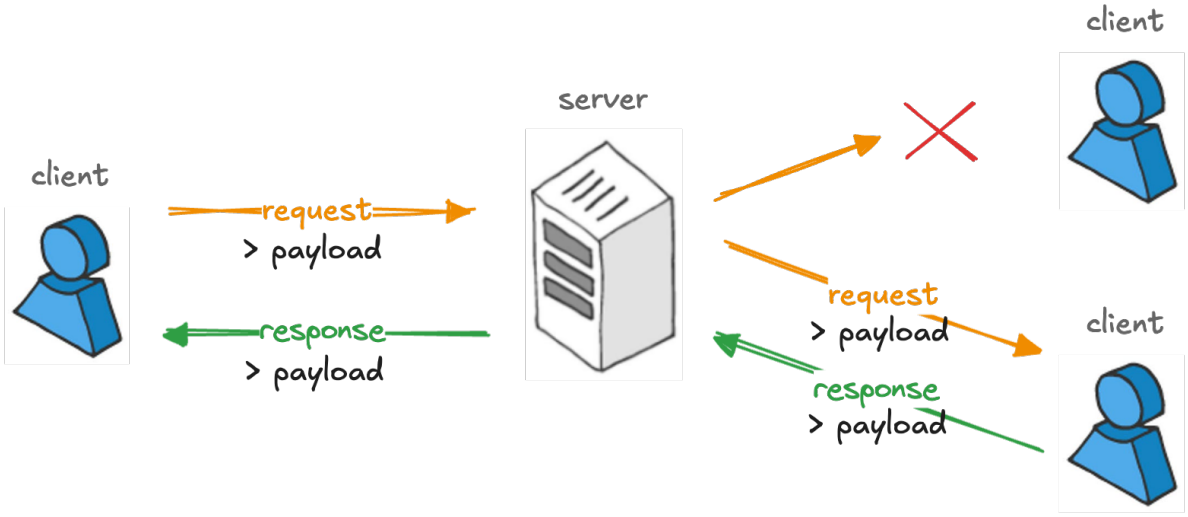
Broadcast: Messages sent with this mode are distributed amongst all connected clients. This is most commonly used for updating the project data or a user's mouse position within the browser window. Gathering individual responses is also supported, either directly through the message's acknowledgement (a combined view of all responses before a deadline) or a continuous stream of replies.



Unicast: These types of messages are specifically attributed to a specific client using their unique identifier. It is not really used in Squavy because there are no use cases that involve sending a message to a single client only, except for direct messages through a chat panel.



Request: A request is very similar to a unicast message, with the difference that the sender does not explicitly target a specific collaboration member, but the server chooses the *"best"* one to forward the message to. This message is used to initially acquire the project from a collaboration member when joining a session.



Furthermore, the server also supports separate messages that do not require the room to have more than one collaborator, such as requesting the current session data to display in the frontend.

3.9.4 Automatic Session Disposal

An important factor for the multiuser collaboration server is automatic self maintenance and cleanup of otherwise endlessly growing resources. Ideally, the server should automatically free empty sessions after a fixed time of inactivity. To track this inactivity, the software contains a disposal queue that the session is added to whenever the last member exits or a new session is created without anyone joining. After the timer has expired, the session is permanently closed, ultimately freeing precious resources.

For every session that is being disposed, the server spawns a new thread with a specific timeout that is currently set to five minutes. After the time has elapsed, the session is deleted from the server. Otherwise, they can be retained, which simply aborts the running thread early.

```rust
pub fn queue_dispose(&self, session: String) {
    // only used to correctly update realtime analytics
    self.get(session).available_until =
        Utc::now().add(chrono::Duration::minutes(5));

    self.queue.insert(ns, task::spawn(async move {
        sleep(Duration::from_secs(300)).await;
        sessions.remove(session.clone());
    }));
}

pub fn retain(&self, session: String) {
    if let Some(task) = self.queue.remove(&session) {
        // cancel deletion
        task.abort();
    }
}
```

3.10 Unit Testing

In a collaborative server environment, maintaining system reliability is crucial, as any change can have a negative impact on performance and numerous internal processes within the server. Therefore, unit testing becomes essential to ensure that code modifications do not introduce new bugs or disrupt existing functionality. After intensive research I have concluded to use *Mocha*[31] together with *Chai*[32] as my prioritized tech-stack, as they satisfy all my needs for testing the collaboration server such as asynchronous tests.

At first glance, testing server software may seem pretty cumbersome, but thankfully it's fairly straightforward to include the Socket.io client in the testing environment. The `forceNew: true` flag is needed so that the Socket.io client does not cache and reuse an existing connection, which would introduce undefined behavior and go against the rules of unit testing.

[31] Mocha (Library) `https://mochajs.org/` a simple, flexible, fun JavaScript test framework for Node.js and The Browser

[32] Chai (Library) `https://www.chaijs.com/` is an assertion library that focuses on an expressive syntax and readable style.

```
describe("Broadcast Tests", function () {
    before(async function () {
        const ns = // generate random namespace

        // wait for two clients to connect to the server.
        await new Promise((resolve, reject) => {
            this.a = io("http://127.0.0.1:8000" + ns, {
                forceNew: true });
            this.a.on("connect", resolve);
        });

        // identical setup with client 'b'
    }

    it('broadcast and receive message', function(done) {
        const enc = new TextEncoder();
        const data = { binary: enc.encode("binary") };
        this.a.on("sqw:data", ({ data }) => {
            chai.expect(data).to.be.deep.equal(data);
            // continue with the next test.
            done();
        });
        this.b.emit("sqw:broadcast", data);
    });
});
```

3.11 Internal Change Detection

A big part of synchronization between users through a network connection is to reliably replicate the data on all clients. The naive way of doing so would be to broadcast the entire app state in a fixed time interval, however, as the program grows, the amount of data required to send to the clients would reach an enormous magnitude. Let's scrap that idea and focus on a mechanism to only get the absolute minimal set of information required to replicate the data without desynchronization[33].

[33]Inconsistent data between clients, which commonly leads to bugs or unintended behaviour.

Using the event driven approach which is most commonly found in game development, event handlers are set up at specific parts of the program where clients need to share replicated data, and are explicitly defined how that data is represented as a plain, serializable JavaScript object. This method grants developers the most control over *how, when and where* specific parts of the application are synchronized which can be beneficial, but is often cumbersome to implement and error-prone, especially for large software solutions like Squavy.

```
1  // listen to mouse events and update the reactive data.
2  createNetworkListener("squavy:mouse", (id, { x, y }) => {
3      setMouses(produce(mouses => {
4          mouses[id] = { x, y };
5      }));
6  });
7
8  function handleMouseMove(event: MouseEvent) {
9      // manually define what is sent over the network.
10     socket.broadcast("squavy:mouse", x, y);
11 }
12
13 return <div onMouseMove={handleMouseMove}>
14     // Display mouses
15 </div>;
```

Earlier in development of the backend server, my first instinct was to implement the aforementioned strategy for data replication between collaborators, but this always led to issues for specific edge cases that were not properly handled and would have taken several painful hours ensuring that the data was synchronized without errors. However, I kept using it for some other purposes like synchronizing the mouse positions between users as seen in the code snippet above.

The main data structure (contextually known as the *"project structure"*) is replicated in a fully automated way. This is important to keep in mind while developing Squavy, as everything in this structure will be shared amongst clients and should only be modified with substantial caution to prevent overloading the network. To better understand how this system is technically implemented, we have to dig deeper and first learn how SolidJS internally propagates updates of its reactive data structures.

In the code snippet below, we create a new reactive data structure of the data type `Project` which holds all information about a Squavy project including notes, tracks and synthesizer configurations - basically everything that should be synchronized between collaborators.

```
const [project, setProject] = createStore<Project>(/* data */);
```

Here, the `setProject` function is used to change the state of the data structure, which then triggers a **side effect**. We can now listen to these changes and execute some logic, in our case sending the object's difference to our peers. The actual difference of the previous value compared to the updated one is processed using `deep-object-diff`[34].

```
let previous: Project;

createEffect(on(project, project => {
    // get the object's difference using deep-object-diff
    const difference = getDiff(previous, project);
    // send the difference to the other clients
    socket.broadcast("squavy:project", difference);
    // deeply clone the current project to be our previous value
    previous = structuredClone(project);
}));
```

To actually receive changes from our peer, we once again set up an event listener and apply the difference to our current project. To help merge the two JavaScript objects, I used the counter-part library `deep-object-diff-apply`[35].

```
createNetworkListener("squavy:project", (id, difference) => {
    setProject(produce(project => {
        // apply the difference using deep-object-diff-apply
        applyDiff(project, difference);
    }));
});
```

[34] deep-object-diff (Library) `https://www.npmjs.com/package/deep-object-diff` is a small library that can deep diff two JavaScript Objects, including nested structures of arrays and objects.

[35] deep-object-diff-apply (Library) `https://www.npmjs.com/package/@transformation-dev/deep-object-diff-apply` takes the output of deep-object-diff's difference and applies it to the original object.

Now, when the server instance is launched and two clients connect to the same session and the state is updated by, for example, placing a note... hold up, something is going terribly wrong. The server logs are spamming the console in a matter of seconds, and it looks like we put ourselves in an infinite loop of sending project updates back and forth. Why is that?

This *"bug"* arises due to SolidJS's way of propagating store updates through the app to trigger side effects. By applying the received difference to our local data structure, the code inside the `createEffect()` function is called too, which in return sends a message to the other client, who does the same thing over and over again, repeating indefinitely.

The solution to this problem is to differentiate between a local update to our project and an update received through a network event. A simple solution is to add a special field to the project model and to update the value of that field whenever a local update was triggered. The value we give it does not matter; its only purpose is to be included in the object's difference, hence Booleans are an optimal solution by being very straightforward to modify. In the resulting side effect of the update, the program simply checks if the field has been updated and does not trigger a network broadcast.

```
createEffect(on(project, project => {
    const difference = getDiff(previous, project);
    // only broadcast if the flag is not set.
    if (diff._INTERNAL_UPDATE === undefined) {
        socket.broadcast("squavy:project", difference);
    }
    previous = structuredClone(project);
}

createNetworkListener("squavy:project", (id, difference) => {
    setProject(produce(project => {
        // modify the flag to include it in the difference.
        project._INTERNAL_UPDATE = !project._INTERNAL_UPDATE;
        applyDiff(project, difference);
    }));
});
```

3.12 Concurrency Control

As mentioned earlier, Squavy uses a pseudo peer to peer protocol with a small relay server between all connected clients. The fact that there is no central source of truth makes reliable data synchronization a challenging task, but there are a couple of solutions that solve this problem altogether.

3.12.1 Conflict-free replicated Data Types

The most prominent strategy is to incorporate conflict-free replicated data types, or `CRDTs` in short. It is a data structure that allows concurrent modifications at different timestamps from different clients and applies the changes in a way that all conflicts *eventually* converge.

CRDTs are most commonly used in a text based environment such as text or code editors (e.g. Zed[36]), but they can be applied to any sort of data structure to support more complex editing like in a canvas (e.g. Figma[37]).

3.12.2 Operational Transformation

Operational Transformation, or `OT` in short, is a technique to synchronize data between collaborators that has been used before CRDTs were invented and is the backbone of many leading programs like Google Docs[38] or Microsoft Word[39]. However, OT has one major disadvantage to its superior successor. It is not consistent in some cases and *may* lead to divergent replicas.

3.12.3 Collaboration in Squavy

Squavy uses a custom approach to synchronize updates between collaborators that is easy to maintain and integrate with our frontend framework. Basically, it neither uses OTs nor CRDTs, but rather *"last writer wins"*. It is not eventually consistent, but we are working hard to find a solution to incorporate a form of CRDTs in the future.

[36]Source: `https://zed.dev/blog/crdts` (March 30, 2025)

[37]Source: `https://www.figma.com/blog/how-figmas-multiplayer-technology-works/` (March 30, 2025)

[38]Source: `https://medium.com/coinmonks/operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447` (March 30, 2025)

[39]Source: `https://news.ycombinator.com/item?id=33820975` (March 30, 2025)

3.13 End-to-End Encryption

Security has never been a more prominent subject in today's world of internet communication. We do not want our servers to be able to read the unencrypted data, hence I decided to integrate full end-to-end encryption into Squavy. Note that the system is heavily inspired by Excalidraw[40] and many parts are taken from their blog post on end to end encryption within the browser: `https://blog.excalidraw.com/end-to-end-encryption/`.

When creating a collaborative session, a new symmetric encryption key is generated on the client's side using the browser's integrated crypto library.

Generating a symmetric crypto key

```
const key = await window.crypto.subtle.generateKey(
    { name: "AES-GCM", length: 128 },
    // allow the key to be extractable
    true,
    ["encrypt", "decrypt"],
);
```

Using the generated key, it is not possible to *"extract"* a JSON Web Key out of it, which essentially allows us to get a string representation of the encryption key that can be shared amongst collaborators.

```
const jwk = await window.crypto.subtle.exportKey("jwk", key);
return jwk.k!; // the key portion of the JWK
```

This key is then appended to the rest of the invitation link, but conveniently placed in the hash-part of the URL, so it is not sent to the server, but can still be read from client-side code.

```
http://editor.squavy.com/?session=u20I#34S8H8ZUfexJn9zPbOf7mw
                                    ^^^^ ^^^^^^^^^^^^^^^^^^^^^^
                                 session ID    encryption key
```

[40] Excalidraw (Software) `https://excalidraw.com/` is an open source, collaborative whiteboard.

When someone uses the collaboration link, the client extracts the key portion of the invitation link, reconstructs the JWK, and is able to communicate with other clients securely. The server simply passes through the messages and only needs to know basic parameters like if the packet should be broadcasted or unicasted to a specific user.

Importing a JWK encoded crypto key

```
return await window.crypto.subtle.importKey(
    "jwk", {
        k: key, alg: "A128GCM", ext: true,
        key_ops: ["encrypt", "decrypt"], kty: "oct"
    },
    { name: "AES-GCM", length: 128 },
    true,
    ["encrypt", "decrypt"],
);
```

The process of setting up an encrypted communication context happens fully automatically in Squavy and is deeply abstracted away to not interfere when developing new features. Behind the scenes, the data that the socket sends to the server is simply wrapped with a call to the encryption function that takes and returns an array buffer. After collecting the responses from the recipients, they are all individually decrypted and constructed into their runtime data type.

Broadcasting an encrypted message and receiveing the peer's responses

```
function broadcast<T extends C2CEvents>(
    event: T,
    payload: Parameters<C2CEvents[T]>
) {
    const responses = await socket.emitWithAck(
        "squavy:broadcast",
        await encrypt({ event, payload }));

    return Promise.all(responses.map(async ([id, data]) => {
        return [id, await decrypt(data)];
    }));
}
```

3.14 Bug Hunting in Foreign Code

During development of the collaboration server I have experienced lots of frustration and setbacks such as countless hours of fighting against the Rust Borrow Checker or debugging strange issues when collaborating with more than two clients; the list goes on forever. However, the strangest problem I have encountered was definitely the *inconsistent binary packet ordering of Socketioxide*. Many issues were my own fault due to lack of knowledge or minor mistakes at programming, but this one was a little different and took a while to figure out.

Problem analysis Initially, nothing appeared to be faulty when developing the server and testing the collaboration feature with my local machine. I decided to push the new version of the server onto our root server in Germany and connect my laptop as well as my desktop computer to a collaborative session; and that's where things fell apart quickly.

Connecting the first client worked fine without any errors in the console, so I connected the second client, which also seemed to work at first; at least for not very long. When moving the mouse, the browser console would be spammed with cryptic error messages originating from the `socket.io-client` library, immediately disconnecting both clients from the session. Essentially, the error boiled down to the message below, which is found in the `Decoder` class of the Socket.io parser:

```
ERROR: got plaintext data when reconstructing a packet
```

Figuring I started off by debugging the client to see what went wrong, which was an easy task because the Squavy editor still ran on my local machine. It was fairly obvious why the program crashed after inspecting the responsible code section[41]:

```
1 public add(obj: any) {
2     if (typeof obj === "string" && this.reconstructor) {
3         throw new Error(
4             "got plaintext data when reconstructing a packet");
5     }
6 }
```

[41] Source: `https://github.com/socketio/socket.io/blob/62e4da125e99d233b5d58a43002f0485a4a7234f/packages/socket.io-parser/lib/index.ts#L161-L166` (March 26, 2025)

The client received a plaintext web socket frame, although it expected one or more binary frame to reconstruct the current packet. What does this mean? Well, the Socket.io protocol states that a packet containing binary data consists of different web socket frames whereas the header (the plaintext part) contains all the primitive information in a stringified JSON format with the binary data interleaved as placeholder attributes. The actual binary data is then sent separately after the header:

```
// Packet:
{ type: BINARY_EVENT, namespace: "/", data: <Buffer <3d 29>>} }

// Encoded:
51-{"_placeholder":true,"num":0}
+ <Buffer <3d 29>>
```

After adding a `console.log()` to the decoder's `add()` function, I reran the experiment and noticed that my mouse position events were received in a completely wrong order. Why is that? Do web sockets not use TCP where all frames are sorted in the correct order? Well, after a quick Google search I made sure it does, so the problem must be within the server implementation itself.

How does one debug code on a production server? I spent countless hours on trying different methods of injecting or attaching a debugger into the binary on the remote machine, like creating a custom docker image with GDB[42] preinstalled, but all of them only led to other major issues and ended unsuccessful.

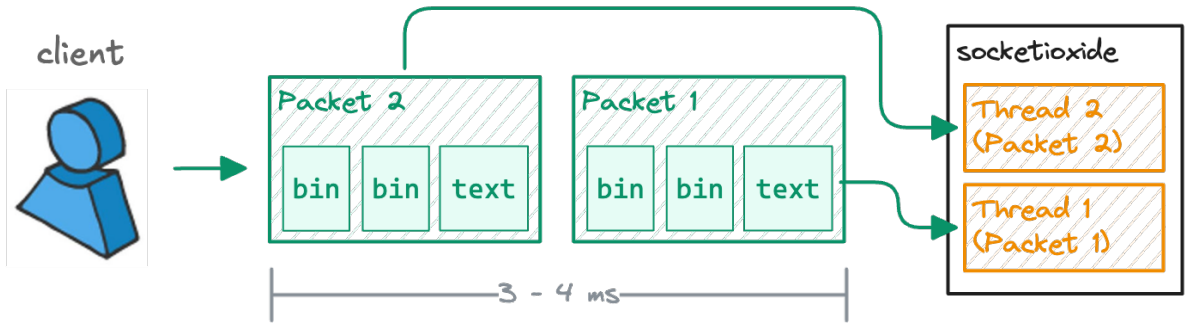
The second most obvious solution would be to examine the actual source code of Socketioxide and trace the issue back to its origin. However, given that the underlying engine powering Socketioxide is highly abstracted and involves multiple layers of complexity, gaining a deep understanding of it would require an extensive amount of time and effort, which consumes way too much time for resolving such an issue.

> *"There comes a time, when one must acknowledge their limitations. Yes, technically I can **[sacrifice my time to investigate Socketioxide]**, but do I really know what I'm doing? No..."* [43]
>
> – Matthew "Matt" K. C. Wong

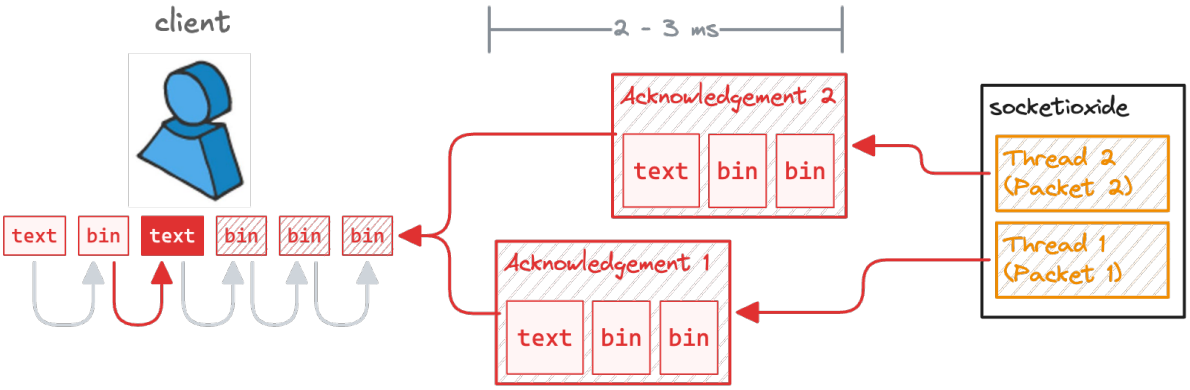
[42] GNU Debugger (Software) `https://sourceware.org/gdb/` is a debugger for Unix-like systems.
[43] Source: `https://youtu.be/ExwqNreocpg?si=cBgbToUKNbmGHMf-&t=530` (March 26, 2025)

Following the quote, I decided to visit Socketioxide's GitHub repository and create an issue[44] stating the problem and the likely cause as well as a possible solution, hoping to find other people with similar problems or more knowledge than me.

Proposal The following example tries to visualize the problem by ping-ponging two consecutive packets each with two binary payloads in a very small timeframe of only a few milliseconds. The browser client seems to send all web socket frames in the correct order to the server because all packets are successfully received and processed:



My educated guess tells me that Socketioxide processes all packets simultaneously with Tokio's concurrency model and has a problem with sending back the acknowledgement in the correct order. Usually, this is not a problem because all packets require equal processing time and are sent back in the same order as they arrive at the server, but hundreds, maybe even thousands of packets let the server left struggling and results in a mixed up order; the client expects two binary data frames after each plaintext header but instead receives a plaintext frame instead and triggers the aforeseen error:



In contrast, Socket.io only uses a single thread and has no issues with messing up the frame order, which I confirmed through testing a similar scenario with this library instead. Within the linked issue a developer noticed that the underlying problem may be fixed by

[44]Source: `https://github.com/Totodore/socketioxide/issues/232` (March 26, 2025)f

explicitly telling Tokio to use a single threaded runtime which would, however, result in the loss of the substantial performance advantage. Another solution to this problem would be to use a different packet protocol such as MessagePack[45] which conveniently supports embedding the entire packet's information including the binary data within a single web socket frame. This, however, only circumvents the problem and solely serves as a temporary solution instead of fixing the underlying issue. A few weeks later, a real solution was implemented by grouping the relevant frames into complete packets that are then sent entirely through a single outlet.

Why was this so difficult to figure out? Often, faulty execution behavior can be reproduced easily by trying different things and observing what and where something breaks. However, the issue stated above plays in an asynchronous context, which makes analyzing and reproducing the problem by many orders of magnitude more difficult. Additionally, the bug would not even trigger consistently; sometimes as short as two seconds into execution, other times nothing happened for up to ten seconds. Even when the bug occurred, it is very hard to trace back thousands of asynchronous function calls. And if this was not already enough, it was not even possible to attach a debugger onto the program because the bug only existed during production on a remote server. It is easier to actually logically think what could be going wrong and try to fix it from another point of view or through educated guesses.

3.15 Conclusion

Developing Squavy's collaboration server was a challenging yet rewarding journey. Choosing the right architecture, ensuring low latency, and implementing end-to-end encryption required multiple iterations and problem-solving on multiple layers.

Despite the challenges like battling the Rust borrow checker or finding a major bug in foreign code, the doors for even better real-time collaboration in Squavy are now wide open.

While the current implementation is decently functional, there is always room for improvement, especially in handling concurrency or improving scalability, which will be the next step for a great collaborative experience in Squavy.

[45] MessagePack (data format) `https://msgpack.org/index.html` is an optimized serialization format which natively supports embedding binary data.

4. Monitoring custom front and backend software

4.1 Introduction

Ensuring the reliability, efficiency and performance of applications is mandatory for maintaining flawless user experiences. Therefore, as the complexity of software applications rises, it is necessary to monitor custom front-end and back-end systems to optimize the performance and identify possible sources of error before the user comes into contact with them.

This thesis will examine the best principles of monitoring custom software, while focusing on current tools and techniques. Additionally, the architecture setup needed for monitoring setups and how to provide hand-crafted metrics for Rust applications is explored while focusing on Docker. I will go over the things that are worth monitoring and the decisions surrounding my choices.

4.2 Topic of Investigation

My topic explores if custom metrics can be made available in Rust and NodeJS software and if it is necessary to create a useful dashboard. The goal is to export useful metrics of our front and back-end application, which then can be used to further understand the server status or to scale services.

4.3 Intended Result

As a result of this thesis, I intend to provide a platform where the current system information as well as custom software data in the context of collaborative servers is provided. The aforementioned custom data stems from fully containerized[46] applications, which is

[46]Packaging programs with dependencies to run in a portable environment.

the reason for the platform to additionally be able to monitor the state of the deployed docker container applications. The created solution should be able to monitor the current state of the front-end application e.g. the CPU load and the total amount of current connections, as well as the amount of sessions and users currently connected to the back-end server. If feasible, the platform should provide an overview of the resources that the deployed applications take up. This should enable the user to determine whether bugs or an increase in user activity is responsible for resource intensive time-frames.

4.4 Understanding Monitoring Practices

Is is important to ask yourself, why something should be monitored and if it helps to identify what's broken or wrong. This helps to identify not only the current issues in the application, but it can give an insight into why that might be. That's why in the following thesis, I try to combine the statistics with each other, to create meaningful data that can be analyzed to see what is wrong before inspecting it closely.

4.4.1 White-box Monitoring

White-box monitoring is the action of monitoring metrics that is internal application data that only the developer is able to see. That means, white-box monitoring includes only internal application data. Request rates and user amount are examples for white-box monitoring that I am going to implement later. Other examples include: SQL queries in a database or monitoring the Java virtual machine .

4.4.2 Black-box Monitoring

This type of monitoring is not only the data that everybody can gather like response times but also metrics like the system resources. Although this kind of monitoring seems obvious, it is present in almost every monitoring platform and can increase the benefit the white-box metrics provide.

This difference is important to understand to effectively design and collect metrics for a useful monitoring platform, as white-box metrics can prove to give a better insight into the works of applications. [47]

[47] https://www.prisma.io/blog/monitoring-best-practices-monitor5g08d0b

4.5 Architecture and Tools

In the context of this thesis, the architecture of the to be monitored application looks like the following:

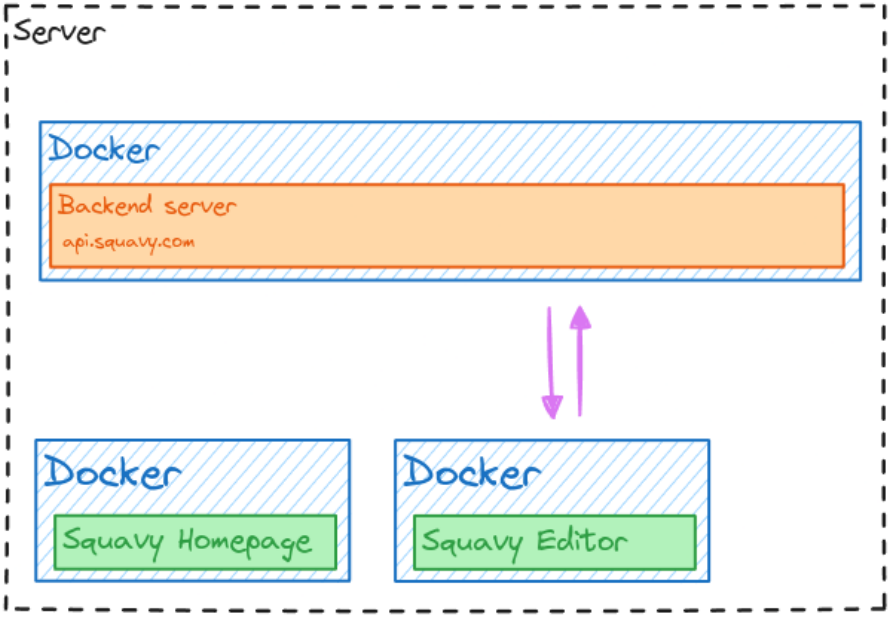


Figure 14: Architecture

The architecture follows a simple server-client design. There is one client, and one server respectively. The client does only communicate with the server to see if it is reachable until a collaborative session is created, which starts a concurrent communication between the two.

Squavy Editor A SolidJS front-end deployed to a NGINX[48] server that hosts a digital audio workstation[49]. The user is able to produce his own music following the MIDI[50] standard and using a custom modular synthesizer to create sound. This synthesizer is written in Rust using WebAssembly to achieve the needed performance. Most importantly, the user is able to create a collaborative session wherein multiple users can edit the same project. Therefore, a server is needed to facilitate communication between the editors.

[48]NGINX (Software) `https://nginx.org/en/` is a web server that manages resource load, mail and much more.
[49]A tool or studio to create digital music.
[50]MIDI (Protocol) describes how digital instruments communicate.

Squavy Homepage The homepage application is basically the same as the editor, made in SolidJS and deployed via NGINX.

Rust Collaborative Back-end The server is a Rust application, chosen for its performance. It uses Socketioxide[51] to communicate and manage connections with the front-end via WebSockets. Between the server and the aforementioned DAW, a Socket connection is created and end-to-end encrypted data is transferred in real-time. This encryption guarantees that any communication between the two applications stay anonymous and protected from unauthorized access.

The server's primary function is rather straightforward; it receives some kind of data in the form of packets which contain *packet types*, so the packet only needs to carry the specific information needed for processing. Upon receiving these packages, the server uses its internal namespace lookup to properly distribute the sent data to only members of the session accordingly. This can either be done by unicasting the data to every session member or by using broadcasts. A client is also able to request specific data from the server, which then will request the other clients in the session to provide the data for the former. This is especially useful in scenarios where two clients already established a session and created a project, whereupon another user joins and is able to request the project data from the other users. Thus, the server does not save any information about the created project.

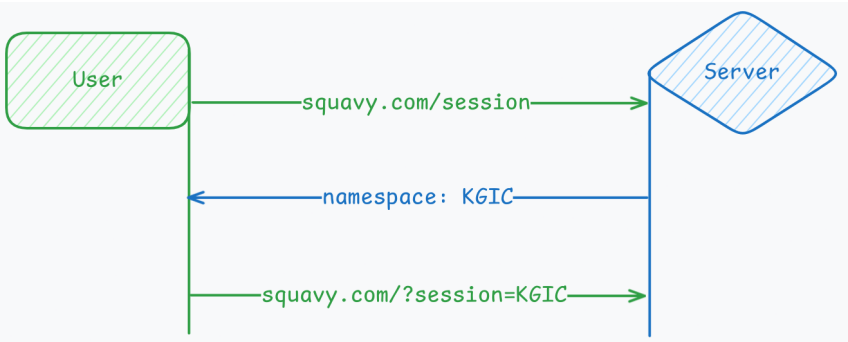


Figure 15: Session Creation

4.5.1 Prometheus

Prometheus is a software application that allows for monitoring of events and has further capabilities for alerting, and additionally ships its own Query Language (PromQL) that

[51] Socketioxide (Software) `https://github.com/Totodore/socketioxide` is a Rust based server implementation of the Socket.io protocol.

eases the evaluation of collected data. To achieve this, it stores all collected values alongside timestamps, in so-called time series databases[52], which enables it to retrieve the data in high speed. It collects this data by *scraping* metrics using simple HTTP requests from applications that serve their statistics. The targets for these scrapes are configured in the Prometheus config as jobs, which it scrapes automatically every few seconds.

Metrics Prometheus periodically scrapes configured targets using HTTP to gather exposed metrics. Usually, an application that provides metrics does so using an HTTP endpoint `/metrics`.

```
# HELP api_call_counter_total
# TYPE api_call_counter_total counter
api_call_counter_total{endpoint="/kurzparkzone"} 0.0
api_call_counter_total{endpoint="/parker"} 0.0
api_call_counter_total{endpoint="/parkschein"} 0.0
# HELP application_started_time_seconds Time taken to start the application
# TYPE application_started_time_seconds gauge
application_started_time_seconds{main_application_class="at.spengergasse.parkschein.Application"} 9.924
# HELP disk_free_bytes Usable space for path
# TYPE disk_free_bytes gauge
disk_free_bytes{path="/home/konrad/Documents/schule/5ahif/dbi/5ahif-dbi-parkschein/."} 5.67332216832E11
# HELP disk_total_bytes Total space for path
# TYPE disk_total_bytes gauge
disk_total_bytes{path="/home/konrad/Documents/schule/5ahif/dbi/5ahif-dbi-parkschein/."} 9.74365782016E11
# HELP executor_active_threads The approximate number of threads that are actively executing tasks
# TYPE executor_active_threads gauge
executor_active_threads{name="applicationTaskExecutor"} 0.0
```

Figure 16: Metrics Endpoint

The metrics are served as plain text and consist of a key, i.e. a name and a value. In the figure above, the metric `application_started_time_seconds` paired with a value of 9.924 seconds is provided.

These metrics can also be further filtered by labels. The `api_call_counter_total` metric has a label called `endpoint` to differentiate the requests by the route of the incoming API requests. To facilitate this, the metric is listed three times with different labels:

- `/kurzparkzone`
- `/parker`
- `/parkschein`

This can be seen later when I setup cAdvisor which provides labels for every docker container currently active.

[52] source

PromQL To evaluate these metrics, Prometheus uses its Prometheus Query Language, which is a small but powerful and flexible tool-set. It utilizes three different data types: *instant vectors*, *range vectors*, and *scalars* to enables users to aggregate and operate on the data in real-time.

Using the UI which Prometheus provides, it is possible to directly execute PromQL queries like these:

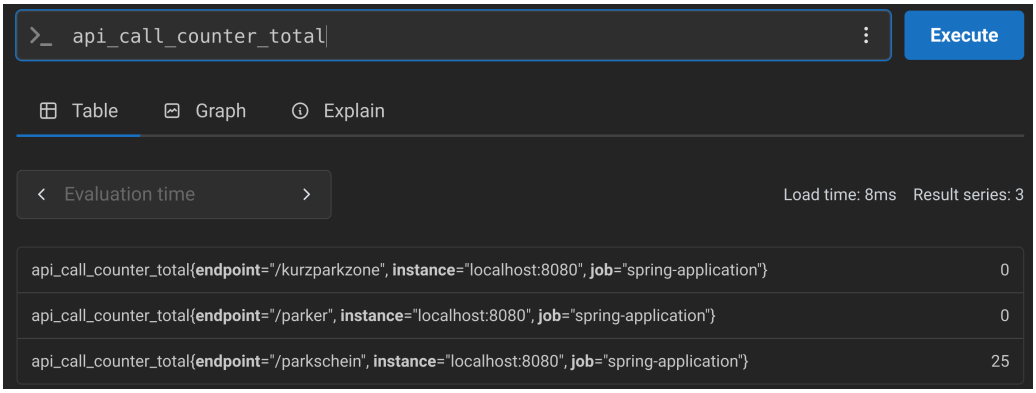


Figure 17: Instant Query Example

This query just displays the value that is associated with the metric for every endpoint, without any additional shenanigans. Thus showing that only the endpoint *parkschein* was called 25 times.

