Todo list

Bei Abgabe: Anweisung nocite in Bachelorarbeit.tex entfernen

Bei Abgabe: In Bachelorarbeit.tex und Expose.tex Dokumentenoption overfullrule entfernen und die Option final eintragen

Bei Abgabe: In Packages.tex beim Package todonotes die Option disable eintragen, um Todos zu deaktivieren

Bei Abgabe: In Packages.tex beim TODO die größte Seitennummer eintragen

Wenn eine Version der Arbeit erstellt wird, die gedruckt werden soll in Packages.tex beim Package hyperref die Option urlcolor=blue entfernen

TECHNIK
HOCHSCHULE MAINZ
UNIVERSITY OF
APPLIED SCIENCES

Masterarbeit

zur Erlangung des akademischen Grades Master of Engineering
im Studiengang Geoinformatik und Vermessung

Compostable Software: A study on task-specific telepresence applications for contemporary dance

Hochschule Mainz
Fachbereich Technik

Vorgelegt von:	Vorname Nachname
	Straße
	PLZ Ort
	Matrikel-Nr. 123456
Betreut von:	Prof. Dr. Betreuer
Eingereicht am:	01.01.1970

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit

> **Compostable Software: A study on task-specific telepresence applications for contemporary dance**

selbstständig und ohne fremde Hilfe angefertigt habe. Ich habe dabei nur die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt.

Zudem versichere ich, dass ich weder diese noch inhaltlich verwandte Arbeiten als Prüfungsleistung in anderen Fächern eingereicht habe oder einreichen werde.

Mainz, den 01.01.1970 Vorname Nachname

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit.

Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

List of figures

List of tables

List of abbreviations

2D 2-dimensional

3D 3-dimensional

AJAX asynchronous JavaScript and XML

API application programming interface

BVH Biovision hierarchy

CLI command-line interface

CPU central processing unit

CNCF Cloud Native Computing Foundation

CRUD create, retrieve, update and delete

CSS Cascading Style Sheets

DIY do-it-yourself

ES6 ECMAScript 6

GATT Generic Attribute Profile

GIS geographic information system

HRTF head-related transfer function

HTML HyperText markup language

HTTP HyperText transmission protocol

I/O input/output

I2C Inter-Integrated Circuit

IETF Internet Engineering Task Force

IMU inertial measurement unit

JS JavaScript

JSON JavaScript Object Notation

JSX JavaScript XML

MCU multipoint control unit

ML machine learning

NoSQL Not-only SQL

NPM Node Package Manager

OCI Open Container Initiative

OOP object-oriented programming

OS operating system

P2P peer-to-peer

PWA progressive web application

RFC request for comments

RTC real-time communication

SATC Software Assurance Technology Center

SDK software development kit

SFU selective forwarding unit

SOFA spatially oriented format for acoustics

SPA single-page application

SQL structured query language

TCP transfer control protocol

TS TypeScript

UDP user datagram protocol

UI user interface

VR virtual reality

W3C World Wide Web Consortium

WebRTC web real-time communication

WebXR web mixed reality

WHATWG Web Hypertext Application Technology Working Group

XR mixed reality

List of listings

1. Introduction

1.1. Background

In recent years, remote collaboration has become increasingly important due to the rise of broader digitalisation strategies, especially since the pandemic. As a result, teleconferencing and telepresence platforms have become more pervasive in many work environments. These platforms allow people to work together remotely in real-time, usually focusing on streaming video and audio, document sharing or collaborative whiteboarding. While this is fine for most use cases and desktop-based workplaces, it lacks certain immersive qualities required for practices such as contemporary dance, where people relate to a shared space and physical presence. This became apparent in March 2020, when dancers could no longer rehearse and work together due to the lockdown. Despite this, there were attempts at using Zoom to stream and record collaborative rehearsals or dance classes. Still, these were confined to a screen-centric interface and limited to audio and video.

While commercial conferencing tools such as Zoom, Google Meet, and Microsoft Teams dominate in popularity among conferencing applications (Brandl, 2023), there are both free and open-source variants such as JitsiMeet and BigBlueButton. However, these all focus on the most basic form of screen-based conferencing mentioned above. While there are various domain-specific solutions for specialised applications, mainly in telemedicine (as well as general industry and the military), that support more immersive and task-specific remote collaboration, these are either unaffordable or unavailable to the general public.

As support for web standards is driven by key industry players (Davis et al., 2023), and with it the availability of a wide range of basic functionality, as well as access to display and sensor technology for deploying applications on desktop and mobile devices, there is an increased potential for smaller and more specific applications to be built and deployed with relative ease. This opens up new possibilities for niche cases of remote collaboration, such as dance, where collaborative functionality needs to be extended from the traditional paradigm of remotely viewing video streams to the creation of shared virtual environments that facilitate a sense of 'being there' together, if only at an abstract level (Skarbez et al., 2017).

The standard for real-time communication (RTC) in Browsers or web real-time communication (WebRTC) ('WebRTC: Real-Time Communication in Browsers', 2023) was first proposed by Google in 2011 and became an official World Wide Web Consortium (W3C) standard in 2021 (Couriol, 2021). It is already being used in a wide range of applications, such as some of the conferencing tools mentioned above, media streaming servers such as Wowza or Ant, or real-time frameworks and servers such as Mediasoup, Janus or LiveKit. In its most basic form, WebRTC establishes peer-to-peer connections between different devices, allowing low-latency exchange of media streams and arbitrary messages on so-called data channels.

1.2. Proposal

I propose a feasibility study for a reference implementation of a domain-specific telepresence platform based entirely on web standards and open-source components. The platform allows streaming of sensor data in addition to audio and video. This means that remote collaborators can share a wide range of data from multiple sources, such as motion capture, wearables and more. As the data flows through the WebRTC data channels alongside the usual video and audio streams, it can be analysed and rendered as required for a specific task on the client device.

A reference implementation will demonstrate the platform's capabilities to simulate essential aspects of presence in a shared virtual environment (Skarbez et al., 2017). The software implementation, written entirely in JavaScript (JS), should rely on existing open-source libraries and frameworks as much as possible and add as little custom code as necessary. Only the minimal viable product will be built to observe its basic functionality, leaving issues of robustness, interface design, security and scalability outside the scope of this study.

For dancers to communicate while moving in remote rehearsal spaces, a video feed is less critical as they need to be free from looking at a screen or typing on a keyboard. A virtual or augmented reality headset is also ineffective because it hinders vision and movement in the physical space. Here, spatial sonification seems the optimal alternative to creating a virtual environment without interfering with the dancers' ability to move freely. The sound generation code should be kept as minimal as possible.

For practical implementation testing, two remote rehearsal spaces of similar size will need to be set up with a motion capture system (TDB) that generates spatial data from the dancers' movements. Both rooms will be within the university network to provide a known and controllable quality of service. The two dancers will use the system in each location. The interaction will be via wireless in-ear sports headsets, commonly used for exercise and providing basic verbal communication via WebRTC's audio channels.

To interact with the spatial dimension of a shared virtual environment consisting of both rehearsal rooms, sonification is used as a means of orientation in the virtual space. The Web Audio application programming interface (API) provides functionalities for generating, mixing and positioning sound sources in spatial audio ('Web Audio API', 2021), which will be used to render each participant's microphone sound alongside their distinct signature sound (one percussive and the other sustained) at their tracked position in space, modulated by two basic movement qualities, the contraction index and the quantity of movement, based on an existing proposal for qualitative movement analysis (Volpe, 2003) and derived from the motion capture data. Each participant can only hear the other to avoid confusion. The spatial audio representation should provide a basic sense of position, while the sound characteristic should indicate the type of movement being performed. In addition, verbal cues can be exchanged.

A separate client implementation uses the same data to visualise the dancers' motion capture data as basic 3-dimensional (3D) images. It adds the combined sonification, which can then be viewed by a third party using the mixed reality (XR) Device API for the web ('WebXR Device API', 2023), allowing viewers to evaluate aesthetic aspects of the combined interaction result. In addition, a recording of the session's streams can later be replayed and reviewed by the dancers using this client implementation.

2. Conceptual foundations

2.1. Telepresence

The term *telepresence* first appears in an article by Marvin Minsky in which the author roughly defines it as a form of remote robotic operation, that ‘emphasises the importance of high-quality sensory feedback’ and posits that its development’s biggest challenge is ‘achieving that sense of “being there.”’ (Minsky, 1980). Minsky argued from a standpoint concerned primarily with robotic manipulators that performed labour either mediated over a distance or enhanced the operator’s abilities and safety.

Today, virtual and augmented reality, telepresence, and general presence research present much more diverse application scenarios. While there are applications of remote robotic control in retail, industry, telemedicine and police or military, the most common instance has become the teleconferencing application relaying video and audio streams and allowing chat and collaborative whiteboards. In this study, the term telepresence is used to explicitly describe a virtual or augmented environment that allows multiple people to experience some form of shared presence, immersion and interaction.

The concepts of *presence* and *immersion* require a precise contextualisation for this study. The article ‘A Survey of Presence and Related Concepts’ (Skarbez et al., 2017) defines a wide range of possible variations for these concepts and related ones in different contexts and environments. There, the authors state that presence ‘is most commonly defined as something akin to the feeling of “being there” in a virtual place’ (Skarbez et al., 2017, p. 2) and immersion can be understood as ‘an objective characteristic of a [virtual environment] system' that, as the authors are citing Slater, 'provides the boundaries within which [presence] can occur' (Skarbez et al., 2017, p. 3). The goal here is not to transport the participants to a different virtual place but rather to make them experience the virtual presence of the other in the physical location they are in. Furthermore, there is no immersion in the usual sense as the participants can remain aware of the divide between the physical and the virtual, much like in a telephone conversation.

However, the study presents various definitions for mediated interaction in virtual environments and distinguish '*copresence* as the sense of being together with another or others, and *social presence* as the moment-by-moment awareness of the copresence of another sentient being accompanied by a sense of engagement with them.' (Skarbez et al., 2017, p. 4). These terms are further distinguished as *Social Presence Illusion* referring 'to the feeling of social presence engendered by characters in virtual or mediated environments' (Skarbez et al., 2017, p. 4) and *Copresence Illusion* as referring 'to the feeling of "being together" in a virtual or mediated space' (Skarbez et al., 2017, p. 5). However, the authors also note that the experience of these forms of presence do not necessarily require the environment to be virtual, as can be experienced in a telephone conversation, where 'you are certainly aware of the person on the other end of the line (Copresence Illusion), and you can interact with that other person (Social Presence Illusion). However, you do not get the impression that you have been transported to another place.' (Skarbez et al., 2017, p. 5) Both definitions lend themselves to describe this study's use-case in as it attempts to establish a virtual copresence in the local phsyical space by means of a sonic avatar and provides a mediated form of interaction through verbal communication to establish an illusion of social presence.

As the notion of immersion also rather refers to a framework within a virtual environment, a conceptual definition is required for the relation between the user and the application environment constructed in this use-case. Again, the survey presents two concepts that are of interest for this study. The terms '*Involvement and Engagement*' (Skarbez et al., 2017, p. 8) are introduced as mostly relating to the same concept in games and virtual environments alike (Skarbez et al., 2017, p. 8). They are roughly defined as 'a state of focused attention or interest' (Skarbez et al., 2017, p. 8), and, citing Witmer and Singer, 'a psychological state experienced as a consequence of focusing one's energy and attention on a coherent set of stimuli or meaningfully related activities and events.' (Skarbez et al., 2017, p. 8) A third related concept is that of *flow*, defined as, citing Csikszentmihalyi, 'an optimal state of concentration, "the state in which individuals are so involved in an activity that nothing else seems to matter".' (Skarbez et al., 2017, p. 9) Furthermore, they cite Brockmeyer et al. in 'argu[ing] that flow, since it involves experiencing an altered state, may be a deeper state of engagement with media than presence' (Skarbez et al., 2017, p. 9). These alternative concepts to the idea of immersion shaping the experience within a virtual or mediated environment are a good basis for the design processes of the modes of expression and interaction within the telepresence application, as a deep involvement or flow would be beneficial to shape a feeling of 'togetherness' in the shared task of creating sound and music.

2.2. Motion capture

The positional tracking of specific key points on a moving body over time is commonly referred to as *motion capture*. The technique is often used in **CGI!** (**CGI!**), enabling puppeteering of 3D avatars for motion picture productions and game character animation. High accuracy is required for these purposes, and the technological and financial entry barriers are relatively high. These applications use systems by Vicon[1] or OptiTrack[2], which use visual markers to track movement in space and require a studio environment to be deployed. Another markerless optical system is Captury Live[3], which tracks humanoid moving actors with a 360° camera setup. In the performance field, the preferred methods are IMU-based tracking systems like the SmartSuit by Rokoko[4] or

[1]https://www.vicon.com
[2]https://www.optitrack.com
[3]https://captury.com
[4]https://www.rokoko.com

the Perception Neuron[5] sensor kit since they are independent of the lighting conditions. Both visual and inertial methods are available in variants from different manufacturers on a cost spectrum that varies from the low thousands to hundreds of thousands of euros in investment.

The ‘grassroots’ setup for motion capture is the Kinect, developed by Microsoft in 2010, featuring an infrared time-of-flight measurement system that produces a depth image from which a 3D pose can then be extracted using 3D pose estimation (see Ye et al., 2011). The Kinect was frequently used among creative coders, although it was initially developed for games. In 2024, the Kinect, now called Azure Kinect, is supposed to be officially discontinued. However, other low-cost 3D cameras are on the market, like the Oak-D[6] with an integrated processing engine or the Orbbec Femto Bolt[7]. These systems produce rather sub-par accuracy but can be used to analyse more general dynamics in the movement data.

Deep learning models for motion capture like PoseNet (Kendall et al., 2016) or BlazePose (Bazarevsky et al., 2020) have also become available and, while primarily used on 2D (surveillance) footage, can be extended into 3D if combined with the proper calibration data (e.g. depth images). These models are fast and can be run on a regular webcam, but they also tend to produce relatively coarse movement data.