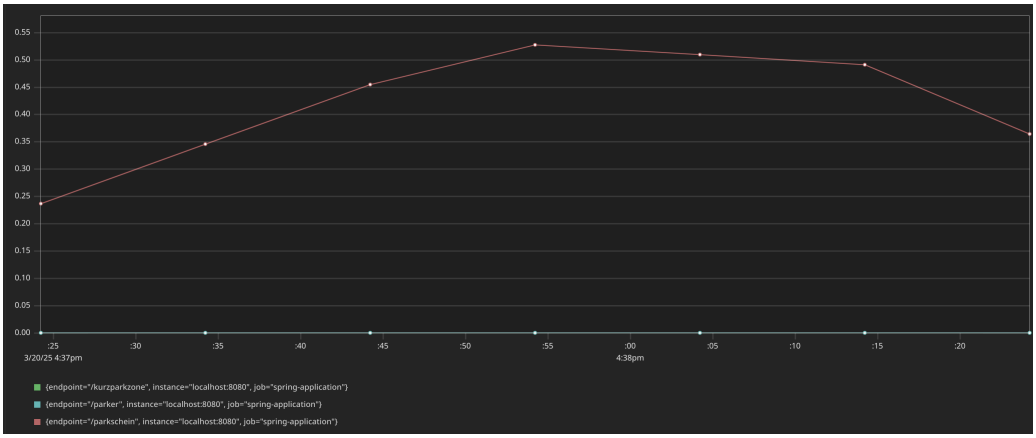


Figure 18: Range Query Example

If you want to get a rate of a metric in a given time frame, PromQL provides a simple *rate* function and range delimiters:

```
rate(api_call_counter_total[1m])
```

The brackets indicate the range of values, and the rate calculates the rate per minute. That is the reason why the query is a range query, because it selects a range of values e.g. in the last minute.

4.5.2 Grafana

Grafana is a platform for monitoring and visualizing a wide array of data sources like Prometheus, enabling users to customize permanent dashboards. It supports a lot of different sources like relational databases to centralize monitoring in one application and allows users to create dynamic visualizations and provide insight into their data. Additionally, it supports Alertmanager just like Prometheus to send alerts at specific thresholds.

Dashboards A dashboard in Grafana is just a collection of panels, providing a information-rich view of important information. These panels transform the raw data from any data source into charts or graphs to allow for effective monitoring while allowing the user to change the time frame in real-time. Another upside of these Dashboards is, that they can be shared with other users of the application. That means that if an application exports some metrics, they can also provide a Grafana Dashboard for users that already visualizes the data fittingly. Dashboards can be shared using a JSON file and can easily be imported.

The Prometheus graph, depicted in Figure 18, looks like this in Grafana:

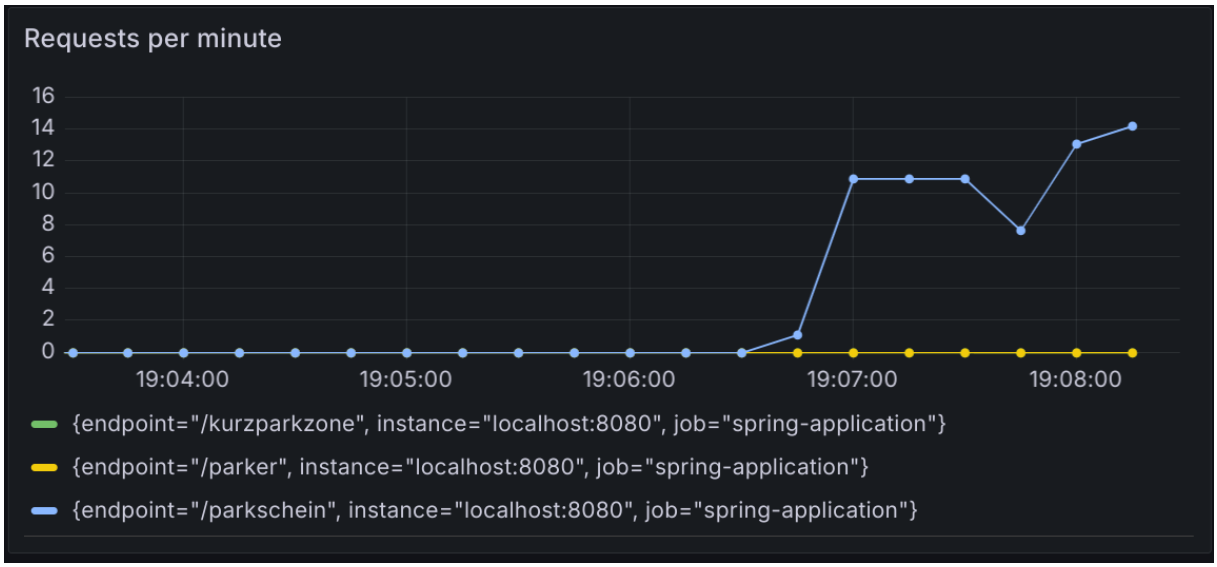


Figure 19: Grafana Range Query

Querying Evaluating the data Grafana collects from its data sources can be done using its querying capabilities tailored to the specific data source where the data came from. For example, if Prometheus is used as a data source, Grafana leverages Prometheus's integrated querying language, but if another source like MariaDB is configured, SQL can be used to analyze data.

Alerts While Prometheus also supported alerting, the goal of Grafana is to centralize the monitoring and alerting capabilities into one place, as seen by their large amount of supported data sources. The goal of these alerts is to inform an interested party or call a webhook[53] when a certain condition is met.

4.6 Tool Evaluation

While Prometheus is excellent at collecting and storing time-series data, it does not have the best visualization capabilities. By integrating Grafana and using them in conjunction, it is possible to achieve a rich and interactive dashboard that transforms the metrics from Prometheus and enables users to monitor the system performance effectively, possibly identifying anomalies, as well as making data-driven decisions.

Grafana offers an enterprise and an open-source version, each tailored to different needs. On the one hand, the OSS version is a free and open-source platform that supports all the necessary features like: visualizing and alerting. It supports a large amount of data sources and integrates with a number of applications like Prometheus.

On the other hand, Grafana enterprise is a commercial edition that is built upon the OSS version but includes additional features for larger organizations. Some of these are LDAP[54] and OAuth[55] support, which focus on the user management abilities of Grafana. Furthermore, it includes data encryption and support from the Grafana team.

For this project, only the OSS version is needed and is more than sufficient. Therefore, because both applications are open-source, they offer flexibility and the ability to adapt to changing environments.

Although each application is able to function without the other, the synergy between them ensure a vital monitoring tool that is more than just a gimmick and enhance the system reliability.

[53] A lightweight communication protocol to emit events between web applications.

[54] LDAP (Protocol) `https://www.redhat.com/en/topics/security/what-is-ldap-authentication` A directory access protocol to show information about organizations and persons.

[55] OAuth (Protocol) `https://oauth.net/2/` An open standard to manage user authentication in the web.

4.7 Configuring a Monitoring Platform

4.7.1 Setting up the Architecture

The whole architecture will be built upon the Docker Engine to utilize the easy-to-setup images the Docker Hub provides, alongside Dockers ability to easily manage containers. This will furthermore allow the platform to monitor itself, as it is running inside docker containers that automatically export metrics which can be gathered by cAdvisor.

Prometheus To setup Prometheus, it is possible to use the official Images uploaded to Docker Hub[56]:

```
docker run \
-p 9090:9090 \
-v /home/dev/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \
-v prometheus-data:/prometheus \
prom/prometheus
```

This command runs the official Image and makes it accessible via port 9090. Furthermore, a persistent volume `prometheus-data` and a bind mount `prometheus.yml` are created. The former stores all data scraped by Prometheus and other operational data, while the latter is the configuration file where scrape-jobs are defined.

Until the collaboration server or another application starts exporting metrics, the configuration will remain minimal, and Prometheus will only attempt to scrape itself:

Minimal Prometheus Configuration

```
global:
    scrape_interval: 15s
scrape_configs:
    - job_name: 'prometheus'

scrape_interval: 5s

static_configs:
  - targets: ['localhost:9090']
```

[56]Source: `https://hub.docker.com/r/prom/prometheus/` (March 30, 2025)

Grafana Just like Prometheus, Grafana also supplies their own official Docker Image[57] which is similarly easy to setup:

```
1 docker run -d -p 3000:3000 --name=grafana \
2 -v grafana-storage:/var/lib/grafana \
3 grafana/grafana-oss
```

Although Grafana also uses a persistent volume to keep track of its data, it is a bit different because there are two Images: a Grafana OSS version, and an enterprise edition. The difference between the two version is mainly additional support for further data sources, and user authentication methods. [58]

Connecting Grafana to Prometheus is just as easy as adding a data source to Grafana using its UI:

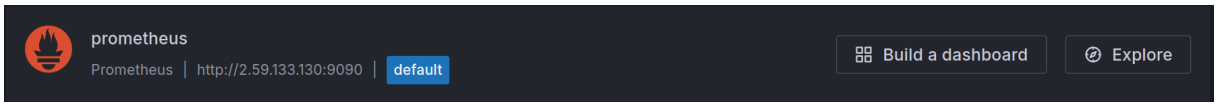


Figure 20: Prometheus Data Source

To ensure that only authorized applications can access the metrics collected by Prometheus, it is possible to create a shared Docker network. This setup prevents the Prometheus port from being exposed outside the machine, which limits direct usability, but the application itself can still be modified by editing the bound config file.

4.7.2 Exporting Metrics for Prometheus

Collaboration Server Using a rust Prometheus crate[59] provided by the cargo repository, it is possible to create and update metrics which can be served by Axum for example. This is especially useful in this case because it enables the application to keep track of:

- collaborative sessions
- processed collaborative packages
- connected users

[57] Source: `https://hub.docker.com/u/grafana` (March 30, 2025)
[58] Source: `https://grafana.com/docs/grafana/latest/` (March 27, 2025)
[59] Rust Client Library for Prometheus: `https://docs.rs/prometheus/latest/prometheus/`

To implement this functionality, I started by adding the Prometheus crate as a dependency. First, I added a route called `/status` to the Axum router[60] that will serve the metrics in a text format. Whenever Prometheus scrapes this endpoint, it retrieves the current values of all the defined metrics.

```
let app = axum::Router::new()
    .route_service("/session", ServiceBuilder::new()
        .service(get(handle_query_available_session))
        .with_state(session_store.clone()))
    .route("/status", get(handle_query_status))
        .with_state(reg)
    .layer(TraceLayer::new_for_http())
    .layer(socket_io_layer)
    .layer(cors_layer);
```

This route also carries a State which in this case is just the Registry, which I created beforehand. A registry is the central component where all metrics and collectors are registered and where they are stored to provide the values for the next Prometheus scrape.

```
let reg = Registry::new();
```

To define the metrics I created an additional file `metrics.rs` wherein all metrics are statically defined and can be registered using the `register_metrics` function. This enables the use of any metric from anywhere inside the project. So they can be accessed and modified wherever they should change.

Metrics

```
lazy_static::lazy_static! {
    pub static ref NAMESPACE_COUNT: Gauge = Gauge::new(
    "namespace_count",
    "Amount of sessions currently active").unwrap();

    pub static ref USER_COUNT: Gauge = Gauge::new(
    "user_count",
    "Amount of users currently connected").unwrap();

```

[60] Axum (Library) `https://github.com/tokio-rs/axum` is a web application framework for Rust that focuses on modularity.

```
    pub static ref REQUEST_COUNT: Counter = Counter::new(
    "request_count",
    "Amount of Squavy requests processed").unwrap();
}

pub fn register_metrics(registry: &prometheus::Registry) {
    registry.register(Box::new(NAMESPACE_COUNT.clone())).unwrap();
    registry.register(Box::new(USER_COUNT.clone())).unwrap();
    registry.register(Box::new(REQUEST_COUNT.clone())).unwrap();
}
```

The server increments the collaborative session counter whenever a new session begins and decrements it when a session is deleted 5 minutes after nobody connects to it. This functionality is implemented in the session store which is the data structure the server uses to keep track of its active sessions.

```
pub fn insert(&self, ns: String, session: Session) {
    info!("adding session {}", ns);
    self.write().unwrap().insert(ns.clone(), session);
    metrics::NAMESPACE_COUNT.inc();
}

pub fn remove(&self, ns: String) {
    info!("removing session {}", ns);
    self.write().unwrap().remove(&ns);
    metrics::NAMESPACE_COUNT.dec();
}
```

The user count as well as the total amount of requests processed can be tracked in the logic the server uses to connect the user to the correct namespace and the functions for unicasting or broadcasting respectively.

Putting all of these metrics together results in a small but efficient way of keeping track how many users currently and concurrently use the collaboration server. Now by sending a GET request to the `/status` route, it is possible to monitor the current server status.

```
# HELP namespace_count Amount of sessions currently active
# TYPE namespace_count gauge namespace_count 2
# HELP request_count Amount of Squavy requests processed
# TYPE request_count counter request_count 73
```

```
# HELP user_count Amount of users currently connected
# TYPE user_count gauge user_count 3
```

The only thing missing for Prometheus to scrape this data, is the correct job in the config to let Prometheus know where to look for the data.

```
scrape_configs:
  - job_name: 'squavy-server'
    scrape_interval: 5s
    metrics_path: '/status' # Specify the correct endpoint
    static_configs:
      - targets: [ 'localhost:8000' ]
```

Now, Prometheus and Grafana can both read the data and evaluate it to monitor the server load.

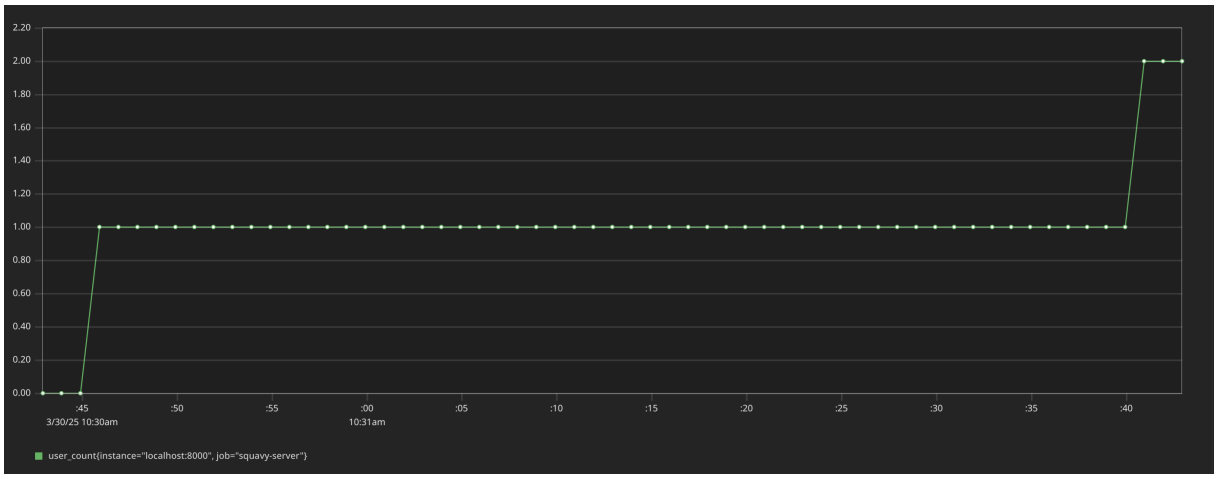


Figure 21: Prometheus Query of Server Data

Node It is possible to create custom metrics e.g. counters and gauges just like in Rust, in Node using a Prometheus client. But in this case, the only data worth collecting, is the amount of requests the server has to deal with. Because our front-end is designed to be accessible without an account or anything else, the only things that could be measured, are things like the amount of patterns created or the amount of notes places, which does not provide any benefit to the monitoring platform. So the only reasonable data is the amount of requests, which can be just as easily done using NGINX which is ultimately used to serve the web application.

`https://www.npmjs.com/package/prom-client`

NGINX Servers Gathering data from the relevant NGINX servers is even easier, because NGINX OSS includes a `stub_status` page that exposes metrics like: total http requests, active connections, and handled connections. Although NGINX has a built in feature to expose these metrics, they are not in a format that Prometheus can read, so to add the server as a data source, a workaround is necessary.

The NGINX Prometheus exporter[61] does exactly that. It collects the necessary data on the `stub_status` page and transforms it into a Prometheus readable format. Additionally it also provides a Grafana Dashboard that can be used to get a quick overview of the NGINX server. Another upside of this tool is that, it is intended to run as a docker container, so it is just as easy to setup as NGINX itself.

So to sum up, the front-end will be deployed in a NGINX container which itself will export its metrics on its `stub_status` route, which then can be scraped by the exporter and Prometheus.

First, the NGINX server needs to export its `stub_status`. This can be achieved by amending the configuration file of NGINX itself:

```
1 location = /status {
2     stub_status;
3 }
```

This is an excerpt of the `default.conf` file in the NGINX server. The location defines the route, so this would result in a URL of `http://<ip>:<port>/status`.

Now, when navigating to the status page, it looks like the following:

```
Active connections: 1
server accepts handled requests
 4 4 61
Reading: 0 Writing: 1 Waiting: 0
```

Figure 22: Stub Status Page

[61] Prometheus exporter (Library) `https://github.com/nginx/nginx-prometheus-exporter` is an interface to export third-party analytical data as Prometheus metrics.

The data just contains the amount of current connections and their state, as well as the amount of requests the server processed. This data will be intentionally reset once the NGINX server is redeployed, because retaining the metrics between different deployments would not only be difficult but also useless.

Although this data is already rather small and accessible via HTTP requests, it still needs to be converted by the Prometheus exporter into a format that can be collected by Prometheus itself. This is where the Docker image of the exporter comes into play:

```
1 docker run -p 9113:9113
2     nginx/nginx-prometheus-exporter:1.4.0
3     --nginx.scrape-uri=http://localhost:8080/status
```

After running the container, the data from Figure 22 can be accessed in a Prometheus readable format by navigating to `http://<ip>:9113/metrics`

Finally, after amending the Prometheus config and adding an additional scrape job for the exporter, it is possible to read the current connections to the NGINX server, and evaluate its health.

For example, it is now possible to gather the amount of requests in a given time frame, and calculate a rate of requests, with this PromQL query:

```
1 rate(nginx_http_requests_total[5m])
```

cAdvisor To gain a valuable insight into the system performance, it is essential to monitor not only custom application metrics but also the system metrics itself, such as CPU or memory usage. While custom metrics provide application-specific data, system metrics will always offer insight into the broader infrastructure health.

Because the whole architecture is deployed using Docker, cAdvisor[62] (Container Advisor) is a powerful tool that collects the resources and performance data from running containers. It exports information about the memory, disk, and network usage in real-time and in an Prometheus readable format. Although monitoring how many users are currently using a service, monitoring the CPU usage of the service itself can be crucial in determining performance issues.

[62] cAdvisor (Program) `https://github.com/google/cadvisor` is used to see resource usage and more about Docker containers

Additionally, cAdvisor is also deployable using Docker, so currently, not one application runs natively on the host. Google provides an image on Docker Hub which can be deployed using a single Docker run command and afterwards it is up and running.

Now it is possible to either use the built-in frontend that cAdvisor publishes, or to configure Prometheus to scrape cAdvisor like the rust Server. To combine everything in a single space, I will configure an additional scrape job for Prometheus:

```
1 scrape_configs:
2     ...
3     - job_name: 'cadvisor'
4         scrape_interval: 5s
5         static_configs:
6             - targets: [ 'localhost:8080' ]
```

Using this configuration, Prometheus now supplies metrics like:

- `container_start_time_seconds`: the time the container took to start
- `container_memory_usage_bytes`: the amount of memory the container uses in bytes
- `container_cpu_usage_seconds_total`: the total amount of time of CPU usage in seconds

Using these metrics in conjunction with custom application data is important to monitor the performance of a service. For example, by comparing the CPU usage with the amount of concurrent sessions hosted by the server, it is possible to visualize the efficiency of the server and how much resources per session are needed to manage them. Furthermore, statistics like the used bandwidth by a container can help determine how much network traffic these sessions take up.

4.8 Visualizing and Evaluating Metrics

The most important part of this process, is to create meaningful and useful visualizations and alerts to create a powerful monitoring application that effectively provides an overview of the current state of the deployed applications.

Figure 23: CPU usage of the Squavy-Server

4.8.1 Grafana Dashboards

Because in the earlier parts of this thesis I praised these dashboards, I will now create two dashboards for the server and client respectively. By combining the correct metrics collected by the different exporters and tools with each other, it is possible to create helpful insights.

Collaboration Server The server currently provides these metrics additionally to the ones provided by cAdvisor:

- namespace or session count
- user count
- request count

The most obvious visualization would be the resources the server currently needs. This is done by taking advantage of the cAdvisor metrics using this PromQL query:

```
container_memory_usage_bytes{name="squavy-server"} / 1024 / 1024
```

This metric exports the memory usage of the container in bytes, which I converted to MB by dividing by 1024 a few times. A simple but useful query that results in a nice visualization:

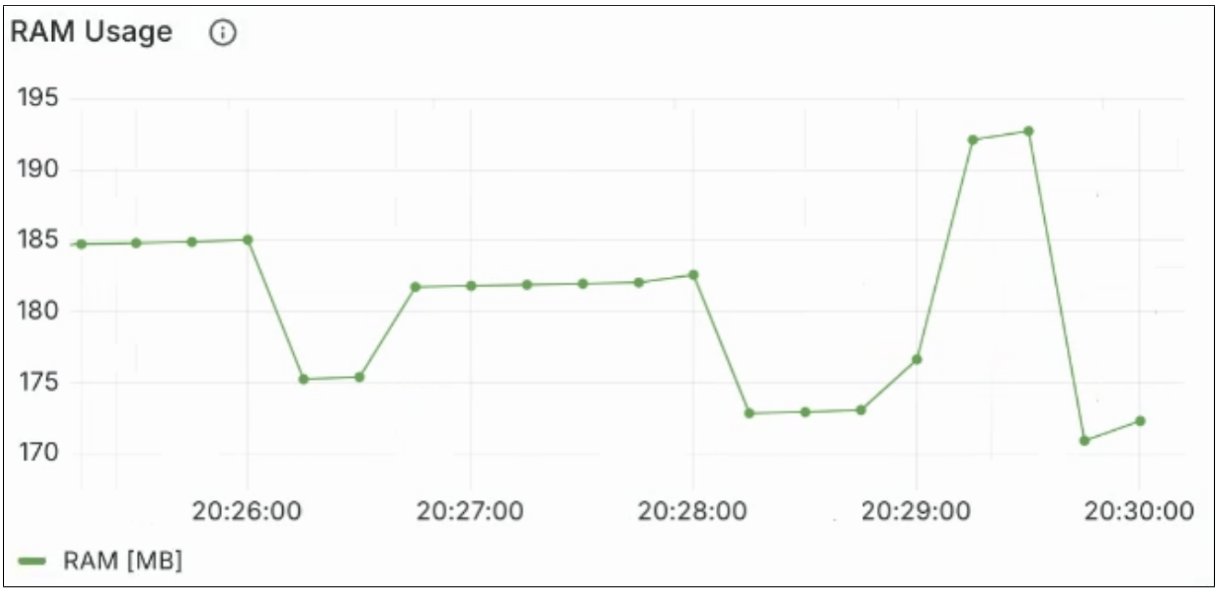


Figure 24: Server RAM Usage

Although this is a good graphic, to profit the most from these metrics is by comparing them. This time, I create a diagram to compare the CPU usage of the server to the current user count.

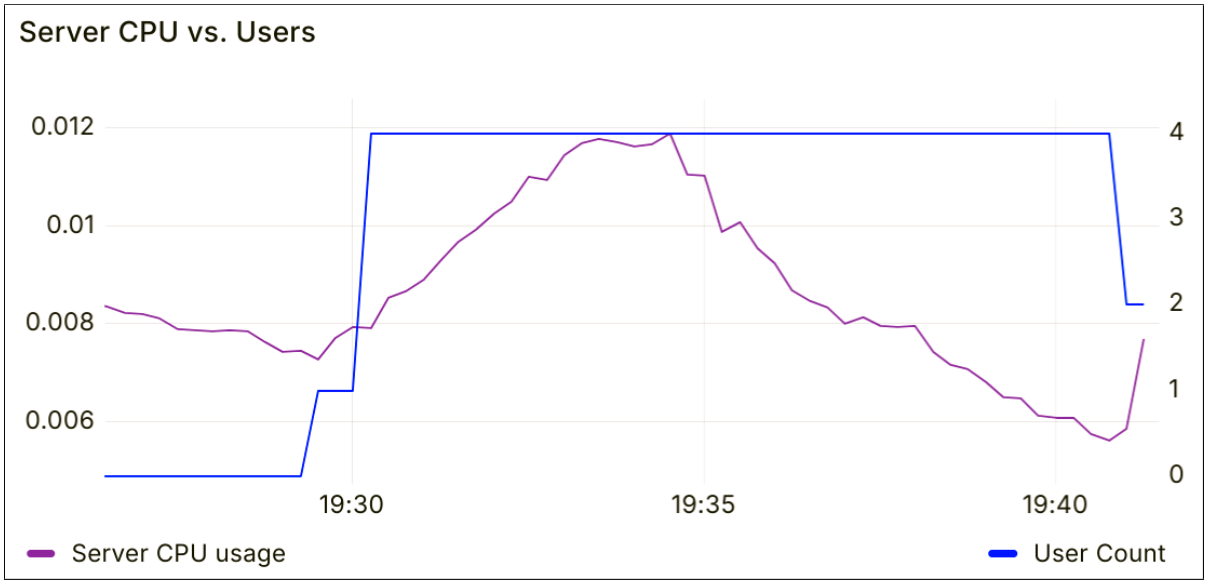


Figure 25: CPU vs. User count

This diagram proves useful when an update is made to the server and suddenly the collaborative feature stops working effectively or starts to lag. Using this visualization, it is easier to determine a correlation between the CPU load and the amount of users using the feature. It goes to say that of course there will be a correlation, but the important part is how big it is. If the CPU load jumps massively with each user, it is safe to say that some adjustments are needed.

Now, same as before, but this time comparing the session amount with the request rate. This will shine some light upon the amount of requests a collaborative session produces. This can prove helpful when minimizing the server resources that the server consumes.

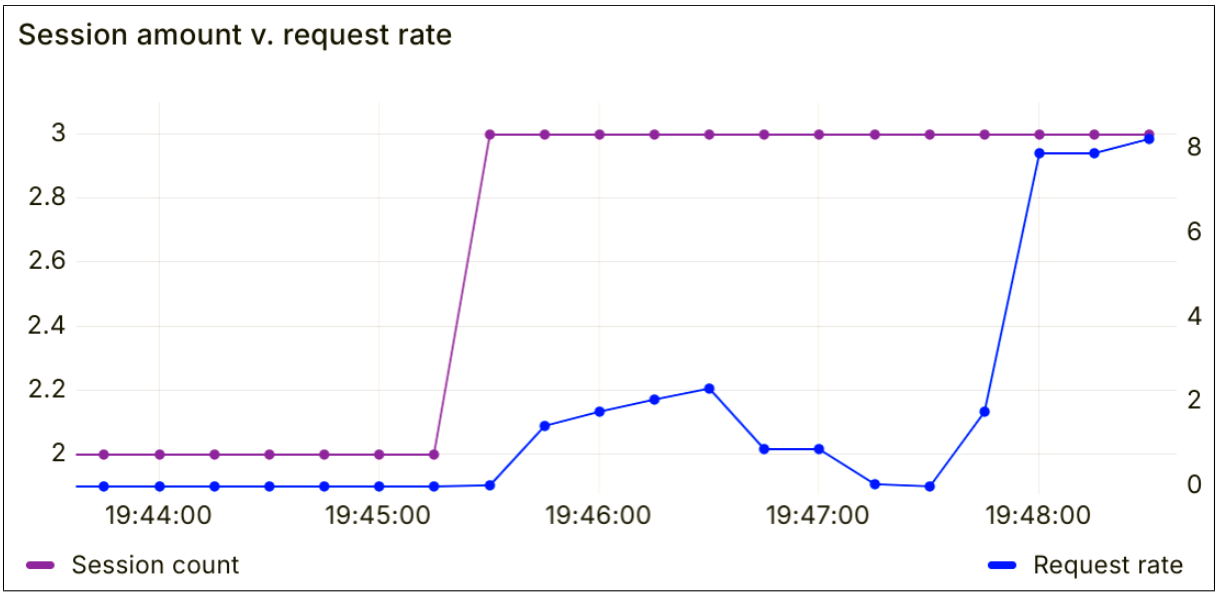
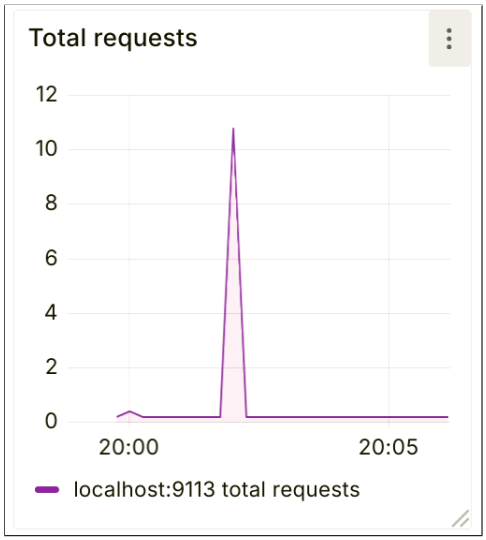


Figure 26: Session Amount vs. Request Rate

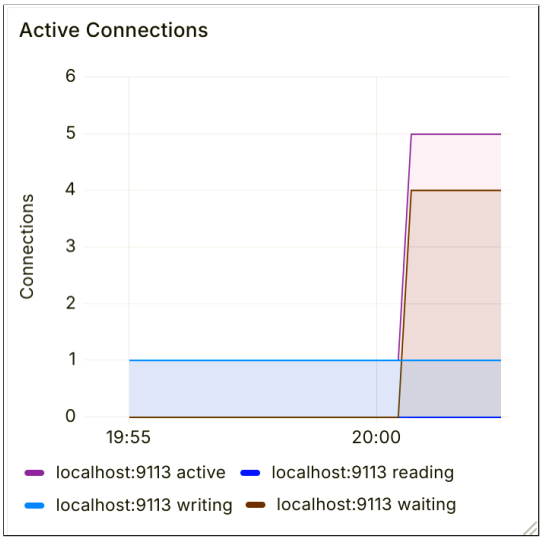
Further visualizations can be made, like a comparison between the amount of sessions and users to estimate how the users are distributed among the collaborative session, or a graph that depicts the amount of total requests processed. Because Grafana enables users to create such queries on demand and quite easily, more visualizations can be set up on demand.

Squavy Client Squavy does not have lots of different metrics that can be displayed in the administrative dashboard, but still a handful of essential statistics that are important to keep our service intact. Essentially, we track the current active connections, the average network throughput of our NGINX and collaboration server and the amount of processed HTTP requests. These metrics can be used to identify threats like DoS attacks when calculating the rate of incoming requests in a given time-frame and makes it easy to trace back the origin of such attacks and potentially issue bans to specific IP-Addresses.

Additionally, it is possible to visualize the amount of current connections. NGINX differentiates them between active, reading, waiting, and writing:



(a) Request Rate per minute



(b) Connection Graph

4.8.2 Alerts

It is important to receive notifications if our services take an unexpected turn, such as leaking memory or exceeding the overall capacity. To get a hold of these issues before the server collapses, I can create specific alerts that trigger an action when specific thresholds are surpassed. Grafana makes configuring these events through the web interface effortless.

In the following figure I created a notification that sends an administrative email to our address if one of our Docker containers uses too much RAM, e.g. if the last read value exceeds `1500MB`. This also helps to detect memory leaks early, and does not require observing the metrics 24/7.

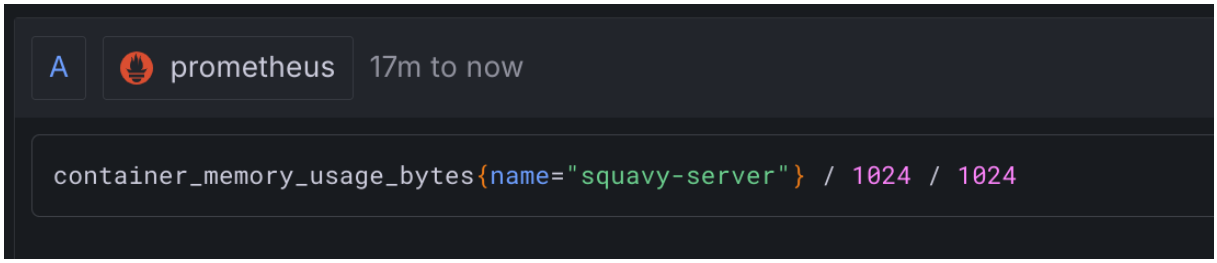


Figure 27: Alert Query

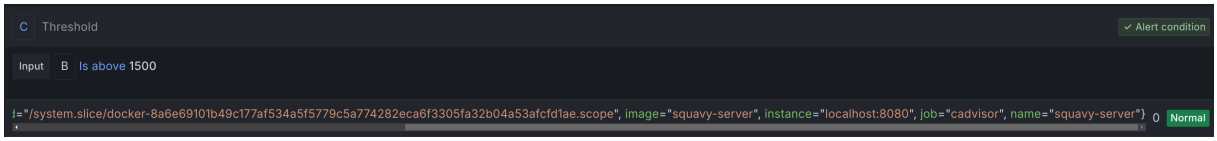


Figure 28: Alert Evaluation

During the test run, the server did not exceed `200MB` of RAM, so the status of the alert always remained normal.

The same principle can be applied to many different scenarios, such as checking if a specific active user count has been surpassed or even to automatically start or shutdown additional servers based on factors like memory or CPU usage but that does not fit into the scope of this thesis. One other case where I made an alert, is the state of the front-end. Although this will send an alert when the application is restarted, in cases of bugs this will prove important to track whether there are some ways the front-end can be overloaded to the point of failure.

4.9 Conclusion

It is safe to say, that a platform which provides a good overview of the current system functions, as well as the application metrics, will prove to be detrimental in the further development of the project. In the process of writing this thesis, I developed a web interface using Grafana that not only enables us to monitor the activity, but also helps us to identify issues, which has already happen in the course of writing this document.

Initially, I thought there was no need for a monitoring solution for our project, but while I researched the available technologies and started to implement the custom metrics, I truly understood what this can be used for. Although the project that the platform monitors is not very large, it still benefits from the monitoring setup by providing a foundation on which to base data-driven decision on.

The tools used to achieve this were rather well picked and suited the project. Furthermore, the architecture that was needed to set all this up, was very easy to configure and to understand. Moreover, it suited the already existing architecture for continuous integration/delivery and is also able to monitor the applications needed for this process.

5. Deployment and Scalability Strategies and their Implementation for Web-Applications

5.1 Introduction

In our rapidly changing digital world, it is crucial to ensure that software can appropriately handle fluctuating user numbers and maintain high performance at the same time. The more features a software product contains, the more difficult it is to design an appropriate architecture for it, since it must not only function, but also scale efficiently and deploy seamlessly. Without careful planning, platforms can face major challenges such as inefficient distribution of resources and difficulties in handling growing user bases.

5.2 Topic of Investigation

This thesis contains an analysis of current methods on how to effectively deploy an application and achieve high performance, making the process as seamless as possible. The focus will be placed on:

- Examining modern deployment and scalability approaches
- Assessing the effectiveness of various methodologies
- The role of containerization in software architecture
- Implementing a CI/CD[63] pipeline for automated builds and deployments

[63] Continuous Integration/Continuous Deployment

5.3 Scopes and limitations

While the study will cover various techniques, I will not deeply dive into security concerns or detailed cost analysis of different deployment models. Additionally, this thesis will focus on web applications, excluding other types of applications or industry-specific projects.

5.4 Intended Result

The primary objective of the thesis is to identify and analyze strategies to optimize scalability and efficiency of application deployments. As a result, the outcome of this research is a conception of a scalable deployment for web applications, with a particular focus on *"Squavy – Browser Based Digital Audio Workstation"*. Moreover, this thesis aims to provide an insight into how some practical methods can be implemented.

5.5 Importance of Scalable Deployment Strategies

Why are safe and reliable deployments crucial for success? First of all, we live in an ultra-competitive, networked world, where you basically cannot go through the day without encountering some sort of software. Smartphones, public transportation and CCTV are only a few areas of application out of millions.