2.3. Movement data sonification

The term ‘sonification’ primarily refers to the auditory expression of data. Some well-known examples include the beeping sounds used for car parking assistance, the electrocardiogram machines used in hospitals to relay the heart rate acoustically, or the Geiger counter, sonifying ionisation to indicate the level of radioactivity in the environment.

---

[5] https://neuronmocap.com
[6] https://shop.luxonis.com/collections/oak-cameras-1
[7] https://www.orbbec.com/products/tof-camera/femto-bolt/

While the technique of expressing quantities acoustically can be traced back as far as 3500 BCE (Worrall, 2018, p. 178), the method of ‘parameter mapping sonification’ (Hermann et al., 2011, Chapter 15), which is most commonly used today and is of particular interest here. Its conceptual foundation is grounded on Western music becoming more abstract and focused on gestural expression and formalised rules during the eighteenth and nineteenth centuries, but especially with the emergence of serial music in the twentieth century (Worrall, 2018, pp. 179–180) and algorithmic composition as its descendent, popularised by composers such as Iannis Xenakis and John Cage.

The sonification of human movement data using parameter mapping is often used in health and therapeutic research to offer an acoustic interface to experience dynamics in movement properties. Examples are movement perception in rehabilitation and active movement practice as part of learning exercises (see Brock et al., 2012). It requires specific data points to be tied to acoustic properties. This can be a direct value connection from one property to another (e.g. velocity to loudness, altitude to pitch). However, it can also be achieved using indirect logical constraints expressed in more complex algorithms (e.g. if multiple thresholds are crossed, a single signal is triggered).

Today, the many possibilities for real-time data analysis combined with digital sound synthesis enable a broad spectrum of practical and artistic applications of movement sonification. Combined with the various means of motion capture available today, it can be used on stage to generate a real-time soundtrack to the movement and offline for recording, analysis and composition of music in a reciprocal process between dancers and composers. Here, the particular data expression is left open, and the focus lies on providing a framework for extracting movement qualities, transmitting these qualities, and generating events based on simple rules. The forms of concrete artistic expression made possible by this functional infrastructure are beyond the scope of this study.

Three basic movement qualities are to be extracted to provide a reference pipeline for evaluating the basic mechanics of this sonification approach. These are the ‘average velocity’, generally defined as the distance travelled over time (e.g. m/s), and two more specialised concepts developed for expressive movement analysis. These are 'Quantity of Movement' (Volpe, 2003, pp. 96–97), describing the amount of difference between poses over time, and the 'Contraction Index' (Volpe, 2003, p. 97) which observes the density of space used by a pose. While the former is easily defined as the velocity calculated for a singular point (the centre of mass with an equal distribution), the latter two have been initially developed to observe pixelated 2-dimensional (2D) camera images and need to be translated to 3D space.

2.4. Embedded computing and open-source hardware

The concept of an embedded system is defined as 'a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, ...' (Barr, 2015)

While this definition applies to most contemporary electronics, it rose to broader awareness through its popularity in the do-it-yourself (DIY) electronics community. In 2003, Hernando Barragán, a student at the Interaction Design Institute Ivrea (IDII) in Italy, created the Wiring project as his Master's Thesis, aiming 'to make it easy for artists and designers to work with electronics, by abstracting away the often complicated details of electronics so they can focus on their own objectives.' (Barragán, 2022) The Wiring project, after successful use in the curriculum at IDII, went on to become the basis for the Arduino project, launched in 2005 by Massimo Banzi and David Mellis as a fork of Wiring and without Barragán's involvement (Barragán, 2022). The Arduino development board line and its software ecosystem became the most popular framework for experimenting with open-source hardware and microcontrollers outside of the field of electronic engineering. At the same time, there are other successful projects like Adafruit Industries, SparkFun, RaspberryPI and more.

2.5. Web standards

The idea behind web standards is to provide stable definitions of core technologies that are used to build and present web content. Apart from providing a consistent display across different browsers, this is especially important for interacting with particular operating system (OS) or hardware functionality via the browser. As JS does not define any specific input/output (I/O) functionality, it is the task of the browser environment to supply this. As the browser is the mediator between the OS and the web page, the idea of standardised APIs was devised and implemented. Several organisations standardise web technologies, with the most prominent of them being the W3C, Web Hypertext Application Technology Working Group (WHATWG), Ecma, Khronos and the Internet Engineering Task Force (IETF).

2.5.1. WebRTC

The standard for real-time communication in browsers was initially proposed and mainly developed by Google. It became an official standard in 2021 (Couriol, 2021). It provides functionality for transmitting video and audio streams over user datagram protocol (UDP) or transfer control protocol (TCP). Additionally, data streams with arbitrary message packets can be used to transmit binary encoded or text data. WebRTC handles all low-level flow control or other transmission aspects. It can be used in direct peer-to-peer (P2P) setups where each party communicates with the others directly, a multipoint control unit (MCU) that receives all communication centrally and then broadcasts a composite signal to everyone, but also as a selective forwarding unit (SFU), relaying only the requested streams to participants and enabling one-to-many or many-to-many communication setups. The choice between the various setups has implications for scalability, infrastructure cost, privacy and security aspects (Iyengar, 2021). Several software solutions for streaming media also support the WebRTC standard. However, in this case, focusing on the concept of a SFU is essential, so the selection is reduced to the packages that support or explicitly focus on this type of topology.

Table 2.1.: WebRTC servers ranked by stars received on GitHub

WebRTC Server	Stars (k)
Janus Gateway, 2014	7.6
LiveKit, 2020	6.4
Mediasoup, 2014	5.7

The *Janus Gateway*[8] server is designed as a general-purpose solution, providing only the core WebRTC functionality and allowing developers to extend it using existing or custom-made plugins. This way, it can implement various schemes, such as P2P, MCU and SFU, but can also be used to create completely custom hybrids.

LiveKit[9] is a dedicated SFU server including software development kit (SDK)s for web, native mobile and desktop and server applications in various languages. It is developed and maintained by a relatively young company, as it was publicly released in 2021 and was 'started amid and in response to the pandemic' with the idea of providing 'free and open infrastructure capable of connecting anyone' (LiveKit, n.d.). While the software is free and open-source, a paid hosted service is also offered for those who want to experiment with real-time communication but don't want to set up an infrastructure. There are many examples of integration into existing frameworks, extensions for recording sessions on the server, as well as extended handling of streams.

Mediasoup[10] is different from the other options in that it does not present a standalone server architecture. It provides a versatile collection of Node, Rust and C++ libraries that allow for building a custom server application from the ground up. While it takes care of the low-level RTC functionality, it provides somewhat granular building blocks

[8] https://janus.conf.meetecho.com
[9] https://livekit.io
[10] https://mediasoup.org

to set up the actual implementation. This allows for building entirely decentralised peer-to-peer applications as well as server-centric setups. It was developed by a small team of contributors around its leading developers, Iñaki Baz Castillo and José Luis Millán.

2.5.2. WebSockets

The transmission protocol *WebSockets*, which was standardised as request for comments (RFC) 6455 by the IETF in 2011 (Fette & Melnikov, 2011), allows full-duplex communication between client and server, running on the same ports and transport layer seven as the half-duplex HyperText transmission protocol (HTTP) protocol, thus being compatible with existing web infrastructure. As the *WebSockets* standard was not fully supported across browsers for some time, there have been various approaches to providing real-time functionality to web applications more or less loosely based on the *WebSockets* specification. However, the current browser landscape shows much more complete support for the original *WebSockets* protocol (Deveria, 2024).

Table 2.2.: JavaScript WebSockets libraries

Framework	Stars on GitHub (k)
Socket.IO, 2010	59.7
ws, 2011	20.7
µWebSockets, 2016	16.4

Socket.IO[11] is billed as a 'realtime [sic] application framework' (Socket.IO, 2010) and provides client and server implementation. While it is popular for real-time communication in the browser, it implements its own protocol instead of building on the WebSockets standard. As stated in the documentation, 'a WebSocket client will not be able to successfully connect to a *Socket.IO* server, and a *Socket.IO* client will not be able to connect to a WebSocket server' (Socket.IO, 2010).

[11] https://socket.io

WS[12] is a standards-compliant implementation of the WebSockets protocol for use in server-side applications written in Node.js. It is written in C++ to provide good performance, and it supports compression via implementation of the standards proposal in RFC RFC 7692 'Compression Extensions for WebSocket [sic]' (Yoshino, 2015).

μWebSockets[13] is focused on robustness and performance while exclusively communicating via standards-compliant WebSockets protocol. Like WS, it is implemented in C++, used via Node.js on the server side, and does not require a specific client library.

2.5.3. WebAudio

WebAudio, the standard for handling audio in the browser, takes care of basic mixing of channels and different sources (e.g. media streams, audio files). It can also be used for generating sound via several synthesis nodes. Another feature commonly used in games or virtual reality experiences is the possibility of placing sound sources on virtual soundstages that are then rendered as ambisonics for psychoacoustics in headphones. Several frameworks provide a high-level abstraction to the WebAudio API and thus enable a speedier development process. While the selections of frameworks presented for selection are supposed to be the top three entries based on GitHub Stars, this list adds the relatively new framework *Elementary*, which takes a different approach to development using the declarative definition of sound structures. It has roughly half the rating of *Flocking* but has been around only since 2022 (2.3).

Howler.js[14] is a complete audio framework that builds on the *WebAudio* API and provides easy access to audio functionality, focusing primarily on interactive audio for web apps or games. It offers various modes of sound playback and mixing as well as spatial audio

[12] https://github.com/websockets/ws
[13] https://github.com/uNetworking/uWebSockets
[14] https://howlerjs.com

Table 2.3.: Popular JavaScript audio frameworks

Framework	Stars on GitHub (k)
Howler.js, 2013	22.6
Tone.js, 2014	12.9
Flocking, 2011	0.7
Elementary, 2023	0.3

as a plugin, but at the time of writing, it only supports connecting live audio sources through a yet unmerged pull request on GitHub (rafern, 2022).

Tone.js[15] is explicitly focused on musical application, working much like a digital audio workstation software, providing various modes of sound synthesis, as well as transport controls, meter and scales. It supports spatialisation using a 3D panner node and, while not explicitly documented, should support external audio stream input through its 'UserMedia' node.

Flocking[16] is more of an outsider, being around since 2011, but having gathered only a small amount of star ratings, but has a unique approach in that it defines sound objects using JavaScript Object Notation (JSON), making them easily portable and allowing for generative approaches to sound generation on a meta-level. According to its developer, 'its goal is to promote a uniquely community-minded approach to instrument design and composition.' (Flocking, 2011) Unfortunately, it currently does not support parallelising audio rendering in special workers. Thus, 'Flocking is not currently well-suited to applications that involve a lot of graphics rendering or user interaction.' (Flocking, 2011)

Another audio framework following a declarative and functional approach is *Elementary*[17], which has only been around since early 2023. It separates the declarative API for creating

[15] https://tonejs.github.io
[16] https://flockingjs.org
[17] https://www.elementary.audio

instruments and musical structures from the sound rendering, allowing it to use the Web Audio API for real-time audio in the browser and an offline renderer for Node.js. The framework offers extendability through native nodes developed in C++ and used in the Node.js environment.

Resonance[18] is a library worth mentioning, as it focuses exclusively on spatial audio. Google developed it based on Omnitone, another one of their projects focusing on ambisonic spatial audio rendering. It received only one release and seems to have remained dormant since then, but it still works without breaking changes. It uses a default head-related transfer function (HRTF) to model audio spatialisation. It allows a virtual room to be created with different materials for walls, floors, and ceilings that provide different reflection types. It also offers custom sources to be defined, connected to web audio nodes, and positioned around the virtual space. It hooks into any existing audio context, thus allowing a combination with virtually any other WebAudio-compliant audio framework.

2.5.4. WebXR

The various virtual and augmented reality devices available are accessible via the WebXR API, and the browser manages the communication with the headset. A 3D scene created in a web-based graphics framework like THREE.js or A-Frame can be instantly experienced on a virtual reality (VR) headset like the HTC Vive or Oculus Quest.

Three.js[19] is a 3D graphics framework with a large community with over 1800 contributors that has been around since 2010. It was initially developed for the ActionScript language used in Macromedia and later Adobe Flash (another Ecma standard-compliant language, see **??**). It features an extensive toolset for graphics generation, rendering and effects

[18] https://resonance-audio.github.io/resonance-audio/
[19] https://threejs.org

Table 2.4.: JavaScript 3D frameworks

Framework	Stars on GitHub (k)
three.js, 2010	97.1
Babylon.js, 2013	22
A-Frame, 2015	16

and relies on the WebGL standard to allow performant rendering via local graphics hardware.

Babylon.js[20] is another fully-fledged 3D framework with a strong focus on games and realistic, high-quality rendering. It was initially developed by Microsoft employees in 2013. The framework features an extensive collection of tools for interaction and animation. Also, it supports integrating the web mixed reality (WebXR) standard to use VR equipment in the browser.

A-frame[21] is a framework that allows the developer to create 3D scenes by composing custom HTML elements that provide geometric primitives, lights, cameras, etc. This way, it has a comparably low entry barrier for people with limited scripting experience. It explicitly focuses on mixed reality applications, implements the WebXR standard and has preset control objects for various headsets and controllers. It is based on Three.js, whose full functionality can optionally be accessed through A-Frame if more custom behaviour is required.

2.5.5. WebBluetooth

This relatively simple API provides access to the computer's Bluetooth functionality. It allows connecting custom Bluetooth senders like Arduinos or other embedded devices

[20] https://www.babylonjs.com
[21] https://aframe.io

with sensors or other DIY electronics, sending and receiving messages to and from the browser (see: section 2.4).