But what does this mean for businesses? Scalability and reliability are crucial factors for apps that are aiming for long-term success. There are numerous addressed business fields, one of them being user satisfaction. Usually, users have little patience for slow or inconsistent applications, which is why as a developer you need to ensure that the app's behavior is performant and consistent to grant yourself the loyalty of the users. Furthermore, by making sure that the application can handle increased traffic resiliently, one is more likely to optimize costs and grow a business more effectively. Having a reliable system hereby reduces the risk of unexpected downtime or disruption of the service.

5.6 Fundamentals and Principles

5.6.1 Scalable Web Applications

Scaling refers to expanding a web application's capacity to handle increased loads. This is crucial for apps to remain functional as the user basis grows. The key factor in achieving this is managing resources efficiently. As the number of users increases, application com-

ponents like servers and databases experience higher strain. There is no universal best practice for scalability. The right approach depends on the type of architecture you have as well as growth patterns and code structure.

5.6.2 Brewer's CAP Theorem

The CAP-Theorem visualizes the impossibility of an "all-in-device" in the context of distributed systems. It states that only two out of three criteria can be fulfilled at once, namely:

Consistency Ensures that all components in a distributed system have the same information on data. Hereby, this factor enhances integrity and reliability.

Availability A system's ability to ensure that services are up and running as well as accessible for the users, even when the infrastructure beneath faces failures.

Partition Tolerance Refers to the ability of a system to continue operating despite network failures which can interfere with the communication between the nodes of a distributed system[64].

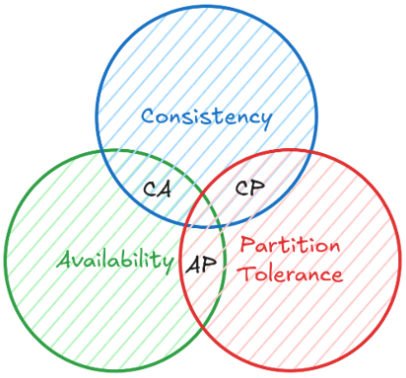


Figure 29: CAP Theorem

5.6.3 BaSe Principles

The BaSe model states that performance (Availability)
is the most crucial factor out of the three in distributed systems, while the other ones are in a conditional status based on system needs.

Basically-Available

- The system remains available in spite of network failures.
- Furthermore, it guarantees that every request gets a response in return, even if it is not accurate or a failure message.

[64] A network of computers that work together to complete a task.

Soft State

- The state of the system may change over time because system nodes are able to update their data, even without an input or new information.
- Soft State enables the system to be highly distributed and fail-safe, but it may provide outdated data.

Eventually Consistent

- This principle allows all nodes in a distributed system to eventually reach a consistent state, assuming no further updates are made within a certain period of time.
- The most important trade-off here is that consistency is sacrificed in favor of higher availability and partition tolerance, ensuring that the system can continue to function despite network splits or other issues.

5.7 Deployment Strategies

5.7.1 Overview of Modern Strategies

Big Bang Deployment The Big Bang Deployment is perhaps the simplest strategy out of them all, but also the riskiest. Hereby, all changes are deployed to production at once. This might sound less time-consuming than incremental deployments, but it is not. For this to work, a high amount of coordination and testing within a short time span is needed to ensure that the different components will work together. If something was to go wrong, it would be more difficult to reverse the process.

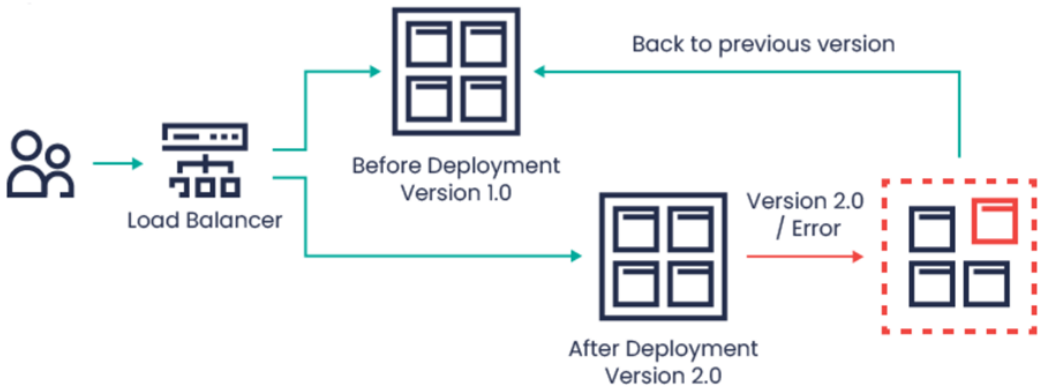


Figure 30: Big Bang Deployment Strategy

Continuous Deployment Continuous Deployment is a popular approach used in the industry where code changes are automatically released into the production environment. Before that occurs, the code undergoes an automated building and testing process to ensure that the code works correctly. Once those changes pass the testing process, they are pushed into production. This process is not to be confused with Continuous Delivery, where the process is similar, but with manual approval to release the software.

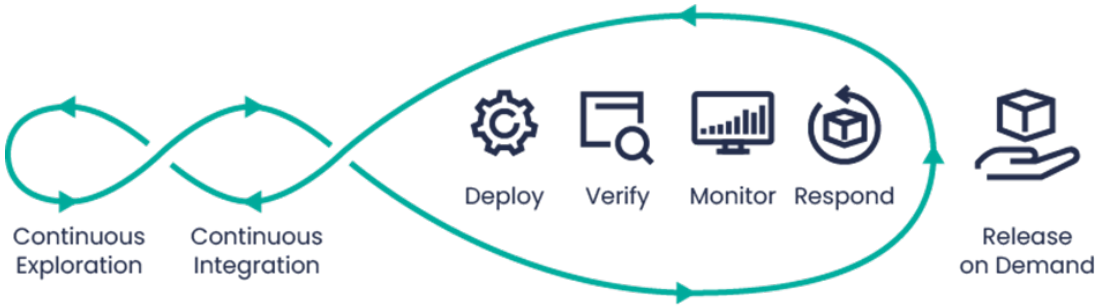


Figure 31: Continuous Deployment Strategy

To fully automate such a strategy, developers will need to eliminate manual regression testing, which can often be quite expensive. Additionally, project management meetings that were needed in order to increase collaboration and transparency before rolling out a new update, e.g. release planning and approval meetings, will no longer be required.

Some key takeaways are that businesses can reach a faster Time-to-Market[65] (TTM) by rapidly delivering software releases. Furthermore, quality and customer experience are enhanced, and the strategy is also cost-effective because automation eliminates tasks that otherwise would have needed to be done manually. By implementing CD, developers get to dedicate their time to implementing other features instead of focusing on manual deployment procedures.

Blue-Green-Deployment Also known as the red/black deployment strategy, Blue-Green has two different versions of the software running simultaneously, namely an old and a new one. The older version is referred to as blue or red, and the newer one as green or black.

When using this strategy, only one version of the software is live at a time. Then, when updating the program, it is done in the version that is not live. For example, if the blue environment is currently in production and the developer team wants to release a new update for the users, they deploy it to the green environment. After that, the users switch over to the green setting.

[65]The needed time for a product to get from initial concept to being available on the market.

This deployment strategy makes it possible to roll back changes easier or go back to an older version of the environment. Furthermore, it is a great way of testing the app accordingly before setting it live, and the methodology can be seen as a load balancer between the two versions. However, double the resources are required to run two different environments, which can be quite complex and time-consuming to set up.

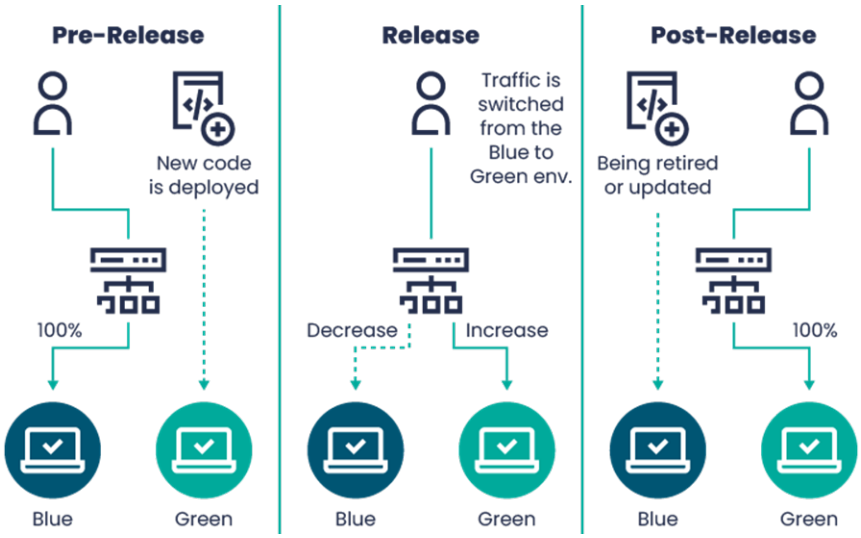


Figure 32: Blue Green Deployment Strategy

Canary Deployment When directing user traffic to a small amount of the user base so that only a few people see an update, it is called a "canary release". This enables the DevOps team to test functionality on a small group of users before making it available for everyone, which is especially practical for businesses that have different groups of users. Moreover, it can be used to gather feedback from a few individuals, which may be helpful in interpreting user behavior.

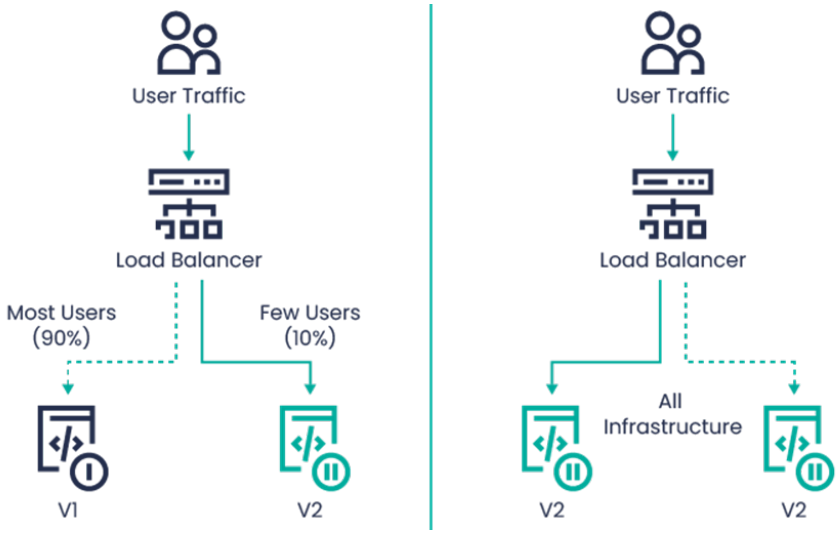


Figure 33: Canary Deployment Strategy

Shadow Deployment A Shadow Deployment involves running a new software next to the current one in production and is also known as Dark Launching. Hereby, the new version does not receive any actual user traffic, but a copy of it. This enables DevOps engineers to test out new features or the performance of the application before fully making them available to the public.

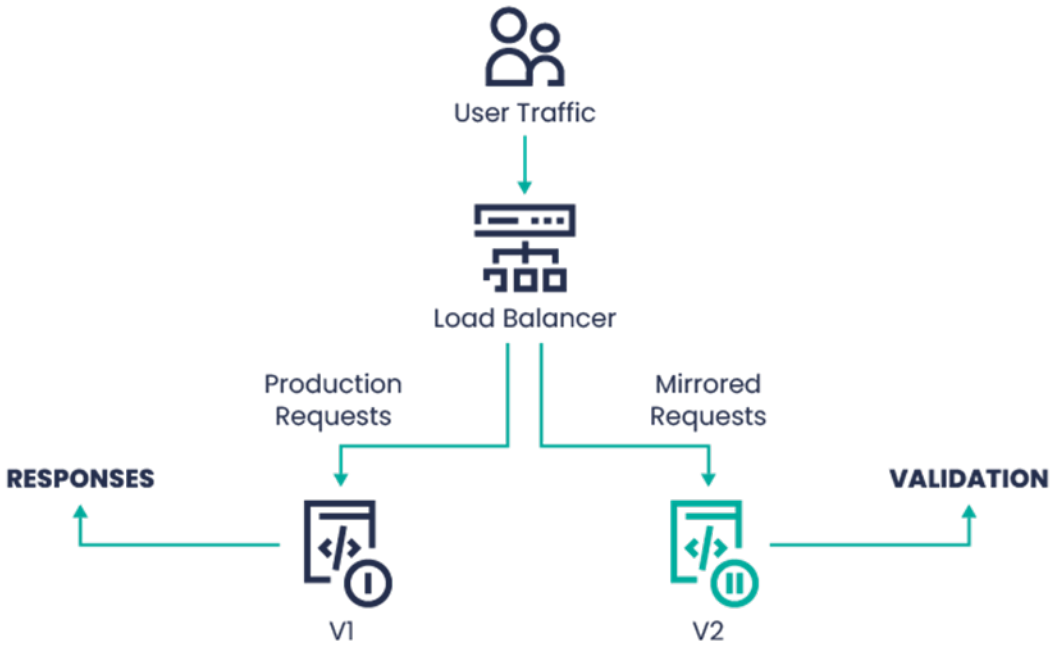


Figure 34: Shadow Deployment Strategy

This deployment strategy allows performance testing under real production conditions without having any impact on actual data load or user traffic, which is a great way of identifying threats or bottlenecks before they become a real problem. Nonetheless, this strategy can be complex to set up and manage, and may require additional resources in order to function correctly.

A/B Testing Deployment A/B Testing means running two versions of an application simultaneously, while one group of users sees one version of it and the rest the other one. By doing this, developers can compare performance and user-behavior of the individual versions. This strategy can also work as load balancing between different releases, like in the Blue-Green strategy.

This approach allows for decision-making based on data and how to optimize the webpage to enhance user experience, which can be useful when testing out new features. However, the difference between the two versions should not be too significant, or else the users might become confused.

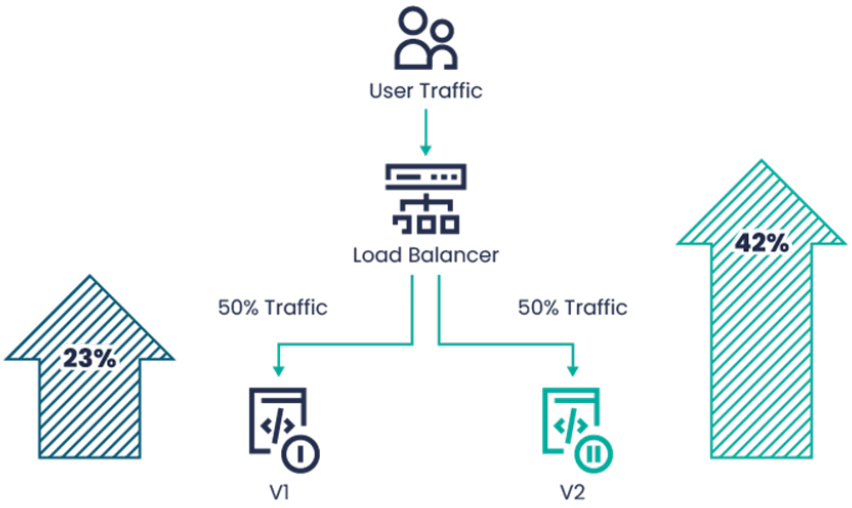


Figure 35: A/B Testing Deployment Strategy

5.7.2 Deployment Factors

When selecting the best deployment strategy for your product, many different factors need to be taken into consideration, some of them being:

- **Application Complexity:** Is my application rather complex? Does it need gradual rollouts like Blue-Green or Canary Deployments or are Big Bang Deployments enough?
- **Risk Tolerance:** If downtime must be avoided at all costs due to critical functionality of an application, e. g. in the health sector, a safer deployment method like Blue-Green or Canary is recommended.
- **Release Frequency:** Does the application have frequent software delivery processes? If so, a Continuous Deployment strategy containing an automated pipeline is the right choice for deploying new features.
- **Monitoring:** Make sure to keep an eye on the state of the application. If threats arise or errors happen, prepare to roll back the application to a stable state. This is especially important for strategies with high risk, such as Big Bang.

5.7.3 Conclusion

All in all, there is no general "best deployment strategy". When implementing an application, you should consider different options and be aware of the risks and opportunities that they bring with them. You might even have to use different strategies for separate parts or stages of your application.

5.8 Different Types of Scalability

There are two primary types: **vertical** and **horizontal** scaling. Scaling up an application vertically is relatively straightforward. You can add more power to your already existing resources such as CPU or RAM, which is cost-effective for smaller applications or at early development stages. However, you cannot upscale hardware infinitely.

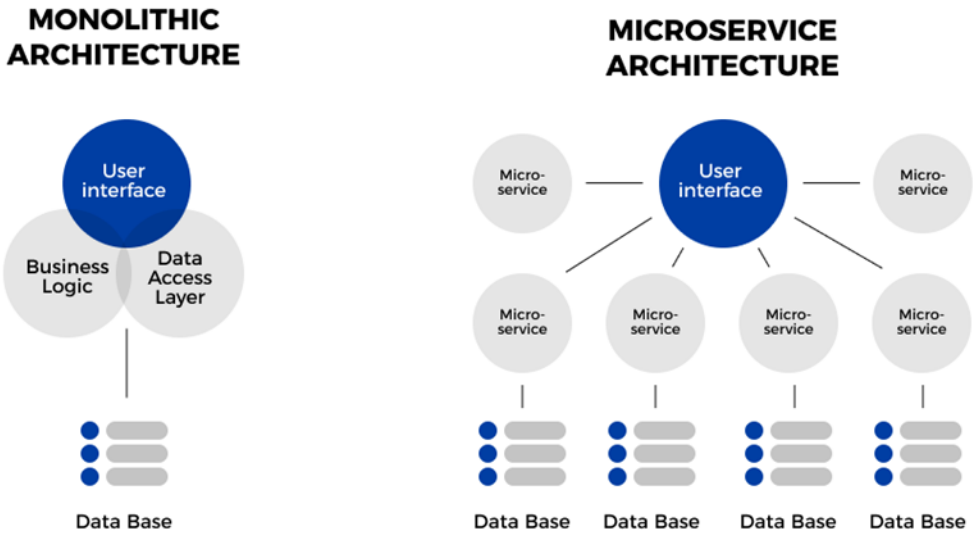
Moreover, scaling out an application horizontally involves introducing multiple instances of application services, most of the time across more than one server. This can be quite pricy, but if an instance goes down, there are many others who can take over and hereby ensure resilience.

There is also **diagonal** scaling that involves a hybrid approach which combines elements from both vertical and horizontal scaling. For example, you may upgrade existing infrastructure and add new hardware as well. It depends on your system's needs which type of scaling is best suited to implement.

5.9 Scalability Strategies

5.9.1 Adopt a Microservices Architecture

Instead of adopting a monolithic architecture, we have implemented several microservices in different GitHub repositories for our project Squavy, which break down the application into smaller components that are each deployable. We have decided upon this approach because it helps us to easily scale individual components such as the server and accelerate development cycles, which is rather difficult and more complex in a monolith.



5.9.2 Implement Caching Mechanisms

To improve response times and reduce server load, one should consider implementing caching into the application. Caching reduces the number of computations and queries the application needs to do for someone to access their data by saving already loaded data in an unused part of the memory. Some caching strategies include:

- Browser caching (on the user's device)
- CDN caching (on geographically distributed servers)
- Database query caching (results of common queries)

5.9.3 Optimize the Database

By regularly analyzing and optimizing queries as well as choosing the right database type for your needs you can increase your database performance. For example, you could start by indexing frequently accessed columns, which makes the querying process faster.

5.9.4 Deploy Load Balancers

Load Balancers distribute traffic accordingly balanced across multiple server instances to improve the overall web experience, which leads to higher availability and reliability. This helps to prevent a single server from becoming overwhelmed by sudden growth of data traffic.

5.9.5 Conclusion

All in all, ensuring that your software is performant is an ongoing process. By regularly planning and carefully considering different strategies like adopting a microservice architecture and optimizing your database, you can ensure that your application maintains that performance over time. This will not only improve user-experience, but also ensure that your application meets current and future standards.

5.10 Testing

Another important factor when developing web applications is testing them thoroughly before deploying them to a production environment. Testing helps discover errors, gaps or missing requirements. Furthermore, it involves implementing various checks to see if the application runs as expected, hereby ensuring that the software functions accurately. Some frequently used types of testing include the following:

5.10.1 End-to-End (E2E) Testing

End-to-End tests replicate user behavior on an application while running in a production environment and therefore ensure that all kinds of user flows work as expected. By doing this, developers can verify that a process works from start to finish without the app encountering any errors. E2E-Testing can be expensive to perform and hard to maintain in an automated environment. The difficulty of replicating user behavior can range from simply loading a web page or creating an account to verifying online payments or other notifications.

5.10.2 Performance Testing

This type of testing evaluates how much workload an app can withstand. Its goal is finding out the limits of software. One example for performance testing would be observing the performance of an application while executing an enormous number of requests, hereby determining how the system responds to that high amount of workload. Through performance testing one can measure a system's stability and reliability during increased traffic phases.

5.10.3 Smoke Testing

Smoke tests are quick tests that ensure that the basic functionality of a program is running correctly. Furthermore, they assure that the significant features of an application are working. This type of testing is especially useful when building an app or after deploying it to ensure that it builds and runs properly in the supposed environment.

5.11 Comparison of associated technical problems and their solutions

5.11.1 Automated Deployment of Web Applications

Automated deployment is essential for quick releases, since it ensures that each new version is consistently integrated into the production environment. Whilst many companies still do this process manually, it is recommended to automate it, e. g. by implementing tests and pipelines.

Continuous Integration/Continuous Deployment (CI/CD) Tools

Jenkins	An open-source automation server used for setting up CI/CD pipelines by implementing a Jenkinsfile. Supports customization and integration with many different plugins.
GitLab CI/CD	Platform for code management and deployment automation. Simplifies creating pipelines using a `.gitlab-ci.yml` File and has version control.
GitHub Actions	Offers native CI/CD functionality for GitHub repositories using GitHub workflows that are easy to set up.

Utility Analysis:

		Jenkins		**GitLab CI/CD**		**GitHub Actions**	
Criteria	**Weighting**	Points	Evaluation	Points	Evaluation	Points	Evaluation
Ease of Setup	30%	7	2,1	7	2,1	9	2,7
Cost & Licensing	25%	10	2,5	7	1,75	8	2
Pipeline Flexibility	20%	10	2	8	1,6	7	1,4
Integrations	15%	10	1,5	7	1,05	6	0,9
Scalability	10%	8	0,8	6	0,6	5	0,5
Total	**100%**		**8,9**		7,1		7,5

According to the utility analysis for Squavy, Jenkins is the most flexible and powerful because of its great customization. For example, there is the Blue Ocean plugin, which is greatly used when creating Jenkins pipelines because it facilitates the process by visualizing the stages of a pipeline, among other things.

For a student setting, GitLab CI/CD and GitHub Actions offer strong functionality with easier setup, while Jenkins is ideal if maximum customization is needed. In addition to that, Jenkins is free and open-source, while GitLab offers free tier with some limitations and GitHub Actions has generous limits for private repositories. However, there are some minor costs associated with hosting and infrastructure on the server.

Containerization and Orchestration

Docker	Packages applications and their dependencies into portable containers. Runs consistently across different environments.
Podman	An open-source container management tool. It has a similar CLI to Docker, but operates without a daemon by running containers as child processes

Orchestration tools like Kubernetes will not be implemented into Squavy yet, as it would exceed the scope of effort for the current user base.

Utility Analysis:

		 **Docker**		 **Podman**	
Criteria	**Weighting**	Points	Evaluation	Points	Evaluation
Ease of Use	25%	8	2	7	1,75
Cost & Licensing	15%	6	0,9	10	1,5
Compatibility	15%	9	1,35	8	1,2
Scalability	20%	8	1,6	7	1,4
Community	25%	9	2,25	6	1,5
Total	**100%**		**8,1**		7,35

Docker is the preferred container technology in our context due to its ease of use, community support, and extensive compatibility with existing tools. It is also a well-established technology used by most companies in the industry. However, Podman's advantages in cost and licensing should not be overlooked, especially in bigger projects with budget constraints.

According to the Docker license agreement, the Desktop version is free for small businesses occupying less than 250 employees and with an annual revenue that does not exceed $10 million. Otherwise, a paid subscription for professional use is required.

5.11.2 Scalability and Load Balancing

As user bases grow and traffic becomes more fluctuating, web applications need to be scalable and handle traffic properly to enhance reliability and performance. There are multiple ways of doing so, therefore it is important to understand the different types of scaling and load balancers.

Scaling Approaches

Vertical Scaling	More data means more storage within one machine. Increase or decrease the resource capacity (CPU, RAM) of existing instances. Runs consistently across different environments.
Horizontal Scaling	More data means more machines are used for storing it. Increase or decrease the number of nodes in a cluster/system.

Utility Analysis:

		Vertical S.		**Horizontal S.**	
Criteria	**Weighting**	Points	Evaluation	Points	Evaluation
Complexity of Implementation	25%	9	2,25	6	1,5
Cost Efficiency	20%	6	1,2	8	1,6
Performance	20%	7	1,4	8	1,6
Scalability Limits	20%	5	1	9	1,8
Fault Tolerance	15%	6	0,9	9	1,35
Total	**100%**		**6,75**		7,85

On the one hand, vertical scaling is easier to implement, since it happens within one machine. It generally is more expensive in the long run due to hardware upgrades and limitations. Additionally, it is also physically limited, since you cannot infinitely upgrade one single machine.

On the other hand, horizontal scaling can be complex requiring orchestration, but it can optimize resource usage better because of its distributed systems. Theoretically, it is physically unlimited, and it provides better fault tolerance as well as redundancy, which is also an important factor.

It really depends on the needs of your application which scaling approach to choose, like performance needs and cost considerations. In general, for applications expecting substantial growth that require high availability or need to handle unpredictable loads, horizontal scaling is the better choice.

Load Balancing Tools

NGINX	Popular open-source load balancer that handles HTTP, TCP, and UDP traffic. Is commonly used for load distribution and as a reverse proxy.
Kubernetes Ingress	Native load-balancing solution that directs traffic from outside a Kubernetes cluster to services within a cluster.

Utility Analysis:

		 **NGINX**		 **Kubernetes Ingress**	
Criteria	**Weighting**	Points	Evaluation	Points	Evaluation
Ease of Use	25%	8	2	5	1,25
Cost & Licensing	20%	9	1,8	7	1,4
Compatibility	20%	8	1,6	9	1,8
Scalability	15%	7	1,05	8	1,2
Community	20%	7	1,4	8	1,6
Total	**100%**		**7,85**		7,25

Nginx comes out ahead overall, particularly in terms of ease of use, flexibility, and customization, while Kubernetes Ingress performs slightly better in scalability and cost-effectiveness. As already stated though, orchestration tools like Kubernetes will not be implemented through the course of this project within the academic year of 2024/25, as they would exceed the scope of effort for the current user base. However, if the user amount will significantly increase in the future, we might take into consideration implementing orchestration tools accordingly.

5.12 Prototypical application

As of September 2024, Squavy uses Jenkins as a build server for deploying web applications. We created a Jenkins pipeline to automate various stages of our application's lifecycle, including code building, testing, and deploying in Docker containers. Jenkins was configured to pull the latest code from our version control system (GitHub), build it, run tests, and, if successful, deploy it within Docker containers.

Furthermore, Squavy uses Docker to create isolated and consistent environments for each stage of the application. To route external traffic to our Dockerized application, we use Nginx as a reverse proxy in front of the Jenkins and application containers. Nginx was configured to:

- Forward requests to appropriate Docker containers, typically by hostname or URL.
- Load balance across multiple containers if necessary.

With Nginx, we could expose only the necessary services to the public while keeping others, such as Jenkins, protected or accessible only internally

5.13 Selection and Implementation

5.13.1 Choosing the right architecture

Squavy is a stand-alone software and built upon a microservice-architecture to support the individual components. The application consists of several systems that are each separately developed in different GitHub repositories, namely a backend server that handles collaboration and a webserver.

5.13.2 CI/CD Architecture

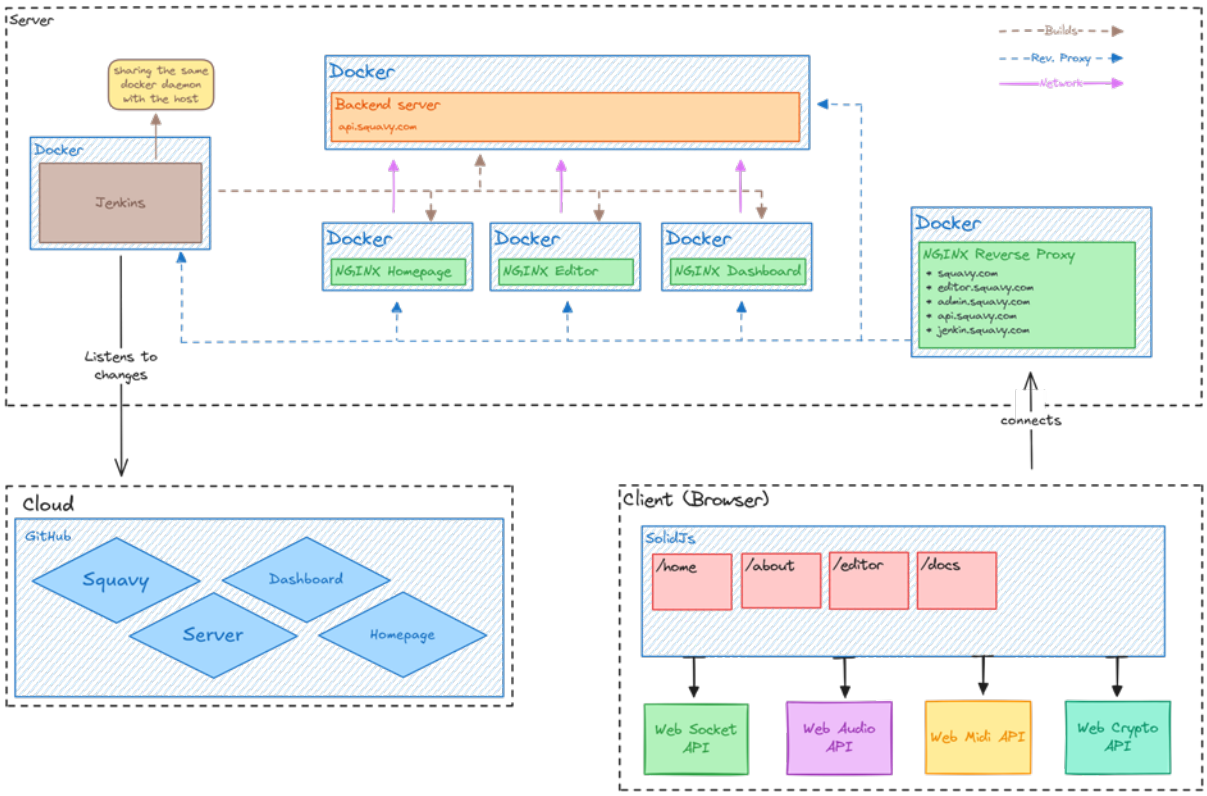


Figure 36: Squavy's CI/CD Architecture

On the lefthand side, you can see all GitHub repositories that are automatically built by our CI/CD pipeline in blue. The upper part of the plan describes the complete set of components needed to build, deploy, and host the Squavy product, whereas on the left there is Jenkins, which actively listens to changes on the repositories and builds them in isolated Docker containers. Then, the containers are launched to run the relevant backend services like the collaboration server or static sites like Squavy, its homepage and the dashboard. On the right side, all reverse proxies are depicted in a separate Nginx component, which serves as the outer interface for communication with the browser.

We have decided on using a Continuous Integration/Continuous Deployment (CI/CD) approach to be able to actively automate the process of building the application and deploying it onto a production environment. This setup ensures that new changes are continuously tested and integrated, hereby reducing the amount of work that we would have needed to do manually otherwise. By using Jenkins as our build server, we can trigger automated builds whenever a new commit is pushed to the repository.

5.13.3 Containerization

The first step into deploying Squavy was containerizing it. We have used Docker for this task, an open-source container technology which facilitates the process of building images and running containers.

An image is a lightweight and executable software package that contains everything you need to run an application, including code, system tools, libraries, dependencies, etc. Basically, it serves as a blueprint for creating Docker containers. Those are instances of the image that can be run on a host computer completely isolated from the rest of the machine.

To begin with, let us start with creating a Dockerfile in the root folder of our main service, the Squavy-DAW. We start by selecting a base image from Dockerhub, here we have used a rust-based image because it already comes with some essential tools for the build.

Squavy Dockerfile

```
1 FROM rust:slim-bookworm AS synth-rust
2
3 # Remove this once we go public
4 RUN apt-get update && apt-get install -y git openssh-client
5 COPY id_rsa /root/.ssh/id_rsa
6 RUN chmod 600 /root/.ssh/id_rsa
7 RUN ssh-keyscan github.com >> /root/.ssh/known_hosts
```

Squavy needs the Synth "SMSE" (Synth) dependency to work, which is another component that is developed in a separate repository. The folder containing the Synth needs to be on the same level as the Squavy project for it to find it. However, this is not so easily done in an isolated container, which is why to replicate it we need to clone that repository inside of the container. Authentication runs via an ssh-key as of March 2024. However, later as the repository will be made available to the public the access will work through https, so no git deploy key will be needed anymore.

So, we start off by creating an rsa-key and saving it. We copy our rsa-key into the /root/.ssh/ folder of the container. After that, we create a deploy key with the public key on GitHub. Deploy keys use an SSH key to grant access to a single repository, in this case the Synth repository. Now we should be able to clone the repository within the Dockerfile.

Squavy Dockerfile

```
RUN git clone git@github.com:Squavy-DAW/Synth.git /Synth
WORKDIR /Synth
RUN cargo install wasm-pack
RUN wasm-pack build --target web

FROM node:22-bullseye AS synth-javascript
COPY --from=synth-rust /Synth /Synth
WORKDIR /Synth
RUN npm install && npm run build
```

After installing other essential dependencies for the project, we start a multi-stage build to run the Synth and make it available for Squavy.

5.13.4 Building Aseprite from Source

The next step is to build Aseprite, an animated sprite editor and pixel art tool. Initially, we had to export the created designs from our Aseprite-project into other formats like PNG and GIF to be able to use them in our project. This was not very efficient though, because at the smallest mistake or redo you needed to update the file and repeat the whole process of manually exporting and converting it again.

This is why we have developed a Vite-plugin that automatically converts Aseprite-project files into PNG- and GIF-files to display in the browser. The plugin is placed in the vite.config.ts file and is responsible for automatically analyzing all files of the project. If it encounters any Aseprite project files, it runs the Aseprite CLI-Tool. Now we can correspondently build the frontend with Vite and edit Aseprite-files without the need to export them to other formats each time.

However, the Aseprite-plugin complicates the building process, because Docker does not have a way to use Aseprite when building the frontend. A solution to this problem is to build the program from source and include its compilation inside the Dockerfile. Hereby, we have excluded the frontend because that part of the application is not needed to load any files within our project and we have saved the path to Aseprite in an .env.local file. Although building Aseprite from source significantly increases build time due to its many dependencies, Docker caches the steps after the first run and therefore minimizes repeated delays.