2.6. Programming languages

Table 2.5.: Ranking among the most used languages on GitHub (Daigle & GitHub, 2023)

Language	Rank
JavaScript	1
Python	2
TypeScript	3
C++	6

JS is a scripting language created by Brendan Eich in 1995 as part of the release of the Netscape 2 browser (Netscape, 1995), then was officially standardised by the Swiss standards body Ecma in 1997 as ECMA-262 or ECMAScript, as it is known today. This standard later was the basis for JScript by Microsoft and ActionScript as part of Macromedia Flash. The version currently being supported by all browsers (except Internet Explorer 11) is ECMAScript 6 (ES6) (W3Schools, n.d.). JS is an object-oriented, weakly-typed programming language that applies multiple programming paradigms. It is primarily used in the browser to add extra functionality to web pages. The underlying ECMAScript standard does not define any input or output methods, which means that this is provided by the specific environment it is being used in (e.g. desktop or mobile browsers).

TypeScript (TS) was released by Microsoft in 2012 'to accommodate an increasing number of developers who are interested in using JavaScript to build large-scale Web applications to run in a browser rather than on the desktop.' (Jackson, 2012) It complies with the underlying Ecma scripting standard and is designed as a superset of JS, adding static typing. It uses a compiler to generate regular JS code.

Node was initially released by developer Ryan Dahl in 2009 as a server-side JS environment. Node runs standard ECMAScript in Google's V8 engine, allowing multithreading and native code integration. Its development was sponsored by the company Joyent after some dissatisfaction in the community about Joyent's stewardship and a fork of Node called io.js. These differences were eventually resolved, and everything was merged under the umbrella of the OpenJS Foundation. Node uses the Node Package Manager (NPM) to package code as modules, which can be used as dependencies. These modules can also integrate native C++ code, enabling bindings to most open-source libraries in the Linux ecosystem. It can be used to develop APIs or other server-side applications and support local web development processes like preprocessing, packaging, and deployment.

Guido van Rossum created the *Python* programming language in 1990 (Python Software Foundation, n.d.). It is a multi-paradigm language that is both dynamically and strongly typed (van Rossum, 2008). It relies heavily on indentation and whitespace to structure the code. The language uses a standard library, and the surrounding ecosystem of available modules and applications based on Python makes it a good choice for data processing and science. There is a native code interface that allows extending Python with bindings to native code, like with Node.

C++ originated as an extension to C in 1985. It is a multi-paradigm, statically typed and object-oriented programming language. It is used to develop code for embedded open-source hardware platforms (see: section 2.4), extend both Node and Python and, more generally, provides direct interaction with the operating system and its APIs. While it is the oldest of the programming languages mentioned here, it has remained essential to the open-source world, not least because of its prevalence in the Linux ecosystem.

2.7. Application design paradigms

The idea of the *single-page application (SPA)* originated around the beginning of the 2000s with the concepts ‘Inner-Browsing’ (Galli et al., 2003) and asynchronous JavaScript and XML (AJAX) (Garrett, 2005). It breaks with the traditional way of moving from one page to another in favour of asynchronous loading and replacing parts of the current page. This allows for a website to evoke the look and feel of a regular desktop application.

The term progressive web application (PWA) was initially coined in 2015 by two Google employees in an online Article (Russell & Berriman, 2015). At its core, it describes the process of a website ‘progressively’ evolving into a proper device application by adding offline functionality and blending with the operating system functionality. It is often built atop the concept of an SPA and can be perceived by the user as an application they own instead of just accessed at a remote location.

A *real-time web application* enhances the user experience by relaying relevant changes on the server to the server as they happen. This can be a simple chat application or a more complex collaborative multi-user environment. While real-time updates can happen on any multi-page website, they can also be a beneficial feature of an SPA or a PWA. Instantaneous updates are commonly realised using WebSockets, allowing updates to be pushed to the client whenever a resource on the server changes.

2.8. Frontend frameworks and libraries

There is a wide range of available JS frameworks to build dynamic frontends for SPAs and PWAs. The three libraries currently dominating the landscape are React, developed by Facebook in 2013, and Vue, created by Evan You in 2014. These libraries can be used with frameworks to offer complete routing and state management solutions. Another popular framework is Angular, initially released by Google in 2010 and re-released in 2016.

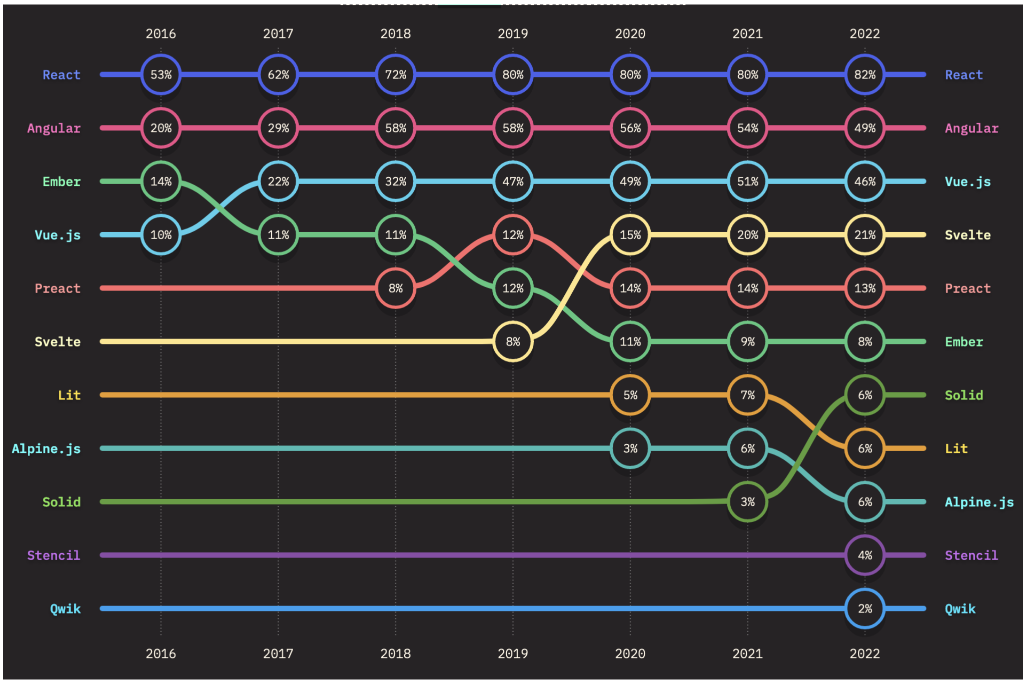

Figure 2.1.: State of JS: Most used frontend frameworks in 2022 (Greif & Burel, 2023c)

React[22], developed by Facebook and maintained by its successor Meta, has become the most widely used tool for building SPAs and is steadily leading the rankings for most used frontend frameworks both in the Stack Overflow (Stack Overflow, 2023b) and the State Of JS (Greif & Burel, 2023c) polls. By definition, it is not a framework but a user interface (UI) library that builds on other extensions to support state management, routing and deployment functionality. Although it is not a framework itself, there are existing frameworks like Next.js[23] for the web and ReactNative[24] for building mobile apps using native functionality. React makes use of JavaScript XML (JSX), which allows directly mixing inline HyperText markup language (HTML) with the JS or TS code structure.

[22] https://react.dev
[23] https://nextjs.org/
[24] https://reactnative.dev/

Vue[25] was developed by Evan You and is maintained by an international team of individuals. It had a relatively marginal presence in the US and Europe in the first years after its inception. This can be partially attributed to its origin in China, as most of its supporting modules were localised in Chinese. Over the years, it grew in popularity and received much more international support, eventually overcoming the language barrier. Unlike React, it is billed as a 'progressive framework' that provides fundamental functionality for building reactive components but also accommodates more complex use-cases (You, 2021). Vue builds on standard JS or TS, HTML and Cascading Style Sheets (CSS) to create components, recommending a simple template mechanism mixed with reactive substitutions. However, it also supports using JSX for specifying inline HTML within JS. As with React, there are extensions and frameworks like Quasar[26] and Nuxt[27] that enable even more sophisticated workflows for application development and deployment.

Angular[28] was initially released by Google in 2010 as AngularJS and officially discontinued in 2022. A wholly overhauled and currently used version 2 was released in 2016 and maintained by Google. It differs from React and Vue in that it is a complete framework containing everything required to build and deploy an application, and it explicitly recommends TS as a programming language. The framework is also less flexible in that it is opinionated and has its own set of best practices baked into the framework's structure.

2.9. Backend frameworks

The *Express*[29] framework provides the basic functionality to create web servers, including routing and middleware functionality. TJ Holowaychuk developed and sold it to Stron-

[25] https://vuejs.org/
[26] https://quasar.dev
[27] https://nuxt.com
[28] https://angular.io/
[29] https://expressjs.com

gLoop, which IBM subsequently acquired. It is currently under the stewardship of the OpenJS Foundation. Express has become the de facto standard for building web services in JS, leading the ranking in the State of JS survey (Greif & Burel, 2023b). Although it contains the necessary parts to make a web service, it does not enforce a specific architecture, which can be problematic for maintaining a robust application structure. For developers who prefer a more explicit structure, various other frameworks that add more opinionated structures or extensions are built on top of it.

Billed as a successor to Express, *Koa*[30] is developed by the team behind Express. It aims to provide a more robust and minimalistic iteration of the middleware-based architecture of Express. Like Express, it allows for building a service from scratch in free form but is also the basis for other, more explicitly structured frameworks.

Other frameworks and a more stringent and structured application structure might be more desirable for complex applications. Numerous JS frameworks, some based on Express or Koa, and others that provide their own basis for routing. To review all possible options is beyond the scope of this study. In the following, three frameworks are selected for their specific nature related to popularity and stability, with an explicit focus on real-time applications.

Table 2.6.: Stack Overflow Developer Survey 23: The top three multi-user databases (Stack Overflow, 2023a)

Database	% of all question respondents	Stars on GitHub (k)
PostgreSQL, 2010	45.55	14
MySQL Server, 2014	41.09	9.9
MongoDB, 2009	25.52	25

Nest[31] is a backend framework for developers who look for a more strictly opinionated and robust setup than Express, e.g. for enterprise applications. It follows a modular

[30] https://koajs.com
[31] https://nestjs.com

concept, making dependencies available to the services via injection. There are multiple database options, and transports can be HTTP and WebSockets. There are command-line interface (CLI) scripts that enable automatic generation of boilerplate application code, and the language used to build Nest applications is TypeScript. It ranks second among the most-used backend frameworks in the State of JS survey (Greif & Burel, 2023b).

The *Feathers*[32] framework takes a different approach, making few assumptions about the specific application structure. It uses aspect-oriented programming and a service-centric architecture and before-, after- and around-hooks (so-called 'cross-cutting concerns') for the services that modify basic behaviour or add functionality. There are adapters for a wide range of databases and authentication methods. The framework has a dedicated concept of channels that enable real-time functionality and messaging to clients. Real-time transports are abstracted and can be deployed using Socket.IO or µWebSockets (see section 2.5). It also provides a CLI to generate application code in JS or TS. Feathers started as a hobby project by David Luecke and Eric Kryski in 2013 (Kryski, 2016) and is currently maintained by David Luecke and a community of individual contributors. It still ranks in the lower percentages in the State of JS survey but almost doubled that percentage from the previous one in 2021 (Greif & Burel, 2022).

Meteor[33] focuses explicitly on real-time applications using WebSockets. The framework is an outlier because its core is open-source, but other parts are proprietary code. Nonetheless, it should be mentioned because it has been around for over ten years and uses WebSockets exclusively. It was released in 2012 by a startup company, immediately received venture capital funding from Andreessen Horowitz and was eventually sold to Tiny Capital in 2019 (Lardinois, 2019). The framework primarily uses MongoDB as a database system and initially provided its own package manager and ecosystem, build system, and template system based on Mustache. This exclusive strategy has been abandoned in favour of adopting the Node Package Manager. Still, it seems to be subject

[32] https://feathersjs.com
[33] https://www.meteor.com

to debate regarding its ease of use versus its ‘growing pains’ and related trouble with wide adoption (doppp & forum users, 2019).

2.10. Databases

Table 2.7.: State of JS survey: Most used backend frameworks (Greif & Burel, 2023b)

Framework	% of question respondents
Nest	30.2
Feathers	8.8
Meteor	2.7

PostgreSQL[34] is a very widely used database which uses a table-based data topology and implements structured query language (SQL) for interaction with the database and its contents. The relatively rigid database schema provides a more solid structure for data storage and retrieval but, on the other hand, requires migrations to be written to transition from one database structure version to another. This can become tedious if the data modelling process is continuously ongoing and volatile. It has an extensive feature set supporting complex data structures, geographic information system (GIS) data and data structured in JSON format. Developed in the 1980s at the University of California and switched to the SQL in the 90s, it has remained a popular choice for enterprise and small-scale use.

Similar to PostgreSQL in that it also uses SQL, *MySQL*[35] supports many of the features of PostgreSQL, but has an overall smaller feature set. It was initially developed in the 1990s by the private Swedish company MySQL AB and was forked as a completely open-source version in 2009 and renamed MariaDB[36]. It is still a popular choice, especially

[34] https://www.postgresql.org
[35] https://www.mysql.com
[36] https://mariadb.org

for smaller web projects that don't need the extra functionality and value its relatively simple setup.

MongoDB[37] is a document store database that is designed to hold large amounts of unstructured data. It has its own query language and features aggregation functionality that allows map/reduce and transformation operations or resolving of relations on the data before being sent to the client. It uses the Not-only SQL (NoSQL) paradigm and focuses on storing documents of any kind, which can become problematic in relatively informal development environments since it can lead to inconsistent data very easily if appropriate care isn't applied during application development (it supports schema validation, but that is not mandatory). However, this loose schematic handling can be beneficial if the application data isn't adequately modelled from the get-go and is subject to more frequent changes. It was initially released as open-source in 2009, then was put under a proprietary license in 2018, but remains available to be used for free with limited support.

2.11. Application deployment

Containerisation, in the context of computing infrastructure, refers to the 'packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure.' (IBM, n.d.-b) It was popularised through the release of the Docker Engine[38], an open-source project devoted to creating an industry standard for application containerisation (Barbier, 2014). The Docker team eventually launched the Open Container Initiative (OCI) in 2015, which serves as 'a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtimes.' It

[37] https://www.mongodb.com
[38] https://www.docker.com

subsequently received Docker's container runtime and format as a donation, which was released as runC version 1.0 in 2020 (Linux Foundation, n.d.). Recently, it has become the de facto standard for packaging and delivering applications in the web development field and beyond. GitHub reports that 'in 2023, 4.3 million public and private repositories used Dockerfiles — and more than 1 million public repositories used Dockerfiles for creating containers.' (Daigle and GitHub, 2023)

Container orchestration builds on the concept of containerisation. 'Container orchestration automates the provisioning, deployment, networking, scaling, availability, and lifecycle management of containers.' (IBM, n.d.-a) The concept first gained popularity as Docker 'swarm mode', a functionality of the Docker software. Still, its most successful instance so far is as the software package Kubernetes[39], which originated at Google in late 2013 (Burns, 2018) and went on to be included in the Cloud Native Computing Foundation (CNCF), a project by the Linux Foundation, that 'aims to advance the state-of-the-art for building cloud-native applications and services' (Linux Foundation, 2015). It can be extended, highly customised and deployed on anything from an embedded device to a large-scale cloud infrastructure, providing a versatile deployment and management tool for many application infrastructures.

[39]https://kubernetes.io

3. Methodology

This feasibility study is based on two essential parts. The first is a reference implementation, providing insights on the work necessary to arrive at the base functionality. Based on the implementation, an analysis of the application's functionality is made, an overview of the time spent on development and there is a qualitative review of the resulting codebase and a reflection on the work process.

3.1. Reference implementation

To produce a valid test subject for the proposal, the reference implementation is created according to a prior selection of tools and methods deemed appropriate for the task at hand. The choice is made from the concepts and tools presented in chapter 2.

First, possible candidates are identified through internet search and then at least three candidates are selected using the number of 'Stars' received on GitHub as an indicator of popularity. In some cases, another metric has to be used where the technology itself predates GitHub (e.g. databases or programming languages) and its popularity should be judged by other means. In this case, there is a yearly developer survey being conducted by the popular technology forum Stack Overflow with over 90.000 participants for 2023 (Stack Overflow, 2023c) and the 'State Of JS' survey with over 20.000 participants that is more focused on web development (Greif & Burel, 2023a). Additionally, GitHub is publishing a yearly statistic on its public repositories, which is helpful for identifying technological trends and popularity among millions of open-source repositories (Daigle & GitHub, 2023). The selection is further narrowed by focusing on specific requirements that the study posits towards its supporting technology.

The decision on choosing a specific candidate is then made not by popularity alone, but with a stronger weight on a good fit to the project's requirements and needs. If a less popular framework fits the specific style of development, it is preferred to the status quo. Another case might be a more recent project that hasn't collected as high of a rating on GitHub, but presents a promising new paradigm or feature set.

The application development works from the most basic boilerplate code towards finding the appropriate structure for the specific use-case. Well-known and easily defined components are built first and the special functionality is then built on top in constant cycles of adding functionality, reviewing the codebase and refactoring towards abstraction and separation of basics from specifics. As there is only a rough architectural model for the project defined beforehand, tests and documentation is written later in the process, as the parts stabilise on their own and in their relationship among each other. This method does not strictly adhere to common development procedures, but borrows loosely from agile development (sprints, review, reorientation) and simple forms of the ideas put forward in the book 'Pattern-Oriented Software Architecture', such as layering, separation, and standardised messaging (Buschmann et al., 2007). A more rigid structure for the development process might be desirable for teams, but the various and disparate 'moving parts' in conjunction with heavy reliance on browser-only APIs complicate the creation of a well-simulated testing environment using either real or mock-data.

The application is implemented in its entirety, documented and packaged. Appropriate test coverage is provided for the core functionality in the API and the messaging components, and the overall time spent is logged in timesheets and categorised by the general work areas. The application's server components are deployed early on to university hardware and made available over the internet. The client application is then run and tested on various consumer computer systems and networking setups.

3.2. Analysis and evaluation

A statistical analysis of the timesheets provides insight into the time spent on various aspects of the software. It should differentiate between basic boilerplate code that can be reused and custom code used for the actual use case, in order to provide insights both into the feasibility of setting up such a system from scratch with only a single developer, as well as the potential cost of just reworking the parts deemed transient and related to the specific use-case.

The application's performance is tested regarding the load put on the central processing unit (CPU) (server and client) as well as the network throughput and latency. It is verified that all message processing works as expected through unit testing and simple testing tasks performed on the application. A practical test that analyses the actual user experience and using performers and real dance interaction is beyond the scope of this study.

The *code quality* is assessed based on volume (lines of code without comments) and cyclomatic complexity, which counts the number of linearly independent paths throughout a piece of code (see McCabe, 1976). Both metrics should be kept low, as code with a high number lines makes it harder to read and a high complexity is harder to follow and understand. It is recommended to limit the cyclomatic complexity to a value of 10, with special justified exceptions going up to 15 as a maximum (Wallace et al., 1996, p. 15), in order to facilitate a development style that aims to reduce the danger of potential errors caused by excessive complexity. The code volume is difficult to reduce to a static maximum, but 'the [Software Assurance Technology Center (SATC)] has found the most effective evaluation is a combination of size and complexity' (Rosenberg et al., 1998, p. 6), so a self-imposed threshold should be set, even if exceptions are made later on. For object-oriented programming (OOP), it is often advised to limit a file to a single class, but depending on the language used, the file can also just contain multiple functions that are better grouped together instead of scattered across multiple files. A general rule of thumb that has proven to be a good measurement from past experience is that a single file should not exceed 160-200 lines of code (without comments), in order to be able to quickly skim it and get a general overview. There are exceptions to these rules where a specific function or class exceeds this threshold, but breaking it up would not make it easier to understand. In those cases, there can either be a discussion about how to change the general application design to partition the functionality differently, or it can be decided to keep this code as is and improve documentation. Reducing volume and complexity is especially important if the code should be passed on to other people that want to maintain and modify it for further use. Here, the general idea is that the transient (hackable) parts (e.g. UI) should be as simple as possible, avoiding unnecessary complexity, and the more static and stable core parts can be the location where the more complex parts is moved to (e.g. the core SDK).

Additionally, a *critical reflection* and analysis of the development process should weigh the expectations against the experiences made during the implementation of the decisions made in planning the application. It should evaluate the notion of general reproducibility, feasibility and discuss the benefits and drawbacks of establishing a task-specific application from scratch.

4. Application concept

4.1. Architecture

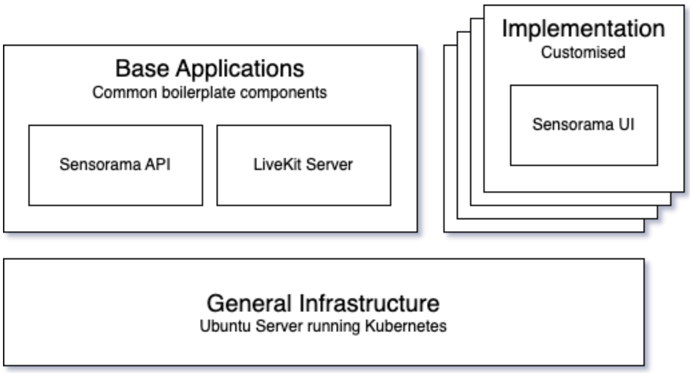

Figure 4.1.: The main components comprising the application architecture

The underlying hardware infrastructure is a bare-metal system running on-premises at the university. Due to the containerised packaging and deployment, it could also easily be deployed in a cloud environment or other hosting platform. No special hardware is required, and the system can run in any environment that provides network access, storage space and standard computing resources.

In an otherwise containerised application environment, the underlying software infrastructure is minimal. The components required are a Linux OS, in this case, Ubuntu, with installations of Docker (with ContainerD) and Kubernetes.

4.2. Application infrastructure

While all the frameworks represented in Figure 2.1 could be used to build an application as envisioned in this study, Vue is selected as the tool of choice due to the relatively high acceptance and the comparably easy learning curve. While it might not be the choice for large-scale or enterprise apps, the low entry barrier and the simple structure make it ideal to get an app up and running quickly, experiment with it and pass it on to others for hacking and custom modifications. To accelerate and simplify the initial development, the Quasar framework is used as it extends the basic functionality Vue provides by a UI library with layout tools, preset interface elements and a comfortable development and deployment environment.

The choice for a backend framework lands on Feathers and, by extension, Koa. The simple structure and code generators allow for a speedy setup and deployment of a simple WebSockets API that provides authentication and resource management. It is connected to a MongoDB database because there is no definitive initial plan of how the stored and retrieved resources are explicitly structured and typed. With a document store, the data can be easily overwritten with updated data and then wiped before the schema is fixed.

LiveKit is chosen as the WebRTC server implementation because it is effortless to set up as a container running along the Redis database in Kubernetes. It is extendable and scalable, and there is even a hosted variant for people who do not want to run their own server. While Mediasoup would allow a more precise implementation and probably more efficiency, the workload overhead for building everything already offered by LiveKit is too much effort for this kind of application. However, it might be interesting to see how components based on Mediasoup could be dropped into this application structure.

4.3. Design paradigms

The basic design paradigm used for the Sensorama application is that of an SPA. As there is already a remote API involved in managing access to shared resources, the PWA paradigm is not immediately of use. Still, it could be implemented with the existing application as well. It is an exclusively real-time application that uses WebSockets for all transmission between app components and uses the WebRTC standard for user communication. It is set up as a monorepo, where all components are developed across languages in one repository.

The application's custom part is partitioned into the user interface, which is a static built HTML/CSS/JS bundle, the API, which is a single-process Node application and the so-called 'Data-Producers', which are external native utilities written in Python and C++ that provide bridges to motion capture hardware.

The primarily favoured coding paradigm is object-oriented programming, but this is not strictly enforced for all components. As some frameworks prefer different, more functional paradigms that are also compositional (Vue) or aspect-oriented (Feathers), it is beneficial not to enforce a singular coding style. While this is usually considered bad practice in terms of maintainability for long-term development, but it serves the purpose of a modular and somewhat unstable 'single-use' application environment.

An essential part of the development concept is the sequence of development phases. As there is no explicit definition beforehand, the development starts out by establishing a functional skeleton first and working within that to carve out the actual functionality. Then, monolithic large blocks of code are built where different approaches to desired functionality are quickly tried and discarded, or kept and subsequently extracted into separate locations, grouped by functional association. In this scenario, it is important to review and refactor regularly and often in order to solidify the application structure and prevent it from dissolving into 'spaghetti code' and to prevent unnecessary side-effects among the components. Only towards the end of the development process are the core features extracted into the separate SDK module and unit-tests are written. As the application would now move into practical testing and thus, some sort of ‘production’ deployment, the core functionality needs to become more rigid and covered by test cases.

The general user interface and data producers are considered transient because they serve a singular use case and should be subject to frequent modification. These application components should be hackable and replaceable, so they are not tested, at least in the scope of this study. However, more stable and general tools and extensions that warrant setting up testing could still be added in the future.

The unit-testing focuses on the data input and output for the core functionality to provide a stable foundation for the system. By modeling the basic request and response cases and formulating them as tests, potential later users have a tangible way of understanding the application’s core mechanics.

For JS, Jest, Mocha and Jasmine are three popular testing frameworks (4.1), among others, that can be used for implementing unit-testing for the project. In this case, the selection forgoes the most popular option of Jest in favour of Mocha, which is used by the Feathers API framework in its generator for boilerplate code. This way, basic tests to base work on are already available and, to keep the project consistent across modules, will be adapted for the core SDK module as well.

Table 4.1.: Popular JavaScript testing frameworks

Framework	Stars on GitHub (k)
Jest, 2013	43.2
Mocha, 2011	22.4
Jasmine, 2008	15.7

4.4. Application components

The application comprises several third-party components merely deployed as-is (WebRTC[redacted] databases, static web server) and the custom-developed parts described here.

4.4.1. Core SDK module

The core functionality is built into a separate module to enable its integration into other setups that use different frameworks or architectures. The module uses the NPM's package format and can be used in the browser as well as in Node.js. While this module carries the most fundamental functionality, it is set up last in the development process, as the essential parts only crystallise during the initial development phase.

4.4.2. Web frontend

The web frontend provides the main entry point for the users. It allows authentication via a local username and password combination and then provides objects modelled as virtual 'Spaces' that are the central anchor to organise all communications. A space object then maps to the the concept of a 'chat room' in LiveKit or other real-time communications environments. Users can create spaces, name them and then join them, becoming active data producers, or choose to view them as passive spectators.

Depending on the participant's role, a space is rendered as a different set of components. Participants who actively join have access to a LocalProducer and a HeadTracker component. These components provide a direct link via WebSockets to the external data-producer utilities and a WebBluetooth connection to the custom head-tracking device built on Arduino. Those who only view the space do so via a dedicated scene viewer component that brings together all incoming streams and signals.

The frontend coordinates connections between the WebRTC server, the backend API and the local utilities. It also implements the various web standard APIs needed for sound, graphics and communication.

4.4.3. API backend

In the backend, the API server is tasked with managing the basic connecting objects (spaces and users), general authentication, and generation of access tokens for the LiveKit server. Through its real-time implementation, it can notify connected clients of changes like other connecting users or updates to data. The Feathers framework exports its own client library that is specifically generated for the current server configuration and can be directly integrated into Vue by using a specific client adapter module ('feathers-pinia') that handles authentication and basic create, retrieve, update and delete (CRUD) operations.

4.4.4. Native utilities

Three different native utilities are additionally implemented.

The general *data producer* is written as a CLI utility in Python, as it implements various Python-specific extensions: the DepthAI framework, used to work with the Oak-D line of 3D-cameras, OpenVino for interacting with various machine learning (ML) models for pose recognition or point-cloud extraction, Open3D (Zhou et al., 2018) for working with point cloud data and general spatial operations and PyMotion, a library for working with recorded Biovision hierarchy (BVH) motion capture data files (Ponton, 2023). Python also allows for easy statistical data analysis using NumPy, which is used for movement quality extraction. This data producer also includes functionality to extract the movement qualities from the

For real-time streaming of live motion capture data from the Captury Live system, there currently only exists a C++ client library provided by the system's manufacturer. Thus, the *Captury data producer* component has to be implemented separately and uses a C++-based WebSockets server streaming the library's received data.