Squavy Aseprite Dockerfile

```
FROM debian:bookworm-slim AS aseprite
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y git unzip curl build-essential cmake \
                       ninja-build libxll-dev libxcursor-dev \
                       libxi-dev libgll-mesa-dev libfontconfig1-dev

RUN git clone -b v1.3.13 --recursive --shallow-submodules --depth 1 \
    https://github.com/aseprite/aseprite.git /aseprite
WORKDIR /aseprite
RUN mkdir build
WORKDIR /aseprite/build
RUN cmake \
  -DCMAKEBUILDTYPE=RelWithDebInfo \
  -DLAFBACKEND=none \
  -G Ninja \
  ..
RUN ninja aseprite
```

5.13.5 Build Pipeline

Jenkins runs in a container itself on our server in Germany. To be able to run the builds and start other containers from within the Jenkins container, we have used the Jenkins server as Docker with Docker in Docker[66] (DinD). We start off by creating a directory `/var/jenkins_home` on the server as a volume, which we will later need to deal with access rights. Moreover, we create a Dockerfile, for example in " /Dockerfile" that looks like the following code-snippet.

Internal Jenkins Dockerfile

```
FROM jenkins/jenkins
USER root
RUN apt-get update && apt-get install -y lsb-release curl gnupg2
RUN curl -fsSL Ittps://download.docker.com/linux/debian/gpg 1 tee \
    /etc/apt/keyrings/docker.asc
RUN echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] \
```

[66] DinD: `https://medium.com/@yann.cardaillac/dockerized-jenkins-dind-c923774893dc`

```
    https://domlload.docker.com/linux/debi,-] \
    $(lsb_release -cs) stable" \
    | tee /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
RUN groupadd -f docker && usermod -aG docker jenkins
RUN touch /var/run/docker.sock && chown root:docker \
    /var/run/docker.sock && chmod 660 /var/run/docker.sock
USER jenkins
```

After starting the container on port 8080 and using the volume from before, we should have a functionable Jenkins build server that can work with Docker. Furthermore, after accessing Jenkins over the browser on port 8080 and creating an admin account as well as installing the suggested plugins, we set the Jenkins job to track a branch on our GitHub repository. Now, every time the developers commit and push new changes to the repository, Jenkins will be able to run a build of the current Jenkinsfile that contains the different build steps.

Even so, because it is a private repository and Squavy needs an RSA-key for authentication in order to build the other repositories, we need to make Jenkins aware of that. Herefore, we have implemented an additional stage in the build pipeline that retrieves an SSH private key stored in Jenkins credentials. It saves it as a file called `id_rsa` and then ensures that it has the correct permissions for secure use.

Adding an RSA Key to the build pipeline

```
stage('Prepare SSH Key') {
    steps {
        withCredentials([sshUSerPrivateKey(
            credentialsId: 'dockerfile-synth-ssh',
            keyFileVariable: 'SSH_KEY'
        )]) {
            sh 'cp $SSH_KEY id_rsa && chmod 600 id_rsa'
        }
    }
}
```

Then, we can finally start a build and deployment stage that builds the Dockerfiles and launches the newly created containers.

```
stage('Build') {
    steps {
        script {
            sh 'docker build -t ${IMAGE_NAME} .'
        }
    }
}

stage('Deployment') {
    steps {
        sh 'docker run --name squavy -d -p 8090:80 ${IMAGE_NAME}'
    }
}
```

5.14 Conclusion

All in all, deploying Squavy and designing a Continuous Integration / Continuous Deployment (CI/CD) architecture has been a challenging but rewarding task. The project gave me a deep understanding of modern development tools and methodologies, such as microservices, containerization, orchestration, and CI/CD automation.

In general, this topic not only requires lots of technical research, but also careful planning and collaboration, and most importantly communication, to ensure that the different components of the system work seamlessly with one another. While many challenges arose, implementing and working with different tools provided valuable insights into building efficient systems.

5.15 Lessons Learned

During this thesis, I explored development and scalability strategies while gaining hands-on experience and learning various implementation approaches. Through the CI/CD pipeline, we experienced automated, scalable processes and how to develop interdependent projects.

Throughout development, I also realized how important flexibility is in complex projects. Even small changes essential for Squavy's growth can lead to long bug-fixing sessions or require rewriting significant portions of code.

5.16 References

- https://master-spring-ter.medium.com/strategies-for-ensuring-software-scalability-and-performance-2813cc239736 (March 29, 2025; 17:48)
- https://vfunction.com/blog/application-scalability/ (March 29, 2025; 18:40)
- https://www.nearshore-it.eu/articles/deployment-strategies-101-types-pros-cons-devops-more/ (March 29, 2025; 19:52)
- https://medium.com/the-software-design-blog/top-10-system-design-concepts-everyone-should-know-066736a15a61 (March 29, 2025; 20:45)
- https://www.apwide.com/8-deployment-strategies-explained-and-compared/ (March 29, 2025; 21:00)
- https://www.youtube.com/watch?v=dvRFHG2-uYs (March 29, 2025; 22:49)
- https://medium.com/@josiahmahachi/exploring-deployment-strategies-for-modern-applications-c8dd48878868 (March 29, 2025; 23:10)
- https://www.ibm.com/think/topics/continuous-deployment (March 29, 2025; 23:50)
- https://www.tracers.com/blog/what-is-data-batch-processing/ (March 29, 2025; 23:55)
- CAP-Theorem and BaSe Principles: The theoretical content is from the subjects "Databases and Information Systems (DBI)" taught by DI. (FH) Johanna Niklas and "Business Applications (BAP)" taught by DI Joachim Grüneis in the academic year of 2024/25.
- https://www.geeksforgeeks.org/brewers-cap-theorem/#what-is-brewers-cap-theorem (March 30, 2025; 16:01)
- https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-for-enterprise-large-scaled-application-825436c9a78a (March 30, 2025; 15:05)

- https://medium.com/@pranabj.aec/acid-cap-and-base-cc73dee43f8c (March 30, 2025; 16:41)
- https://katalon.com/resources-center/blog/continuous-delivery-vs-continuous-deployment (March 30, 2025; 17:09)
- https://www.cloudflight.io/en/blog/monolithic-architecture-vs-microservices/ (Image, March 30, 2025; 17:43)
- https://circleci.com/blog/docker-image-vs-container/ (March 30, 2025; 23:00)
- https://medium.com/@yann.cardaillac/dockerized-jenkins-dind-c923774893dc (March 30, 2025; 23:20)
- https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (March 31, 2025; 17:03)
- https://www.techtarget.com/searchsoftwarequality/definition/End-to-end-testing (March 31, 2025; 17:03)
- https://commons.wikimedia.org/wiki/File:Logo_Aseprite.png (01.04.2025, 00:16)
- https://www.jenkins.io/ (March 29, 2025; 21:21)
- https://plugins.jenkins.io/blueocean/ (March 29, 2025; 21:38)
- https://docs.gitlab.com/ci/ (March 29, 2025; 21:28)
- https://github.com/features/actions (March 29, 2025; 21:30)
- https://docs.github.com/en/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions (March 29, 2025; 21:54)
- https://www.docker.com/ (March 29, 2025; 22:22)
- https://docs.docker.com/subscription/desktop-license/ (March 29, 2025; 22:35)
- https://podman.io/ (March 29, 2025; 22:22)
- https://nginx.org/ (March 30, 2025; 00:10)
- https://kubernetes.io/docs/concepts/services-networking/ingress/ (March 30, 2025; 00:10)
- https://www.aseprite.org/ (March 30, 2025; 23:10)

SQUAVY

Browser Based Digital Audio Workstation

Project Documentation

Fabian Mild

Fabian Hummel

Konrad Simlinger

Alejandro Dario Tomeniuc

Jänner 2025 | squavy.daw@gmail.com

https://www.squavy.com | Spengergasse 20, 1050 Wien

March 2025

Digimagical GmbH

Legstadtgasse 4-6/C25, 3001 Mauerbach

1. Project Management

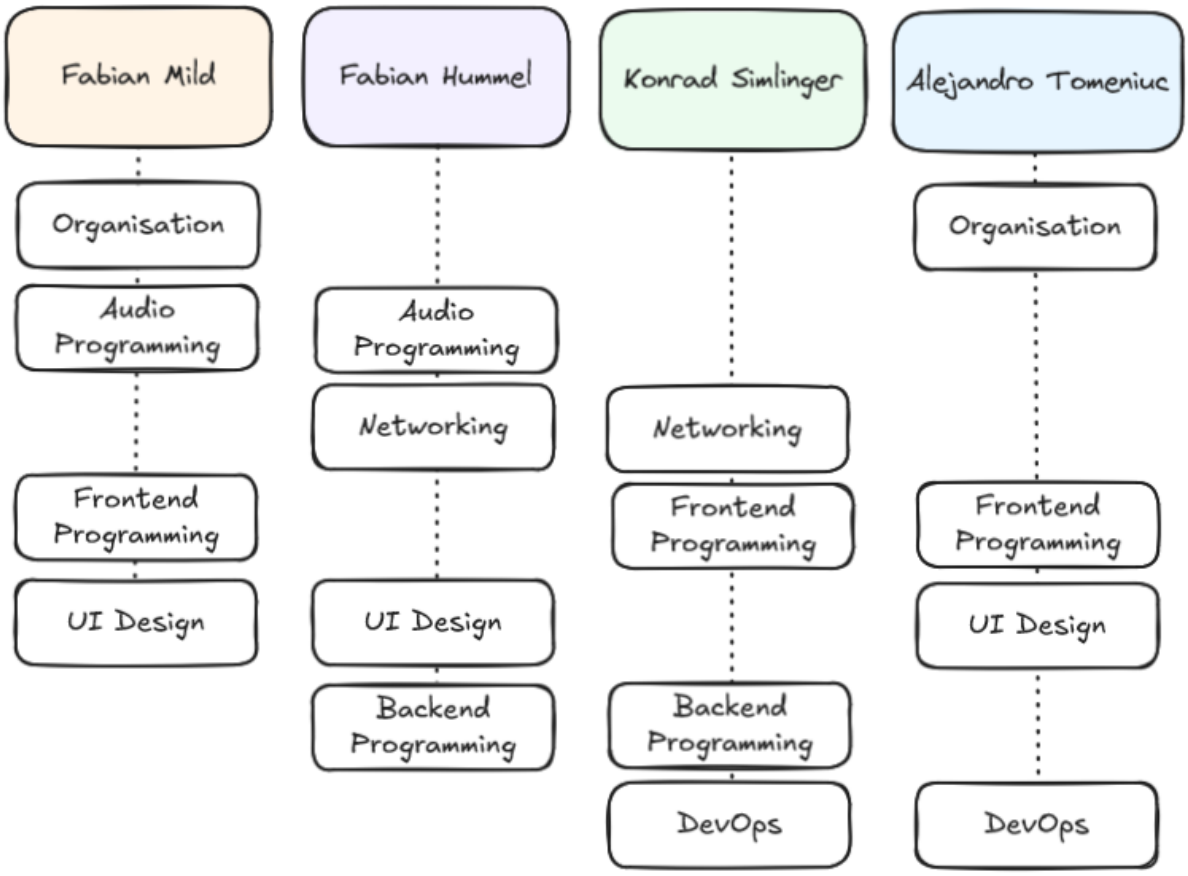
The Squavy project is managed using agile methods. This document provides details on the management and development practices used in the project. All information is accessible to the team members.

1.1. Definition of Done

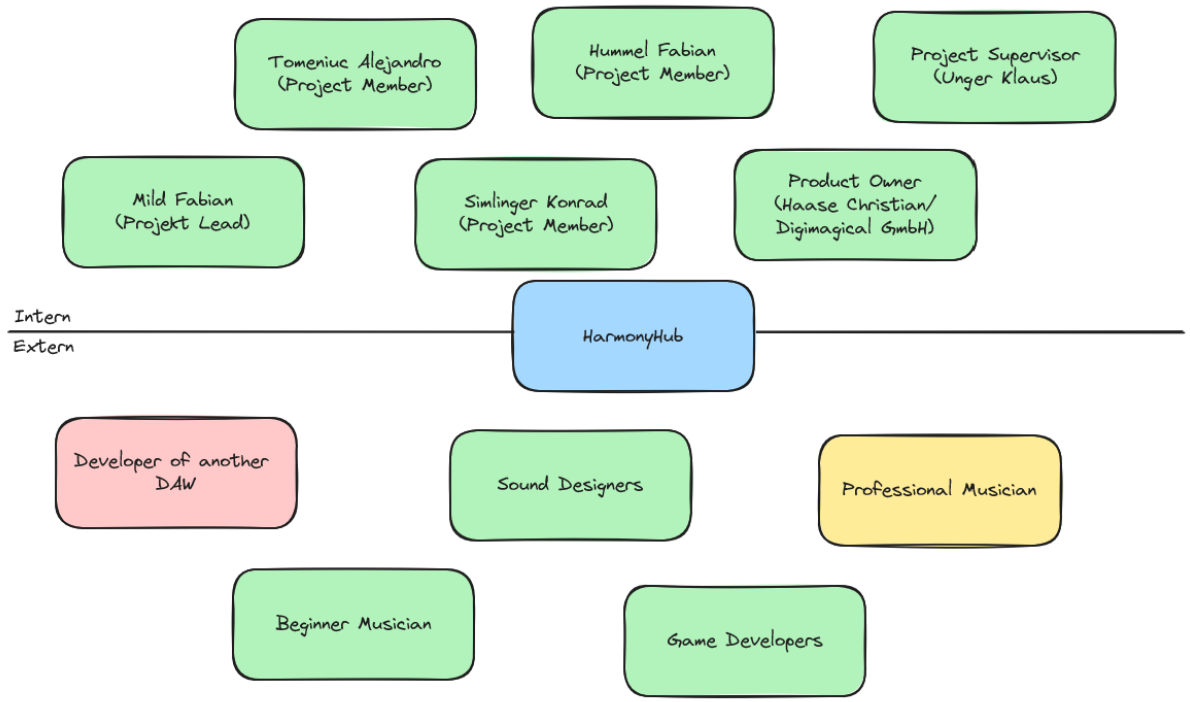
For Squavy, each one of the team members has his preferred features that he works on, and the development team is responsible for reviewing them and ensuring that they are functional.

One criterion is that the predefined test project can be loaded without errors. The project should play back correctly, with all expected audio and sequencing features functioning as intended. Team members will supervise these features and provide feedback. Additionally, a multiplayer session with more than two users should be created and processed correctly. For the features to work, the build of Squavy should be successful and changes to the architecture should also be documented for the project to be done.

1.2. Responsibilities



1.3. Environmental Analysis



Team

Mild Fabian (PL): He holds overall responsibility for the project, coordinating the team's activities to ensure progress and goal achievement.

Tomeniuc Alejandro, Hummel Fabian, and Simlinger Konrad (PTM): These team members contribute to the project's development by handling various tasks necessary to bring Squavy to life.

Company

The company, in this case **Digimagical GmbH**, has an important role represented by the **Product Owner (Haase Christian)**. The Product Owner represents the company's interests and ensures that Squavy meets business needs and incorporates market-relevant features.

Supervisor

Unger Klaus acts as our project supervisor and takes on the role of a teacher. He supports the project team throughout the development process, ensuring that the work quality remains high and the project goals stay in focus.

1.4. Stakeholder Analysis

Stakeholder	Relationship (Potential/Conflict)	Measures	Impact
Beginner Musician	A beginner wants to work with a DAW that is very easy to understand but doesn't lack important features!	Clear documentation and a welcoming interface are essential to get newcomers to like the product. The features should be intuitive, so they can be understood with ease.	 High
Professional Musician	Squavy isn't the best established DAW option. Sure, it is cheap, but money isn't the problem when choosing a tool for a professional musician.	To make this product more appealing to professional musicians, the networking features should be perfect. That way, Squavy can be used by pros to create small loops and previews with their partners quickly.	 Medium
Developer of another DAW	The developer of another DAW doesn't want Squavy to succeed. Therefore, misinformation may be spread.	To avoid any inconveniences like this, it is advised to be careful with planning and releasing information of the project.	 Low
Sound Designer	A sound designer wants to create an enormous variety of sounds and wants to easily understand the tool.	The modular synthesizer must include a variety of features, so that the audio signal can be modified to one's liking. Also, an easy-to-understand interface is essential for the success of the editor of the synthesizer.	 Medium
Game Developer	A game developer wants to create music for their game at an extremely fast pace.	The midi editor must be easy to understand and have a variety of shortcuts and quality of life features. Also, there should be a variety of instrument presets for them to choose from.	 High

1.5. Project Sprints

The "Project Sprints" process consists of structured meetings designed to maintain steady progress, address any issues, and align the team on upcoming tasks. Due to the team's preferences the **two-week-long sprints** start and end on **Fridays**. Each sprint includes:

- **Weekly Standup**: This is a 30-minute meeting where team members discuss their current progress, identify any blockers, and outline the next steps. The purpose is to ensure everyone is aligned and aware of any potential obstacles.

- **Sprint Planning**: During this meeting, which takes place alongside the weekly standup, the team sets clear goals for the sprint, prioritizes items from the backlog, and assigns tasks to each member. This ensures that all team members are clear on their responsibilities and that the sprint's objectives are well-defined.

- **Sprint Meeting**: At the end of each two-week-sprint, team members present a demo of their completed work, allowing others to give feedback and discuss potential improvements. This meeting also serves as a foundation for planning future tasks based on insights gained from the feedback.

During each meeting, a meeting protocol is written. The topics that need to be discussed during the project sprints are stated as a checklist:

Sprint Review:

- What have I done during the last sprint?
- What am I planning to do this week?
- Are there any blockers? What can I do to minimize the blockers?

Sprint Planning:

- Grooming of the backlog
  - Push important tasks up
  - Every team member pulls from the backlog to the doing list
- Considering how much work something is

Sprint Retrospective:

- Discussion:
  - What can be done better?
  - Status of the Key Performance Indicators (KPIs)
- Snapshots from progress

1.6. Sprint Parameters

- **Sprint Content:** Detailed list of completed, changed, added, or removed tasks of the projects kanban board.
- **Current state of blockers:** Are there things outside the project that are preventing further development, and can these blockers be counteracted?
- **Sprint duration:** Typically, sprints last two weeks.
- **Project velocity:** How fast the project is being developed. Categorized into three groups: Green, Yellow and Red.
- **Current state of team communication:** How well the team is interchanging information about new features, changes, and general discussion topics. Categorized into three groups: Green, Yellow and Red.

1.7. Kanban

Our kanban board is structured into 5 groups that help organize our tasks in a sensible way. Especially the blocked column helps focus on tasks that need to be resolved quickly to continue steady development:

- **Blocked**: When a task must be done before another task can start, the latter must be marked blocked and placed in “blocked” group.
- **Not Started**: Contains the backlog items of the current sprint. Everyone can assign themselves to the desired task whenever they want.
- **In Development**: This is the set of tasks that are currently being worked on. One or more assignees may take part in a task.
- **Review**: This column includes the tasks that have been completed but not yet accepted by the whole team. Only significant tasks/features that have a dedicated pull request on GitHub may be in “Review”.
- **Done**: The task has been fulfilled.

1.8. Deadlines

Milestone	Date
Post-Kickoff Presentation completed	04.11.2024
Prototype submitted	12.12.2024
Compliance Analysis completed	10.02.2025
Final Rollout completed	09.04.2025

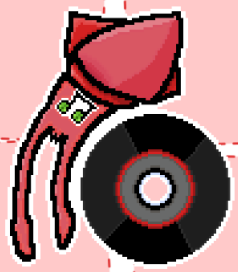
1.9. SWOT Analysis

S

1. Widespread knowledge across team members
2. Appealing visual design
3. Relatively gentle learning curve
4. Simple and easy operability
5. Extensive functions

W

1. Major project supported by a small team
2. Lack of publicity (Only few people know of the project)
3. Not many waysfor generating revenue
4. Little to no customer retention/loyalty
5. Difficult and complicated technologies



O

1. Growth of the mobile advertising market
2. Fast growth and distinction due to niche market
3. Easily accessible through the internet
4. Supporting the open-source community
5. Supporting general research in digital audio processing

T

1. Leading applications are led by big corporations
2. Relatively small market
3. High risk of failing in spite of lots of effort
4. Dependent on external open-source software
5. Changing regulations regarding data privacy

1.10. TOWS Matrix for Strategy Development

TOWS		External Analysis: Opportunities	External Analysis: Threats
Internal Analysis	Strengths	1. Using the appealing design, we increase growth and distinction of our application in the market. **(S2/O2)** 2. Supporting different open-source projects leveraging the knowledge of our team members. **(S1/O4)** 3. Implementation of new ways to process audio digitally. **(S5/O5)**	1. Developing extensive features increases the psychological pain when ultimately failing **(S5/T3)** 2. Because our product is easy for beginners, we grow the general market for music composition by inspiring new people. **(S4/T2)** 3. Higher expenditure for the adjustments regarding the regulations. **(S6/T5)**
	Weaknesses	1. Due to the growth of the mobile advertising market, it is an easy feat to generate a lot of publicity using social media etc. **(W2/O1)** 2. Increase customer loyalty and support by providing easy access to the application through the internet. This also affects possible ways of generating revenue. **(W3/W4/O3)** 3. Our team supports the open-source community wholly; therefore, we can absorb a lot of knowledge not spread otherwise that can be ultimately important **(W1/O4/O5)**	1. Increase publicity using different marketing techniques so that people choose our product instead of more popular alternatives. **(W4/T1)** 2. Drastically increase customer retention to maintain a supporting user base to achieve success and not fail. **(W2/W4/T3)**

1.11. Resource and Version Management Approach

The team saves most resources under the **documents** section of the Squavy **Microsoft-Teams** channel. This includes links to other software, time tracking and important documents.	
Flow graphs, architectures and similar are stored in the teams **Excalidraw** board.	
For general front-end design, the team uses **Figma** as their application of choice. This space is also used whenever quick mockups are needed in a meeting.	
For most things regarding project management, **Notion** is used. This includes the kanban board and meeting protocols.	
When it comes to quick communication and short-term resource storage (small images, ideas, etc.), **WhatsApp** is our tool of choice.	
Version management for **source code** is done with **Git** on **GitHub**. Here lies a total of 6 active repositories.	

All team members have full access to every resource of Squavy.

1.12. Collaboration Approach

We use an agile approach for collaboration in Squavy. This flexible and iterative method enables us to adapt quickly to changes and consistently deliver value. We have weekly meetings (Stand-ups), in which we discuss progress and blockers to make sure to stay on track and use a shared notion board as our kanban for project management.

Code Reviewing:

- GitHub with **pull request model** for feature development.
- **Peer Review** for large features, the pull request is reviewed by the entire team before merging to check for quality and functionality.

Communication:

We encourage open communication and the exchange of ideas, creating a supportive environment where feedback is constructive. This helps us create a positive and productive space where everyone feels comfortable sharing their thoughts and giving feedback to one another.

For our daily communication, we use **WhatsApp**. Furthermore, we track tasks with **Notion** and manage documentation through **Microsoft Teams**.

1.13. Ceremonies

Squavy's team thrives for productive time usage while having fun at developing and working with each other, which is why we've agreed on the following ceremonies to add more fun to the project:

1. **Talculator**: Whenever a meeting takes place, a calculator is used to determine the current speaker. This helps reduce interruption by speakers that are not currently supposed to present. However, the other members that do not hold the Talculator are allowed to ask questions or mention additional, small things.

2. **Demo Showcase**: Even in small meetings, the team inspects the newest sets of features that are being worked on. This is because we believe that the full picture can only be understood when everyone sees the project working or failing.

3. **Team Time:** Halfway through home meetings, we take a break to drink tea and eat jalapeno crisps in order to recharge before continuing project work.

1.14. Key Performance Indicators (KPIs)

1. **Velocity**: Development speed and how fast milestones are reached. We measure this by the completion of individual tasks.

2. **Team Communication**: Although hard to measure, this crucial indicator is a component of our team's effectiveness, and assessed every meeting by asking all team members if they believed the communication was sufficient.

3. **Shared Understanding**: The team's collective understanding of the project from a conceptual and technical point of view

4. **Quality Control**: This indicator describes the overall code quality of our product and is measured after completion of a task and once more at every sprint meeting.

5. **Stakeholder Communication**: Scheduled meetings with the partner are held as well as communication with the supervisor.

6. **Budgeting**: Difference between the actual cost and earned value.

1.15. Project Health Monitors

Area	Result
Shared Understanding: The team has a common understanding of why they're here, the problem/need, are convinced about the idea, confident they have what they need, and trust each other.	The team has issues understanding each other's problems and ideas sometimes. More team communication should solve this issue.
Runtime Integrity: After completing the development of a feature, the basic functionality of the project is positively tested.	Currently, the product contains lots of minor technical issues and some technical parts are being rewritten entirely.
Build Status: Confirm the build is successful and can be deployed to the public.	An older version of the project successfully compiles and could be deployed to the public.
Sprint Progress: Track sprint goals and progress using the shared Notion board.	The tasks are being worked on slowly but steadily.
KPIs Monitoring: Regularly track KPIs like velocity, team communication, shared understanding, quality control, stakeholder communication, and budgeting	KPIs are tracked every week during the stand-up meeting.
Milestone Tracking: Ensure milestones (Prototype, Compliance Analysis, etc.) are met according to the timeline.	Milestones are met according to the timeline.
Document Health: Ensures documents are always held up to date if something technical or project management-wise changes.	All documents are held up to date including technical architectures.

1.16. Retrospective Summary

Project status in November 2024:

The collaboration went perfectly. Feature development was well coordinated by the team, and nothing was done redundantly. Tasks were completed carefully and at a moderate speed. There were only very short phases where progress came to a halt.

State of the project in February 2025:

The projects' developing velocity is relatively high on average. There were weeks with sparse development time, but right now the velocity is higher than ever. Most of the current work focusses on library rewrites behind the scenes and progress moves steadily. In the weeks to come, it is expected that the remaining tasks that are in development will be completed, and the team can follow up with the next pool of tasks.

A hard thing to measure is the shared understanding of Squavy's code. There are many repositories that make up Squavy and the codebase can be hard to navigate at times, especially if one has not worked on some specific part of the project before. This may be the worst of the teams used metrics, as it is hard to keep up with every change as an individual, but there have not been any problems related to this metric so far.

Another important aspect of the project is the research expenditure. As Squavy's dependencies such as SMSE (=Squavy Modular Synthesizer Engine) and Squidge (=Squavy Bridge) are incredibly complex, the expense regarding research is higher than anticipated. Neither the less, progress is happening steadily, and the team doesn't plan any stops soon.

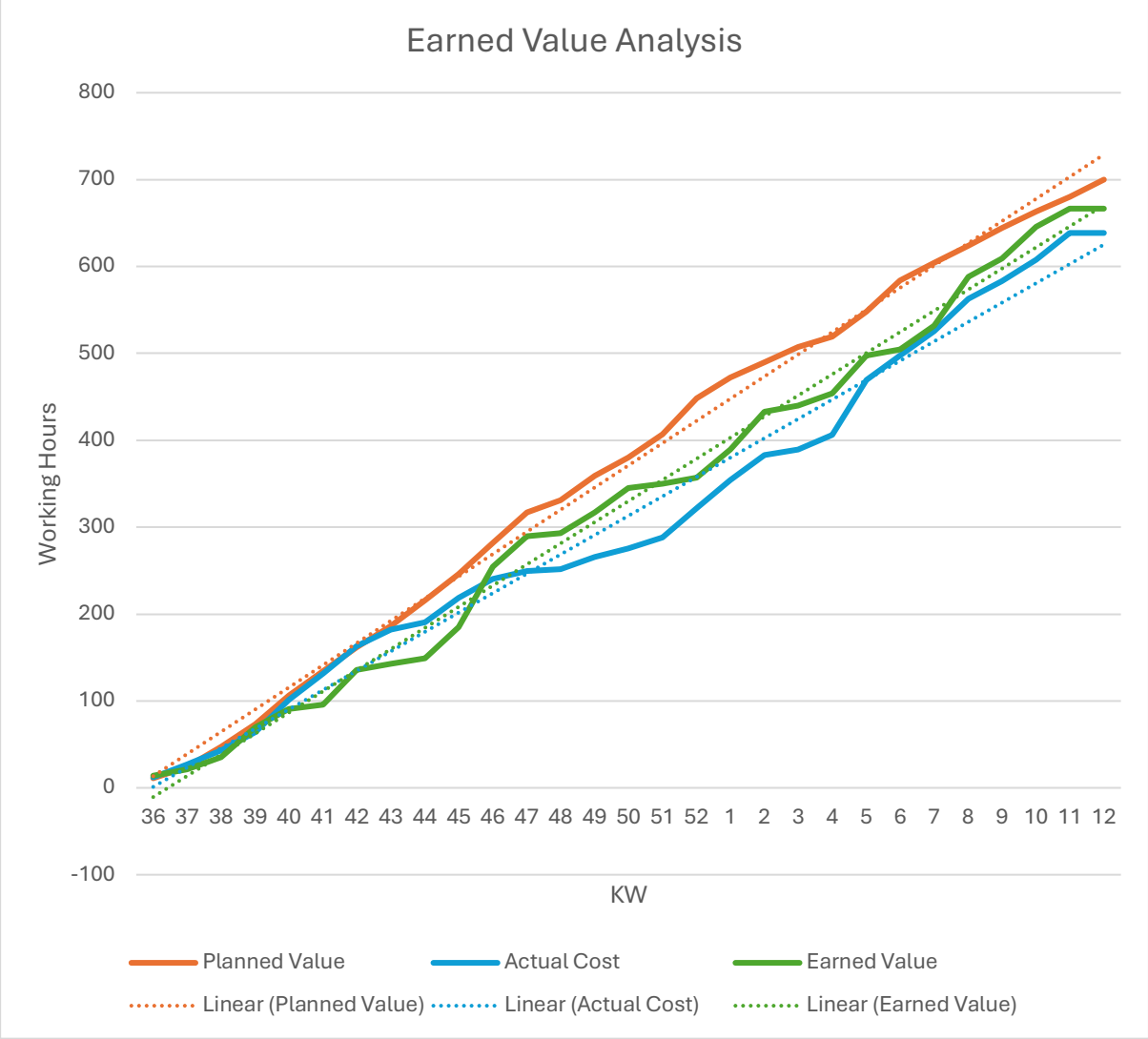
Final retrospective in March 2025:

The project has made great progress, with most planned tasks completed. Overall, development speed stayed high, even with some slower periods. The team successfully improved frontend and backend systems, creating a solid base for the future.

Navigating Squavy's complex codebase remains a challenge, but teamwork and knowledge-sharing have kept things running smoothly. Especially the documentation helps to make this easier. Furthermore, research costs were higher than expected, especially for Squavy's dependencies like SMSE and Squidge. However, the investment has led to strong improvements, and the team plans to keep pushing forward.

Overall, Squavy is in a great position for the next phase of development and deployment.

1.17. Earned Value Analysis



Whereby our cost performance index (CPI) is at 104,39%. It shows a comparison between the earned value and the actual cost. As it is above 100%, it means that we spent less time than what we have gained in terms of the project's completion.

2. Compliance Analysis

2.1. Records of Processing Activities

Name and Contact Details (1a):

- **Controller:** Squavy
- **Contact Details:** squavy.daw@gmail.com / Spengergasse 20, 1050 Wien

Activity ID	**Purpose of Processing** (1b)	**Categories of Data Subjects** (1c)	**Categories of Personal Data** (1c)	**Envisaged Time Limits for Erasure of Data** (1f)	**Categories of Recipients of Personal Data** (1d)
1	Facilitate browser-based musical editing and real-time online collaboration.	Anonymous users	Temporary collaboration data: song info, mouse positions, username	Volatile; deleted post-session.	None
2	Enforce IP-based bans for technical functionality.	Anonymous users	IP-Addresses of banned users.	Indefinite or until ban is lifted.	None
3	Process IP addresses when users visit the website to enable access to web hosting services	Anonymous users	IP-Addresses	Volatile; stored temporarily during the session.	None
4	Track server statistics for operational purposes (e.g., user count, server load).	Not applicable	Non-personal data	Retained for administrative purposes.	None

Technical and Organizational Security Measures:

- End-to-end encryption for all app-state-related collaboration data, ensuring no backend visibility for administrators on in case of cybercriminal attacks.
- Secure storage and access control mechanisms for server statistics.
- Network and server security measures in compliance with GDPR Art. 32.

2.2. Privacy Policy

1. Information We Collect

1.1. Automatically Collected Information

When you access our website or use our collaboration features, we collect:

- **IP Address**: Used to facilitate access to our web hosting services and enforce technical bans (if necessary).

1.2. Collaboration Data

When using online collaboration, we temporarily process:

- App state data (e.g., song information, mouse positions, usernames).
  This data is end-to-end encrypted and only transmitted between collaborators.

1.3. Server Statistics

For operational oversight, we collect and store non-personal data such as:

- User Count.
- Server load.

---

2. How We Use Your Information

We use the information we collect for the following purposes:

- To enable browser-based editing and real-time collaboration.
- To ensure secure and stable operation of our services.
- To enforce IP-based bans where necessary for security.
- To monitor server performance and optimize services.

---

3. Data Retention

- Usernames: Retained for as long as the collaborative session.
- IP addresses (bans): Retained indefinitely or until the ban is lifted.
- Server statistics: Retained for administrative purposes.

4. Data Sharing

We do not share your personal data with third parties.

---

5. Data Transfers

All data is processed and stored within the European Union. Users outside the EU can connect to our servers, but no personal data is transferred to third countries.

---

6. Security Measures

We implement robust technical and organizational measures, including:

- End-to-end encryption for collaboration data.
- Secure storage of server statistics and IP addresses.
- Access controls for administrative purposes.

---

7. Your Rights

As a user, you have the following rights under GDPR:

- Right to access your data.
- Right to request rectification of your data.
- Right to request deletion of your data (where applicable).
- Right to restriction of processing.
- Right to data portability, where technically feasible.
- Right to object to processing (e.g., IP-based bans).
- Right to lodge a complaint with a supervisory authority.

---

8. Contact Information

For any questions or to exercise your rights, please contact:

Squavy
squavy.daw@gmail.com
Spengergasse 20, 1050 Wien

2.3. Terms of Agreement

Terms of Agreement

Last updated: 12.01.2025

Welcome to Squavy (**“we”**, **“us”**, or **“our”**). By accessing or using our browser-based Digital Audio Workstation (DAW) (**“the Service”**), you agree to be bound by these Terms of Agreement (**“Terms”**). Please read them carefully before using the Service.

---

1. Use of the Service

1.1. **Eligibility**

- You must be at least 14 years old or of legal age in your jurisdiction to use the Service.

1.2. **Permitted Use**

- You may use the Service solely for lawful purposes and in compliance with these Terms.
- You are responsible for ensuring the security of your session and preventing unauthorized access.

1.3. **Prohibited Use**

You agree not to:

- Use the Service for any illegal or harmful activities.
- Attempt to interfere with or disrupt the Service, including through hacking or unauthorized data access.
- Abuse or misuse collaborative features to harm other users.

---

2. Collaboration and Data Handling

2.1. **Collaboration Features**

- Collaboration allows users to share app-state data (e.g., song edits, mouse positions) with other participants in real-time.
- Collaboration data is transmitted securely using end-to-end encryption.

2.2. **Data Responsibility**

- You retain ownership and responsibility for any data you create using the Service. We do not store collaboration data long-term; it is deleted after each session.
- You are responsible for ensuring any shared content complies with copyright laws and does not infringe on the rights of others.

---

3. Intellectual Property

3.1. **Service Ownership**

- All intellectual property rights in the Service, including the software, design, and branding, remain the exclusive property of Squavy.

3.2. **User Content**

- You retain ownership of the musical content you create.

---

4. Limitation of Liability

4.1. We are not liable for:

- Loss of data or content due to user error or technical issues.
- Any damage arising from unauthorized use of the Service.
- Compatibility or performance issues on your browser or device.