As there was no affordable, open and platform-independent *head-tracking* solution, this component was quickly prototyped using a BluetoothLE-ready Arduino device (Nano RP2040 Connect) and an inertial measurement unit (IMU) component for absolute orientation measurement by Adafruit (9-DOF Absolute Orientation IMU Fusion Breakout) that can be directly connected to the Arduino using the Inter-Integrated Circuit (I2C) serial bus. The data read from the IMU device is then posted as binary messages on a simple Bluetooth service. This device can be directly integrated using the browser's WebBluetooth web standard.

4.5. Messaging

To enable a streamlined messaging among the disparate application components that are based on different languages and run in different environments, standardised web protocols are used. Starting on the client side, the motion capture data producing utility starts a local WebSockets server that the web application running in the browser can connect to and receive the live data. The browser application can also connect to the custom head-tracking device using the WebBluetooth standard and receiving data messages using the Generic Attribute Profile (GATT).

The conferencing functionality implemented in the web application is sending audio and the producer utilities' data to other participants via the LiveKit server using the WebRTC protocol for communication. The LiveKit server can push status updates as HTTP webhook calls to the API server to notify the API server about connects and disconnects. The API server uses the WebSockets protocol to relay updates on spaces and users to the client browser and receive authentication and general data requests.

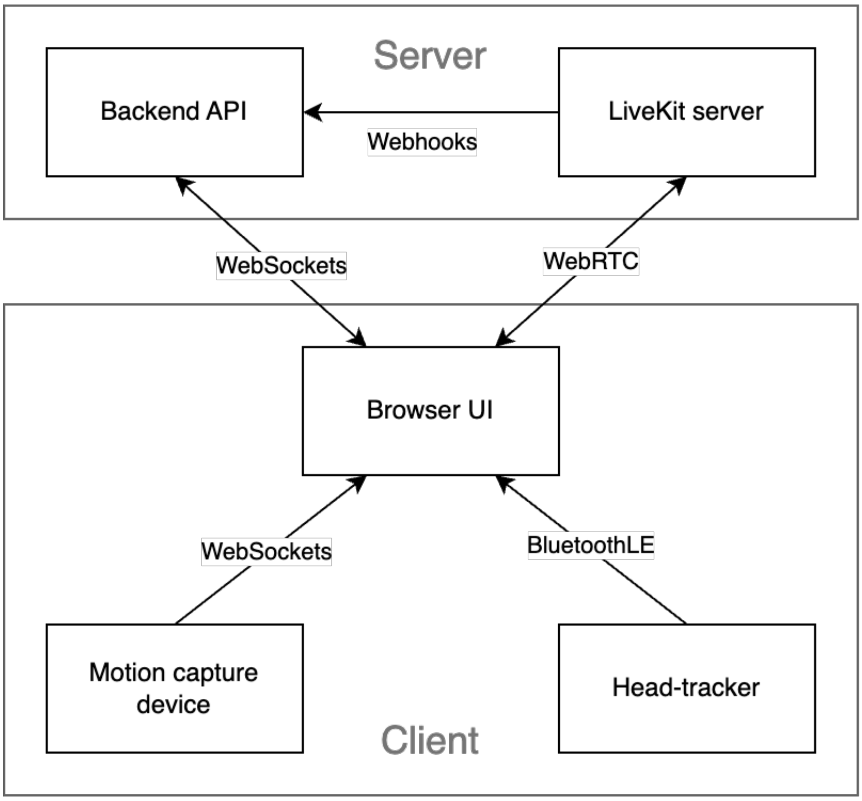

Figure 4.2.: Messaging flow between the application's components

4.6. Data modeling

There are four core data models being used within the application. The two models stored by the API server are spaces and users. These are very simple reference objects that bring together multiple participants in a common space, authenticated by personalised tokens, allowing them to exchange messages, which are instances of the third data model.

Each *User* can own multiple *Space* objects. A *Space* is a container object that references a coherent shared space which is constructed from multiple parties' sensor readings. *Users* can request one or more *Token* objects that allow them to connect to a ‘conference room’ on the LiveKit server that maps to a specific ID of a *Space*. Once connected, the LiveKit server notifies the API server of the new connection and now the connected *User’s* ID can be found in the list retrieved from the virtual ‘connected’ property in the *Space* object.

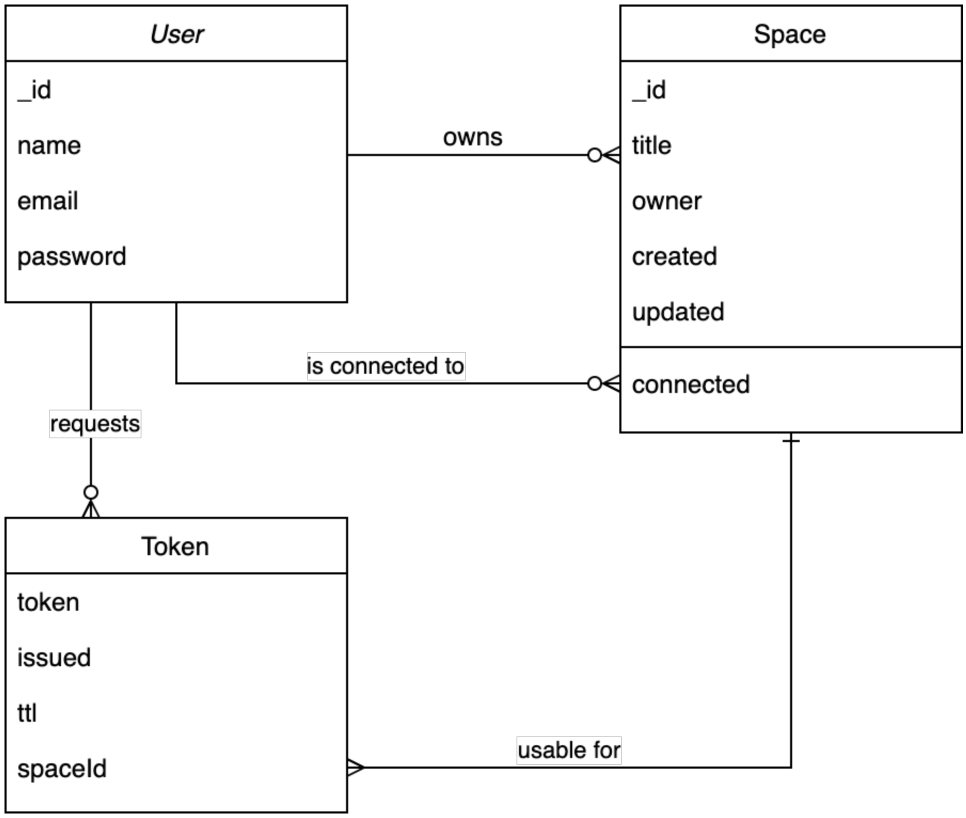

Figure 4.3.: Basic data model used in API server

The data messages are not encoded as JSON text messages, but sent as raw data to make them as small as possible. Messages are structured as byte sequences, with a 64bit long integer timestamp using the first eight bytes, then a single byte with an unsigned integer for selecting a message schema from the enumerated message types and then a freely defined sequence of different number types (Figure 4.4). Here, 32bit floating point numbers are used for all of the sensor readings as the numbers stay sufficiently small and the precision is enough for millimetre measurements, statistical values or angles. The numeric values are encoded in Little Endian format that is consistent across the tested environments, but should be explicitly adhered to if other components are added to the application.

Generic Message Structure				
Message Header		Message Data ...		
8	1	4	4	...
64bit Integer Timestamp	UInt Type	32bit Float Number	32bit Float Number	...

Figure 4.4.: The basic message structure for transmitting numeric sensor readings

The message types are stored as JSON. Shown here is a simple example message schema for a nanosecond timestamp (*t_ns*), *type* and one or more 3D *points* stored as floats.

The root object's property names resolve to the key under which the value can later be accessed. The *index* property specifies the byte index in the message, *count* specifies if the value repeats in sequence or is singular. *dims* sets the dimensions for the value (e.g. '3' for a 3D point). The *type* can be a 'Uint8', 'Float32' or a 'BigInt64' and the property 'le' specifies if this value is encoded as Little Endian (Listing 4.1).

```json
{
  "t_ns": {
    "index": 0,
    "count": 1,
    "dims": 1,
    "type": "BigInt64",
    "le": true
  },
  "type": {
    "index": 8,
    "count": 1,
    "dims": 1,
    "type": "Uint8",
    "le": true
  },
  "points": {
    "index": 9,
    "dims": 3,
    "type": "Float32",
    "le": true
  }
}
```

Listing 4.1.: Example pose message schema

5. Implementation

5.1. Project Setup

The supporting *server infrastructure* was deployed first to allow development on a working remote WebRTC infrastructure. The basis was a clean, freshly bootstrapped Kubernetes installation running on a single server with 16 CPU-cores (with multithreading), 64GB RAM and a 512GB SSD drive, located at Mainz University and connected to the internet via a one gigabit network connection. LiveKit and its Redis database were installed via the application deployment manager Helm, using an official installation from its maintainers. To simplify the deployment, LiveKit was placed behind a reverse proxy to manage SSL termination via the LetsEncrypt service, as well as routing to the actual service running inside the cluster. However, this simplified setup required LiveKit to be configured to listen on a single TCP port instead of a range of UDP ports, as it would usually be deployed. The potential downside of this deployment configuration was deemed insignificant, since the service does not have to scale to more than a handful of users. The detailed Kubernetes setup instructions are documented in the according folder in the Git repository [1].

The *LiveKit* installation was deployed with only slight deviation from the default configuration. It was set up to use TCP as a transport protocol to allow easier integration with SSL termination using the reverse-proxy. Otherwise, the configuration defined the endpoints for sending webhook requests and custom credentials for making requests

[1] Kubernetes setup instructions: https://github.com/dasantonym/sensorama/tree/master/kubernetes

to it via the server-side SDK and generating valid access tokens for users to connect to rooms. The Helm chart used for the installation set up the system along its Redis database installation in a single command and the server was immediately ready for connections.

The development process was conducted in a desktop environment, using the suite of tools developed by JetBrains (WebStorm, PyCharm and CLion), as these are free for educational use and provide a complete out-of-the-box-environment for development including debugging, smart code-completion, versioning, containerisation and deployment. A local Docker Desktop installation provided the possibility to run services and databases locally to support development before publishing to the production environment. Versioning was done via Git on the GitLab platform provided by Mainz University for the Rhineland-Palatinate [2].

5.2. API server

The first implementation, the API server, was generated using the Feathers CLI tool with standard username and password authentication and WebSockets as well as HTTP transports enabled. It implements the core services for Users, Spaces, Tokens and LiveKit events. These services were also autogenerated using the Feathers CLI utility and were used largely unmodified, except for adding the properties on the models for Users and Spaces as defined in section 4.6. A custom service class was added for the Tokens, as these do not persist in the database, but are generated on the fly by the LiveKit server SDK. All other boilerplate code for the API, including the MongoDB integration, the authentication mechanism, as well as the REST and WebSockets transport integrations were also autogenerated using the CLI. An additional custom LivekitEvent service was added to the API and allowed it to receive webhook requests via HTTP from the LiveKit server containing updates on connecting and disconnecting users. These events were then

[2] https://gitlab.rlp.net

not persisted, but relayed to the real-time channels that the users are connected to. The channels feature provided by Feathers was used to automatically subscribe connecting users to updates on the services for Spaces and Livekit events.

5.3. Core SDK

To make the base code independent of the use-case, the basic functionality was bundled in an NPM module, which can be used in other projects. This module contains the abstract classes 'DataProducer' and 'HeadTracker' alongside the message specification for the various types of transmitted data. It was written to propagate updates via events instead of the reactive patterns used in Vue so that it can also be used independently from the framework used for the study. Unit tests were added to the module to support a stable implementation.

5.4. User interface

The Quasar framework provides a CLI to generate new projects, which allowed to select basic implementation details (e.g. language, state management) and then produced a complete and working empty Vue project with sample components that served as the starting point for the UI implementation. The first thing added to the project was the library 'feathers-pinia'[3], which is provided by the Feathers developer community and promises easy integration of an existing Feathers API with any Vue project using the Pinia state management system. The extension was integrated by linking the client library that the Feathers server project automatically generates and then configuring basic authentication settings. The routing configuration and page-components were set up according to the sitemap.

[3] https://feathers-pinia.pages.dev

The overview page for the spaces is a list of the names alongside the currently connected users and buttons to either join as a participant or to passively view it as a spectator. On joining a space, the user first needs to activate their microphone to allow the audio context to be activated. The user is then presented with three basic control panels. The ‘Data producer’ panel allows setting a URL of a local WebSockets server published by a data producer, setting the message type received from it and enabling a tracing function to log the packet transmission statistics. Once connected, the panel switches to showing a preview of the incoming points data, transmission statistics and a button to disconnect. Internally, the panel creates a reactive data store that instantiates the ‘DataProducer’ class from the core SDK, watches incoming message events and populates the received data as reactive properties to be used across the UI in various other components without the need of instantiating more class instances. The second panel, ‘Head tracker’ works in the sae way but uses the Web Bluetooth API to select a nearby device and connect to it to receive its data messages. The third component is used to configure the sonification by setting thresholds on incoming movement quality values. Once a threshold is crossed, a sound event is triggered.

When using the page to view a space, there is only a single component, the ‘SpaceViewer’ that shows a 3D room in which all incoming points are rendered as small spheres, giving the impression of a human figure and for each participant there is a differently coloured light source that follows the centre of mass of the points associated with the participant. The viewer also renders all sonifications and audio streams at their respective spatial positions. This allows the user to view the 3D scene on screen while listening to binaural audio or to use the built-in VR functionality to experience the scene in an immersive way.