4.2. The Service is provided “as is,” and we make no guarantees regarding uptime, functionality, or the absence of errors.

---

5. Account and IP Address Restrictions

5.1. **Guest Usernames**

- Collaboration is conducted using temporary guest usernames. You are responsible for choosing a username that complies with these Terms.

5.2. **IP-Based Bans**

- We may issue bans based on IP addresses to enforce security and integrity.
- Bans are indefinite unless automatically lifted or following a review process (see Privacy Policy for details).

6. Modifications to the Service

- We reserve the right to modify, suspend, or terminate the Service at any time. Users will be notified accordingly.

7. Governing Law

- These Terms shall be governed by and construed in accordance with the laws of the European Union.

8. Contact Information

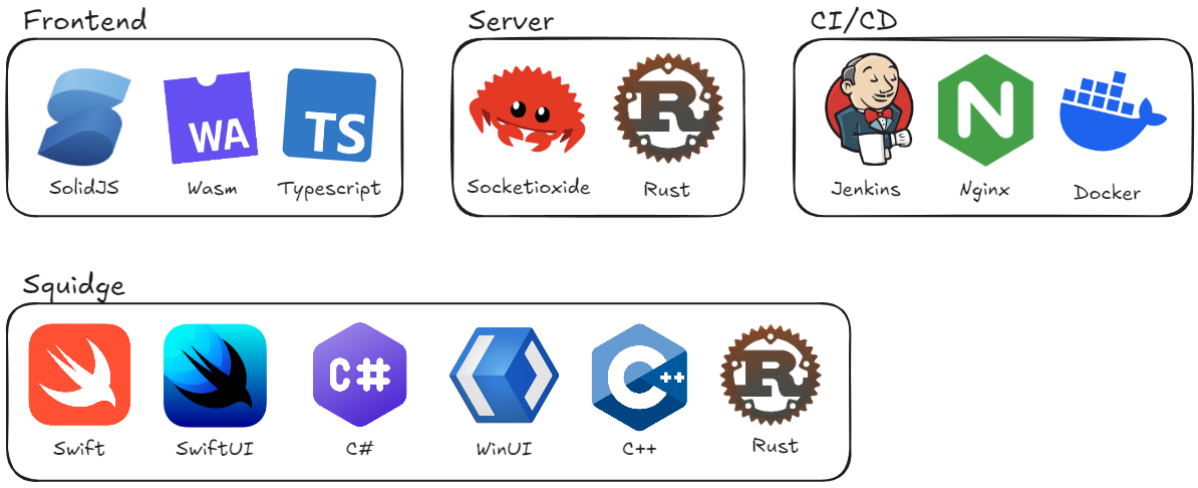
If you have questions or concerns about these Terms, please contact us at:

Squavy
squavy.daw@gmail.com
Spengergasse 20, 1050 Wien

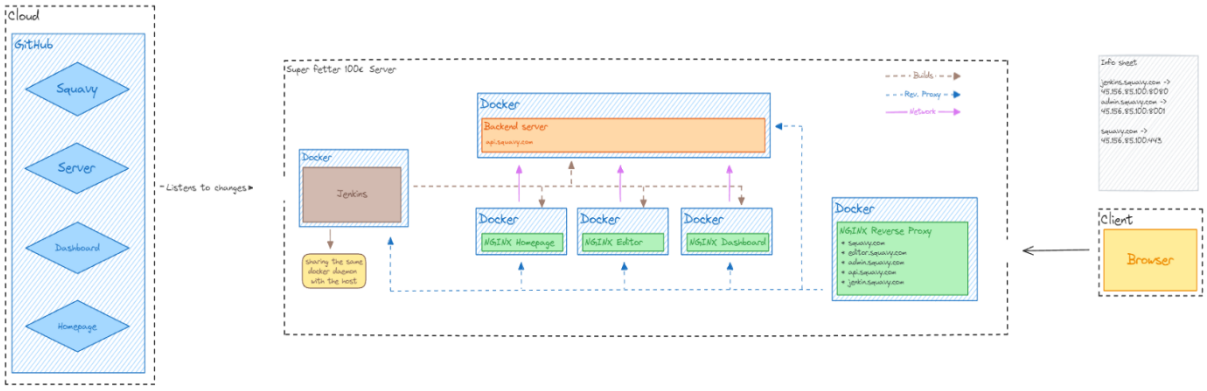
By using the Service, you acknowledge that you have read, understood, and agreed to these Terms.

3. System Architecture

3.1. Tech Stack



3.2. CI/CD Architecture

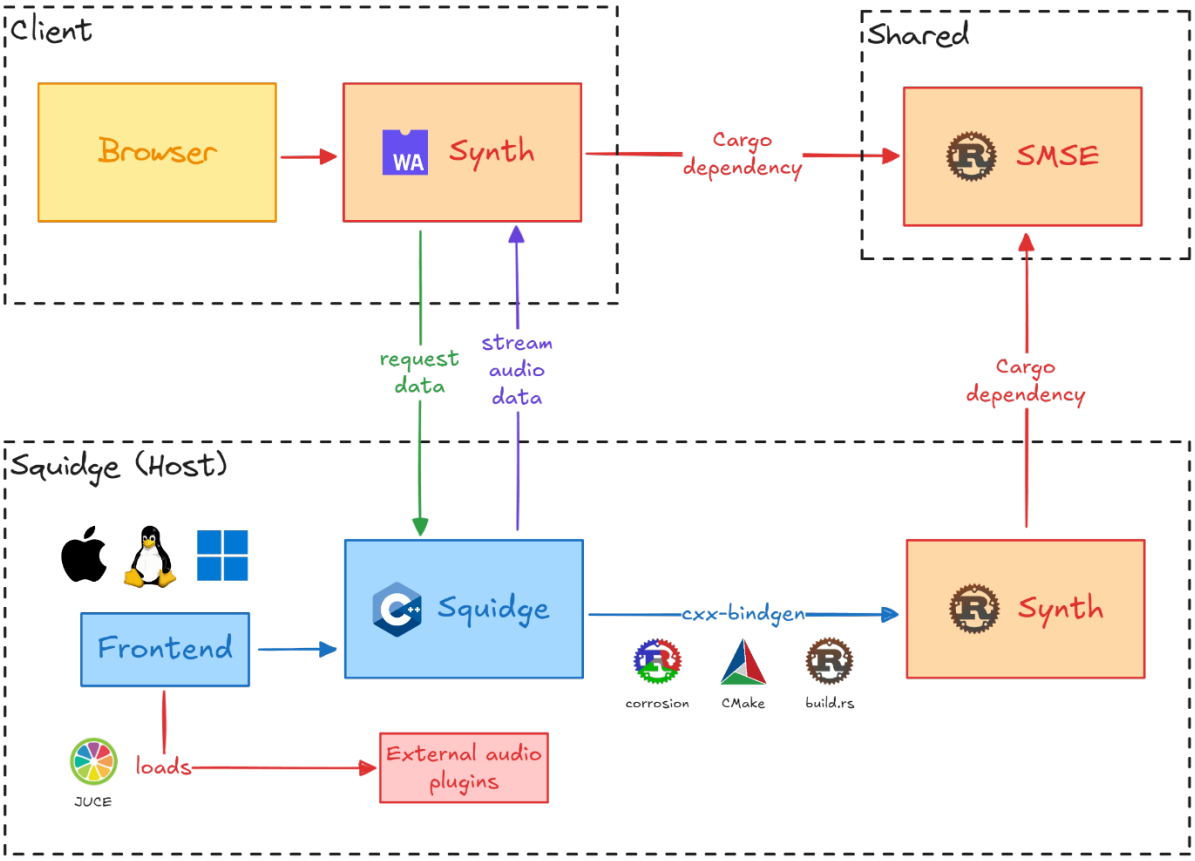


- On the lefthand side, there are all GitHub repositories that are automatically built by our CI/CD pipeline.

- The middle part describes the complete set of components needed to build, deploy, and host the Squavy products, whereas on the left there is Jenkins, which listens to changes to the repositories and builds them. Then, docker containers are launched to run relevant backend services like the collaboration server or static sites like Squavy, its homepage and dashboard.

- On the right, all reverse proxies are stated in a separate NGINX component which is the outer interface to communicate with the browser (far right of the architecture diagram).

3.3. Synthesizer Architecture



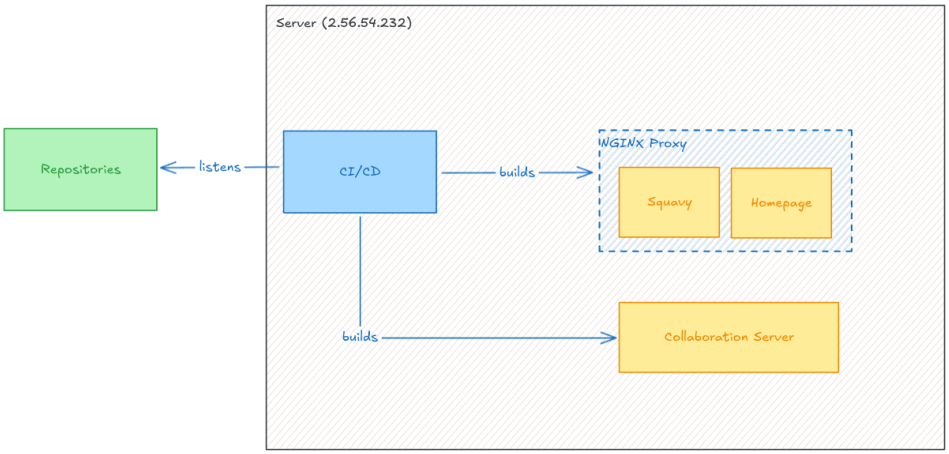
- **Top Left:** The client application that runs within the browser. Is uses the synthesizer implementation written in Rust and pairs it with a small JavaScript wrapper that leverages Web Audio Worklets to make the sound audible. The code is compiled to Web Assembly for best performance.

- **Bottom:** The optional synthesizer runtime for increased performance and external plugin support using JUCE. As long as Squidge runs, Squavy tries to move the entire synthesizer stack over to Squidge fully automatically with no downtime. Additionally, the small Rust crate on the right side contains synthesizer extension to allow communication between SMSE and externally loaded plugins. The communication between Squidge and the browser happens via Web Sockets on the localhost.

- **Top Right:** The shared synthesizer code written purely in Rust. This crate contains everything that is needed to process a modular synthesizer and can be cross compiled to either Web Assembly or natively with C++ interoperability.

3.4. Rollout Procedure

All the necessary repositories are cloned on the server and built for a docker environment via our CI/CD Pipeline, which also deploys all the containers correctly configured to interoperate with another. All of the built and deployed software can be accessed via the browser by accessing our domain that is configured to route to our reverse proxy.



The product will be rolled out as the first stable version on 09.04.2025. By then, all major bugs of the frontend are going to be resolved and final features implemented, including a new, robust synthesizer and improved user experience.

The order of deployment is:

- Start a collaboration server instance on rented hardware and make sure connectivity is stable by internally testing it with the unreleased product.
- Publicize the main product on our configured domain www.squavy.com together with a work-in-progress version of the native plugin bridge “Squidge”.
- Inform interested parties about the publication on our social media presence.
- Actively monitor the server loads for the first few hours of publication and quickly fix potential issues to ensure the best experience to the users.

3.5. Report on deployed Features

- **Homepage:** The Homepage is the starting point of the platform, providing an intuitive overview of all key features. Users can navigate easily, access recent projects, and get important updates immediately.

- **Collaboration Server**: The Collaboration Server enables real-time collaboration by allowing users to share projects, edit projects together, and manage permissions efficiently.

- **In-House Synthesizer**: The Synthesizer is a built-in tool for generating and modifying synthetic audio or data. It supports creative and technical applications, seamlessly integrating with other platform features.

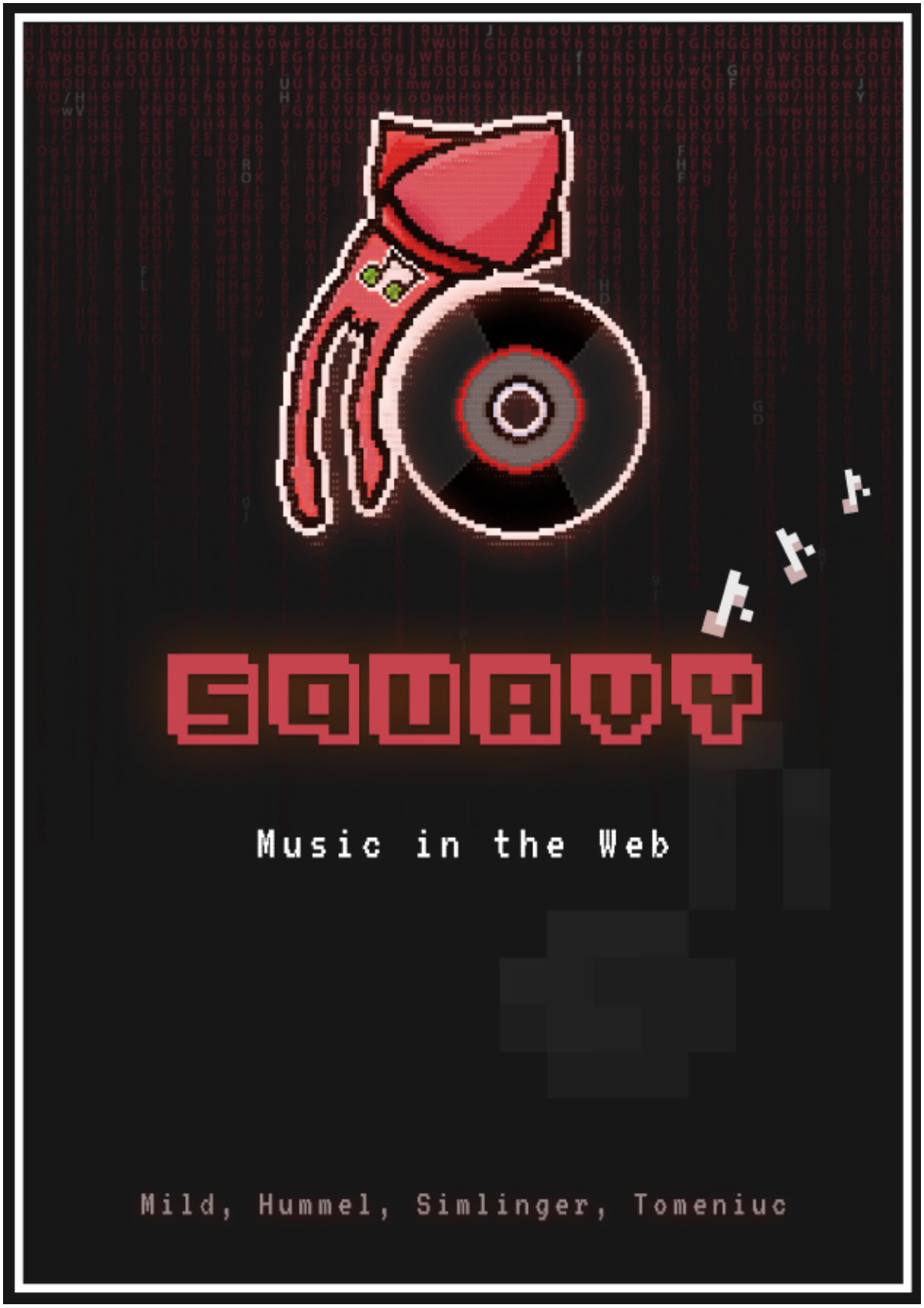
- **Online Editor**: The Online Editor is a browser-based tool that enables users to create and edit projects directly online. It offers a user-friendly interface with essential features for seamless content creation and modification, eliminating the need for additional software.

- **Alpha Version Squidge**: A major feature that allows pretesting usage of external audio plugins. As a complex and evolving system, this version allows users to explore its core functionalities and provide feedback, helping the team refine and improve the system before its full release.

4. Organisation

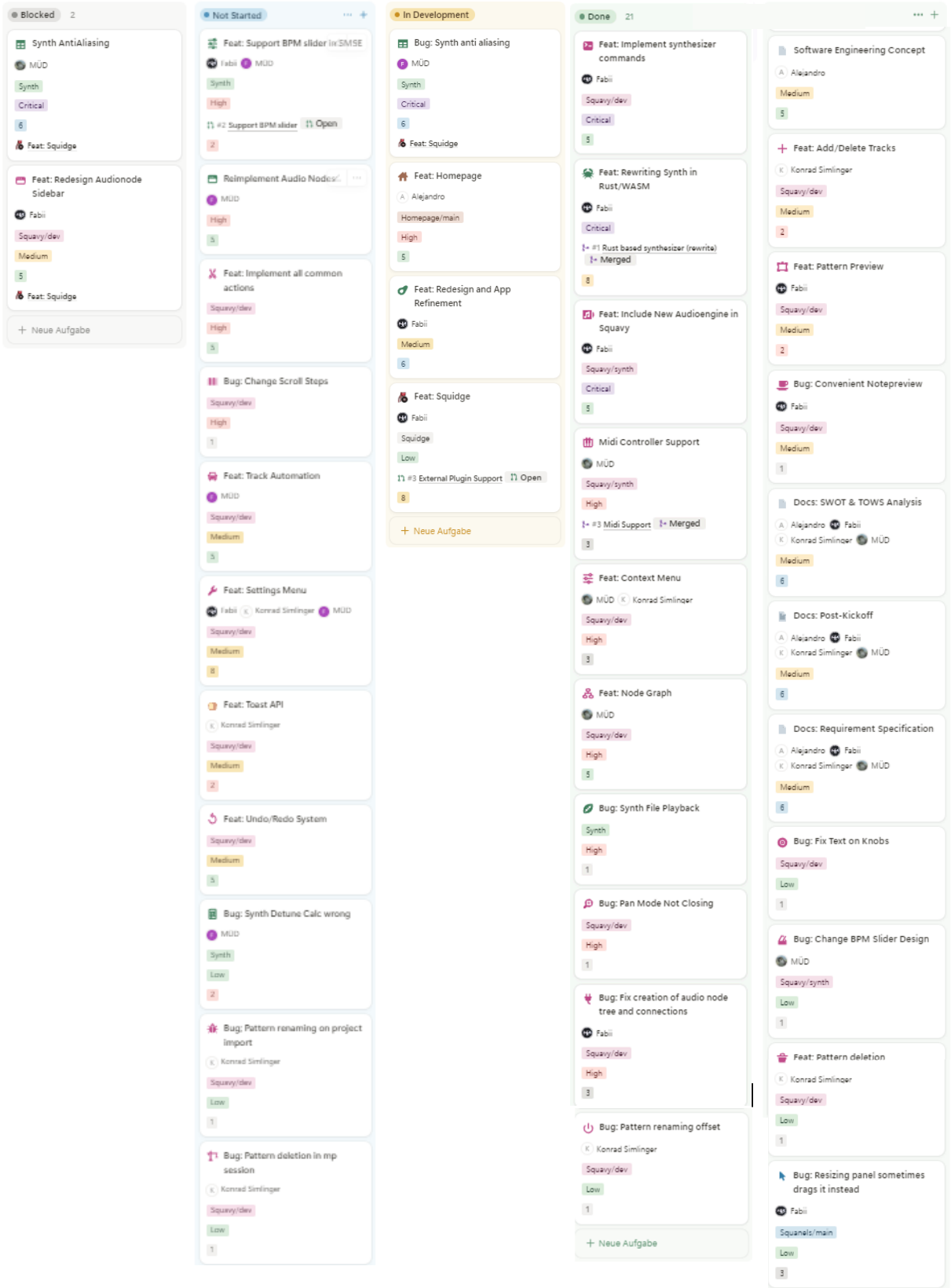
4.1. Design Mockups

The first ever design concept of Squavy's main window



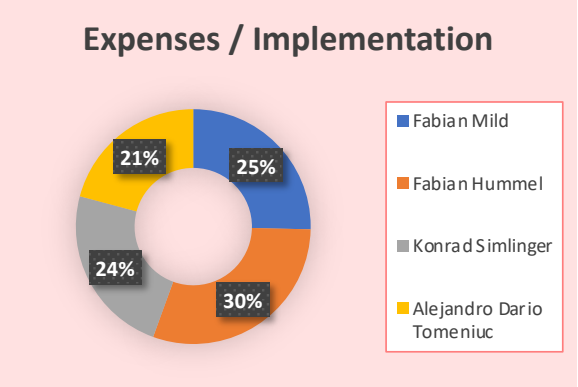
A Poster of Squavy

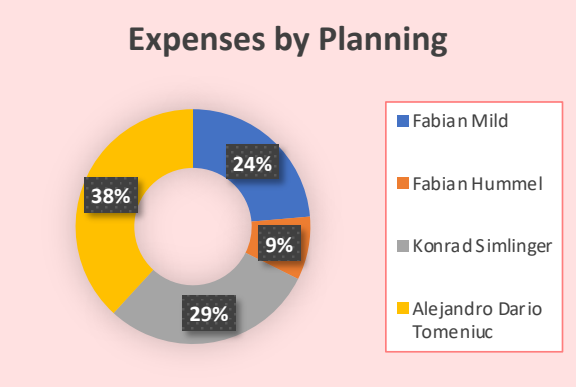
4.2. Kanban

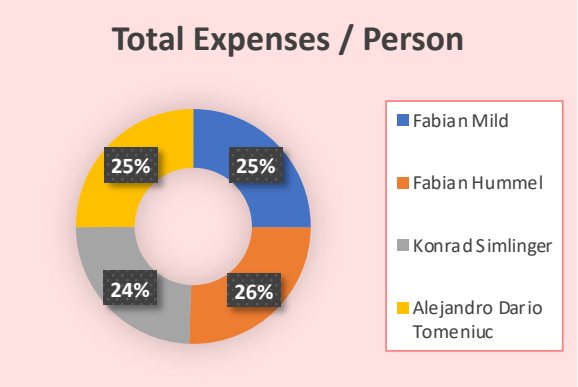

Blocked 2
Synth AntiAliasing
MÜD
Synth
Critical
6
Feat: Squidge
Feat: Redesign Audionode Sidebar
Fabii
Squavy/dev
Medium
5
Feat: Squidge
Neue Aufgabe
Not Started
Feat: Support BPM slider in SMSE
Fabii MÜD
Synth
High
#2 Support BPM slider Open
2
Reimplement Audio Nodes
MÜD
High
5
Feat: Implement all common actions
Squavy/dev
High
5
Bug: Change Scroll Steps
Squavy/dev
High
1
Feat: Track Automation
MÜD
Squavy/dev
Medium
5
Feat: Settings Menu
Fabii Konrad Simlinger MÜD
Squavy/dev
Medium
8
Feat: Toast API
Konrad Simlinger
Squavy/dev
Medium
2
Feat: Undo/Redo System
Squavy/dev
Medium
5
Bug: Synth Detune Calc wrong
MÜD
Synth
Low
2
Bug: Pattern renaming on project import
Konrad Simlinger
Squavy/dev
Low
1
Bug: Pattern deletion in mp session
Konrad Simlinger
Squavy/dev
Low
1
In Development
Bug: Synth anti aliasing
MÜD
Synth
Critical
6
Feat: Squidge
Feat: Homepage
Alejandro
Homepage/main
High
5
Feat: Redesign and App Refinement
Fabii
Medium
6
Feat: Squidge
Fabii
Squidge
Low
#3 External Plugin Support Open
8
Neue Aufgabe
Done 21
Feat: Implement synthesizer commands
Fabii
Squavy/dev
Critical
5
Feat: Rewriting Synth in Rust/WASM
Fabii
Critical
#1 Rust based synthesizer (rewrite) Merged
8
Feat: Include New Audioengine in Squavy
Fabii
Squavy/synth
Critical
5
Midi Controller Support
MÜD
Squavy/synth
High
#3 Midi Support Merged
3
Feat: Context Menu
MÜD Konrad Simlinger
Squavy/dev
High
3
Feat: Node Graph
MÜD
Squavy/dev
High
5
Bug: Synth File Playback
Synth
High
1
Bug: Pan Mode Not Closing
Squavy/dev
High
1
Bug: Fix creation of audio node tree and connections
Fabii
Squavy/dev
High
3
Bug: Pattern renaming offset
Konrad Simlinger
Squavy/dev
Low
1
Neue Aufgabe
Software Engineering Concept
Alejandro
Medium
5
Feat: Add/Delete Tracks
Konrad Simlinger
Squavy/dev
Medium
2
Feat: Pattern Preview
Fabii
Squavy/dev
Medium
2
Bug: Convenient Notepreview
Fabii
Squavy/dev
Medium
1
Docs: SWOT & TOWS Analysis
Alejandro Fabii
Konrad Simlinger MÜD
Medium
6
Docs: Post-Kickoff
Alejandro Fabii
Konrad Simlinger MÜD
Medium
6
Docs: Requirement Specification
Alejandro Fabii
Konrad Simlinger MÜD
Medium
6
Bug: Fix Text on Knobs
Squavy/dev
Low
1
Bug: Change BPM Slider Design
MÜD
Squavy/synth
Low
1
Feat: Pattern deletion
Konrad Simlinger
Squavy/dev
Low
1
Bug: Resizing panel sometimes drags it instead
Fabii
Squanels/main
Low
3

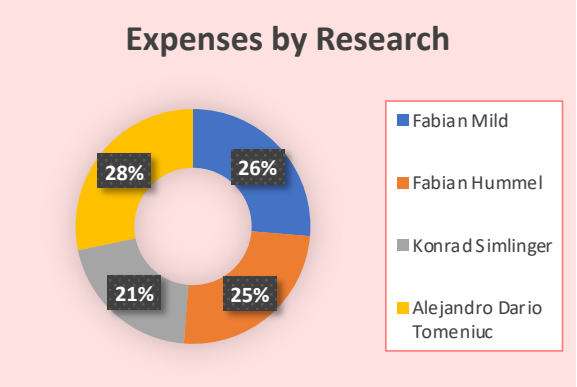
4.3. Time Records

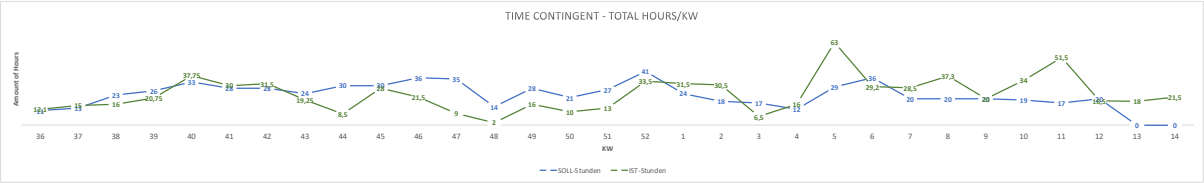
PKZ	Personen	Implementation	Planning	Research	Total/Person
MIL	Fabian Mild	123,6	32,5	23,75	**179,9h**
HUM	Fabian Hummel	147	12	22,5	**181,5h**
SIM	Konrad Simlinger	115	40,5	18,5	**174,0h**
TOM	Alejandro Dario Tomeniuc	101,5	52,5	25,5	**179,5h**
	Total/Kategorie	**487,1h**	**137,5h**	**90,3h**	**714,9h**











Fabian Hummel:

Date	Start	Duration	Work package	Description	Kategorie	Repository	KW
04.09.24	16:00:00	1	Software	added new smse to squavy	Implementation	Squavy	36
05.09.24	23:00:00	2	Software	added maximizing and restoring panels	Implementation	Squavy	36
07.09.24	12:00:00	3	Software	added overflow menu to panel toolbars	Implementation	Squavy	36
07.09.24	20:00:00	2	Software	added note preview	Implementation	Squavy	36
09.09.24	23:00:00	0,5	Software	added playback controls	Implementation	Synth	37
10.09.24	18:00:00	1,5	Software	added song playback	Implementation	Squavy	37
10.09.24	11:00:00	1	Software	added synthesizer command queueing	Implementation	Synth	37
10.09.24	19:00:00	0,5	Software	switched to floating point time calculation (bad decision)	Implementation	Synth	37
12.09.24	20:00:00	2	Software	added modal provider	Implementation	Squavy	37
14.09.24	15:00:00	3	Software	added aseprite importing	Implementation	Squavy	37
15.09.24	19:00:00	2	Software	added connection editing	Implementation	Squavy	38
17.09.24	17:00:00	0,5	Software	made audio nodes movable	Implementation	Squavy	38
18.09.24	12:00:00	2	Software	several smaller fixes and additions	Implementation	Squavy	38
19.09.24	21:00:00	2	Software	added synthesizer synchronization	Implementation	Squavy	38
19.09.24	23:00:00	0,5	Software	fixed critical multiplayer synchronisation bug	Implementation	Squavy	38
21.09.24	15:00:00	1,5	Software	abstracted visual audio nodes	Implementation	Squavy	38
22.09.24	23:00:00	2	Software	added buffers to audio nodes	Implementation	Synth	39
22.09.24	23:00:00	2	Software	added visual audio graphs	Implementation	Squavy	39
27.09.24	18:00:00	0,5	Software	fixed a critical bug with envelopes	Implementation	Synth	39
27.09.24	14:00:00	2	Software	added remaining synthesizer synchronizations	Implementation	Squavy	39
01.10.24	10:00:00	1	Software	modularized some components	Implementation	Squavy	40
01.10.24	09:00:00	1	Software	fixed minor issues with the synthesizer	Implementation	Synth	40
02.10.24	18:00:00	1	Diploma Thesis	imported and rephrased texts from ABA portal	Research	Server	40
02.10.24	23:00:00	2	Diploma Thesis	wrote early stages and chosen architecture	Research	Server	40
03.10.24	17:00:00	3	Diploma Thesis	wrote internal change detection	Research	Server	40
03.10.24	22:00:00	2,5	Diploma Thesis	wrote end to end encryption	Research	Server	40
04.10.24	10:00:00	0,5	Software	fixed panic with dead note duration on envelopes	Implementation	Synth	40
04.10.24	19:00:00	1	Software	fixed floating point precision error	Implementation	Synth	40
04.10.24	13:00:00	2	Diploma Thesis	wrote comparing communication protocols	Research	Server	40
04.10.24	22:00:00	1	Software	fixed convenient note preview bug	Implementation	Squavy	40
04.10.24	23:00:00	0,5	Software	fixed resizing panel sometimes drags it instead	Implementation	Squanels	40
06.10.24	11:00:00	1	Software	added sprite animations to the noise node	Implementation	Squavy	41
06.10.24	22:00:00	2	Software	added dynamic adsr and lfo previews	Implementation	Synth	41
07.10.24	20:00:00	2	Software	worked on squidge	Implementation	Squidge	41
08.10.24	10:00:00	1	Project Management	jour fixe	Implementation		41
08.10.24	13:30:00	1	Software	set up Aseprite for Team Member	Implementation	Squavy	41
13.10.24	12:00:00	6	Software	integrated JUCE into Squidge	Implementation	Squidge	42
14.10.24	12:00:00	1	Software	researched VST plugins	Research	Squidge	42
15.10.24	18:00:00	5	Software	added plugin loading with JUCE and rearranged the frontend	Implementation	Squidge	42
17.10.24	21:00:00	3	Software	frontend progress	Implementation	Squidge	42
18.10.24	11:00:00	4	Software	integrated uWebSockets	Implementation	Squidge	42
22.10.24	18:00:00	4	Diploma Thesis	wrote comparing server architectures and other sections	Research		43
30.10.24	15:00:00	3	Software	added uWebSockets communication	Implementation	Squidge	44
03.11.24	20:00:00	3	Software	native plugin window wrapper for external plugins	Implementation	Squidge	45
06.11.24	23:00:00	2	Software	ported code from SMSE to Squavy	Implementation	Synth	45
13.11.24	10:00:00	2	Software	ported code from SMSE to Squidge	Implementation	Squidge	46
13.11.24	10:00:00	6	Software	made code architecture independent	Implementation	Synth	46
15.11.24	12:00:00	4	Software	integrated new Synthesizer API	Implementation	Squavy	46
18.12.24	18:00:00	6	Software	reworked Squidge to support multiple clients	Implementation	Squidge	51
27.12.24	12:00:00	3	Software	researched Rust crate configurations	Research	Synth	52
01.01.25	15:00:00	6	Software	created cxx_bindgen and cxx_bindgen-build for automated FFI generation between Rust and C++	Implementation	Synth	1
03.01.25	10:00:00	1	Software	found an issue within corrosion	Research	Squidge	1
03.01.25	13:00:00	1	Software	fixed the issue in a fork of corrosion	Implementation	Squidge	1
06.01.25	15:00:00	3	Software	researched modular synthesizers and fleshed out a new strategy for digital signal processing in Rust	Research	Synth	2
06.01.25	14:00:00	0,5	Software	fixed crate build order to fix 50% failure rate	Implementation	Squidge	2
07.01.25	18:00:00	4	Software	refactored the synthesizer to prepare for new iteration	Implementation	Synth	2
10.01.25	19:30:00	1	Project Management	created song samples for trailer tomorrow	Project Management		2
11.01.25	14:00:00	6	Project Management	trailer	Project Management		2
31.01.25	12:00:00	5	Project Management	businessplan	Project Management		5
05.02.25	12:00:00	7	Software	new synthesizer, most notably arpeggiator	Implementation	Synth	6
06.02.25	12:00:00	2	Software	aseprite setup with Alejandro	Implementation		6
14.02.25	12:00:00	12	Software	midi-editor rewrite	Implementation	Squavy	7
22.02.25	12:00:00	9	Software	synth serialization + deserialization	Implementation	Synth	8
03.03.25	12:00:00	2	Software	fixed synth issues	Implementation	Synth	10
09.03.25	12:00:00	2	Software	added node tree for sidebar	Implementation	Squavy	11
11.03.25	12:00:00	3	Software	added drop-zone system to frontend	Implementation	Squavy	11
14.03.25	12:00:00	4	Software	synth editor mobile support	Implementation	Squavy	11
16.03.25	16:00:00	4	Project Management	final handin	Implementation		12
16.03.25	18:00:00	3	Software	dockerfile (build aseprite)	Implementation	Squavy	12

Fabian Mild:

Date	Start	Duration	Work package	Description	Kategorie	Repository	KW
03.09.24	17:00:00	2	Software	Fixed ConnectionLines Positioning	Implementation	Squavy	36
07.09.24	23:00:00	2,1	Software	Created Context-Menu	Implementation	Squavy	36
10.09.24	18:30:00	1,5	Software	Added Current connection and the ability to delete connections	Implementation	Squavy	37
15.09.24	18:00:00	2	Software	Implemented bezierJs	Implementation	Squavy	38
19.09.24	19:00:00	1	Software	Added Connection Selection and Deselection	Implementation	Squavy	38
22.09.24	20:00:00	0,25	Project Management	Wrote the meeting protocol for todays meeting	Project Management		39
22.09.24	22:00:00	3	Software	Fixed Connection Selection and Deselection	Implementation	Squavy	39
24.09.24	19:30:00	3,5	Software	Synth Bug fixing	Implementation	Synth	39
26.09.24	18:30:00	2	Software	Modified Connection Selection	Implementation	Squavy	39
27.09.24	15:30:00	1	Software	Fixed Color Scheme	Implementation	Squavy	39
28.09.24	17:00:00	0,5	Software	Fixed issue so dragging a new connection doesnt select the old ones	Implementation	Squavy	39
01.10.24	09:55:00	0,25	Project Management	Wrote the meeting protocol for todays meeting	Project Management		40
02.10.24	17:30:00	3	Software	Fixed a linux specific issue regarding selection menus	Implementation	Squavy	40
02.10.24	21:00:00	3	Software	Reimplemented Noise generators	Implementation	Squavy	40
04.10.24	17:30:00	1	Software	Fixed a detune and noise generator issue	Implementation	Squavy	40
05.10.24	22:00:00	1,5	Software	Worked on noise generators	Implementation	Synth	40
06.10.24	00:00:00	2	Software	Finalized noise generators	Implementation	Squavy	41
03.10.24	23:00:00	2	Diploma Thesis	Initialized Diploma	Research		40
04.10.24	19:30:00	0,5	Project Management	Created a short Presentation	Project Management		40
05.10.24	17:00:00	4	Software	Frog Sprite, Node deletion, Project Name	Implementation	Squavy	40
07.10.24	20:00:00	1	Software	Node deletion	Implementation	Squavy	41
07.10.24	21:00:00	0,5	Project Management	Updated Notion Board	Project Management		41
08.10.24	10:00:00	1	Project Management	Jour Fixe	Project Management		41
09.10.24	19:00:00	3	Diploma Thesis	Research and writing of a Techstudy	Research		41
10.10.24	17:00:00	3,5	Software	Fixed some bugs regarding the previously implemented Node deletion	Implementation	Squavy	41
14.10.24	19:00:00	4	Software	Researched AudioNode implementations for Delay and Compressor	Research	Squavy	42
18.10.24	19:00:00	0,5	Software	Fixed a bug with the previously fixed Node deletion	Implementation	Squavy	42
19.10.24	21:00:00	1	Software	Further researched AudioNode implementations for Biquad Filter, as this should be the first extra audionode	Research	Squavy	42
20.10.24	10:25:00	5,25	Software	Fixing Frontend Bugs, Synth Bugs, and more Bugs	Implementation	Squavy	43
24.10.24	19:45:00	2	Diploma Thesis	Researched technologie usage over the market	Research		43
26.10.24	13:00:00	3	Software	Researched use of Web Midi Api and discussed implementation with Fabian Hummel	Research	Squavy	43
30.10.24	20:00:00	2,5	Software	Implemented basic Midi Support	Implementation	Squavy	44
03.11.24	09:45:00	3	Project Management	Wrote the requirements definition and project playbook	Project Management		45
09.11.24	08:30:00	1	Software	Fixed a bug with the midi support	Implementation	Squavy	45
10.11.24	10:00:00	1,25	Project Management	Research for Diploma Thesis	Research		46
12.11.24	21:00:00	0,25	Software	Fixed Midi Knob support	Implementation	Squavy	46
16.11.24	13:00:00	4	Software	Finalized Midi Controller Support	Implementation	Squavy	46
17.11.24	20:10:00	2	Software	Started reworking knobs	Implementation	Squavy	47
20.11.24	21:00:00	3	Software	Progress on the knob rework	Implementation	Squavy	47
26.11.24	20:15:00	1,5	Project Management	Research for Diploma Thesis	Research	Squavy	48
27.11.24	19:30:00	0,5	Software	Fixed a bug with the synthesizers sound generation	Implementation	Synth	48
02.12.24	21:00:00	2	Software	Adjusted Knob design	Implementation	Squavy	49
07.12.24	19:15:00	4	Software	Fixed a variety of bugs and changed the knob design again	Implementation	Squavy	49
27.12.25	13:00:00	2,5	Software	Fixed Knob Stepping and made them smoother overall	Implementation	Squavy	52
03.01.25	13:30:00	3	Software	I fixed the synthesizer aliasing problem. This. Was. Tough...	Research	Synth	1
03.01.25	16:00:00	5	Software	I fixed the synthesizer aliasing problem. This. Was. Tough...	Implementation	Synth	1
04.01.25	14:15:00	5,5	Software	Added additive synth for pulse and saw waveforms	Implementation	Synth	1
10.01.25	11:00:00	2	Software	Added an option to reset the knob to its default value	Implementation	Squavy	2
10.01.25	16:00:00	1,5	Software	Worked on a panel that shows the current knobs value	Implementation	Squavy	2
14.01.25	09:55:00	1	Project Management	Jour Fixe	Project Management		3
25.01.25	18:00:00	5	Project Management	Analysis and Businessplan	Project Management		4
26.01.25	18:00:00	6	Documentation	Businessplan	Project Management		5
27.01.25	19:00:00	4	Documentation	Businessplan and Projektbericht	Project Management		5
28.01.25	22:00:00	1	Documentation	Businessplan and Projektbericht	Project Management		5
29.01.25	22:00:00	2	Documentation	Projektbericht	Project Management		5
30.01.25	20:40:00	4	Documentation	Projektbericht	Project Management		5
31.01.25	17:00:00	1	Documentation	Changed Projektbericht	Project Management		5
02.02.25	22:00:00	3	Software	Working on Infopanel	Implementation	Squavy	6
03.02.25	23:00:00	2	Software	Progress on Infopanel: Now responsive	Implementation	Squavy	6
09.02.25	20:00:00	3	Software	better knob feel and cleaner looks	Implementation	Squavy	7
19.02.25	14:00:00	4	Software	tried out different solutions (libraries) for displaying the infopanel content	Implementation	Squavy	8
25.02.25	19:00:00	3	Software	experimented with oversampling in Rust	Implementation	Synth	9
28.02.25	21:00:00	2	Software	working on oversampling. Sounds bad =(	Implementation	Synth	9
02.03.25	18:00:00	1,5	Software	Setup of the new Squidge and Frontend with the help of FabianHummel	Implementation	Squavy	10
04.03.25	16:00:00	3	Software	Researched Oversampling implementations and Infinite Impulse Response Filters	Research	Synth	10
05.03.25	20:30:00	2	Software	Got a oversampling demo working. NiceNice	Implementation	Synth	10
08.03.25	18:00:00	3	Software	Research and implementation ideas of ADAA-IIR. HOW DOES THIS WORK?!!?	Implementation	Synth	10
12.03.25	17:00:00	5	Software	Seems like I finally got it working. NiceNice.	Implementation	Synth	11
14.03.25	15:00:00	7	Software	Worked on the Synth-Editor Frontend. Finally looking pixel perfect.	Implementation	Squavy	11
15.03.25	17:30:00	3	Project Management	Meeting	Project Management		11
28.03.25	18:00:00	2	Software	Implemented the iir-filter into the SMSE rust codebase	Implementation	Synth	13
02.04.25	17:25:00	3	Software	Started with implementing Wavetable synthesis for SMSE	Implementation	Synth	14
03.04.25	20:14:00	4	Software	Progress on SMSE (still thinking about how the wavetables should be generated)	Implementation	Synth	14

Konrad Simlinger:

Date	Start	Duration	Work package	Description	Kategorie	Repository	KW
18.09.24	12:00:00	1	Software	Various bug fixes	Implementation	Squavy	38
17.09.24	18:00:00	1,5	Software	Context Menu	Implementation	Squavy	38
15.09.24	12:00:00	2	Software	Renaming fix	Implementation	Squavy	38
13.09.24	13:00:00	1	Software	Bug fixing	Implementation	Squavy	37
10.09.24	09:00:00	4	Software	ArgoCD, Git	Implementation	Squavy	37
04.10.24	09:00:00	1	Software	Synthesizer bug	Implementation	*Synth*	40
04.10.24	15:00:00	2	Software	More bug fixes	Project Management	Squavy	40
05.10.24	13:30:00	2	Diploma Thesis	Set up my part of the document	Research		40
08.10.24	09:00:00	1	Project Management	Jour Fixe	Project Management		41
10.10.24	17:00:00	3	Diploma Thesis	Researched some ways to approach thesis	Research		41
11.10.24	15:30:00	1	Software	Gathered ideas for the Toast API	Implementation	Squavy	41
15.10.24	09:00:00	1	Project Management	Jour Fixe	Project Management		42
16.10.24	16:40:00	3	Diploma Thesis	Setup test project and outlined the structure of the thesis	Research		42
20.10.24	10:20:00	2	Software	Reordering tracks but failed miserably	Implementation		43
03.11.24	19:00:00	4	Project Management	Playbook	Project Management		45
04.11.24	20:00:00	4	Project Management	Playbook & Presentation	Project Management		45
06.11.24	14:00:00	3	Software	Server Cleanup and preperation for Infotage	Implementation	Server	45
10.11.24	13:00:00	4	Software	Catching up to speed on the synthesizer code	Research		46
18.11.24	19:00:00	2	Software	Outlined some ideas for the Toast API and made a mockup	Implementation	Squavy	47
01.12.24	16:00:00	3	Software	Settings Modal	Implementation	Squavy	49
02.12.24	18:25:00	2	Software	Fixed a weird browser dependend bug	Implementation	Squavy	49
03.12.24	19:00:00	3	Software	Settings Modal contd.	Implementation	Squavy	49
11.12.24	16:50:00	2	Software	Another try of reordering of the tracks	Implementation	Squavy	50
18.12.24	15:00:00	6	Software	Rewrite	Implementation		51
26.12.24	14:00:00	5	Software	Bug fixing session in the frontend + major git issue	Implementation	Squavy	52
27.12.24	17:00:00	3	Software	Modified the SynthEditor	Implementation	Squavy	52
02.01.25	22:00:00	2	Software	Started working on a displacement issue while dragging panels	Implementation	Squavy	1
03.01.25	15:00:00	8	Software	Displacing issues is tougher than expected	Implementation	Squavy	1
07.01.25	09:00:00	1	Project Management	Jour Fixe	Project Management		2
08.01.25	20:00:00	3	Software	Reviewing Squanels Code & Start to implement Closing of Panels	Implementation	Squanels	2
08.01.25	23:00:00	2	Software	Implemented basic panel hiding	Implementation	Squanels	2
09.01.25	20:00:00	2	Software	Modified the panel hiding process	Implementation	Squanels	2
10.01.25	15:00:00	1,5	Software	Made some adjustments to the frontend for the trailer	Implementation	Squavy	2
14.01.25	09:55:00	1,5	Project Management	Jour Fixe	Project Management		3
25.01.25	18:00:00	5	Project Management	Analysis and Businessplan	Project Management		4
26.01.25	18:00:00	6	Documentation	Businessplan	Project Management		5
27.01.25	19:00:00	4	Documentation	Businessplan and Projektbericht	Project Management		5
28.01.25	22:00:00	1	Documentation	Businessplan and Projektbericht	Project Management		5
29.01.25	22:00:00	2	Documentation	Projektbericht	Project Management		5
30.01.25	20:40:00	4	Documentation	Projektbericht	Project Management		5
31.01.25	17:00:00	1	Documentation	Changed Projektbericht	Project Management		5
07.02.25	18:00:00	2,2	Software	Started on the rework of the frontend with hummel	Implementation	Squavy	6
14.02.25	18:00:00	3	Software	Continued work on the rework	Implementation	Squavy	7
15.02.25	16:40:00	4,5	Software	Reworking Midi-Editor	Implementation	Squavy	7
18.02.25	17:30:00	4	Software	Continued work on the Midi-Editor	Implementation	Squavy	8
20.02.25	18:00:00	2	Software	Mockup of the Homepage	Implementation	Homepage	8
22.02.25	13:00:00	8,3	Software	Struggling with the Aseprite build for the deployment of the frontend	Implementation	Squavy	8
23.02.25	15:00:00	3	Software	Aseprite build works, now hopefully also with automatic deployment	Implementation	Squavy	9
01.03.25	16:00:00	3	Software	Protoyped the CI/CD layout with Alejandro	Implementation		9
02.03.25	16:00:00	1	Software	Protoyped the CI/CD layout with Alejandro	Implementation		10
04.03.25	15:30:00	3	Software	Boring Synthesizer work	Implementation	*Synth*	10
05.03.25	18:00:00	2,5	Software	Finalized a look of the knobs with hummel	Implementation	Squavy	10
08.03.25	19:20:00	2	Software	Context Menu bug and overflow	Implementation	Squavy	10
14.03.25	16:20:00	4	Software	Continued rework on the editors for the frontend	Implementation	Squavy	11
15.03.25	17:30:00	3	Project Management	Meeting	Project Management		11
16.03.25	16:30:00	4	Software	Fixed the Squavy Dockerfile with Fabian Hummel and Alejandro	Implementation	Squavy	11
16.03.2025	18:30:00	1,5	Software	Synth Editor node connection progress	Implementation	Squavy	12
16.03.2025	12:20:00	2	Diploma Thesis	Started working on the thesis	Research		12
22.03.25	13:35:00	2	Software	Rust Server metrics implementation	Implementation	Server	12
23.03.2025	14:00:00	4	Software	implemented everything else for the diploma thesis	Implementation		13
05.04.2025	08:00:00	4,5	Diploma Thesis	Worked on the implementation part of the thesis	Research		14

Alejandro Dario Tomeniuc:

Date	Start	Duration	Work package	Description	Kategorie	Repository	KW
22.09.24	20:00:00	4	Project Management	Setting up and automating the timekeeping document for the team.	Project Management		39
01.10.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		40
04.10.24	20:00:00	1	Project Management	Squint Meeting	Project Management		40
07.10.24	12:35:00	1	Documentation	Prepare Template for exercises	Project Management		41
08.10.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		41
08.10.24	13:30:00	2	Software	Set up Aseprite locally	Implementation		41
08.10.24	19:00:00	3	Documentation	SWOT Analysis and TOWS Matrix	Project Management		41
15.10.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		42
15.10.24	23:00:00	1	Documentation	Formatting Notion Templates	Project Management		42
18.10.24	19:00:00	1	Project Management	Squint Meeting	Project Management		42
22.10.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		43
22.10.25	20:00:00	2	Documentation	Formatting Templates	Project Management		43
02.11.24	21:00:00	3	Diploma Thesis	Preparation of the Techstudy	Research		44
03.11.24	13:00:00	4	Diploma Thesis	Finalize the Techstudy and decision about technologies	Research		45
03.11.24	20:00:00	3	Documentation	Post-Kickoff Documentation	Project Management		45
04.11.24	18:00:00	1	Project Management	Preparation in advance for Meetings I couldn't attend	Project Management		45
20.11.24	18:00:00	2	Project Management	Squint Meeting - Jour Fixe and Templates	Project Management		47
03.12.24	08:50:00	1	Project Management	Preparation of Templates	Project Management		49
03.12.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		49
08.12.24	13:00:00	3	Software	Software Engineering Concept, AsciiDoc	Implementation		50
08.12.24	20:00:00	4	Software	Software Engineering Concept, AsciiDoc	Implementation		50
10.12.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		50
17.12.24	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		51
29.12.24	12:00:00	6	Diploma Thesis	Research different deployment strategies and technology	Research		52
30.12.24	16:00:00	6	Documentation	Research different deployment strategies and technology	Research		52
31.12.24	10:00:00	8	Software	Try out deployment technologies like puppet	Implementation		52
07.01.25	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		2
07.01.25	09:55:00	2	Documentation	Prepare Template for exercises	Project Management		2
12.01.25	22:00:00	3	Documentation	Status Report, Docs and Presentation	Implementation		3
14.01.25	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		3
25.01.25	18:00:00	6	Project Management	Analysis and Businessplan	Implementation		4
26.01.25	18:00:00	7	Documentation	Businessplan	Project Management		5
27.01.25	19:00:00	5	Documentation	Businessplan and Projektbericht	Project Management		5
28.01.25	22:00:00	3	Documentation	Businessplan and Projektbericht	Project Management		5
29.01.25	22:00:00	2	Documentation	Projektbericht	Project Management		5
30.01.25	20:40:00	3	Documentation	Projektbericht	Project Management		5
31.01.25	17:00:00	2	Documentation	Überarbeitung und Korrektur	Project Management		5
05.02.25	20:30:00	6	Software	Setup Aseprite (Again and Again & Again)	Implementation		6
05.02.25	12:00:00	3	Software	CICD Research	Implementation		6
06.02.25	22:00:00	4	Software	Research Aseprite Design, Solid JS Concepts, Homepage	Implementation		6
11.02.25	09:55:00	1	Project Management	Squint Meeting - Jour Fixe	Project Management		7
11.02.25	15:30:00	1	Project Management	Managing Registration	Project Management		7
15.02.25	15:00:00	1	Software	Setup Repo Locally, Research on SolidJS	Implementation	Homepage	7
15.02.25	18:00:00	3	Software	Homepage Design and SolidJS Research	Implementation	Homepage	7
16.02.25	11:00:00	5,5	Software	SquavyHeader, RetroStyle (Homepage)	Implementation	Homepage	8
16.02.25	18:00:00	2	Software	Loader	Implementation	Homepage	8
18.02.25	12:00:00	0,5	Software	Fixed Loading Animation	Implementation	Homepage	8
19.02.25	17:30:00	1,5	Software	Homepage	Implementation	Homepage	8
19.02.25	19:00:00	0,5	Project Management	Preparing the Sprint Meeting Presentation	Project Management		8
01.03.25	13:00:00	6	Software	Homepage Layout, Experiment with Framer Motion	Implementation	Homepage	9
01.03.25	22:00:00	3	Software	Still working on the homepage and fixing bugs	Implementation	Homepage	9
04.03.25	13:00:00	1	Software	Homepage	Implementation	Homepage	10
04.03.25	22:00:00	3	Software	FullpageJS	Implementation	Homepage	10
05.03.25	09:00:00	3	Software	CSS Carousel	Implementation	Homepage	10
06.03.25	15:00:00	3	Software	Homepage	Implementation	Homepage	10
08.03.25	13:00:00	4	Software	Achievements, Footer and PriceTable Rework	Implementation	Homepage	10
12.03.25	15:00:00	3	Software	Homepage	Implementation	Homepage	11
13.03.25	22:00:00	3	Software	Homepage	Implementation	Homepage	11
14.03.25	07:00:00	0,5	Software	Dockerizing	Implementation	Homepage	11
14.03.25	22:00:00	4	Software	Dockerizing Homepage, Nginx	Implementation	Homepage	11
15.03.25	17:30:00	3	Project Management	Meeting	Project Management		11
15.03.25	20:30:00	3	Software	Docker and Jenkins Pipeline	Project Management		11
16.03.25	13:00:00	5	Software	Docker and Jenkins Pipeline	Implementation	Squavy	12
16.03.25	18:00:00	1	Documentation	Meeting and Final Handin	Project Management		12
28.03.25	20:00:00	4	Software	Docker and Jenkins Pipeline	Implementation	Squavy	13
29.03.25	16:00:00	8	Diploma Thesis	Diploma Thesis	Research		13
30.03.25	13:00:00	5	Diploma Thesis	Diploma Thesis	Research		14
30.03.25	19:00:00	2	Software	Jenkins Pipeline SSH Key	Implementation	Squavy	14
30.03.25	21:00:00	3	Diploma Thesis	Diploma Thesis	Research		14

4.4. Meeting Protocols

Project Kickoff Meeting

Date	@September 22, 2024
Organizer	MÜD
Internal Participants	Ⓐ Alejandro Fabii Ⓚ Konrad Simlinger MÜD
Location	Remote

Meeting Protocol 2024-09-22

Topics:

Organization:

- We will use additional remotes for our GitHub repos (With Unger Klaus as the owner and not under the Spengergasse organization)
- Meetings will be held twice a month, with a big one at the end of each month (30 min) and a small one every two weeks in between (15 min)
- Time tracking in the excel file in the teams-channel
- The first small meeting is scheduled on 2024.10.04

The ABA-Portal registration is completed

Squint Meeting

Date	@October 4, 2024
Location	

Meeting Protocol 2024-10-01

Topics:

Discussed Kanban board:

(screenshot usw.)

Assigned Tasks:

Bug: Convenient Note Preview → Fabian Hummel

Feat: Reorder Tracks → Konrad Simlinger

Squour Fixe

Date	@October 8, 2024 9:55 AM → 10:45 AM
Organizer	Ⓐ Alejandro
Internal Participants	Ⓐ Alejandro Fabii Ⓚ Konrad Simlinger MÜD
Location	HTL Spengergasse

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~
    - Removed “move to pipeline” task

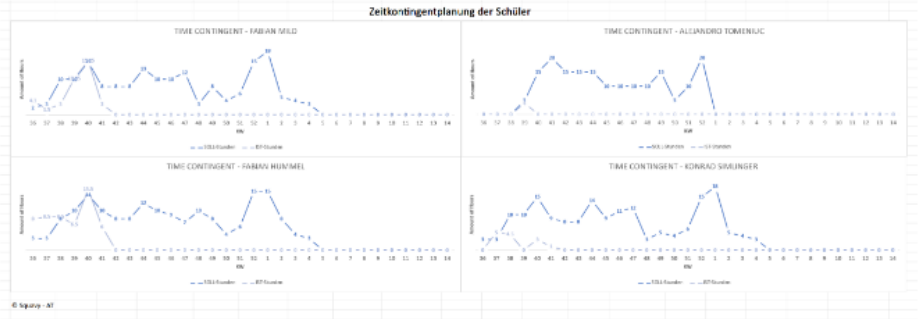
Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meeitngs.	yellow
Shared understanding	The team’s collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green

Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~



Discussion Topics and Tasks:

Name	Last Tasks	Tasks for next Sprint
Hummel	SMSW, Plugins	Squidge, Synthesizer improvements
Mild	Noise Audionode	Node Deletion
Simlinger	Fix Renaming, Bug Fixing	Toast API, Track reorder
Tomeniuc	SW Engineering Concept	Reimagine the Knobs, SWE Concept

Squour Fixe

Date	@October 15, 2024 9:55 AM → 10:45 AM
Organizer	Ⓐ Alejandro
Internal Participants	Ⓐ Alejandro Fabii Ⓚ Konrad Simlinger MÜD
Location	HTL Spengergasse

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
- Budgeting and Sponsoring
- Participation in Jugend Innovativ
- Trademark law
- JUICE (loading of VSt plugins)

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Load plugins using Squidge and thought how it will be integrated into Squavy	Apply native window around plugins
Mild	Node Deletion	Bug Fixing, BPM Support, Finish Context Menu

Name	Last Tasks	Tasks for next Sprint
Simlinger	Rust routes and prometheus exporter	Toast API, Track reorder
Tomeniuc	Reimagine the Knobs, SWE Concept	Reimagine the Knobs, SWE Concept

Notes:

- Hummel: Unassigned from some tasks due to prioritisation of other tasks

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	yellow
Team Communication	Frequency and quality of team meetings.	green
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

✨ VITAL finally loads!!!!! ✨ (Totale Vitale Entladung)

Squint Meeting

Date	@October 18, 2024 9:55 AM → 10:45 AM
Organizer	MÜD
Internal Participants	Ⓐ Alejandro Fabii Ⓚ Konrad Simlinger MÜD
Location	Remote

Weekly Meeting Checklist:

- [x] ~~What did we do this sprint?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Open Topics (to mention):

- Meeting(s) with project partner

Discussion Topics and Tasks:

- [x] ~~Update~~
- [x] ~~New Tasks~~
- Meeting with project partner when a suitable prototype is ready

Notes:

- Board Progress, show amount of tasks

Squour Fixe

Date	22. Oktober 2024 09:55 → 10:45
Organizer	A Alejandro
Internal Participants	A Alejandro, Fabii, K Konrad Simlinger, MÜD
External Participants	Leer
Location	HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Time management for the autumn holidays
  - VST integration

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	green
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

Squour Fixe

Date	5. November 2024 09:55 → 10:45
Organizer	MÜD
Internal Participants	Fabii, Konrad Simlinger, MÜD
External Participants	Leer
Location	HTL Spengergasse

Eigenschaft hinzufügen

Kommentare

Alejandro 05.11.2024 (bearbeitet)
Task registration prepared in Advance by Alejandro Tomeniuc because of Absence due to medical conditions

Alejandro 05.11.2024
Please define the KPI status without me this sprint

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Midi Controller support partly working as intended
  - Progress regarding VST support
  - FrontEnd bug fixes

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Playback Head	Squidge, Playback Head
Mild	Midi Support, Audionode Implementation	Reimagine the Knob, Midi Support (test, complete, and merge)
Simlinger		
Tomeniuc	SWE Concept, Post Kickoff, (Reimagine the Knob)	SWE Concept, Architecture, Feature Homepage

Notes:

- Tomeniuc: Unassigned from "Reimagine the Knobs"

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	yellow
Team Communication	Frequency and quality of team meetings.	green
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date	12. November 2024 09:55 → 10:45
Organizer	MÜD
Internal Participants	Fabii, Konrad Simlinger, MÜD
External Participants	Leer
Location	HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Discussed post-kickoff and other school related topics

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

  No changes in tasks

Name	Last Tasks	Tasks for next Sprint
Hummel	-	-
Mild	-	-
Simlinger	-	-
Tomeniuc	-	-

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squint Meeting

Date	16. November 2024 18:00 → 18:30
Organizer	MÜD
Internal Participants	Fabii, Konrad Simlinger, MÜD
External Participants	Leer
Location	Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this sprint?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Open Topics (to mention):

- Meeting(s) with project partner

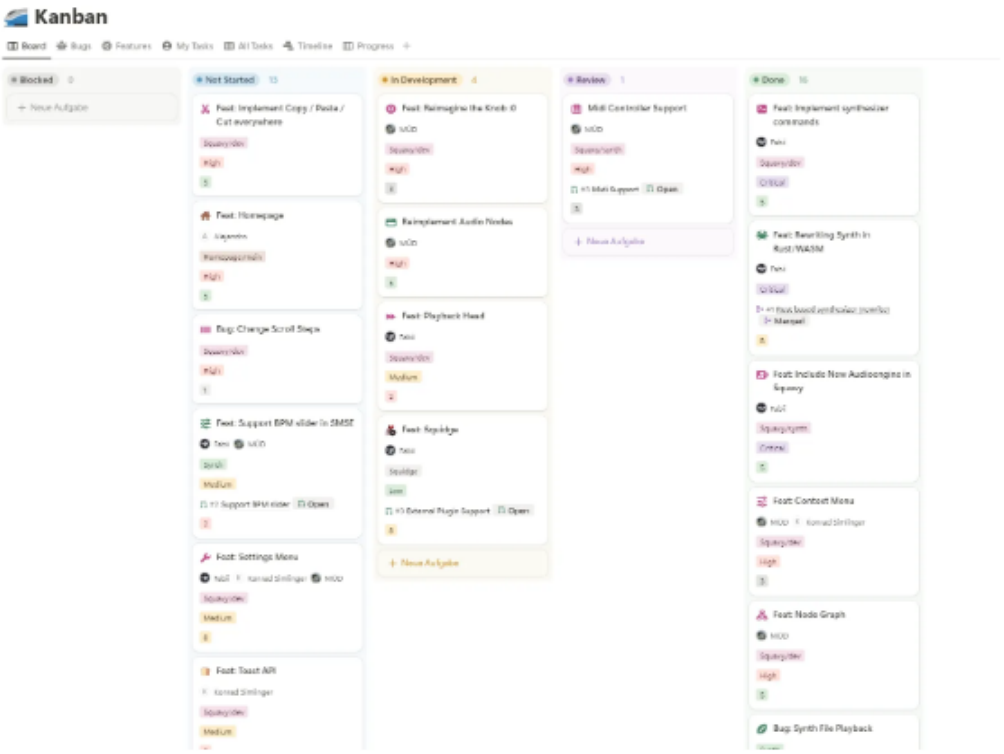
Discussion Topics and Tasks:

- [x] ~~Update~~
- [x] ~~New Tasks~~
- Discussed Kanban Board
- Assigned Tasks:
  - Feat: Reimagine the knob → Fabian Mild
  - Feat: Squidge → Fabian Hummel
- Discussed tasks to be done till the next meeting

Notes:

- Board Progress, show amount of tasks

Snapshots from Kanban and Timeline:



Squour Fixe

Date	20. November 2024 18:00 → 18:50
Organizer	Alejandro MÜD
Internal Participants	Fabii MÜD Alejandro
External Participants	Leer
Location	Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Particion at Jugend Innovativ

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Squidge C++ and Rust interop	Get Rust to work with C++ via CMake and CXX
Mild	MIDI control support	Renew knobs with bindings (+ together with master and limiter)
Simlinger		
Tomeniuc	-	SWE Concept, Architecture, Feature Homepage

Notes:

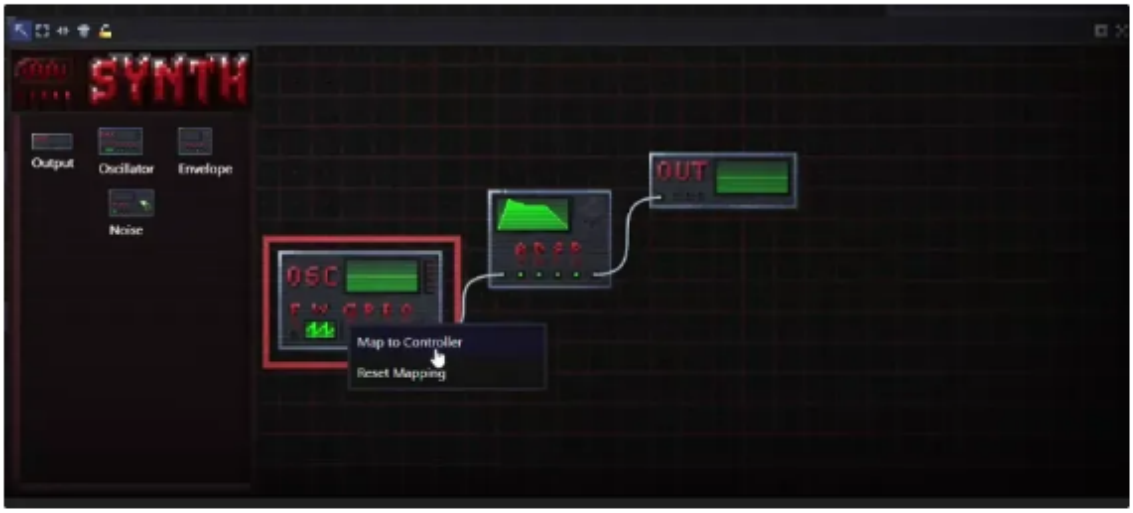
- Less progress due to sick leave (2 weeks, Tomeniuc)

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] Update

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~



Map controls to a MIDI keyboard.

Squour Fixe

Date	3. Dezember 2024 09:55 → 10:45
Organizer	A Alejandro
Internal Participants	Fabii MÜD A Alejandro K Konrad Simlinger
External Participants	Leer
Location	Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Synth is built in Squavy
  - Time Tracking Categories | Updated Tables

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Squidge (Plugin Support)	Squidge (Plugin Support), Squavy Audionode Picker (+Plugins)
Mild		
Simlinger		
Tomeniuc	SWE Concept, Architecture, Code Coverage	SWE Concept, Architecture, Code Coverage

Notes:

- Start Track Automation, Code Coverage, CI/CD
- Documents integrated in Kanban

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date 10. Dezember 2024 09:55 → 10:45

Organizer A Alejandro

Internal Participants Fabii A Alejandro

External Participants Leer

Location Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - SWE Concept (written in AsciiDoc) push to squavy? (ongoing documentation for changes)

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Synth SMSE	Rust CXX
Mild	-	-
Simlinger		
Tomeniuc	Finalized SWE Concept Version 1.0 and Architecture v2	Homepage, Code-Coverage

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squint Meeting

Date	13. Dezember 2024 18:00 → 18:30
Organizer	Alejandro
Internal Participants	Fabii, Konrad Simlinger
External Participants	Leer
Location	HTL Spengergasse

Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this sprint?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Open Topics (to mention):

- Meeting(s) with project partner

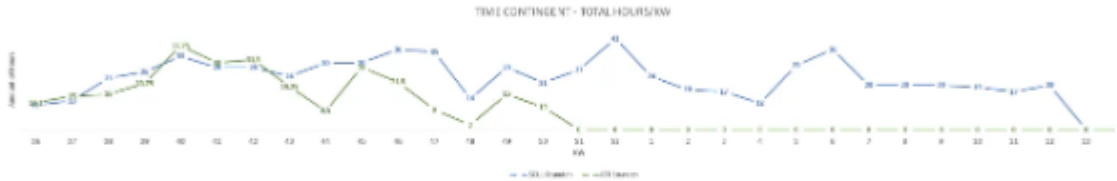
Discussion Topics and Tasks:

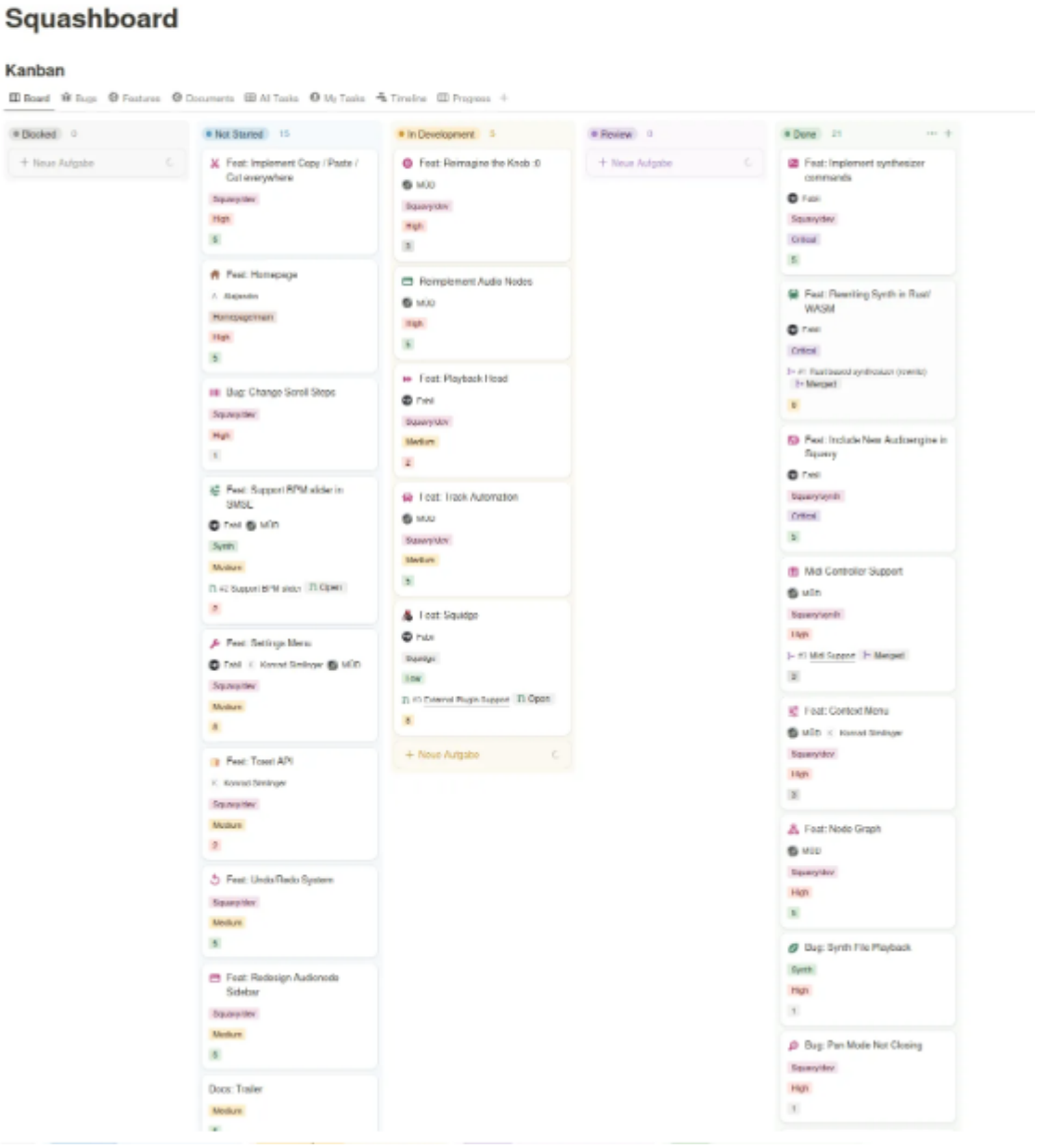
- [x] ~~Update~~
- [x] ~~New Tasks~~
- Discussed Kanban Board
- Assigned Tasks:
  - -
- Discussed tasks to be done till the next meeting