5.5. Data producers

The general *data producer* was written in Python and provides multiple data sources: an interface to a Blazepose implementation on the Oak-D 3D camera, as well as reading depth images as point clouds from the camera and an interface to load and playback motion capture data in the BVH file format. All three data sources were implemented as separate Python classes, because the classes related to the Oak-D camera were built by modifying existing code for pose recognition (geaxgx, 2021) and pointcloud processing (Erol444, 2022). The BVH data producer however, was created from scratch. The separate producer classes were set up to all support calling a function to return current point data as a multidimensional array in a loop that is quantised to a fixed framerate of 25 frames. The returned data is then packed as a byte sequence and sent over websockets according to the appropriate format (see section 4.6). This way, the disparate sources could all be imported in a single main file that uses a combined set of utility functions for running a WebSockets server and for packing data.

Due to the lack of a Python-based client for the Captury Live system, the Captury producer had to be implemented separately as a C++ project using CMake as a build system and based on the 'RemoteCaptury' client library (The Captury GmbH, 2018), as well as an example project for a WebSockets server implementation in C++ (Rehn, 2016).

The custom built head tracking device was implemented as an Arduino project and as such, was first realised as a hardware setup and then outfitted with a custom built firmware written in the Arduino-specifc flavour of C/C++. The hardware implementation was created using the 'Arduino Connect RP2040'[4], which is based on the Raspberry PI 2040 microcontroller and has an onboard BluetoohLE module. The IMU module used was the '9-DOF Absolute Orientation IMU Fusion Breakout'[5] which uses the BNO055

[4] https://docs.arduino.cc/hardware/nano-rp2040-connect/
[5] https://www.adafruit.com/product/4646

chip produced by Bosch[6]. This chip has the benefit of already pre-processing the data coming from the gyroscope, accelerometer and magnetometer into an absolute world position which can be directly read out from the breakout board via the I2C bus. As a third component, a small 3.7V lithium battery was added alongside a charging module[7]. Only six connections needed to be soldered between the three modules (2x charger and 4x IMU) and the resulting circuit was ready to function as a custom head tracker. For the software implementation, the basic example code for the Adafruit module was used to set up a continuous polling of the positioning module, reading the values for position and device calibration status and sending them as byte sequences at a fixed rate of 25fps over BluetoothLE.

[6] https://www.bosch-sensortec.com/products/smart-sensor-systems/bno055/

[7] https://www.adafruit.com/product/1905

6. Evaluation

6.1. Statistics

The statistical evaluation focused both at the extrinsic properties of the application and the development process itself (e.g. code volume, complexity, time spent on development), as well as the intrinsic functional properties of the core functionality that are reflected in the cost of CPU-, network-usage and message latency. Cost of rendering audio and video, as well as general user experience metrics were beyond the scope of the study, as these are highly specific to the task being implemented and are considered transient.

Computers used in the performance evaluation were end-user laptops with the following specifications: *Computer A* is equipped with an 3.1GHz dual-core Intel Core i5 CPU, 16GB of **RAM!** (**RAM!**) and a 500GB **SSD!** (**SSD!**). *Computer B* is equipped with an Apple M1 Max CPU with ten cores, 64GB of **RAM!** and a 1TB **SSD!**. Both systems used the Google Chrome browser (version 121.0.6167.85), ran macOS 13.6.3 and connected to the network via 5GHz WiFi. Both computers used the same **NTP!** (**NTP!**) server to synchronise clocks and the synchronisation was refreshed before each data sampling.

The *latency measurements* were conducted for the data channels and over a consumer 50Mbit **DSL!** (**DSL!**) connection. They were repeated at different times to account for variance in overall network service quality, and were also conducted over different international **VPN!** (**VPN!**) connections to simulate connection distances within and outside of Europe. The measurements always used the BVH data producer sending the message type for movement qualities alongside 29 key points at a rate of 25 messages per second. Payload size was 453 bytes for each message, amounting to a required bandwidth of about 11.3 kilobytes per second for each motion capture data stream. The audio streams were published alongside the data packets, but were not measured for latency.

The results for the latency analysis are shown in separate graphs for each computer containing the datasets for the local and remote messages on each device.

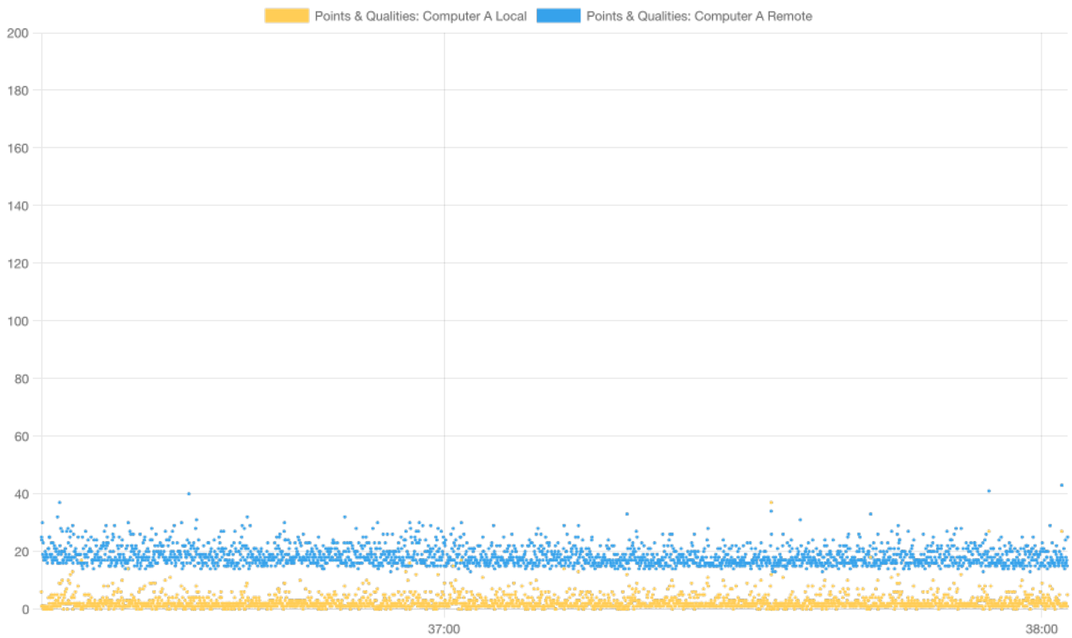

Figure 6.1.: Computer A: Latency in milliseconds for local and remote messages

Results for computer A (6.1) show a median latency of 18ms for remote messages received over the WebRTC connection and 2ms for the connection from the local data producer to the browser. Both values show a jitter at a variance of about 11ms for the remote connection and about 6ms for the local connection.

The results gathered on computer B (6.2) show a median latency of 48ms for the remote messages and 1ms for the local data producer connection. Here, the jitter happens

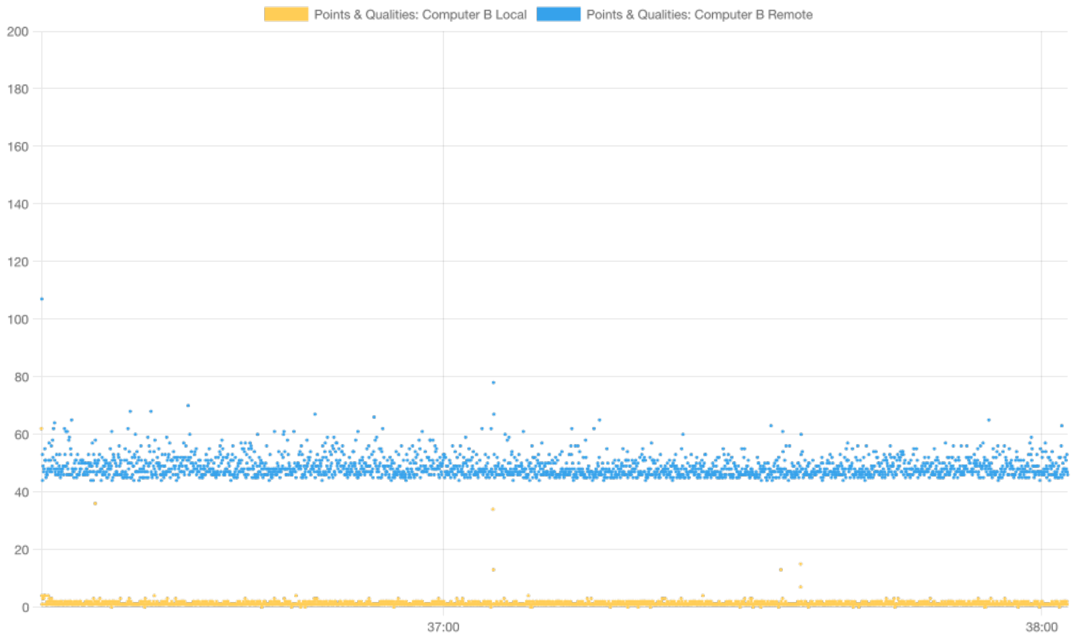

Figure 6.2.: Computer B: Latency in milliseconds for local and remote messages

at a variance of about 13ms for the remote connection and about 3ms for the local connection.

The development process evaluation was based off the timesheets that were kept during the development process. All recorded tasks were categorised by the language used (e.g. JavaScript), the component worked on and the type of work

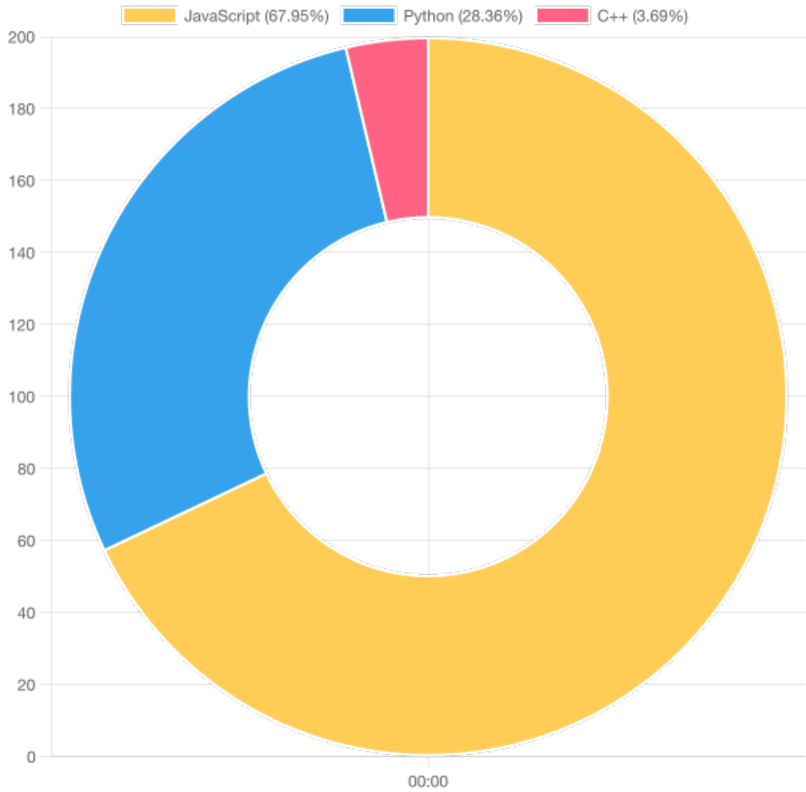

Figure 6.3.: Time spent on various programming languages

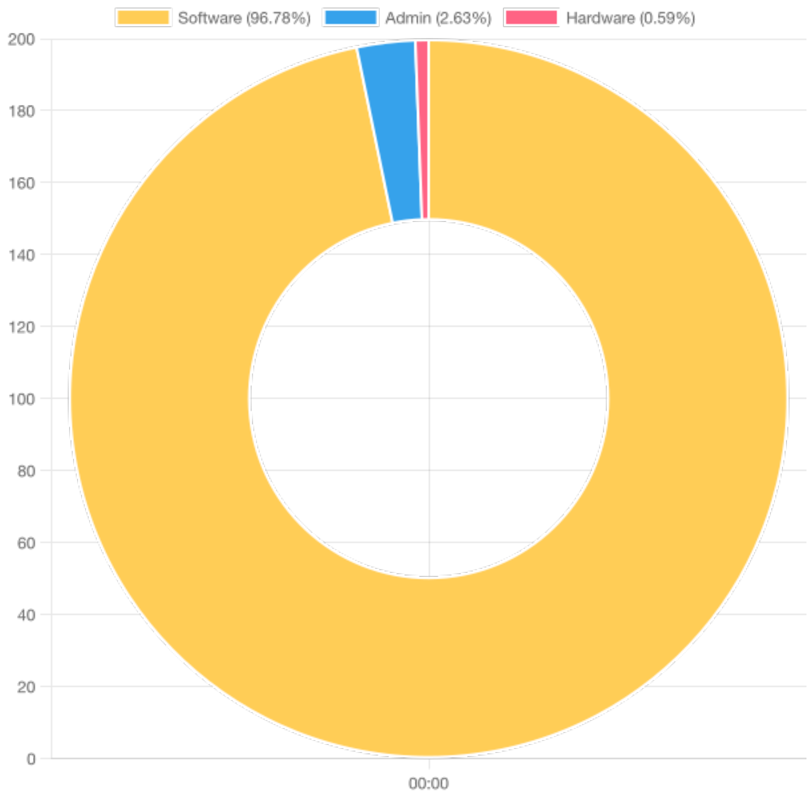

Figure 6.4.: Time spent on areas of work

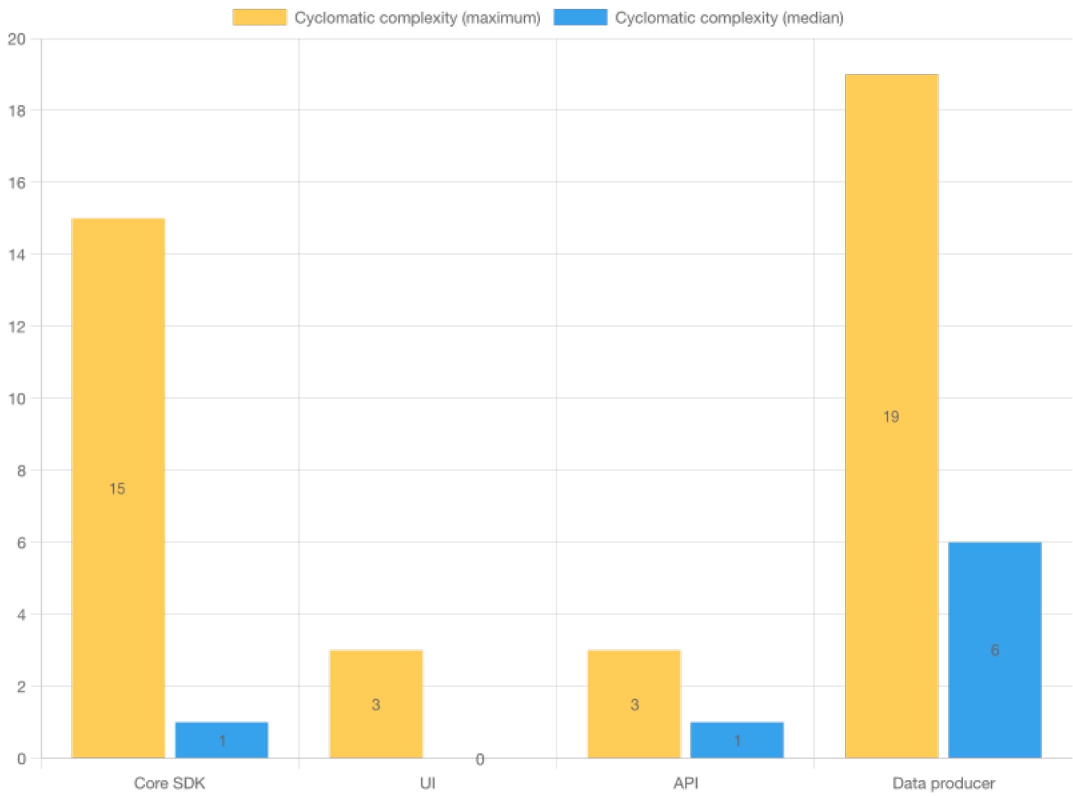

Figure 6.5.: Cyclomatic complexity per component

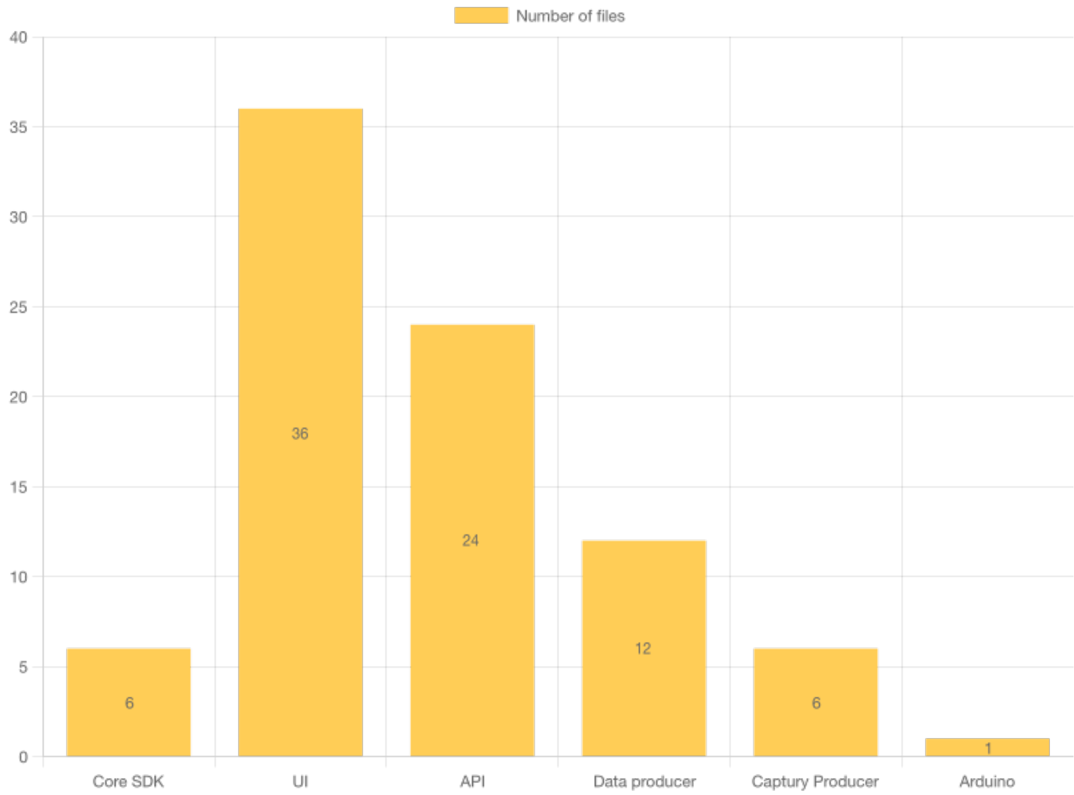

Figure 6.6.: Number of source files per application component

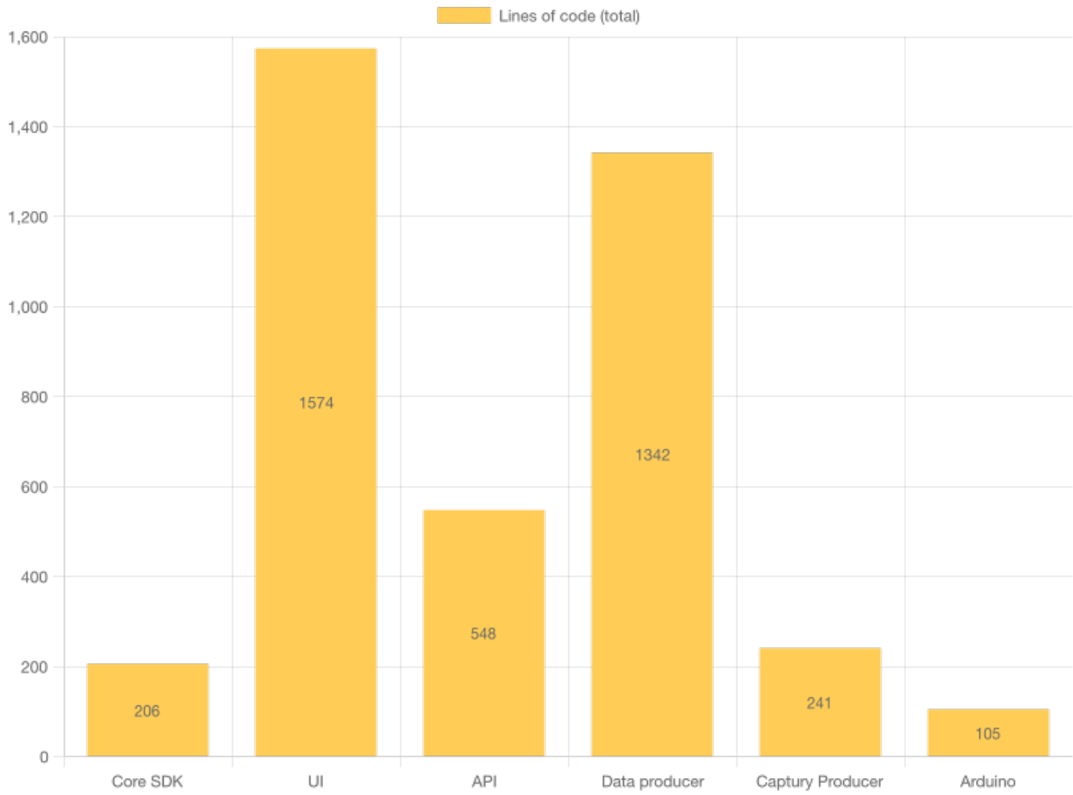

Figure 6.7.: Total lines of code per application component

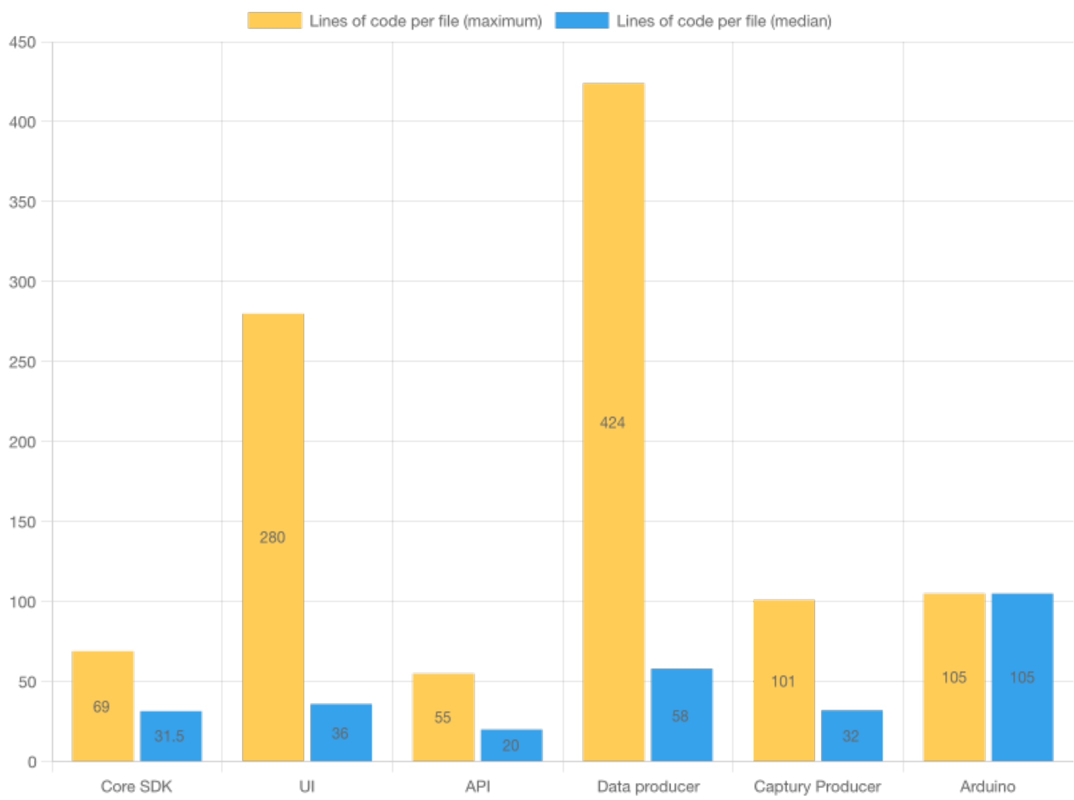

Figure 6.8.: Maximum and median number of lines of code per application component

6.2. Critical reflection

The development process was carried out in two main timeframes (X and Y) and followed the guiding principles decided in the concept. It was a mostly pleasant experience with the selected frameworks delivering on their promised functionality and ease of use. The initial setup was extremely quick due to the ease of setup of the WebRTC server and the quick generation of boilerplate code for the API and the UI. However, the web audio standard implementation leaves a lot to desire, especially the support for spatial audio in the browser. Currently, there is no built-in way to load custom HRTF data, which should drastically improve the accuracy of spatial positioning for sound. There are approaches using a custom build of the Chromium browser (Pike et al., 2015) or a custom audio node (Carpentier, 2015) which unfortunately does not work with the spatially oriented format for acoustics (SOFA) file format, so it is generally rather disappointing not seeing this thoroughly implemented in the general standard API.

The initial setup for the data producers was also relatively quick, due to the large number of examples for the DepthAI SDK and the clarity and hands-on approach to the main Python documentation. The only thing lacking here, was a straightforward way to manage dependencies, as there were different results for the same dependencies being installed via PIP or Conda.

7. Conclusion

The result of this feasibility study supports the recommendation for ‘single-use’ or rather ‘ephemeral’ or ‘transient’ development in cases where there is not a focus on commercial deployment of services or products, but rather specialised tools that are an intrinsic part of smaller self-contained or short-lived projects. In this case, a full-time developer could get the application up and running in about three weeks, not accounting for the value of reusable components for later iterations.

Regarding the bulk of the application as transient and extracting the core functionality into well-documented and tested modules also has the benefit, that if the application is passed on to other developers for another project or task, they can decide to either go with the existing infrastructure or to take only the core and implementing it in their own favoured environment. It would even enable them to more easily port the core to other languages. In the web development area, this is essential, because, while the standards and the application’s feature set might stay the same, but the framework and tooling landscape most certainly doesn’t and even just a few years can render the application obsolete, if it is not constantly maintained.

As this study was conducted by a single developer, the aspect of team dynamics is not accounted for. The unstable nature of the initial development process might pose a problem for development teams, because even if the work is strictly divided among the various team-members, it might still lead to conflicts when bringing the disparate application parts together. However, this could be alleviated by formulating strict message formats and protocols beforehand and then assigning separate components to developers that communicate via these protocols. Additionally, the reality for most digital projects in niche culture or arts disciplines usually do not get funding allocations that would allow for more than one or two developers to be included on the team.

In general, this method of development can explicitly be recommended for experimental, non-commercial endeavours where factors like scale or long-term maintainability are secondary and where development should happen from within the actual practice as opposed to trying to communicate wishes and needs to external service providers. It aims to 'decommodify' the notion of an application and to turn it into an ephemeral statement made within an artistic dialogue.

This results in 'compostable software' that grows and eventually decays but leaves its essential functionality in the form of small modules or even just documented methods and thought processes that can be used to support future development processes. Additionally, even if the application is meant to be discarded in its entirety after the project, the focus no longer needs to be on coding virtuosity and can help to lower the barrier between engineer and artist. This should enable a more intertwined and participatory dialogue between the project's participants that is not just expressed in development meetings, but actual proposed hacks and modifications or additions, given the interest on behalf of all parties involved.