Notes:

- Board Progress, show amount of tasks

Snapshots from Kanban and Timeline:





Squour Fixe

Date	17. Dezember 2024 09:55 → 10:45
Organizer	Alejandro
Internal Participants	Fabii, MÜD, Alejandro, Konrad Simlinger
External Participants	Leer
Location	HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Status

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

No changes in tasks

Name	Last Tasks	Tasks for next Sprint
Hummel	-	-
Mild	-	-
Simlinger	-	-
Tomeniuc	-	-

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] Update

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date	7. Januar 2025 09:55 → 10:45
Organizer	Alejandro
Internal Participants	Fabii, MÜD, Alejandro, Konrad Simlinger
External Participants	Leer
Location	Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
- Liste

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel		
Mild	Reimagine the Knob, Track Automation	Track Automation, Synth Antialiasing, BPM Support
Simlinger		
Tomeniuc	Research different deployment strategies and Technologies	Code-Coverage/Testplan

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date	14. Januar 2025 09:55 → 10:45
Organizer	Alejandro
Internal Participants	Fabii, MÜD, Alejandro, Konrad Simlinger
External Participants	Leer
Location	HTL Spengergasse

Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Business Plan

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Squidge, BPM Support, Status Report	Squidge, BPM Support
Mild	Track Automation, Synth Antialiasing, BPM Support, Status Report	Track Automation, Synth Antialiasing, BPM Support
Simlinger	History, Status Report	CICD/Monitoring Platform
Tomeniuc	Testplan, Status Report	CICD/Monitoring Platform

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date	21. Januar 2025 09:55 → 10:45
Organizer	Alejandro
Internal Participants	Fabii, MÜD, Alejandro, Konrad Simlinger
External Participants	Leer
Location	HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Business Plan

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Squidge, BPM Support	Squidge, BPM Support
Mild	Track Automation, Synth Antialiasing, BPM Support	Track Automation, Synth Antialiasing, BPM Support
Simlinger	CICD/Monitoring Platform	CICD/Monitoring Platform
Tomeniuc	CICD/Monitoring Platform	CICD/Monitoring Platform

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date	28. Januar 2025 09:55 → 10:45
Organizer	Alejandro
Internal Participants	Fabii, MÜD, Alejandro, Konrad Simlinger
External Participants	Leer
Location	Remote

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
- Liste

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Squidge	Squidge, BPM Support
Mild	Reimagine the Knob, Track Automation	Track Automation, Synth Antialiasing, BPM Support
Simlinger	Better Knob support	History
Tomeniuc	Research different deployment strategies and Technologies	Code-Coverage/Testplan

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squour Fixe

Date 11. Februar 2025 09:55 → 10:45

Organizer Alejandro

Internal Participants Fabii MÜD Alejandro

External Participants Leer

Location HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
  - Liste

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	New Synth; Start of new Design	Embed it into Squavy
Mild	Complete the Knob	Review and work on track automation
Simlinger	Implementing parts of diploma thesis in the project	continue to do so
Tomeniuc	Setup Aseprite, Start Homepage	Homepage

Notes:

- Liste

Sprint Retrospective | Discussion:

- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	yellow

- [x] ~~Snapshots from progress~~

Squint Meeting

Date	19. Februar 2025 18:00 → 18:30
Organizer	Alejandro
Internal Participants	Fabii, MÜD
External Participants	Leer
Location	Remote

Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this sprint?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Open Topics (to mention):

- Liste

Discussion Topics and Tasks:

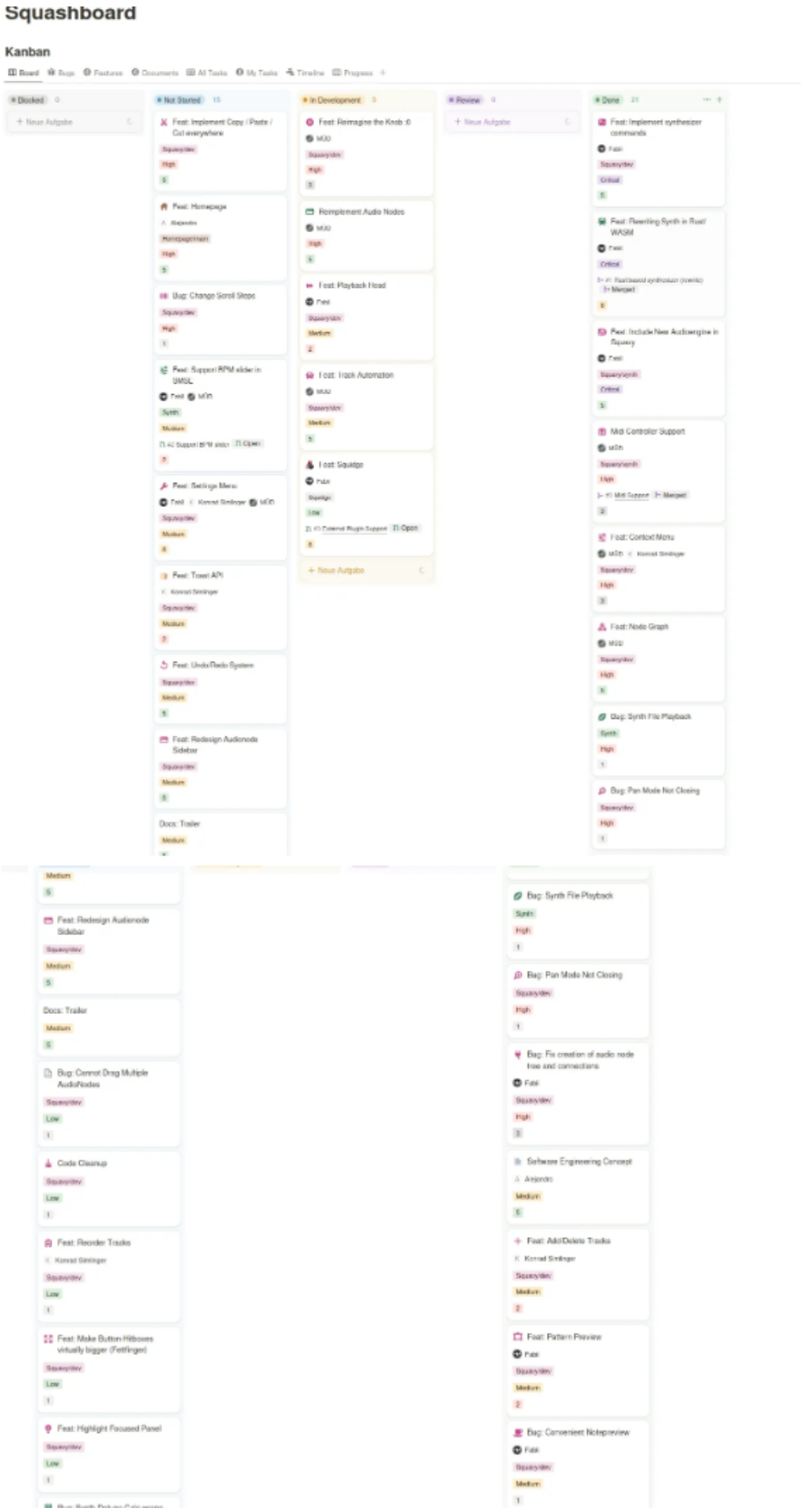
- [x] ~~Update~~
- [x] ~~New Tasks~~
- Discussed Kanban Board
- Assigned Tasks:
  - **Hummel:** Redesign and App Refinement; good Progress on new Synth
  - **Tomeniuc**: Dockerize the different Repositories and set up the CI/CD Environment
- Discussed tasks to be done till the next meeting

Notes:

- Board Progress, show amount of tasks

Snapshots from Kanban and Timeline:

(Muss aktualisiert werden):


Squashboard
Kanban
Board
Bugs
Features
Documents
AI Tasks
My Tasks
Timeline
Progress
Blocked
Not Started
In Development
Review
Done
Feat: Implement Copy / Paste / Cut everywhere
Feat: Homepage
Bug: Change Scroll Steps
Feat: Support RPM slider in SMSL
Feat: Settings Menu
Feat: Towel API
Feat: Undo/Redo System
Feat: Redesign Audionode Sidebar
Docs: Trailer
Feat: Reimagine the Knob :0
Reimplement Audio Nodes
Feat: Playback Head
Feat: Track Automation
Feat: Squidpp
Feat: Implement synthesizer commands
Feat: Rewriting Synth in Rust/WASM
Feat: Include New Audioengine in Squavy
Midi Controller Support
Feat: Context Menu
Feat: Node Graph
Bug: Synth File Playback
Bug: Pan Mode Not Closing
Bug: Cannot Drag Multiple AudioNodes
Code Cleanup
Feat: Reorder Tracks
Feat: Make Button Hitboxes virtually bigger (Fettfinger)
Feat: Highlight Focused Panel
Bug: Fix creation of audio node tree and connections
Software Engineering Concept
Feat: Add/Delete Tracks
Feat: Pattern Preview
Bug: Convenient Notepreview

Squint Meeting

Date	10. März 2025 18:00 → 18:30
Organizer	Konrad Simlinger
Internal Participants	Fabii, MÜD, Konrad Simlinger
External Participants	Leer
Location	HTL Spengergasse

+ Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this sprint?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

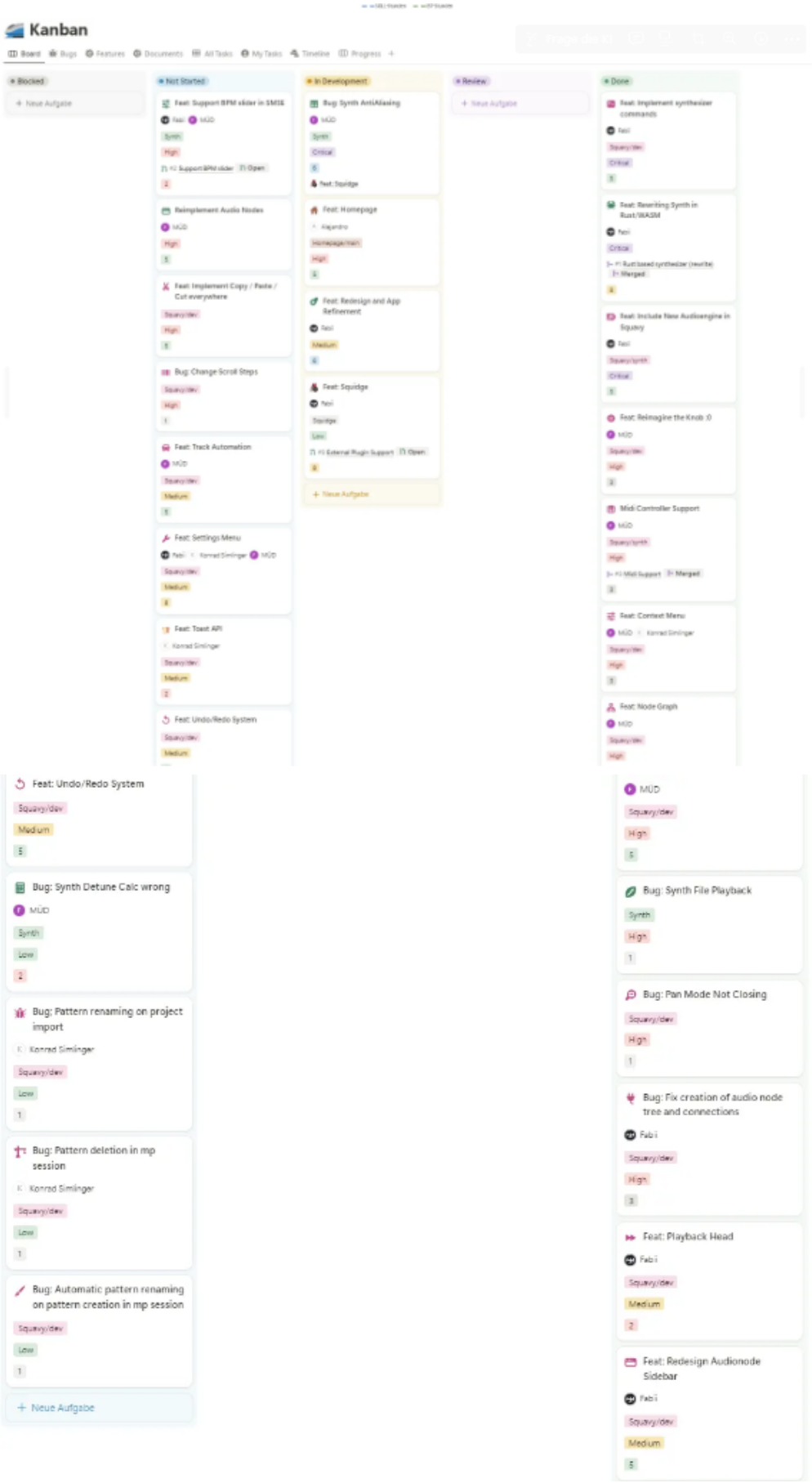
Open Topics (to mention):

- Diploma Thesis

Discussion Topics and Tasks:

- [x] ~~Update~~
- [x] ~~New Tasks~~
- Discussed Kanban Board
- Discussed Diploma Thesis of the individual students
- Removed completed tasks from kanban board to allow for more notion text blocks

Snapshots from Kanban and Timeline:


Kanban
Board
Bugs
Features
Documents
All Tasks
My Tasks
Timeline
Progress
Blocked
Not Started
In Development
Review
Done
Feat: Undo/Redo System
Squavy/dev
Medium
5
Bug: Synth Detune Calc wrong
MUD
Synth
Low
2
Bug: Pattern renaming on project import
Konrad Simlinger
Squavy/dev
Low
1
Bug: Pattern deletion in mp session
Konrad Simlinger
Squavy/dev
Low
1
Bug: Automatic pattern renaming on pattern creation in mp session
Squavy/dev
Low
1
+ Neue Aufgabe
MUD
Squavy/dev
High
5
Bug: Synth File Playback
Synth
High
1
Bug: Pan Mode Not Closing
Squavy/dev
High
1
Bug: Fix creation of audio node tree and connections
Fabi
Squavy/dev
High
3
Feat: Playback Head
Fabi
Squavy/dev
Medium
2
Feat: Redesign Audionode Sidebar
Fabi
Squavy/dev
Medium
5

Squour Fixe

Date 15. März 2025 09:55 → 10:45

Organizer Alejandro

Internal Participants Fabii MÜD Alejandro Konrad Simlinger

External Participants Leer

Location Remote

Eigenschaft hinzufügen

Kommentare

Kommentar hinzufügen...

Weekly Meeting Checklist:

- [x] ~~What did we do this week?~~
- [x] ~~Are there any blockers? What external resources can minimize them?~~

Discussion Topics and Tasks:

- [x] ~~Update~~
- Liste

Sprint Planning:

- [x] ~~Grooming the backlog~~
  - [x] ~~Push important artifacts~~
  - [x] ~~Assign package sizes~~
  - [x] ~~Pull tasks from the backlog~~

Name	Last Tasks	Tasks for next Sprint
Hummel	Rewrote MIDI and Synth editor (70% done)	Further improvements of frontend and integration of Squidge
Mild	Review and work on track automation	Finally add anti-aliasing for the synthesizer, improve frontend
Simlinger	The last additions for the diploma thesis	look into hummel rewrite
Tomeniuc	Homepage, Docker	Docker, CI/CD, Diploma Thesis

Notes:

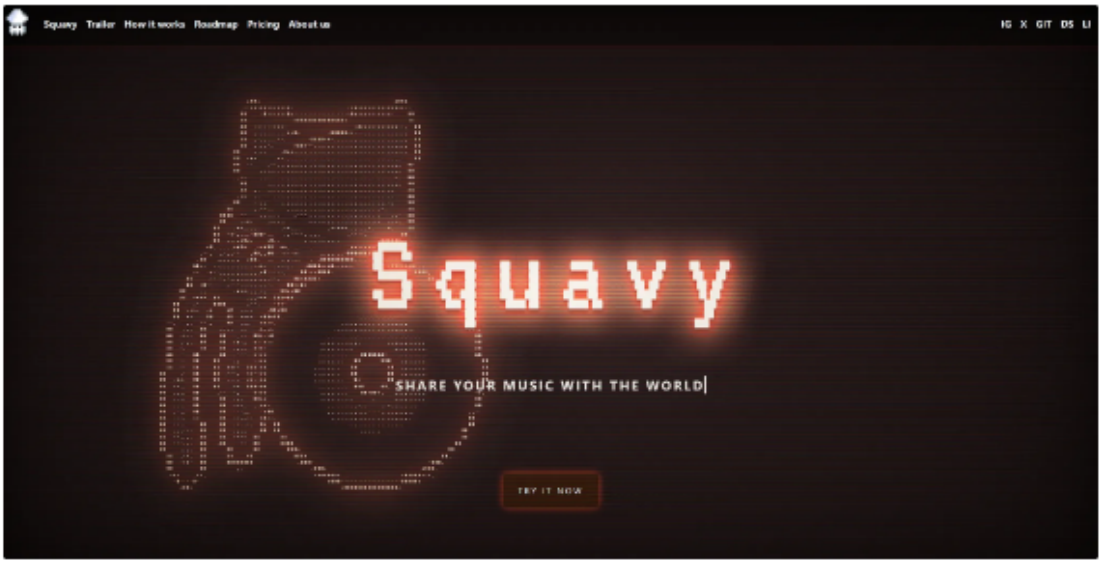
- Clean the notion board
- Work on the final project documentation for Squavy

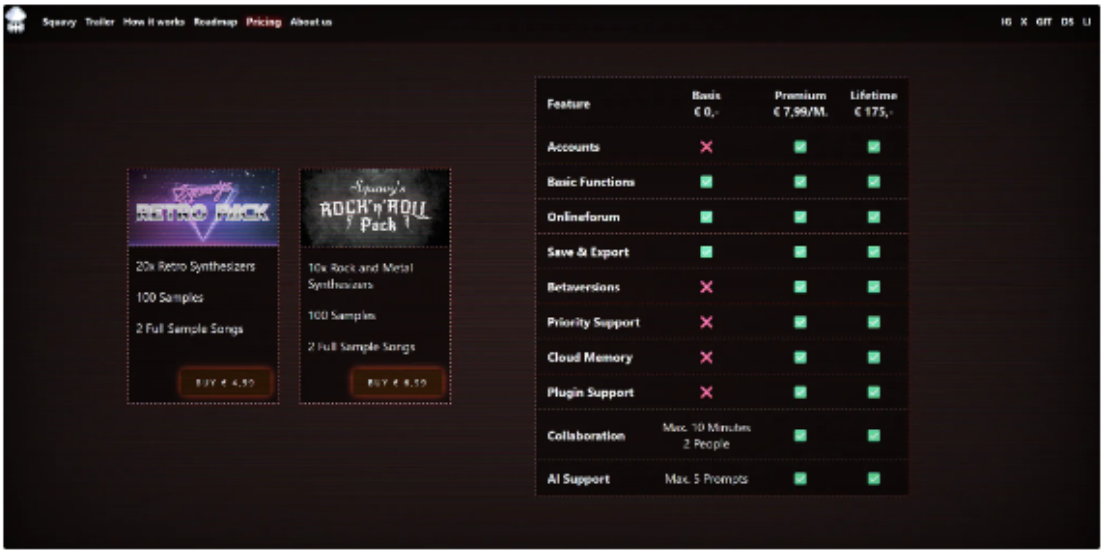
Sprint Retrospective | Discussion:

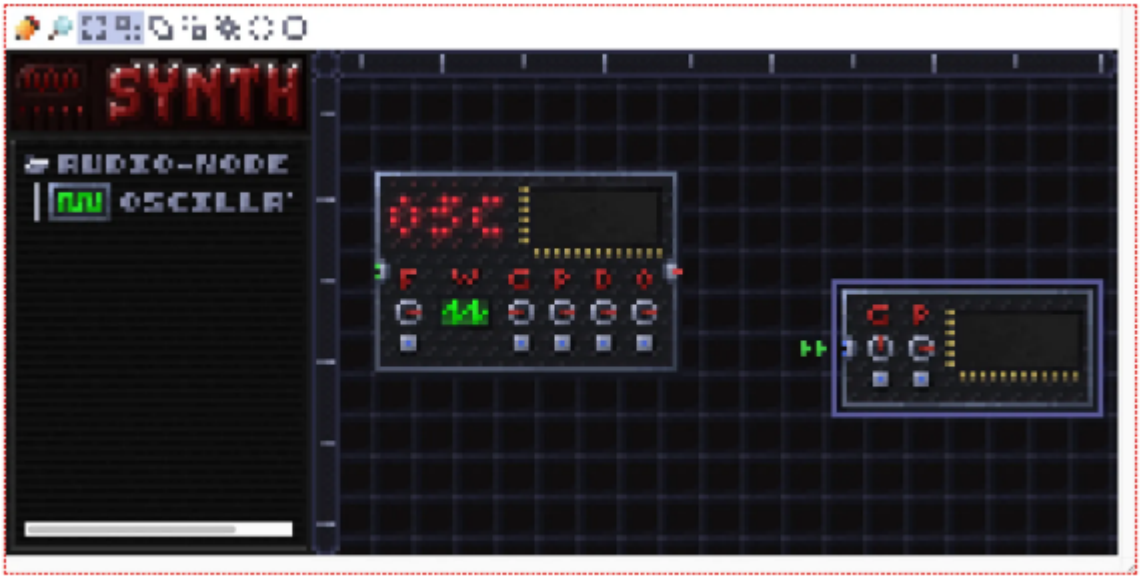
- [x] ~~3 KPIs (red, yellow, green):~~
- [x] ~~Update~~

Name	Description	Status
Squelocity	How well the project is adhering to its timeline.	green
Team Communication	Frequency and quality of team meetings.	yellow
Shared understanding	The team's collective understanding of the project from a conceptual and technical point of view	yellow
Squality Control	number of tracked bugs, code quality	✨ top-tier ✨
Stakeholder Communication	Scheduled meetings with the partner are held as well as communication with the supervisor.	green
Budgeting	Difference the difference between actual cost and earned value	green

- [x] ~~Snapshots from progress~~






SYNTH
AUDIO-NODE
OSCILLA


0.25
0
0.25
0.5
0.75
1
1.25
1.5

4.5. Legal declaration

Berufsbildende Schulen

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß den geltenden schulrechtlichen Bestimmungen die Ausarbeitung einer abschließenden Arbeit (Diplomarbeit bzw. Abschlussarbeit) mit folgender Aufgabenstellung gewählt:

Squavy - Browser Based Digital Audio Workstation

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- Alejandro Dario Tomeniuc (5AHIF): **Deployment and Scalability Strategies and their Implementation for Web-Applications**
- Fabian Mild (5AHIF): **Implementation of audio and signal processing in modern web browsers**
- Konrad Simlinger (5AHIF): **Monitoring of custom front and backend software using tools like Prometheus and Grafana.**
- Fabian Hummel (5AHIF): **Leveraging RFC6455 for low-latency, concurrent collaboration**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die abschließende Arbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen abschließenden Arbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **09.04.2025** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen auch zur Kenntnis, dass ein Abbruch der abschließenden Arbeit nicht möglich ist.

Kandidaten/Kandidatinnen:	**Datum und Unterschrift bzw. Handysignatur:**
Alejandro Dario Tomeniuc (5AHIF)	20.09.2024 Tomeniuc
Fabian Mild (5AHIF)	20.09.2024 Mild
Konrad Simlinger (5AHIF)	20.09.2024 Simlinger
Fabian Hummel (5AHIF)	20.09.2024 Hummel

4.6. Cooperation Contract

DIPLOMPROJEKT SPENGERGASSE
ausbildung mit zukunft

KOOPERATIONSVEREINBARUNG

zwischen

Digimagical Gmbh

Legstattgasse 4-6/C25
3001 Mauerbach

Christian Haase
(in Folge Projektpartner genannt)

und

Fabian Mild

Fabian Hummel

Konrad Simlinger

Alejandro Dario Tomeniuc

(in Folge Projektteam genannt)

PRÄAMBEL

Das Projektteam und der Projektpartner/die Projektpartnerin beabsichtigen gemäß der Prüfungsordnung BMHS, BGBl II Nr. 177/2012 i.d.g.F., die Planung und Durchführung eines Diplomprojektes mit dem Titel:

Squavy

welches die Erstellung von Arbeitsergebnissen, zum Thema des Diplomprojekts, zum Gegenstand hat.

Durch die Zusammenarbeit soll insbesondere den Mitgliedern des Projektteams die Möglichkeit eingeräumt werden, im Rahmen ihrer schulischen Ausbildung bei der Durchführung eines Diplomprojektes an die Verhältnisse im technischen Berufsleben herangeführt zu werden, um dabei die in der Schule erworbenen theoretischen Kenntnisse und Fähigkeiten in der Praxis anzuwenden bzw. zu erweitern. Hingewiesen wird in diesem Zusammenhang auf den unentgeltlichen Charakter dieser Vereinbarung.

§1
Gegenstand

Gegenstand ist die Erstellung von Arbeitsergebnissen zum Thema des Diplomprojekts gemäß beiliegender Projektbeschreibung. Der Projektpartner / die Projektpartnerin wird jedoch darauf hingewiesen, dass es sich um ein Projekt im Zusammenhang mit der schulischen Ausbildung handelt und daher jede Haftung des Projektteams, insbesondere in Hinsicht auf die Unentgeltlichkeit des Vertrages, ausgeschlossen ist.

§2
Laufzeit

Die vorliegende Kooperation tritt am 2024.09.20 in Kraft und wird bis zum schulrechtlich verordneten Termin am 2025.04.09 abgeschlossen.

§ 3
Rechte und Pflichten des Projektteams

Das Projektteam verpflichtet sich, die im Gegenstand genannten Arbeiten sorgfältig und unter möglichster Schonung der Interessen des Projektpartners / der Projektpartnerin durzuführen.
Das Projektteam verpflichtet sich zur Geheimhaltung aller ihm zur Kenntnis gelangenden Geschäfts- und Betriebsgeheimnisse gegenüber Dritten. Ausnahmen siehe §5 Einsicht und Präsentation.

§4
Rechte und Pflichten des Projektpartners / der Projektpartnerin

Der Projektpartner / die Projektpartnerin verpflichtet sich, das Projektteam nach Kräften zu unterstützen und dem Projektteam folgende Hilfsmittel zeitgerecht und für die Dauer des Diplomprojektes kostenfrei zur Verfügung zu stellen:

- Ausreichende Kommunikationsbereitschaft, um die Anforderungen des Projekts sicherzustellen

Sofern der Projektpartner / die Projektpartnerin dem Projektteam urheberrechtlich geschütztes Material oder Daten zur Verfügung stellt, stellt der Projektpartner / die Projektpartnerin sicher, dass dieses Material frei von Rechten Dritter ist. Der Projektpartner / die Projektpartnerin hält das Projektteam diesbezüglich schad- und klaglos. Sollte das Projektteam im Rahmen dieser Kooperationsvereinbarung ein Werk schaffen, dem Schutz im Sinne des Urheberrechtsgesetzes zukommt, behält das Projektteam jegliche Verarbeitungsrechte an diesem Werk. Das Projektteam verpflichtet sich in diesem Zusammenhang keine Ansprüche bezüglich einer Vergütung zu erheben.

§5
Einsicht und Präsentation

Da die Tätigkeit des Projektteams auch Inhalt bzw. Grundlage der an der Schule HTBLuVA Wien V, Spengergasse 20 zu erstellenden Diplomarbeit ist, berechtigt der Projektpartner / die Projektpartnerin die zuständigen Organe des Bundes zur Einsicht und Kontrolle, um die Aufgaben gem. Prüfungsordnung BMHS, SchUG bzw. SchUG BKV zu erfüllen. Das Projektteam ist auch berechtigt, Ergebnisse der Diplomarbeit im Rahmen des Schulunterrichts und bei Schul- und schulbezogenen Veranstaltungen, sowie bei der Präsentation und Diskussion der Diplomarbeit zu verwenden.

§6
Änderungen

Änderungen dieser Vereinbarung bedürfen der Schriftform. Sollte ein Schüler / eine Schülerin, der / die Mitglied des Projektteams ist, während der Laufzeit dieser Vereinbarung aus der HTBLuVA Wien V, Spengergasse 20 ausscheiden (Abmeldung vom Schulbesuch), bleibt die Kooperationsvereinbarung für die verbleibenden Unterzeichner, mit Rücksichtnahme auf eine etwaige Reduktion des Projektumfanges, aufrecht.

Wien, 17.09.2024 .....

Wien, 17.09.2024 .....

Wien, 17.09.2024 .....

Wien, 17.09.2024 .....

(Ort, Datum) (für das Projektteam)

.....

(Ort, Datum)

.....

(für den/die Projektpartner/in)

	Unterzeichner	Christian Haase
	Datum/Zeit-UTC	2024-09-19T08:06:47+02:00
	Prüfinformation	Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at
Hinweis	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.	

4.7. Initial Project Proposal

Thema: Squavy - Browser Based Digital Audio Workstation

Date: 20.09.2024

Team Members: Fabian Mild, Fabian Hummel, Konrad Simlinger, Alejandro D. Tomeniuc

Individuelle Themenstellung der Kandidatin/des Kandidaten (Teilthemen)

Schüler/in	Individuelle Themenstellung	Abteilung	Verantwortlich
Fabian Hummel	Leveraging RFC6455 for low-latency, concurrent collaboration	HIF	
Fabian Mild	Implementation of audio and signal processing in modern web browsers	HIF	Ja
Konrad Simlinger	Monitoring of custom front and backend software using tools like Prometheus and Grafana.	HIF	
Alejandro Dario Tomeniuc	Deployment and Scalability Strategies and their Implementation for Web-Applications	HIF	

Ausgangslage (max. 400 Zeichen)*

There are numerous DAWs available with varying levels of complexity and features, but the majority are not meant to suit beginners. Squavy is meant to be an introductory and feature rich application, which serves as a start into the music production journey. Squavy is in active development, with many features (audio engine, interactive UI, backend server) in their second development iteration.

Untersuchungsanliegen der individuellen Themenstellungen (max. 2400 Zeichen)*

Hummel: Efficient, fast and robust communication over the network is something we take for granted every day but is easier said than done. Over the course of this diploma, I will analyze and test several different architectures and technologies that score in flexibility, maintainability and performance to seamlessly integrate real-time collaboration into Squavy.

Mild: Since the start of Squavy, we have been testing many different libraries that handle browser audio and signal processing. None of them sufficiently suit the needs of a modern DAW, so we started to develop our own solution. I will explain these different options and highlight the performance, benefits and shortcomings of each of them.

Simlinger: My topic explores how custom metrics can be made available in node and rust software. The goal is to export useful metrics of our front and backend application which then can be used to further understand the server status or to scale services.

Tomeniuc: I will explore how modern web applications can achieve high availability, flexibility, and scalability. The focus will be on efficient scaling with fluctuating user numbers and the challenges posed by on-prem and cloud technologies and containers. The goal is to identify strategies to optimize scalability and efficiency.

Zielsetzung (max. 400 Zeichen)*

Squavy aims to create a user-friendly platform for music enthusiasts and students to collaborate easily. It targets new users looking to start their music production journey without needing to spend countless hours learning the basics. To make music composition even more fun, users can choose to seamlessly get together and collaboratively work on a project.

Geplantes Ergebnis der individuellen Themenstellungen (max. 2400 Zeichen)*

Hummel: The aim for this part of the diploma project is to develop a robust communication protocol for real-time online collaboration, ensuring low-latency and high reliability. The application should be seamlessly integrated into the main program, allowing multiple users to collaborate on music editing in real-time all through within the browser and minimal setup. Additionally, the software should be tailored to our distributed system, facilitating easy deployment and scalability whilst granting us maximum control over the analytics and vitals of the server.

Mild: The audio engine is one of the core systems of Squavy. Here, sounds are synthesized, mixed, modulated, etc. The system must be robust and fast, with little to no room for errors on runtime. It must also include playback-related features and sound import/export.

Simlinger: Deliver a monitoring platform using different tools and services to provide a basis for further processing (e.g. scaling of services) and create a platform to view the current server status.

Tomeniuc: Conception of a scalable deployment for the web application.

Software Requirements Specification

for

Squavy - Browser Based Digital Audio Workstation

Version 1.2 approved

Prepared by Fabian Mild, Fabian Hummel, Konrad Simlinger, Alejandro Dario Tomeniuc

Squavy | Digimagical GmbH

IEEE 830-1998

17/03/2025

Revision History

Name	Date	Reason For Changes	Version
Fabian Mild	2024-09-24	Initial Version	0.1
Fabian Mild, Fabian Hummel	2024-11-03	Edited the open points except for 5.1	0.2
Fabian Mild, Fabian Hummel, Alejandro Tomeniuc	2024-11-04	Revised the remaining points.	0.3
Fabian Mild, Alejandro Tomeniuc, Konrad Simlinger, Fabian Hummel	2024-11-04	Finalized the first version of the document	1.0
Fabian Mild, Alejandro Tomeniuc, Konrad Simlinger, Fabian Hummel	2025-01-10	Made some adjustments to the architecture to fit the improved version of Squavy	1.1
Fabian Hummel	2025-01-11	Synthesizer Architecture	1.2

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a simple and intuitive starting point in music production for beginner music producers and sound designers. The document includes all functional and non-functional requirements for Squavy version 1.0.0. This is a living document, so changes may and should occur throughout the development of Squavy.

1.2 Document Conventions

High-level requirements are specified by detailed requirements in this document.

- Squavy is the project name.
- Project Wiki is to be found on the website and provides all necessary technical documentation.

1.3 Intended Audience and Reading Suggestions

This document is intended for external as well as internal project team members. It provides a general overview of the product and the project's technical frameworks. By reading this document, project members are equipped with the essential understanding required to use Squavy effectively. The intended reading sequence is from top to bottom.

1.4 Product Scope

This document is valid for the project Squavy, the involved organizational units and all work referring to this project. The application serves as a user-friendly environment for music enthusiasts and students which enables them to easily collaborate and be creative with each other. It features different editors for song creation, samples, automation tracks, and many effects such as reverb, envelopes and equalisers. Non-Goals for this project include (for the meanwhile) full VST-Plugin support, sound fonts, a desktop application, and a subscription service. Squavy will not support user accounts and won't feature distracting ads inside the editor.

1.5 Objectives

The project is aimed at developing a user-friendly web-based interface for a digital audio workstation, with a specific focus on enhancing the environment for audio production. Users should have the ability to efficiently create and edit audio tracks with the assistance of a computer. This includes an integrated synthesizer featuring controls for Velocity, Pan, Fine tuning, and many other properties.

The digital audio workstation web-interface should also include a MIDI editor, allowing users to seamlessly create, edit, and integrate MIDI data into their audio projects. Furthermore, users should be able to engage in real-time collaboration through temporary session codes, enabling multiple users to work on audio projects simultaneously within the web-interface.

1.6 Result

The desired result of Squavy is a user-friendly web-based digital audio workstation that is directed to a wide range of music producers but focusses mainly on beginners. The product aims to provide a feature-rich environment for audio production, offering users the ability to create and edit audio tracks, integrate *MIDI*[1] data, engage in real-time collaboration, save and export audio projects, incorporate samples, and utilize various effects. Squavy strives to offer a user-friendly and accessible platform for music production without the need for immediate account registration, making it inclusive for all users.

1.7 Target user groups

Target users of the product are band members and aspiring musicians who want to bring their unique skills and creativity to the table. Collaborative songwriting often benefits from the diverse talents of each person.