Compostable software development is about valuing processes over products and embracing the ephemerality of digital technological techniques as intrinsic design factors. Thus, it should focus on constant reflection and re-evaluation of development results and being ready to potentially discarding everything but a seed for a fresh start, but keeping the knowledge and insights gained in the process alive through continuous inclusive, transdisciplinary dialogue and experimentation. It positively values, what otherwise would negatively be described as 'rot', the result of a messy growth process in an amalgamation of engineering and artistic experimentation.

Bibliography

A-Frame. (2015, September 15). *Web framework for building virtual reality experiences.* Retrieved January 25, 2024, from https://github.com/aframevr/aframe

Babylon.js. (2013, June 27). *Babylon.js is a powerful, beautiful, simple, and open game and rendering engine packed into a friendly javascript framework.* Retrieved January 25, 2024, from https://github.com/BabylonJS/Babylon.js

Barbier, J. (2014, June 9). *It's here: Docker 1.0.* Retrieved January 15, 2024, from https://web.archive.org/web/20220518024454/https://www.docker.com/blog/its-here-docker-1-0/

Barr, M. (2015, October 8). *Embedded systems glossary.* Retrieved January 20, 2024, from https://barrgroup.com/embedded-systems/glossary

Barragán, H. (2022, June 16). *The untold history of arduino.* Retrieved January 20, 2024, from https://arduinohistory.github.io/

Bazarevsky, V., Grishchenko, I., Raveendran, K., Zhu, T., Zhang, F., & Grundmann, M. (2020). Blazepose: On-device real-time body pose tracking.

Brandl, R. (2023, January 4). *The Most Popular Video Call Conferencing Platforms Worldwide.* Retrieved April 3, 2023, from https://www.emailtooltester.com/en/blog/video-conferencing-market-share/

Brock, H., Schmitz, G., Baumann, J., & Effenberg, A. O. (2012). If motion sounds: Movement sonification based on inertial sensor data [ENGINEERING OF SPORT CONFERENCE 2012]. *Procedia Engineering*, *34*, 556–561. https://doi.org/https://doi.org/10.1016/j.proeng.2012.04.095

Burns, B. (2018, July 20). *The history of Kubernetes & the community behind it*. Retrieved January 15, 2024, from https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/

Buschmann, F., Henney, K., & Schmidt, D. (2007). *Pattern-oriented software architecture: A pattern language for distributed computing*. John Wiley & Sons, Inc.

Carpentier, T. (2015, January). Binaural synthesis with the web audio api. In S. Goldszmidt, N. Schnell, V. Saiz & B. Matuszewski (Eds.), *Proceedings of the international web audio conference*. IRCAM.

Couriol, B. (2021, April 7). *10 years after inception, WebRTC becomes an official web standard*. Retrieved April 3, 2023, from https://www.infoq.com/news/2021/04/webrtc-official-web-standard/

Daigle, K., & GitHub. (2023, November 8). *Octoverse: The state of open source and rise of AI in 2023*. Retrieved January 14, 2024, from https://github.blog/2023-11-08-the-state-of-open-source-and-ai/

Davis, J., Nguyen, T., & Simmons, J. (2023, February 1). *Interop 2023: Pushing interoperability forward*. Retrieved April 3, 2023, from https://webkit.org/blog/13706/interop-2023/

Deveria, A. (2024, January 23). *Can i use: Websockets*. Retrieved January 25, 2024, from https://caniuse.com/websockets

doppp & forum users. (2019, October 2). *Hackernews: Tiny acquires Meteor*. Retrieved January 17, 2024, from https://news.ycombinator.com/item?id=21137653

Elementary. (2023, January 9). *Elementary is a javascript library for digital audio signal processing*. Retrieved January 25, 2024, from https://github.com/elemaudio/elementary

Erol444. (2022, March 9). *Depthai pointcloud generation*. Retrieved January 25, 2024, from https://github.com/luxonis/depthai-experiments/tree/master/gen2-pointcloud

Fette, I., & Melnikov, A. (2011, November 1). *The WebSocket protocol*. Retrieved January 14, 2024, from https://datatracker.ietf.org/doc/html/rfc6455

Flocking. (2011, March 27). *Flocking - creative audio synthesis for the web*. Retrieved January 25, 2024, from https://github.com/lichen-community-systems/Flocking

Galli, M., Soares, R., & Oeschger, I. (2003, May 16). *Inner-browsing: Extending web browsing the navigation paradigm*. Retrieved January 11, 2024, from https://web.archive.org/web/20030810102320/http://devedge.netscape.com/viewsource/2003/inner-browsing/

Garrett, J. J. (2005, February 18). *Ajax: A new approach to web applications*. Retrieved January 11, 2024, from https://web.archive.org/web/20150910072359/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

geaxgx. (2021, March 12). *Blazepose tracking with DepthAI*. Retrieved January 25, 2024, from https://github.com/geaxgx/depthai_blazepose

Greif, S., & Burel, E. (2022, February 16). *The state of JS: Most used backend frameworks in 2021*. Retrieved January 17, 2024, from https://2021.stateofjs.com/en-US/libraries/back-end-frameworks/

Greif, S., & Burel, E. (2023a, January 11). *The state of JS*. Retrieved January 11, 2024, from https://stateofjs.com

Greif, S., & Burel, E. (2023b, January 11). *The state of JS: Most used backend frameworks in 2022*. Retrieved January 11, 2024, from https://2022.stateofjs.com/en-US/other-tools/#backend_frameworks

Greif, S., & Burel, E. (2023c, January 11). *The state of JS: Most used frontend frameworks in 2022*. Retrieved January 11, 2024, from https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

Hermann, T., Hunt, A., & Neuhoff, J. G. (Eds.). (2011). *The Sonification Handbook*. Logos Publishing House. https://nbn-resolving.org/urn:nbn:de:0070-pub-24423492,%20https://pub.uni-bielefeld.de/record/2442349

Howler.js. (2013, January 28). *Javascript audio library for the modern web*. Retrieved January 25, 2024, from https://github.com/goldfire/howler.js

IBM. (n.d.-a). *What is container orchestration?* Retrieved January 15, 2024, from https://www.ibm.com/topics/container-orchestration

IBM. (n.d.-b). *What is containerization?* Retrieved January 15, 2024, from https://www.ibm.com/topics/containerization

Iyengar, M. (2021, March 10). *WebRTC architecture basics: P2P, SFU, MCU, and hybrid approaches*. Retrieved January 29, 2024, from https://medium.com/securemeeting/webrtc-architecture-basics-p2p-sfu-mcu-and-hybrid-approaches-6e7d77a46a66

Jackson, J. (2012, October 1). *Microsoft augments Javascript for large-scale development*. Retrieved January 15, 2024, from https://web.archive.org/web/20131217223751/http://www.cio.com/article/717679/Microsoft_Augments_Javascript_for_Large_scale_Development

Janus Gateway. (2014, February 11). *Janus webrtc server*. Retrieved January 25, 2024, from https://github.com/meetecho/janus-gateway

Jasmine. (2008, December 3). *Simple javascript testing framework for browsers and node.js*. Retrieved January 25, 2024, from https://github.com/jasmine/jasmine

Jest. (2013, December 10). *Delightful javascript testing*. Retrieved January 25, 2024, from https://github.com/jestjs/jest

Kendall, A., Grimes, M., & Cipolla, R. (2016). Posenet: A convolutional network for real-time 6-dof camera relocalization.

Kryski, E. (2016, April 6). *Why we built the best web framework you've probably never heard of (until now)*. Retrieved January 17, 2024, from https://blog.feathersjs.com/why-we-built-the-best-web-framework-you-ve-probably-never-heard-of-until-now-176afc5c6aac

Lardinois, F. (2019, October 2). *Tiny acquires Meteor*. Retrieved January 17, 2024, from https://techcrunch.com/2019/10/02/tiny-acquires-meteor/

Linux Foundation. (2015, July 21). *New Cloud Native Computing Foundation to drive alignment among container technologies*. Retrieved January 15, 2024, from https://www.cncf.io/announcements/2015/06/21/new-cloud-native-computing-foundation-to-drive-alignment-among-container-technologies/

Linux Foundation. (n.d.). *About the Open Container Initiative*. Retrieved January 15, 2024, from https://opencontainers.org/about/overview/

LiveKit. (2020, September 30). *End-to-end stack for webrtc. sfu media server and sdks.* Retrieved January 25, 2024, from https://github.com/livekit/livekit

LiveKit. (n.d.). *Livekit: About*. Retrieved January 19, 2024, from https://livekit.io/about

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Mediasoup. (2014, December 12). *Cutting edge webrtc video conferencings*. Retrieved January 25, 2024, from https://github.com/versatica/mediasoup

Minsky, M. (1980, June 1). *Telepresence*. Retrieved January 15, 2024, from https://web.media.mit.edu/~minsky/papers/Telepresence.html

Mocha. (2011, March 7). *Simple, flexible, fun javascript test framework for node.js & the browser*. Retrieved January 25, 2024, from https://github.com/mochajs/mocha

MongoDB. (2009, January 15). *The mongodb database*. Retrieved January 25, 2024, from https://github.com/mongodb/mongo

MySQL Server. (2014, September 26). *MySQL server, the world's most popular open source database, and MySQL cluster, a real-time, open source transactional database*. Retrieved January 25, 2024, from https://github.com/mysql/mysql-server

Netscape. (1995, December 4). *Netscape and Sun announce JavaScript*. Retrieved January 15, 2024, from https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html

Pike, C., Taylour, P., & Melchior, F. (2015, January). Delivering object-based 3d audio using the web audio api and the audio definition model. In S. Goldszmidt, N. Schnell, V. Saiz & B. Matuszewski (Eds.), *Proceedings of the international web audio conference*. IRCAM.

Ponton, J. L. (2023, June 2). *Pymotion: A python library for motion data*. Retrieved January 25, 2024, from https://github.com/UPC-ViRVIG/pymotion

PostgreSQL. (2010, September 21). *Mirror of the official PostgreSQL git repository*. Retrieved January 25, 2024, from https://github.com/postgres/postgres

Python Software Foundation. (n.d.). *History of the software*. Retrieved January 15, 2024, from https://docs.python.org/3/license.html

rafern. (2022, December 8). *Mediastream support #1634*. Retrieved January 29, 2024, from https://github.com/goldfire/howler.js/pull/1634

Rehn, A. (2016, December 3). *C++ WebSocket server demo*. Retrieved January 25, 2024, from https://github.com/adamrehn/websocket-server-demo

Rosenberg, L. H., Hammer, T., & Shaw, J. G. (1998). Software metrics and reliability. https://api.semanticscholar.org/CorpusID:59798472

Russell, A., & Berriman, F. (2015, October 8). *Progressive web apps: Escaping tabs without losing our soul*. Retrieved January 11, 2024, from https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955

Skarbez, R., Brooks Jr., F. P., & Whitton, M. C. (2017). A survey of presence and related concepts. *ACM Computing Surveys*, *50*(96), 1–39. https://doi.org/10.1145/3134301

Socket.IO. (2010, March 11). *Realtime application framework (node.js server)*. Retrieved January 25, 2024, from https://github.com/socketio/socket.io

Stack Overflow. (2023a, June 13). *2023 Stack Overflow developer survey: Databases*. Retrieved January 25, 2024, from https://survey.stackoverflow.co/2023/#section-most-popular-technologies-databases

Stack Overflow. (2023b, June 13). *2023 Stack Overflow developer survey: Web frameworks and technologies*. Retrieved January 13, 2024, from https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies

Stack Overflow. (2023c, June 13). *Stack Overflow developer survey*. Retrieved January 13, 2024, from https://survey.stackoverflow.co

The Captury GmbH. (2018, October 18). *Streaming library from CapturyLive*. Retrieved January 25, 2024, from https://github.com/thecaptury/RemoteCaptury

three.js. (2010, March 23). *Javascript 3d library*. Retrieved January 25, 2024, from https://github.com/mrdoob/three.js

Tone.js. (2014, March 11). *A web audio framework for making interactive music in the browser*. Retrieved January 25, 2024, from https://github.com/Tonejs/Tone.js

van Rossum, G. (2008, August 11). *Why is Python a dynamic language and also a strongly typed language?* Retrieved January 15, 2024, from https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language

Volpe, G. (2003, April 22). *Computational models of expressive gesture in multimedia systems*. Retrieved April 4, 2023, from https://theses.eurasip.org/theses/157/computational-models-of-expressive-gesture-in/download/

W3Schools. (n.d.). *Javascript history*. Retrieved January 15, 2024, from https://www.w3schools.com/js/js_history.asp

Wallace, D., Watson, A., & Mccabe, T. (1996, 1996-08-01). Structured testing: A testing methodology using the cyclomatic complexity metric.

Web Audio API. (2021, June 17). Retrieved April 4, 2023, from https://www.w3.org/TR/webaudio/

WebRTC: Real-time communication in browsers. (2023, March 6). Retrieved April 3, 2023, from https://www.w3.org/TR/webrtc/

WebXR Device API. (2023, March 3). Retrieved April 4, 2023, from https://www.w3.org/TR/webxr/

Worrall, D. (2018). Sonification: A prehistory, 177–182. https://doi.org/10.21785/icad2018.019

ws. (2011, November 9). *Simple to use, blazing fast and thoroughly tested websocket client and server for node.js*. Retrieved January 25, 2024, from https://github.com/websockets/ws

Ye, M., Wang, X., Yang, R., Ren, L., & Pollefeys, M. (2011). Accurate 3D pose estimation from a single depth image. *2011 International Conference on Computer Vision*, 731–738. https://doi.org/10.1109/ICCV.2011.6126310

Yoshino, T. (2015, December 7). *Compression extensions for WebSocket*. Retrieved January 25, 2024, from https://datatracker.ietf.org/doc/html/rfc7692

You, E. (2021, September 29). *Introduction: The progressive framework*. Retrieved January 14, 2024, from https://vuejs.org/guide/introduction.html#the-progressive-framework

Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.

µWebSockets. (2016, March 21). *Simple, secure & standards compliant web server for the most demanding of applications*. Retrieved January 25, 2024, from https://github.com/uNetworking/uWebSockets

Appendix Listing

A. Appendix