Other target groups are music producers, sound designers and songwriters, which contain both professional and amateur musicians who create music as a profession or hobby. Additionally, sound designers may be working in the game development, video and audio production industries, who require music and sound editing software for production work.

Furthermore, there are also educational institutions like schools, music academies and colleges whose instructors can use the software as an educational tool to teach effective music production, sound design and digital audio skills.

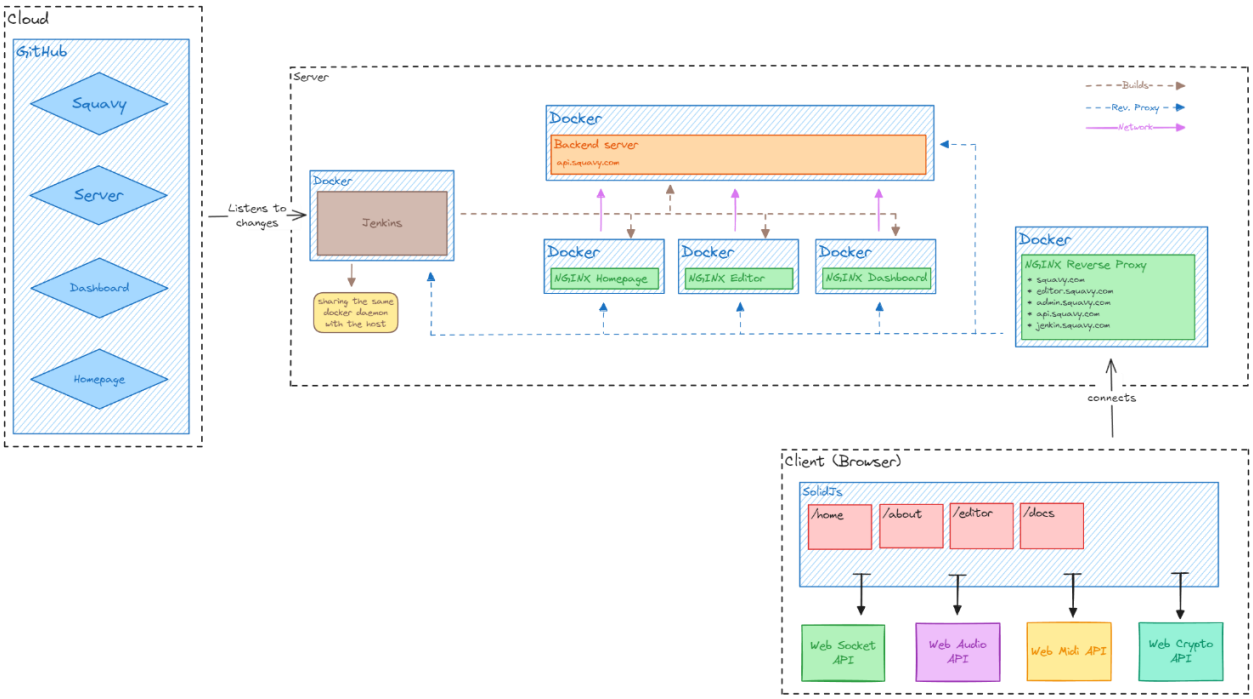
---

1 Musical Instrument Digital Interface (MIDI) is a standard to transmit and store musical notes and timings. Source: https://www.techtarget.com/whatis/definition/MIDI-Musical-Instrument-Digital-Interface

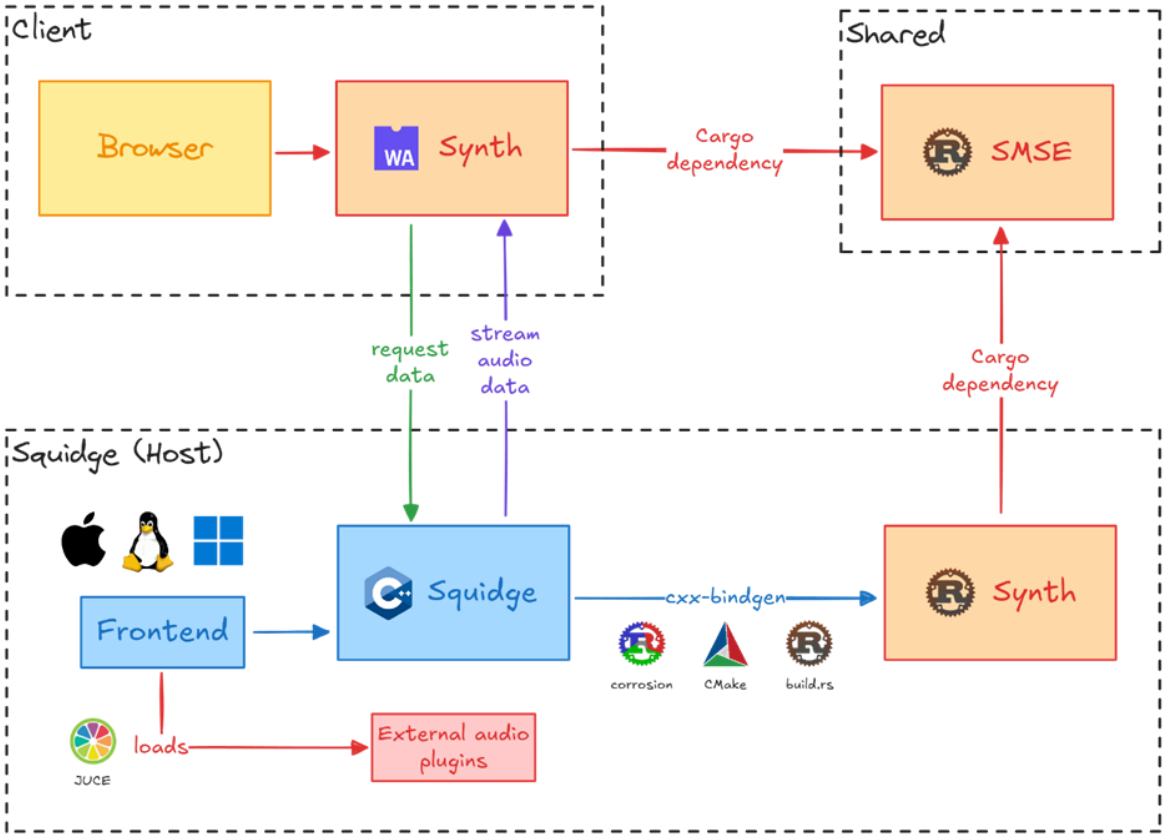
2. Overall Description

2.1 Product Perspective

Squavy is stand-alone software. It consists of a backend server that handles collaboration and a webserver. Jenkins is used in combination with NGINX, to build new git commits on the main branch as docker containers. The main frameworks of the website include SolidJS, WebAudioAPI (with WASM), WebMidiAPI, WebCryptoAPI and Socket IO.



The synthesizer architecture consists of a shared Rust crate containing all DSG-relevant code whereas the browser and the native bridge each implement their own small Rust crate for bridging support between Rust and the respective platform. External plugins are loaded via JUCE from the frontend which can be on either of the three most common operating systems.



2.2 Product Functions Summary

- **Create Patterns:** Patterns give access to the MIDI-editor. They are the fundamental building blocks of the application.
- **Create Tracks:** On track creation, the corresponding synthesizer is created too.
- **Modify Songs:** Edit an existing music composition within the DAW, making changes, improvements or adjustments to the music and its components.
- **Modify Patterns:** MIDI-editor opens and can be used to make modifications to the pattern.
- **Modify Tracks: Adjust** individual audio tracks within a music project.
- **Modify Synthesizer:** The Synthesizer can be adjusted in many ways. But it's important to notice that there are many presets that can be used and tweaked.
- **Create Session:** Establish a session for collaborative music creation and generating a unique session code that can be shared with other musicians to join the session.
- **Join Session:** Connect to a session for collaborative music creation. Sessions can be created by sending a notification over web sockets to the relay server. The server then creates a space for members to join and collaborate with each other.
- **Leave Session:** As soon as the user clicks the button, he leaves the space and gets prompted back to the home screen.
- **Create Song:** Create new songs (projects). This is done by clicking on a button on the homepage. The user is them immediately sent to the editor.
- **Play Song:** Play the entire composition within the DAW. By clicking the "Play" button, the software initiates playback.
- **Pause Song:** Temporarily halt the playback of their music project within the DAW.
- **Export Song:** Export an existing song in various file formats. Upon clicking the export button, the user selects a preferred file type, confirms the export, and receives a downloadable processed file in the chosen format.
- **Save Song:** When a user wants to save their work to either continue working on it later or sending the project to another person, they can do so by a button in the toolbar of the editor.

Independent of the actual web application, there is an **admin dashboard** to monitor network traffic of the relay server. This helps the administrator to make decisions based on several statistics. This dashboard is hosted on the same server because it needs to go hand in hand with the relay server.

2.3 User Classes and Characteristics

Role: User

Description: The User represents individuals who interact with Squavy to create, edit and produce music. This role is characterized by its unique feature of not requiring any form of registration or account creation, ensuring that all users have equal access and privileges. Users can be both professional music producers and hobbyists, having a wide range of expertise and musical interests. Since all users are considered equal and there is no distinction based on registration, the “User” role represents a broad and inclusive category of individuals who use Squavy for their creative and musical needs.

2.4 Operating Environment

Squavy is a web-based program, so it is most likely compatible with the most popular browsers; mainly chromium-based browsers, Firefox and Safari. The backend will run in a containerized docker environment, which enables easy scalability and isolation from the host system and other applications. Jenkins is used as the primary build server for deploying Squavy. During runtime, we track analytics and vitals of the deployed server instances using Prometheus hooks and ultimately display the data in a designated dashboard using Grafana.

2.5 Design and Implementation Constraints

Squavy is subject to certain constraints, primarily in the realms of language support and security features. Currently, due to resource limitations, Squavy is planned to exclusively offer support for the English language.

Backend server resources: Our current backend server for online collaboration has very limited resources and is often limited in capacity by running several docker containers at once, which reduces overall throughput.

Browser-based application: Since Squavy is designed to be a browser-based application, we face restrictions on certain features that are typically available in native applications. This includes limitations on how we can handle file uploads, access device hardware, and implement some performance optimizations.

Security Considerations: We must ensure that all data is encrypted and that the application follows best practices to protect user information. This focus on security may limit our ability to use certain technologies or even negatively impact our development velocity.

2.6 User Documentation

Squavy includes a basic guide within the editor itself to help new users get started. This guide features a sample project that allows users to explore the application's functionality hands-on. For more complex concepts and detailed information, users can refer to our comprehensive online guide, which covers various topics and advanced features of Squavy.

2.7 Assumptions and Dependencies

The following assumptions and dependencies are identified for the Squavy project. Failure to validate these assumptions or changes in any of these dependencies could lead to significant impacts on the project's requirements and overall functionality:

Real-time collaboration: The use of Socketioxide for online collaboration assumes that the underlying infrastructure can handle real-time WebSocket connections efficiently. Limitations in server capacity or network performance could negatively affect the user experience.

Frontend framework: The project relies on SolidJS for the frontend. It is assumed that this framework will remain actively maintained and that updates will not introduce breaking changes that impact the implementation of Squavy.

CI/CD: For continuous integration and deployment, the project depends on NGINX, Docker, and Jenkins. It is assumed that these tools will be properly configured and available throughout the development process. Any changes or disruptions in these tools may impact deployment cycles.

Browser compatibility: It is assumed that users will have access to modern web browsers (primarily Chrome, followed by Firefox and Safari). Any significant changes in browser policies or functionalities could affect how Squavy operates across different platforms.

Network availability: Squavy assumes that users will have stable internet connections to facilitate online collaboration. Variations in user network conditions could affect performance and user experience like disruptions or desyncs.

3. External Interface Requirements

3.1 User Interfaces

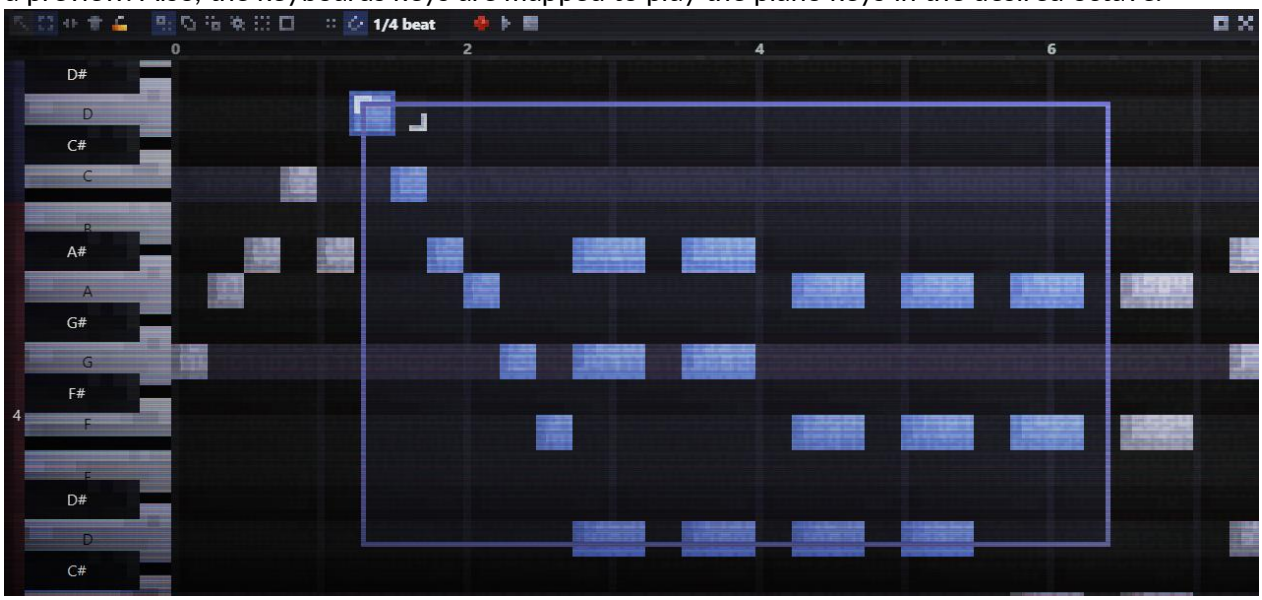
(IMPORTANT: All images are part of the current prototype and most definitely not final!!!)

The user is greeted by the home screen, where buttons can be used to create a new project or load an existing one (opened by the user by choosing a file from the local filesystem):

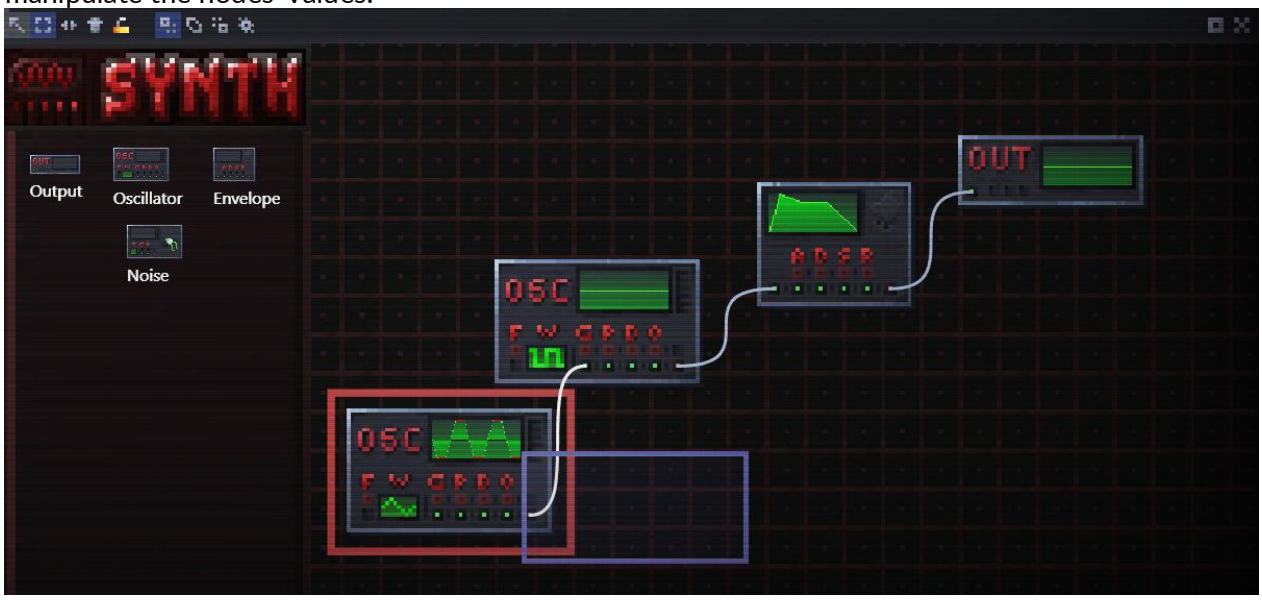
The following screenshot of Squavy shows its variety of editors to create songs in a fluent, accessible way, whilst maintaining that crunchy, playful look. At the center is the track editor, where patterns (on the far right) are placed in the song's timeline. At the bottom left is the pattern editor, where notes are placed into a reusable pattern. On the bottom right is the synth editor, where the audio nodes are arranged to an instrument with a unique sound.

Collaboration access: Start a session and enter a username below. To get others to join, one needs to give the provided link to them.

Pattern Editor: Here, one can place notes free of choice. This pattern can be placed in the timeline later. The example below shows a famous edm-chord-progression as an example. The user can edit the notes length, and placement. The user can select a tracks instrument too and play the pattern as a preview. Also, the keyboards keys are mapped to play the piano keys in the desired octave.



Synth Editor: Manipulates a tracks instrument. Here, the user can drag and drop audio-nodes and connect them in interesting ways to generate unique sounds. Also, knobs can be rotated to manipulate the nodes' values.



3.2 Hardware Interfaces

The hardware required to make this project possible consists of a Server, that has 4 docker containers running simultaneously. The user must only have a functioning computer, which has a modern browser of choice installed, as well as an internet connection. Also, some form of sound chip/card and Speakers/Headphones should be installed in said computer.

3.3 Software Interfaces

SMSE `v0.7.0 by Squavy`: Essential in-house created library to synthesize audio from scratch using individual audio nodes in a modular fashion. We created this library to replace the built-in audio synthesizer defined by W3C.

Socketioxide `v0.9.1 by Théodore Prévot`: Supercharged version of Socket.IO written in the Rust programming language that is built upon the Tokio-async framework. Used for the backend server implementation. Communication is performed over web-sockets.

Socket.IO-client `v4.7.5 by Guillermo Rauch`: The client sided library for communicating with the backend server. Communication is performed over web-sockets.

Jenkins `v2.426.2 by Kohsuke Kawaguchi`: Serves as our preferred platform for the building

and deployment of our product, featuring dedicated pipelines for both client and server components. In our case deployed on an isolated docker container.

Squavy will run on any modern web browser, ensuring broad accessibility for our users. The application is best supported on Google Chrome, providing optimal performance and functionality. Firefox is also compatible, though some features may perform better on Chrome. While Squavy can be accessed via Safari, users may encounter occasional limitations in functionality. An active internet connection is required for online collaboration, enabling real-time editing and communication among users.

3.4 External interfaces

Project files: An exported file of the project that can be opened later. The file is a JSON-based format that stores the entire in-memory model.

Internal data format: As the project being a web-app inside the browser, all data is handled and stored using JavaScript, the browser's internal scripting and data-persistence language.

Serialized command format: To communicate with Squavy in collaboration mode, we use web-socket messages between the client and the server. There, all packet-types with their individual fields and properties are explicitly defined for both parties in the languages JavaScript and Rust. For example, packets are defined like this, to ensure optimal type safety:

- `Packets::update_note: (args: {id: string, update: NoteStruct}) -> void`
- `Packets::request_project: (args: null) -> ProjectStruct`

Invalid packets are simply discarded by both the client and the server to ensure safety from external attacks through manipulation of the network traffic.

Source input and output: These web-socket messages are picked up by the native browser's web-socket implementation and passed through our custom multiplayer pipeline. On the server side, all network traffic including HTTP-requests are handled through *Axum*[2]. Messages are subsequently propagated through Socketioxide's custom pipeline for a better development experience.

Screen layout: Squavy is supposed to be used on a normal 16:9 horizontal screen for the best experience. Screen sizes from 720p up to 4k are recommended, with smaller screens unintentionally cutting off important parts of the web application. Vertical layouts are not considered supported, as major reworkings of the user interface would be required to guarantee a usable experience.

[2] All-purpose networking library written in Rust.

3.5 Communications Interfaces

HTTPS: (Hyper-Text-Transfer-Protocol-Secure) The browser is the interface between the computer and the human. The browser first connects to the webserver through HTTP, downloading a local installation of Squavy, just like an ordinary website.

WSS: (Web-Sockets-Secure) When collaborating with other users, computers and servers exchange information through Web Sockets with our backend server. The exchanged messages are wrapped in our own format to support encryption and a developer friendly API.

4. System Features

4.1 Seamless Collaboration Between Editors

4.1.1 Description and Priority

This feature enables real-time, seamless collaboration between multiple users editing a single project, allowing them to work together without disrupting each other's workflow. **Priority: High**.

4.1.2 Stimulus/Response Sequences

- **User Action:** A collaborator joins the editing session. **System Response:** The system synchronizes changes in real time, displaying edits made by each user without delays.
- **User Action:** A user makes an edit or adds a comment. **System Response:** The system immediately updates all other collaborators' views to reflect the change.

4.1.3 Functional Requirements

- **REQ-1:** The system must support multiple simultaneous users editing without overwriting each other's work.
- **REQ-2:** The system must display all collaborators' actions and comments in real time.
- **REQ-3:** The system should offer internal version control to handle potential conflicts.

4.2 Various Editors for Music Production

4.2.1 Description and Priority

Provides different editors, such as pattern editors, sample editors, and synthesizers, tailored to pattern-based music production. **Priority: High**.

4.2.2 Stimulus/Response Sequences

- **User Action:** User opens a pattern. **System Response:** The system opens a pattern editor interface optimized for note and timing adjustments.
- **User Action:** User switches to a sample editor. **System Response:** The system displays sample-specific tools for waveform editing.

4.2.3 Functional Requirements

- **REQ-1:** The system must offer a pattern editor with standard music composition tools.
- **REQ-2:** The system must include a sample editor with precise audio trimming and waveform manipulation tools.
- **REQ-3:** The system must provide a way to use patterns to assemble the final song.

4.3 Robust In-House Audio Synthesis

4.3.1 Description and Priority

A powerful audio synthesis engine is provided to generate high-quality sounds, reducing the need for external plugins, samples, or even real instruments. **Priority: Medium**.

4.3.2 Stimulus/Response Sequences

- **User Action:** User accesses the synthesis engine. **System Response:** The system opens the synthesis interface with controls for waveform, modulation, and effects.
- **User Action:** User modifies a parameter (e.g., frequency). **System Response:** The system instantly updates the sound output according to the new settings.

4.3.3 Functional Requirements

- **REQ-1:** The system must include a synthesis engine with waveform generation and modulation capabilities.
- **REQ-2:** The system should allow users to create custom sound patches and save them within the project.
- **REQ-3:** The system must provide real-time audio feedback as parameters are adjusted.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Network speed: The primary performance constraint lies in network speed. Each user can initiate a collaborative session at no cost, wherein they establish a socket connection with our server to engage with their colleagues. In the future a system could be designed to deploy secondary or multiple servers when the data load on the main server surpasses a predetermined threshold.

Frontend performance: The frontend of Squavy incorporates many expensive design options that should be optimal in performance, so that the users won't experience frame drops or decay in sound quality.

5.2 Safety Requirements

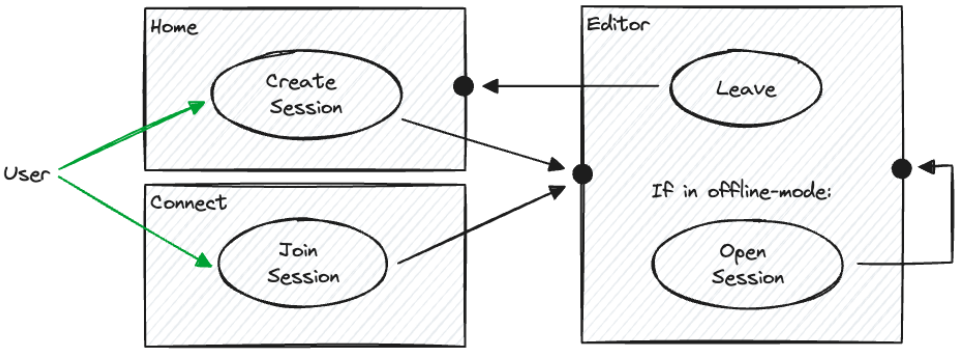
Data Loss Prevention: Squavy must include automatic data backup features to prevent loss of work in the event of a system failure or unexpected shutdown of the browser window. Users should have the option to restore previous versions of their projects.

User Error Protection: The product should include safeguards against irreversible actions. This includes confirmation prompts for critical actions and an undo feature for recent changes. Technical implementation details on how this will be handled during collaboration are yet to be decided.

5.3 Security Requirements

We take security seriously, especially when it comes to online collaboration. All data exchanged during collaborative sessions is encrypted to protect user information and project files.

The key feature of real-time collaboration is based on relayed peer-to-peer networking via **RFC 6455** (Web Sockets), **RFC 5694** (Peer-to-peer networking) and **RFC 8445** (Interactive Connectivity Establishment) to ensure a stable and high-performance connection between clients. Collaboration is kept relatively simple by making use of temporary session tokens that clients use to join the host. Sessions are created on the project screen and can be joined by others on the connect screen.



5.4 Software Quality Attributes

- **Usability:** The user interface follows an 80's retro-terminal approach with old-looking stylized effects. The components inside the application have pixel-art images that go hand in hand with the style we are looking for. This, however, may cause impairments with handicapped users, thus there is an option to turn them off, reverting the application to a cleaner and more readable version.
- **Reliability:** To use Squavy, a web browser with a working internet connection is required. On the service operations side, it is important to maintain the collaboration server in such a way that it runs at any given time and is able to withstand the network traffic of all users currently collaborating on Squavy at the same time.
- **Performance:** Realtime collaboration runs over a relay at a central server. This is enough for users in that region, but latency will be experienced at larger distances to that server. However, keeping a low latency is not necessary in order to collaborate in Squavy, as it is only noticeable when e.g., the cursor position of a user changes.
- **Security:** Collaboration on Squavy is cryptographically secure, as all network traffic will be encrypted on an end-to-end basis. When creating a session, the initial client generates a symmetric key, that's encoded into the invitation link. The receiving client uses this key to decrypt the network traffic of the socket.
- **Portability:** Projects are not synced via Squavy itself, but rather through the users own favourite choice of synchronization. This may be done through services like Dropbox, Google Drive or simply by using USB-Sticks. If the user has access to the project files, they can work on them anytime, anywhere, on whatever they want. Even if Squavy servers are offline, it is still possible to work on projects offline.
- **Maintainability:** Squavy is built upon the SolidJS framework which is a good option to keep a fast and maintainable codebase. The architecture should be split into small components that can be assembled to the developers' liking.

- **Scalability:** The server runs on a hosting service that allows for good scalability. Therefore, no problem in scalability should arise.
- **Code quality**: We prioritize writing clean, maintainable, and efficient code to ensure a robust and reliable product. Best practices are consistently applied, including adherence to coding standards, and partial rewrites when code gets outdated. By upholding high code quality, we aim to minimize bugs, and ultimately improve performance and make updates easier in the future.

Appendix A: Glossary

Term	Explanation
AES	Acronym for “Advanced Encryption Standard”. A mathematical function for symmetrically encrypting & decrypting data with a shared key. Used in Squavy to secure traffic between clients in online collaboration.
API	Acronym for "Application Programming Interface". Defines how different computer programs communicate with each other, like a designated language that is tailored to a specific need or requirement.
IT	Information Technology. Everything that has to do with generating, manipulating and processing digital data.
JS	A widely used programming language for building dynamic and interactive web applications. JavaScript is primarily executed in web browsers to enhance the functionality and behaviour of websites.
NAT	The term "NAT" stands for Network Address Translation. NAT is a technique used in computer networking to map private or local IP addresses to a public IP address. It is primarily employed to enable multiple devices within a private network to share a single public IP address for accessing resources on the Internet.
NGINX	Abbreviation for “Engine X” Application that allows for hosting of files and routing of different paths / endpoints on the server.
React	An open-source JavaScript library used for building user interfaces or UI components, particularly for single-page applications where smooth and efficient user interactions are crucial. It was developed and is maintained by Facebook.
VST	Acronym for “Virtual Studio Technology”. A universal format for digital sound synthesis.
Jenkins	A server application that allows for automation of several tasks that would otherwise have to be done manually. Used by Squavy to automatically update the published application when new changes are pushed to the development server.

Oscillator	An electronic circuit or software module that generates a continuous waveform. In the context of a synthesizer, oscillators produce the basic sound source for creating musical tones.
Pattern	Either a piano roll sequence or a sample. In a pattern-based DAW like Squavy, patterns are placed on a track at a specific point in the timeline.
Piano Roll	A graphical representation of musical notes on a timeline. It allows users to create and edit melodies, chords, and other musical elements by placing and manipulating note events. Also known as MIDI-editor.
Sample	Pre-recorded audio that can be used throughout the song.
Synthesizer	Musical instrument that consists of one or more oscillators.
Track	Tracks consist of many patterns. Patterns are placed onto the track at a specific point in time.
Webapp	An online application that is used and displayed by the browser and served by the server.
DAW	Acronym for “Digital Audio Workstation”. A term that describes a digital platform to create, edit and manage musical arrangements.

Appendix B: Analysis Models

Classes and Modules:

All visible elements on the page are functional SolidJS components that have improved performance over class components. All in-memory values are stored in Solid contexts, so they are accessible throughout the components. The logic behind the scenes happens in standalone "engines" which have very little to nothing to do with Solid itself. These include the synthesizer, multiplayer framework, and the playback engine. The components are then able to interact with the business logic of these systems.

Synthesizer: The synthesizer engine is a separate module called SMSE[3] that can be called from the main application. The standard usage would be something like:

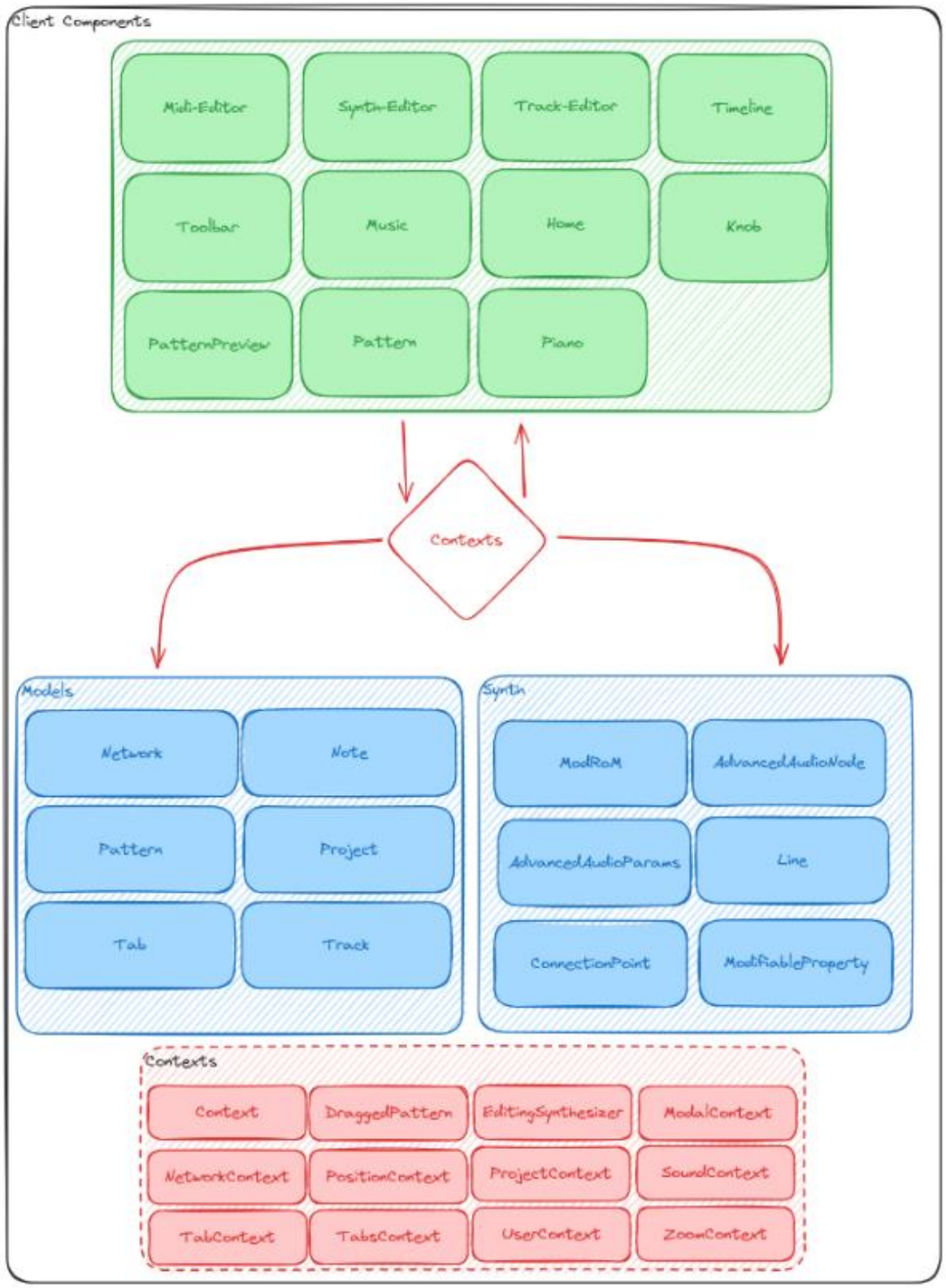
```
import Synthesizer, { Waveform } from "smse";

const synth = new Synthesizer();

// the following code needs to run after a user gesture
synth.initialize();
synth.addInstrument("example instrument");

synth.createOscillator("example oscillator", Waveform.EighthPulse, 0.5, 0.0, 0.0, 0.0);
synth.connectRootNode("example oscillator", "example instrument");

// the following code can run at any time, for example on a button click
synth.previewNote(id, "example instrument", 0.5, 440, 1);
```

Multiplayer API: Semi-standalone API for easier multiplayer services. Supports methods for connecting to sessions, sending broadcast events with optional acknowledgements and requests to specific clients within the session. All traffic is encrypted through the browser's AES encryption standard and is not visible to the server as it simply relays the ciphertext. Methods include:

- `CryptoSocket.broadcast(data:any, ack?:Fn<any[]>);`
- `CryptoSocket.request(data:any) -> any;`

Additionally, the crypto-socket class contains fields that are initialized when creating the instance such as the actual cryptographical key to encrypt / decrypt data and an event map with callbacks that is queried whenever a message is received from another client.

[3] Squavy Modular Synthetization Engine is a browser library developed and maintained by the Squavy team. Source: https://github.com/Squavy-DAW/Synth


Client Components
Midi-Editor
Synth-Editor
Track-Editor
Timeline
Toolbar
Music
Home
Knob
PatternPreview
Pattern
Piano
Contexts
Models
Network
Note
Pattern
Project
Tab
Track
Synth
ModRoM
AdvancedAudioNode
AdvancedAudioParams
Line
ConnectionPoint
ModifiableProperty
Contexts
Context
DraggedPattern
EditingSynthesizer
ModalContext
NetworkContext
PositionContext
ProjectContext
SoundContext
TabContext
TabsContext
UserContext
ZoomContext